

# IITM GROUP OF INSTITUTIONS, MURTHAL SONIPAT

DEPARTMENT: COMPUTER SCIENCE AND ENGG.

SUBJECT: ADVANCE JAVA CSE-306B

## UNIT : 1 JAVA LANGUAGE BASICS

### JAVA:

**Java** is a general-purpose, concurrent, object-oriented, class-based, and the runtime environment(JRE) which consists of **JVM** which is the cornerstone of the Java platform. This blog on **What is Java** will clear all your doubts about why to learn java, features and how it works.

In this What is Java blog, I would be covering following topics:

- [What is Java used for?](#)
- [History of Java](#)
- [What is Java?](#)
- [Features of Java](#)
- [Components in Java](#)

### What is Java used for?

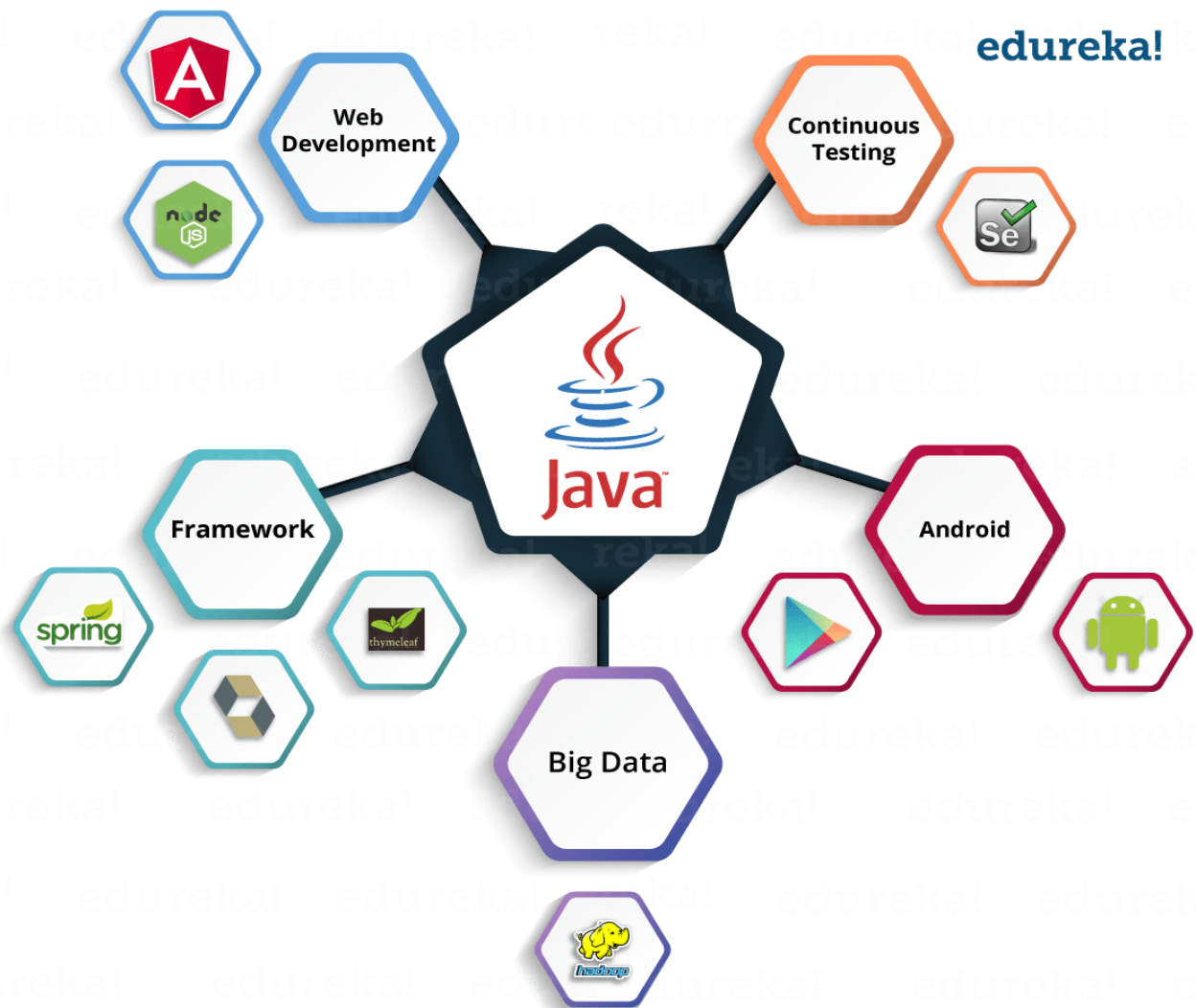
Before I answer the question, what is Java used for, let me brief you about why you should choose Java. Java is highly popular and has dominated this field from early 2000's till the present 2018.

Java has been used in different domains. Some of them are listed below:

- **Banking:** To deal with transaction management.
- **Retail:** Billing applications that you see in a store/restaurant are completely written in Java.
- **Information Technology:** Java is designed to solve implementation dependencies.
- **Android:** Applications are either written in Java or use Java API.
- **Financial services:** It is used in server-side applications.
- **Stock market:** To write algorithms as to which company they should invest in.
- **Big Data:** Hadoop MapReduce framework is written using Java.
- **Scientific and Research Community:** To deal with huge amount of data.

***Wait! Java can do more.***

Let's see how some of the technologies make use of Java as an essential core of their functionalities.



Let's see how some of the technologies make use of Java as an essential core of their functionalities.

You can see in the above image, Java is an ***ocean of opportunities***.

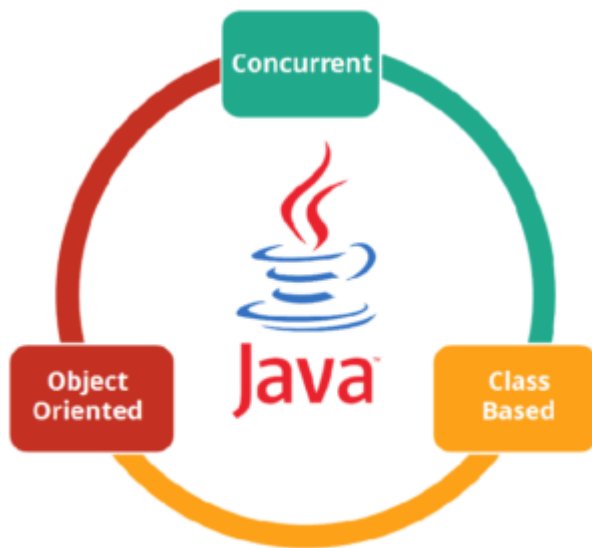
Let us see a brief history of Java.

## History of Java

Java is a programming language developed by **James Gosling** with other team members named **Mike Sheridan** and **Patrick Naughton** also called as **Green Team** in **1995** for **Sun Microsystems** for digital devices such as set-top boxes, televisions etc. Now, let us see in detail what is Java.

## What is Java?

It is an object-oriented language similar to C++, but with advanced and simplified features. Java is **free to access** and can **run on all platforms**.



Java is: –

- **Concurrent** where you can execute many statements instead of sequentially executing it.
- **Class-based** and an **object-oriented** programming language.
- **Independent** programming language that follows the logic of “**Write once, Run anywhere**” i.e. the compiled code can run on all platforms which supports java.

In simple words, it is a computing platform where you can develop applications.

You may go through this Java recording where our [Java Certification Training](#) expert has explained the topics in a detailed manner with examples which will help you to understand the concepts better.

## Features of Java



**Simple:** Java has made life easier by removing all the complexities such as pointers, operator overloading as you see in C++ or any other programming language.

**Portable:** Java is platform independent which means that any application written on one platform can be easily ported to another platform.



**Object-oriented:** Everything is considered to be an “**object**” which possess some state, behavior and all the operations are performed using these objects.



**Secured:** All the code is converted in **bytecode** after compilation, which is not readable by a human. and java does not use an explicit pointer and run the programs inside the sandbox to prevent any activities from untrusted sources. It enables to develop virus-free, tamper-free systems/applications.



**Dynamic:** It has the ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.



**Distributed:** Java provides a feature which helps to create distributed applications. Using Remote Method Invocation (RMI), a program can invoke a method of another program across a network and get the output. You can access files by calling the methods from any machine on the internet.



**Robust:** Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.

**High Performance:** Java achieves high performance through the use of bytecode which can be easily translated into native machine code. With the use of JIT (Just-In-Time) compilers, Java enables high performance.



**Interpreted:** Java is compiled to bytecodes, which are interpreted by a Java run-time environment.



**Multithreaded:** Java supports multiple threads of execution (a.k.a., lightweight processes), including a set of synchronization primitives. This makes programming with threads much easier.

## Components in Java

### JVM (Java Virtual Machine)

It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed. It follows three notations:

- **Specification:** It is a document that describes the implementation of the Java virtual machine. It is provided by Sun and other companies.
- **Implementation:** It is a program that meets the requirements of JVM specification.
- **Runtime Instance:** An instance of JVM is created whenever you write a java command on the command prompt and run the class.

### JRE (Java Runtime Environment)

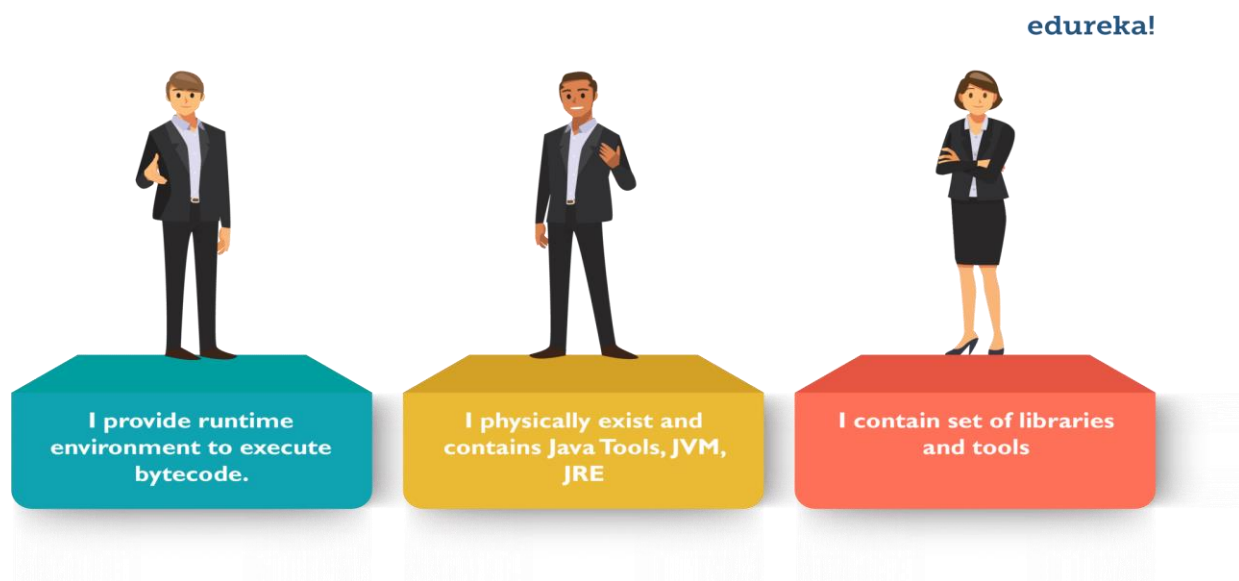
JRE refers to a runtime environment in which Java bytecode can be executed. It implements the JVM (Java Virtual Machine) and provides all the class libraries and other support files that JVM uses at runtime. So JRE is a software package that contains what is required to run a Java program. Basically, it's an implementation of the JVM which physically exists.

### JDK(Java Development Kit)

It is the tool necessary to:-

- Compile
- Document
- Package Java programs.

The JDK completely includes JRE which contains tools for Java programmers. The Java Development Kit is provided free of charge. Along with JRE, it includes an interpreter/loader, a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development. In short, it contains JRE + development tools.



## What is Class?

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object.

### Syntax

```
class <class_name>{
    field;
    method;
}
```

## What is an Object?

An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

From a programming point of view, an object can be a data structure, a variable or a function. It has a memory location allocated. The object is designed as class hierarchies.

## Syntax

```
ClassName ReferenceVariable = new ClassName();
```

## What is the Difference Between Object & Class?

A **class** is a **blueprint or prototype** that defines the variables and the methods (functions) common to all objects of a certain kind.

An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

## What is Class?

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object.

## Syntax

```
class <class_name>{  
    field;  
    method;  
}
```

## What is an Object?

An object is nothing but a self-contained component which consists of methods and properties to make a particular type of data useful. Object determines the behavior of the class. When you send a message to an object, you are asking the object to invoke or execute one of its methods.

From a programming point of view, an object can be a data structure, a variable or a function. It has a memory location allocated. The object is designed as class hierarchies.

## Syntax

```
ClassName ReferenceVariable = new ClassName();
```

## What is the Difference Between Object & Class?

A **class** is a **blueprint or prototype** that defines the variables and the methods (functions) common to all objects of a certain kind.

An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life.

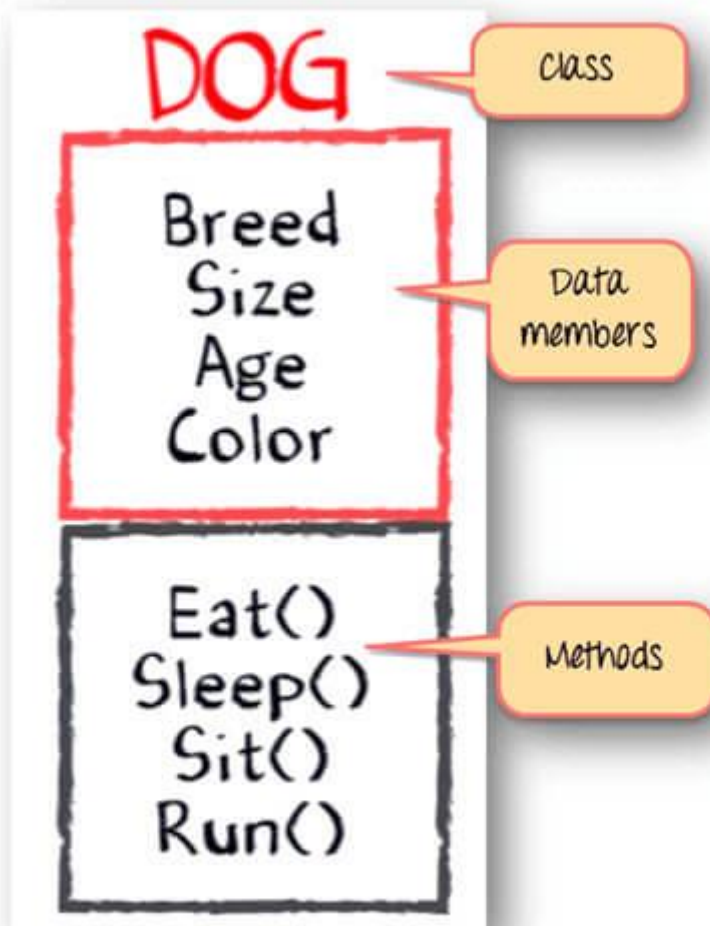
## Understand the concept of Java Classes and Objects with an example.

Let's take an example of developing a pet management system, specially meant for dogs. You will need various information about the dogs like different breeds of the dogs, the age, size, etc.

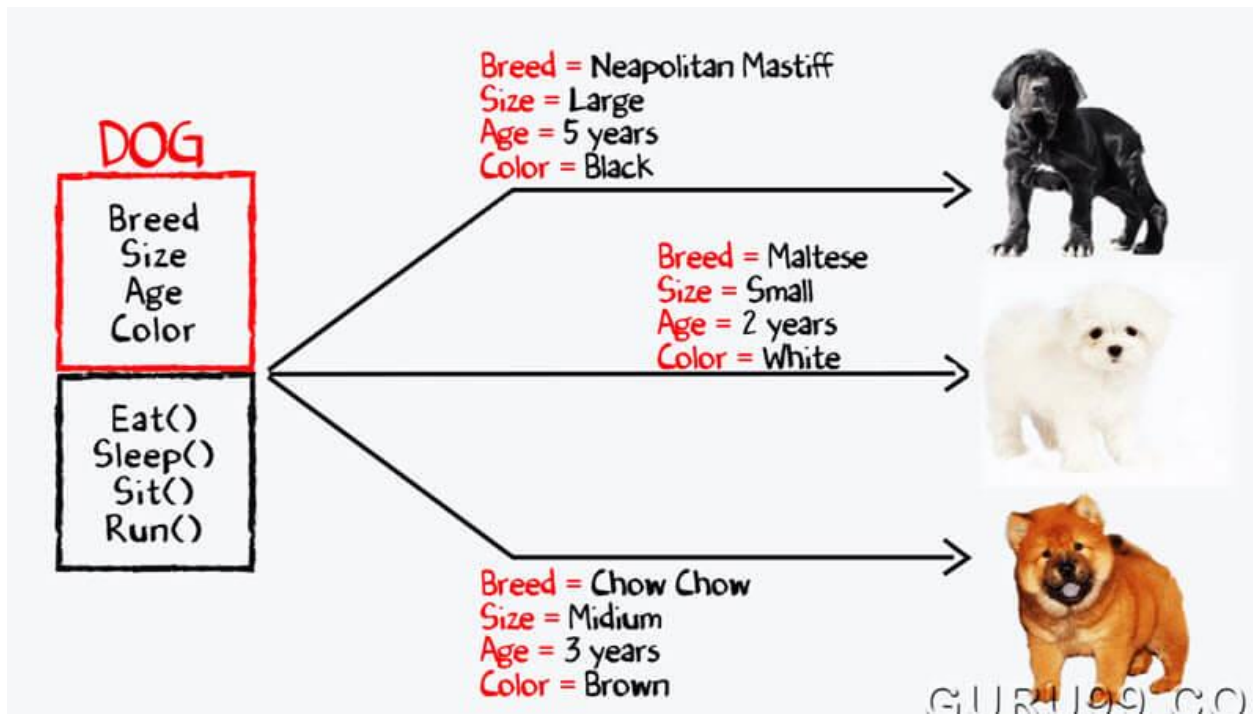
You need to model real-life beings, i.e., dogs into software entities.

- **Class** - Dogs
- **Data members** or **objects**- size, age, color, breed, etc.
- **Methods**- eat, sleep, sit and run.
- **Class** - Dogs
- **Data members** or **objects**- size, age, color, breed, etc.
- **Methods**- eat, sleep, sit and run.





Now, for different values of data members (breed size, age, and color) in Java class, you will get different dog objects.



You can design any program using this OOPs approach.

While creating a class, one must follow the following principles.

- **Single Responsibility Principle (SRP)**- A class should have only one reason to change
- **Open Closed Responsibility (OCP)**- It should be able to extend any classes without modifying it
- **Liskov Substitution Responsibility (LSR)**- Derived classes must be substitutable for their base classes
- **Dependency Inversion Principle (DIP)**- Depend on abstraction and not on concretions
- **Interface Segregation Principle (ISP)**- Prepare fine grained interfaces that are client specific.

## Example Code: Class and Object

```
// Class Declaration
public class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;

    // method 1
```

```

    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+c
olor);
    }

    public static void main(String[] args) {
        Dog maltese = new Dog();
        maltese.breed="Maltese";
        maltese.size="Small";
        maltese.age=2;
        maltese.color="white";
        System.out.println(maltese.getInfo());
    }
}

```

### Output:

```
Breed is: Maltese Size is:Small Age is:2 color is: white
```

## Object and Class Example: main outside class

In previous program, we are creating main() method inside the class. Now, we create classes and define main() method in another class. This is a better way than previous one.

```

// Class Declaration
class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;

    // method 1
    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+c
olor);
    }
}

public class Execute{
    public static void main(String[] args) {
        Dog maltese = new Dog();
        maltese.breed="Maltese";
        maltese.size="Small";
        maltese.age=2;
        maltese.color="white";
    }
}

```

```
        System.out.println(maltese.getInfo());  
    }  
}
```

### Output:

```
Breed is: Maltese Size is:Small Age is:2 color is: white
```

### Summary:

- Java Class is an entity that determines how an object will behave and what the object will contain
- A Java object is a self-contained component which consists of methods and properties to make certain type of data useful
- A class system allows the program to define a new class (derived class) in terms of an existing class (superclass) by using a technique like inheritance, overriding and augmenting.

## What is Constructor in Java?

A **constructor** is a special method that is used to initialize a newly created object and is called just after the memory is allocated for the object. It can be used to initialize the objects to desired values or default values at the time of object creation. It is not mandatory for the coder to write a constructor for a class.

If no user-defined constructor is provided for a class, compiler initializes member variables to its default values.

- numeric data types are set to 0
- char data types are set to null character('\0')
- reference variables are set to null

## Rules for creating a Java Constructor

1. It has the **same name** as the class
2. It should not return a value not even **void**

**Example 1:** Create your First Constructor Java

**Step 1)** Type following code in your editor.

```
class Demo{
```

```

int value1;
int value2;
Demo(){
    value1 = 10;
    value2 = 20;
    System.out.println("Inside Constructor");
}

public void display(){
    System.out.println("Value1 === "+value1);
    System.out.println("Value2 === "+value2);
}

public static void main(String args[]){
    Demo d1 = new Demo();
    d1.display();
}
}

```

**Step 2)** Save , Run & Compile the code. Observe the output.

### Output:

```

Inside Constructor
Value1 === 10
Value2 === 20

```

## Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter list. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Examples of valid constructors for class Account are

```

Account(int a);
Account (int a,int b);
Account (String a,int b);

```

**Example 2:** To understand Constructor Overloading

**Step 1)** Type the code in the editor.

```

class Demo{
    int value1;

```

```

    int value2;
    /*Demo(){
        value1 = 10;
        value2 = 20;
        System.out.println("Inside 1st Constructor");
    }*/
    Demo(int a){
        value1 = a;
        System.out.println("Inside 2nd Constructor");
    }
    Demo(int a,int b){
        value1 = a;
        value2 = b;
        System.out.println("Inside 3rd Constructor");
    }
    public void display(){
        System.out.println("Value1 === "+value1);
        System.out.println("Value2 === "+value2);
    }
    public static void main(String args[]){
        Demo d1 = new Demo();
        Demo d2 = new Demo(30);
        Demo d3 = new Demo(30,40);
        d1.display();
        d2.display();
        d3.display();
    }
}

```

**Step 2)** Save, Compile & Run the Code.

**Step 3)** Error = ?. Try and debug the error before proceeding to next step.

**Step 4)** Every class has a default Constructor. Default Constructor for **class Demo** is **Demo()**. In case you do not provide this constructor the compiler creates it for you and initializes the variables to default values. You may choose to override this default constructor and initialize variables to your desired values as shown in Example 1.

**But if you specify a parametrized constructor like Demo(int a), and want to use the default constructor Demo(), it is mandatory for you to specify it.**

**In other words, in case your Constructor is overridden, and you want to use the default constructor, its need to be specified.**

**Step 5)** Uncomment line # 4-8. Save, Compile & Run the code.

## Constructor Chaining

Consider a scenario where a base class is extended by a child. Whenever an object of the child class is created, the constructor of the parent class is invoked first. This is called **Constructor chaining**.

**Example 3:** To understand constructor chaining

**Step 1)** Copy the following code into the editor.

```
class Demo{
    int value1;
    int value2;
    Demo(){
        value1 = 1;
        value2 = 2;
        System.out.println("Inside 1st Parent Constructor");
    }
    Demo(int a){
        value1 = a;
        System.out.println("Inside 2nd Parent Constructor");
    }
    public void display(){
        System.out.println("Value1 === "+value1);
        System.out.println("Value2 === "+value2);
    }
    public static void main(String args[]){
        DemoChild d1 = new DemoChild();
        d1.display();
    }
}

class DemoChild extends Demo{
    int value3;
    int value4;
    DemoChild(){
        //super(5);
        value3 = 3;
        value4 = 4;
        System.out.println("Inside the Constructor of Child");
    }
    public void display(){
        System.out.println("Value1 === "+value1);
        System.out.println("Value2 === "+value2);
        System.out.println("Value1 === "+value3);
    }
}
```

```

        System.out.println("Value2 === "+value4);
    }
}

```

**Step 2)** Run the Code. Owing to constructor chaining, when the object of child class DemoChild is created, constructor Demo() of the parent class is invoked first and later constructor DemoChild() of the child is created.  
Expected Output =

```

Inside 1st Parent Constructor
Inside the Constructor of Child
Value1 === 1
Value2 === 2
Value1 === 3
Value2 === 4

```

**Step 3)** You may observe the constructor of the parent class Demo is overridden. What if you want to call the overridden constructor Demo(int a) instead of the default constructor Demo() when your child object is created?

In such cases, you can use the keyword "**super**" to call overridden constructors of the parent class.

### Syntax:-

```

super();
--or--
super(parameter list);

```

**Example:** If your constructor is like **Demo(String Name,int a)** you will specify **super("Java",5)** If used, the keyword **super** needs to be the **first line of code** in the constructor of the child class.

**Step 4)** Uncomment Line # 26 and run the code. Observe the Output.

### Output:

```

Inside 2nd Parent Constructor
Inside the Constructor of Child
Value1 === 5
Value2 === 0
Value1 === 3
Value2 === 4

```



## UNIT:2 ( INHERITANCE AND POLYMORPHISM )

### What is Inheritance?

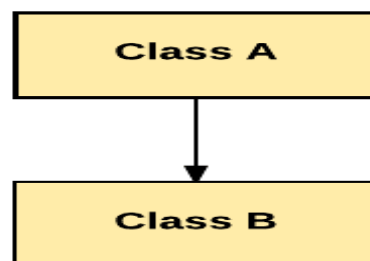
**Inheritance** is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.

### Types of Inheritance

There are Various types of inheritance in Java:

#### Single Inheritance:

In Single Inheritance one class extends another class (one class only).

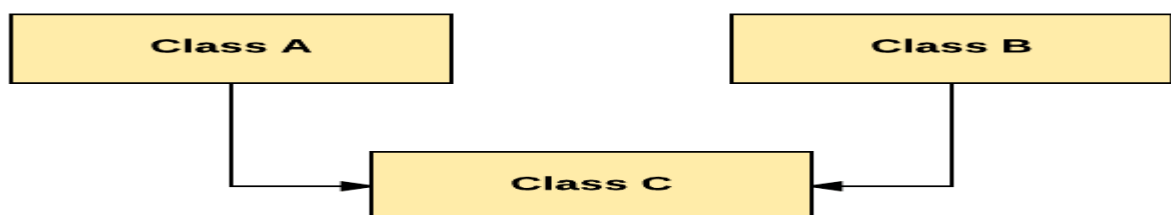


Single Inheritance

In above diagram, Class B extends only Class A. Class A is a super class and Class B is a Sub-class.

#### Multiple Inheritance:

In Multiple Inheritance, one class extending more than one class. Java does not support multiple inheritance.

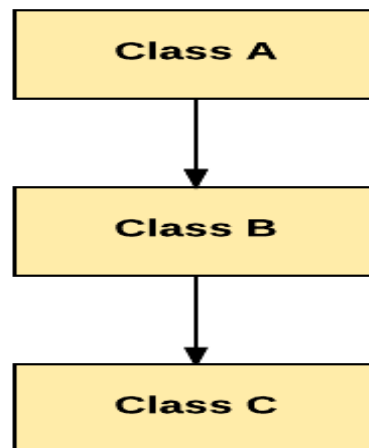


Multiple Inheritance

As per above diagram, Class C extends Class A and Class B both.

### **Multilevel Inheritance:**

In Multilevel Inheritance, one class can inherit from a derived class. Hence, the derived class becomes the base class for the new class.

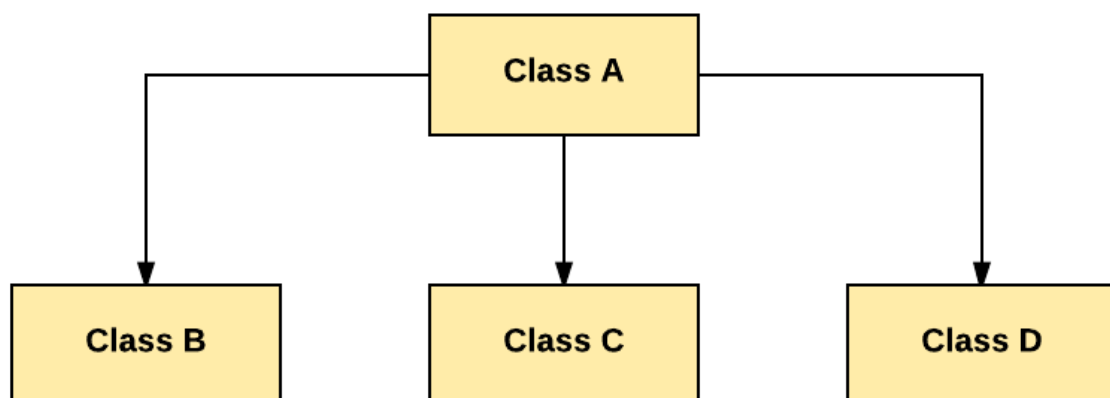


Multilevel Inheritance

As per shown in diagram Class C is subclass of B and B is a of subclass Class A.

### **Hierarchical Inheritance:**

In Hierarchical Inheritance, one class is inherited by many sub classes.

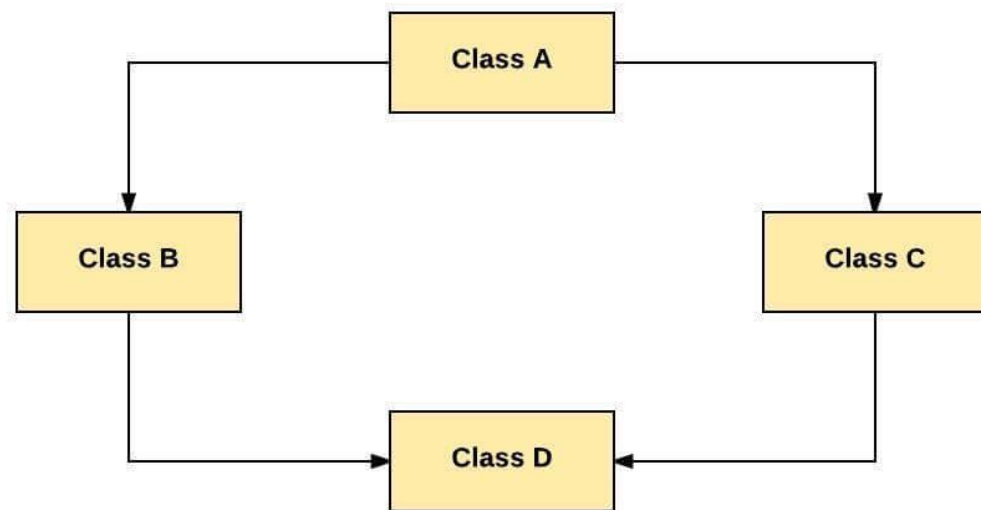


Hierarchical Inheritance

As per above example, Class B, C, and D inherit the same class A.

## Hybrid Inheritance:

Hybrid inheritance is a combination of Single and Multiple inheritance.



Inheritance

Hybrid

As per above example, all the public and protected members of Class A are inherited into Class D, first via Class B and secondly via Class C.

**Note:** Java doesn't support hybrid/Multiple inheritance

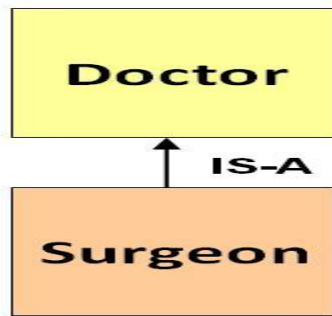
## Inheritance in Java

**Java Inheritance** is a mechanism in which one class acquires the property of another class. In Java, when an "Is-A" relationship exists between two classes, we use Inheritance. The parent class called a super class and the inherited class called as a sub class. The keyword `extends` is used by the sub class to inherit the features of super class. Inheritance is important since it leads to the reusability of code.

### Java Inheritance Syntax:

```
class subClass extends superClass
{
    //methods and fields
}
```

## Java Inheritance Example



```
class Doctor {
    void Doctor_Details() {
        System.out.println("Doctor Details...");
    }
}

class Surgeon extends Doctor {
    void Surgeon_Details() {
        System.out.println("Surgen Detail...");
    }
}

public class Hospital {
    public static void main(String args[]) {
        Surgeon s = new Surgeon();
        s.Doctor_Details();
        s.Surgeon_Details();
    }
}
```

## Super Keyword

The super keyword is similar to "this" keyword.

The keyword super can be used to access any data member or methods of the parent class.

Super keyword can be used at variable, method and constructor level.

### Syntax:

```
super.<method-name>();
```

## Learn Inheritance in OOP's with Example

Consider the same banking application from the [PREVIOUS EXAMPLE](#).

We are supposed to open two different account types, one for saving and another for checking (also known as current).

- 1) saving
- 2) Current / Checking

Let's compare and study how we can approach coding from a **structured and object-oriented programming perspective**. **Structural approach:** In structured programming, we will create two functions –

1. One to withdraw
2. And the other for deposit action.

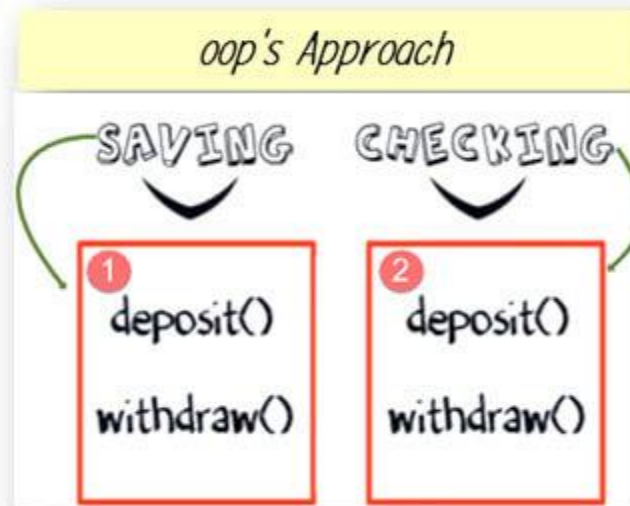
Since the working of these functions remains same across the accounts.

*Structural Approach*

```
public withdraw(){  
    //code to withdraw 1  
}  
  
public deposit(){  
    //code to deposit 2  
}
```

**OOP's approach:** While using the OOPs programming approach. We would create two classes.

- Each having implementation of the deposit and withdraw functions.
- This will redundant extra work.

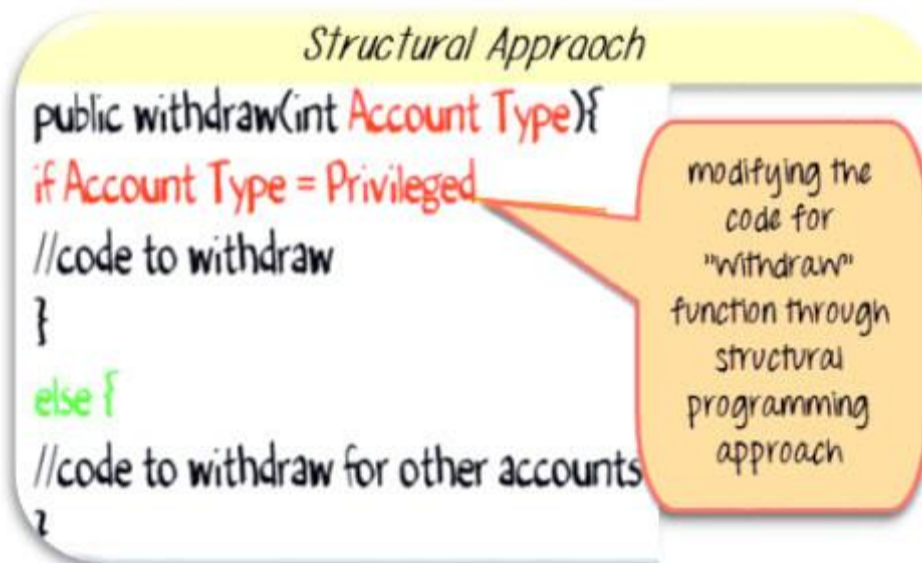


## Change Request in Software

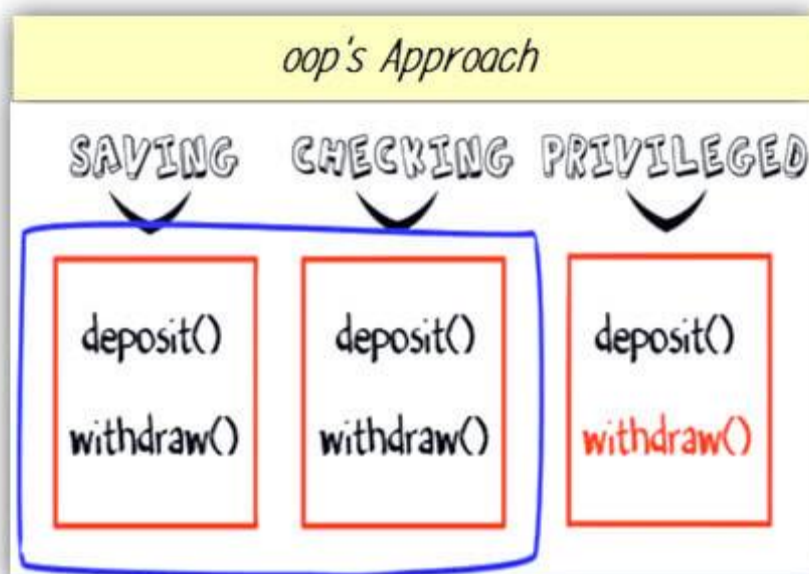
Now there is a change in the requirement specification for something that is so common in the software industry. You are supposed to add functionality privileged Banking Account with Overdraft Facility. For a background, overdraft is a facility where you can withdraw an amount more than available the balance in your account.



**Structural approach:** Using functional approach, I have to modify my withdraw function, which is already tested and baselined. And add a method like below will take care of new requirements.



**OOP's approach:** Using OOP's approach, you just need to write a new class with unique implementation of withdraw function. We never touched the tested piece of code.

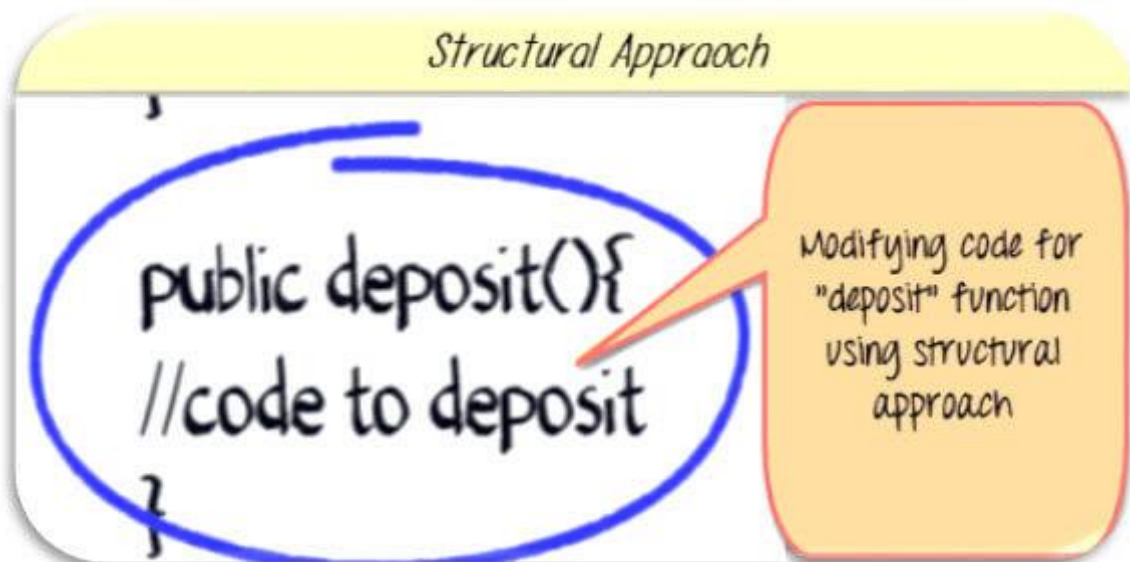


## Another Change Request

What if the requirement changes further? Like to add credit card account with its own unique requirement of deposits.

- 1) saving
- 2) Current / Checking
- 3) Privileged
- 4) Credit Card

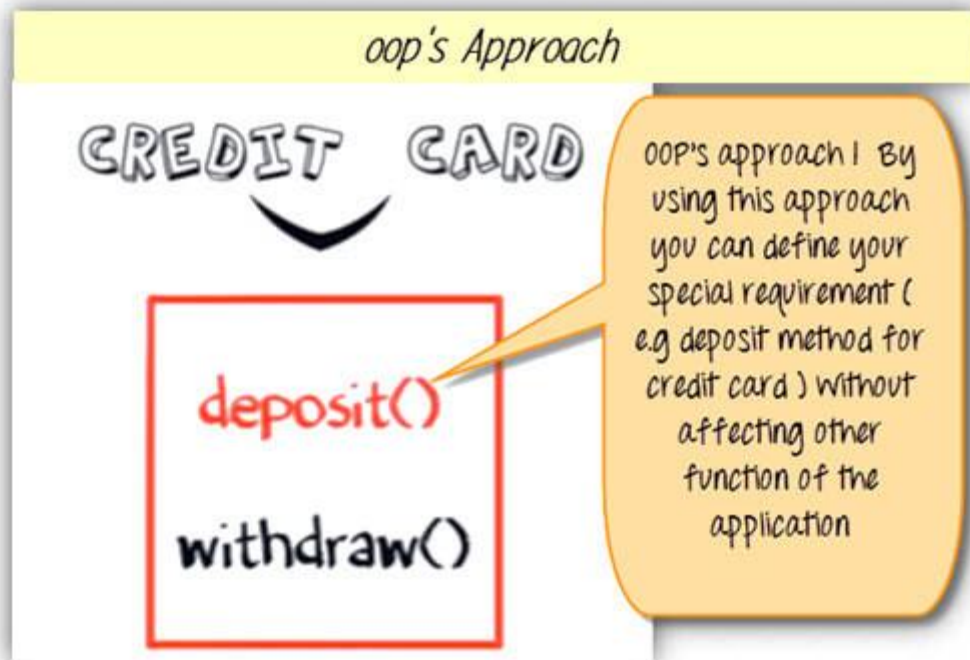
**Structural approach:** Using structural approach you have to change tested piece of deposit code again.



**OOP's approach:** But using object-oriented approach, you will just create a new class with its unique implementation of deposit method (highlighted red in the image below).

So even though the structural programming seems like an easy approach initially, OOP's wins in a long term.



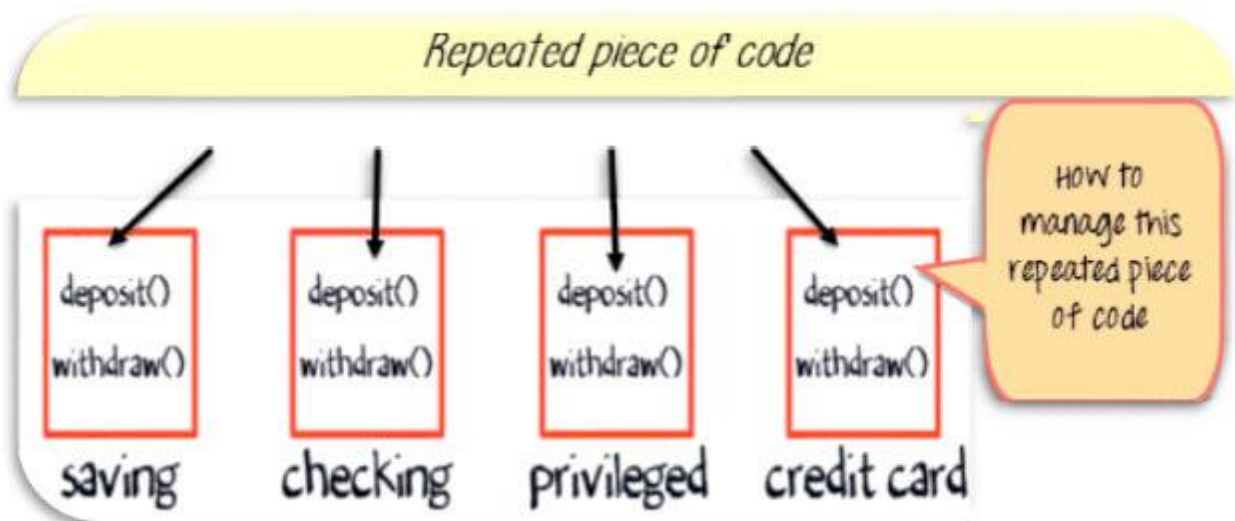


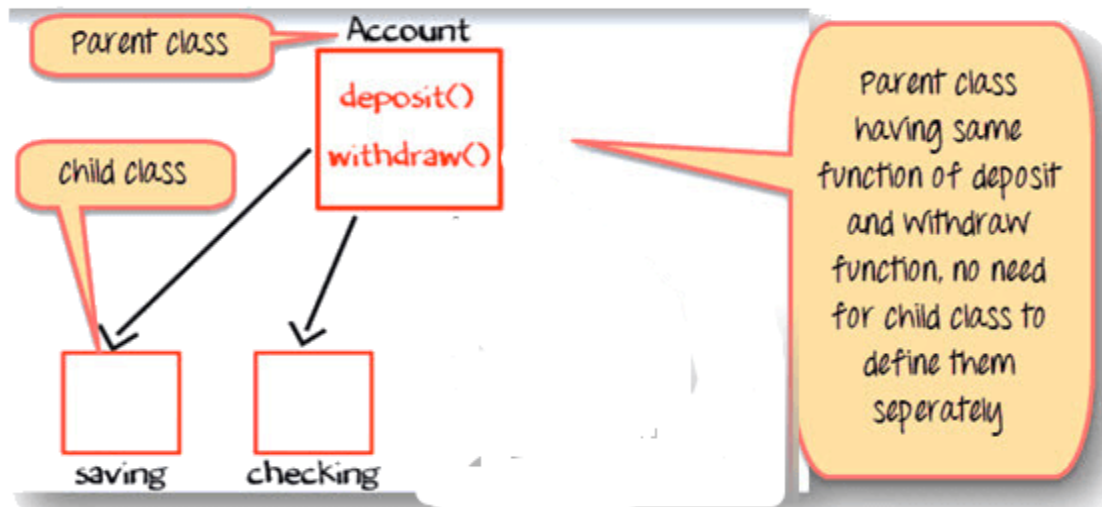
## Advantage of Inheritance in OOPs

But one may argue that across all classes, you have a repeated pieces of code.

To overcome this, you create a parent class, say "account" and implement the same function of deposit and withdraw. And make child classes inherited "account" class. So that they will have access to withdraw and deposit functions in account class.

The functions are not required to be implemented individually. This is **Inheritance in java**.





## Overloading vs Overriding in Java

1. Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition happens at compile-time however binding of overridden method call to its definition happens at runtime.
2. Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.
3. The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
4. Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.
5. Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
6. private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
7. Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the overriding method can have more specific return type (refer this).
8. Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

## Overloading example

```
//A class for adding upto 5 numbers
class Sum
{
    int add(int n1, int n2)
    {
        return n1+n2;
    }
    int add(int n1, int n2, int n3)
    {
        return n1+n2+n3;
    }
    int add(int n1, int n2, int n3, int n4)
    {
        return n1+n2+n3+n4;
    }
    int add(int n1, int n2, int n3, int n4, int n5)
    {
        return n1+n2+n3+n4+n5;
    }
    public static void main(String args[])
    {
        Sum obj = new Sum();
        System.out.println("Sum of two numbers: "+obj.add(20, 21));
        System.out.println("Sum of three numbers: "+obj.add(20, 21, 22));
        System.out.println("Sum of four numbers: "+obj.add(20, 21, 22, 23));
        System.out.println("Sum of five numbers: "+obj.add(20, 21, 22, 23, 24));
    }
}
```

Output:

```
Sum of two numbers: 41
Sum of three numbers: 63
Sum of four numbers: 86
Sum of five numbers: 110
```

Here we have 4 versions of same method `add`. We are overloading the method `add()` here.

## Overriding example

```
package beginnersbook.com;
class CarClass
{
    public int speedLimit()
    {
        return 100;
    }
}
class Ford extends CarClass
{
    public int speedLimit()
    {
        return 150;
    }
    public static void main(String args[])
    {
    }
```

```

{
    CarClass obj = new Ford();
    int num= obj.speedLimit();
    System.out.println("Speed Limit is: "+num);
}
}

```

OUTPUT:

```

peed Limit is: 150

```

Here speedLimit() method of class Ford is overriding the speedLimit() method of class CarClass.

## Polymorphism in Java with example

Polymorphism is one of the [OOps](#) feature that allows us to perform a single action in different ways. For example, lets say we have a class Animal that has a method sound(). Since this is a generic class so we can't give it a implementation like: Roar, Meow, Oink etc. We had to give a generic message.

```

public class Animal{
    ...
    public void sound(){
        System.out.println("Animal is making a sound");
    }
}

```

Now lets say we two subclasses of Animal class: Horse and Cat that extends (see [Inheritance](#)) Animal class. We can provide the implementation to the same method like this:

```

public class Horse extends Animal{
    ...
    @Override
    public void sound(){
        System.out.println("Neigh");
    }
}

```

and

```

public class Cat extends Animal{
    ...
    @Override
    public void sound(){
        System.out.println("Meow");
    }
}

```

As you can see that although we had the common action for all subclasses sound() but there were different ways to do the same action. This is a perfect example of polymorphism (feature that allows us to perform a single

action in different ways). It would not make any sense to just call the generic sound() method as each Animal has a different sound. Thus we can say that the action this method performs is based on the type of object.

## What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. As we have seen in the above example that we have defined the method sound() and have the multiple implementations of it in the different-2 sub classes.

Which sound() method will be called is determined at runtime so the example we gave above is a **runtime polymorphism example**.

Types of polymorphism and method overloading & overriding are covered in the separate tutorials. You can refer them here:

1. [Method Overloading in Java](#) – This is an example of compile time (or static polymorphism)
2. [Method Overriding in Java](#) – This is an example of runtime time (or dynamic polymorphism)
3. [Types of Polymorphism – Runtime and compile time](#) – This is our next tutorial where we have covered the types of polymorphism in detail. I would recommend you to go through method overloading and overriding before going through this topic.

Lets write down the complete code of it:

## Example 1: Polymorphism in Java

### Runtime Polymorphism example:

Animal.java

```
public class Animal{
    public void sound(){
        System.out.println("Animal is making a sound");
    }
}
```

Horse.java

```
class Horse extends Animal{
    @Override
    public void sound(){
        System.out.println("Neigh");
    }
    public static void main(String args[]){
        Animal obj = new Horse();
        obj.sound();
    }
}
```

```
}  
}
```

Output:

Neigh

Cat.java

```
public class Cat extends Animal{  
    @Override  
    public void sound(){  
        System.out.println("Meow");  
    }  
    public static void main(String args[]){  
        Animal obj = new Cat();  
        obj.sound();  
    }  
}
```

Output:

Meow

## Example 2: Compile time Polymorphism

Method Overloading on the other hand is a compile time polymorphism example.

```
class Overload  
{  
    void demo (int a)  
    {  
        System.out.println ("a: " + a);  
    }  
    void demo (int a, int b)  
    {  
        System.out.println ("a and b: " + a + "," + b);  
    }  
    double demo(double a) {  
        System.out.println("double a: " + a);  
        return a*a;  
    }  
}  
class MethodOverloading  
{  
    public static void main (String args [])  
    {  
        Overload Obj = new Overload();  
        double result;  
        Obj .demo(10);  
        Obj .demo(10, 20);  
        result = Obj .demo(5.5);  
        System.out.println("O/P : " + result);  
    }  
}
```

Here the method `demo()` is overloaded 3 times: first method has 1 int parameter, second method has 2 int parameters and third one is having

double parameter. Which method is to be called is determined by the arguments we pass while calling methods. This happens at ~~runtime~~ compile time so this type of polymorphism is known as compile time polymorphism.

### Output:

```
a: 10
a and b: 10,20
double a: 5.5
O/P : 30.25
```

## Types of polymorphism in java- Runtime and Compile time polymorphism

In the last tutorial we discussed [Polymorphism in Java](#). In this guide we will see **types of polymorphism**. There are two types of polymorphism in java:

- 1) **Static Polymorphism** also known as compile time polymorphism
- 2) **Dynamic Polymorphism** also known as runtime polymorphism

### Compile time Polymorphism (or Static polymorphism)

Polymorphism that is resolved during compiler time is known as static polymorphism. Method overloading is an example of compile time polymorphism.

**Method Overloading:** This allows us to have more than one method having the same name, if the parameters of methods are different in number, sequence and data types of parameters. We have already discussed Method overloading here: If you didn't read that guide, refer: [Method Overloading in Java](#)

### Example of static Polymorphism

Method overloading is one of the way java supports static polymorphism. Here we have two definitions of the same method add() which add method would be called is determined by the parameter list at the compile time. That is the reason this is also known as compile time polymorphism.

```
class SimpleCalculator
{
    int add(int a, int b)
    {
```

```

        return a+b;
    }
    int add(int a, int b, int c)
    {
        return a+b+c;
    }
}
public class Demo
{
    public static void main(String args[])
    {
        SimpleCalculator obj = new SimpleCalculator();
        System.out.println(obj.add(10, 20));
        System.out.println(obj.add(10, 20, 30));
    }
}

```

**Output:**

```

30
60

```

## Runtime Polymorphism (or Dynamic polymorphism)

It is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, that's why it is called runtime polymorphism. I have already discussed method overriding in detail in a separate tutorial, refer it: [Method Overriding in Java](#).

### Example

In this example we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of parent class. In this example we have child class object assigned to the parent class reference so in order to determine which method would be called, the type of the object would be determined at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

To understand the concept of overriding, you should have the basic knowledge of [inheritance in Java](#).

```

class ABC{
    public void myMethod(){
        System.out.println("Overridden Method");
    }
}
public class XYZ extends ABC{

    public void myMethod(){
        System.out.println("Overriding Method");
    }
}

```



```

    public static void main(String args[]){
        ABC obj = new XYZ();
        obj.myMethod();
    }
}

```

### Output:

#### Overriding Method

When an overridden method is called through a reference of parent class, then type of the object determines which method is to be executed. Thus, this determination is made at run time.

Since both the classes, child class and parent class have the same method `animalSound`. Which version of the method(child class or parent class) will be called is determined at runtime by JVM.

### Few more overriding examples:

```

ABC obj = new ABC();
obj.myMethod();
// This would call the myMethod() of parent class ABC

XYZ obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ

ABC obj = new XYZ();
obj.myMethod();
// This would call the myMethod() of child class XYZ

```

In the third case the method of child class is to be executed because which method is to be executed is determined by the type of object and since the object belongs to the child class, the child class version of `myMethod()` is called.

## Java - Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A **Package** can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and namespace management.

Some of the existing packages in Java are –

- **java.lang** – bundles the fundamental classes
- **java.io** – classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a

programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

## Creating a Package

While creating a package, you should choose a name for the package and include a **package** statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package.

The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.

If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.

To compile the Java programs with package statements, you have to use -d option as shown below.

```
javac -d Destination_folder file_name.java
```

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

### Example

Let us look at an example that creates a package called **animals**. It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes and interfaces.

Following package example contains interface named *animals* –

```
/* File name : Animal.java */
package animals;

interface Animal {
    public void eat();
    public void travel();
}
```

Now, let us implement the above interface in the same package *animals* –

```
package animals;
/* File name : MammalInt.java */

public class MammalInt implements Animal {

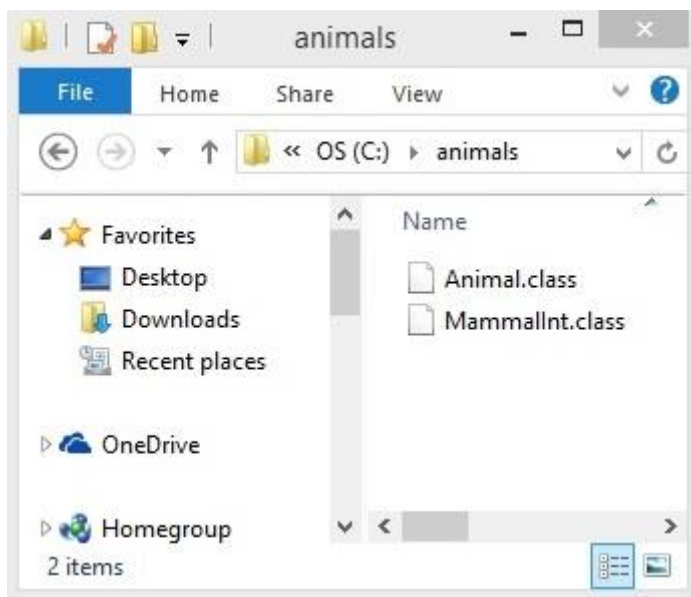
    public void eat() {
        System.out.println("Mammal eats");
    }
}
```

```
public void travel() {  
    System.out.println("Mammal travels");  
}  
  
public int noOfLegs() {  
    return 0;  
}  
  
public static void main(String args[]) {  
    MammalInt m = new MammalInt();  
    m.eat();  
    m.travel();  
}  
}
```

Now compile the java files as shown below –

```
$ javac -d . Animal.java  
$ javac -d . MammalInt.java
```

Now a package/folder with the name **animals** will be created in the current directory and these class files will be placed in it as shown below.



You can execute the class file within the package and get the result as shown below.

```
Mammal eats  
Mammal travels
```

## The import Keyword

If a class wants to use another class in the same package, the package name need not be used. Classes in the same package find each other without any special syntax.

### Example

Here, a class named Boss is added to the payroll package that already contains Employee. The Boss can then refer to the Employee class without using the payroll prefix, as demonstrated by the following Boss class.

```
package payroll;
public class Boss {
    public void payEmployee(Employee e) {
        e.mailCheck();
    }
}
```

What happens if the Employee class is not in the payroll package? The Boss class must then use one of the following techniques for referring to a class in a different package.

- The fully qualified name of the class can be used. For example –

```
payroll.Employee
```

- The package can be imported using the import keyword and the wild card (\*). For example –

```
import payroll.*;
```

- The class itself can be imported using the import keyword. For example –

```
import payroll.Employee;
```

**Note** – A class file can contain any number of import statements. The import statements must appear after the package statement and before the class declaration.

## The Directory Structure of Packages

Two major results occur when a class is placed in a package –

- The name of the package becomes a part of the name of the class, as we just discussed in the previous section.
- The name of the package must match the directory structure where the corresponding bytecode resides.

Here is simple way of managing your files in Java –

Put the source code for a class, interface, enumeration, or annotation type in a text file whose name is the simple name of the type and whose extension is **.java**.

For example –

```
// File Name : Car.java
package vehicle;

public class Car {
    // Class implementation.
}
```

Now, put the source file in a directory whose name reflects the name of the package to which the class belongs –

```
....\vehicle\Car.java
```

Now, the qualified class name and pathname would be as follows –

- Class name → vehicle.Car
- Path name → vehicle\Car.java (in windows)

In general, a company uses its reversed Internet domain name for its package names.

**Example** – A company's Internet domain name is apple.com, then all its package names would start with com.apple. Each component of the package name corresponds to a subdirectory.

**Example** – The company had a com.apple.computers package that contained a Dell.java source file, it would be contained in a series of subdirectories like this –

```
....\com\apple\computers\Dell.java
```

At the time of compilation, the compiler creates a different output file for each class, interface and enumeration defined in it. The base name of the output file is the name of the type, and its extension is **.class**.

For example –

```
// File Name: Dell.java
package com.apple.computers;

public class Dell {
}

class Ups {
}
```

Now, compile this file as follows using -d option –

```
$javac -d . Dell.java
```

The files will be compiled as follows –

```
.\com\apple\computers\Dell.class
.\com\apple\computers\Ups.class
```

You can import all the classes or interfaces defined in `\com\apple\computers\` as follows –

```
import com.apple.computers.*;
```

Like the .java source files, the compiled .class files should be in a series of directories that reflect the package name. However, the path to the .class files does not have to be the same as the path to the .java source files. You can arrange your source and class directories separately, as –

```
<path-one>\sources\com\apple\computers\Dell.java
```

```
<path-two>\classes\com\apple\computers\Dell.class
```

By doing this, it is possible to give access to the classes directory to other programmers without revealing your sources. You also need to manage source and class files in this manner so that the compiler and the Java Virtual Machine (JVM) can find all the types your program uses.

The full path to the classes directory, <path-two>\classes, is called the class path, and is set with the CLASSPATH system variable. Both the compiler and the JVM construct the path to your .class files by adding the package name to the class path.

Say <path-two>\classes is the class path, and the package name is com.apple.computers, then the compiler and JVM will look for .class files in <path-two>\classes\com\apple\computers.

A class path may include several paths. Multiple paths should be separated by a semicolon (Windows) or colon (Unix). By default, the compiler and the JVM search the current directory and the JAR file containing the Java platform classes so that these directories are automatically in the class path.

## Set CLASSPATH System Variable

To display the current CLASSPATH variable, use the following commands in Windows and UNIX (Bourne shell) –

- In Windows → C:\> set CLASSPATH
- In UNIX → % echo \$CLASSPATH

To delete the current contents of the CLASSPATH variable, use –

- In Windows → C:\> set CLASSPATH =
- In UNIX → % unset CLASSPATH; export CLASSPATH

To set the CLASSPATH variable –

- In Windows → set CLASSPATH = C:\users\jack\java\classes
- In UNIX → % CLASSPATH = /home/jack/java/classes; export CLASSPATH

## UNIT: 3 (INTERFACES )

### INTERFACE:

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is similar to a class in the following ways –

- An interface can contain any number of methods.
- An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
- The byte code of an interface appears in a **.class** file.
- Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

However, an interface is different from a class in several ways, including –

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

## Declaring Interfaces

The **interface** keyword is used to declare an interface. Here is a simple example to declare an interface –

### Example

Following is an example of an interface –

```
/* File name : NameOfInterface.java */
import java.lang.*;
// Any number of import statements
```

```
public interface NameOfInterface {  
    // Any number of final, static fields  
    // Any number of abstract method declarations\  
}
```

Interfaces have the following properties –

- An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.

## Example

```
/* File name : Animal.java */  
interface Animal {  
    public void eat();  
    public void travel();  
}
```

## Implementing Interfaces

When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface. If a class does not perform all the behaviors of the interface, the class must declare itself as abstract.

A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

## Example

```
/* File name : MammalInt.java */  
public class MammalInt implements Animal {  
  
    public void eat() {  
        System.out.println("Mammal eats");  
    }  
  
    public void travel() {  
        System.out.println("Mammal travels");  
    }  
  
    public int noOfLegs() {  
        return 0;  
    }  
  
    public static void main(String args[]) {  
        MammalInt m = new MammalInt();  
    }  
}
```



```
m.eat();  
m.travel();  
}  
}
```

This will produce the following result –

## Output

```
Mammal eats  
Mammal travels
```

When overriding methods defined in interfaces, there are several rules to be followed –

- Checked exceptions should not be declared on implementation methods other than the ones declared by the interface method or subclasses of those declared by the interface method.
- The signature of the interface method and the same return type or subtype should be maintained when overriding the methods.
- An implementation class itself can be abstract and if so, interface methods need not be implemented.

When implementing interfaces, there are several rules –

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, in a similar way as a class can extend another class.

## Extending Interfaces

An interface can extend another interface in the same way that a class can extend another class. The **extends** keyword is used to extend an interface, and the child interface inherits the methods of the parent interface.

The following Sports interface is extended by Hockey and Football interfaces.

### Example

```
// Filename: Sports.java  
public interface Sports {  
    public void setHomeTeam(String name);  
    public void setVisitingTeam(String name);  
}  
  
// Filename: Football.java  
public interface Football extends Sports {  
    public void homeTeamScored(int points);  
    public void visitingTeamScored(int points);  
    public void endOfQuarter(int quarter);  
}
```

```
// Filename: Hockey.java
public interface Hockey extends Sports {
    public void homeGoalScored();
    public void visitingGoalScored();
    public void endOfPeriod(int period);
    public void overtimePeriod(int ot);
}
```

The Hockey interface has four methods, but it inherits two from Sports; thus, a class that implements Hockey needs to implement all six methods. Similarly, a class that implements Football needs to define the three methods from Football and the two methods from Sports.

## Extending Multiple Interfaces

A Java class can only extend one parent class. Multiple inheritance is not allowed. Interfaces are not classes, however, and an interface can extend more than one parent interface.

The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

For example, if the Hockey interface extended both Sports and Event, it would be declared as –

### Example

```
public interface Hockey extends Sports, Event
```

## Tagging Interfaces

The most common use of extending interfaces occurs when the parent interface does not contain any methods. For example, the MouseListener interface in the java.awt.event package extended java.util.EventListener, which is defined as –

### Example

```
package java.util;
public interface EventListener
{ }
```

An interface with no methods in it is referred to as a **tagging** interface. There are two basic design purposes of tagging interfaces –

**Creates a common parent** – As with the EventListener interface, which is extended by dozens of other interfaces in the Java API, you can use a tagging interface to create a common parent among a group of interfaces. For example, when an interface extends EventListener, the JVM knows that this particular interface is going to be used in an event delegation scenario.

**Adds a data type to a class** – This situation is where the term, tagging comes from. A class that implements a tagging interface does not need to define any methods (since the interface does not have any), but the class becomes an interface type through polymorphism.

Packages and Interfaces both acts as a container. The content in packages and interfaces can be used by the classes by importing and implementing it correspondingly. The basic difference between packages and interfaces is that a package contains a group of classes and interfaces whereas, an interface contains methods and fields. Let's study some other differences with the help of comparison chart.

### **Content: Packages Vs Interfaces in Java**

1. Comparison Chart
2. Definition
3. Key Differences
4. Conclusion

#### **Comparison Chart**

**Basic:**A package can be imported.

An interface can be extended by another interface and implemented by the class.

#### **Keyword:**

Packages are created using "Package" keyword.

Interface are created using "Interface" keyword.

#### **Syntax:**

```
package package_name;
public class class_name{
.
(body of class)
.
}
```

```
interface interface_name{
variable declaration;
method declaration;
}
```

#### **Access:**

A package can be imported.

An interface can be extended by another interface and implemented by the class.

### **Access keyword:**

Packages can be imported using "import" keyword.

Interfaces can be implemented using "implement" keyword.

### **Definition of Packages**

Packages are collection or groups of the variety of classes and interfaces. The classes in packages are related to each other in some scope or by inheritance. You can also create your package and use it for your program.

### **Creating a package**

For creating a package just follow the following steps.

1. Open a file and then declare the name of the package at the top of the file, like [ package package\_name; ] the package name is the name you want to give to the package.
2. Next, you define a class that you want to put in the package, and remember that you declare it public.
3. Save the file as a .java file and then compile the file, then ".class" is obtain for that file.
4. To create a package for this file the command used is "javac -d . file\_name.java. You can see that the package is created containing a ".class" file in the current directory. To place it in parent directory use "javac -d . . file\_name.java" command.
5. You can also create a subpackage by declaring subpackage name as [ package package\_name1. package\_name2; ] at the top of the file.
6. package Mypackage;
7. public class myclass{
8. public void displayMypackage( ){
9. system.out.println("method displayMypackage of class myclass of package Mypackage".

### **Using the Package**

The packages created or available in the directory can be used in the program by using an import statement. The keyword used to import any package in your program is "import". The import statement can be written in two ways, or you can say that there are two ways to access any package. First, if you want to use a particular class from a package, The "import" keyword is followed by the package name further followed by the dot operator and the class name which you want to use from the package. Second, if you want to use many classes that are contained in the packages,

then the import keyword is followed by the package name further followed by the dot and the " \* " operator.

1. import package\_name . class\_name;
2. or
3. import package\_name . \*;

In above code, you can see the \* sign which indicates that second method imports all the classes contained in the packages.

Now, let's view the use of the package with an example.

1. import Mypackage . myclass{
2. class TestMypackage{
3. public static void main(string args[ ]){
4. myclass ob1= new myclass( );
5. ob1.displayMypackage( );
6. }
7. }
8. //output
9. method displayMypackage of class myclass of package Mypackage.

In above code, the class TestMypackage has imported the package Mypackage and used its displayMypackage( ) method.

## Definition of Interface

Interface is a kind of a class, but, differs in a sense that the methods declared in the interface are abstract that means the methods are only declared but not defined. The fields in the interface are always public, static, final. The fields must be initialized at the time of declaration. The methods declared by the interface are defined by the class which implements that interface according to its requirement. As the methods in the interface do not perform any function, so there is no use of creating any object of the interface. Hence, no object can be created for the interface.

The interface can also inherit the other interface but, the class inheriting such an interface must also implement all the methods of the inherited interface. As the fields are initialized at the time of their declaration in the interface, so there is no need of constructor in the interface hence, the interface doesn't contain any constructor. Let's see the example of creating and using an interface.

1. interface Area {
2. float pi= 3.14;
3. float find\_area(float a, float b){
4. }
5. class Circle implements Area{
6. float find\_area(float a, float b){

```

7. return (pi*a*a);
8. }
9. Class Shapes{
10. public static void main(string args[ ]){
11. Area A=new Area ( );
12. Circle C= new circle ( );
13. A=C;
14. float F= Area. find_area(10,10);
15. system.out.println("Area of the circle is :"+F);
16. }

```

In above code, we had created an interface Area, and the class Circle has implemented the interface Area. The field "pi" has been initialized in the interface at the time of its declaration. The class Circle has defined the abstract method of the class area according to its requirement.

**Exception Handling in Java:** The Exception Handling in Java is one of the powerful *mechanism to handle the runtime errors so that normal flow of the application can be maintained.*

## What is Exception in Java

**Dictionary Meaning:** Exception is an abnormal condition.

Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

**What is Exception Handling :** Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

**Advantage of Exception Handling :** The core advantage of exception handling is **to maintain the normal flow of the application.** An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5; //exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;

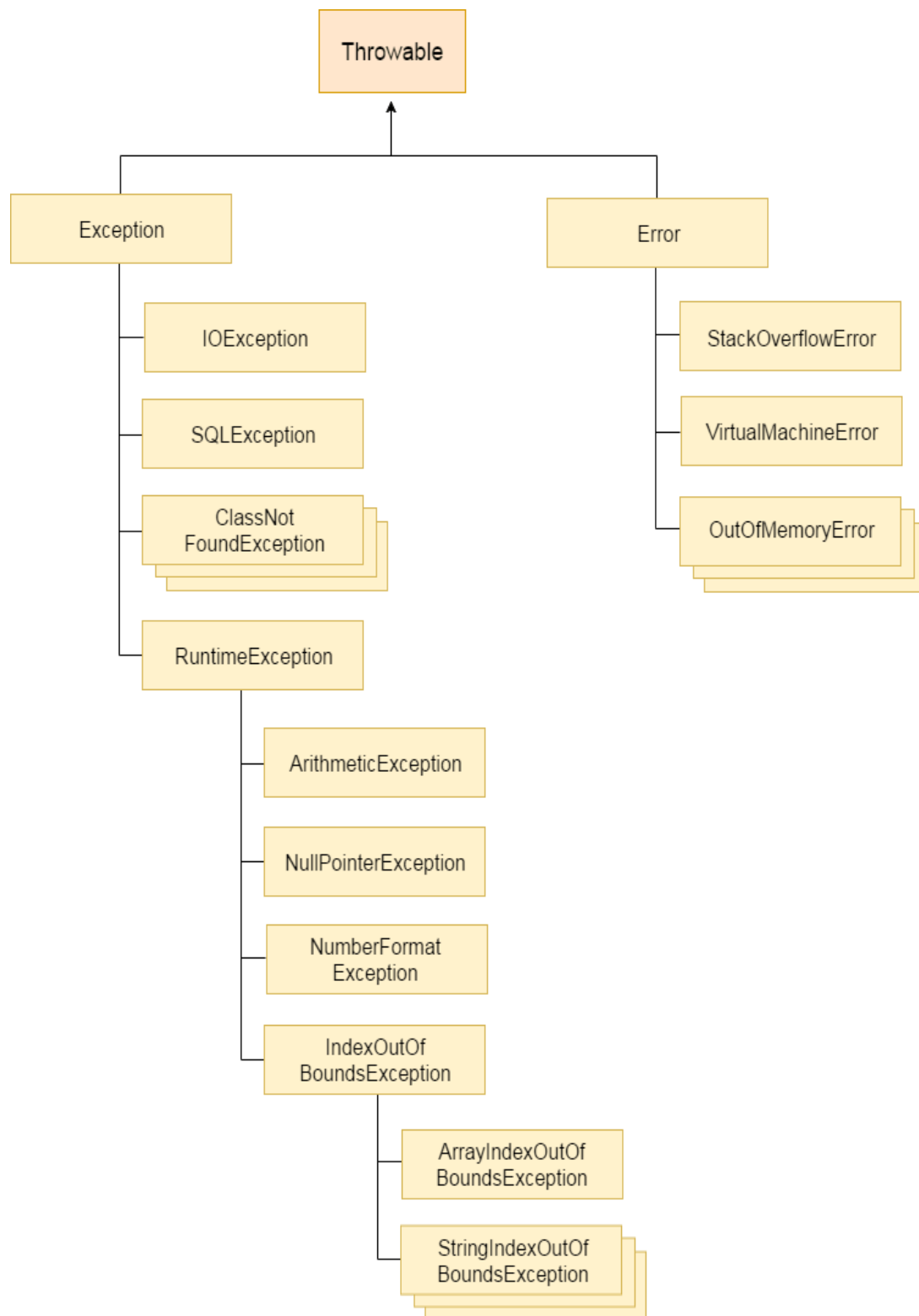
```

10. statement 10; Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

- What is the difference between checked and unchecked exceptions?
- What happens behind the code `int data=50/0;?`
- Why use multiple catch block?
- Is there any possibility when finally block is not executed?
- What is exception propagation?
- What is the difference between throw and throws keyword?
- What are the 4 rules for using exception handling with method overriding?

## Hierarchy of Java Exception classes

The `java.lang.Throwable` class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



## Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:



1. Checked Exception
2. Unchecked Exception
3. Error



## Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

## Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

```
1. public class JavaExceptionExample{  
2.   public static void main(String args[]){  
3.     try{  
4.       //code that may raise exception  
5.       int data=100/0;  
6.     }catch(ArithmeticException e){System.out.println(e);}  
7.     //rest code of the program  
8.     System.out.println("rest of the code...");  
9.   }  
10. }
```

**Test it Now**

Output:

Exception in thread main java.lang.ArithmeticException:/ by zero  
rest of the code...

In the above example, 100/0 raises an ArithmeticException which is handled by a try-catch block.

---

---

## Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

### 1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

1. `int a=50/0;//ArithmeticException`

---

---

### 2) A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

1. `String s=null;`  
2. `System.out.println(s.length());//NullPointerException`

---

---

### 3) A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

1. `String s="abc";`  
2. `int i=Integer.parseInt(s);//NumberFormatException`

---

---

### 4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

1. `int a[]=new int[5];`  
2. `a[10]=50; //ArrayIndexOutOfBoundsException`

---

## Java Exceptions Index

- [1. Java Try-Catch Block](#)
- [2. Java Multiple Catch Block](#)
- [3. Java Nested Try](#)
- [4. Java Finally Block](#)
- [5. Java Throw Keyword](#)
- [6. Java Exception Propagation](#)
- [7. Java Throws Keyword](#)
- [8. Java Throw vs Throws](#)
- [9. Java Final vs Finally vs Finalize](#)
- [10. Java Exception Handling with Method Overriding](#)
- [11. Java Custom Exceptions](#)

## Java - Files and I/O

The java.io package contains nearly every class you might ever need to perform input and output (I/O) in Java. All these streams represent an input source and an output destination. The stream in the java.io package supports many data such as primitives, object, localized characters, etc.

### Stream

A stream can be defined as a sequence of data. There are two kinds of Streams –

- **InPutStream** – The InputStream is used to read data from a source.
- **OutPutStream** – The OutputStream is used for writing data to a destination.



Java provides strong but flexible support for I/O related to files and networks but this tutorial covers very basic functionality related to streams and I/O. We will see the most commonly used examples one by one –

### Byte Streams

Java byte streams are used to perform input and output of 8-bit bytes. Though there are many classes related to byte streams but the most frequently used classes are, **FileInputStream** and **FileOutputStream**. Following is an example which makes use of these two classes to copy an input file into an output file –

#### Example

```
import java.io.*;
public class CopyFile {
```

```

public static void main(String args[]) throws IOException {
    FileInputStream in = null;
    FileOutputStream out = null;

    try {
        in = new FileInputStream("input.txt");
        out = new FileOutputStream("output.txt");

        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }
    } finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
    }
}

```

Now let's have a file **input.txt** with the following content –

This is test for copy file.

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following –

```

$javac CopyFile.java
$java CopyFile

```

## Character Streams

Java **Byte** streams are used to perform input and output of 8-bit bytes, whereas Java **Character** streams are used to perform input and output for 16-bit unicode. Though there are many classes related to character streams but the most frequently used classes are, **FileReader** and **FileWriter**. Though internally FileReader uses FileInputStream and FileWriter uses FileOutputStream but here the major difference is that FileReader reads two bytes at a time and FileWriter writes two bytes at a time.

We can re-write the above example, which makes the use of these two classes to copy an input file (having unicode characters) into an output file –

### Example

```

import java.io.*;
public class CopyFile {

    public static void main(String args[]) throws IOException {
        FileReader in = null;
        FileWriter out = null;
    }
}

```

```

try {
    in = new FileReader("input.txt");
    out = new FileWriter("output.txt");

    int c;
    while ((c = in.read()) != -1) {
        out.write(c);
    }
}finally {
    if (in != null) {
        in.close();
    }
    if (out != null) {
        out.close();
    }
}
}

```

Now let's have a file **input.txt** with the following content –

This is test for copy file.

As a next step, compile the above program and execute it, which will result in creating output.txt file with the same content as we have in input.txt. So let's put the above code in CopyFile.java file and do the following –

```

$javac CopyFile.java
$java CopyFile

```

## Standard Streams

All the programming languages provide support for standard I/O where the user's program can take input from a keyboard and then produce an output on the computer screen. If you are aware of C or C++ programming languages, then you must be aware of three standard devices STDIN, STDOUT and STDERR. Similarly, Java provides the following three standard streams –

- **Standard Input** – This is used to feed the data to user's program and usually a keyboard is used as standard input stream and represented as **System.in**.
- **Standard Output** – This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as **System.out**.
- **Standard Error** – This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as **System.err**.

Following is a simple program, which creates **InputStreamReader** to read standard input stream until the user types a "q" –

### Example

[Live Demo](#)

```

import java.io.*;
public class ReadConsole {

    public static void main(String args[]) throws IOException {
        InputStreamReader cin = null;

        try {
            cin = new InputStreamReader(System.in);
            System.out.println("Enter characters, 'q' to quit.");
            char c;
            do {
                c = (char) cin.read();
                System.out.print(c);
            } while(c != 'q');
        } finally {
            if (cin != null) {
                cin.close();
            }
        }
    }
}

```

Let's keep the above code in ReadConsole.java file and try to compile and execute it as shown in the following program. This program continues to read and output the same character until we press 'q' –

```

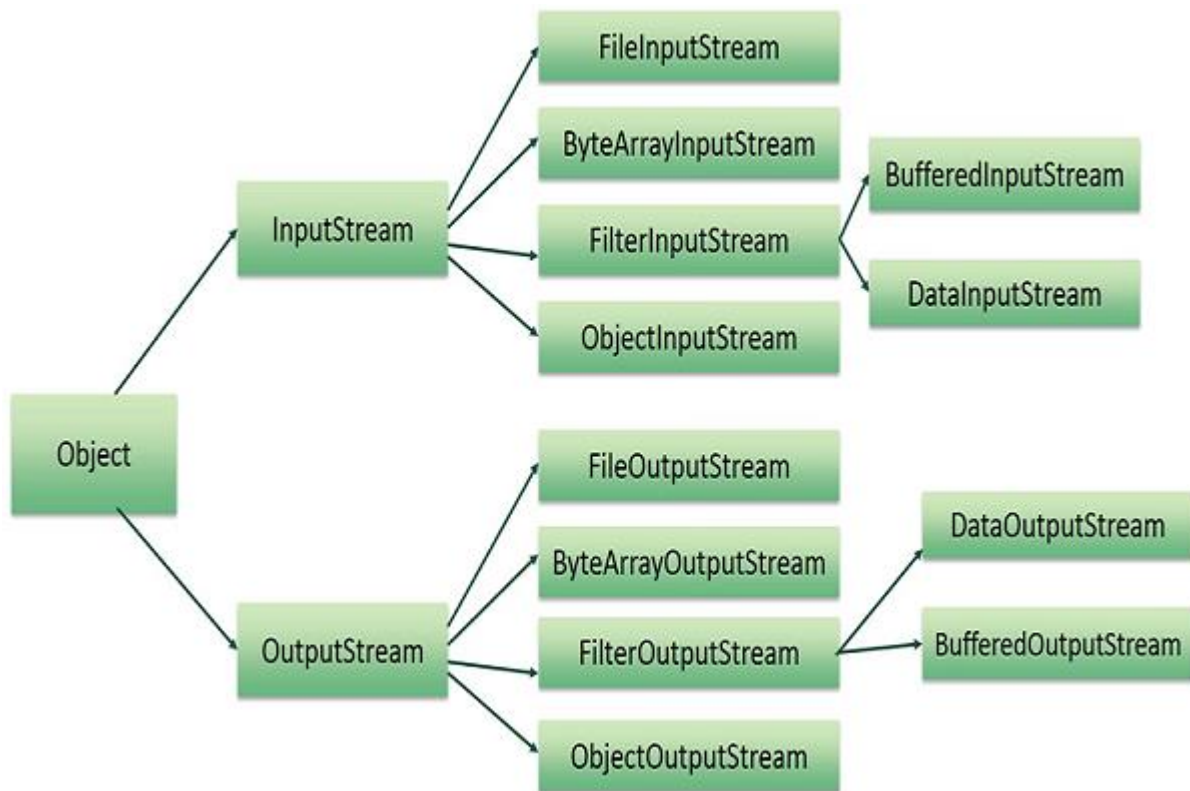
$javac ReadConsole.java
$java ReadConsole
Enter characters, 'q' to quit.
1
1
e
e
q
q

```

## Reading and Writing Files

As described earlier, a stream can be defined as a sequence of data. The **InputStream** is used to read data from a source and the **OutputStream** is used for writing data to a destination.

Here is a hierarchy of classes to deal with Input and Output streams.



The two important streams are **FileInputStream** and **FileOutputStream**, which would be discussed in this tutorial.

## FileInputStream

This stream is used for reading data from the files. Objects can be created using the keyword **new** and there are several types of constructors available.

Following constructor takes a file name as a string to create an input stream object to read the file –

```
InputStream f = new FileInputStream("C:/java/hello");
```

Following constructor takes a file object to create an input stream object to read the file. First we create a file object using File() method as follows –

```
File f = new File("C:/java/hello");
InputStream f = new FileInputStream(f);
```

Once you have *InputStream* object in hand, then there is a list of helper methods which can be used to read to stream or to do other operations on the stream.

Sr.No.	Method & Description
1	<b>public void close() throws IOException{</b> This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.



2	<b>protected void finalize()throws IOException {}</b> This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	<b>public int read(int r)throws IOException{}</b> This method reads the specified byte of data from the InputStream. Returns an int. Returns the next byte of data and -1 will be returned if it's the end of the file.
4	<b>public int read(byte[] r) throws IOException{}</b> This method reads r.length bytes from the input stream into an array. Returns the total number of bytes read. If it is the end of the file, -1 will be returned.
5	<b>public int available() throws IOException{}</b> Gives the number of bytes that can be read from this file input stream. Returns an int.

There are other important input streams available, for more detail you can refer to the following links –

- [ByteArrayInputStream](#)
- [DataInputStream](#)

## FileOutputStream

FileOutputStream is used to create a file and write data into it. The stream would create a file, if it doesn't already exist, before opening it for output.

Here are two constructors which can be used to create a FileOutputStream object.

Following constructor takes a file name as a string to create an input stream object to write the file –

```
OutputStream f = new FileOutputStream("C:/java/hello")
```

Following constructor takes a file object to create an output stream object to write the file. First, we create a file object using File() method as follows –

```
File f = new File("C:/java/hello");
OutputStream f = new FileOutputStream(f);
```

Once you have *OutputStream* object in hand, then there is a list of helper methods, which can be used to write to stream or to do other operations on the stream.

Sr.No.	Method & Description
1	<b>public void close() throws IOException{}</b>

	This method closes the file output stream. Releases any system resources associated with the file. Throws an IOException.
2	<b>protected void finalize()throws IOException {}</b> This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IOException.
3	<b>public void write(int w)throws IOException{}</b> This methods writes the specified byte to the output stream.
4	<b>public void write(byte[] w)</b> Writes w.length bytes from the mentioned byte array to the OutputStream.

There are other important output streams available, for more detail you can refer to the following links –

- ByteArrayOutputStream
- DataOutputStream

## Example

Following is the example to demonstrate InputStream and OutputStream –

```
import java.io.*;
public class FileStreamTest {

    public static void main(String args[]) {

        try {
            byte bWrite [] = {11,21,3,40,5};
            OutputStream os = new FileOutputStream("test.txt");
            for(int x = 0; x < bWrite.length ; x++) {
                os.write( bWrite[x] );    // writes the bytes
            }
            os.close();

            InputStream is = new FileInputStream("test.txt");
            int size = is.available();

            for(int i = 0; i < size; i++) {
                System.out.print((char)is.read() + " ");
            }
            is.close();
        } catch (IOException e) {
            System.out.print("Exception");
        }
    }
}
```

The above code would create file test.txt and would write given numbers in binary format. Same would be the output on the stdout screen.

## File Navigation and I/O

There are several other classes that we would be going through to get to know the basics of File Navigation and I/O.

- File Class
- FileReader Class
- FileWriter Class

## Directories in Java

A directory is a File which can contain a list of other files and directories. You use **File** object to create directories, to list down files available in a directory. For complete detail, check a list of all the methods which you can call on File object and what are related to directories.

### Creating Directories

There are two useful **File** utility methods, which can be used to create directories –

- The **mkdir( )** method creates a directory, returning true on success and false on failure. Failure indicates that the path specified in the File object already exists, or that the directory cannot be created because the entire path does not exist yet.
- The **mkdirs()** method creates both a directory and all the parents of the directory.

Following example creates "/tmp/user/java/bin" directory –

#### Example

```
import java.io.File;
public class CreateDir {

    public static void main(String args[]) {
        String dirname = "/tmp/user/java/bin";
        File d = new File(dirname);

        // Create directory now.
        d.mkdirs();
    }
}
```

Compile and execute the above code to create "/tmp/user/java/bin".

**Note** – Java automatically takes care of path separators on UNIX and Windows as per conventions. If you use a forward slash (/) on a Windows version of Java, the path will still resolve correctly.

## Listing Directories

You can use **list( )** method provided by **File** object to list down all the files and directories available in a directory as follows –

### Example

```
import java.io.File;
public class ReadDir {

    public static void main(String[] args) {
        File file = null;
        String[] paths;

        try {
            // create new file object
            file = new File("/tmp");

            // array of files and directory
            paths = file.list();

            // for each name in the path array
            for(String path:paths) {
                // prints filename and directory name
                System.out.println(path);
            }
        } catch (Exception e) {
            // if any error occurs
            e.printStackTrace();
        }
    }
}
```

This will produce the following result based on the directories and files available in your **/tmp** directory –

### Output

```
test1.txt
test2.txt
ReadDir.java
ReadDir.class
```

**Java Applet** Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

### Advantage of Applet

- It works at client side so less response time.
- Secured

- It can be executed by browsers running under many platforms, including Linux, Windows, Mac OS etc.

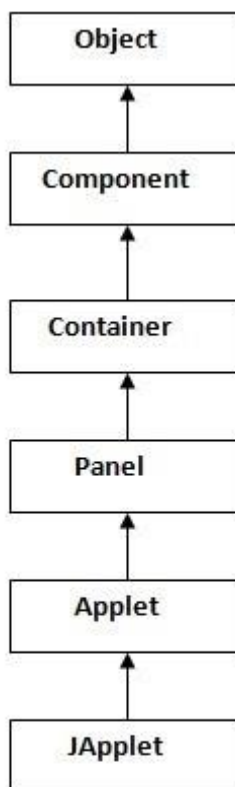
## Drawback of Applet

- Plugin is required at client browser to execute applet.

### Do You Know

- Who is responsible to manage the life cycle of an applet ?
- How to perform animation in applet ?
- How to paint like paint brush in applet ?
- How to display digital clock in applet ?
- How to display analog clock in applet ?
- How to communicate two applets ?

## Hierarchy of Applet

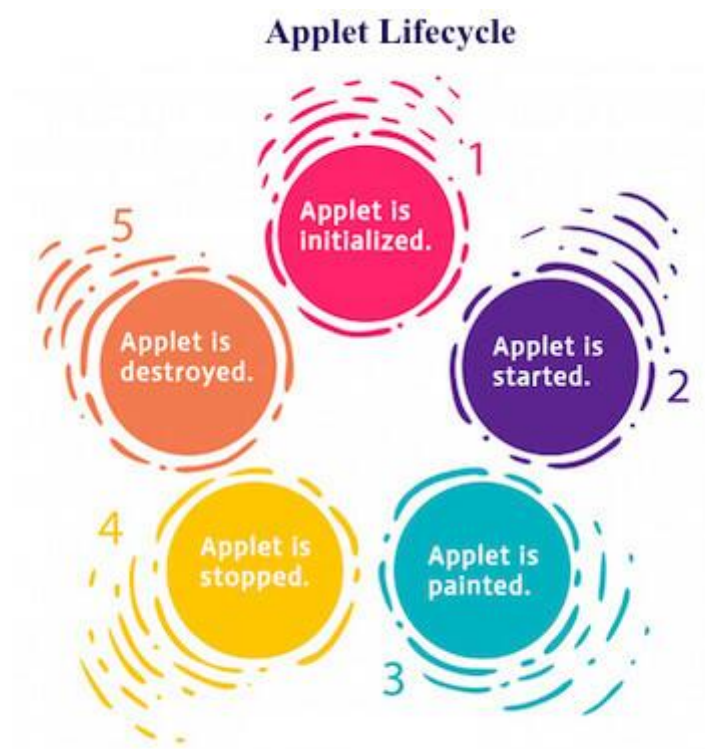


As displayed in the above diagram, Applet class extends Panel. Panel class extends

Container which is the subclass of Component.

## Lifecycle of Java Applet

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



---

## Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

### java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialize the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.

3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

## java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

---

## Who is responsible to manage the life cycle of an applet?

Java Plug-in software.

---

## How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

---

## Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6.     public void paint(Graphics g){
7.         g.drawString("welcome",150,150);
8.     }
9.
10. }
```

Note: class must be public because its object is created by Java Plugin software that resides on the browser.

### myapplet.html

```
1. <html>
2. <body>
3. <applet code="First.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

### Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
1. //First.java
2. import java.applet.Applet;
3. import java.awt.Graphics;
4. public class First extends Applet{
5.
6. public void paint(Graphics g){
7. g.drawString("welcome to applet",150,150);
8. }
9.
10.}
11./*
12.<applet code="First.class" width="300" height="300">
13.</applet>
14.*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

## Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

### Commonly used methods of Graphics class:



1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

## Example of Graphics in applet:

```
1. import java.applet.Applet;
2. import java.awt.*;
3.
4. public class GraphicsDemo extends Applet{
5.
6.     public void paint(Graphics g){
7.         g.setColor(Color.red);
8.         g.drawString("Welcome",50, 50);
9.         g.drawLine(20,30,20,300);
10.        g.drawRect(70,100,30,30);
11.        g.fillRect(170,100,30,30);
12.        g.drawOval(70,200,30,30);
13.
14.        g.setColor(Color.pink);
15.        g.fillOval(170,200,30,30);
```

```
16. g.drawArc(90,150,30,30,30,270);
17. g.fillArc(270,150,30,30,0,180);
18.
19. }
20. }
```

## myapplet.html

```
1. <html>
2. <body>
3. <applet code="GraphicsDemo.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>
```

## Displaying Image in Applet

Applet is mostly used in games and animation. For this purpose image is required to be displayed. The java.awt.Graphics class provide a method drawImage() to display the image.

### Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

### How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image. Syntax:

1. **public** Image getImage(URL u, String image){}

### Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

---

### Example of displaying image in applet:

```

1. import java.awt.*;
2. import java.applet.*;
3.
4.
5. public class DisplayImage extends Applet {
6.
7.     Image picture;
8.
9.     public void init() {
10.         picture = getImage(getDocumentBase(),"sonoo.jpg");
11.     }
12.
13.     public void paint(Graphics g) {
14.         g.drawImage(picture, 30,30, this);
15.     }
16.
17. }

```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

## myapplet.html

```

1. <html>
2. <body>
3. <applet code="DisplayImage.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

```

## Animation in Applet

Applet is mostly used in games and animation. For this purpose image is required to be moved.

### Example of animation in applet:

```

1. import java.awt.*;
2. import java.applet.*;
3. public class AnimationExample extends Applet {
4.

```

```

5. Image picture;
6.
7. public void init() {
8.     picture =getImage(getDocumentBase(),"bike_1.gif");
9. }
10.
11. public void paint(Graphics g) {
12.     for(int i=0;i<500;i++){
13.         g.drawImage(picture, i,30, this);
14.
15.         try{Thread.sleep(100);}catch(Exception e){}
16.     }
17. }
18. }

```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

## myapplet.html

```

1. <html>
2. <body>
3. <applet code="DisplayImage.class" width="300" height="300">
4. </applet>
5. </body>
6. </html>

```

DESIGN OF USER INTERFACE:

## Java Swing

**Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

## Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

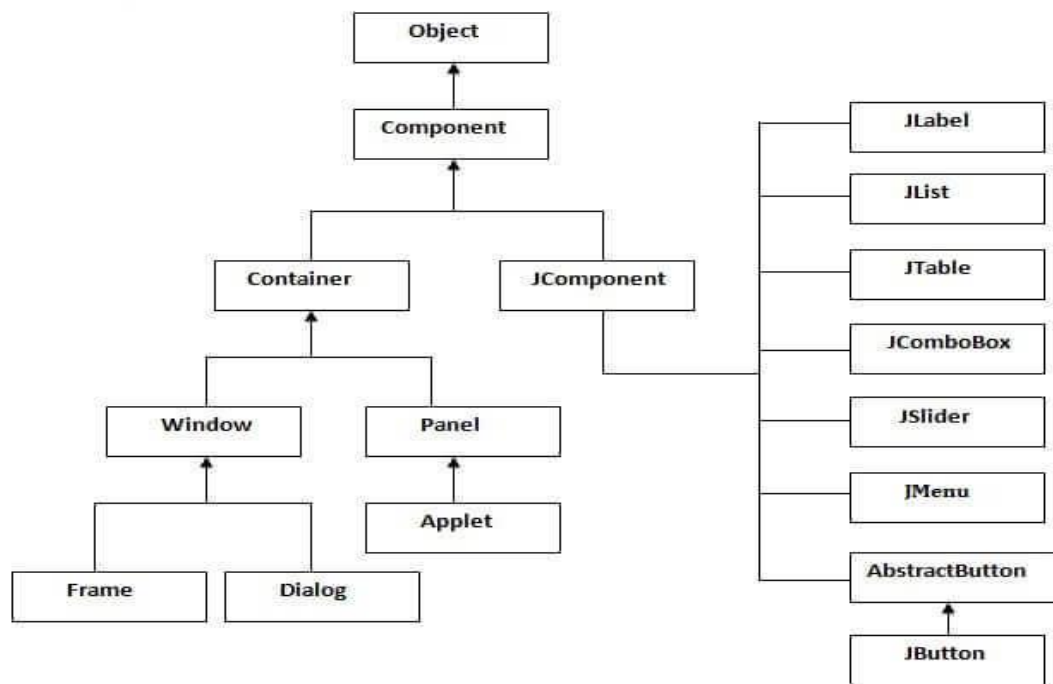
No.	Java AWT	Java Swing
1)	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .
4)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

## What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

### Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



## Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

## Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)

- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

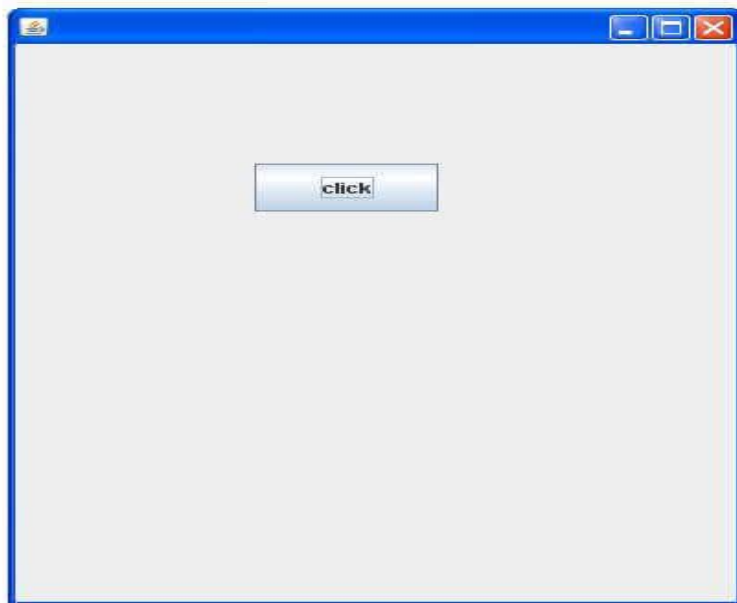
---

## Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4.   JFrame f=new JFrame();//creating instance of JFrame
5.
6.   JButton b=new JButton("click");//creating instance of JButton
7.   b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9.   f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
14. }
15. }
```



## Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
1. import javax.swing.*;
2. public class Simple {
3.     JFrame f;
4.     Simple(){
5.         f=new JFrame();//creating instance of JFrame
6.
7.         JButton b=new JButton("click");//creating instance of JButton
8.         b.setBounds(130,100,100, 40);
9.
10.        f.add(b);//adding button in JFrame
11.
12.        f.setSize(400,500);//400 width and 500 height
13.        f.setLayout(null);//using no layout managers
14.        f.setVisible(true);//making the frame visible
15.    }
16.
17.    public static void main(String[] args) {
18.        new Simple();
19.    }
20.}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

---

## Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
1. import javax.swing.*;
2. public class Simple2 extends JFrame{//inheriting JFrame
3.     JFrame f;
4.     Simple2(){
5.         JButton b=new JButton("click");//create button
6.         b.setBounds(130,100,100, 40);
```



```
7. _____  
8. add(b);//adding button on frame  
9. setSize(400,500);  
10. setLayout(null);  
11. setVisible(true);  
12. }  
13. public static void main(String[] args) {  
14. new Simple2();  
15. }}
```

[download this example](#)

### *What we will learn in Swing Tutorial*

- [JButton class](#)
- [JRadioButton class](#)
- [JTextArea class](#)
- [JComboBox class](#)
- [JTable class](#)
- [JColorChooser class](#)
- [JProgressBar class](#)
- [JSlider class](#)
- [Digital Watch](#)
- [Graphics in swing](#)
- [Displaying image](#)
- [Edit menu code for Notepad](#)
- [OpenDialog Box](#)
- [Notepad](#)
- [Puzzle Game](#)
- [Pic Puzzle Game](#)
- [Tic Tac Toe Game](#)
- [BorderLayout](#)
- [GridLayout](#)
- [FlowLayout](#)
- [CardLayout](#)

## Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

## JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public class** JButton **extends** AbstractButton **implements** Accessible

### Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

### Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.

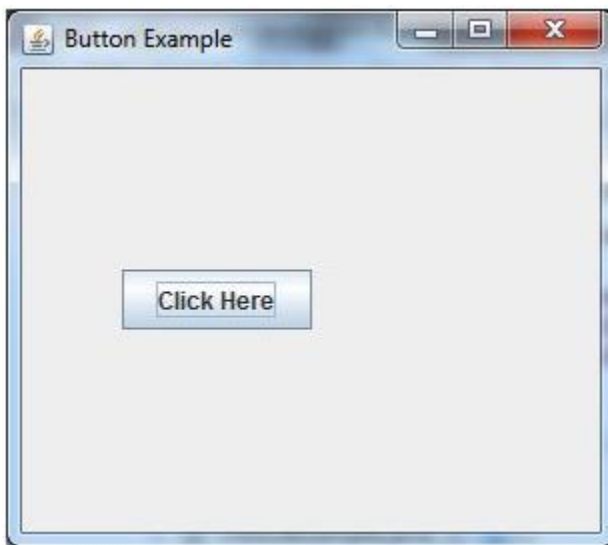
void addActionListener(ActionListener  
a)

It is used to add the [action listener](#) to this object.

## Java JButton Example

```
1. import javax.swing.*;
2. public class ButtonExample {
3. public static void main(String[] args) {
4.     JFrame f=new JFrame("Button Example");
5.     JButton b=new JButton("Click Here");
6.     b.setBounds(50,100,95,30);
7.     f.add(b);
8.     f.setSize(400,400);
9.     f.setLayout(null);
10.    f.setVisible(true);
11. }
12. }
```

Output:



---

## Java JButton Example with ActionListener

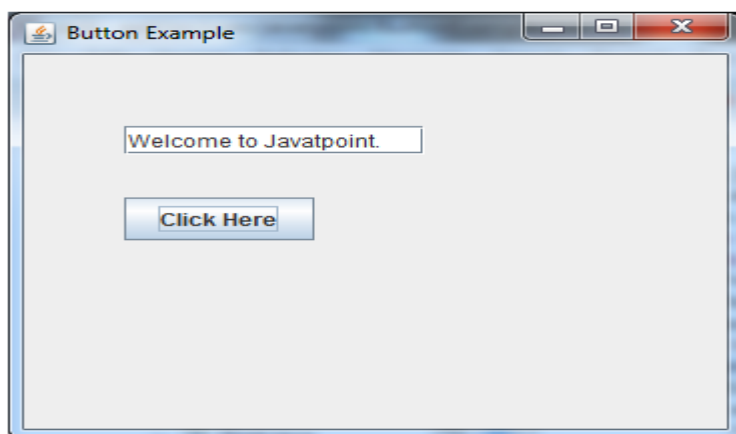
```
1. import java.awt.event.*;
2. import javax.swing.*;
3. public class ButtonExample {
4. public static void main(String[] args) {
5.     JFrame f=new JFrame("Button Example");
```

```

6.   final JTextField tf=new JTextField();
7.   tf.setBounds(50,50, 150,20);
8.   JButton b=new JButton("Click Here");
9.   b.setBounds(50,100,95,30);
10.  b.addActionListener(new ActionListener(){
11.  public void actionPerformed(ActionEvent e){
12.      tf.setText("Welcome to Javatpoint.");
13.  }
14.  });
15.  f.add(b);f.add(tf);
16.  f.setSize(400,400);
17.  f.setLayout(null);
18.  f.setVisible(true);
19. }
20. }

```

Output:




---

## Example of displaying image on the button:

```

1. import javax.swing.*;
2. public class ButtonExample{
3.     ButtonExample(){
4.         JFrame f=new JFrame("Button Example");
5.         JButton b=new JButton(new ImageIcon("D:\\icon.png"));
6.         b.setBounds(100,100,100, 40);
7.         f.add(b);
8.         f.setSize(300,400);
9.         f.setLayout(null);
10.        f.setVisible(true);
11.        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```

12. }
13. public static void main(String[] args) {
14.     new ButtonExample();
15. }
16. }

```

Output:



## Java JLabel

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

### JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

1. **public class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

### Commonly used Constructors:

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified

	image.
JLabel(String s, Icon i, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.

## Commonly used Methods:

Methods	Description
String getText()	t returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

## Java JLabel Example

```

1. import javax.swing.*;
2. class LabelExample
3. {
4.     public static void main(String args[])
5.     {
6.         JFrame f= new JFrame("Label Example");
7.         JLabel l1,l2;
8.         l1=new JLabel("First Label.");
9.         l1.setBounds(50,50, 100,30);
10.        l2=new JLabel("Second Label.");

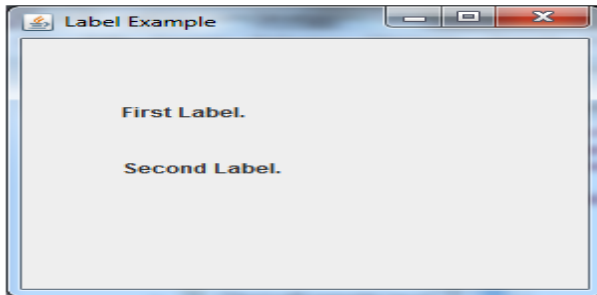
```

```

11. l2.setBounds(50,100, 100,30);
12. f.add(l1); f.add(l2);
13. f.setSize(300,300);
14. f.setLayout(null);
15. f.setVisible(true);
16. }
17. }

```

Output:



## Java JLabel Example with ActionListener

```

1. import javax.swing.*;
2. import java.awt.*;
3. import java.awt.event.*;
4. public class LabelExample extends Frame implements ActionListener{
5.     JTextField tf; JLabel l; JButton b;
6.     LabelExample(){
7.         tf=new JTextField();
8.         tf.setBounds(50,50, 150,20);
9.         l=new JLabel();
10.        l.setBounds(50,100, 250,20);
11.        b=new JButton("Find IP");
12.        b.setBounds(50,150,95,30);
13.        b.addActionListener(this);
14.        add(b);add(tf);add(l);
15.        setSize(400,400);
16.        setLayout(null);
17.        setVisible(true);
18.    }
19.    public void actionPerformed(ActionEvent e) {
20.        try{
21.            String host=tf.getText();
22.            String ip=java.net.InetAddress.getByName(host).getHostAddress();
23.            l.setText("IP of "+host+" is: "+ip);
24.        }catch(Exception ex){System.out.println(ex);}

```

```

25. }
26. public static void main(String[] args) {
27.     new LabelExample();
28. } }

```

Output:



## Java JTextField

The object of a JTextField class is a text component that allows the editing of a single line text. It inherits JTextComponent class.

### JTextField class declaration

Let's see the declaration for javax.swing.JTextField class.

1. **public class** JTextField **extends** JTextComponent **implements** SwingConstants

### Commonly used Constructors:

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.



Methods	Description
---------	-------------

<code>JTextField(String text, int columns)</code>	Creates a new TextField initialized with the specified text and columns.
<code>JTextField(int columns)</code>	Creates a new empty TextField with the specified number of columns.

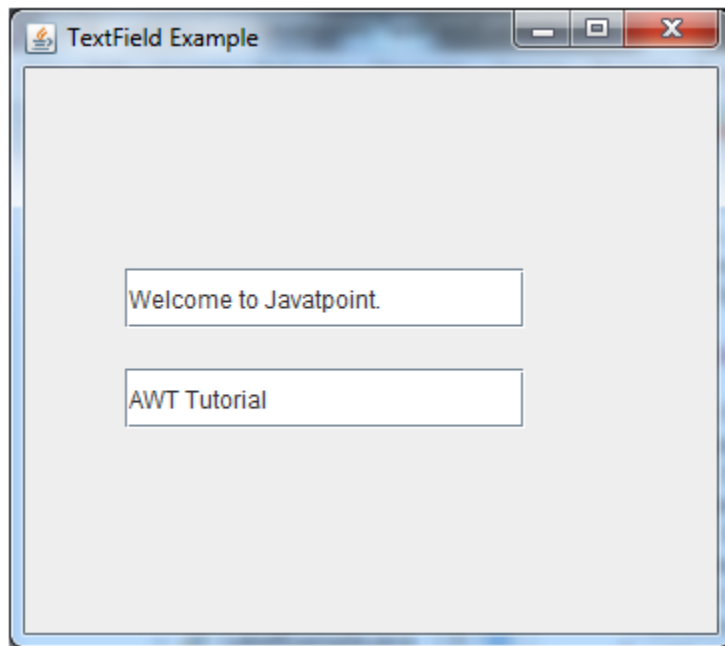
void addActionListener(ActionListener l)	It is used to add the specified action listener to receive action events from this textfield.
Action getAction()	It returns the currently set Action for this ActionEvent source, or null if no Action is set.
void setFont(Font f)	It is used to set the current font.
void removeActionListener(ActionListener l)	It is used to remove the specified action listener so that it no longer receives action events from this textfield.

## Java JTextFiel d Example

```
import javax.swing.*;
2. class T
   JTextFieldExamp
   e
   {
```

```
5. public static void main(String args[])
6. {
7.     JFrame f= new JFrame("TextField Example");
8.     JTextField t1,t2;
9.     t1=new JTextField("Welcome to Javatpoint.");
10.    t1.setBounds(50,100, 200,30);
11.    t2=new JTextField("AWT Tutorial");
12.    t2.setBounds(50,150, 200,30);
13.    f.add(t1); f.add(t2);
14.    f.setSize(400,400);
15.    f.setLayout(null);
16.    f.setVisible(true);
17. }
18. }
```

Output:



---

## Java JTextField Example with ActionListener

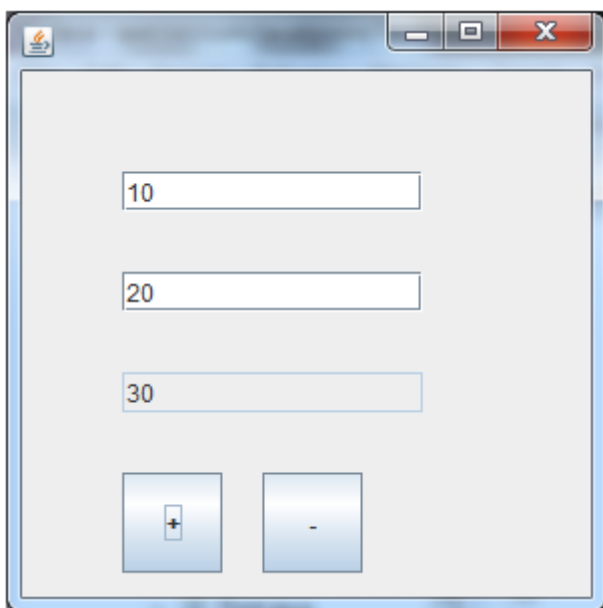
```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class TextFieldExample implements ActionListener{
4.     JTextField tf1,tf2,tf3;
5.     JButton b1,b2;
6.     TextFieldExample(){
7.         JFrame f= new JFrame();
8.         tf1=new JTextField();
9.         tf1.setBounds(50,50,150,20);
10.        tf2=new JTextField();
11.        tf2.setBounds(50,100,150,20);
12.        tf3=new JTextField();
13.        tf3.setBounds(50,150,150,20);
14.        tf3.setEditable(false);
15.        b1=new JButton("+");
16.        b1.setBounds(50,200,50,50);
17.        b2=new JButton("-");
18.        b2.setBounds(120,200,50,50);
19.        b1.addActionListener(this);
20.        b2.addActionListener(this);
21.        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
22.        f.setSize(300,300);
23.        f.setLayout(null);
```

```

24.      f.setVisible(true);
25.  }
26.  public void actionPerformed(ActionEvent e) {
27.      String s1=tf1.getText();
28.      String s2=tf2.getText();
29.      int a=Integer.parseInt(s1);
30.      int b=Integer.parseInt(s2);
31.      int c=0;
32.      if(e.getSource()==b1){
33.          c=a+b;
34.      }else if(e.getSource()==b2){
35.          c=a-b;
36.      }
37.      String result=String.valueOf(c);
38.      tf3.setText(result);
39.  }
40. public static void main(String[] args) {
41.     new TextFieldExample();
42. } }

```

Output:



## Java JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

### JTextArea class declaration

Let's see the declaration for javax.swing.JTextArea class.

1. **public class** JTextArea **extends** JTextComponent

### Commonly used Constructors:

Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

### Commonly used Methods:

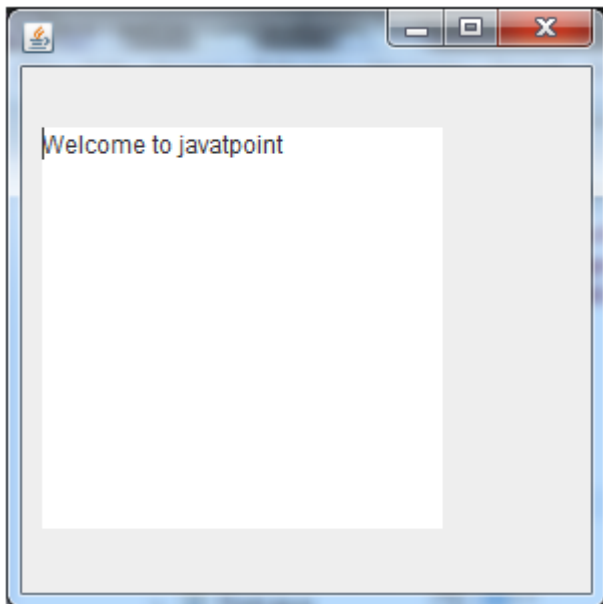
Methods	Description
void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

---

## Java JTextArea Example

```
1. import javax.swing.*;
2. public class TextAreaExample
3. {
4.     TextAreaExample(){
5.         JFrame f= new JFrame();
6.         JTextArea area=new JTextArea("Welcome to javatpoint");
7.         area.setBounds(10,30, 200,200);
8.         f.add(area);
9.         f.setSize(300,300);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13. public static void main(String args[])
14. {
15.     new TextAreaExample();
16. }}
```

Output:



---

## Java JTextArea Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
```

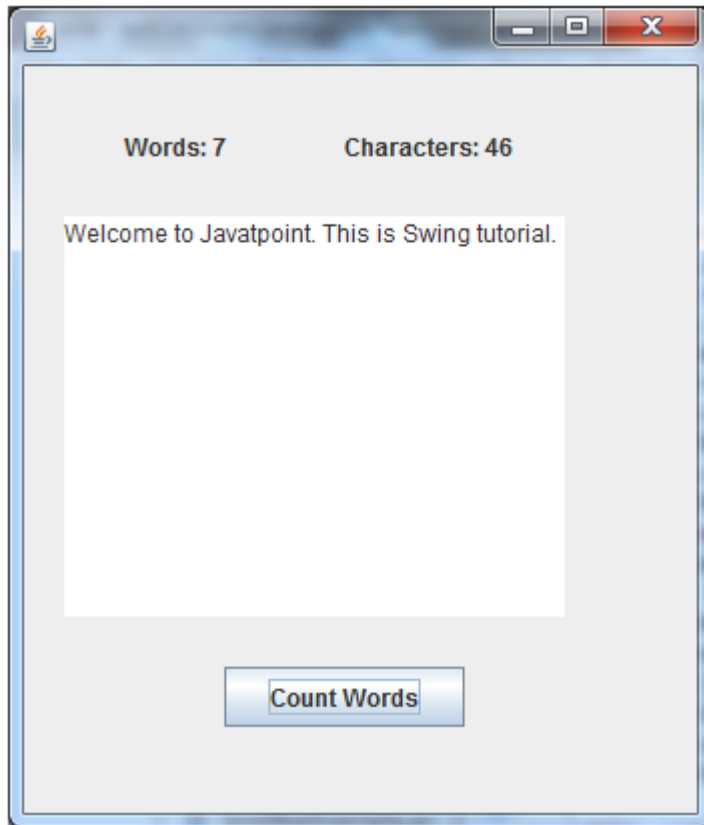
```

3. public class TextAreaExample implements ActionListener{
4.     JLabel l1,l2;
5.     JTextArea area;
6.     JButton b;
7.     TextAreaExample() {
8.         JFrame f= new JFrame();
9.         l1=new JLabel();
10.        l1.setBounds(50,25,100,30);
11.        l2=new JLabel();
12.        l2.setBounds(160,25,100,30);
13.        area=new JTextArea();
14.        area.setBounds(20,75,250,200);
15.        b=new JButton("Count Words");
16.        b.setBounds(100,300,120,30);
17.        b.addActionListener(this);
18.        f.add(l1);f.add(l2);f.add(area);f.add(b);
19.        f.setSize(450,450);
20.        f.setLayout(null);
21.        f.setVisible(true);
22.    }
23. public void actionPerformed(ActionEvent e){
24.     String text=area.getText();
25.     String words[]=text.split("\\s");
26.     l1.setText("Words: "+words.length);
27.     l2.setText("Characters: "+text.length());
28. }
29. public static void main(String[] args) {
30.     new TextAreaExample();
31. }
32. }

```

Output:

Output:



## Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

---

### JPasswordField class declaration

Let's see the declaration for javax.swing.JPasswordField class.

1. **public class** JPasswordField **extends** JTextField



## Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

## Java JPasswordField Example

```
1. import javax.swing.*;
2. public class PasswordFieldExample {
3.     public static void main(String[] args) {
4.         JFrame f=new JFrame("Password Field Example");
5.         JPasswordField value = new JPasswordField();
6.         JLabel l1=new JLabel("Password:");
7.         l1.setBounds(20,100, 80,30);
8.         value.setBounds(100,100,100,30);
9.         f.add(value); f.add(l1);
10.        f.setSize(300,300);
11.        f.setLayout(null);
12.        f.setVisible(true);
13.    }
14. }
```

Output:



---

## Java JPasswordField Example with ActionListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class PasswordFieldExample {
4.     public static void main(String[] args) {
5.         JFrame f=new JFrame("Password Field Example");
6.         final JLabel label = new JLabel();
7.         label.setBounds(20,150, 200,50);
8.         final JPasswordField value = new JPasswordField();
9.         value.setBounds(100,75,100,30);
10.        JLabel l1=new JLabel("Username:");
11.        l1.setBounds(20,20, 80,30);
12.        JLabel l2=new JLabel("Password:");
13.        l2.setBounds(20,75, 80,30);
14.        JButton b = new JButton("Login");
15.        b.setBounds(100,120, 80,30);
16.        final JTextField text = new JTextField();
17.        text.setBounds(100,20, 100,30);
18.        f.add(value); f.add(l1); f.add(label); f.add(l2); f.add(b); f.add(text);
19.        f.setSize(300,300);
20.        f.setLayout(null);
21.        f.setVisible(true);
22.        b.addActionListener(new ActionListener() {
23.            public void actionPerformed(ActionEvent e) {
24.                String data = "Username " + text.getText();
```

```

25.         data += ", Password: "
26.         + new String(value.getPassword());
27.         label.setText(data);
28.     }
29. });
30. }
31. }

```

Output:



## Java JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits [JToggleButton](#) class.

## JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

1. **public class** JCheckBox **extends** JToggleButton **implements** Accessible

## Commonly used Constructors:

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button

	with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

### Commonly used Methods:

Methods	Description
AccessibleContext getAccessibleContext()	It is used to get the AccessibleContext associated with this JCheckBox.
protected String paramString()	It returns a <a href="#">string</a> representation of this JCheckBox.

## Java JCheckBox Example

```

1. import javax.swing.*;
2. public class CheckBoxExample
3. {
4.     CheckBoxExample(){
5.         JFrame f= new JFrame("CheckBox Example");
6.         JCheckBox checkBox1 = new JCheckBox("C++");
7.         checkBox1.setBounds(100,100, 50,50);
8.         JCheckBox checkBox2 = new JCheckBox("Java", true);
9.         checkBox2.setBounds(100,150, 50,50);
10.        f.add(checkBox1);
11.        f.add(checkBox2);
12.        f.setSize(400,400);
13.        f.setLayout(null);
14.        f.setVisible(true);

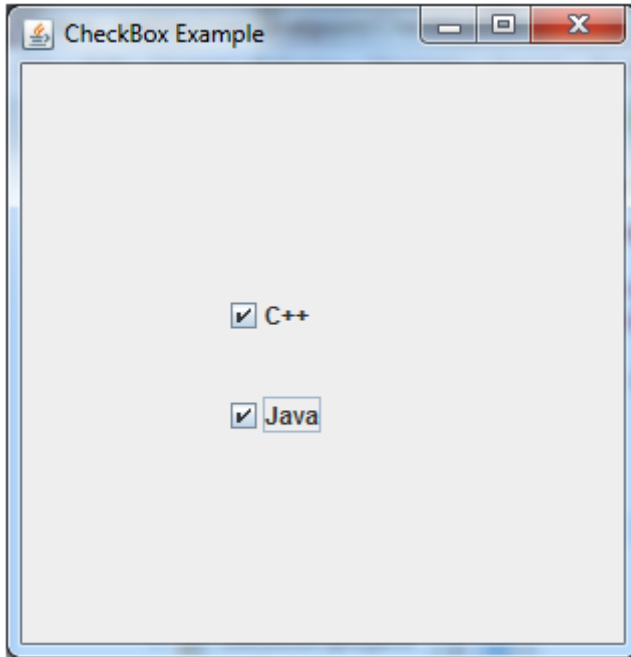
```

```

15. }
16. public static void main(String args[])
17. {
18.     new CheckBoxExample();
19. }

```

Output:




---

## Java JCheckBox Example with ItemListener

```

1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample
4. {
5.     CheckBoxExample(){
6.         JFrame f= new JFrame("CheckBox Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        JCheckBox checkbox1 = new JCheckBox("C++");
11.        checkbox1.setBounds(150,100, 50,50);
12.        JCheckBox checkbox2 = new JCheckBox("Java");
13.        checkbox2.setBounds(150,150, 50,50);
14.        f.add(checkbox1); f.add(checkbox2); f.add(label);
15.        checkbox1.addItemListener(new ItemListener() {
16.            public void itemStateChanged(ItemEvent e) {

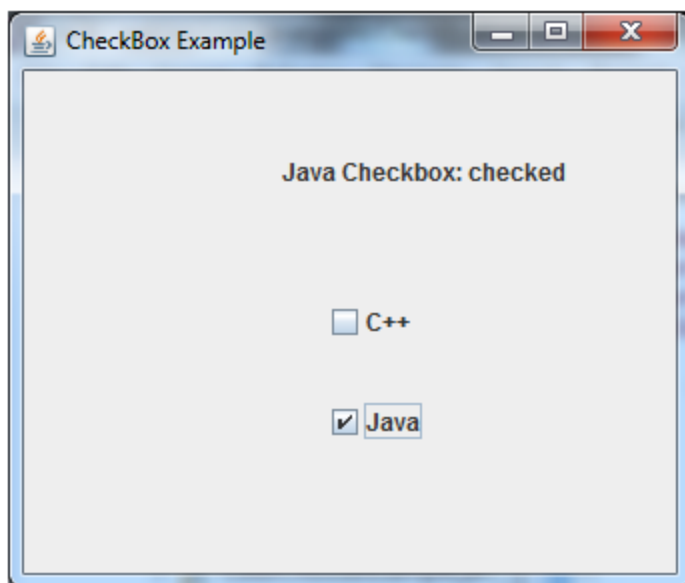
```

```

17.         label.setText("C++ Checkbox: "
18.         + (e.getStateChange()==1?"checked":"unchecked"));
19.     }
20. });
21. checkbox2.addItemListener(new ItemListener() {
22.     public void itemStateChanged(ItemEvent e) {
23.         label.setText("Java Checkbox: "
24.         + (e.getStateChange()==1?"checked":"unchecked"));
25.     }
26. });
27. f.setSize(400,400);
28. f.setLayout(null);
29. f.setVisible(true);
30. }
31. public static void main(String args[])
32. {
33.     new CheckBoxExample();
34. }
35. }

```

Output:




---

## Java JCheckBox Example: Food Order

```

1. import javax.swing.*;
2. import java.awt.event.*;
3. public class CheckBoxExample extends JFrame implements ActionListener{

```

```

4.   JLabel l;
5.   JCheckBox cb1,cb2,cb3;
6.   JButton b;
7.   CheckBoxExample(){
8.       l=new JLabel("Food Ordering System");
9.       l.setBounds(50,50,300,20);
10.      cb1=new JCheckBox("Pizza @ 100");
11.      cb1.setBounds(100,100,150,20);
12.      cb2=new JCheckBox("Burger @ 30");
13.      cb2.setBounds(100,150,150,20);
14.      cb3=new JCheckBox("Tea @ 10");
15.      cb3.setBounds(100,200,150,20);
16.      b=new JButton("Order");
17.      b.setBounds(100,250,80,30);
18.      b.addActionListener(this);
19.      add(l);add(cb1);add(cb2);add(cb3);add(b);
20.      setSize(400,400);
21.      setLayout(null);
22.      setVisible(true);
23.      setDefaultCloseOperation(EXIT_ON_CLOSE);
24.  }
25.  public void actionPerformed(ActionEvent e){
26.      float amount=0;
27.      String msg="";
28.      if(cb1.isSelected()){
29.          amount+=100;
30.          msg="Pizza: 100\n";
31.      }
32.      if(cb2.isSelected()){
33.          amount+=30;
34.          msg+="Burger: 30\n";
35.      }
36.      if(cb3.isSelected()){
37.          amount+=10;
38.          msg+="Tea: 10\n";
39.      }
40.      msg+="-----\n";
41.      JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
42.  }
43.  public static void main(String[] args) {
44.      new CheckBoxExample();
45.  }
46. }

```

Output:



## Java JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

## JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public class** JRadioButton **extends** JToggleButton **implements** Accessible

## Commonly used Constructors:



Constructor	Description
JRadioButton()	Creates an unselected radio button with no text.
JRadioButton(String s)	Creates an unselected radio button with specified text.
JRadioButton(String s, boolean selected)	Creates a radio button with the specified text and selected status.

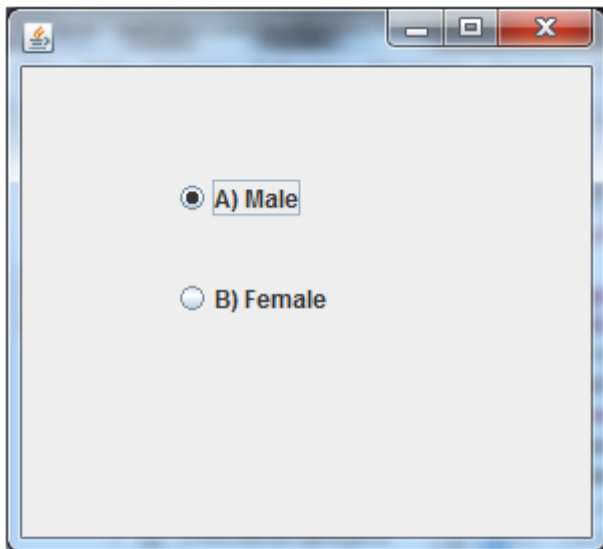
## Commonly used Methods:

Methods	Description
void setText(String s)	It is used to set specified text on button.
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

## Java JRadioButton Example

```
1. import javax.swing.*;
2. public class RadioButtonExample {
3.     JFrame f;
4.     RadioButtonExample(){
5.         f=new JFrame();
6.         JRadioButton r1=new JRadioButton("A) Male");
7.         JRadioButton r2=new JRadioButton("B) Female");
8.         r1.setBounds(75,50,100,30);
9.         r2.setBounds(75,100,100,30);
10.    ButtonGroup bg=new ButtonGroup();
11.    bg.add(r1);bg.add(r2);
12.    f.add(r1);f.add(r2);
13.    f.setSize(300,300);
14.    f.setLayout(null);
15.    f.setVisible(true);
16. }
17. public static void main(String[] args) {
18.     new RadioButtonExample();
19. }
20. }
```

Output:



---

## Java JRadioButton Example with ActionListener

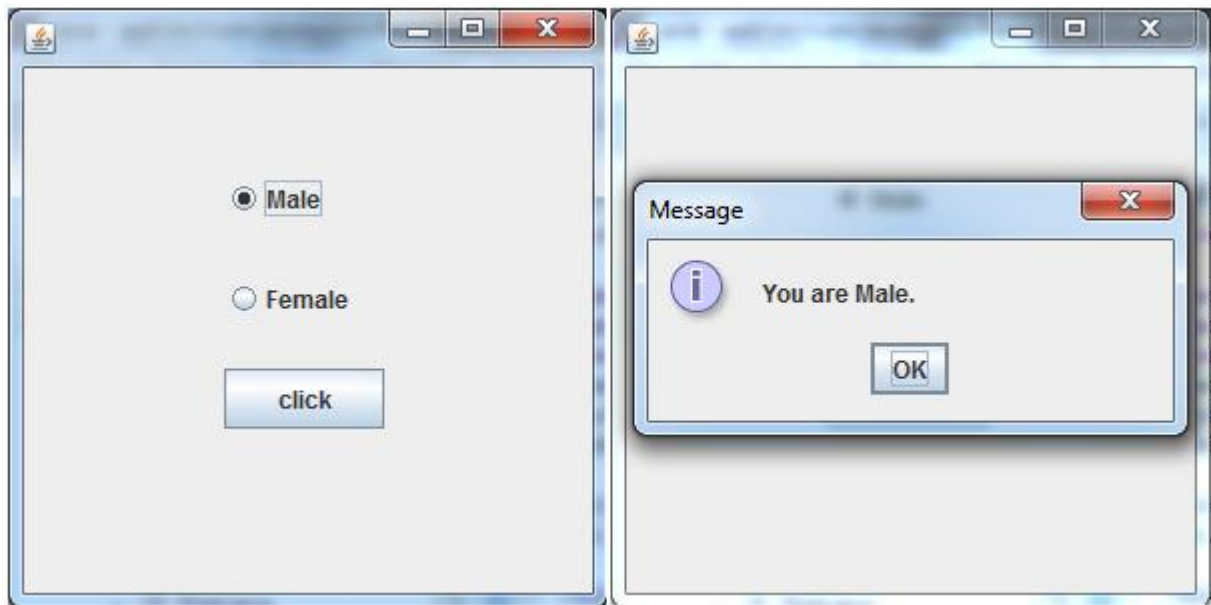
```
1. import javax.swing.*;
```

```

2. import java.awt.event.*;
3. class RadioButtonExample extends JFrame implements ActionListener{
4.   JRadioButton rb1,rb2;
5.   JButton b;
6.   RadioButtonExample(){
7.     rb1=new JRadioButton("Male");
8.     rb1.setBounds(100,50,100,30);
9.     rb2=new JRadioButton("Female");
10.    rb2.setBounds(100,100,100,30);
11.    ButtonGroup bg=new ButtonGroup();
12.    bg.add(rb1);bg.add(rb2);
13.    b=new JButton("click");
14.    b.setBounds(100,150,80,30);
15.    b.addActionListener(this);
16.    add(rb1);add(rb2);add(b);
17.    setSize(300,300);
18.    setLayout(null);
19.    setVisible(true);
20. }
21. public void actionPerformed(ActionEvent e){
22.   if(rb1.isSelected()){
23.     JOptionPane.showMessageDialog(this, "You are Male.");
24.   }
25.   if(rb2.isSelected()){
26.     JOptionPane.showMessageDialog(this, "You are Female.");
27.   }
28. }
29. public static void main(String args[]){
30.   new RadioButtonExample();
31. }}

```

Output:



## Java JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a [menu](#). It inherits [JComponent](#) class.

## JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

1. **public class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

## Commonly used Constructors:

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified <a href="#">array</a> .
JComboBox(Vector<?> items)	Creates a JComboBox that contains the elements in the specified <a href="#">Vector</a> .

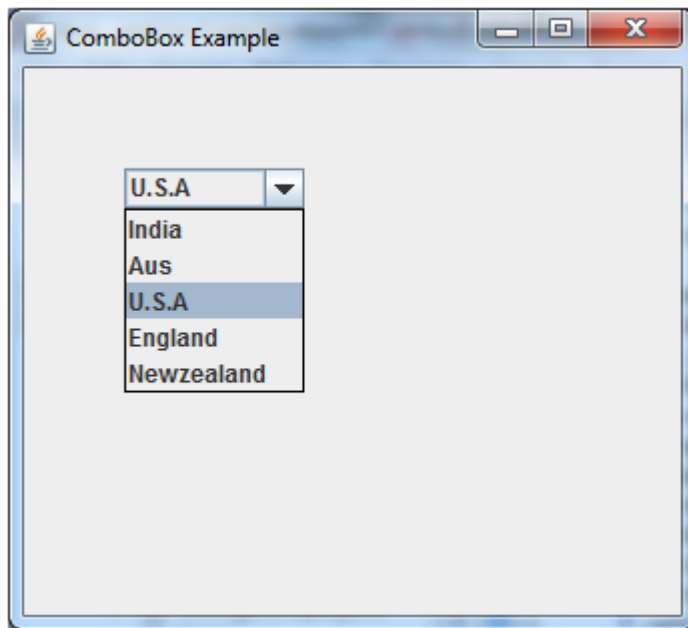
## Commonly used Methods:

Methods	Description
<code>void addItem(Object anObject)</code>	It is used to add an item to the item list.
<code>void removeItem(Object anObject)</code>	It is used to delete an item to the item list.
<code>void removeAllItems()</code>	It is used to remove all the items from the list.
<code>void setEditable(boolean b)</code>	It is used to determine whether the JComboBox is editable.
<code>void addActionListener(ActionListener a)</code>	It is used to add the <a href="#">ActionListener</a> .
<code>void addItemListener(ItemListener i)</code>	It is used to add the <a href="#">ItemListener</a> .

## Java JComboBox Example

```
1. import javax.swing.*;
2. public class ComboBoxExample {
3.     JFrame f;
4.     ComboBoxExample(){
5.         f=new JFrame("ComboBox Example");
6.         String country[]={"India","Aus","U.S.A","England","Newzealand"};
7.         JComboBox cb=new JComboBox(country);
8.         cb.setBounds(50, 50,90,20);
9.         f.add(cb);
10.        f.setLayout(null);
11.        f.setSize(400,500);
12.        f.setVisible(true);
13.    }
14. public static void main(String[] args) {
15.     new ComboBoxExample();
16. }
17. }
```

Output:



---

## Java JComboBox Example with ActionListener

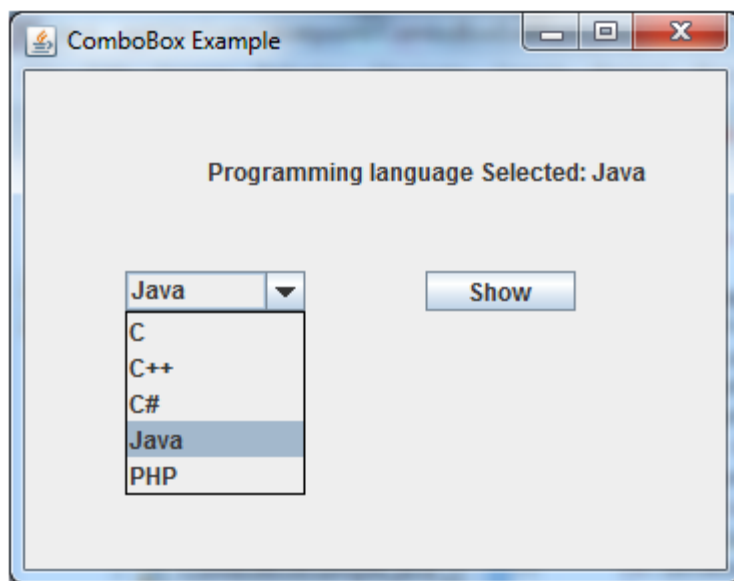
```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class ComboBoxExample {
4.     JFrame f;
5.     ComboBoxExample(){
6.         f=new JFrame("ComboBox Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        JButton b=new JButton("Show");
11.        b.setBounds(200,100,75,20);
12.        String languages[]={"C","C++","C#","Java","PHP"};
13.        final JComboBox cb=new JComboBox(languages);
14.        cb.setBounds(50, 100,90,20);
15.        f.add(cb); f.add(label); f.add(b);
16.        f.setLayout(null);
17.        f.setSize(350,350);
18.        f.setVisible(true);
19.        b.addActionListener(new ActionListener() {
20.            public void actionPerformed(ActionEvent e) {
21.                String data = "Programming language Selected: "
22.                + cb.getItemAt(cb.getSelectedIndex());
```

```

23. label.setText(data);
24. }
25. });
26. }
27. public static void main(String[] args) {
28.     new ComboBoxExample();
29. }
30. }

```

Output:



## Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

## JTable class declaration

Let's see the declaration for javax.swing.JTable class.

## Commonly used Constructors:

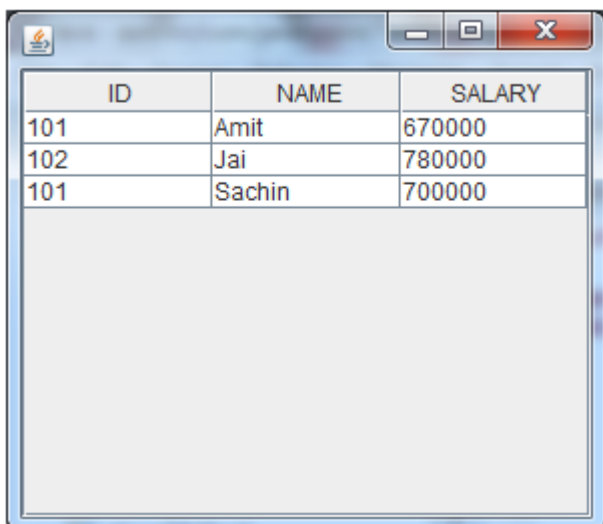
Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

---

## Java JTable Example

```
1. import javax.swing.*;
2. public class TableExample {
3.     JFrame f;
4.     TableExample(){
5.         f=new JFrame();
6.         String data[][]={ {"101","Amit","670000"},
7.                             {"102","Jai","780000"},
8.                             {"101","Sachin","700000"}};
9.         String column[]={ "ID", "NAME", "SALARY"};
10.        JTable jt=new JTable(data,column);
11.        jt.setBounds(30,40,200,300);
12.        JScrollPane sp=new JScrollPane(jt);
13.        f.add(sp);
14.        f.setSize(300,400);
15.        f.setVisible(true);
16.    }
17.    public static void main(String[] args) {
18.        new TableExample();
19.    }
20. }
```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

---

## Java JTable Example with ListSelectionListener

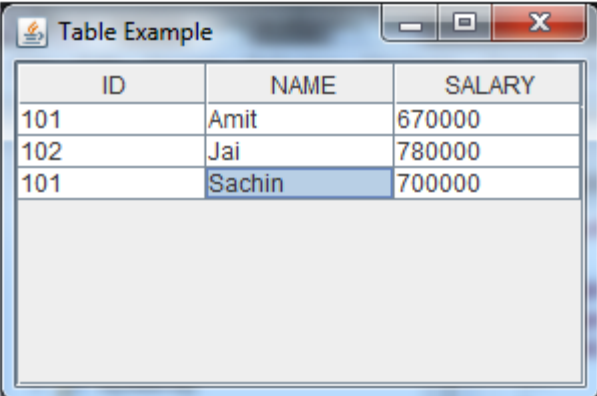


```

1. import javax.swing.*;
2. import javax.swing.event.*;
3. public class TableExample {
4.     public static void main(String[] a) {
5.         JFrame f = new JFrame("Table Example");
6.         String data[][]={ {"101","Amit","670000"},
7.                             {"102","Jai","780000"},
8.                             {"101","Sachin","700000"}};
9.         String column[]={ "ID","NAME","SALARY"};
10.        final JTable jt=new JTable(data,column);
11.        jt.setCellSelectionEnabled(true);
12.        ListSelectionModel select= jt.getSelectionModel();
13.        select.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
14.        select.addListSelectionListener(new ListSelectionListener() {
15.            public void valueChanged(ListSelectionEvent e) {
16.                String Data = null;
17.                int[] row = jt.getSelectedRows();
18.                int[] columns = jt.getSelectedColumns();
19.                for (int i = 0; i < row.length; i++) {
20.                    for (int j = 0; j < columns.length; j++) {
21.                        Data = (String) jt.getValueAt(row[i], columns[j]);
22.                    } }
23.                System.out.println("Table element selected is: " + Data);
24.            }
25.        });
26.        JScrollPane sp=new JScrollPane(jt);
27.        f.add(sp);
28.        f.setSize(300, 200);
29.        f.setVisible(true);
30.    }
31.}

```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

If you select an element in column **NAME**, name of the element will be displayed on the console:

1. Table element selected is: Sachin

## Java JList

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items. It inherits JComponent class.

### JList class declaration

Let's see the declaration for javax.swing.JList class.

1. **public class** JList **extends** JComponent **implements** Scrollable, Accessible

### Commonly used Constructors:

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.
JList(ListModel<ary> dataModel)	Creates a JList that displays elements from the specified, non-null, model.

### Commonly used Methods:

Methods	Description
Void addListSelectionListener(ListSelectionListener listener)	It is used to add a listener to the list, to be notified each time a change to the selection occurs.

<code>int getSelectedIndex()</code>	It is used to return the smallest selected cell index.
<code>ListModel getModel()</code>	It is used to return the data model that holds a list of items displayed by the JList component.
<code>void setListData(Object[] listData)</code>	It is used to create a read-only ListModel from an array of objects.

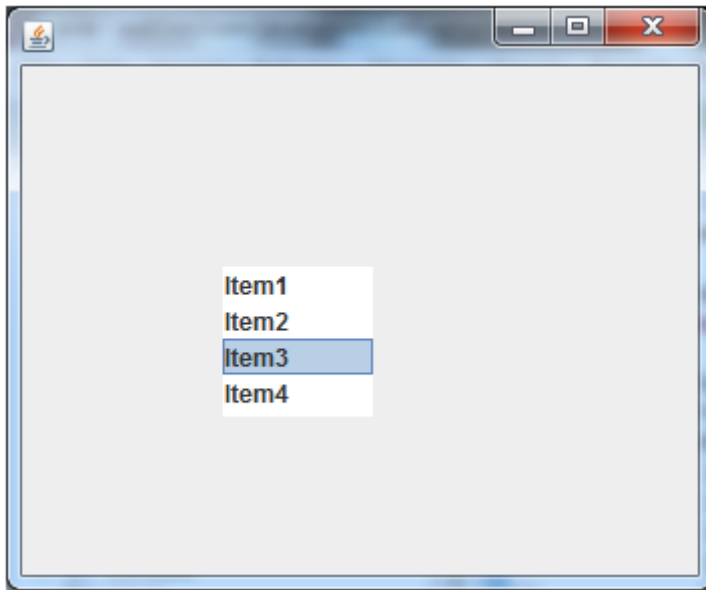
## Java JList Example

```

1. import javax.swing.*;
2. public class ListExample
3. {
4.     ListExample(){
5.         JFrame f= new JFrame();
6.         DefaultListModel<String> l1 = new DefaultListModel<>();
7.         l1.addElement("Item1");
8.         l1.addElement("Item2");
9.         l1.addElement("Item3");
10.        l1.addElement("Item4");
11.        JList<String> list = new JList<>(l1);
12.        list.setBounds(100,100, 75,75);
13.        f.add(list);
14.        f.setSize(400,400);
15.        f.setLayout(null);
16.        f.setVisible(true);
17.    }
18. public static void main(String args[])
19. {
20.     new ListExample();
21. }

```

Output:



---

## Java JList Example with ActionListener

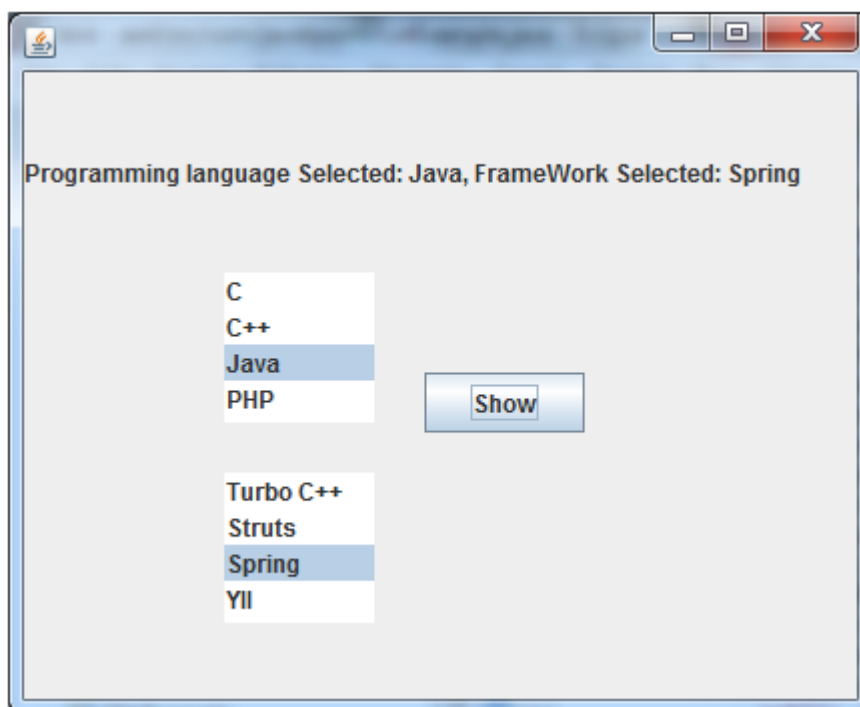
```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class ListExample
4. {
5.     ListExample(){
6.         JFrame f= new JFrame();
7.         final JLabel label = new JLabel();
8.         label.setSize(500,100);
9.         JButton b=new JButton("Show");
10.        b.setBounds(200,150,80,30);
11.        final DefaultListModel<String> l1 = new DefaultListModel<>();
12.        l1.addElement("C");
13.        l1.addElement("C++");
14.        l1.addElement("Java");
15.        l1.addElement("PHP");
16.        final JList<String> list1 = new JList<>(l1);
17.        list1.setBounds(100,100, 75,75);
18.        DefaultListModel<String> l2 = new DefaultListModel<>();
19.        l2.addElement("Turbo C++");
20.        l2.addElement("Struts");
21.        l2.addElement("Spring");
22.        l2.addElement("YII");
23.        final JList<String> list2 = new JList<>(l2);
24.        list2.setBounds(100,200, 75,75);
25.        f.add(list1); f.add(list2); f.add(b); f.add(label);
```

```

26.      f.setSize(450,450);
27.      f.setLayout(null);
28.      f.setVisible(true);
29.      b.addActionListener(new ActionListener() {
30.          public void actionPerformed(ActionEvent e) {
31.              String data = "";
32.              if (list1.getSelectedIndex() != -1) {
33.                  data = "Programming language Selected: " + list1.getSelectedValue();
34.                  label.setText(data);
35.              }
36.              if(list2.getSelectedIndex() != -1){
37.                  data += ", FrameWork Selected: ";
38.                  for(Object frame :list2.getSelectedValues()){
39.                      data += frame + " ";
40.                  }
41.              }
42.              label.setText(data);
43.          }
44.      });
45.  }
46. public static void main(String args[])
47.  {
48.      new ListExample();
49.  }}

```

Output:



# Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

## JOptionPane class declaration

1. **public class** JOptionPane **extends** JComponent **implements** Accessible

## Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType)	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

## Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title)	It is used to create a message dialog with given title and

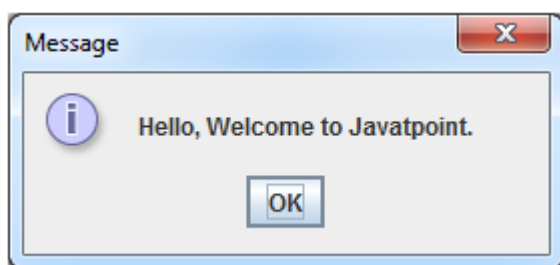
title, int messageType)	messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

## Java JOptionPane Example: showMessageDialog()

```

1. import javax.swing.*;
2. public class OptionPaneExample {
3.     JFrame f;
4.     OptionPaneExample(){
5.         f=new JFrame();
6.         JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
7.     }
8.     public static void main(String[] args) {
9.         new OptionPaneExample();
10.    }
11. }
```

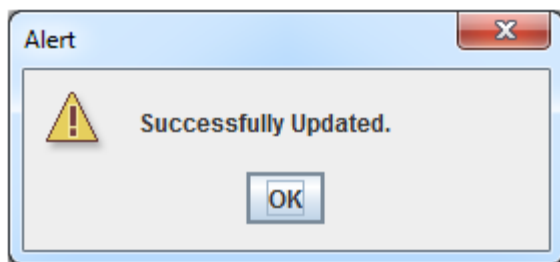
Output:



## Java JOptionPane Example: showMessageDialog()

```
1. import javax.swing.*;
2. public class OptionPaneExample {
3.     JFrame f;
4.     OptionPaneExample(){
5.         f=new JFrame();
6.         JOptionPane.showMessageDialog(f,"Successfully Updated.","Alert",JOptionPane.WARN
7.             ING MESSAGE);
8.     }
9.     public static void main(String[] args) {
10.         new OptionPaneExample();
11.     }
```

Output:



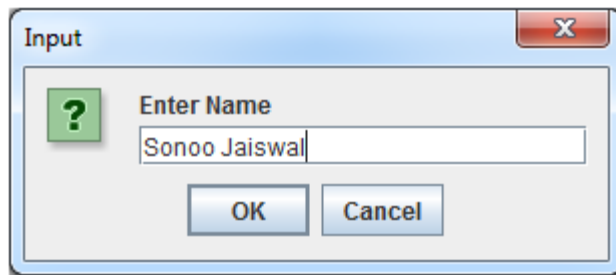
---

## Java JOptionPane Example: showInputDialog()

```
1. import javax.swing.*;
2. public class OptionPaneExample {
3.     JFrame f;
4.     OptionPaneExample(){
5.         f=new JFrame();
6.         String name=JOptionPane.showInputDialog(f,"Enter Name");
7.     }
8.     public static void main(String[] args) {
9.         new OptionPaneExample();
10.     }
11. }
```

Output:



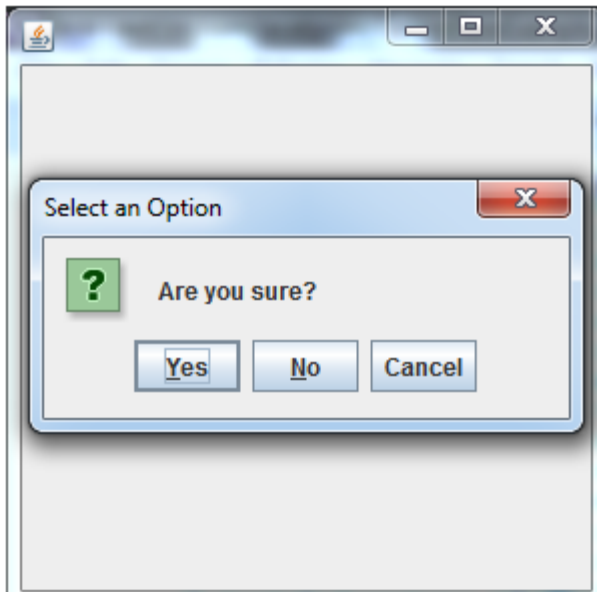


---

## Java JOptionPane Example: showConfirmDialog()

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. public class OptionPaneExample extends WindowAdapter{
4.     JFrame f;
5.     OptionPaneExample(){
6.         f=new JFrame();
7.         f.addWindowListener(this);
8.         f.setSize(300, 300);
9.         f.setLayout(null);
10.        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
11.        f.setVisible(true);
12.    }
13. public void windowClosing(WindowEvent e) {
14.     int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
15. if(a==JOptionPane.YES_OPTION){
16.         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17.     }
18. }
19. public static void main(String[] args) {
20.     new OptionPaneExample();
21. }
22. }
```

Output:



## Java JScrollBar

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

### JScrollBar class declaration

Let's see the declaration for javax.swing.JScrollBar class.

1. **public class** JScrollBar **extends** JComponent **implements** Adjustable, Accessible

### Commonly used Constructors:

Constructor	Description
JScrollBar()	Creates a vertical scrollbar with the initial values.
JScrollBar(int orientation)	Creates a scrollbar with the specified orientation and the initial values.
JScrollBar(int orientation, int value, int extent, int min, int max)	Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum.

## Java JScrollBar Example

```
1. import javax.swing.*;
2. class ScrollBarExample
3. {
4.     ScrollBarExample(){
5.         JFrame f= new JFrame("Scrollbar Example");
6.         JScrollBar s=new JScrollBar();
7.         s.setBounds(100,100, 50,100);
8.         f.add(s);
9.         f.setSize(400,400);
10.        f.setLayout(null);
11.        f.setVisible(true);
12.    }
13. public static void main(String args[])
14. {
15.     new ScrollBarExample();
16. }}
```

Output:

---

## Java JScrollBar Example with AdjustmentListener

```
1. import javax.swing.*;
2. import java.awt.event.*;
3. class ScrollBarExample
4. {
5.     ScrollBarExample(){
6.         JFrame f= new JFrame("Scrollbar Example");
7.         final JLabel label = new JLabel();
8.         label.setHorizontalAlignment(JLabel.CENTER);
9.         label.setSize(400,100);
10.        final JScrollBar s=new JScrollBar();
11.        s.setBounds(100,100, 50,100);
12.        f.add(s); f.add(label);
13.        f.setSize(400,400);
14.        f.setLayout(null);
15.        f.setVisible(true);
16.        s.addAdjustmentListener(new AdjustmentListener() {
17.            public void adjustmentValueChanged(AdjustmentEvent e) {
```

```

18. label.setText("Vertical Scrollbar value is:"+ s.getValue());
19. }
20. });
21. }
22. public static void main(String args[])
23. {
24. new ScrollBarExample();
25. }

```

Output:

## Swings

**Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

## Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

N o.	Java AWT	Java Swing
1 )	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2 )	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3 )	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .

4 )	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5 )	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

## What is JFC

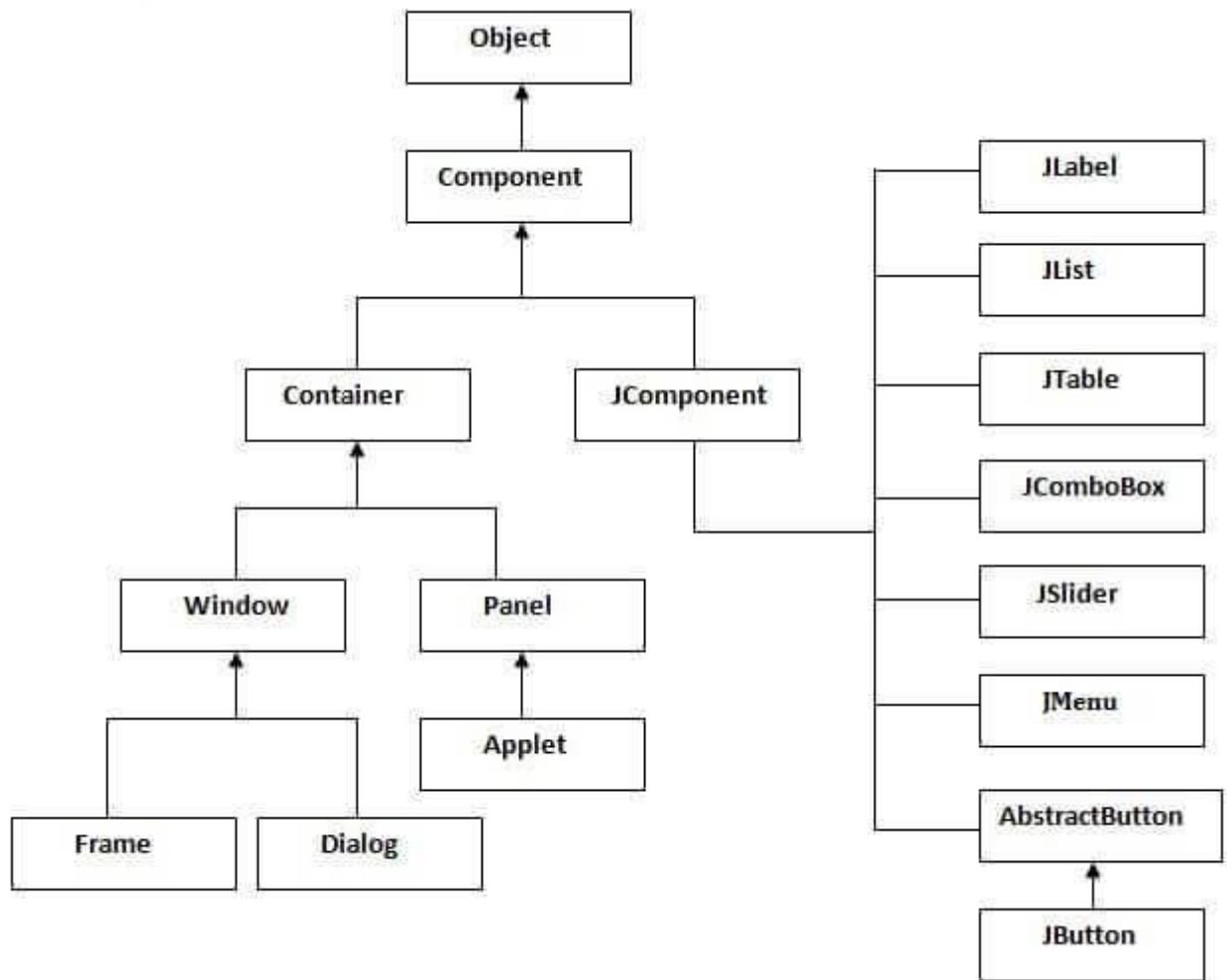
The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

### Do You Know

- [How to create runnable jar file in java?](#)
- [How to display image on a button in swing?](#)
- [How to change the component color by choosing a color from ColorChooser ?](#)
- [How to display the digital watch in swing tutorial ?](#)
- [How to create a notepad in swing?](#)
- [How to create puzzle game and pic puzzle game in swing ?](#)
- [How to create tic tac toe game in swing ?](#)

## Hierarchy of Java Swing classes

[The hierarchy of java swing API is given below.](#)



## Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.

`public void setVisible(boolean b)`

sets the visibility of the component. It is by default false.

## Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

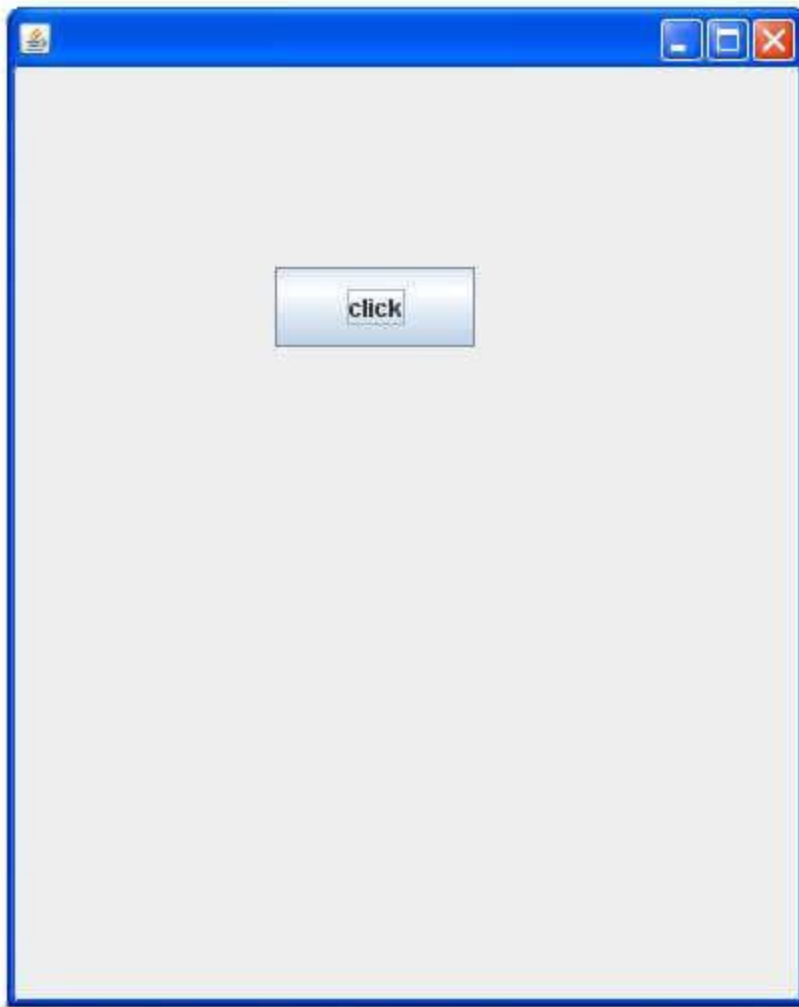
---

## Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4.   JFrame f=new JFrame();//creating instance of JFrame
5.
6.   JButton b=new JButton("click");//creating instance of JButton
7.   b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9.   f.add(b);//adding button in JFrame
10.
11. f.setSize(400,500);//400 width and 500 height
12. f.setLayout(null);//using no layout managers
13. f.setVisible(true);//making the frame visible
14. }
15. }
```



---

## Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

*File: Simple.java*

```
1. import javax.swing.*;
2. public class Simple {
3.     JFrame f;
4.     Simple(){
5.         f=new JFrame();//creating instance of JFrame
6.         _____
7.         JButton b=new JButton("click");//creating instance of JButton
8.         b.setBounds(130,100,100, 40);
9.         _____
10.        f.add(b);//adding button in JFrame
```



```

11. _____
12. f.setSize(400,500); //400 width and 500 height
13. f.setLayout(null); //using no layout managers
14. f.setVisible(true); //making the frame visible
15. }
16. _____
17. public static void main(String[] args) {
18. new Simple();
19. }
20. }

```

The `setBounds(int xaxis, int yaxis, int width, int height)` is used in the above example that sets the position of the button.

---

## Simple example of Swing by inheritance

We can also inherit the `JFrame` class, so there is no need to create the instance of `JFrame` class explicitly.

*File: Simple2.java*

```

1. import javax.swing.*;
2. public class Simple2 extends JFrame { //inheriting JFrame
3.     JFrame f;
4.     Simple2() {
5.         JButton b = new JButton("click"); //create button
6.         b.setBounds(130, 100, 100, 40);
7.         _____
8.         add(b); //adding button on frame
9.         setSize(400, 500);
10.        setLayout(null);
11.        setVisible(true);
12.    }
13. public static void main(String[] args) {
14. new Simple2();
15. }}

```

[download this example](#)

### What we will learn in Swing Tutorial

- JButton class
- JRadioButton class
- JTextArea class

- [JComboBox class](#)
- [JTable class](#)
- [JColorChooser class](#)
- [JProgressBar class](#)
- [JSlider class](#)
- [Digital Watch](#)
- [Graphics in swing](#)
- [Displaying image](#)
- [Edit menu code for Notepad](#)
- [OpenDialog Box](#)
- [Notepad](#)
- [Puzzle Game](#)
- [Pic Puzzle Game](#)
- [Tic Tac Toe Game](#)
- [BorderLayout](#)
- [GridLayout](#)
- [FlowLayout](#)
- [CardLayout](#)



## Multithreading programming

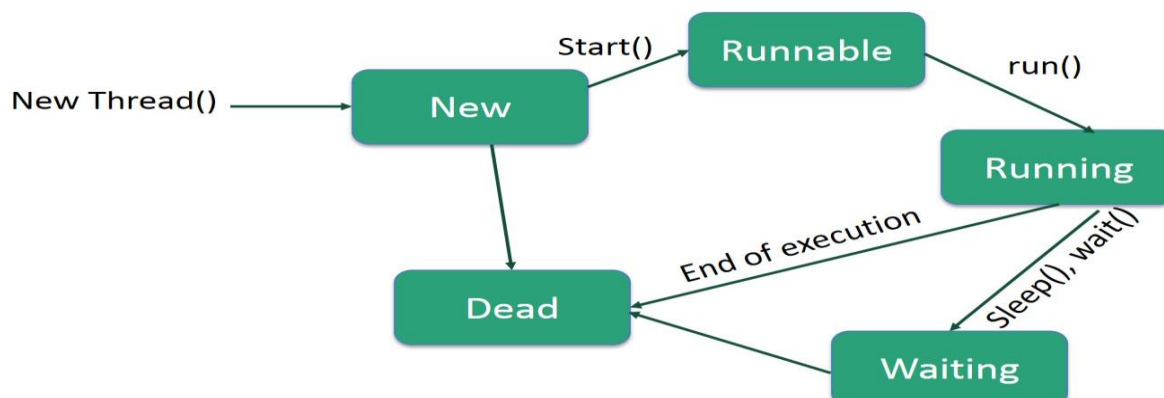
Java is a *multi-threaded programming language* which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

## Life Cycle of a Thread

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.



Following are the stages of the life cycle –

- **New** – A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** – After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** – Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** – A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** – A runnable thread enters the terminated state when it completes its task or otherwise terminates.

## Thread Priorities

Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

Java thread priorities are in the range between MIN\_PRIORITY (a constant of 1) and MAX\_PRIORITY (a constant of 10). By default, every thread is given priority NORM\_PRIORITY (a constant of 5).

Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

## Create a Thread by Implementing a Runnable Interface

If your class is intended to be executed as a thread then you can achieve this by implementing a **Runnable** interface. You will need to follow three basic steps –

## Step 1

As a first step, you need to implement a `run()` method provided by a **Runnable** interface. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of the `run()` method –

```
public void run( )
```

## Step 2

As a second step, you will instantiate a **Thread** object using the following constructor –

```
Thread(Runnable threadObj, String threadName);
```

Where, *threadObj* is an instance of a class that implements the **Runnable** interface and **threadName** is the name given to the new thread.

## Step 3

Once a Thread object is created, you can start it by calling **start()** method, which executes a call to `run()` method. Following is a simple syntax of `start()` method –

```
void start();
```

## Example

Here is an example that creates a new thread and starts running it –

[Live Demo](#)

```
class RunnableDemo implements Runnable {
    private Thread t;
    private String threadName;

    RunnableDemo( String name) {
        threadName = name;
        System.out.println("Creating " + threadName );
    }

    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " +
i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + "
interrupted.");
        }
    }
}
```

```

    }
    System.out.println("Thread " + threadName + " exiting.");
}

public void start () {
    System.out.println("Starting " + threadName );
    if (t == null) {
        t = new Thread (this, threadName);
        t.start ();
    }
}
}

public class TestThread {

    public static void main(String args[]) {
        RunnableDemo R1 = new RunnableDemo( "Thread-1");
        R1.start();

        RunnableDemo R2 = new RunnableDemo( "Thread-2");
        R2.start();
    }
}

```

This will produce the following result –

## Output

```

Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.

```

## Create a Thread by Extending a Thread Class

The second way to create a thread is to create a new class that extends **Thread** class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in Thread class.

## Step 1

You will need to override **run( )** method available in Thread class. This method provides an entry point for the thread and you will put your complete business logic inside this method. Following is a simple syntax of run() method –

```
public void run( )
```

## Step 2

Once Thread object is created, you can start it by calling **start()** method, which executes a call to run( ) method. Following is a simple syntax of start() method –

```
void start( );
```

## Example

Here is the preceding program rewritten to extend the Thread –

[Live Demo](#)

```
class ThreadDemo extends Thread {
    private Thread t;
    private String threadName;

    ThreadDemo( String name) {
        threadName = name;
        System.out.println("Creating " + threadName );
    }

    public void run() {
        System.out.println("Running " + threadName );
        try {
            for(int i = 4; i > 0; i--) {
                System.out.println("Thread: " + threadName + ", " +
i);
                // Let the thread sleep for a while.
                Thread.sleep(50);
            }
        } catch (InterruptedException e) {
            System.out.println("Thread " + threadName + "
interrupted.");
        }
        System.out.println("Thread " + threadName + " exiting.");
    }

    public void start () {
        System.out.println("Starting " + threadName );
        if (t == null) {
            t = new Thread (this, threadName);
            t.start ();
        }
    }
}
```

```

}

public class TestThread {

    public static void main(String args[]) {
        ThreadDemo T1 = new ThreadDemo( "Thread-1");
        T1.start();

        ThreadDemo T2 = new ThreadDemo( "Thread-2");
        T2.start();
    }
}

```

This will produce the following result –

## Output

```

Creating Thread-1
Starting Thread-1
Creating Thread-2
Starting Thread-2
Running Thread-1
Thread: Thread-1, 4
Running Thread-2
Thread: Thread-2, 4
Thread: Thread-1, 3
Thread: Thread-2, 3
Thread: Thread-1, 2
Thread: Thread-2, 2
Thread: Thread-1, 1
Thread: Thread-2, 1
Thread Thread-1 exiting.
Thread Thread-2 exiting.

```

## Thread Methods

Following is the list of important methods available in the Thread class.

Sr.No.	Method & Description
1	<b>public void start()</b> Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	<b>public void run()</b> If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.

3	<b>public final void setName(String name)</b> Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	<b>public final void setPriority(int priority)</b> Sets the priority of this Thread object. The possible values are between 1 and 10.
5	<b>public final void setDaemon(boolean on)</b> A parameter of true denotes this Thread as a daemon thread.
6	<b>public final void join(long millisec)</b> The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	<b>public void interrupt()</b> Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	<b>public final boolean isAlive()</b> Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

The previous methods are invoked on a particular Thread object. The following methods in the Thread class are static. Invoking one of the static methods performs the operation on the currently running thread.

Sr.No.	Method & Description
1	<b>public static void yield()</b> Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled.
2	<b>public static void sleep(long millisec)</b> Causes the currently running thread to block for at least the specified number of milliseconds.
3	<b>public static boolean holdsLock(Object x)</b> Returns true if the current thread holds the lock on the given Object.



4	<b>public static Thread currentThread()</b> Returns a reference to the currently running thread, which is the thread that invokes this method.
5	<b>public static void dumpStack()</b> Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

## Example

The following ThreadClassDemo program demonstrates some of these methods of the Thread class. Consider a class **DisplayMessage** which implements **Runnable** –

```
// File Name : DisplayMessage.java
// Create a thread to implement Runnable

public class DisplayMessage implements Runnable {
    private String message;

    public DisplayMessage(String message) {
        this.message = message;
    }

    public void run() {
        while(true) {
            System.out.println(message);
        }
    }
}
```

Following is another class which extends the Thread class –

```
// File Name : GuessANumber.java
// Create a thread to extend Thread

public class GuessANumber extends Thread {
    private int number;
    public GuessANumber(int number) {
        this.number = number;
    }

    public void run() {
        int counter = 0;
        int guess = 0;
        do {
            guess = (int) (Math.random() * 100 + 1);
            System.out.println(this.getName() + " guesses " +
guess);
            counter++;
        }
    }
}
```

```

        } while(guess != number);
        System.out.println("*** Correct!" + this.getName() + "in" +
counter + "guesses.**");
    }
}

```

Following is the main program, which makes use of the above-defined classes –

```

// File Name : ThreadClassDemo.java
public class ThreadClassDemo {

    public static void main(String [] args) {
        Runnable hello = new DisplayMessage("Hello");
        Thread thread1 = new Thread(hello);
        thread1.setDaemon(true);
        thread1.setName("hello");
        System.out.println("Starting hello thread...");
        thread1.start();

        Runnable bye = new DisplayMessage("Goodbye");
        Thread thread2 = new Thread(bye);
        thread2.setPriority(Thread.MIN_PRIORITY);
        thread2.setDaemon(true);
        System.out.println("Starting goodbye thread...");
        thread2.start();

        System.out.println("Starting thread3...");
        Thread thread3 = new GuessANumber(27);
        thread3.start();
        try {
            thread3.join();
        } catch (InterruptedException e) {
            System.out.println("Thread interrupted.");
        }
        System.out.println("Starting thread4...");
        Thread thread4 = new GuessANumber(75);

        thread4.start();
        System.out.println("main() is ending...");
    }
}

```

This will produce the following result. You can try this example again and again and you will get a different result every time.

## Output

```

Starting hello thread...
Starting goodbye thread...
Hello
Hello
Hello
Hello

```

Hello  
Hello  
Goodbye  
Goodbye  
Goodbye  
Goodbye  
Goodbye  
.....

## Major Java Multithreading Concepts

While doing Multithreading programming in Java, you would need to have the following concepts very handy –

- [What is thread synchronization?](#)
- [Handling interthread communication](#)
- [Handling thread deadlock](#)
- [Major thread operations](#)

## UNIT: 2

**JDBC:** Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. ... It provides methods to query and update data in a database, and is oriented towards relational databases.

### Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

These steps are described in detail in the sections that follow.

### Import JDBC Packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

Additionally, depending on the features being used, Oracle-supplied JDBC packages might need to be imported. For example, the following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;
```

```
import oracle.sql.*;
```

## Load and Register the JDBC Driver

This is for establishing a communication between the JDBC program and the Oracle database. This is done by using the static `registerDriver()` method of the `DriverManager` class of the JDBC API. The following line of code does this job:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

### JDBC Driver Registration

For the entire Java application, the JDBC driver is registered only once per each database that needs to be accessed. This is true even when there are multiple database connections to the same data server.

Alternatively, the `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

However, the `forName()` method is valid for only JDK-compliant Java Virtual Machines and implicitly creates an instance of the Oracle driver, whereas the `registerDriver()` method does this explicitly.

## Connecting to a Database

Once the required packages have been imported and the Oracle JDBC driver has been loaded and registered, a database connection must be established. This is done by using the `getConnection()` method of the `DriverManager` class. A call to this method creates an object instance of the `java.sql.Connection` class. The `getConnection()` requires three input parameters, namely, a connect string, a username, and a password. The connect string should specify the JDBC driver to be used and the database instance to connect to.

The `getConnection()` method is an overloaded method that takes

- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

The following lines of code illustrate using the `getConnection()` method:

```
Connection conn = DriverManager.getConnection(URL, username, passwd);
```

```
Connection conn = DriverManager.getConnection(URL);
```

where URL, username, and passwd are of `String` data types.  
We will discuss the methods of opening a connection using the Oracle JDBC OCI and thin \_drivers.

When using the OCI driver, the database can be specified using the TNSNAMES entry in the tnsnames.ora file. For example, to connect to a database on a particular host as user oratest and password oratest that has a TNSNAMES entry of oracle.world, use the following code:

```
Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:
@oracle.world", "oratest", "oratest");
```

Both the ":" and "@" are mandatory.

When using the JDBC thin driver, the TNSNAMES entry cannot be used to identify the database. There are two ways of specifying the connect string in this case, namely,

- Explicitly specifying the hostname, the TCP/IP port number, and the Oracle SID of the database to connect to. This is for thin driver only.
- Specify a Net8 keyword-value pair list.

For example, for the explicit method, use the following code to connect to a database on host `training` where the TCP/IP listener is on port 1521, the SID for the database instance is Oracle, the username and password are both oratest:

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@training:1521:Oracle",
"oratest", "oratest");
```

For the Net8 keyword-value pair list, use the following:

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin@(description=(address=
(host=training) (protocol=tcp) (port=1521))
(connect_data=(sid=Oracle))) ", "_oratest", "oratest");
```

This method can also be used for the JDBC OCI driver. Just specify `oci8` instead of `thin` in the above keyword-value pair list.

## Querying the Database

Querying the database involves two steps: first, creating a statement object to perform a query, and second, executing the query and returning a resultset.

### Creating a Statement Object

This is to instantiate objects that run the query against the database connected to. This is done by the `createStatement()` method of the `conn Connection` object created above. A call to this method creates an object instance of the `Statement` class. The following line of code illustrates this:

```
Statement sql_stmt = conn.createStatement();
```

### Executing the Query and Returning a ResultSet

Once a `Statement` object has been constructed, the next step is to execute the query. This is done by using the `executeQuery()` method of the `Statement` object. A call to this method takes as parameter a SQL `SELECT` statement and returns a `JDBC ResultSet` object. The following line of code illustrates this using the `sql_stmt` object created above:

```
ResultSet rset = sql_stmt.executeQuery  
  
    ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
```

Alternatively, the SQL statement can be placed in a string and then this string passed to the `executeQuery()` function. This is shown below.

```
String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";  
  
ResultSet rset = sql_stmt.executeQuery(sql);
```

### Statement and ResultSet Objects

`Statement` and `ResultSet` objects open a corresponding cursor in the database for `SELECT` and other DML statements.

The above statement executes the `SELECT` statement specified in between the double quotes and stores the resulting rows in an instance of the `ResultSet` object named `rset`.

### Processing the Results of a Database Query That Returns Multiple Rows

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

The first step is done using the `next()` method of the `ResultSet` object. A call to `next()` is executed in a loop to fetch the rows one row at a time, with each call to `next()` advancing the control to the next available row. The `next()` method returns the Boolean value `true` while rows are still available for fetching and returns `false` when all the rows have been fetched.

The second step is done by using the `getXXX()` methods of the `JDBC rset` object. Here `getXXX()` corresponds to the `getInt()`, `getString()` etc with `XXX` being replaced by a Java datatype.

The following code demonstrates the above steps:

```
String str;  
  
while (rset.next())
```

```

{

    str = rset.getInt(1)+ " " + rset.getString(2)+ "

        "+rset.getFloat(3)+ " " +rset.getInt(4)+ "\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();

```

Here the 1, 2, 3, and 4 in `rset.getInt()`, `rset.getString()`, `getFloat()`, and `getInt()` respectively denote the position of the columns in the SELECT statement, that is, the first column `empno`, second column `ename`, third column `sal`, and fourth column `deptno` of the SELECT statement respectively.

#### Specifying `get()` Parameters

The parameters for the `getXXX()` methods can be specified by position of the corresponding columns as numbers 1, 2, and so on, or by directly specifying the column names enclosed in double quotes, as `getString("ename")` and so on, or a combination of both.

### Closing the ResultSet and Statement

Once the `ResultSet` and `Statement` objects have been used, they must be closed explicitly. This is done by calls to the `close()` method of the `ResultSet` and `Statement` classes. The following code illustrates this:

```

rset.close();

sql_stmt.close();

```

If not closed explicitly, there are two disadvantages:

- Memory leaks can occur
- Maximum Open cursors can be exceeded

Closing the `ResultSet` and `Statement` objects frees the corresponding cursor in the database.

### Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the `close()` method of the `Connection` class.

The following line of code does this:

```

conn.close();

```

### Explicitly Close your Connection

Closing the `ResultSet` and `Statement` objects does not close the connection. The connection should be closed by explicitly invoking the `close()` method of the `Connection` class.

A complete example of the above procedures using a JDBC thin driver is given below. This program queries the `emp` table and writes the output rows to an operating system file.

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) throws SQLException, IOException

    {

        //Load and register Oracle driver

        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

        //Establish a connection

        Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

@training:1521:Oracle", "oratest", "oratest");

        //Create a Statement object

        Statement sql_stmt = conn.createStatement();

        //Create a ResultSet object, execute the query and return a

        // resultset

        ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");
```



```

//Process the resultset, retrieve data in each row, column by column

//and write to an operating system file

String str = "";

while (rset.next())

{

    str += rset.getInt(1)+" "+ rset.getString(2)+" "+

    rset.getFloat(3)+" "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

}

}

```

### Processing the Results of a Database Query That Returns a Single Row

The above sections and the complete example explained the processing of a query that returned multiple rows. This section highlights the processing of a single-row query and explains how to write code that is the analogue of the PL/SQL exception `NO_DATA_FOUND`.

## NO DATA FOUND Exception

`NO_DATA_FOUND` exception in PL/SQL is simulated in JDBC by using the return value of the `next()` method of the `ResultSet` object. A value of `false` returned by the `next()` method identifies a `NO_DATA_FOUND` exception. Consider the following code (this uses the `ResultSet` object `rset` defined in the above sections):

```
if (rset.next())

    // Process the row returned

else

    System.out.println("The Employee with Empno " + args[1] +

                       "does not exist");
```

Instead of the while loop used earlier, an if statement is used to determine whether the `SELECT` statement returned a row or not.

## Datatype Mappings

Corresponding to each SQL data type, there exist mappings to the corresponding JDBC Types, standard Java types, and the Java types provided by Oracle extensions. These are required to be used in JDBC programs that manipulate data and data structures based on these types.

There are four categories of Data types any of which can be mapped to the others. These are:

- **SQL Data types**—These are Oracle SQL data types that exist in the database.
- **JDBC Typecodes**—These are the data typecodes supported by JDBC as defined in the `java.sql.Types` class or defined by Oracle in `oracle.jdbc.driver.OracleTypes` class.
- **Java Types**—These are the standard types defined in the Java language.
- **Oracle Extension Java Types**—These are the Oracle extensions to the SQL data types and are defined in the `oracle.sql.*` class. Mapping SQL data types to the `oracle.sql.*` Java types enables storage and retrieval of SQL data without first converting into Java format thus preventing any loss of information.

Table 3.1 lists the default mappings existing between these four different types.

**Table 3.1 Standard and Oracle-specific SQL-Java Data Type Mappings**

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
<b>Standard JDBC 1.0 Types</b>			
CHAR	<code>java.sql.Types.CHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
VARCHAR2	<code>java.sql.Types.VARCHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
LONG	<code>java.sql.Types.</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR_</code>

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
	LONGVARCHAR		
NUMBER	java.sql.Types.NUMERIC	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.DECIMAL	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIT	Boolean	oracle.sql.NUMBER
NUMBER	java.sql.Types.TINYINT	byte	oracle.sql.NUMBER
NUMBER	java.sql.Types.SMALLINT	short	oracle.sql.NUMBER
NUMBER	java.sql.Types.INTEGER	int	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIGINT	long	oracle.sql.NUMBER
NUMBER	java.sql.Types.REAL	float	oracle.sql.NUMBER
NUMBER	java.sql.Types.FLOAT	double	oracle.sql.NUMBER
NUMBER	java.sql.Types.DOUBLE	double	oracle.sql.NUMBER
RAW	java.sql.Types.BINARY	byte[]	oracle.sql.RAW
RAW	java.sql.Types.VARBINARY	byte[]	oracle.sql.RAW
LONGRAW	java.sql.Types.LONGVARBINARY	byte[]	oracle.sql.RAW
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE
DATE	java.sql.Types.TIME	java.sql.Time	oracle.sql.DATE
DATE	java.sql.Types.TIMESTAMP	java.sql.Timestamp	oracle.sql.DATE

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
<b>Standard JDBC 2.0 Types</b>			
BLOB	java.sql.Types.BLOB	java.sql.Blob	Oracle.sql.BLOB
CLOB	Java.sql.Types.CLOB	java.sql.Clob	oracle.sql.CLOB
user-defined	java.sql.Types.STRUCT	java.sql.Struct	oracle.sql.STRUCT_object
user-defined	java.sql.Types.REF	java.sql.Ref	oracle.sql.REF_reference
user-defined	java.sql.Types.ARRAY	java.sql.Array	oracle.sql.ARRAY_collecti on
<b>Oracle Extensions</b>			
BFILE	oracle.jdbc.driver. oracle.sql.BFILE_	n/a	OracleTypes.BFILE
ROWID	oracle.jdbc.driver. oracle.sql.ROWID_	n/a	OracleTypes.ROWID
REFCURSOR R type	oracle.jdbc.driver. OracleTypes.CURSOR	java.sql.ResultSet	oracle.jdbc.driver._ OracleResultSet

## Exception Handling in JDBC

Like in PL/SQL programs, exceptions do occur in JDBC programs. Notice how the `NO_DATA_FOUND` exception was simulated in the earlier section "Processing the Results of a Database Query That Returns a Single Row."

Exceptions in JDBC are usually of two types:

- Exceptions occurring in the JDBC driver
- Exceptions occurring in the Oracle 8i database itself

Just as PL/SQL provides for an implicit or explicit `RAISE` statement for an exception, Oracle JDBC programs have a `throw` statement that is used to inform that JDBC calls throw the SQL exceptions. This is shown below.

```
throws SQLException
```

This creates instances of the class `java.sql.SQLException` or a subclass of it. And, like in PL/SQL, SQL exceptions in JDBC have to be handled explicitly. Similar to PL/SQL exception handling sections, Java provides a `try..catch` section that can handle all exceptions including SQL exceptions. Handling an exception can basically include retrieving the error code, error text, the SQL state, and/or printing the error stack trace. The `SQLException` class provides methods for obtaining all of this information in case of error conditions.

### Retrieving Error Code, Error Text, and SQL State

There are the methods `getErrorCode()` and `getMessage()` similar to the functions `SQLCODE` and `SQLERRM` in PL/SQL. To retrieve the SQL state, there is the method `getSQLState()`. A brief description of these methods is given below:

- `getErrorCode()`
  - This function returns the five-digit ORA number of the error in case of exceptions occurring in the JDBC driver as well as in the database.
- `getMessage()`
  - This function returns the error message text in case of exceptions occurring in the JDBC driver. For exceptions occurring in the database, this function returns the error message text prefixed with the ORA number.
- `getSQLState()`
  - This function returns the five digit code indicating the SQL state only for exceptions occurring in the database.

The following code illustrates the use of exception handlers in JDBC:

```
try { <JDBC code> }

catch (SQLException e) { System.out.println("ERR: "+ e.getMessage()) }
```

We now show the `QueryExample` class of the earlier section with complete exception handlers built in it. The code is as follows:

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) {

        int ret_code;

        try {

            //Load and register Oracle driver
```

```

    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

//Establish a connection

Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

@training:1521:Oracle", "oratest", "oratest");

//Create a Statement object

Statement sql_stmt = conn.createStatement();

//Create a ResultSet object, execute the query and return a

// resultset

ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");

//Process the resultset, retrieve data in each row, column by column

// and write to an operating system file

String str = "";

while (rset.next())

{

    str += rset.getInt(1)+" "+ rset.getString(2)+" "+rset.getFloat(3)+

    " "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

```

```

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

} catch (SQLException e) {ret_code = e.getErrorCode();

System.err.println("Oracle Error: " + ret_code + e.getMessage());}

catch (IOException e) {System.out.println("Java Error: " +

e.getMessage()); }

}

}

```

### Printing Error Stack Trace

The `SQLException` has the method `printStackTrace()` for printing an error stack trace. This method prints the stack trace of the throwable object to the standard error stream. The following code illustrates this:

```

catch (SQLException e) { e.printStackTrace(); }

```

## JAVABEANS:

### 1. How to Install NetBeans 8.2

#### 1.1 How to Install NetBeans on Windows

Step 0: Install JDK

To use NetBeans for Java programming, you need to first install Java Development Kit (JDK). See "[JDK - How to Install](#)".

Step 1: Download

Download "NetBeans IDE" installer from <http://netbeans.org/downloads/index.html>. There are many "bundles" available. For beginners, choose the 1st entry "Java SE" (e.g., "netbeans-8.2-javase-windows.exe" 95MB).

## Step 2: Run the Installer

Run the downloaded installer.

### 1.2 How to Install NetBeans on Mac OS X

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Mac](#)".

To install NetBeans:

1. Download NetBeans from <http://netbeans.org/downloads/>. Set "Platform" to "Mac OS X". There are many "bundles" available. For beginners, choose "Java SE" (e.g., "netbeans-8.2-javase-macosx.dmg" 116MB).
2. Double-click the download Disk Image (DMG) file.
3. Double-click the "NetBeans 8.x.mpkg", and follow the instructions to install NetBeans. NetBeans will be installed under "/Applications/NetBeans".
4. Eject the Disk Image (".dmg").

You can launch NetBeans from the "Applications".

**Notes:** To uninstall NetBeans, drag the "/Applications/NetBeans" folder to trash.

### 1.3 How to Install NetBeans on Ubuntu Linux

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Ubuntu](#)".

To install NetBeans:

1. Download NetBeans from <http://netbeans.org/downloads/>. Choose platform "Linux (x86/x64)" ⇒ "Java SE". You shall receive a sh file (e.g., "netbeans-7.x-m1-javase-linux.sh") in "~/Downloads".
2. Set the downloaded sh file to executable and run the sh file. Open a Terminal:

```
3. $ cd ~/Downloads
4. $ chmod a+x netbeans-7.x-m1-javase-linux.sh // Set to executable for all (a+x)

$ ./netbeans-7.x-m1-javase-linux.sh // Run
```

Follow the instructions to install NetBeans.

To start NetBeans, run the script "netbeans" in the NetBeans' bin directory:

```
$ cd netbeans-bin-directory
$ ./netbeans
```

## 2. Writing a Hello-world Java Program in NetBeans

### Step 0: Launch NetBeans

Launch NetBeans. If the "Start Page" appears, close it by clicking the "cross" button next to the "Start Page" title.

### Step 1: Create a New Project

For each Java application, you need to create a "project" to keep all the source files, classes and relevant resources.

1. From "File" menu ⇒ Choose "New Project...".
2. The "Choose Project" dialog pops up ⇒ Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ "Next".
3. The "Name and Location" dialog pops up ⇒ Under "Project Name", enter "FirstProject" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Uncheck "Use Dedicated Folder for Storing Libraries" ⇒ **Uncheck "Create Main class"** ⇒ Finish.

### Step 2: Write a Hello-world Java Program

1. Right-click on "FirstProject" ⇒ New ⇒ Java Class (OR choose the "File" menu ⇒ "New File..." ⇒ Categories: "Java", File Types: "Java Class" ⇒ "Next").
2. The "Name and Location" dialog pops up ⇒ In "Class Name", enter "Hello" ⇒ Delete the content in "Package" if it is not empty ⇒ "Finish".
3. The source file "Hello.java" appears in the editor panel. Enter the following codes:

```
4. public class Hello {
```



```

5.     public static void main(String[] args) {
6.         System.out.println("Hello, world");
7.     }
}

```

### Step 3: Compile & Execute

There is no need to "compile" the source code in NetBeans explicitly, as NetBeans performs the so-called *incremental compilation* (i.e., the source statement is compiled as and when it is entered).

To run the program, right-click anywhere in the source (or from the "Run" menu) ⇒ Run File. Observe the output on the output console.

#### Notes:

- You should create a NEW Java project for EACH of your Java application.
- Nonetheless, NetBeans allows you to keep more than one programs in a project, which is handy for writing toy programs (such as your tutorial exercises). To run a particular program, open and right-click on the source file ⇒ Run File.

## 2.1 Correcting Syntax Error

NetBeans performs incremented compilation, as and when a source line is entered. It marked a source line with syntax error with a RED CROSS. Point your cursor at the RED CROSS to view the error message.

You CANNOT RUN the program if there is any syntax error (marked by a RED CROSS before the filename). Correct all the syntax errors; and RUN the program.

[TODO] Diagram

HINTS: In some cases, NetBeans shows a ORANGE LIGHT-BULB (for HINTS) next to the ERROR RED-CROSS (Line 5 in the above diagram). You can click on the LIGHT-BULB to get a list of HINTS to resolve this particular error, which may or may not work!

SYNTAX WARNING: marked by a orange triangular exclamation sign. Unlike errors, warnings may or may not cause problems. Try to fix these warnings as well. But you can RUN your program with warnings.

## 3. Read the NetBeans Documentation

At a minimum, you SHOULD READ the "IDE Basics, Getting Started, Java Application", which is accessible via NetBeans's "HELP" menu ⇒ Help Contents. This will save you many agonizing hours trying to figure out how to do somethings later.

The "Help" ⇒ "Online Doc and Support" (@ <http://netbeans.org/kb/index.html>) contains many articles and tutorial on using NetBeans.

The NetBeans "Start Page" also provides many useful links to get you started.

## 4. Debugging Program in NetBeans

### Step 0: Write a Java Program

The following program computes and prints the factorial of  $n$  ( $=1*2*3*\dots*n$ ). The program, however, has a logical error and produce a wrong answer for  $n=20$  ("The Factorial of 20 is -2102132736" - a negative number?!).

```

1/** Compute the factorial of n */
2public class Factorial {
3    // Print factorial of n
4    public static void main(String[] args) {
5        int n = 20;
6        int factorial = 1;
7
8        // n! = 1*2*3...*n
9        for (int i = 1; i <= n; i++) {
10            factorial *= i;
11        }

```

```
12     System.out.println("The Factorial of " + n + " is " + factorial);
13 }
14 }
```

Let us use the graphic debugger to debug the program.

#### Step 1: Set an initial Breakpoint

A *breakpoint* suspends program execution for you to examine the internal states of the program. Before starting the debugger, you need to set at least one breakpoint to suspend the execution inside the program. Set a breakpoint at `main()` method by clicking on the *left-margin* of the line containing `main()`. A *red circle* or an inverted Triangle appears in the left-margin indicating a breakpoint is set at that line.

#### Step 2: Start Debugging

Right click anywhere on the source code ⇒ "Debug File". The program begins execution but suspends its operation at the breakpoint, i.e., the `main()` method.

As illustrated in the following diagram, the highlighted line (also pointed to by a green arrow) indicates the statement to be executed in the *next* step.

#### Step 3: Step-Over and Watch the Variables and Outputs

Click the "Step Over" button (or select "Step Over" in "Debug" menu) to *single-step* thru your program. At each of the step, examine the value of the variables (in the "Variable" panel) and the outputs produced by your program (in the "Output" Panel), if any. You can also place your cursor at any variable to inspect the content of the variable.

Single-stepping thru the program and watching the values of internal variables and the outputs produced is the *ultimate* mean in debugging programs - because it is exactly how the computer runs your program!

#### Step 4: Breakpoint, Run-To-Cursor, Continue and Finish

As mentioned, a breakpoint *suspends* program execution and let you examine the internal states of the program. To set a breakpoint on a particular statement, click on the left-margin of that line (or select "Toggle Breakpoint" from "Run" menu).

"Continue" resumes the program execution, up to the next breakpoint, or till the end of the program.

"Single-step" thru a loop with a large count is time-consuming. You could set a breakpoint at the statement immediately outside the loop (e.g., Line 11 of the above program), and issue "Continue" to complete the loop.

Alternatively, you can place the cursor on a particular statement, and issue "Run-To-Cursor" to resume execution up to the line.

"Finish" ends the debugging session. Always terminate your current debugging session using "Finish" or "Continue" till the end of the program.

### 4.1 Other Debugger's Features:

#### Modify the Value of a Variable

You can modify the value of a variable by entering a new value in the "Variable" panel. This is handy for temporarily modifying the behaviour of a program, without changing the source code.

#### Step-Into and Step-Out

To debug a *method*, you need to use "Step-Into" to step into the *first* statement of the method. You could use "Step-Out" to return back to the caller, anywhere within the method. Alternatively, you could set a breakpoint inside a method.

## 5. NetBeans - Tips & Tricks

## 5.1 General Usage

---

These are the features that I find to be most useful in NetBeans:

1. **Maximizing Window (double-click):** You can double-click on the "header" of any panel to *maximize* that particular panel, and double-click again to *restore* it back. This is particularly useful for editing source code in full panel.
2. **Code Auto-Complete (or Intelli-Sense) (ctrl-space):** Enter a partial statement (e.g., Sys) and press control-space to activate the auto-complete, which displays all the available choices.
3. **Javadoc (ctrl-space, alt-F1):** Place the cursor on a method or class, and press ctrl-space to view the javadoc; or right-click ⇒ Show Javadoc (alt-F1) to open it on a browser.
4. **Code Shorthand (tab):** For example, you can enter "sout" and press TAB for "System.out.println"; "psvm" for "public static void main(String[] args) { }" or "fori" + tab for a for-loop. To view and configure code template, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Code Templates".
5. **Formatting Source Code (alt-shift-f):** Right-click on the source (or from the "Source" menu) ⇒ Choose "Format". NetBeans will layout your source codes with the proper indents and format. To configure the formatting, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Formatting". You can also select the section of codes to be formatted, instead of the entire file.
6. **Hints for Correcting Syntax Error:** If there is a syntax error on a statement, a red mark will show up on the left-margin on that statement. You could click on the "light bulb" to display the error message, and also select from the available hints for correcting that syntax error.
7. **Rename (Refactor) (ctrl-r):** To rename a variable, place the cursor on that variable, right-click ⇒ "Refactor" ⇒ "Rename" ⇒ Enter the new name. All the appearances of that variables in the project will be renamed.
8. **Small Programs:** You can keep many small toy programs (with `main()`) in one Java project instead of create a new project for each small program. To run the desired program, on the "editor" panel ⇒ right-click ⇒ "Run File".
9. **Source Toggle Comment:** To temporarily comment-off a block of codes, choose "Source" ⇒ "Toggle Comment".
10. **Error Message Hyperlink:** Click on an error message will hyperlink to the corresponding source statement.
11. **Command-Line Arguments:** To provide command-line arguments to your Java program in NetBeans, right-click on the "project" ⇒ "Set as Main Project" ⇒ "Set Configurations" ⇒ "Customize..." ⇒ "Run" ⇒ select the "Main" class ⇒ type your command-line arguments inside the "Arguments" field ⇒ choose "Run" menu ⇒ "Run Main Project".
12. **Line Numbers:** To show the line numbers, right-click on the left-margin ⇒ "Show Line Numbers".
13. **Changing Font Face and Size:** Tools ⇒ Options ⇒ Fonts & Colors ⇒ In "Category", select "Default" ⇒ In "Font", choose the font face and size.
14. **Resetting Window View:** If you mess up the window view (e.g., you accidentally close a window and cannot find it anymore), you can reset the view via "Window" menu ⇒ "Reset Windows".
15. **Code Templates:** For example, when you create a *new* Java class, NetBeans retrieves the initial contents from the "Java Class" code template. To configure code templates, select "Tools" menu ⇒ "Templates" ⇒ Choose the desired template ⇒ "Open in Editor". To set a value of a variable used in the all the code templates (e.g., `$User`), select "Tools" menu ⇒ "Templates" ⇒ "Settings".
16. **Displaying Chinese Character:** Need to choose a font that support chinese character display, such as "Monospace", in Tools ⇒ Options ⇒ Fonts & Colors ⇒ Syntax ⇒ default.
17. **Changing the JDK Location:** The Netbeans configuration file is located at "etc\netbeans.conf". Edit the directive "netbeans\_jdkhome".
18. Let me know if you have more tips to be included here.

## 5.2 Java Application Development

---

1. **Choosing the JDK version for your program:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ You can select the JDK level of your project in pull-down menu "Source/Binary Format".
2. **Enabling JDK 7 support:** If JDK 7 is already installed in your system, right-click on your Project ⇒ "Properties" ⇒ "Source" node ⇒ "Source/Binary Format" ⇒ Select "JDK 7". Also check "Libraries" ⇒ Java

Platform ⇒ JDK 7.  
 If JDK 7 is not installed/configured, install JDK 7. Add JDK 7 support to NetBeans via "Tool" menu ⇒ "Java Platforms" ⇒ "Add Platform...".

3. **Choosing Default Charset:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your desired charset for the text-file I/O from the pull-down menu.
4. **Enabling Unicode Support for File Encoding:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your Unicode encoding (e.g., UTF-8, UTF-16, UTF-16LE, UTF-16GE) for the text-file I/O.
5. **To include Javadoc/Source:** Use "Library Manager" (select the "Tools" menu ⇒ "Libraries"); or "Java Platform Manager" (select "Tools" menu ⇒ "Java Platforms")
6. **Adding External JAR files & Native Libraries (".dll", ".lib", ".a", ".so"):** Many external Java packages (such as JOGL, Java3D, JAMA, etc) are available to extend the functions of JDK. These packages typically provide a "lib" directory containing JAR files (".jar") (Java Archive - a single-file package of Java classes) and native libraries (".dll", ".lib" for windows, ".a", ".so" for Linux and Mac).

To include an external JAR file (".jar") into a project: Expand the project node ⇒ Right-click on "Libraries" ⇒ "Add JAR/Folder..." ⇒ Select the desired JAR file or the folder containing the classes. If the external package contains many JAR files, you could create a user library to contain all the JAR files, and add the library to all the projects that required these JAR files. From "Tools" menu ⇒ "Libraries" ⇒ "New Library..." ⇒ Enter a library name ⇒ Use "Add JAR/Folder..." to add JAR files into this library. Many JAR files come with native libraries in the form of ".dll", ".lib" (for Windows) and ".a", ".so" for Linux/Mac. The directory *path* of these libraries must be included in JRE's property "java.library.path". This can be done via right-click the project ⇒ Set Configuration ⇒ Customize... ⇒ Run ⇒ In "VM options", enter "-Djava.library.path=xxx", where xxx is path of the native libraries.

**Notes:** The JAR files must be included in the CLASSPATH. The native library directories must be included in JRE's property "java.library.path", which normally but not necessarily includes all the paths from the PATH environment variable. Read "[External JAR files and Native Libraries](#)".

## 6. Writing Java GUI (AWT/Swing) Application in NetBeans

Step 0: Read

1. Java GUI Application Learning Trail @ <http://www.netbeans.org/kb/trails/matisse.html>.
2. Swing Tutorial's "Learning Swing with the NetBeans IDE" @ <http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html>.

Step 1: Create a New "Java Application" Project

1. Launch NetBeans ⇒ File ⇒ New Project...
2. Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ Next.
3. In "Project Name", enter "FirstNetBeansGUI" ⇒ Choose a suitable directory for your "Project Location" ⇒ *Uncheck* the "Create Main class" box ⇒ Finish.

Step 2: Write a Java File "JFrame Form"

1. Right-click on the project "FirstNetBeansGUI" ⇒ "New" ⇒ "JFrame Form..." (or "Others" ⇒ "Swing GUI Forms" ⇒ "JFrame Form").
2. In "Class Name", enter "NetBeansSwingCounter" ⇒ Finish.
3. Create the GUI Components visually:
  - a. From the "Platte" panel ⇒ "Swing Controls" ⇒ Drag and drop a "Label", "TextField", and "Button" into the *design* panel.
  - b. Click on the "jLabel1" ⇒ In the "Properties" panel, enter "Count" in "text" (You can also single-click on the jLabel1 to change the text). Right-click on the jLabel1 ⇒ Change Variable Name ⇒ In "New Name", enter "lblCount".
  - c. Similarly, for "jTextField1" ⇒ Change the "text" to 0, and change the "Variable Name" to "tfCount" ⇒ Resize the text field if necessary.
  - d. For "jButton1" ⇒ Change the "text" to "Count", and change the "Variable Name" to "btnCount".
4. Write the event handler for the button by double-clicking the button and enter the following codes:

```

5. private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {
6.     count++;
7.     tfCount.setText(count + "");
}

```

8. Create an instance variable `count` (just below the class declaration) as follows:

```

9. public class Counter extends javax.swing.JFrame {

    int count = 0;
}

```

### Step 3: Compile & Execute

Right-click the source and select "Run File".

### Step 4: Study the Generated Source Code

Expand the "Generated Code" and study how the GUI builder declare, allocate and initialize the GUI Components in the  `initComponents()`. Note how the `JButton` registers an `ActionEvent` listener and how an inner class is used as the listener and provide the event handler  `actionPerformed()`. Also notice that the `main()` method uses a Swing's worker to run the GUI on the Event-Dispatcher thread, instead of the main thread, for thread-safe operations.

```

public class NetBeansSwingCounter extends javax.swing.JFrame {
    int count = 0;

    // Constructor to setup the UI via initComponents()
    public NetBeansSwingCounter() {
        initComponents();
    }

    private void initComponents() {
        lblCount = new javax.swing.JLabel();
        tfCount = new javax.swing.JTextField();
        btnCount = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        lblCount.setText("Counter");
        tfCount.setText("0");

        btnCount.setText("Count");
        // Create an anonymous inner as the listener for the ActionEvent fired by btnCount
        btnCount.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnCountActionPerformed(evt);
            }
        });

        // Laying out the components
        // .....

        pack();
    }

    // ActionEvent handler for btnCount
    private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {
        count++;
        tfCount.setText(count + "");
    }

    public static void main(String args[]) {
        // Setup the Look and Feel
        // .....

        // Run the constructor on the Event-Dispatcher Thread for thread-safe
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new NetBeansSwingCounter().setVisible(true);
            }
        });
    }
}

```

```

}

// private variables
private javax.swing.JButton btnCount;
private javax.swing.JLabel lblCount;
private javax.swing.JTextField tfCount;
}

```

## 7. NetBeans and MySQL

**Reference:** "Connecting to a MySQL Database" @ <http://netbeans.org/kb/docs/ide/mysql.html>.

NetBeans (JavaEE) provides direct support to MySQL server. You can use NetBeans as a GUI client to access a MySQL server, as well as an administrative tool (e.g., starting and stopping the server).

### Configuring NetBeans to Support MySQL

From NetBeans "Window" menu ⇒ Select "Services". The "Services" tab shall appear on the *left* pane

1. Right-click on the "Databases" node ⇒ "Register MySQL Server". (If you have already registered a MySQL server, you can right-click on Server node "MySQL Server at *hostname:port*" ⇒ Properties, to modify its properties.)
2. Select the "Basic Properties" tab, enter the hostname, port number, root user and password.
3. Select the "Admin Properties" tab:
  - a. Leave the "Path/URL to admin tool" empty.
  - b. In "Path to start command", enter "<MYSQL\_HOME>\bin\mysqld.exe"; in the "Arguments", enter "--console"
  - c. In "Path to stop command", enter "<MYSQL\_HOME>\bin\mysqladmin.exe", in the "Arguments", enter "-u root -ppassword shutdown".
4. A server node "MySQL Server at *hostname:port*" appears.

### Database Administration - Start/Stop the Server and Create Databases

1. You can start the MySQL server by right-clicking on the server node ⇒ select "start". [There seems to be a problem here. If a "connection refused: connect" error occurs, enter the password again.]
2. Once the MySQL server is started and connected, you can see the list of databases by expanding the MySQL server node. You can create a new database by right-clicking on it and choose "Create Database...".

### Create a new Connection

You need a connection to manipulate data. You can create multiple connections with different users and default databases.

1. Right-click on the "Databases" ⇒ "New Connection..." ⇒ Select the driver "MySQL Connector/J" ⇒ Next ⇒ Enter hostname, port number, default database, a general username and password ⇒ "Test Connection" (make sure that MySQL is started) ⇒ Finish.
2. A connection node "jdbc:mysql://*hostname:port/defaultDatabase*" appears.

### Manipulating Data via a Connection

1. Right-click on a connection node (e.g., "jdbc:mysql://*hostname:port/defaultDatabase*") ⇒ Choose "Connect" (if not connected, provided that the MySQL server has been started).
2. You can expand the connection node to view all the databases.
3. Expand an existing database. There are three sub-nodes "Tables", "View" and "Procedures". Right-click on the "Tables" to create table or execute command. Similarly, right-click on the "View" and "Procedures".
4. To view/manipulate the records in a table, right-click on the selected table ⇒ You can choose to "View Data...", "Execute Command...", etc.
5. You can right-click on the connection to "connect" or "disconnect" from the server.

### Create a SQL Script and Run the Script

You can create a SQL script by right-clicking on a project ⇒ New ⇒ "SQL File". You can run the script by right-clicking on the SQL script ⇒ "Run File" ⇒ Select an existing connection (or create a new connection) to run the

script. You could also run a single statement (right-click on the statement ⇒ Run Statement) or a selected group of statements (highlight the statements ⇒ Right-click ⇒ Run Selection).

## 8. Developing and Deploying Web Application in NetBeans

Read:

- "Introduction to Developing Web Applications" @ <http://netbeans.org/kb/docs/web/quickstart-webapps.html>.
- More articles in "Java EE & Java Web Learning Trail" @ <http://netbeans.org/kb/trails/java-ee.html>.

### 8.1 Web (HTTP) Servers

Configuring Web Server

You could configure the web server via "Tools" menu ⇒ "Servers".

Tomcat Server

To configure Tomcat Server, select "Tools" menu ⇒ "Servers" ⇒ click "Add Servers":

1. Choose Server: Select the desired Tomcat version ⇒ Next.
2. Installation and Login Details: In "Server Location", fill in the Tomcat installation directory (\$CATALINA\_HOME) ⇒ Enter the username/password of a tomcat user with "manager" role. You could either check the "create user if it does not exist" or define the tomcat user in "\$CATALINA\_HOME\conf\tomcat-users.xml" as follows:

```
3. <tomcat-users>
4.     <role rolename="manager"/>
5.     <user username="tomcatmanager" password="xxxx" roles="manager,manager-script,admin" />

</tomcat-users>
```

Running the Web Server

Choose "Services" ⇒ Expand "Servers" node ⇒ Right-click on the desired server ⇒ Start/Stop/Restart.

### 8.2 MySQL Database Server

You can also manage the MySQL database server directly from Tomcat. Read "[NetBeans and MySQL](#)" Section.

### 8.3 Writing a Hello-World Servlet/JSP Web Application

Create a New Servlet/JSP Project

1. From "File" menu ⇒ choose "New Project...".
2. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
3. "Name and Location" ⇒ In "Project Name", enter "HelloServletJSP" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
4. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
5. "Frameworks" ⇒ Select none for pure servlet/JSP application ⇒ Finish.

Writing a Hello-World JSP

A JSP page called "index.jsp" is automatically created, which says "Hello world!". To execute this JSP, right-click on the project ⇒ "Run". The URL is <http://localhost:8080/HelloServletJSP/index.jsp>.

Writing a Hello-World Servlet

1. Right-click on the project "HelloServletJSP" ⇒ New ⇒ Servlet.
2. "Name and Location" ⇒ In "Class Name", enter "HelloServlet" ⇒ In "Package", enter "hello" ⇒ Next.
3. "Configure Servlet Deployment" ⇒ In "Servlet Name", enter "HelloServletExample" ⇒ In "URL Pattern", enter "sayhello" ⇒ Finish.
4. Enter the following codes for "HelloServlet.java":

```
5. package hello;
6.
```



```

7. import java.io.IOException;
8. import java.io.PrintWriter;
9. import javax.servlet.ServletException;
10. import javax.servlet.http.HttpServlet;
11. import javax.servlet.http.HttpServletRequest;
12. import javax.servlet.http.HttpServletResponse;
13.
14. public class HelloServlet extends HttpServlet {
15.
16.     @Override
17.     public void doGet(HttpServletRequest request, HttpServletResponse response)
18.         throws IOException, ServletException {
19.         // Set the response message's MIME type (in Content-Type response header)
20.         response.setContentType("text/html;charset=UTF-8");
21.         // Get an output Writer to write the response message over the network
22.         PrintWriter out = response.getWriter();
23.         // Write the response message (in an HTML page) to display "Hello, world!"
24.         try {
25.             out.println("<!DOCTYPE html>");
26.             out.println("<html>");
27.             out.println("<head><title>Hello Servlet</title></head>");
28.             out.println("<body><h1>Hello, World (from Java Servlet)!</h1></body>");
29.             out.println("</html>");
30.         } finally {
31.             out.close(); // Always close the output writer
32.         }
33.     }
}

```

34. To execute the servlet: Right-click on the project ⇒ run ⇒ Change the URL to `http://localhost:8080/HelloServletJSP/sayhello`.

#### Generating a WAR-file for a Web Application

A WAR (Web Archive) file is basically a zip file for distributing web application in single file. You can use WinZip or WinRAR to inspect or unzip the war file.

To distribute the project as a war-file, right-click project ⇒ "Clean and Build". The war file is created in the "dist" directory. You can deploy the web application by dropping the war-file into Tomcat's "webapps" directory. Tomcat will automatically unzip the war-file and deploy the application upon startup.

#### Debugging Web Application

The most important reason for using IDE is to use the graphic debugger for debugging the program. You can set a breakpoint in your server-side Java codes, and "Debug" a web application, similar to a standalone application.

## 8.4 Writing a Hello-world JSF 2.0 Web Application

### Create a New JSF 2.0 Project

1. From "File" menu ⇒ choose "New Project...".
2. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
3. "Name and Location" ⇒ In "Project Name", enter "HelloJSF20" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
4. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
5. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 2.0" ⇒ Finish.
6. An "index.xhtml" JSF page is generated, as follows:

```

7. <?xml version='1.0' encoding='UTF-8' ?>
8. <!DOCTYPE html>
9. <html xmlns="http://www.w3.org/1999/xhtml"
10.     xmlns:h="http://java.sun.com/jsf/html">

```



```

11.     <h:head>
12.         <title>Facelet Title</title>
13.     </h:head>
14.     <h:body>
15.         Hello from Facelets
16.     </h:body>

```

```
</html>
```

To run this facelet, right-click on the project ⇒ Run.

Create a new JSF 2.0 Facelet

1. Right-click on the project ⇒ New ⇒ "Other..."
2. "Choose File Type" ⇒ Under "Category", select "JavaServer Faces" ⇒ Under "File Type", select "JSF Page" ⇒ Next.
3. "Name and Location" ⇒ In "File Name", enter "HelloJSF20" ⇒ In "Options", check "Facelets" ⇒ Finish.
4. In "HelloJSF20.xhtml", enter the following codes:

```

5. <?xml version='1.0' encoding='UTF-8' ?>
6. <!DOCTYPE html>
7. <html xmlns="http://www.w3.org/1999/xhtml"
8.     xmlns:h="http://java.sun.com/jsf/html">
9.     <h:head>
10.         <title>Hello JSF 2.0</title>
11.     </h:head>
12.     <h:body>
13.         <h1>Hello from Facelets</h1>
14.     </h:body>

```

```
</html>
```

15. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF20/HelloJSF20.xhtml`.

## 8.5 Writing a Hello-world JSF 1.2 Web Application

Create a New JSF 1.2 Project

1. From "File" menu ⇒ choose "New Project..."
2. "Choose Project" ⇒ In "Categories", choose "Java Web" ⇒ In "Projects", choose "Web Application" ⇒ "Next".
3. "Name and Location" ⇒ In "Project Name", enter "HelloJSF12" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
4. "Server and settings" ⇒ choose your server, or "add" a new server ⇒ Next.
5. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 1.2" ⇒ Finish.
6. A "WelcomeJSF.jsp" page is generated, as follows:

```

7. <%@page contentType="text/html" pageEncoding="UTF-8"%>
8. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
9. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
10. <!DOCTYPE html>
11. <!--
12.     This file is an entry point for JavaServer Faces application.
13. --%>
14. <f:view>

```

```

15. <html>
16.   <head>
17.     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
18.     <title>JSP Page</title>
19.   </head>
20.   <body>
21.     <h1><h:outputText value="JavaServer Faces"/></h1>
22.   </body>
23. </html>

```

```
</f:view>
```

To run this page, right-click on the project ⇒ Run.

Create a new JSF 1.2 Page

1. Right-click on the project ⇒ New ⇒ "Other..."
2. "Choose File Type" ⇒ In "Category", select "JavaServer Faces" ⇒ In "File Type", select "JSF Page" ⇒ Next.
3. "Name and Location" ⇒ In "File Name", enter "HelloJSF12" ⇒ In "Options", check "JSP File (Standard Syntax)" ⇒ Finish.
4. In "HelloJSF12.jsp", enter the following codes:

```

5. <%@page contentType="text/html" pageEncoding="UTF-8"%>
6. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
7. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
8. <!DOCTYPE html>
9.
10. <f:view>
11.   <html>
12.     <head>
13.       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
14.       <title>Hello JSF 1.2</title>
15.     </head>
16.     <body>
17.       <h1><h:outputText value="Hello World!"/></h1>
18.     </body>
19.   </html>

```

```
</f:view>
```

20. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF12/faces/HelloJSF12.jsp`.

## 8.6 Debugging Web Applications in NetBeans

You can debug a webapp just like standalone application. For example, you can set breakpoints, single-step through the programs, etc

## UNIT: 2

**JDBC:** Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. ... It provides methods to query and update data in a database, and is oriented towards relational databases.

### Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

These steps are described in detail in the sections that follow.

## Import JDBC Packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

Additionally, depending on the features being used, Oracle-supplied JDBC packages might need to be imported. For example, the following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;

import oracle.sql.*;
```

## Load and Register the JDBC Driver

This is for establishing a communication between the JDBC program and the Oracle database. This is done by using the static `registerDriver()` method of the `DriverManager` class of the JDBC API. The following line of code does this job:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

### JDBC Driver Registration

For the entire Java application, the JDBC driver is registered only once per each database that needs to be accessed. This is true even when there are multiple database connections to the same data server.

Alternatively, the `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

However, the `forName()` method is valid for only JDK-compliant Java Virtual Machines and implicitly creates an instance of the Oracle driver, whereas the `registerDriver()` method does this explicitly.

### Connecting to a Database

Once the required packages have been imported and the Oracle JDBC driver has been loaded and registered, a database connection must be established. This is done by using the `getConnection()` method of the `DriverManager` class. A call to this method creates an object instance of the `java.sql.Connection` class. The `getConnection()` requires three input parameters, namely, a connect string, a username, and a password. The connect string should specify the JDBC driver to be used and the database instance to connect to.

The `getConnection()` method is an overloaded method that takes

- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

The following lines of code illustrate using the `getConnection()` method:

```
Connection conn = DriverManager.getConnection(URL, username, passwd);

Connection conn = DriverManager.getConnection(URL);
```

where URL, username, and passwd are of `String` data types.

We will discuss the methods of opening a connection using the Oracle JDBC OCI and thin \_drivers.

When using the OCI driver, the database can be specified using the TNSNAMES entry in the tnsnames.ora file. For example, to connect to a database on a particular host as user oratest and password oratest that has a TNSNAMES entry of oracle.world, use the following code:

```
Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:

@oracle.world", "oratest", "oratest");
```

Both the ":" and "@" are mandatory.

When using the JDBC thin driver, the TNSNAMES entry cannot be used to identify the database. There are two ways of specifying the connect string in this case, namely,

- Explicitly specifying the hostname, the TCP/IP port number, and the Oracle SID of the database to connect to. This is for thin driver only.
- Specify a Net8 keyword-value pair list.

For example, for the explicit method, use the following code to connect to a database on host `training` where the TCP/IP listener is on port 1521, the SID for the database instance is Oracle, the username and password are both oratest:

```
Connection conn = DriverManager.getConnection
```

```
        ("jdbc:oracle:thin:@training:1521:Oracle",  
  
        "oratest", "oratest");
```

For the Net8 keyword-value pair list, use the following:

```
Connection conn = DriverManager.getConnection  
  
        ("jdbc:oracle:thin@(description=(address=  
  
        (host=training) (protocol=tcp) (port=1521))  
  
        (connect_data=(sid=Oracle))) ", _"oratest", "oratest");
```

This method can also be used for the JDBC OCI driver. Just specify `oci8` instead of `thin` in the above keyword-value pair list.

## Querying the Database

Querying the database involves two steps: first, creating a statement object to perform a query, and second, executing the query and returning a resultset.

### Creating a Statement Object

This is to instantiate objects that run the query against the database connected to. This is done by the `createStatement()` method of the `conn Connection` object created above. A call to this method creates an object instance of the `Statement` class. The following line of code illustrates this:

```
Statement sql_stmt = conn.createStatement();
```

### Executing the Query and Returning a ResultSet

Once a `Statement` object has been constructed, the next step is to execute the query. This is done by using the `executeQuery()` method of the `Statement` object. A call to this method takes as parameter a SQL `SELECT` statement and returns a `JDBC ResultSet` object. The following line of code illustrates this using the `sql_stmt` object created above:

```
ResultSet rset = sql_stmt.executeQuery  
  
        ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
```

Alternatively, the SQL statement can be placed in a string and then this string passed to the `executeQuery()` function. This is shown below.

```
String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";
```

```
ResultSet rset = sql_stmt.executeQuery(sql);
```

### Statement and ResultSet Objects

Statement and ResultSet objects open a corresponding cursor in the database for SELECT and other DML statements.

The above statement executes the SELECT statement specified in between the double quotes and stores the resulting rows in an instance of the `ResultSet` object named `rset`.

### Processing the Results of a Database Query That Returns Multiple Rows

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

The first step is done using the `next()` method of the `ResultSet` object. A call to `next()` is executed in a loop to fetch the rows one row at a time, with each call to `next()` advancing the control to the next available row. The `next()` method returns the Boolean value `true` while rows are still available for fetching and returns `false` when all the rows have been fetched.

The second step is done by using the `getXXX()` methods of the JDBC `rset` object. Here `getXXX()` corresponds to the `getInt()`, `getString()` etc with `XXX` being replaced by a Java datatype.

The following code demonstrates the above steps:

```
String str;

while (rset.next())

{

    str = rset.getInt(1)+ " " + rset.getString(2)+ "

        "+rset.getFloat(3)+ " " +rset.getInt(4)+ "\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();
```

Here the 1, 2, 3, and 4 in `rset.getInt()`, `rset.getString()`, `getFloat()`, and `getInt()` respectively denote the position of the columns in the SELECT statement, that is, the first column `empno`, second column `ename`, third column `sal`, and fourth column `deptno` of the SELECT statement respectively.

### Specifying `get()` Parameters

The parameters for the `getXXX()` methods can be specified by position of the corresponding columns as numbers 1, 2, and so on, or by directly specifying the column names enclosed in double quotes, as `getString("ename")` and so on, or a combination of both.

### Closing the ResultSet and Statement

Once the `ResultSet` and `Statement` objects have been used, they must be closed explicitly. This is done by calls to the `close()` method of the `ResultSet` and `Statement` classes. The following code illustrates this:

```
rset.close();

sql_stmt.close();
```

If not closed explicitly, there are two disadvantages:

- Memory leaks can occur
- Maximum Open cursors can be exceeded

Closing the `ResultSet` and `Statement` objects frees the corresponding cursor in the database.

### Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the `close()` method of the `Connection` class.

The following line of code does this:

```
conn.close();
```

### Explicitly Close your Connection

Closing the `ResultSet` and `Statement` objects does not close the connection. The connection should be closed by explicitly invoking the `close()` method of the `Connection` class.

A complete example of the above procedures using a JDBC thin driver is given below. This program queries the `emp` table and writes the output rows to an operating system file.

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) throws SQLException, IOException

    {
```

```

//Load and register Oracle driver

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

//Establish a connection

Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

@training:1521:Oracle", "oratest", "oratest");

//Create a Statement object

Statement sql_stmt = conn.createStatement();

//Create a ResultSet object, execute the query and return a

// resultset

ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");

//Process the resultset, retrieve data in each row, column by column

//and write to an operating system file

String str = "";

while (rset.next())

{

str += rset.getInt(1)+" "+ rset.getString(2)+" "+

rset.getFloat(3)+" "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

```



```

fp.write(buf);

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

}

}

```

## Processing the Results of a Database Query That Returns a Single Row

The above sections and the complete example explained the processing of a query that returned multiple rows. This section highlights the processing of a single-row query and explains how to write code that is the analogue of the PL/SQL `exception NO_DATA_FOUND`.

### NO DATA FOUND Exception

`NO_DATA_FOUND` exception in PL/SQL is simulated in JDBC by using the return value of the `next()` method of the `ResultSet` object. A value of `false` returned by the `next()` method identifies a `NO_DATA_FOUND` exception. Consider the following code (this uses the `ResultSet` object `rset` defined in the above sections):

```

if (rset.next())

    // Process the row returned

else

    System.out.println("The Employee with Empno " + args[1] +

                       "does not exist");

```

Instead of the while loop used earlier, an if statement is used to determine whether the `SELECT` statement returned a row or not.

## Datatype Mappings

Corresponding to each SQL data type, there exist mappings to the corresponding JDBC Types, standard Java types, and the Java types provided by Oracle extensions. These are required to be used in JDBC programs that manipulate data and data structures based on these types.

There are four categories of Data types any of which can be mapped to the others. These are:

- **SQL Data types**—These are Oracle SQL data types that exist in the database.
- **JDBC Typecodes**—These are the data typecodes supported by JDBC as defined in the `java.sql.Types` class or defined by Oracle in `oracle.jdbc.driver.OracleTypes` class.
- **Java Types**—These are the standard types defined in the Java language.
- **Oracle Extension Java Types**—These are the Oracle extensions to the SQL data types and are defined in the `oracle.sql.*` class. Mapping SQL data types to the `oracle.sql.*` Java types enables storage and retrieval of SQL data without first converting into Java format thus preventing any loss of information.

Table 3.1 lists the default mappings existing between these four different types.

**Table 3.1 Standard and Oracle-specific SQL-Java Data Type Mappings**

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
<b>Standard JDBC 1.0 Types</b>			
CHAR	<code>java.sql.Types.CHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
VARCHAR2	<code>java.sql.Types.VARCHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
LONG	<code>java.sql.Types.LONGVARCHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR_</code>
NUMBER	<code>java.sql.Types.NUMERIC</code>	<code>java.math.BigDecimal</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.DECIMAL</code>	<code>java.math.BigDecimal</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.BIT</code>	<code>Boolean</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.TINYINT</code>	<code>byte</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.SMALLINT</code>	<code>short</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.INTEGER</code>	<code>int</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.BIGINT</code>	<code>long</code>	<code>oracle.sql.NUMBER</code>

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
NUMBER	java.sql.Types.REAL	float	oracle.sql.NUMBER
NUMBER	java.sql.Types.FLOAT	double	oracle.sql.NUMBER
NUMBER	java.sql.Types.DOUBLE	double	oracle.sql.NUMBER
RAW	java.sql.Types.BINARY	byte[]	oracle.sql.RAW
RAW	java.sql.Types.VARBINARY	byte[]	oracle.sql.RAW
LONGRAW	java.sql.Types.LONGVARBINARY	byte[]	oracle.sql.RAW
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE
DATE	java.sql.Types.TIME	java.sql.Time	oracle.sql.DATE
DATE	java.sql.Types.TIMESTAMP	java.sql.Timestamp	oracle.sql.DATE
<b>Standard JDBC 2.0 Types</b>			
BLOB	java.sql.Types.BLOB	java.sql.Blob	Oracle.sql.BLOB
CLOB	Java.sql.Types.CLOB	java.sql.Clob	oracle.sql.CLOB
user-defined	java.sql.Types.STRUCT	java.sql.Struct	oracle.sql.STRUCT_object
user-defined	java.sql.Types.REF	java.sql.Ref	oracle.sql.REF_reference
user-defined	java.sql.Types.ARRAY	java.sql.Array	oracle.sql.ARRAY_collection
<b>Oracle Extensions</b>			
BFILE	oracle.jdbc.driver.	n/a	OracleTypes.BFILE

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
	oracle.sql.BFILE_		
ROWID	oracle.jdbc.driver. oracle.sql.ROWID_	n/a	OracleTypes.ROWID
REFCURSOR R type	oracle.jdbc.driver. OracleTypes.CURSOR	java.sql.ResultSet	oracle.jdbc.driver._  OracleResultSet

## Exception Handling in JDBC

Like in PL/SQL programs, exceptions do occur in JDBC programs. Notice how the `NO_DATA_FOUND` exception was simulated in the earlier section "Processing the Results of a Database Query That Returns a Single Row." Exceptions in JDBC are usually of two types:

- Exceptions occurring in the JDBC driver
- Exceptions occurring in the Oracle 8i database itself

Just as PL/SQL provides for an implicit or explicit `RAISE` statement for an exception, Oracle JDBC programs have a `throw` statement that is used to inform that JDBC calls throw the SQL exceptions. This is shown below.

```
throws SQLException
```

This creates instances of the class `java.sql.SQLException` or a subclass of it.

And, like in PL/SQL, SQL exceptions in JDBC have to be handled explicitly. Similar to PL/SQL exception handling sections, Java provides a `try...catch` section that can handle all exceptions including SQL exceptions. Handling an exception can basically include retrieving the error code, error text, the SQL state, and/or printing the error stack trace.

The `SQLException` class provides methods for obtaining all of this information in case of error conditions.

### Retrieving Error Code, Error Text, and SQL State

There are the methods `getErrorCode()` and `getMessage()` similar to the functions `SQLCODE` and `SQLERRM` in PL/SQL. To retrieve the SQL state, there is the method `getSQLState()`. A brief description of these methods is given below:

- `getErrorCode()`
- This function returns the five-digit ORA number of the error in case of exceptions occurring in the JDBC driver as well as in the database.
- `getMessage()`
- This function returns the error message text in case of exceptions occurring in the JDBC driver. For exceptions occurring in the database, this function returns the error message text prefixed with the ORA number.
- `getSQLState()`
- This function returns the five digit code indicating the SQL state only for exceptions occurring in the database.

The following code illustrates the use of exception handlers in JDBC:

```
try { <JDBC code> }

catch (SQLException e) { System.out.println("ERR: "+ e.getMessage()) }
```

We now show the `QueryExample` class of the earlier section with complete exception handlers built in it. The code is as follows:

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) {

        int ret_code;

        try {

            //Load and register Oracle driver

            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

            //Establish a connection

            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

            @training:1521:Oracle", "oratest", "oratest");

            //Create a Statement object

            Statement sql_stmt = conn.createStatement();

            //Create a ResultSet object, execute the query and return a

            // resultset
```

```

ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");

//Process the resultset, retrieve data in each row, column by column

// and write to an operating system file

String str = "";

while (rset.next())

{

str += rset.getInt(1)+" "+ rset.getString(2)+" "+rset.getFloat(3)+

" "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

} catch (SQLException e) {ret_code = e.getErrorCode();

System.err.println("Oracle Error: "+ ret_code + e.getMessage());}

```

```

catch (IOException e) {System.out.println("Java Error: "+

e.getMessage()); }

}

}

```

### Printing Error Stack Trace

The `SQLException` has the method `printStackTrace()` for printing an error stack trace. This method prints the stack trace of the throwable object to the standard error stream. The following code illustrates this:

```

catch (SQLException e) { e.printStackTrace(); }

```

## JAVABEANS:

### 1. How to Install NetBeans 8.2

#### 1.1 How to Install NetBeans on Windows

Step 0: Install JDK

To use NetBeans for Java programming, you need to first install Java Development Kit (JDK). See "[JDK - How to Install](#)".

Step 1: Download

Download "NetBeans IDE" installer from <http://netbeans.org/downloads/index.html>. There are many "bundles" available. For beginners, choose the 1st entry "Java SE" (e.g., "netbeans-8.2-javase-windows.exe" 95MB).

Step 2: Run the Installer

Run the downloaded installer.

#### 1.2 How to Install NetBeans on Mac OS X

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Mac](#)".

To install NetBeans:

5. Download NetBeans from <http://netbeans.org/downloads/>. Set "Platform" to "Mac OS X". There are many "bundles" available. For beginners, choose "Java SE" (e.g., "netbeans-8.2-javase-macosx.dmg" 116MB).
6. Double-click the download Disk Image (DMG) file.
7. Double-click the "NetBeans 8.x.mpkg", and follow the instructions to install NetBeans. NetBeans will be installed under "/Applications/NetBeans".
8. Eject the Disk Image (".dmg").

You can launch NetBeans from the "Applications".

**Notes:** To uninstall NetBeans, drag the "/Applications/NetBeans" folder to trash.

#### 1.3 How to Install NetBeans on Ubuntu Linux

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Ubuntu](#)".

To install NetBeans:

5. Download NetBeans from <http://netbeans.org/downloads/>. Choose platform "Linux (x86/x64)" ⇒ "Java SE". You shall receive a sh file (e.g., "netbeans-7.x-m1-javase-linux.sh") in "~/Downloads".
6. Set the downloaded sh file to executable and run the sh file. Open a Terminal:

```
7. $ cd ~/Downloads
8. $ chmod a+x netbeans-7.x-m1-javase-linux.sh // Set to executable for all (a+x)

$ ./netbeans-7.x-m1-javase-linux.sh // Run
```

Follow the instructions to install NetBeans.

To start NetBeans, run the script "netbeans" in the NetBeans' bin directory:

```
$ cd netbeans-bin-directory
$ ./netbeans
```

## 2. Writing a Hello-world Java Program in NetBeans

Step 0: Launch NetBeans

Launch NetBeans. If the "Start Page" appears, close it by clicking the "cross" button next to the "Start Page" title.

Step 1: Create a New Project

For each Java application, you need to create a "project" to keep all the source files, classes and relevant resources.

4. From "File" menu ⇒ Choose "New Project...".
5. The "Choose Project" dialog pops up ⇒ Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ "Next".
6. The "Name and Location" dialog pops up ⇒ Under "Project Name", enter "FirstProject" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Uncheck "Use Dedicated Folder for Storing Libraries" ⇒ **Uncheck "Create Main class"** ⇒ Finish.

Step 2: Write a Hello-world Java Program

8. Right-click on "FirstProject" ⇒ New ⇒ Java Class (OR choose the "File" menu ⇒ "New File..." ⇒ Categories: "Java", File Types: "Java Class" ⇒ "Next").
9. The "Name and Location" dialog pops up ⇒ In "Class Name", enter "Hello" ⇒ Delete the content in "Package" if it is not empty ⇒ "Finish".
10. The source file "Hello.java" appears in the editor panel. Enter the following codes:

```
11. public class Hello {
12.     public static void main(String[] args) {
13.         System.out.println("Hello, world");
14.     }
}
```

Step 3: Compile & Execute

There is no need to "compile" the source code in NetBeans explicitly, as NetBeans performs the so-called *incremental compilation* (i.e., the source statement is compiled as and when it is entered).

To run the program, right-click anywhere in the source (or from the "Run" menu) ⇒ Run File. Observe the output on the output console.

**Notes:**

- You should create a NEW Java project for EACH of your Java application.
- Nonetheless, NetBeans allows you to keep more than one programs in a project, which is handy for writing toy programs (such as your tutorial exercises). To run a particular program, open and right-click on the source file ⇒ Run File.

### 2.1 Correcting Syntax Error

NetBeans performs incremented compilation, as and when a source line is entered. It marked a source line with syntax error with a RED CROSS. Point your cursor at the RED CROSS to view the error message.



You CANNOT RUN the program if there is any syntax error (marked by a RED CROSS before the filename). Correct all the syntax errors; and RUN the program.

[TODO] Diagram

HINTS: In some cases, NetBeans shows a ORANGE LIGHT-BULB (for HINTS) next to the ERROR RED-CROSS (Line 5 in the above diagram). You can click on the LIGHT-BULB to get a list of HINTS to resolve this particular error, which may or may not work!

SYNTAX WARNING: marked by a orange triangular exclamation sign. Unlike errors, warnings may or may not cause problems. Try to fix these warnings as well. But you can RUN your program with warnings.

### 3. Read the NetBeans Documentation

At a minimum, you SHOULD READ the "IDE Basics, Getting Started, Java Application", which is accessible via NetBeans's "HELP" menu  $\Rightarrow$  Help Contents. This will save you many agonizing hours trying to figure out how to do somethings later.

The "Help"  $\Rightarrow$  "Online Doc and Support" (@ <http://netbeans.org/kb/index.html>) contains many articles and tutorial on using NetBeans.

The NetBeans "Start Page" also provides many useful links to get you started.

### 4. Debugging Program in NetBeans

Step 0: Write a Java Program

The following program computes and prints the factorial of  $n$  ( $=1*2*3*\dots*n$ ). The program, however, has a logical error and produce a wrong answer for  $n=20$  ("The Factorial of 20 is -2102132736" - a negative number?!).

```
1/** Compute the factorial of n */
2public class Factorial {
3    // Print factorial of n
4    public static void main(String[] args) {
5        int n = 20;
6        int factorial = 1;
7
8        // n! = 1*2*3...*n
9        for (int i = 1; i <= n; i++) {
10            factorial *= i;
11        }
12        System.out.println("The Factorial of " + n + " is " + factorial);
13    }
14}
```

Let us use the graphic debugger to debug the program.

Step 1: Set an initial Breakpoint

A *breakpoint* suspends program execution for you to examine the internal states of the program. Before starting the debugger, you need to set at least one breakpoint to suspend the execution inside the program. Set a breakpoint at `main()` method by clicking on the *left-margin* of the line containing `main()`. A *red circle* or an inverted Triangle appears in the left-margin indicating a breakpoint is set at that line.

Step 2: Start Debugging

Right click anywhere on the source code  $\Rightarrow$  "Debug File". The program begins execution but suspends its operation at the breakpoint, i.e., the `main()` method.

As illustrated in the following diagram, the highlighted line (also pointed to by a green arrow) indicates the statement to be executed in the *next* step.

### Step 3: Step-Over and Watch the Variables and Outputs

Click the "Step Over" button (or select "Step Over" in "Debug" menu) to *single-step* thru your program. At each of the step, examine the value of the variables (in the "Variable" panel) and the outputs produced by your program (in the "Output" Panel), if any. You can also place your cursor at any variable to inspect the content of the variable.

Single-stepping thru the program and watching the values of internal variables and the outputs produced is the *ultimate* mean in debugging programs - because it is exactly how the computer runs your program!

### Step 4: Breakpoint, Run-To-Cursor, Continue and Finish

As mentioned, a breakpoint *suspends* program execution and let you examine the internal states of the program. To set a breakpoint on a particular statement, click on the left-margin of that line (or select "Toggle Breakpoint" from "Run" menu).

"Continue" resumes the program execution, up to the next breakpoint, or till the end of the program.

"Single-step" thru a loop with a large count is time-consuming. You could set a breakpoint at the statement immediately outside the loop (e.g., Line 11 of the above program), and issue "Continue" to complete the loop.

Alternatively, you can place the cursor on a particular statement, and issue "Run-To-Cursor" to resume execution up to the line.

"Finish" ends the debugging session. Always terminate your current debugging session using "Finish" or "Continue" till the end of the program.

## 4.1 Other Debugger's Features:

---

### Modify the Value of a Variable

You can modify the value of a variable by entering a new value in the "Variable" panel. This is handy for temporarily modifying the behaviour of a program, without changing the source code.

### Step-Into and Step-Out

To debug a *method*, you need to use "Step-Into" to step into the *first* statement of the method. You could use "Step-Out" to return back to the caller, anywhere within the method. Alternatively, you could set a breakpoint inside a method.

## 5. NetBeans - Tips & Tricks

---

### 5.1 General Usage

---

These are the features that I find to be most useful in NetBeans:

19. **Maximizing Window (double-click):** You can double-click on the "header" of any panel to *maximize* that particular panel, and double-click again to *restore* it back. This is particularly useful for editing source code in full panel.
20. **Code Auto-Complete (or Intelli-Sense) (ctrl-space):** Enter a partial statement (e.g., Sys) and press control-space to activate the auto-complete, which displays all the available choices.
21. **Javadoc (ctrl-space, alt-F1):** Place the cursor on a method or class, and press ctrl-space to view the javadoc; or right-click ⇒ Show Javadoc (alt-F1) to open it on a browser.
22. **Code Shorthand (tab):** For example, you can enter "sout" and press TAB for "System.out.println"; "psvm" for "public static void main(String[] args) { }" or "fori" + tab for a for-loop. To view and configure code template, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Code Templates".
23. **Formatting Source Code (alt-shift-f):** Right-click on the source (or from the "Source" menu) ⇒ Choose "Format". NetBeans will layout your source codes with the proper indents and format. To configure the formatting, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Formatting". You can also select the section of codes to be formatted, instead of the entire file.
24. **Hints for Correcting Syntax Error:** If there is a syntax error on a statement, a red mark will show up on the left-margin on that statement. You could click on the "light bulb" to display the error message, and also select from the available hints for correcting that syntax error.

25. **Rename (Refactor) (ctrl-r):** To rename a variable, place the cursor on that variable, right-click ⇒ "Refactor" ⇒ "Rename" ⇒ Enter the new name. All the appearances of that variables in the project will be renamed.
26. **Small Programs:** You can keep many small toy programs (with `main()`) in one Java project instead of create a new project for each small program. To run the desired program, on the "editor" panel ⇒ right-click ⇒ "Run File".
27. **Source Toggle Comment:** To temporarily comment-off a block of codes, choose "Source" ⇒ "Toggle Comment".
28. **Error Message Hyperlink:** Click on an error message will hyperlink to the corresponding source statement.
29. **Command-Line Arguments:** To provide command-line arguments to your Java program in NetBeans, right-click on the "project" ⇒ "Set as Main Project" ⇒ "Set Configurations" ⇒ "Customize..." ⇒ "Run" ⇒ select the "Main" class ⇒ type your command-line arguments inside the "Arguments" field ⇒ choose "Run" menu ⇒ "Run Main Project".
30. **Line Numbers:** To show the line numbers, right-click on the left-margin ⇒ "Show Line Numbers".
31. **Changing Font Face and Size:** Tools ⇒ Options ⇒ Fonts & Colors ⇒ In "Category", select "Default" ⇒ In "Font", choose the font face and size.
32. **Resetting Window View:** If you mess up the window view (e.g., you accidentally close a window and cannot find it anymore), you can reset the view via "Window" menu ⇒ "Reset Windows".
33. **Code Templates:** For example, when you create a new Java class, NetBeans retrieves the initial contents from the "Java Class" code template. To configure code templates, select "Tools" menu ⇒ "Templates" ⇒ Choose the desired template ⇒ "Open in Editor". To set a value of a variable used in the all the code templates (e.g., `$User`), select "Tools" menu ⇒ "Templates" ⇒ "Settings".
34. **Displaying Chinese Character:** Need to choose a font that support chinese character display, such as "Monospace", in Tools ⇒ Options ⇒ Fonts & Colors ⇒ Syntax ⇒ default.
35. **Changing the JDK Location:** The Netbeans configuration file is located at "etc\netbeans.conf". Edit the directive "netbeans\_jdkhome".
36. Let me know if you have more tips to be included here.

## 5.2 Java Application Development

7. **Choosing the JDK version for your program:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ You can select the JDK level of your project in pull-down menu "Source/Binary Format".
8. **Enabling JDK 7 support:** If JDK 7 is already installed in your system, right-click on your Project ⇒ "Properties" ⇒ "Source" node ⇒ "Source/Binary Format" ⇒ Select "JDK 7". Also check "Libraries" ⇒ Java Platform ⇒ JDK 7.  
If JDK 7 is not installed/configured, install JDK 7. Add JDK 7 support to NetBeans via "Tool" menu ⇒ "Java Platforms" ⇒ "Add Platform...".
9. **Choosing Default Charset:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your desired charset for the text-file I/O from the pull-down menu.
10. **Enabling Unicode Support for File Encoding:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your Unicode encoding (e.g., UTF-8, UTF-16, UTF-16LE, UTF-16GE) for the text-file I/O.
11. **To include Javadoc/Source:** Use "Library Manager" (select the "Tools" menu ⇒ "Libraries"); or "Java Platform Manager" (select "Tools" menu ⇒ "Java Platforms")
12. **Adding External JAR files & Native Libraries (".dll", ".lib", ".a", ".so"):** Many external Java packages (such as JOGL, Java3D, JAMA, etc) are available to extend the functions of JDK. These packages typically provide a "lib" directory containing JAR files (".jar") (Java Archive - a single-file package of Java classes) and native libraries (".dll", ".lib" for windows, ".a", ".so" for Linux and Mac).  
To include an external JAR file (".jar") into a project: Expand the project node ⇒ Right-click on "Libraries" ⇒ "Add JAR/Folder..." ⇒ Select the desired JAR file or the folder containing the classes. If the external package contains many JAR files, you could create a user library to contain all the JAR files, and add the library to all the projects that required these JAR files. From "Tools" menu ⇒ "Libraries" ⇒ "New Library..." ⇒ Enter a library name ⇒ Use "Add JAR/Folder..." to add JAR files into this library. Many JAR files come with native libraries in the form of ".dll", ".lib" (for Windows) and ".a", ".so" for Linux/Mac. The directory *path* of these libraries must be included in JRE's property

"`java.library.path`". This can be done via right-click the project ⇒ Set Configuration ⇒ Customize... ⇒ Run ⇒ In "VM options", enter "`-Djava.library.path=xxx`", where xxx is path of the native libraries.

**Notes:** The JAR files must be included in the CLASSPATH. The native library directories must be included in JRE's property "`java.library.path`", which normally but not necessarily includes all the paths from the PATH environment variable. Read "[External JAR files and Native Libraries](#)".

## 6. Writing Java GUI (AWT/Swing) Application in NetBeans

Step 0: Read

3. Java GUI Application Learning Trail @ <http://www.netbeans.org/kb/trails/matisse.html>.
4. Swing Tutorial's "Learning Swing with the NetBeans IDE" @ <http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html>.

Step 1: Create a New "Java Application" Project

4. Launch NetBeans ⇒ File ⇒ New Project...
5. Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ Next.
6. In "Project Name", enter "`FirstNetBeansGUI`" ⇒ Choose a suitable directory for your "Project Location" ⇒ *Uncheck* the "Create Main class" box ⇒ Finish.

Step 2: Write a Java File "JFrame Form"

10. Right-click on the project "`FirstNetBeansGUI`" ⇒ "New" ⇒ "JFrame Form..." (or "Others" ⇒ "Swing GUI Forms" ⇒ "JFrame Form").
11. In "Class Name", enter "`NetBeansSwingCounter`" ⇒ Finish.

12. Create the GUI Components visually:

- a. From the "Platte" panel ⇒ "Swing Controls" ⇒ Drag and drop a "`Label`", "`TextField`", and "`Button`" into the *design* panel.
- b. Click on the "`jLabel1`" ⇒ In the "Properties" panel, enter "Count" in "text" (You can also single-click on the `jLabel1` to change the text). Right-click on the `jLabel1` ⇒ Change Variable Name ⇒ In "New Name", enter "`lblCount`".
- c. Similarly, for "`jTextField1`" ⇒ Change the "text" to 0, and change the "Variable Name" to "`tfCount`" ⇒ Resize the text field if necessary.
- d. For "`jButton1`" ⇒ Change the "text" to "Count", and change the "Variable Name" to "`btnCount`".

13. Write the event handler for the button by double-clicking the button and enter the following codes:

```
14. private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {  
15.     count++;  
16.     tfCount.setText(count + "");  
  
    }
```

17. Create an instance variable `count` (just below the class declaration) as follows:

```
18. public class Counter extends javax.swing.JFrame {  
  
    int count = 0;
```

Step 3: Compile & Execute

Right-click the source and select "Run File".

Step 4: Study the Generated Source Code

Expand the "Generated Code" and study how the GUI builder declare, allocate and initialize the GUI Components in the  `initComponents()`. Note how the `JButton` registers an `ActionEvent` listener and how an inner class is used as the listener and provide the event handler  `actionPerformed()`. Also notice that the `main()` method uses a Swing's worker to run the GUI on the Event-Dispatcher thread, instead of the `main` thread, for thread-safe operations.

```
public class NetBeansSwingCounter extends javax.swing.JFrame {  
    int count = 0;
```

```

// Constructor to setup the UI via initComponents()
public NetBeansSwingCounter() {
    initComponents();
}

private void initComponents() {
    lblCount = new javax.swing.JLabel();
    tfCount = new javax.swing.JTextField();
    btnCount = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    lblCount.setText("Counter");
    tfCount.setText("0");

    btnCount.setText("Count");
    // Create an anonymous inner as the listener for the ActionEvent fired by btnCount
    btnCount.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            btnCountActionPerformed(evt);
        }
    });

    // Laying out the components
    // .....

    pack();
}

// ActionEvent handler for btnCount
private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {
    count++;
    tfCount.setText(count + "");
}

public static void main(String args[]) {
    // Setup the Look and Feel
    // .....

    // Run the constructor on the Event-Dispatcher Thread for thread-safe
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new NetBeansSwingCounter().setVisible(true);
        }
    });
}

// private variables
private javax.swing.JButton btnCount;
private javax.swing.JLabel lblCount;
private javax.swing.JTextField tfCount;
}

```

## 7. NetBeans and MySQL

**Reference:** "Connecting to a MySQL Database" @ <http://netbeans.org/kb/docs/ide/mysql.html>.

NetBeans (JavaEE) provides direct support to MySQL server. You can use NetBeans as a GUI client to access a MySQL server, as well as an administrative tool (e.g., starting and stopping the server).

Configuring NetBeans to Support MySQL

From NetBeans "Window" menu ⇒ Select "Services". The "Services" tab shall appear on the *left* pane

5. Right-click on the "Databases" node ⇒ "Register MySQL Server". (If you have already registered a MySQL server, you can right-click on Server node "MySQL Server at *hostname:port*" ⇒ Properties, to modify its properties.)
6. Select the "Basic Properties" tab, enter the hostname, port number, root user and password.
7. Select the "Admin Properties" tab:
  - a. Leave the "Path/URL to admin tool" empty.

- b. In "Path to start command", enter "<MYSQL\_HOME>\bin\mysqld.exe"; in the "Arguments", enter "--console"
- c. In "Path to stop command", enter "<MYSQL\_HOME>\bin\mysqladmin.exe", in the "Arguments", enter "-u root -ppassword shutdown".

8. A server node "MySQL Server at *hostname:port*" appears.

#### Database Administration - Start/Stop the Server and Create Databases

3. You can start the MySQL server by right-clicking on the server node ⇒ select "start". [There seems to be a problem here. If a "connection refused: connect" error occurs, enter the password again.]
4. Once the MySQL server is started and connected, you can see the list of databases by expanding the MySQL server node. You can create a new database by right-clicking on it and choose "Create Database...".

#### Create a new Connection

You need a connection to manipulate data. You can create multiple connections with different users and default databases.

3. Right-click on the "Databases" ⇒ "New Connection..." ⇒ Select the driver "MySQL Connector/J" ⇒ Next ⇒ Enter hostname, port number, default database, a general username and password ⇒ "Test Connection" (make sure that MySQL is started) ⇒ Finish.
4. A connection node "jdbc:mysql://*hostname:port/defaultDatabase*" appears.

#### Manipulating Data via a Connection

6. Right-click on a connection node (e.g., "jdbc:mysql://*hostname:port/defaultDatabase*") ⇒ Choose "Connect" (if not connected, provided that the MySQL server has been started).
7. You can expand the connection node to view all the databases.
8. Expand an existing database. There are three sub-nodes "Tables", "View" and "Procedures". Right-click on the "Tables" to create table or execute command. Similarly, right-click on the "View" and "Procedures".
9. To view/manipulate the records in a table, right-click on the selected table ⇒ You can choose to "View Data...", "Execute Command...", etc.
10. You can right-click on the connection to "connect" or "disconnect" from the server.

#### Create a SQL Script and Run the Script

You can create a SQL script by right-clicking on a project ⇒ New ⇒ "SQL File". You can run the script by right-clicking on the SQL script ⇒ "Run File" ⇒ Select an existing connection (or create a new connection) to run the script. You could also run a single statement (right-click on the statement ⇒ Run Statement) or a selected group of statements (highlight the statements ⇒ Right-click ⇒ Run Selection).

## 8. Developing and Deploying Web Application in NetBeans

Read:

- "Introduction to Developing Web Applications" @ <http://netbeans.org/kb/docs/web/quickstart-webapps.html>.
- More articles in "Java EE & Java Web Learning Trail" @ <http://netbeans.org/kb/trails/java-ee.html>.

### 8.1 Web (HTTP) Servers

#### Configuring Web Server

You could configure the web server via "Tools" menu ⇒ "Servers".

##### Tomcat Server

To configure Tomcat Server, select "Tools" menu ⇒ "Servers" ⇒ click "Add Servers":

6. Choose Server: Select the desired Tomcat version ⇒ Next.
7. Installation and Login Details: In "Server Location", fill in the Tomcat installation directory (\$CATALINA\_HOME) ⇒ Enter the username/password of a tomcat user with "manager" role. You could either check the "create user if it does not exist" or define the tomcat user in "\$CATALINA\_HOME\conf\tomcat-users.xml" as follows:

```

8. <tomcat-users>
9.     <role rolename="manager"/>
10.    <user username="tomcatmanager" password="xxxx" roles="manager,manager-script,admin" />

</tomcat-users>

```

Running the Web Server

Choose "Services" ⇒ Expand "Servers" node ⇒ Right-click on the desired server ⇒ Start/Stop/Restart.

## 8.2 MySQL Database Server

You can also manage the MySQL database server directly from Tomcat. Read "[NetBeans and MySQL](#)" Section.

## 8.3 Writing a Hello-World Servlet/JSP Web Application

Create a New Servlet/JSP Project

6. From "File" menu ⇒ choose "New Project...".
7. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
8. "Name and Location" ⇒ In "Project Name", enter "HelloServletJSP" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
9. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
10. "Frameworks" ⇒ Select none for pure servlet/JSP application ⇒ Finish.

Writing a Hello-World JSP

A JSP page called "index.jsp" is automatically created, which says "Hello world!". To execute this JSP, right-click on the project ⇒ "Run". The URL is `http://localhost:8080/HelloServletJSP/index.jsp`.

Writing a Hello-World Servlet

35. Right-click on the project "HelloServletJSP" ⇒ New ⇒ Servlet.
36. "Name and Location" ⇒ In "Class Name", enter "HelloServlet" ⇒ In "Package", enter "hello" ⇒ Next.
37. "Configure Servlet Deployment" ⇒ In "Servlet Name", enter "HelloServletExample" ⇒ In "URL Pattern", enter "sayhello" ⇒ Finish.
38. Enter the following codes for "HelloServlet.java":

```

39. package hello;
40.
41. import java.io.IOException;
42. import java.io.PrintWriter;
43. import javax.servlet.ServletException;
44. import javax.servlet.http.HttpServlet;
45. import javax.servlet.http.HttpServletRequest;
46. import javax.servlet.http.HttpServletResponse;
47.
48. public class HelloServlet extends HttpServlet {
49.
50.     @Override
51.     public void doGet(HttpServletRequest request, HttpServletResponse response)
52.         throws IOException, ServletException {
53.         // Set the response message's MIME type (in Content-Type response header)
54.         response.setContentType("text/html;charset=UTF-8");
55.         // Get an output Writer to write the response message over the network
56.         PrintWriter out = response.getWriter();
57.         // Write the response message (in an HTML page) to display "Hello, world!"
58.         try {
59.             out.println("<!DOCTYPE html>");
60.             out.println("<html>");
61.             out.println("<head><title>Hello Servlet</title></head>");
62.             out.println("<body><h1>Hello, World (from Java Servlet)!</h1></body>");
63.             out.println("</html>");
64.         } finally {

```

```

65.         out.close(); // Always close the output writer
66.     }
67. }
    }

```

68. To execute the servlet: Right-click on the project ⇒ run ⇒ Change the URL to `http://localhost:8080/HelloServletJSP/sayhello`.

#### Generating a WAR-file for a Web Application

A WAR (Web Archive) file is basically a zip file for distributing web application in single file. You can use WinZip or WinRAR to inspect or unzip the war file.

To distribute the project as a war-file, right-click project ⇒ "Clean and Build". The war file is created in the "dist" directory. You can deploy the web application by dropping the war-file into Tomcat's "webapps" directory. Tomcat will automatically unzip the war-file and deploy the application upon startup.

#### Debugging Web Application

The most important reason for using IDE is to use the graphic debugger for debugging the program. You can set a breakpoint in your server-side Java codes, and "Debug" a web application, similar to a standalone application.

### 8.4 Writing a Hello-world JSF 2.0 Web Application

#### Create a New JSF 2.0 Project

17. From "File" menu ⇒ choose "New Project...".
18. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
19. "Name and Location" ⇒ In "Project Name", enter "HelloJSF20" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
20. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
21. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 2.0" ⇒ Finish.
22. An "index.xhtml" JSF page is generated, as follows:

```

23. <?xml version='1.0' encoding='UTF-8' ?>
24. <!DOCTYPE html>
25. <html xmlns="http://www.w3.org/1999/xhtml"
26.       xmlns:h="http://java.sun.com/jsf/html">
27.     <h:head>
28.       <title>Facelet Title</title>
29.     </h:head>
30.     <h:body>
31.       Hello from Facelets
32.     </h:body>
    </html>

```

To run this facelet, right-click on the project ⇒ Run.

#### Create a new JSF 2.0 Facelet

16. Right-click on the project ⇒ New ⇒ "Other..."
17. "Choose File Type" ⇒ Under "Category", select "JavaServer Faces" ⇒ Under "File Type", select "JSF Page" ⇒ Next.
18. "Name and Location" ⇒ In "File Name", enter "HelloJSF20" ⇒ In "Options", check "Facelets" ⇒ Finish.
19. In "HelloJSF20.xhtml", enter the following codes:

```

20. <?xml version='1.0' encoding='UTF-8' ?>
21. <!DOCTYPE html>

```



```

22. <html xmlns="http://www.w3.org/1999/xhtml"
23.     xmlns:h="http://java.sun.com/jsf/html">
24.     <h:head>
25.         <title>Hello JSF 2.0</title>
26.     </h:head>
27.     <h:body>
28.         <h1>Hello from Facelets</h1>
29.     </h:body>

```

```
</html>
```

30. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF20/HelloJSF20.xhtml`.

## 8.5 Writing a Hello-world JSF 1.2 Web Application

Create a New JSF 1.2 Project

24. From "File" menu ⇒ choose "New Project...".
25. "Choose Project" ⇒ In "Categories", choose "Java Web" ⇒ In "Projects", choose "Web Application" ⇒ "Next".
26. "Name and Location" ⇒ In "Project Name", enter "HelloJSF12" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
27. "Server and settings" ⇒ choose your server, or "add" a new server ⇒ Next.
28. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 1.2" ⇒ Finish.
29. A "WelcomeJSF.jsp" page is generated, as follows:

```

30. <%@page contentType="text/html" pageEncoding="UTF-8"%>
31. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
32. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
33. <!DOCTYPE html>
34. <!--
35.     This file is an entry point for JavaServer Faces application.
36. --%>
37. <f:view>
38.     <html>
39.         <head>
40.             <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
41.             <title>JSP Page</title>
42.         </head>
43.         <body>
44.             <h1><h:outputText value="JavaServer Faces"/></h1>
45.         </body>
46.     </html>

```

```
</f:view>
```

To run this page, right-click on the project ⇒ Run.

Create a new JSF 1.2 Page

21. Right-click on the project ⇒ New ⇒ "Other..."
22. "Choose File Type" ⇒ In "Category", select "JavaServer Faces" ⇒ In "File Type", select "JSF Page" ⇒ Next.
23. "Name and Location" ⇒ In "File Name", enter "HelloJSF12" ⇒ In "Options", check "JSP File (Standard Syntax)" ⇒ Finish.
24. In "HelloJSF12.jsp", enter the following codes:

```

25. <%@page contentType="text/html" pageEncoding="UTF-8"%>
26. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
27. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

```

```

28. <!DOCTYPE html>
29.
30. <f:view>
31.   <html>
32.     <head>
33.       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
34.       <title>Hello JSF 1.2</title>
35.     </head>
36.     <body>
37.       <h1><h:outputText value="Hello World!"/></h1>
38.     </body>
39.   </html>

```

```
</f:view>
```

40. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF12/faces/HelloJSF12.jsp`.

## 8.6 Debugging Web Applications in NetBeans

You can debug a webapp just like standalone application. For example, you can set breakpoints, single-step through the programs, etc.

### Unit : 3 servlet

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

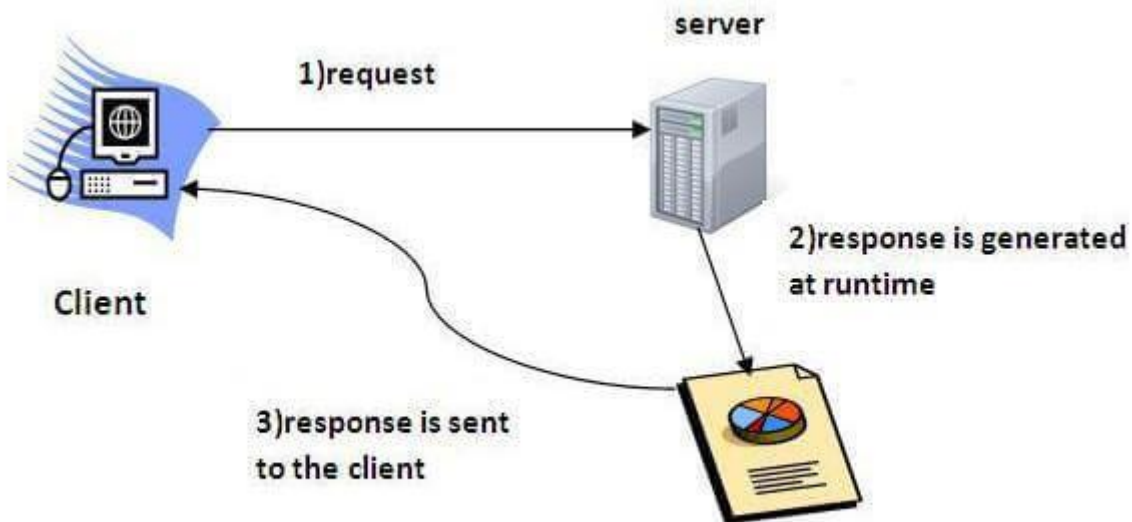
**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

## What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

## The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this –

```
public void init() throws ServletException {  
    // Initialization code...  
}
```

## The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method –

```
public void service(ServletRequest request, ServletResponse
response)
    throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

## The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
    // Servlet code
}
```

## The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

## The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

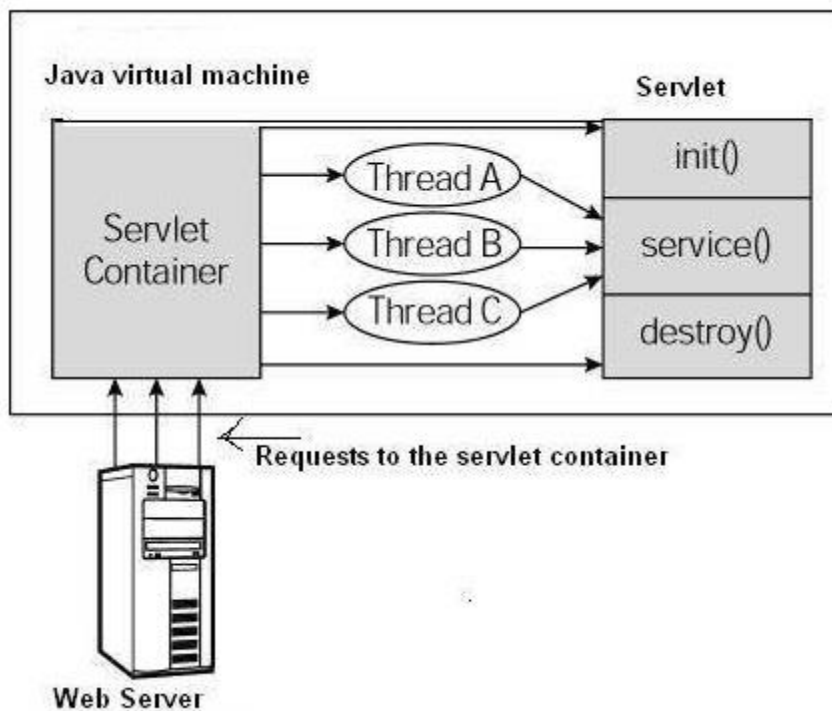
After the `destroy()` method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy() {  
    // Finalization code...  
}
```

## Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the `service()` method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the `service()` method of a single instance of the servlet.



## Six Steps to Running Your First Servlet

Once Tomcat is installed and configured, you can put it to work. Six steps take you from writing your servlet to running it. These steps are as follows:

1. Create a directory structure under Tomcat for your application.
2. Write the servlet source code. You need to import the `javax.servlet` package and the `javax.servlet.http` package in your source file.
3. Compile your source code.

4. Create a deployment descriptor.
5. Run Tomcat.
6. Call your servlet from a web browser.

## Step 1: Create a Directory Structure under Tomcat

When you install Tomcat, several subdirectories are automatically created under the Tomcat home directory (%TOMCAT\_HOME%). One of the subdirectories is webapps. The webapps directory is where you store your web applications. A *web application* is a collection of servlets and other contents installed under a specific subset of the server's URL namespace. A separate directory is dedicated for each servlet application. Therefore, the first thing to do when you build a servlet application is create an application directory. This section explains how to create a directory structure for an application called myApp.

1. Create a directory called myApp under the webapps directory. The directory name is important because this also appears in the URL to your servlet.
2. Create the src and WEB-INF directories under myApp, and create a directory named classes under WEB-INF. The directory structure is shown in [Figure 1.4](#). The src directory is for your source files, and the classes directory under WEB-INF is for your Java classes. If you have html files, you put them directly in the myApp directory. You also may want to create a directory called images under myApp for all your image files.

Note that the admin, ROOT, and examples directories are for applications created automatically when you install Tomcat.



**Figure 1.4** Tomcat application directory structure.

## Step 2: Write the Servlet Source Code

In this step, you prepare your source code. You can write the source code yourself using your favorite text editor or copy it from the CD included with the book.

The code in Listing 1.1 shows a simple servlet called TestingServlet. The file, named TestingServlet.java, sends to the browser a few HTML tags and some text. For now, don't worry if you haven't got a clue about how it works.

### Listing 1—TestingServlet.java

```
import javax.servlet.*;

import javax.servlet.http.*;
```

```

import java.io.*;

import java.util.*;

public class TestingServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,

        HttpServletResponse response)

        throws ServletException, IOException {

        PrintWriter out = response.getWriter();

        out.println("<HTML>");

        out.println("<HEAD>");

        out.println("<TITLE>Servlet Testing</TITLE>");

        out.println("</HEAD>");

        out.println("<BODY>");

        out.println("Welcome to the Servlet Testing Center");

        out.println("</BODY>");

        out.println("</HTML>");

    }

}

```

Now, save your `TestingServlet.java` file to the `src` subdirectory under `myApp`. You actually can place the source files anywhere; however, it is always a good idea to be organized by storing all your source code files in the `src` directory.

### Step 3: Compile Your Source Code

For your servlet source code to compile, you need to include in your `CLASSPATH` environment variable the path to the `servlet.jar` file. The `servlet.jar` is located in the `common\lib\` subdirectory under `%CATALINA_HOME%`.

#### NOTE

If you have forgotten how to edit the `CLASSPATH` environment variable, refer to Appendix A, "Tomcat Installation and Configuration."

If you are using Windows, remember that the new environment variable takes effect only for new console windows. In other words, after changing a new environment variable, open a new console window for typing your command lines.

Now, change directory to your working directory and type the following if you are using Windows:

```
javac -d ..\WEB-INF\classes\ TestingServlet.java
```

If you are using Linux/UNIX, the command is very similar, except that `/` is used to separate a directory from a subdirectory.

```
javac -d ../WEB-INF/classes/ TestingServlet.java
```

The `-d` option specifies where to place the generated class files. The command also assumes that you have placed the JDK's `bin` directory in the path so you can call any program in it from any directory.

### Step 4: Create the Deployment Descriptor

A *deployment descriptor* is an optional component in a servlet application, taking the form of an XML document called `web.xml`. The descriptor must be located in the `WEB-INF` directory of the servlet application. When present, the deployment descriptor contains configuration settings specific to that application. Deployment descriptors are discussed in detail in Chapter 16, "Application Deployment."



For this step, you now need to create a web.xml file and place it under the WEB-INF directory under myApp.

The web.xml for this example application must have the following content.<?xml version="1.0" encoding="ISO-8859-1"?>

```
<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>

    <servlet-name>Testing</servlet-name>

    <servlet-class>TestingServlet</servlet-class>

  </servlet>

</web-app>
```

The web.xml file has one element: web-app. You should write all your servlets under <web-app>. For each servlet, you have a <servlet> element and you need the <servlet-name> and <servlet-class> elements. The <servlet-name> is the name for your servlet, by which it is known to Tomcat. The <servlet-class> is the compiled file of your servlet without the .class extension.

Having more than one servlet in an application is common. For every servlet, you need a <servlet> element in the web.xml file. For example, the following code shows how the web.xml looks if you add another servlet called Login.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <servlet>

        <servlet-name>Testing</servlet-name>

        <servlet-class>TestingServlet</servlet-class>

    </servlet>

    <servlet>

        <servlet-name>Login</servlet-name>

        <servlet-class>LoginServlet</servlet-class>

    </servlet>

</web-app>
```

## Step 5: Run Tomcat

If it is not already running, you need to start Tomcat. For information on how to do that, see Appendix A, "Tomcat Installation and Configuration."

## Step 6: Call Your Servlet from a Web Browser

You are ready to call your servlet from a web browser. By default, Tomcat runs on port 8080 in myApp virtual directory under the servlet subdirectory. The servlet that you just wrote is named Testing. The URL for that servlet has the following format:

```
http://domain-name/virtual-directory/servlet/servlet-name
```

If you run the web browser from the same computer as Tomcat, you can replace *domain-name* with *localhost*. Therefore, the URL for your servlet would be `http://localhost:8080/myApp/servlet/Testing`.

In the deployment descriptor you wrote in Step 4, you actually mapped the servlet class file called `TestingServlet` with the name "Testing" so that your servlet can be called by specifying its class file (`TestingServlet`) or its name (`Testing`). Without a deployment descriptor, your servlet must be called by specifying its class name; that is, `TestingServlet`. This means that if you had not written a deployment descriptor in Step 4, you would have to use the following URL to call your servlet:

```
http://localhost:8080/myApp/servlet/TestingServlet
```

Typing the URL in the Address or Location box of your web browser will give you the string "Welcome to the Servlet Testing Center," as shown in [Figure 1.5](#).



**Figure 1.5** The Testing servlet.

Congratulations. You have just written your first servlet.

[yet another insignificant programming notes...](#) | [HOME](#)

## TABLE OF CONTENTS [\(HIDE\)](#)

### [1. Introduction](#)

### [2. Review of HTTP](#)

### [3. First "Hello-world" Servlet](#)

#### [3.1 Create a new Webapp "helloservlet"](#)

#### [3.2 Write a Hello-world Java Servlet - "HelloServlet.java"](#)

#### [3.3 Configure the Application Deployment Descriptor - "web.xml"](#)

#### [3.4 Run the Hello-world Servlet](#)

### [4. Processing HTML Form Data](#)

#### [4.1 Write an HTML Form](#)

#### [4.2 Write a Servlet to Process Form Data - "EchoServlet.java"](#)

#### [4.3 Configure the Servlet URL mapping in "web.xml"](#)

#### [4.4 Run the EchoServlet](#)

#### [4.5 Form-Data Submission Methods: GET | POST](#)

### [5. Request Header and Response Header](#)

#### [5.1 HttpServletRequest](#)

#### [5.2 HttpServletResponse](#)

- [6. Session Tracking](#)
  - [6.1 HttpSession](#)
  - [6.2 Example](#)
- [7. ServletConfig and ServletContext](#)
- [8. Developing and Deploying Web Applications using IDE](#)
- [9. Tomcat's Servlet Examples](#)
- [10. Database Servlet](#)
- [11. Servlet API – A Deeper Look](#)
  - [11.1 Interface Servlet](#)
  - [11.2 A Servlet's Life cycle](#)
  - [11.3 Interface ServletContext](#)
  - [11.4 Dispatch Request - RequestDispatcher](#)
  - [11.5 Filtering](#)
- [12. Web Application Deployment Descriptor "web.xml"](#)
  - [12.1 A Sample "web.xml"](#)
  - [12.2 Syntax for "web.xml"](#)
  - [12.3 Servlet Deployment Descriptor](#)
  - [12.4 Servlet Initialization Parameters](#)
  - [12.5 Application Initialization Parameters](#)
  - [12.6 Server-wide Initialization Parameters](#)
  - [12.7 Welcome Page](#)
- [13. Servlet 3.0](#)
  - [13.1 @WebServlet](#)
  - [13.2 @WebInitParam](#)
  - [13.3 @WebFilter](#)
  - [13.4 @WebListener](#)
  - [13.5 @MultipartConfig](#)

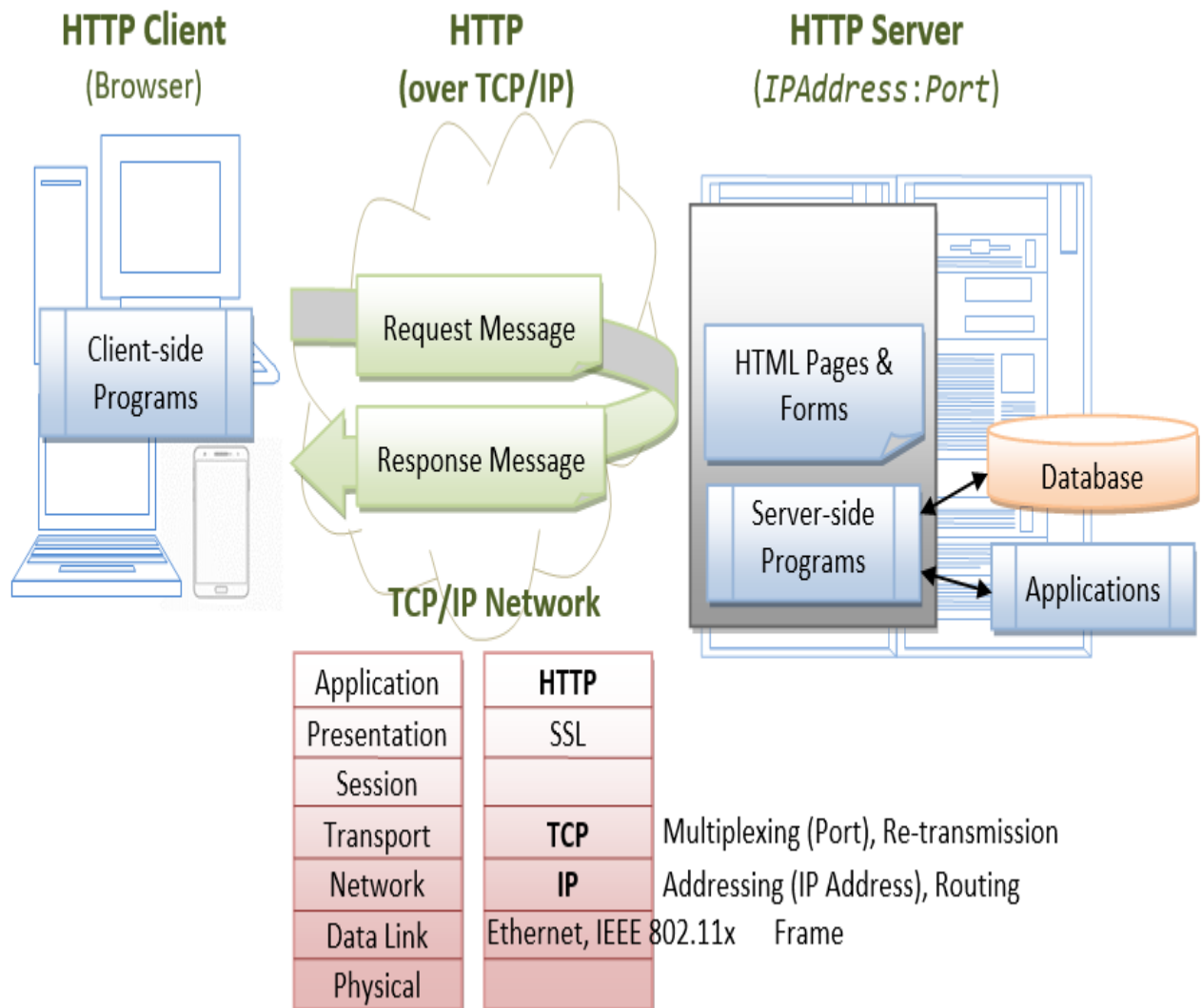
# Java Server-Side Programming

## Java Servlets

### 1. Introduction

In the early days, web servers deliver *static* contents that are indifferent to users' requests. Java servlets are *server-side programs* (running inside a web server) that handle clients' requests and return a *customized* or *dynamic response* for each request. The dynamic response could be based on user's input (e.g., search, online shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices).

Java servlets typically run on the HTTP protocol. HTTP is an *asymmetrical request-response protocol*. The client sends a *request message* to the server, and the server returns a *response message* as illustrated.



## Server-Side Technologies

There are many (competing) server-side technologies available: Java-based (servlet, JSP, JSF, Struts, Spring, Hibernate), ASP, PHP, CGI Script, and many others.

Java servlet is the *foundation* of the Java server-side technology, JSP (JavaServer Pages), JSF (JavaServer Faces), Struts, Spring, Hibernate, and others, are extensions of the servlet technology.

### Pre-requisites

HTML, Java Programming Language, HTTP and Apache Tomcat Server, SQL and MySQL Database System, and many others.

### Apache Tomcat Server

Servlets are server-side programs run inside a *Java-capable* HTTP server. Apache Tomcat Server (@ <http://tomcat.apache.org>) is the official Reference Implementation (RI) for Java servlet and JSP, provided free by open-source foundation Apache (@ <http://www.apache.org>).

You need to install Tomcat to try out Java servlets. Read "[How to Install Tomcat and Get Started Java Servlet Programming](#)".

I shall denote Tomcat's installed directory as <CATALINA\_HOME>, and assume that Tomcat server is running in port 8080.

Tomcat provides many excellent servlet examples in "<CATALINA\_HOME>\webapps\examples\servlets". You can run these examples by launching Tomcat and issuing URL <http://localhost:8080/examples>.

## Java Servlet Versions

Java Servlet has these versions: [TODO features and what is new]

- J2EE 1.2 (December 12, 1999) (**Java Servlet 2.2**, JSP 1.1, EJB 1.1, JDBC 2.0)
- J2EE 1.3 (September 24, 2001) (**Java Servlet 2.3**, JSP 1.2, EJB 2.0, JDBC 2.1)
- J2EE 1.4 (November 11, 2003) (**Java Servlet 2.4**, JSP 2.0, EJB 2.1, JDBC 3.0)
- Java EE 5 (May 11, 2006) (**Java Servlet 2.5**, JSP 2.1, JSTL 1.2, JSF 1.2, EJB 3.0, JDBC 3.0)
- Java EE 6 (December 10, 2009) (**Java Servlet 3.0**, JSP 2.2/EL 2.2, JSTL 1.2, JSF 2.0, EJB 3.1, JDBC 4.0)
- Java EE 7: expected in end of 2012.

The Java Servlets Home Page is @ <http://java.sun.com/products/servlet> (<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>). For developers, check out the Servlet Developers @ <http://java.net/projects/servlet/>.

Java Servlet is the *foundation* technology for Java server-side programming. You need to understand Servlet thoroughly before you could proceed to other Java server-side technologies such as JavaServer Pages (JSP) and JavaServer Faces (JSF).

## 2. Review of HTTP

A HTTP Servlet runs under the HTTP protocol. It is important to understanding the HTTP protocol in order to understand server-side programs (servlet, JSP, ASP, PHP, etc) running over the HTTP. Read "[HTTP Basics](#)", if needed.

In brief, HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The messages consists of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs.

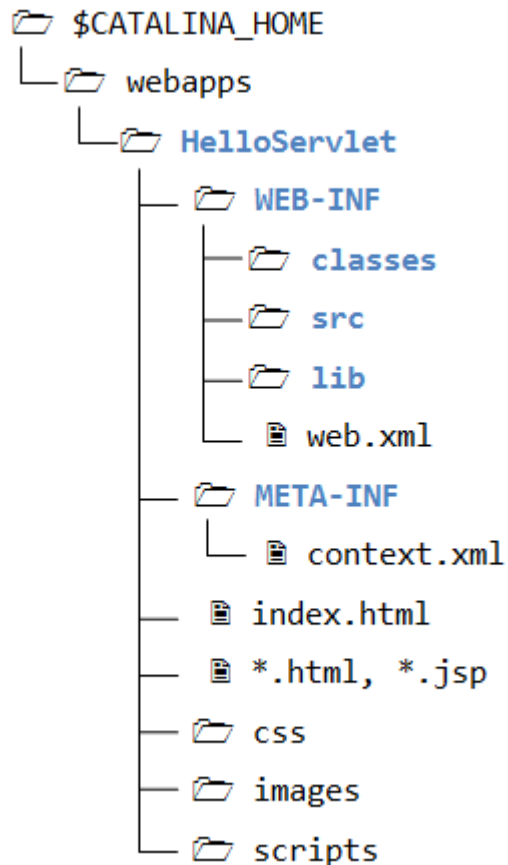
Read "[HTTP Request and Response Messages](#)" for the format, syntax of request and response messages, and examples.

## 3. First "Hello-world" Servlet

Let us begin by writing a servlet that says hello in response to a client's request. We shall use JDK and Tomcat to understand the basics, instead of IDE such as Eclipse/NetBeans. Once you understand the basics, you should use Eclipse/NetBeans to develop your webapp for better productivity.

### 3.1 Create a new Webapp "helloservlet"

We shall begin by defining a new webapp (web application) called "helloservlet" in Tomcat. A webapp, known as a *web context* in Tomcat, comprises a set of resources, such as HTML files, CSS, JavaScripts, images, programs and libraries.



A Java webapp has a *standardized directory structure* for storing various types of resources. Create a directory "helloservlet" under Tomcat's "webapps" directory (i.e., "<CATALINA\_HOME>\webapps\helloservlet", where <CATALINA\_HOME> denotes Tomcat's installed directory). Create sub-directories "WEB-INF" and "META-INF" under "helloservlet". Create sub-sub-directories "classes", "lib" and "src" under "WEB-INF". Take note that the directory names are case-sensitive.

The resources must be kept in the respective directories:

- **<CATALINA\_HOME>\webapps\helloservlet**: This directory is known as *context root* for the web context "helloservlet". It contains the resources that are *accessible by the clients*, such as HTML, CSS, Scripts and images. These resources will be delivered to the clients *as it is*. You could create sub-directories such as *images*, *css* and *scripts*, to further categories the resources.
- **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF**: This directory is NOT accessible by the clients directly. This is where you keep your application-specific configuration files (such as "web.xml"), and its sub-directories contain program classes, source files, and libraries.
  - **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF\src**: Keep the Java program source files. It is a good practice to separate the source files and classes to facilitate deployment.
  - **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF\classes**: Keep the Java classes (compiled from the source codes). Classes defined in packages must be kept according to the package directory structure.

- `<CATALINA_HOME>\webapps\helloservlet\WEB-INF\lib`: keep the JAR files provided by external packages, available to this webapp only.
- `<CATALINA_HOME>\webapps\helloservlet\META-INF`: This directory is also NOT accessible by the clients. It keeps resources and configurations (e.g., "context.xml") related to the particular server (e.g., Tomcat, Glassfish). In contrast, "WEB-INF" is for resources related to this webapp, independent of the server.

### 3.2 Write a Hello-world Java Servlet - "HelloServlet.java"

Servlets are Java programs that runs inside a Java-capable HTTP server. A user can invoke a servlet by issuing a specific URL from the browser (HTTP client). In this example, we shall write a servlet called "HelloServlet.java" and compiled into "HelloServlet.class". A client can invoke "HelloServlet.class" by issuing URL `http://hostname:port/helloServlet/sayhello` (i.e., "sayhello" relative to the webapp). A servlet shall be kept inside a Java package (instead of the default *no-name* package) for proper deployment. Let's call our package "mypkg". Create a sub-directory called "mypkg" under "WEB-INF\src". Use a programming text editor to enter the following source codes, and save as "HelloServlet.java" in "`<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg`".

```

1// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\HelloServlet.java"
2package mypkg;
3
4import java.io.*;
5import javax.servlet.*;
6import javax.servlet.http.*;
7
8public class HelloServlet extends HttpServlet {
9    @Override
10    public void doGet(HttpServletRequest request, HttpServletResponse response)
11        throws IOException, ServletException {
12        // Set the response message's MIME type
13        response.setContentType("text/html;charset=UTF-8");
14        // Allocate a output writer to write the response message into the network socket
15        PrintWriter out = response.getWriter();
16
17        // Write the response message, in an HTML page
18        try {
19            out.println("<!DOCTYPE html>");
20            out.println("<html><head>");
21            out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
22            out.println("<title>Hello, World</title></head>");
23            out.println("<body>");
24            out.println("<h1>Hello, world!</h1>"); // says Hello
25            // Echo client's request information
26            out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
27            out.println("<p>Protocol: " + request.getProtocol() + "</p>");
28            out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
29            out.println("<p>Remote Address: " + request.getRemoteAddr() + "</p>");
30            // Generate a random number upon each request
31            out.println("<p>A Random Number: <strong>" + Math.random() + "</strong></p>");
32            out.println("</body>");

```



```

33     out.println("</html>");
34 } finally {
35     out.close(); // Always close the output writer
36 }
37 }
38}

```

### Dissecting the Program:

- We define a Java class called `HelloServlet` (in Line 8). Line 2 places this class in a package called `mypkg`. Hence, we save the source file under "`mypkg`" of the "`helloservlet\WEB-INF\src`" directory, following the Java's standard package directory structure.
- We need the Servlet API library to compile this program. Servlet API is not part of JDK or Java SE (but belongs to Java EE). Tomcat provides a copy of servlet API called "`servlet-api.jar`" in "`<CATALINA_HOME>\lib`". You could copy "`servlet-api.jar`" from "`<CATALINA_HOME>\lib`" to "`<JAVA_HOME>\jre\lib\ext`" (the JDK Extension Directory), or include the Servlet JAR file in your `CLASSPATH`.
- To compile the program under JDK, we need to use the `-d` option to specify the output *destination* directory to place the compiled class in "`helloservlet\WEB-INF\classes\mypkg`" directory.

- `// Change directory to <CATALINA_HOME>\webapps\helloservlet\WEB-INF`
- `d:\...> cd <CATALINA_HOME>\webapps\helloservlet\WEB-INF`
- `// Compile the source file and place the class in the specified destination directory`

```

d:\<CATALINA_HOME>\webapps\helloservlet\WEB-INF> javac -d classes
src\mypkg\HelloServlet.java

```

The option "`-d classes`" specifies the output destination directory, relative to the current directory. The output is `<CATALINA_HOME>\webapps\helloservlet\WEB-INF\classes\mypkg\HelloServlet.class`. The compiler creates the package directory "`mypkg`" automatically.

- We don't write a servlet from scratch. Instead, we create a servlet by subclassing `javax.servlet.http.HttpServlet` (in Line 8).
- As mentioned, a servlet is invoked in response to a request URL issued by a client. Specifically, a client issues an HTTP request, the server routes the request message to the servlet for processing. The servlet returns a response message to the client.
- An HTTP request could use either GET or POST request methods, which will be processed by the servlet's `doGet()` or `doPost()` method, respectively.
- In the `HelloServlet`, we override the `doGet()` method (as denoted by the annotation `@Override`). The `doGet()` runs in response to an HTTP GET request issued by a user via an URL. `doGet()` takes two arguments, an `HttpServletRequest` object and an `HttpServletResponse` object, corresponding to the request and response messages.
- The `HttpServletRequest` object can be used to retrieve incoming HTTP *request headers* and *form data*. The `HttpServletResponse` object can be used to set the HTTP *response headers* (e.g., content-type) and the *response message body*.
- In Line 13, we set the "MIME" type of the response message to "`text/html`". The client need to know the message type in order to correctly display the data received. (Other MIME types include `text/plain`, `image/jpeg`, `video/mpeg`, `application/xml`, and many others.) In Line

15, we retrieve a `Writer` object called out for writing the response message to the client over the network. We then use the `out.println()` to print out a proper HTML page containing the message "Hello, world!". This servlet also echoes some of the clients's request information, and prints a random number for each request.

### 3.3 Configure the Application Deployment Descriptor - "web.xml"

A web user invokes a servlet, which is kept in the web server, by issuing a specific URL from the browser. In this example, we shall configure the following request URL to trigger the "HelloServlet":

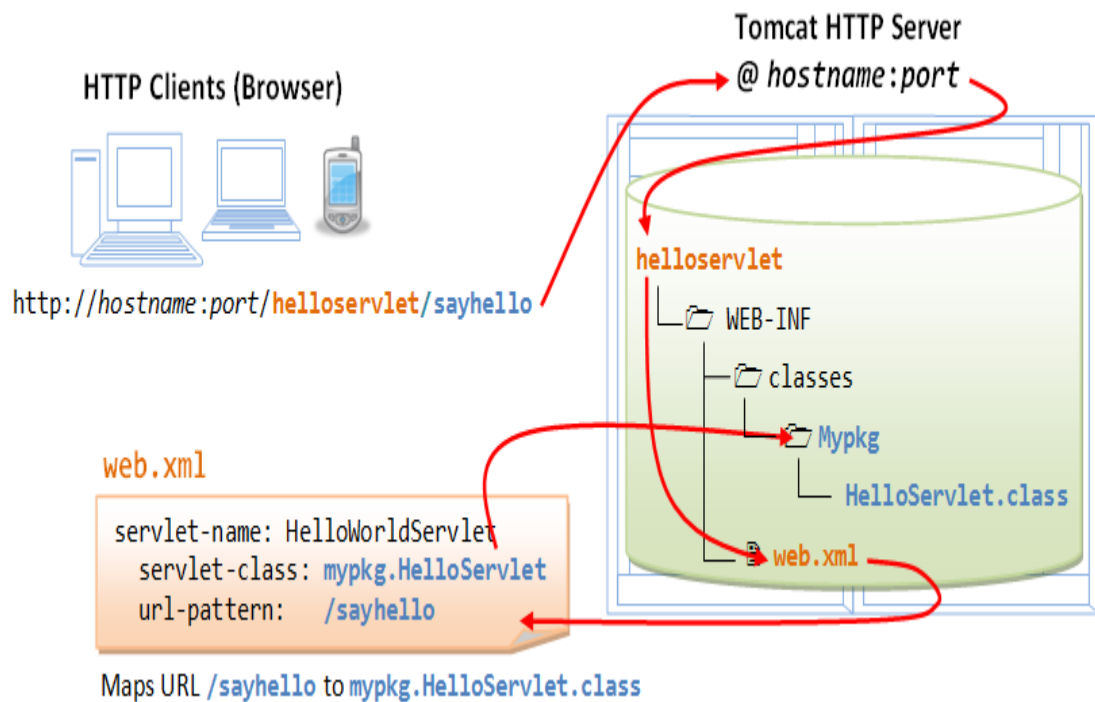
```
http://hostname:port/helloservlet/sayhello
```

Create a configuration file called "web.xml", and save it under "webapps\helloservlet\WEB-INF", as follows:

```
1<?xml version="1.0" encoding="ISO-8859-1"?>
2<web-app version="3.0"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee
6
7  <!-- To save as <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml -->
8
9  <servlet>
10    <servlet-name>HelloWorldServlet</servlet-name>
11    <servlet-class>mypkg.HelloServlet</servlet-class>
12  </servlet>
13
14  <!-- Note: All <servlet> elements MUST be grouped together and
15    placed IN FRONT of the <servlet-mapping> elements -->
16
17  <servlet-mapping>
18    <servlet-name>HelloWorldServlet</servlet-name>
19    <url-pattern>/sayhello</url-pattern>
20  </servlet-mapping>
21</web-app>
```

- The "web.xml" is called *web application deployment descriptor*. It provides the configuration options for that particular web application, such as defining the the *mapping* between URL and servlet class.
- The above configuration defines a servlet named "HelloWorldServlet", implemented in "mypkg.HelloServlet.class" (written earlier), and maps to URL "/sayhello", where "/" denotes the context root of this webapp "helloservlet". In other words, the absolute URL

for this servlet is `http://hostname:port/helloservlet/sayhello`.



- Take note that EACH servlet requires a pair of `<servlet>` and `<servlet-mapping>` elements to do the mapping, via an arbitrary but unique `<servlet-name>`. Furthermore, all the `<servlet>` elements must be grouped together and placed before the `<servlet-mapping>` elements (as specified in the XML schema).

### 3.4 Run the Hello-world Servlet

To run the servlet, first start the Tomcat server. Verify that the web context "helloservlet" has been deployed by observing the following messages in the Tomcat's console:

```
xxx x, xxxx xx:xx:xx xx org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory helloservlet
.....
```

Start a web browser (Firefox, IE or Chrome), and issue the following URL (as configured in the "web.xml"). Assume that Tomcat is running in port number 8080.

```
http://localhost:8080/helloservlet/sayhello
```

We shall see the output "Hello, world!".

# Hello, world!

Request URI: /helloservlet/sayhello

Protocol: HTTP/1.1

PathInfo: null

Remote Address: 127.0.0.1

A Random Number: **0.4320795689818858**

Try selecting "View Source" in your browser, which produces these output:

```
<!DOCTYPE html>
<html><head>
<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
<title>Hello, World</title></head>
<body>
<h1>Hello, world!</h1>
<p>Request URI: /helloservlet/sayhello</p>
<p>Protocol: HTTP/1.1</p>
<p>PathInfo: null</p>
<p>Remote Address: 127.0.0.1</p>
<p>A Random Number: <strong>0.4320795689818858</strong></p>
</body>
</html>
```

It is important to take note that users receive the *output* of the servlet. User does not receive the servlet's program codes, which are kept under a hidden directory "WEB-INF" and not directly accessible by web users.

**Everything that can possibly go wrong will go wrong...** Read "[Common Error Messages](#)". The likely errors are "404 File Not Found" and "500 Internal Server Error".

## 4. Processing HTML Form Data

### 4.1 Write an HTML Form

HTML provides a `<form>...</form>` tag, which can be used to build a user input form containing elements such as text fields, password field, radio buttons, pull-down menu, checkboxes, text area, hidden field, submit and reset buttons. This allows web users to interact with the web server by submit data. For example,

**User Input Form**

**Personal Particular**

Name:

Password:

Gender: ☐ Male ☐ Female

Age:

**Languages**

☐ Java ☐ C/C++ ☐ C#

**Instruction**

Create the following HTML script, and save as "form\_input.html" under the context root "helloservlet".

```

1<!DOCTYPE html>
2<html>
3<head>
4  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
5  <title>User Input Form</title>
6</head>
7
8<body>
9<h2>User Input Form</h2>
10<form method="get" action="echo">
11  <fieldset>
12    <legend>Personal Particular</legend>
13    Name: <input type="text" name="username" /><br /><br />
14    Password: <input type="password" name="password" /><br /><br />
15    Gender: <input type="radio" name="gender" value="m" checked />Male
16            <input type="radio" name="gender" value="f" />Female<br /><br />
17    Age: <select name = "age">
18        <option value="1">&lt; 1 year old</option>
19        <option value="99">1 to 99 years old</option>

```

```

20     <option value="100">&gt; 99 years old</option>
21 </select>
22 </fieldset>
23
24 <fieldset>
25     <legend>Languages</legend>
26     <input type="checkbox" name="language" value="java" checked />Java
27     <input type="checkbox" name="language" value="c" />C/C++
28     <input type="checkbox" name="language" value="cs" />C#
29 </fieldset>
30
31 <fieldset>
32     <legend>Instruction</legend>
33     <textarea rows="5" cols="30" name="instruction">Enter your instruction here...</tex
34 </fieldset>
35
36 <input type="hidden" name="secret" value="888" />
37 <input type="submit" value="SEND" />
38 <input type="reset" value="CLEAR" />
39</form>
40</body>
41</html>

```

Start the tomcat server. Issue the following URL to request for the HTML page:

```
http://localhost:8080/helloservlet/form_input.html
```

### Explanation

- The <fieldset>...</fieldset> tag groups related elements and displays them in a box. The <legend>...</legend> tag provides the legend for the box.
  - This HTML form (enclosed within <form>...</form>) contains the following types of input elements:
    1. Text field (<input type="text">): for web users to enter text.
    2. Radio buttons (<input type="radio">): choose any one (and possibly none).
    3. Pull-down menu (<select> and <option>): pull-down menu of options.
    4. Checkboxes (<input type="checkbox">): chose none or more.
    5. Text area (<textarea>...<textarea>): for web users to enter multi-line text. (Text field for single line only.)
    6. Hidden field (<input type="hidden">): for submitting hidden name=value pair.
    7. Submit button (<input type="submit">): user clicks this button to submit the form data to the server.
    8. Reset button (<input type="reset">): resets all the input field to their default value.
- Each of the input elements has an attribute "name", and an optional attribute "value". If an element is selected, its "name=value" pair will be submitted to the server for processing.
- The <form> start-tag also specifies the URL for submission in the action="*url*" attribute, and the request method in the method="get|post" attribute.

For example, suppose that we enter "Alan Smith" in the text field, select "male", and click the "SEND" button, we will get a "404 page not found" error (because we have yet to write the processing script). BUT observe the destination URL:

```
http://localhost:8080/helloservlet/echo?username=Alan+Smith&gender=m&....
```

Observe that:

- The URL `http://localhost:8080/helloservlet/echo` is retrieved from the attribute `action="echo"` of the `<form>` start-tag. *Relative URL* is used in this example. The *base URL* for the current page `"form_input.html"` is `http://localhost:8080/helloservlet/`. Hence, the relative URL `"echo"` resolves into `http://localhost:8080/helloservlet/echo`.
- A `'?'` follows the URL, which separates the URL and the so-called *query string* (or *query parameters*, *request parameters*) followed.
- The query string comprises the `"name=value"` pairs of the *selected* input elements (i.e., `"username=Alan+Smith"` and `"gender=m"`). The `"name=value"` pairs are separated by an `'&'`. Also take note that the blank (in `"Alan Smith"`) is replaced by a `'+'`. This is because special characters are not permitted in the URL and have to be encoded (known as *URL-encoding*). Blank is encoded as `'+'` (or `%20`). Other characters are encoded as `%xx`, where `xx` is the ASCII code in hex. For example, `'&'` as `%26`, `'?'` as `%3F`.
- Some input elements such as checkboxes may trigger multiple parameter values, e.g., `"language=java&language=c&language=cs"` if all three boxes are checked.
- HTTP provides two request methods: GET and POST. For GET request, the query parameters are appended behind the URL. For POST request, the query string is sent in the request message's body. POST request is often preferred, as users will not see the strange string in the URL and it can send an unlimited amount of data. The amount of data that can be sent via the GET request is limited by the length of the URL. The request method is specified in the `<form method="get|post" ... >` start-tag. In this tutorial, we use the GET request, so that you can inspect the query string.

## 4.2 Write a Servlet to Process Form Data - "EchoServlet.java"

The form that we have written sends its data to a server-side program having relative URL of `"echo"` (as specified in the `action="url"` attribute of the `<form>` start-tag). Let us write a servlet called `EchoServlet`, which shall be mapped to the URL `"echo"`, to process the incoming form data. The servlet simply echoes the data back to the client.

Similar to the `"HelloServlet"`, we define the `"EchoServlet"` under package `"mypkg"`, and save the source file as `"<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\EchoServlet.java"`.

```
// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\EchoServlet.java"
package mypkg;
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
```

```
public class EchoServlet extends HttpServlet {
```

```
    @Override
```

```

public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    // Set the response message's MIME type
    response.setContentType("text/html; charset=UTF-8");
    // Allocate a output writer to write the response message into the network socket
    PrintWriter out = response.getWriter();

    // Write the response message, in an HTML page
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html><head>");
        out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
        out.println("<title>Echo Servlet</title></head>");
        out.println("<body><h2>You have enter</h2>");

        // Retrieve the value of the query parameter "username" (from text field)
        String username = request.getParameter("username");
        // Get null if the parameter is missing from query string.
        // Get empty string or string of white spaces if user did not fill in
        if (username == null
            || (username = htmlFilter(username.trim())).length() == 0) {
            out.println("<p>Name: MISSING</p>");
        } else {
            out.println("<p>Name: " + username + "</p>");
        }

        // Retrieve the value of the query parameter "password" (from password field)
        String password = request.getParameter("password");
        if (password == null
            || (password = htmlFilter(password.trim())).length() == 0) {
            out.println("<p>Password: MISSING</p>");
        } else {
            out.println("<p>Password: " + password + "</p>");
        }

        // Retrieve the value of the query parameter "gender" (from radio button)
        String gender = request.getParameter("gender");
        // Get null if the parameter is missing from query string.
        if (gender == null) {
            out.println("<p>Gender: MISSING</p>");
        } else if (gender.equals("m")) {
            out.println("<p>Gender: male</p>");
        } else {
            out.println("<p>Gender: female</p>");
        }

        // Retrieve the value of the query parameter "age" (from pull-down menu)
        String age = request.getParameter("age");
        if (age == null) {

```



```

        out.println("<p>Age: MISSING</p>");
    } else if (age.equals("1")) {
        out.println("<p>Age: &lt; 1 year old</p>");
    } else if (age.equals("99")) {
        out.println("<p>Age: 1 to 99 years old</p>");
    } else {
        out.println("<p>Age: &gt; 99 years old</p>");
    }
}

// Retrieve the value of the query parameter "language" (from checkboxes).
// Multiple entries possible.
// Use getParameterValues() which returns an array of String.
String[] languages = request.getParameterValues("language");
// Get null if the parameter is missing from query string.
if (languages == null || languages.length == 0) {
    out.println("<p>Languages: NONE</p>");
} else {
    out.println("<p>Languages: ");
    for (String language : languages) {
        if (language.equals("c")) {
            out.println("C/C++ ");
        } else if (language.equals("cs")) {
            out.println("C# ");
        } else if (language.equals("java")) {
            out.println("Java ");
        }
    }
    out.println("</p>");
}

// Retrieve the value of the query parameter "instruction" (from text area)
String instruction = request.getParameter("instruction");
// Get null if the parameter is missing from query string.
if (instruction == null
    || (instruction = htmlFilter(instruction.trim())).length() == 0
    || instruction.equals("Enter your instruction here...")) {
    out.println("<p>Instruction: NONE</p>");
} else {
    out.println("<p>Instruction: " + instruction + "</p>");
}

// Retrieve the value of the query parameter "secret" (from hidden field)
String secret = request.getParameter("secret");
out.println("<p>Secret: " + secret + "</p>");

// Get all the names of request parameters
Enumeration names = request.getParameterNames();
out.println("<p>Request Parameter Names are: ");
if (names.hasMoreElements()) {

```

```

        out.print(htmlFilter(names.nextElement().toString()));
    }
    do {
        out.print(", " + htmlFilter(names.nextElement().toString()));
    } while (names.hasMoreElements());
    out.println("</p>");

    // Hyperlink "BACK" to input page
    out.println("<a href='form_input.html'>BACK</a>");

    out.println("</body></html>");
} finally {
    out.close(); // Always close the output writer
}
}

// Redirect POST request to GET request.
@Override
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
    doGet(request, response);
}

// Filter the string for special HTML characters to prevent
// command injection attack
private static String htmlFilter(String message) {
    if (message == null) return null;
    int len = message.length();
    StringBuffer result = new StringBuffer(len + 20);
    char aChar;

    for (int i = 0; i < len; ++i) {
        aChar = message.charAt(i);
        switch (aChar) {
            case '<': result.append("&lt;"); break;
            case '>': result.append("&gt;"); break;
            case '&': result.append("&amp;"); break;
            case '"': result.append("&quot;"); break;
            default: result.append(aChar);
        }
    }
    return (result.toString());
}
}
}

```

## Dissecting the Program

- The query string comprises name=value pairs. We can retrieve the query parameters from the request message (captured in `doGet()`'s argument `HttpServletRequest request`) via one of the following methods:

- `request.getParameter("paramName")`
- `// Returns the parameter value in a String.`
- `// Returns null if parameter name does not exist.`
- `// Returns the first parameter value for a multi-value parameter.`
- 
- `request.getParameterValues("paramName")`
- `// Return all the parameter values in a String[].`
- `// Return null if the parameter name does not exist.`
- 
- `request.getParameterNames()`
- 
- `// Return all the parameter names in a java.util.Enumeration, possibly empty.`

- Take note that the parameter name is case sensitive.
- We use `request.getParameter("paramName")` to retrieve the parameter value for most of the single-value input elements (such as text field, radio button, text area, etc). If the parameter is present (not null), we `trim()` the returned string to remove the leading and trailing white spaces.
- We also replace the special HTML characters (>, <, &, ") with the HTML escape sequences in the input strings, before we echo them back to the client via `out.println()`. This step is necessary to prevent the so-called *command-injection attack*, where user enters a script into the text field. The replacement is done via a static helper method `htmlFilter()`. [Rule of thumb: Any text string taken from the client and echoing back via `out.println()` needs to be filtered!]
- If the parameter could possess multiple values (e.g., checkboxes), we use `request.getParameterValues()`, which returns an array of `String` or null if the parameter does not exist.
- One of the nice features of Java servlet is that all the form data decoding (i.e., *URL-decoding*) is handled automatically. That is, '+' will be decoded to blank, %xx decoded into the corresponding character.

### 4.3 Configure the Servlet URL mapping in "web.xml"

Our `<form>`'s action attribute refers to relative URL "echo", which has to be mapped to the `EchoServlet.class` in the web application deployment descriptor file "WEB-INF/web.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

  <!-- To save as <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml -->

  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>mypkg.HelloServlet</servlet-class>
  </servlet>
```

```

<servlet>
  <servlet-name>EchoServletExample</servlet-name>
  <servlet-class>mypkg.EchoServlet</servlet-class>
</servlet>

<!-- Note: All <servlet> elements MUST be grouped together and
      placed IN FRONT of the <servlet-mapping> elements -->

<servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>/sayhello</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>EchoServletExample</servlet-name>
  <url-pattern>/echo</url-pattern>
</servlet-mapping>
</web-app>

```

#### 4.4 Run the EchoServlet

Start the Tomcat server. Issue URL `http://localhost:8080/helloservlet/form_input.html`. Fill up the form, click the submit button to trigger the servlet. Alternatively, you could issue a URL with query string.

#### 4.5 Form-Data Submission Methods: GET/POST

Two request methods, GET and POST, are available for submitting form data, to be specified in the `<form>`'s attribute `"method=GET|POST"`. GET and POST performs the same basic function. That is, gather the name-value pairs of the selected input elements, URL-encode, and pack them into a query string. However, in a GET request, the query string is appended behind the URL, separated by a `'?'`. Whereas in a POST request, the query string is kept in the request body (and not shown in the URL). The length of query string in a GET request is limited by the maximum length of URL permitted, whereas it is unlimited in a POST request. I recommend POST request for production, as it does not show the strange looking query string in the URL, even if the amount of data is limited. In this tutorial, I use GET method, so that you can inspect the query string on the URL. To try out the POST request, modify the `"form_input.html"`:

```

<form method="post" action="echo">
  .....
</form>

```

Inside the servlet, GET request is processed by the method `doGet()`, while POST request is processed by the method `doPost()`. Since they often perform identical operations, we redirect `doPost()` to `doGet()` (or vice versa), as follows:

```

public class MyServlet extends HttpServlet {
    // doGet() handles GET request
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        .....
        .....
    }
}

```

```

    }

    // doPost() handles POST request
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response); // call doGet()
    }
}

```

## 5. Request Header and Response Header

HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The request and response messages consists of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs. Read "[HTTP Request and Response Messages](#)" for the format, syntax of request and response messages.

### 5.1 *HttpServletRequest*

The request message is encapsulated in an `HttpServletRequest` object, which is passed into the `doGet()` methods. `HttpServletRequest` provides many methods for you to retrieve the headers:

- General methods: `getHeader(name)`, `getHeaders(name)`, `getHeaderNames()`.
- Specific methods: `getContentType()`, `getCookies()`, `getAuthType()`, etc.
- URL related: `getRequestURI()`, `getQueryString()`, `getProtocol()`, `getMethod()`.

Example: Read "[Request Header Example](#)".

### 5.2 *HttpServletResponse*

The response message is encapsulated in the `HttpServletResponse`, which is passed into `doGet()` by reference for receiving the servlet output.

- `setStatuscode(int statuscode)`, `sendError(int code, String message)`, `sendRedirect(url)`.
- `response.setHeader(String headerName, String headerValue)`.
- `setContentType(String mimeType)`, `setContentLength(int length)`, etc.

Example: [TODO]

## 6. Session Tracking

HTTP is a *stateless* protocol. In other words, the current request does not know what has been done in the previous requests. This creates a problem for applications that runs over many requests, such as online shopping (or shopping cart). You need to maintain a so-called *session* to pass data among the multiple requests.

You can maintain a session via one of these three approaches:

1. **Cookie:** A cookie is a small text file that is stored in the client's machine, which will be send to the server on each request. You can put your session data inside the cookie. The biggest problem in using cookie is clients may disable the cookie.
2. **URL Rewriting:** Passes data by appending a short text string at the end of every URL, e.g., `http://host/path/file.html;jsessionid=123456`. You need to rewrite all the URLs (e.g., the "action" attribute of `<form>`) to include the session data.

3. Hidden field in an HTML form: pass data by using hidden field tag (<input type="hidden" name="session" value="...." />). Again, you need to include the hidden field in all the pages.

For detailed information, read "[HTTP state and session management](#)".

## 6.1 HttpSession

Programming your own session tracking (using the above approaches) is tedious and cumbersome. Fortunately, Java Servlet API provides a session tracking facility, via an interface called `javax.servlet.http.HttpSession`. It allows servlets to:

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.
- Bind objects to sessions, allowing user information to persist across multiple user requests.

The procedure is as follows:

1. Check if a session already exists. If so, use the existing session object; otherwise, create a new session object. Servlet API automates this step via the `getSession()` method of `HttpServletRequest`:

```
2. // Retrieve the current session. Create one if not exists
3. HttpSession session = request.getSession(true);
4. HttpSession session = request.getSession(); // same as above
5.
6. // Retrieve the current session.
7. // Do not create new session if not exists but return null
```

```
HttpSession session = request.getSession(false);
```

The first statement returns the existing session if exists, and create a new `HttpSession` object otherwise. Each session is identified via a session ID. You can use `session.getID()` to retrieve the session ID string. `HttpSession`, by default, uses cookie to pass the session ID in all the client's requests within a session. If cookie is disabled, `HttpSession` switches to URL-rewriting to append the session ID behind the URL. To ensure robust session tracking, all the URLs emitted from the server-side programs should pass thru the method `response.encodeURL(url)`. If cookie is used for session tracking, `encodeURL(url)` returns the `url` unchanged. If URL-rewriting is used, `encodeURL(url)` encodes the specified `url` by including the session ID.

8. The session object maintains data in the form of *key-value* pairs. You can use `session.getAttribute(key)` to retrieve the *value* of an existing *key*, `session.setAttribute(key, value)` to store new *key-value* pair, and `session.removeAttribute(key)` to remove an existing *key-value* pair. For example,

```
9. // Allocate a shopping cart (assume to be a list of String)
10. List<String> shoppingCart = new ArrayList<>();
11. // Populate the shopping cart
12. shoppingCart.add("Item 1");
13. ....
14. // Retrieve the current session, create one if not exists
15. HttpSession session = request.getSession(true);
16. // Place the shopping cart inside the session
17. synchronized (session) { // synchronized to prevent concurrent updates
```

```
18. session.setAttribute("cart", shoppingCart);
19. }
```

```
.....
```

Any page within the session can retrieve the shopping cart:

```
// Retrieve the current session, do not create new session
HttpSession session = request.getSession(false);
if (session != null) {
    List<String> theCart = (List<String>)session.getAttribute("cart");
    if (theCart != null) { // cart exists?
        for (String item : theCart) {
            .....
        }
    }
}
```

20. You can use `session.invalidate()` to terminate and remove a session. You can use `setMaxInactiveInterval()` and `getMaxInactiveInterval()` to set and get the inactive interval from the last client request, before the server invalidate the session.

## 6.2 Example

The following servlet demonstrates the use of session, by counting the number of accesses within this session from a particular client. We also use `getSessionID()` to retrieve the session ID, `getSessionCreationTime()` and `getSessionLastAccessedTime()` to get the session creation and last accessed times.

`SessionServlet.java`

```
// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\SessionServlet.java"
package mypkg;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class SessionServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // Set the response message's MIME type
        response.setContentType("text/html;charset=UTF-8");
        // Allocate a output writer to write the response message into the network
        socket
        PrintWriter out = response.getWriter();

        // Return the existing session if there is one. Create a new session
        otherwise.
        HttpSession session = request.getSession();
        Integer accessCount;
        synchronized(session) {
```

```

        accessCount = (Integer)session.getAttribute("accessCount");
        if (accessCount == null) {
            accessCount = 0;    // autobox int to Integer
        } else {
            accessCount = new Integer(accessCount + 1);
        }
        session.setAttribute("accessCount", accessCount);
    }

    // Write the response message, in an HTML page
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><meta    http-equiv='Content-Type'    content='text/html;
charset=UTF-8'>");
        out.println("<title>Session Test Servlet</title></head><body>");
        out.println("<h2>You have access this site " + accessCount + " times in
this session.</h2>");
        out.println("<p>(Session ID is " + session.getId() + ")</p>");

        out.println("<p>(Session creation time is " +
            new Date(session.getCreationTime()) + ")</p>");
        out.println("<p>(Session last access time is " +
            new Date(session.getLastAccessedTime()) + ")</p>");
        out.println("<p>(Session max inactive interval is " +
            session.getMaxInactiveInterval() + " seconds)</p>");

        out.println("<p><a            href='"            +            request.getRequestURI()            +
"'>Refresh</a>");
        out.println("<p><a    href='" + response.encodeURL(request.getRequestURI())
+
            "'>Refresh with    URL rewriting</a>");
        out.println("</body></html>");
    } finally {
        out.close();    // Always close the output writer
    }
}
}

```

web.xml

```

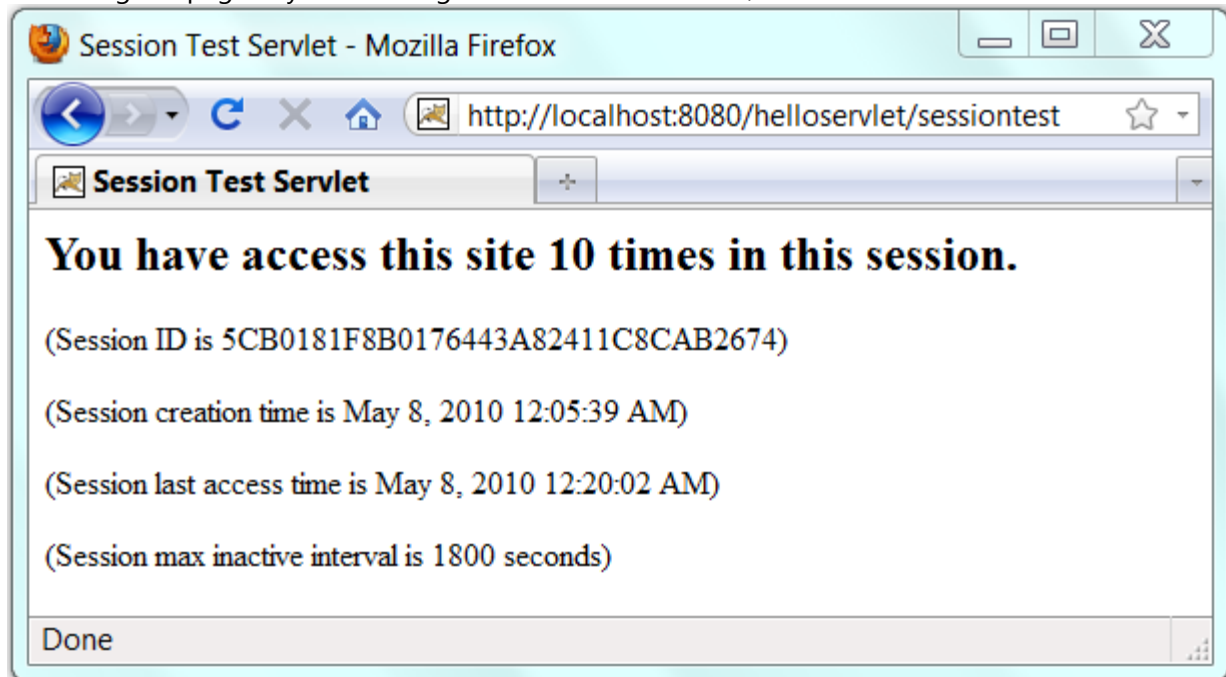
.....
<servlet>
    <servlet-name>SessionTestServlet</servlet-name>
    <servlet-class>mypkg.SessionServlet</servlet-class>
</servlet>
.....
.....
<servlet-mapping>
    <servlet-name>SessionTestServlet</servlet-name>
    <url-pattern>/sessiontest</url-pattern>
</servlet-mapping>

```

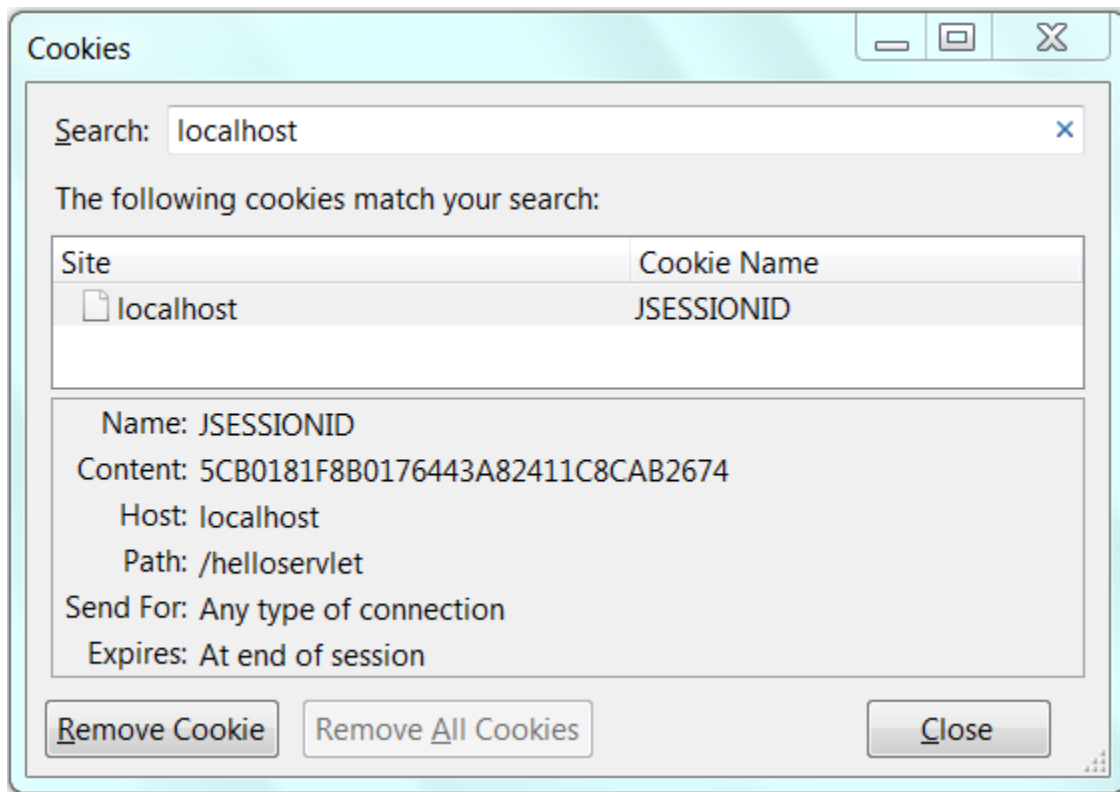


## Running the Servlet

You can use URL <http://localhost:8080/helloservlet/sessiontest> to access this servlet. Try refreshing the page. Try also closing and restart the browser, and issue the URL.



Under Firefox, a cookie named `jsessionid` is created for this session. The value of the cookie is the same as the return value of `session.getID()`. By default, Servlet API uses a cookie for managing session, but will automatically switch into URL rewriting if cookie is disabled. To ensure robust session tracking, all the URLs emitted from the server-side programs should pass thru the method `response.encodeURL(url)`. If cookie is used for session tracking, `encodeURL(url)` returns the `url` unchanged. If URL-rewriting is used, `encodeURL(url)` encodes the specified `url` by including the session ID. The session data are kept in the server, only a session ID is passed to the client.



Try disabling the cookie, and use (a) the refresh button (F5), (b) refresh and clear cache (Ctrl-F5), (c) the refresh link, and (d) the refresh with URL re-writing, to refresh the page.

## 7. ServletConfig and ServletContext

### ServletConfig

ServletConfig is a servlet configuration object used by a servlet container (e.g., Tomcat, GlassFish) to pass information to a servlet during *initialization*. It is passed as the argument in the `init()` method. The init parameters are declared in the application-specific deployment descriptor "web.xml". You can retrieve the init parameters via `ServletConfig.getInitParam("paramName")` method. For example, suppose the application's "web.xml" declares these initialization parameters about database connection:

```
<web-app ...>
...
<servlet>
...
  <init-param>
    <param-name>databaseURL</param-name>
    <param-value>jdbc:mysql://localhost:3306/ebookshop</param-value>
  </init-param>
  <init-param>
    <param-name>user</param-name>
    <param-value>myuser</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
```

```

        <param-value>xxxx</param-value>
    </init-param>
</servlet>
...
</web-app>

```

You can retrieve the init parameters in the servlet's `init()` method, as follow:

```

@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    // Read the init params and save them in web context for use by
    // servlets and JSP within this web app.
    ServletContext context = config.getServletContext();
    context.setAttribute("databaseURL", config.getInitParameter("databaseURL"));
    context.setAttribute("user", config.getInitParameter("user"));
    context.setAttribute("password", config.getInitParameter("password"));
    .....
}

```

## ServletContext

Each *webapp* is represented in a single *context* within the servlet container (such as Tomcat, Glassfish). In Servlet API, this context is defined in `javax.servlet.ServletContext` interface (a better name is probably `WebappContext`). A webapp may use many servlets. Servlets deployed in the same webapp can share information between them using the shared `ServletContext` object. There is one `ServletContext` per webapp (or web context). It can be retrieved via `ServletConfig.getServletContext()`. A servlet can use it to communicate with its servlet container (e.g., Tomcat, Glassfish), for example, to get the MIME type of a file, dispatch requests, or write to a log file. `ServletContext` has an "application" scope, and can also be used to pass information between servlets and JSPs within the same application, via methods `setAttribute("name", object)` and `getAttribute("name")`.

Example [TODO]

## 8. Developing and Deploying Web Applications using IDE

It is a lot more productive and efficient to use an IDE (such as Eclipse or NetBeans) to develop your web application. You could start/stop your servers from IDE directly. You could debug your web application in IDE, like debugging standalone application.

**NetBeans:** Read "[Developing and Deploying Web Applications in NetBeans](#)".

**Eclipse:** Read "[Developing and Deploying Web Applications in Eclipse](#)".

## 9. Tomcat's Servlet Examples

Tomcat provides a number of *excellent* servlet examples in "`<CATALINA_HOME>\webapps\examples`". The servlet source files are kept under "`<CATALINA_HOME>\webapps\examples\WEB-INF\classes`", together with the compiled classes. To run the examples, start Tomcat server and issue URL `http://localhost:8080/examples`.

I strongly encourage you to study the examples, Read "[Tomcat's Java Servlet Examples Explained](#)".

## 10. Database Servlet

Read "[Java Servlet Case Study](#)" and "[Java Servlet Case Study Continue](#)".

## 11. Servlet API – A Deeper Look

A servlet is a Java web component, managed by a servlet container (such as Apache Tomcat or Glassfish), which generates dynamic content in response to client's request. A servlet container (or servlet engine) is a web server extension which provides servlet functionality. A servlet container contains and manages servlets throughout their life cycle.

### 11.1 Interface Servlet

The Servlet interface is the central abstraction of the Java servlet API. `HttpServlet` - the most commonly servlet which handles HTTP requests, is a subclass of `GenericServlet` which implements Servlet interface.

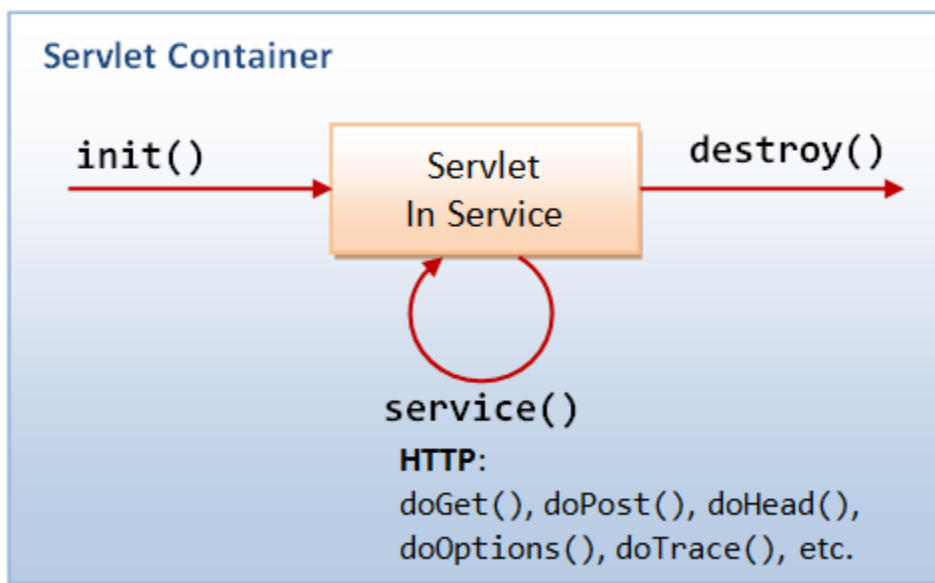
The Servlet interface declares these abstract methods:

```
// Servlet's lifecycle
void init(ServletConfig config)
void destroy()
void service(ServletRequest request, ServletResponse response)

// Servlet configuration and information
ServletConfig getServletConfig()
String getServletInfo()
```

### 11.2 A Servlet's Life cycle

A servlet's life cycle is managed via the `init()`, `service()` and `destroy()` methods.



#### Loading and Initialization

Servlet container (e.g., Tomcat or Glassfish) is responsible for loading and instantiating servlets. It may load and instantiate servlets when it is started, or delay until it determines that the servlet is needed to service a request (usually at the first request to the servlet).

The servlet container invokes the `init(ServletConfig)` method of the servlet, providing a `ServletConfig` object as an argument. `init()` runs only once. It is usually used to read persistent configuration data and initialize costly resource.

This `ServletConfig` object allows the servlet to access *initialization parameters* for this particular servlet. These parameters are defined in the *web application deployment descriptor file* (i.e., "web.xml"), under the servlet's name, as follows:

```
<servlet>
  <servlet-name>ServletName</servlet-name>
  <servlet-class>ServletClassFile</servlet-class>
  <init-param>
    <param-name>initParam1</param-name>
    <param-value>initParam1Value</param-value>
  </init-param>
  <init-param>
    <param-name>initParam2</param-name>
    <param-value>initParam2Value</param-value>
  </init-param>
</servlet>
```

The `ServletConfig` interface defines these methods to retrieve the initialization parameters for this servlet.

```
String getInitParameter(String name)
java.util.Enumeration getInitParameterNames()
```

For example,

```
public void init(ServletConfig config) throws ServletException {
    // Read all the init parameters for this servlet
    Enumeration e = config.getInitParameterNames();
    while (e.hasMoreElements()) {
        String initParamName = (String)e.nextElement();
        String initParamValue = config.getInitParameter(initParamName);
        .....
    }
}
```

The `ServletConfig` interface is implemented by `HTTPServlet` and `GenericServlet`. Hence, the `getInitParameter()` and `getInitParameterNames()` method can be called directly within `init()` or `service()`.

The `ServletConfig` also gives servlet access to a `ServletContext` object that provides information about this web context (aka web application). `ServletContext` will be discussed later.

### In Service

Once a servlet is initialized, the servlet container invokes its `service()` method to handle client requests. This method is called once for each request. Generally, the servlet container handle concurrent request to the same servlet by running `service()` on different threads (unless `SingleThreadModel` interface is declared).

For `HttpServlet`, `service()` dispatches `doGet()`, `doPost()`, `doHead()`, `doOptions()`, `doTrace()`, etc, to handle HTTP GET, POST, HEAD, OPTIONS, TRACE, etc, request respectively.

The `service()` method of an `HttpServlet` takes two arguments, an `HttpServletRequest` object and an `HttpServletResponse` object that corresponds to the HTTP request and response messages respectively.

End of Service

When the servlet container decides that a servlet should be removed from the container (e.g., shutting down the container or time-out, which is implementation-dependent), it calls the `destroy()` method to release any resource it is using and save any persistent state. Before the servlet container calls the `destroy()`, it must allow all `service()` threads to complete or time-out.

### 11.3 Interface *ServletContext*

The `ServletContext` interface defines a servlet's view of the webapp (or web context) in which it is running (a better name is actually `ApplicationContext`). Via the `ServletContext` object, a servlet can communicate with the container, e.g., write to event log, get the URL reference to resources, and get and set attributes that other servlets in the same context can access.

There is one `ServletContext` object for each web application deployed into a container. You can specify initialization parameters for a web context (that are available to all the servlet under the web context) in the web application deployment descriptor, e.g.,

```
<web-app .....>
  <context-param>
    <param-name>jdbcDriver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
  </context-param>
  <context-param>
    <param-name>databaseUrl</param-name>
    <param-value>jdbc:mysql://localhost/eshop</param-value>
  </context-param>
  .....
</web-app>
```

Servlets under this web context can access the context's initialization parameters via the `ServletConfig`'s methods:

```
// ServletConfig
String getInitParameter(String name)
java.util.Enumeration getInitParameterNames()
```

A servlet can bind an attribute of name-value pair into the `ServletContext`, which will then be available to other servlet in the same web application. The methods available are:

```
// ServletContext
Object getAttribute(String name)
void setAttribute(String name, Object value)
void removeAttribute(String name)
java.util.Enumeration getAttributeNames()
```

Other methods in `ServletContext` are:

```
// Write message to event log
```

```
void log(String message)
// Get container info
String getServerInfo()
int getMajorVersion()
int getMinorVersion()
```

The ServletContext provides direct access to static content of the web application (such as HTML, GIF files), via the following methods:

```
java.net.URL getResource(String path)
java.io.InputStream getResourceAsStream(String path)
```

## 11.4 Dispatch Request - RequestDispatcher

When building a web application, it is often useful to forward a request to another servlet, or to include the output of another servlet in the response. The RequestDispatcher interface supports these. The RequestDispatcher can be obtained via ServletContext:

```
// ServletContext
RequestDispatcher getRequestDispatcher(String servletPath)
RequestDispatcher getNamedDispatcher(String servletName)
```

Once the servlet obtained a RequestDispatcher of another servlet within the same web application, it could include or forward the request to that servlet, e.g.,

```
RequestDispatcher rd = context.getRequestDispatcher("/test.jsp?isbn=123");
rd.include(request, response);
// or
rd.forward(request, response);
```

## 11.5 Filtering

A filter is a reusable piece of code that can transform the content of HTTP requests, responses, and header information. Examples of filtering components are:

- Authentication filters
- Logging and auditing filters
- Image conversion filters
- Data compression filters
- Encryption filters
- Tokenizing filters
- Filters that trigger resource access events
- XSL/T filters that transform XML content
- MIME-type chain filters
- Caching filters

[TODO] more

## 12. Web Application Deployment Descriptor "web.xml"

---

The "web.xml" contains the web application *deployment descriptors*. Tomcat's has a system-wide (global) "web.xml" in "<CATALINA\_HOME>\conf". Each web application has its own "web.xml" in "ContextRoot\WEB-INF", which overrides the global settings. Tomcat monitors web.xml for all web applications and reloads the web application when web.xml changes, if reloadable is set to true.

## 12.1 A Sample "web.xml"

```
1<?xml version="1.0" encoding="ISO-8859-1"?>
2<web-app version="3.0"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee
6
7  <!-- General Description of the web application -->
8    <display-name>Workshop Continue</display-name>
9    <description>We shall continue our e-bookstore...</description>
10
11  <!-- Context initialization parameters -->
12  <!-- Provide the database related parameters -->
13    <context-param>
14      <param-name>jdbcDriver</param-name>
15      <param-value>com.mysql.jdbc.Driver</param-value>
16    </context-param>
17    <context-param>
18      <param-name>databaseUrl</param-name>
19      <param-value>jdbc:mysql://localhost/eshop</param-value>
20    </context-param>
21
22  <!-- Define servlets -->
23    <servlet>
24      <servlet-name>BookQuery</servlet-name>
25      <servlet-class>BookQueryServlet</servlet-class>
26      <init-param>
27        <param-name>popularAuthor</param-name>
28        <param-value>Kelvin Jones</param-value>
29      </init-param>
30    </servlet>
31
32  <!-- Define servlet's URL mapping -->
33    <servlet-mapping>
34      <servlet-name>BookQuery</servlet-name>
35      <url-pattern>/query</url-pattern>
36    </servlet-mapping>
37
38    <session-config>
39      <session-timeout>30</session-timeout>
40    </session-config>
41
42    <mime-mapping>
```



```

43     <extension>pdf</extension>
44     <mime-type>application/pdf</mime-type>
45 </mime-mapping>
46
47 <!-- For directory request -->
48 <welcome-file-list>
49     <welcome-file>index.jsp</welcome-file>
50     <welcome-file>index.html</welcome-file>
51     <welcome-file>index.htm</welcome-file>
52 </welcome-file-list>
53
54 <error-page>
55     <error-code>404</error-code>
56     <location>/404.html</location>
57 </error-page>
58</web-app>

```

## 12.2 Syntax for "web.xml"

### Servlets 3.0 "web.xml" Syntax

Tomcat 7 and Glassfish 3.1 supports Servlet 3.0.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    metadata-complete="true">
    .....
</web-app>

```

### Servlets 2.5 "web.xml" Syntax

Tomcat 6 and Glassfish 3 supports Servlets 2.5, JSP 2.1 and JSF 2.0.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    .....
</web-app>

```

### Servlets 2.4 "web.xml" Syntax

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"

```

```

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    .....
</web-app>

```

## 12.3 Servlet Deployment Descriptor

To deploy a servlet, you need to write one pair of `<servlet>` and `<servlet-mapping>` elements, with a matching (but arbitrary and unique) `<servlet-name>`. The `<servlet-class>` specifies the fully-qualified name of the servlet class. The `<url-pattern>` specifies the URL. For example,

```

<web-app ...>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servlet-class>mypkg.MyServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ServletName</servlet-name>
    <url-pattern>/MyURL</url-pattern>
  </servlet-mapping>
</web-app>

```

The resultant URL is `http://hostname:port/WebContext/MyURL`.

You can use wildcard `'*'` in the `<url-pattern>` for pattern matching. For example, `/MyURL.*` (which is matched by `/MyURL.html` and etc.), `/MyURL/*` (which is matched by `/MyURL/test`, and etc.)

Always use a custom URL for servlet, as you could choose a short and meaningful URL and include initialisation, parameters, filter, security setting in the deployment descriptor (see the next section).

## 12.4 Servlet Initialization Parameters

You can pass *initialization parameters* in the form of *name-value* pairs into a particular servlet from "web.xml". For example,

```

<web-app ...>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servlet-class>mypkg.MyServlet</servlet-class>

    <init-param>
      <param-name>debug</param-name>
      <param-value>false</param-value>
    </init-param>
    <init-param>
      <param-name>listing</param-name>
      <param-value>true</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>

```

```

    <servlet-name>ServletName</servlet-name>
    <url-pattern>/MyURL</url-pattern>
</servlet-mapping>
</web-app>

```

Inside the servlet, you can retrieve the init parameters via the `ServletConfig` object:

```

package mypkg;
public class MyServlet extends HttpServlet {

    private boolean debug = false, listing = false;

    @Override
    public void init() {
        ServletConfig config = getServletConfig();
        String strDebug = config.getInitParameter("debug");
        if (strDebug.equals("true")) debug = true;
        String strListing = config.getInitParameter("listing");
        if (strListing.equals("true")) listing = true;
    }
    .....
}

```

## 12.5 Application Initialization Parameters

Specified in webapp's "WEB-INF\web.xml", and available to all the servlets under this webapp. You can use the `getInitParameter()` method of `ServletContext` object to retrieve the init parameters.

```

<web-app .....>
    <context-param>
        <param-name>email</param-name>
        <param-value>query@abcde.com</param-value>
    </context-param>
    .....
</web-app>

```

## 12.6 Server-wide Initialization Parameters

Similar to application init parameters, but defined in the global "<CATALINA\_HOME>\conf\web.xml".

```

<context-param>
    <param-name>email</param-name>
    <param-value>query@abcde.com</param-value>
</context-param>

```

Use the `getInitParameter()` method of `ServletContext` object to retrieve the init parameters.

## 12.7 Welcome Page

Specifies the page to be displayed for request to web context root. For example,

```
<web-app ...>

.....

<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>test/index.html</welcome-file>
</welcome-file-list>
</web-app>
```

## 13. Servlet 3.0

Servlet API 3.0 introduces these annotations to simplify deployment in `javax.servlet.annotation` package:

- `@WebServlet`: Define a servlet component
- `@WebInitParam`: Define initialization parameters for a servlet
- `@WebListener`: Define a listener
- `@WebFilter`: Define a filter
- `@MultipartConfig`: For multipart file upload

For example,

```
@WebServlet(
    name = "HelloServletExample",
    urlPatterns = {"/sayhello"},
    initParams = {
        @WebInitParam(name = "param1", value = "value1"),
        @WebInitParam(name = "param2", value = "value2")}
)
public class HelloServlet extends HttpServlet { ..... }
```

The above is equivalent to the following configuration in "web.xml" prior to Servlet 3.0. The web application deployment descriptor "web.xml" has become optional in Servlet 3.0. Instead, the container at run time will process the annotations of the classes in WEB-INF/classes and JAR files in lib directory.

```
// web.xml
<servlet>
  <servlet-name>HelloServletExample</servlet-name>
  <servlet-class>hello.HelloServlet</servlet-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>value1</param-value>
  </init-param>
  <init-param>
    <param-name>param2</param-name>
    <param-value>value2</param-value>
  </init-param>
```

```

</servlet>

<servlet-mapping>
  <servlet-name>HelloServletExample</servlet-name>
  <url-pattern>/sayhello</url-pattern>
</servlet-mapping>

```

### 13.1 @WebServlet

@WebServlet defines a servlet component and its metadata, with the following attributes:

- `String[] urlPatterns`: An array of `String` declaring the `url-pattern` for `servlet-mapping`. Default is an empty array `{}`.
- `String value`: `urlPatterns`.
- `String name`: `servlet-name`, default is empty string `""`.
- `loadOnStartup`: The load-on-startup order of the servlet, default is `-1`.
- `WebInitParam[] initParams`: The init parameters of the servlet, default is an empty array `{}`.
- `boolean asyncSupported`: Declares whether the servlet supports asynchronous operation mode, default is `false`.
- `String smallIcon`, `String largeIcon`, `String description`: icon and description of the servlet.

Example:

```

@WebServlet("/sayHello")
public class Hello1Servlet extends HttpServlet { ..... }
// One URL pattern

@WebServlet(urlPatterns = {"/sayhello", "/sayhi"})
public class Hello2Servlet extends HttpServlet { ..... }
// More than one URL patterns

```

### 13.2 @WebInitParam

@WebInitParam is Used to declare init params in servlet, with the following attributes:

- `String name` and `String value` (required): Declare the name and value of the init parameter.
- `String description` (optional) description, default empty string `""`.

See the above example.

### 13.3 @WebFilter

@WebFilter defines a filter (which implements `javax.servlet.Filter` interface).

For example, the following filter log the request time for all the requests (`urlPattern="/*"`).

```

1package mypkg;
2
3import java.io.*;
4import java.util.logging.Logger;
5import javax.servlet.*;
6import javax.servlet.annotation.*;
7import javax.servlet.http.*;
8

```

```

9@WebFilter(urlPatterns={"//*"})
10public class RequestTimerFilter implements Filter {
11    private static final Logger logger
12        = Logger.getLogger(RequestTimerFilter.class.getName());
13
14    @Override
15    public void init(FilterConfig config) throws ServletException {
16        logger.info("RequestTimerFilter initialized");
17    }
18
19    @Override
20    public void doFilter(ServletRequest request, ServletResponse response,
21        FilterChain chain)
22        throws IOException, ServletException {
23        long before = System.currentTimeMillis();
24        chain.doFilter(request, response);
25        long after = System.currentTimeMillis();
26        String path = ((HttpServletRequest)request).getRequestURI();
27        logger.info(path + ": " + (after - before) + " msec");
28    }
29
30    @Override
31    public void destroy() {
32        logger.info("RequestTimerFilter destroyed");
33    }
34}

```

### 13.4 @WebListener

@WebListener defines a listener (which extends ServletContextListener, ServletRequestListener or HttpSessionListener). For example,

```

@WebListener()
public class MyContextListener extends ServletContextListener { ..... }

```

### 13.5 @MultipartConfig

For uploading file using multipart/form-data POST Request. Read "[Uploading Files in Servlet 3.0](#)".

## UNIT: 2

**JDBC:** Java Database Connectivity (**JDBC**) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. ... It provides methods to query and update data in a database, and is oriented towards relational databases.

### Fundamental Steps in JDBC

The fundamental steps involved in the process of connecting to a database and executing a query consist of the following:

- Import JDBC packages.
- Load and register the JDBC driver.
- Open a connection to the database.
- Create a statement object to perform a query.
- Execute the statement object and return a query resultset.
- Process the resultset.
- Close the resultset and statement objects.
- Close the connection.

These steps are described in detail in the sections that follow.

## Import JDBC Packages

This is for making the JDBC API classes immediately available to the application program. The following import statement should be included in the program irrespective of the JDBC driver being used:

```
import java.sql.*;
```

Additionally, depending on the features being used, Oracle-supplied JDBC packages might need to be imported. For example, the following packages might need to be imported while using the Oracle extensions to JDBC such as using advanced data types such as BLOB, and so on.

```
import oracle.jdbc.driver.*;

import oracle.sql.*;
```

## Load and Register the JDBC Driver

This is for establishing a communication between the JDBC program and the Oracle database. This is done by using the `static registerDriver()` method of the `DriverManager` class of the JDBC API. The following line of code does this job:

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

### JDBC Driver Registration

For the entire Java application, the JDBC driver is registered only once per each database that needs to be accessed. This is true even when there are multiple database connections to the same data server.

Alternatively, the `forName()` method of the `java.lang.Class` class can be used to load and register the JDBC driver:

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

However, the `forName()` method is valid for only JDK-compliant Java Virtual Machines and implicitly creates an instance of the Oracle driver, whereas the `registerDriver()` method does this explicitly.

### Connecting to a Database

Once the required packages have been imported and the Oracle JDBC driver has been loaded and registered, a database connection must be established. This is done by using the `getConnection()` method of the `DriverManager` class. A call to this method creates an object instance of the `java.sql.Connection` class. The `getConnection()` requires three input parameters, namely, a connect string, a username, and a password. The connect string should specify the JDBC driver to be used and the database instance to connect to.

The `getConnection()` method is an overloaded method that takes

- Three parameters, one each for the URL, username, and password.
- Only one parameter for the database URL. In this case, the URL contains the username and password.

The following lines of code illustrate using the `getConnection()` method:

```
Connection conn = DriverManager.getConnection(URL, username, passwd);

Connection conn = DriverManager.getConnection(URL);
```

where URL, username, and passwd are of `String` data types.

We will discuss the methods of opening a connection using the Oracle JDBC OCI and thin \_drivers.

When using the OCI driver, the database can be specified using the TNSNAMES entry in the `tnsnames.ora` file. For example, to connect to a database on a particular host as user `oratest` and password `oratest` that has a TNSNAMES entry of `oracle.world`, use the following code:

```
Connection conn = DriverManager.getConnection("jdbc:oracle:oci8:

@oracle.world", "oratest", "oratest");
```

Both the `":"` and `"@"` are mandatory.

When using the JDBC thin driver, the TNSNAMES entry cannot be used to identify the database. There are two ways of specifying the connect string in this case, namely,

- Explicitly specifying the hostname, the TCP/IP port number, and the Oracle SID of the database to connect to. This is for thin driver only.
- Specify a Net8 keyword-value pair list.

For example, for the explicit method, use the following code to connect to a database on host `training` where the TCP/IP listener is on port 1521, the SID for the database instance is `Oracle`, the username and password are both `oratest`:

```
Connection conn = DriverManager.getConnection
```



```
        ("jdbc:oracle:thin:@training:1521:Oracle",  
  
        "oratest", "oratest");
```

For the Net8 keyword-value pair list, use the following:

```
Connection conn = DriverManager.getConnection  
  
        ("jdbc:oracle:thin@(description=(address=  
  
        (host=training) (protocol=tcp) (port=1521))  
  
        (connect_data=(sid=Oracle))) ", _"oratest", "oratest");
```

This method can also be used for the JDBC OCI driver. Just specify `oci8` instead of `thin` in the above keyword-value pair list.

## Querying the Database

Querying the database involves two steps: first, creating a statement object to perform a query, and second, executing the query and returning a resultset.

### Creating a Statement Object

This is to instantiate objects that run the query against the database connected to. This is done by the `createStatement()` method of the `conn Connection` object created above. A call to this method creates an object instance of the `Statement` class. The following line of code illustrates this:

```
Statement sql_stmt = conn.createStatement();
```

### Executing the Query and Returning a ResultSet

Once a `Statement` object has been constructed, the next step is to execute the query. This is done by using the `executeQuery()` method of the `Statement` object. A call to this method takes as parameter a SQL `SELECT` statement and returns a `JDBC ResultSet` object. The following line of code illustrates this using the `sql_stmt` object created above:

```
ResultSet rset = sql_stmt.executeQuery  
  
        ("SELECT empno, ename, sal, deptno FROM emp ORDER BY ename");
```

Alternatively, the SQL statement can be placed in a string and then this string passed to the `executeQuery()` function. This is shown below.

```
String sql = "SELECT empno, ename, sal, deptno FROM emp ORDER BY ename";
```

```
ResultSet rset = sql_stmt.executeQuery(sql);
```

### Statement and ResultSet Objects

Statement and ResultSet objects open a corresponding cursor in the database for SELECT and other DML statements.

The above statement executes the SELECT statement specified in between the double quotes and stores the resulting rows in an instance of the `ResultSet` object named `rset`.

### Processing the Results of a Database Query That Returns Multiple Rows

Once the query has been executed, there are two steps to be carried out:

- Processing the output resultset to fetch the rows
- Retrieving the column values of the current row

The first step is done using the `next()` method of the `ResultSet` object. A call to `next()` is executed in a loop to fetch the rows one row at a time, with each call to `next()` advancing the control to the next available row. The `next()` method returns the Boolean value `true` while rows are still available for fetching and returns `false` when all the rows have been fetched.

The second step is done by using the `getXXX()` methods of the JDBC `rset` object. Here `getXXX()` corresponds to the `getInt()`, `getString()` etc with `XXX` being replaced by a Java datatype.

The following code demonstrates the above steps:

```
String str;

while (rset.next())

{

    str = rset.getInt(1)+ " " + rset.getString(2)+ "

        "+rset.getFloat(3)+ " " +rset.getInt(4)+ "\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();
```

Here the 1, 2, 3, and 4 in `rset.getInt()`, `rset.getString()`, `getFloat()`, and `getInt()` respectively denote the position of the columns in the SELECT statement, that is, the first column `empno`, second column `ename`, third column `sal`, and fourth column `deptno` of the SELECT statement respectively.

### Specifying `get()` Parameters

The parameters for the `getXXX()` methods can be specified by position of the corresponding columns as numbers 1, 2, and so on, or by directly specifying the column names enclosed in double quotes, as `getString("ename")` and so on, or a combination of both.

### Closing the ResultSet and Statement

Once the `ResultSet` and `Statement` objects have been used, they must be closed explicitly. This is done by calls to the `close()` method of the `ResultSet` and `Statement` classes. The following code illustrates this:

```
rset.close();

sql_stmt.close();
```

If not closed explicitly, there are two disadvantages:

- Memory leaks can occur
- Maximum Open cursors can be exceeded

Closing the `ResultSet` and `Statement` objects frees the corresponding cursor in the database.

### Closing the Connection

The last step is to close the database connection opened in the beginning after importing the packages and loading the JDBC drivers. This is done by a call to the `close()` method of the `Connection` class.

The following line of code does this:

```
conn.close();
```

### Explicitly Close your Connection

Closing the `ResultSet` and `Statement` objects does not close the connection. The connection should be closed by explicitly invoking the `close()` method of the `Connection` class.

A complete example of the above procedures using a JDBC thin driver is given below. This program queries the `emp` table and writes the output rows to an operating system file.

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) throws SQLException, IOException

    {
```

```

//Load and register Oracle driver

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

//Establish a connection

Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

@training:1521:Oracle", "oratest", "oratest");

//Create a Statement object

Statement sql_stmt = conn.createStatement();

//Create a ResultSet object, execute the query and return a

// resultset

ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");

//Process the resultset, retrieve data in each row, column by column

//and write to an operating system file

String str = "";

while (rset.next())

{

str += rset.getInt(1)+" "+ rset.getString(2)+" "+

rset.getFloat(3)+" "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

```

```

fp.write(buf);

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

}

}

```

## Processing the Results of a Database Query That Returns a Single Row

The above sections and the complete example explained the processing of a query that returned multiple rows. This section highlights the processing of a single-row query and explains how to write code that is the analogue of the PL/SQL `exception NO_DATA_FOUND`.

### NO DATA FOUND Exception

`NO_DATA_FOUND` exception in PL/SQL is simulated in JDBC by using the return value of the `next()` method of the `ResultSet` object. A value of `false` returned by the `next()` method identifies a `NO_DATA_FOUND` exception. Consider the following code (this uses the `ResultSet` object `rset` defined in the above sections):

```

if (rset.next())

    // Process the row returned

else

    System.out.println("The Employee with Empno " + args[1] +

                       "does not exist");

```

Instead of the while loop used earlier, an if statement is used to determine whether the `SELECT` statement returned a row or not.

## Datatype Mappings

Corresponding to each SQL data type, there exist mappings to the corresponding JDBC Types, standard Java types, and the Java types provided by Oracle extensions. These are required to be used in JDBC programs that manipulate data and data structures based on these types.

There are four categories of Data types any of which can be mapped to the others. These are:

- **SQL Data types**—These are Oracle SQL data types that exist in the database.
- **JDBC Typecodes**—These are the data typecodes supported by JDBC as defined in the `java.sql.Types` class or defined by Oracle in `oracle.jdbc.driver.OracleTypes` class.
- **Java Types**—These are the standard types defined in the Java language.
- **Oracle Extension Java Types**—These are the Oracle extensions to the SQL data types and are defined in the `oracle.sql.*` class. Mapping SQL data types to the `oracle.sql.*` Java types enables storage and retrieval of SQL data without first converting into Java format thus preventing any loss of information.

Table 3.1 lists the default mappings existing between these four different types.

**Table 3.1 Standard and Oracle-specific SQL-Java Data Type Mappings**

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
<b>Standard JDBC 1.0 Types</b>			
CHAR	<code>java.sql.Types.CHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
VARCHAR2	<code>java.sql.Types.VARCHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR</code>
LONG	<code>java.sql.Types.LONGVARCHAR</code>	<code>java.lang.String</code>	<code>oracle.sql.CHAR_</code>
NUMBER	<code>java.sql.Types.NUMERIC</code>	<code>java.math.BigDecimal</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.DECIMAL</code>	<code>java.math.BigDecimal</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.BIT</code>	<code>Boolean</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.TINYINT</code>	<code>byte</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.SMALLINT</code>	<code>short</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.INTEGER</code>	<code>int</code>	<code>oracle.sql.NUMBER</code>
NUMBER	<code>java.sql.Types.BIGINT</code>	<code>long</code>	<code>oracle.sql.NUMBER</code>

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
NUMBER	java.sql.Types.REAL	float	oracle.sql.NUMBER
NUMBER	java.sql.Types.FLOAT	double	oracle.sql.NUMBER
NUMBER	java.sql.Types.DOUBLE	double	oracle.sql.NUMBER
RAW	java.sql.Types.BINARY	byte[]	oracle.sql.RAW
RAW	java.sql.Types.VARBINARY	byte[]	oracle.sql.RAW
LONGRAW	java.sql.Types.LONGVARBINARY	byte[]	oracle.sql.RAW
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE
DATE	java.sql.Types.TIME	java.sql.Time	oracle.sql.DATE
DATE	java.sql.Types.TIMESTAMP	java.sql.Timestamp	oracle.sql.DATE
<b>Standard JDBC 2.0 Types</b>			
BLOB	java.sql.Types.BLOB	java.sql.Blob	Oracle.sql.BLOB
CLOB	Java.sql.Types.CLOB	java.sql.Clob	oracle.sql.CLOB
user-defined	java.sql.Types.STRUCT	java.sql.Struct	oracle.sql.STRUCT_object
user-defined	java.sql.Types.REF	java.sql.Ref	oracle.sql.REF_reference
user-defined	java.sql.Types.ARRAY	java.sql.Array	oracle.sql.ARRAY_collection
<b>Oracle Extensions</b>			
BFILE	oracle.jdbc.driver.	n/a	OracleTypes.BFILE

SQL Data types	JDBC Type codes	Standard Java Types	Oracle Extension Java _ Types
	oracle.sql.BFILE_		
ROWID	oracle.jdbc.driver. oracle.sql.ROWID_	n/a	OracleTypes.ROWID
REFCURSOR R type	oracle.jdbc.driver. OracleTypes.CURSOR	java.sql.ResultSet	oracle.jdbc.driver._  OracleResultSet

## Exception Handling in JDBC

Like in PL/SQL programs, exceptions do occur in JDBC programs. Notice how the `NO_DATA_FOUND` exception was simulated in the earlier section "Processing the Results of a Database Query That Returns a Single Row." Exceptions in JDBC are usually of two types:

- Exceptions occurring in the JDBC driver
- Exceptions occurring in the Oracle 8i database itself

Just as PL/SQL provides for an implicit or explicit `RAISE` statement for an exception, Oracle JDBC programs have a `throw` statement that is used to inform that JDBC calls throw the SQL exceptions. This is shown below.

```
throws SQLException
```

This creates instances of the class `java.sql.SQLException` or a subclass of it.

And, like in PL/SQL, SQL exceptions in JDBC have to be handled explicitly. Similar to PL/SQL exception handling sections, Java provides a `try...catch` section that can handle all exceptions including SQL exceptions. Handling an exception can basically include retrieving the error code, error text, the SQL state, and/or printing the error stack trace.

The `SQLException` class provides methods for obtaining all of this information in case of error conditions.

### Retrieving Error Code, Error Text, and SQL State

There are the methods `getErrorCode()` and `getMessage()` similar to the functions `SQLCODE` and `SQLERRM` in PL/SQL. To retrieve the SQL state, there is the method `getSQLState()`. A brief description of these methods is given below:

- `getErrorCode()`
- This function returns the five-digit ORA number of the error in case of exceptions occurring in the JDBC driver as well as in the database.
- `getMessage()`
- This function returns the error message text in case of exceptions occurring in the JDBC driver. For exceptions occurring in the database, this function returns the error message text prefixed with the ORA number.
- `getSQLState()`
- This function returns the five digit code indicating the SQL state only for exceptions occurring in the database.

The following code illustrates the use of exception handlers in JDBC:



```
try { <JDBC code> }

catch (SQLException e) { System.out.println("ERR: "+ e.getMessage()) }
```

We now show the `QueryExample` class of the earlier section with complete exception handlers built in it. The code is as follows:

```
//Import JDBC package

import java.sql.*;

// Import Java package for File I/O

import java.io.*;

public class QueryExample {

    public static void main (String[] args) {

        int ret_code;

        try {

            //Load and register Oracle driver

            DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

            //Establish a connection

            Connection conn = DriverManager.getConnection("jdbc:oracle:thin:

            @training:1521:Oracle", "oratest", "oratest");

            //Create a Statement object

            Statement sql_stmt = conn.createStatement();

            //Create a ResultSet object, execute the query and return a

            // resultset
```

```

ResultSet rset = sql_stmt.executeQuery("SELECT empno, ename, sal,

deptno FROM emp ORDER BY ename");

//Process the resultset, retrieve data in each row, column by column

// and write to an operating system file

String str = "";

while (rset.next())

{

str += rset.getInt(1)+" "+ rset.getString(2)+" "+rset.getFloat(3)+

" "+rset.getInt(4)+"\n";

}

byte buf[] = str.getBytes();

OutputStream fp = new FileOutputStream("query1.lst");

fp.write(buf);

fp.close();

//Close the ResultSet and Statement

rset.close();

sql_stmt.close();

//Close the database connection

conn.close();

} catch (SQLException e) {ret_code = e.getErrorCode();

System.err.println("Oracle Error: "+ ret_code + e.getMessage());}

```

```

catch (IOException e) {System.out.println("Java Error: "+

e.getMessage()); }

}

}

```

### Printing Error Stack Trace

The `SQLException` has the method `printStackTrace()` for printing an error stack trace. This method prints the stack trace of the throwable object to the standard error stream. The following code illustrates this:

```

catch (SQLException e) { e.printStackTrace(); }

```

## JAVABEANS:

### 1. How to Install NetBeans 8.2

#### 1.1 How to Install NetBeans on Windows

Step 0: Install JDK

To use NetBeans for Java programming, you need to first install Java Development Kit (JDK). See "[JDK - How to Install](#)".

Step 1: Download

Download "NetBeans IDE" installer from <http://netbeans.org/downloads/index.html>. There are many "bundles" available. For beginners, choose the 1st entry "Java SE" (e.g., "netbeans-8.2-javase-windows.exe" 95MB).

Step 2: Run the Installer

Run the downloaded installer.

#### 1.2 How to Install NetBeans on Mac OS X

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Mac](#)".

To install NetBeans:

9. Download NetBeans from <http://netbeans.org/downloads/>. Set "Platform" to "Mac OS X". There are many "bundles" available. For beginners, choose "Java SE" (e.g., "netbeans-8.2-javase-macosx.dmg" 116MB).
10. Double-click the download Disk Image (DMG) file.
11. Double-click the "NetBeans 8.x.mpkg", and follow the instructions to install NetBeans. NetBeans will be installed under "/Applications/NetBeans".
12. Eject the Disk Image (".dmg").

You can launch NetBeans from the "Applications".

**Notes:** To uninstall NetBeans, drag the "/Applications/NetBeans" folder to trash.

#### 1.3 How to Install NetBeans on Ubuntu Linux

To use NetBeans for Java programming, you need to first install JDK. Read "[How to install JDK on Ubuntu](#)".

To install NetBeans:

- Download NetBeans from <http://netbeans.org/downloads/>. Choose platform "Linux (x86/x64)" ⇒ "Java SE". You shall receive a sh file (e.g., "netbeans-7.x-m1-javase-linux.sh") in "~/Downloads".
- Set the downloaded sh file to executable and run the sh file. Open a Terminal:

```
11. $ cd ~/Downloads
12. $ chmod a+x netbeans-7.x-m1-javase-linux.sh // Set to executable for all (a+x)

$ ./netbeans-7.x-m1-javase-linux.sh // Run
```

Follow the instructions to install NetBeans.

To start NetBeans, run the script "netbeans" in the NetBeans' bin directory:

```
$ cd netbeans-bin-directory
$ ./netbeans
```

## 2. Writing a Hello-world Java Program in NetBeans

Step 0: Launch NetBeans

Launch NetBeans. If the "Start Page" appears, close it by clicking the "cross" button next to the "Start Page" title.

Step 1: Create a New Project

For each Java application, you need to create a "project" to keep all the source files, classes and relevant resources.

- From "File" menu ⇒ Choose "New Project...".
- The "Choose Project" dialog pops up ⇒ Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ "Next".
- The "Name and Location" dialog pops up ⇒ Under "Project Name", enter "FirstProject" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Uncheck "Use Dedicated Folder for Storing Libraries" ⇒ **Uncheck "Create Main class"** ⇒ Finish.

Step 2: Write a Hello-world Java Program

- Right-click on "FirstProject" ⇒ New ⇒ Java Class (OR choose the "File" menu ⇒ "New File..." ⇒ Categories: "Java", File Types: "Java Class" ⇒ "Next").
- The "Name and Location" dialog pops up ⇒ In "Class Name", enter "Hello" ⇒ Delete the content in "Package" if it is not empty ⇒ "Finish".
- The source file "Hello.java" appears in the editor panel. Enter the following codes:

```
18. public class Hello {
19.     public static void main(String[] args) {
20.         System.out.println("Hello, world");
21.     }
}
```

Step 3: Compile & Execute

There is no need to "compile" the source code in NetBeans explicitly, as NetBeans performs the so-called *incremental compilation* (i.e., the source statement is compiled as and when it is entered).

To run the program, right-click anywhere in the source (or from the "Run" menu) ⇒ Run File. Observe the output on the output console.

**Notes:**

- You should create a NEW Java project for EACH of your Java application.
- Nonetheless, NetBeans allows you to keep more than one programs in a project, which is handy for writing toy programs (such as your tutorial exercises). To run a particular program, open and right-click on the source file ⇒ Run File.

### 2.1 Correcting Syntax Error

NetBeans performs incremented compilation, as and when a source line is entered. It marked a source line with syntax error with a RED CROSS. Point your cursor at the RED CROSS to view the error message.

You CANNOT RUN the program if there is any syntax error (marked by a RED CROSS before the filename). Correct all the syntax errors; and RUN the program.

[TODO] Diagram

HINTS: In some cases, NetBeans shows a ORANGE LIGHT-BULB (for HINTS) next to the ERROR RED-CROSS (Line 5 in the above diagram). You can click on the LIGHT-BULB to get a list of HINTS to resolve this particular error, which may or may not work!

SYNTAX WARNING: marked by a orange triangular exclamation sign. Unlike errors, warnings may or may not cause problems. Try to fix these warnings as well. But you can RUN your program with warnings.

### 3. Read the NetBeans Documentation

---

At a minimum, you SHOULD READ the "IDE Basics, Getting Started, Java Application", which is accessible via NetBeans's "HELP" menu  $\Rightarrow$  Help Contents. This will save you many agonizing hours trying to figure out how to do somethings later.

The "Help"  $\Rightarrow$  "Online Doc and Support" (@ <http://netbeans.org/kb/index.html>) contains many articles and tutorial on using NetBeans.

The NetBeans "Start Page" also provides many useful links to get you started.

### 4. Debugging Program in NetBeans

---

Step 0: Write a Java Program

The following program computes and prints the factorial of  $n$  ( $=1*2*3*\dots*n$ ). The program, however, has a logical error and produce a wrong answer for  $n=20$  ("The Factorial of 20 is -2102132736" - a negative number?!).

```
1/** Compute the factorial of n */
2public class Factorial {
3    // Print factorial of n
4    public static void main(String[] args) {
5        int n = 20;
6        int factorial = 1;
7
8        // n! = 1*2*3...*n
9        for (int i = 1; i <= n; i++) {
10            factorial *= i;
11        }
12        System.out.println("The Factorial of " + n + " is " + factorial);
13    }
14}
```

Let us use the graphic debugger to debug the program.

Step 1: Set an initial Breakpoint

A *breakpoint* suspends program execution for you to examine the internal states of the program. Before starting the debugger, you need to set at least one breakpoint to suspend the execution inside the program. Set a breakpoint at `main()` method by clicking on the *left-margin* of the line containing `main()`. A *red circle* or an inverted Triangle appears in the left-margin indicating a breakpoint is set at that line.

Step 2: Start Debugging

Right click anywhere on the source code  $\Rightarrow$  "Debug File". The program begins execution but suspends its operation at the breakpoint, i.e., the `main()` method.

As illustrated in the following diagram, the highlighted line (also pointed to by a green arrow) indicates the statement to be executed in the *next* step.

### Step 3: Step-Over and Watch the Variables and Outputs

Click the "Step Over" button (or select "Step Over" in "Debug" menu) to *single-step* thru your program. At each of the step, examine the value of the variables (in the "Variable" panel) and the outputs produced by your program (in the "Output" Panel), if any. You can also place your cursor at any variable to inspect the content of the variable.

Single-stepping thru the program and watching the values of internal variables and the outputs produced is the *ultimate* mean in debugging programs - because it is exactly how the computer runs your program!

### Step 4: Breakpoint, Run-To-Cursor, Continue and Finish

As mentioned, a breakpoint *suspends* program execution and let you examine the internal states of the program. To set a breakpoint on a particular statement, click on the left-margin of that line (or select "Toggle Breakpoint" from "Run" menu).

"Continue" resumes the program execution, up to the next breakpoint, or till the end of the program.

"Single-step" thru a loop with a large count is time-consuming. You could set a breakpoint at the statement immediately outside the loop (e.g., Line 11 of the above program), and issue "Continue" to complete the loop.

Alternatively, you can place the cursor on a particular statement, and issue "Run-To-Cursor" to resume execution up to the line.

"Finish" ends the debugging session. Always terminate your current debugging session using "Finish" or "Continue" till the end of the program.

## 4.1 Other Debugger's Features:

---

### Modify the Value of a Variable

You can modify the value of a variable by entering a new value in the "Variable" panel. This is handy for temporarily modifying the behaviour of a program, without changing the source code.

### Step-Into and Step-Out

To debug a *method*, you need to use "Step-Into" to step into the *first* statement of the method. You could use "Step-Out" to return back to the caller, anywhere within the method. Alternatively, you could set a breakpoint inside a method.

## 5. NetBeans - Tips & Tricks

---

### 5.1 General Usage

---

These are the features that I find to be most useful in NetBeans:

37. **Maximizing Window (double-click):** You can double-click on the "header" of any panel to *maximize* that particular panel, and double-click again to *restore* it back. This is particularly useful for editing source code in full panel.
38. **Code Auto-Complete (or Intelli-Sense) (ctrl-space):** Enter a partial statement (e.g., Sys) and press control-space to activate the auto-complete, which displays all the available choices.
39. **Javadoc (ctrl-space, alt-F1):** Place the cursor on a method or class, and press ctrl-space to view the javadoc; or right-click ⇒ Show Javadoc (alt-F1) to open it on a browser.
40. **Code Shorthand (tab):** For example, you can enter "sout" and press TAB for "System.out.println"; "psvm" for "public static void main(String[] args) { }" or "fori" + tab for a for-loop. To view and configure code template, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Code Templates".
41. **Formatting Source Code (alt-shift-f):** Right-click on the source (or from the "Source" menu) ⇒ Choose "Format". NetBeans will layout your source codes with the proper indents and format. To configure the formatting, choose "Tools" menu ⇒ "Options" ⇒ "Editor" ⇒ "Formatting". You can also select the section of codes to be formatted, instead of the entire file.
42. **Hints for Correcting Syntax Error:** If there is a syntax error on a statement, a red mark will show up on the left-margin on that statement. You could click on the "light bulb" to display the error message, and also select from the available hints for correcting that syntax error.

43. **Rename (Refactor) (ctrl-r):** To rename a variable, place the cursor on that variable, right-click ⇒ "Refactor" ⇒ "Rename" ⇒ Enter the new name. All the appearances of that variables in the project will be renamed.
44. **Small Programs:** You can keep many small toy programs (with `main()`) in one Java project instead of create a new project for each small program. To run the desired program, on the "editor" panel ⇒ right-click ⇒ "Run File".
45. **Source Toggle Comment:** To temporarily comment-off a block of codes, choose "Source" ⇒ "Toggle Comment".
46. **Error Message Hyperlink:** Click on an error message will hyperlink to the corresponding source statement.
47. **Command-Line Arguments:** To provide command-line arguments to your Java program in NetBeans, right-click on the "project" ⇒ "Set as Main Project" ⇒ "Set Configurations" ⇒ "Customize..." ⇒ "Run" ⇒ select the "Main" class ⇒ type your command-line arguments inside the "Arguments" field ⇒ choose "Run" menu ⇒ "Run Main Project".
48. **Line Numbers:** To show the line numbers, right-click on the left-margin ⇒ "Show Line Numbers".
49. **Changing Font Face and Size:** Tools ⇒ Options ⇒ Fonts & Colors ⇒ In "Category", select "Default" ⇒ In "Font", choose the font face and size.
50. **Resetting Window View:** If you mess up the window view (e.g., you accidentally close a window and cannot find it anymore), you can reset the view via "Window" menu ⇒ "Reset Windows".
51. **Code Templates:** For example, when you create a new Java class, NetBeans retrieves the initial contents from the "Java Class" code template. To configure code templates, select "Tools" menu ⇒ "Templates" ⇒ Choose the desired template ⇒ "Open in Editor". To set a value of a variable used in the all the code templates (e.g., `$User`), select "Tools" menu ⇒ "Templates" ⇒ "Settings".
52. **Displaying Chinese Character:** Need to choose a font that support chinese character display, such as "Monospace", in Tools ⇒ Options ⇒ Fonts & Colors ⇒ Syntax ⇒ default.
53. **Changing the JDK Location:** The Netbeans configuration file is located at "etc\netbeans.conf". Edit the directive "netbeans\_jdkhome".
54. Let me know if you have more tips to be included here.

## 5.2 Java Application Development

13. **Choosing the JDK version for your program:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ You can select the JDK level of your project in pull-down menu "Source/Binary Format".
14. **Enabling JDK 7 support:** If JDK 7 is already installed in your system, right-click on your Project ⇒ "Properties" ⇒ "Source" node ⇒ "Source/Binary Format" ⇒ Select "JDK 7". Also check "Libraries" ⇒ Java Platform ⇒ JDK 7.  
If JDK 7 is not installed/configured, install JDK 7. Add JDK 7 support to NetBeans via "Tool" menu ⇒ "Java Platforms" ⇒ "Add Platform...".
15. **Choosing Default Charset:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your desired charset for the text-file I/O from the pull-down menu.
16. **Enabling Unicode Support for File Encoding:** Right-click on your project ⇒ "Properties" ⇒ "Source" node ⇒ "Encoding" ⇒ choose your Unicode encoding (e.g., UTF-8, UTF-16, UTF-16LE, UTF-16GE) for the text-file I/O.
17. **To include Javadoc/Source:** Use "Library Manager" (select the "Tools" menu ⇒ "Libraries"); or "Java Platform Manager" (select "Tools" menu ⇒ "Java Platforms")
18. **Adding External JAR files & Native Libraries (".dll", ".lib", ".a", ".so"):** Many external Java packages (such as JOGL, Java3D, JAMA, etc) are available to extend the functions of JDK. These packages typically provide a "lib" directory containing JAR files (".jar") (Java Archive - a single-file package of Java classes) and native libraries (".dll", ".lib" for windows, ".a", ".so" for Linux and Mac).  
To include an external JAR file (".jar") into a project: Expand the project node ⇒ Right-click on "Libraries" ⇒ "Add JAR/Folder..." ⇒ Select the desired JAR file or the folder containing the classes. If the external package contains many JAR files, you could create a user library to contain all the JAR files, and add the library to all the projects that required these JAR files. From "Tools" menu ⇒ "Libraries" ⇒ "New Library..." ⇒ Enter a library name ⇒ Use "Add JAR/Folder..." to add JAR files into this library. Many JAR files come with native libraries in the form of ".dll", ".lib" (for Windows) and ".a", ".so" for Linux/Mac. The directory *path* of these libraries must be included in JRE's property

"`java.library.path`". This can be done via right-click the project ⇒ Set Configuration ⇒ Customize... ⇒ Run ⇒ In "VM options", enter "`-Djava.library.path=xxx`", where xxx is path of the native libraries.

**Notes:** The JAR files must be included in the CLASSPATH. The native library directories must be included in JRE's property "`java.library.path`", which normally but not necessarily includes all the paths from the PATH environment variable. Read "[External JAR files and Native Libraries](#)".

## 6. Writing Java GUI (AWT/Swing) Application in NetBeans

Step 0: Read

5. Java GUI Application Learning Trail @ <http://www.netbeans.org/kb/trails/matisse.html>.
6. Swing Tutorial's "Learning Swing with the NetBeans IDE" @ <http://docs.oracle.com/javase/tutorial/uiswing/learn/index.html>.

Step 1: Create a New "Java Application" Project

7. Launch NetBeans ⇒ File ⇒ New Project...
8. Under "Categories", choose "Java" ⇒ Under "Projects", choose "Java Application" ⇒ Next.
9. In "Project Name", enter "`FirstNetBeansGUI`" ⇒ Choose a suitable directory for your "Project Location" ⇒ *Uncheck* the "Create Main class" box ⇒ Finish.

Step 2: Write a Java File "JFrame Form"

19. Right-click on the project "`FirstNetBeansGUI`" ⇒ "New" ⇒ "JFrame Form..." (or "Others" ⇒ "Swing GUI Forms" ⇒ "JFrame Form").
20. In "Class Name", enter "`NetBeansSwingCounter`" ⇒ Finish.

21. Create the GUI Components visually:

- a. From the "Platte" panel ⇒ "Swing Controls" ⇒ Drag and drop a "`Label`", "`TextField`", and "`Button`" into the *design* panel.
- b. Click on the "`jLabel1`" ⇒ In the "Properties" panel, enter "Count" in "text" (You can also single-click on the `jLabel1` to change the text). Right-click on the `jLabel1` ⇒ Change Variable Name ⇒ In "New Name", enter "`lblCount`".
- c. Similarly, for "`jTextField1`" ⇒ Change the "text" to 0, and change the "Variable Name" to "`tfCount`" ⇒ Resize the text field if necessary.
- d. For " `jButton1`" ⇒ Change the "text" to "Count", and change the "Variable Name" to "`btnCount`".

22. Write the event handler for the button by double-clicking the button and enter the following codes:

```
23. private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {  
24.     count++;  
25.     tfCount.setText(count + "");  
  
    }
```

26. Create an instance variable `count` (just below the class declaration) as follows:

```
27. public class Counter extends javax.swing.JFrame {  
  
    int count = 0;
```

Step 3: Compile & Execute

Right-click the source and select "Run File".

Step 4: Study the Generated Source Code

Expand the "Generated Code" and study how the GUI builder declare, allocate and initialize the GUI Components in the  `initComponents()`. Note how the `JButton` registers an `ActionEvent` listener and how an inner class is used as the listener and provide the event handler  `actionPerformed()`. Also notice that the `main()` method uses a Swing's worker to run the GUI on the Event-Dispatcher thread, instead of the `main` thread, for thread-safe operations.

```
public class NetBeansSwingCounter extends javax.swing.JFrame {  
    int count = 0;
```



```

// Constructor to setup the UI via initComponents()
public NetBeansSwingCounter() {
    initComponents();
}

private void initComponents() {
    lblCount = new javax.swing.JLabel();
    tfCount = new javax.swing.JTextField();
    btnCount = new javax.swing.JButton();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    lblCount.setText("Counter");
    tfCount.setText("0");

    btnCount.setText("Count");
    // Create an anonymous inner as the listener for the ActionEvent fired by btnCount
    btnCount.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            btnCountActionPerformed(evt);
        }
    });

    // Laying out the components
    // .....

    pack();
}

// ActionEvent handler for btnCount
private void btnCountActionPerformed(java.awt.event.ActionEvent evt) {
    count++;
    tfCount.setText(count + "");
}

public static void main(String args[]) {
    // Setup the Look and Feel
    // .....

    // Run the constructor on the Event-Dispatcher Thread for thread-safe
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new NetBeansSwingCounter().setVisible(true);
        }
    });
}

// private variables
private javax.swing.JButton btnCount;
private javax.swing.JLabel lblCount;
private javax.swing.JTextField tfCount;
}

```

## 7. NetBeans and MySQL

**Reference:** "Connecting to a MySQL Database" @ <http://netbeans.org/kb/docs/ide/mysql.html>.

NetBeans (JavaEE) provides direct support to MySQL server. You can use NetBeans as a GUI client to access a MySQL server, as well as an administrative tool (e.g., starting and stopping the server).

Configuring NetBeans to Support MySQL

From NetBeans "Window" menu ⇒ Select "Services". The "Services" tab shall appear on the *left* pane

9. Right-click on the "Databases" node ⇒ "Register MySQL Server". (If you have already registered a MySQL server, you can right-click on Server node "MySQL Server at *hostname:port*" ⇒ Properties, to modify its properties.)
10. Select the "Basic Properties" tab, enter the hostname, port number, root user and password.
11. Select the "Admin Properties" tab:
  - a. Leave the "Path/URL to admin tool" empty.

- b. In "Path to start command", enter "<MYSQL\_HOME>\bin\mysqld.exe"; in the "Arguments", enter "--console"
- c. In "Path to stop command", enter "<MYSQL\_HOME>\bin\mysqladmin.exe", in the "Arguments", enter "-u root -ppassword shutdown".

12. A server node "MySQL Server at *hostname:port*" appears.

Database Administration - Start/Stop the Server and Create Databases

5. You can start the MySQL server by right-clicking on the server node ⇒ select "start". [There seems to be a problem here. If a "connection refused: connect" error occurs, enter the password again.]
6. Once the MySQL server is started and connected, you can see the list of databases by expanding the MySQL server node. You can create a new database by right-clicking on it and choose "Create Database...".

Create a new Connection

You need a connection to manipulate data. You can create multiple connections with different users and default databases.

5. Right-click on the "Databases" ⇒ "New Connection..." ⇒ Select the driver "MySQL Connector/J" ⇒ Next ⇒ Enter hostname, port number, default database, a general username and password ⇒ "Test Connection" (make sure that MySQL is started) ⇒ Finish.
6. A connection node "jdbc:mysql://*hostname:port/defaultDatabase*" appears.

Manipulating Data via a Connection

11. Right-click on a connection node (e.g., "jdbc:mysql://*hostname:port/defaultDatabase*") ⇒ Choose "Connect" (if not connected, provided that the MySQL server has been started).
12. You can expand the connection node to view all the databases.
13. Expand an existing database. There are three sub-nodes "Tables", "View" and "Procedures". Right-click on the "Tables" to create table or execute command. Similarly, right-click on the "View" and "Procedures".
14. To view/manipulate the records in a table, right-click on the selected table ⇒ You can choose to "View Data...", "Execute Command...", etc.
15. You can right-click on the connection to "connect" or "disconnect" from the server.

Create a SQL Script and Run the Script

You can create a SQL script by right-clicking on a project ⇒ New ⇒ "SQL File". You can run the script by right-clicking on the SQL script ⇒ "Run File" ⇒ Select an existing connection (or create a new connection) to run the script. You could also run a single statement (right-click on the statement ⇒ Run Statement) or a selected group of statements (highlight the statements ⇒ Right-click ⇒ Run Selection).

## 8. Developing and Deploying Web Application in NetBeans

Read:

- "Introduction to Developing Web Applications" @ <http://netbeans.org/kb/docs/web/quickstart-webapps.html>.
- More articles in "Java EE & Java Web Learning Trail" @ <http://netbeans.org/kb/trails/java-ee.html>.

### 8.1 Web (HTTP) Servers

Configuring Web Server

You could configure the web server via "Tools" menu ⇒ "Servers".

Tomcat Server

To configure Tomcat Server, select "Tools" menu ⇒ "Servers" ⇒ click "Add Servers":

11. Choose Server: Select the desired Tomcat version ⇒ Next.
12. Installation and Login Details: In "Server Location", fill in the Tomcat installation directory (\$CATALINA\_HOME) ⇒ Enter the username/password of a tomcat user with "manager" role. You could either check the "create user if it does not exist" or define the tomcat user in "\$CATALINA\_HOME\conf\tomcat-users.xml" as follows:

```

13. <tomcat-users>
14.     <role rolename="manager"/>
15.     <user username="tomcatmanager" password="xxxx" roles="manager,manager-script,admin" />

</tomcat-users>

```

Running the Web Server

Choose "Services" ⇒ Expand "Servers" node ⇒ Right-click on the desired server ⇒ Start/Stop/Restart.

## 8.2 MySQL Database Server

You can also manage the MySQL database server directly from Tomcat. Read "[NetBeans and MySQL](#)" Section.

## 8.3 Writing a Hello-World Servlet/JSP Web Application

Create a New Servlet/JSP Project

11. From "File" menu ⇒ choose "New Project...".
12. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
13. "Name and Location" ⇒ In "Project Name", enter "HelloServletJSP" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
14. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
15. "Frameworks" ⇒ Select none for pure servlet/JSP application ⇒ Finish.

Writing a Hello-World JSP

A JSP page called "index.jsp" is automatically created, which says "Hello world!". To execute this JSP, right-click on the project ⇒ "Run". The URL is <http://localhost:8080/HelloServletJSP/index.jsp>.

Writing a Hello-World Servlet

69. Right-click on the project "HelloServletJSP" ⇒ New ⇒ Servlet.
70. "Name and Location" ⇒ In "Class Name", enter "HelloServlet" ⇒ In "Package", enter "hello" ⇒ Next.
71. "Configure Servlet Deployment" ⇒ In "Servlet Name", enter "HelloServletExample" ⇒ In "URL Pattern", enter "sayhello" ⇒ Finish.
72. Enter the following codes for "HelloServlet.java":

```

73. package hello;
74.
75. import java.io.IOException;
76. import java.io.PrintWriter;
77. import javax.servlet.ServletException;
78. import javax.servlet.http.HttpServlet;
79. import javax.servlet.http.HttpServletRequest;
80. import javax.servlet.http.HttpServletResponse;
81.
82. public class HelloServlet extends HttpServlet {
83.
84.     @Override
85.     public void doGet(HttpServletRequest request, HttpServletResponse response)
86.         throws IOException, ServletException {
87.         // Set the response message's MIME type (in Content-Type response header)
88.         response.setContentType("text/html;charset=UTF-8");
89.         // Get an output Writer to write the response message over the network
90.         PrintWriter out = response.getWriter();
91.         // Write the response message (in an HTML page) to display "Hello, world!"
92.         try {
93.             out.println("<!DOCTYPE html>");
94.             out.println("<html>");
95.             out.println("<head><title>Hello Servlet</title></head>");
96.             out.println("<body><h1>Hello, World (from Java Servlet)!</h1></body>");
97.             out.println("</html>");
98.         } finally {

```

```

99.         out.close(); // Always close the output writer
100.     }
101. }
    }

```

102. To execute the servlet: Right-click on the project ⇒ run ⇒ Change the URL to `http://localhost:8080/HelloServletJSP/sayhello`.

#### Generating a WAR-file for a Web Application

A WAR (Web Archive) file is basically a zip file for distributing web application in single file. You can use WinZip or WinRAR to inspect or unzip the war file.

To distribute the project as a war-file, right-click project ⇒ "Clean and Build". The war file is created in the "**dist**" directory. You can deploy the web application by dropping the war-file into Tomcat's "**webapps**" directory. Tomcat will automatically unzip the war-file and deploy the application upon startup.

#### Debugging Web Application

The most important reason for using IDE is to use the graphic debugger for debugging the program. You can set a breakpoint in your server-side Java codes, and "Debug" a web application, similar to a standalone application.

### 8.4 Writing a Hello-world JSF 2.0 Web Application

#### Create a New JSF 2.0 Project

33. From "File" menu ⇒ choose "New Project...".
34. "Choose Project" ⇒ Under "Categories", choose "Java Web" ⇒ Under "Projects", choose "Web Application" ⇒ "Next".
35. "Name and Location" ⇒ In "Project Name", enter "HelloJSF20" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
36. "Server and settings" ⇒ Choose your server, or "add" a new server ⇒ Next.
37. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 2.0" ⇒ Finish.
38. An "index.xhtml" JSF page is generated, as follows:

```

39. <?xml version='1.0' encoding='UTF-8' ?>
40. <!DOCTYPE html>
41. <html xmlns="http://www.w3.org/1999/xhtml"
42.       xmlns:h="http://java.sun.com/jsf/html">
43.     <h:head>
44.         <title>Facelet Title</title>
45.     </h:head>
46.     <h:body>
47.         Hello from Facelets
48.     </h:body>
    </html>

```

To run this facelet, right-click on the project ⇒ Run.

#### Create a new JSF 2.0 Facelet

31. Right-click on the project ⇒ New ⇒ "Other..."
32. "Choose File Type" ⇒ Under "Category", select "JavaServer Faces" ⇒ Under "File Type", select "JSF Page" ⇒ Next.
33. "Name and Location" ⇒ In "File Name", enter "HelloJSF20" ⇒ In "Options", check "Facelets" ⇒ Finish.
34. In "HelloJSF20.xhtml", enter the following codes:

```

35. <?xml version='1.0' encoding='UTF-8' ?>
36. <!DOCTYPE html>

```

```

37. <html xmlns="http://www.w3.org/1999/xhtml"
38.     xmlns:h="http://java.sun.com/jsf/html">
39.     <h:head>
40.         <title>Hello JSF 2.0</title>
41.     </h:head>
42.     <h:body>
43.         <h1>Hello from Facelets</h1>
44.     </h:body>

```

```
</html>
```

45. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF20/HelloJSF20.xhtml`.

## 8.5 Writing a Hello-world JSF 1.2 Web Application

Create a New JSF 1.2 Project

47. From "File" menu ⇒ choose "New Project...".
48. "Choose Project" ⇒ In "Categories", choose "Java Web" ⇒ In "Projects", choose "Web Application" ⇒ "Next".
49. "Name and Location" ⇒ In "Project Name", enter "HelloJSF12" ⇒ In "Project Location", select a suitable directory to save your works ⇒ Check "Set as Main Project" ⇒ Next.
50. "Server and settings" ⇒ choose your server, or "add" a new server ⇒ Next.
51. "Frameworks" ⇒ Check "JavaServer Faces" ⇒ In "Libraries", "Registered Libraries", select "JSF 1.2" ⇒ Finish.
52. A "WelcomeJSF.jsp" page is generated, as follows:

```

53. <%@page contentType="text/html" pageEncoding="UTF-8"%>
54. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
55. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
56. <!DOCTYPE html>
57. <!--
58.  This file is an entry point for JavaServer Faces application.
59. --%>
60. <f:view>
61.     <html>
62.         <head>
63.             <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
64.             <title>JSP Page</title>
65.         </head>
66.         <body>
67.             <h1><h:outputText value="JavaServer Faces"/></h1>
68.         </body>
69.     </html>

```

```
</f:view>
```

To run this page, right-click on the project ⇒ Run.

Create a new JSF 1.2 Page

41. Right-click on the project ⇒ New ⇒ "Other..."
42. "Choose File Type" ⇒ In "Category", select "JavaServer Faces" ⇒ In "File Type", select "JSF Page" ⇒ Next.
43. "Name and Location" ⇒ In "File Name", enter "HelloJSF12" ⇒ In "Options", check "JSP File (Standard Syntax)" ⇒ Finish.
44. In "HelloJSF12.jsp", enter the following codes:

```

45. <%@page contentType="text/html" pageEncoding="UTF-8"%>
46. <%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
47. <%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

```

```

48. <!DOCTYPE html>
49.
50. <f:view>
51.   <html>
52.     <head>
53.       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
54.       <title>Hello JSF 1.2</title>
55.     </head>
56.     <body>
57.       <h1><h:outputText value="Hello World!"/></h1>
58.     </body>
59.   </html>

```

```
</f:view>
```

60. To execute the JSF page, right-click on the project ⇒ Run ⇒ Change the URL to `http://localhost:8080/HelloJSF12/faces/HelloJSF12.jsp`.

## 8.6 Debugging Web Applications in NetBeans

You can debug a webapp just like standalone application. For example, you can set breakpoints, single-step through the programs, etc.

### Unit : 3 servlet

**Servlet** technology is used to create a web application (resides at server side and generates a dynamic web page).

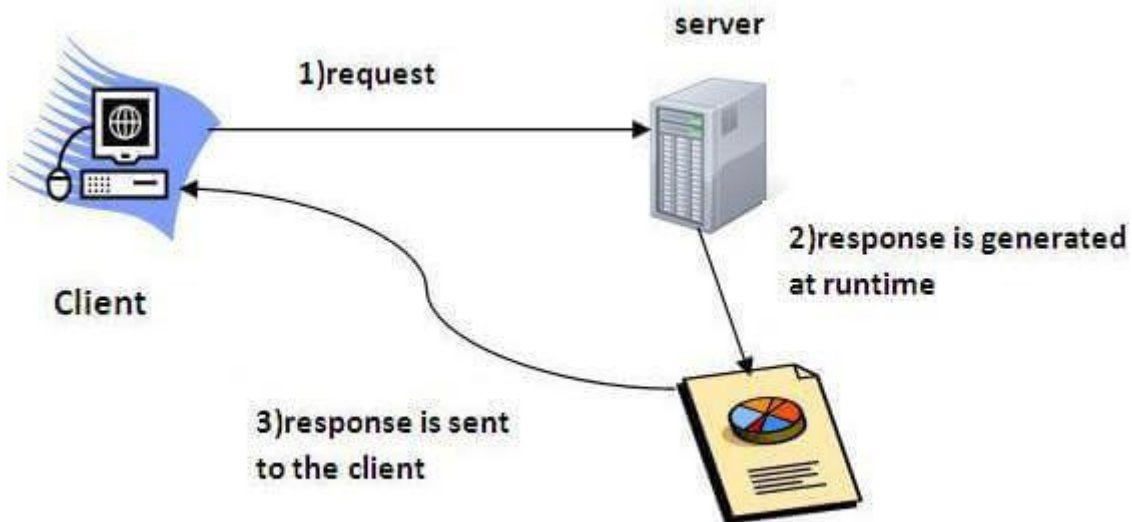
**Servlet** technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

## What is a Servlet?

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.



A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the **init()** method.
- The servlet calls **service()** method to process a client's request.
- The servlet is terminated by calling the **destroy()** method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

## The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this –

```
public void init() throws ServletException {  
    // Initialization code...  
}
```

## The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method –

```
public void service(ServletRequest request, ServletResponse
response)
    throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

## The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
    // Servlet code
}
```

## The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

## The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.



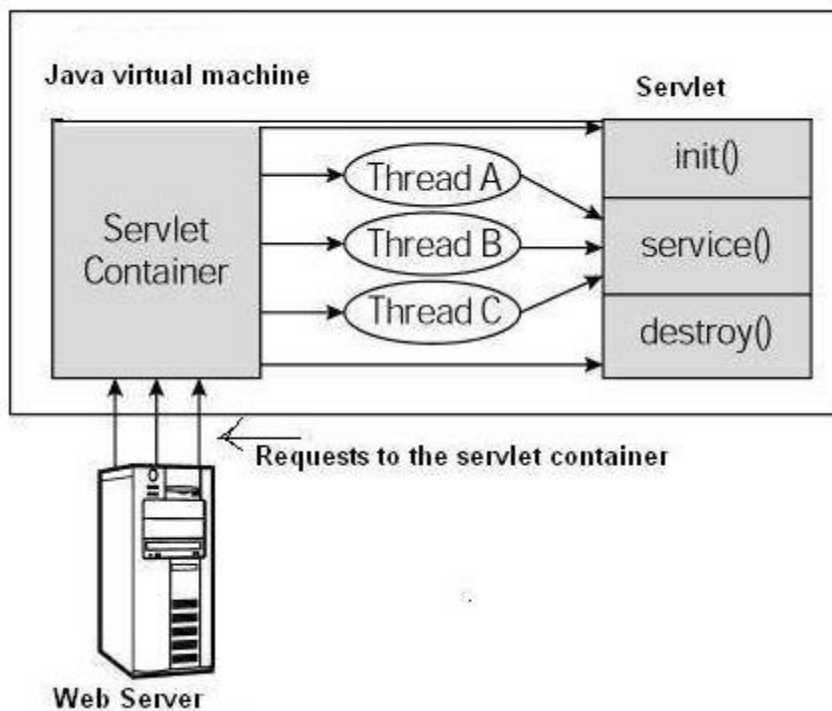
After the `destroy()` method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this –

```
public void destroy() {  
    // Finalization code...  
}
```

## Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the `service()` method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the `service()` method of a single instance of the servlet.



## Six Steps to Running Your First Servlet

Once Tomcat is installed and configured, you can put it to work. Six steps take you from writing your servlet to running it. These steps are as follows:

7. Create a directory structure under Tomcat for your application.
8. Write the servlet source code. You need to import the `javax.servlet` package and the `javax.servlet.http` package in your source file.
9. Compile your source code.

10. Create a deployment descriptor.
11. Run Tomcat.
12. Call your servlet from a web browser.

## Step 1: Create a Directory Structure under Tomcat

When you install Tomcat, several subdirectories are automatically created under the Tomcat home directory (%TOMCAT\_HOME%). One of the subdirectories is webapps. The webapps directory is where you store your web applications. A *web application* is a collection of servlets and other contents installed under a specific subset of the server's URL namespace. A separate directory is dedicated for each servlet application. Therefore, the first thing to do when you build a servlet application is create an application directory. This section explains how to create a directory structure for an application called myApp.

3. Create a directory called myApp under the webapps directory. The directory name is important because this also appears in the URL to your servlet.
4. Create the src and WEB-INF directories under myApp, and create a directory named classes under WEB-INF. The directory structure is shown in [Figure 1.4](#). The src directory is for your source files, and the classes directory under WEB-INF is for your Java classes. If you have html files, you put them directly in the myApp directory. You also may want to create a directory called images under myApp for all your image files.

Note that the admin, ROOT, and examples directories are for applications created automatically when you install Tomcat.



**Figure 1.4** Tomcat application directory structure.

## Step 2: Write the Servlet Source Code

In this step, you prepare your source code. You can write the source code yourself using your favorite text editor or copy it from the CD included with the book.

The code in Listing 1.1 shows a simple servlet called TestingServlet. The file, named TestingServlet.java, sends to the browser a few HTML tags and some text. For now, don't worry if you haven't got a clue about how it works.

### Listing 1—TestingServlet.java

```
import javax.servlet.*;

import javax.servlet.http.*;
```

```
import java.io.*;

import java.util.*;

public class TestingServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,

        HttpServletResponse response)

        throws ServletException, IOException {

        PrintWriter out = response.getWriter();

        out.println("<HTML>");

        out.println("<HEAD>");

        out.println("<TITLE>Servlet Testing</TITLE>");

        out.println("</HEAD>");

        out.println("<BODY>");

        out.println("Welcome to the Servlet Testing Center");

        out.println("</BODY>");

        out.println("</HTML>");

    }

}
```

Now, save your `TestingServlet.java` file to the `src` subdirectory under `myApp`. You actually can place the source files anywhere; however, it is always a good idea to be organized by storing all your source code files in the `src` directory.

### Step 3: Compile Your Source Code

For your servlet source code to compile, you need to include in your `CLASSPATH` environment variable the path to the `servlet.jar` file. The `servlet.jar` is located in the `common\lib\` subdirectory under `%CATALINA_HOME%`.

#### NOTE

If you have forgotten how to edit the `CLASSPATH` environment variable, refer to Appendix A, "Tomcat Installation and Configuration."

If you are using Windows, remember that the new environment variable takes effect only for new console windows. In other words, after changing a new environment variable, open a new console window for typing your command lines.

Now, change directory to your working directory and type the following if you are using Windows:

```
javac -d ..\WEB-INF\classes\ TestingServlet.java
```

If you are using Linux/UNIX, the command is very similar, except that `/` is used to separate a directory from a subdirectory.

```
javac -d ../WEB-INF/classes/ TestingServlet.java
```

The `-d` option specifies where to place the generated class files. The command also assumes that you have placed the JDK's `bin` directory in the path so you can call any program in it from any directory.

### Step 4: Create the Deployment Descriptor

A *deployment descriptor* is an optional component in a servlet application, taking the form of an XML document called `web.xml`. The descriptor must be located in the `WEB-INF` directory of the servlet application. When present, the deployment descriptor contains configuration settings specific to that application. Deployment descriptors are discussed in detail in Chapter 16, "Application Deployment."

For this step, you now need to create a web.xml file and place it under the WEB-INF directory under myApp.

The web.xml for this example application must have the following content.<?xml version="1.0" encoding="ISO-8859-1"?>

```
<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <servlet>

    <servlet-name>Testing</servlet-name>

    <servlet-class>TestingServlet</servlet-class>

  </servlet>

</web-app>
```

The web.xml file has one element: web-app. You should write all your servlets under <web-app>. For each servlet, you have a <servlet> element and you need the <servlet-name> and <servlet-class> elements. The <servlet-name> is the name for your servlet, by which it is known to Tomcat. The <servlet-class> is the compiled file of your servlet without the .class extension.

Having more than one servlet in an application is common. For every servlet, you need a <servlet> element in the web.xml file. For example, the following code shows how the web.xml looks if you add another servlet called Login.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE web-app

PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

    <servlet>

        <servlet-name>Testing</servlet-name>

        <servlet-class>TestingServlet</servlet-class>

    </servlet>

    <servlet>

        <servlet-name>Login</servlet-name>

        <servlet-class>LoginServlet</servlet-class>

    </servlet>

</web-app>
```

## Step 5: Run Tomcat

If it is not already running, you need to start Tomcat. For information on how to do that, see Appendix A, "Tomcat Installation and Configuration."

## Step 6: Call Your Servlet from a Web Browser

You are ready to call your servlet from a web browser. By default, Tomcat runs on port 8080 in myApp virtual directory under the servlet subdirectory. The servlet that you just wrote is named Testing. The URL for that servlet has the following format:

```
http://domain-name/virtual-directory/servlet/servlet-name
```

If you run the web browser from the same computer as Tomcat, you can replace *domain-name* with *localhost*. Therefore, the URL for your servlet would be `http://localhost:8080/myApp/servlet/Testing`.

In the deployment descriptor you wrote in Step 4, you actually mapped the servlet class file called `TestingServlet` with the name "Testing" so that your servlet can be called by specifying its class file (`TestingServlet`) or its name (`Testing`). Without a deployment descriptor, your servlet must be called by specifying its class name; that is, `TestingServlet`. This means that if you had not written a deployment descriptor in Step 4, you would have to use the following URL to call your servlet:

```
http://localhost:8080/myApp/servlet/TestingServlet
```

Typing the URL in the Address or Location box of your web browser will give you the string "Welcome to the Servlet Testing Center," as shown in [Figure 1.5](#).



**Figure 1.5** The Testing servlet.

Congratulations. You have just written your first servlet.

[yet another insignificant programming notes...](#) | [HOME](#)

## TABLE OF CONTENTS [\(HIDE\)](#)

### [1. Introduction](#)

### [2. Review of HTTP](#)

### [3. First "Hello-world" Servlet](#)

#### [3.1 Create a new Webapp "helloservlet"](#)

#### [3.2 Write a Hello-world Java Servlet - "HelloServlet.java"](#)

#### [3.3 Configure the Application Deployment Descriptor - "web.xml"](#)

#### [3.4 Run the Hello-world Servlet](#)

### [4. Processing HTML Form Data](#)

#### [4.1 Write an HTML Form](#)

#### [4.2 Write a Servlet to Process Form Data - "EchoServlet.java"](#)

#### [4.3 Configure the Servlet URL mapping in "web.xml"](#)

#### [4.4 Run the EchoServlet](#)

#### [4.5 Form-Data Submission Methods: GET | POST](#)

### [5. Request Header and Response Header](#)

#### [5.1 HttpServletRequest](#)

#### [5.2 HttpServletResponse](#)

- [6. Session Tracking](#)
  - [6.1 HttpSession](#)
  - [6.2 Example](#)
- [7. ServletConfig and ServletContext](#)
- [8. Developing and Deploying Web Applications using IDE](#)
- [9. Tomcat's Servlet Examples](#)
- [10. Database Servlet](#)
- [11. Servlet API – A Deeper Look](#)
  - [11.1 Interface Servlet](#)
  - [11.2 A Servlet's Life cycle](#)
  - [11.3 Interface ServletContext](#)
  - [11.4 Dispatch Request - RequestDispatcher](#)
  - [11.5 Filtering](#)
- [12. Web Application Deployment Descriptor "web.xml"](#)
  - [12.1 A Sample "web.xml"](#)
  - [12.2 Syntax for "web.xml"](#)
  - [12.3 Servlet Deployment Descriptor](#)
  - [12.4 Servlet Initialization Parameters](#)
  - [12.5 Application Initialization Parameters](#)
  - [12.6 Server-wide Initialization Parameters](#)
  - [12.7 Welcome Page](#)
- [13. Servlet 3.0](#)
  - [13.1 @WebServlet](#)
  - [13.2 @WebInitParam](#)
  - [13.3 @WebFilter](#)
  - [13.4 @WebListener](#)
  - [13.5 @MultipartConfig](#)

# Java Server-Side Programming

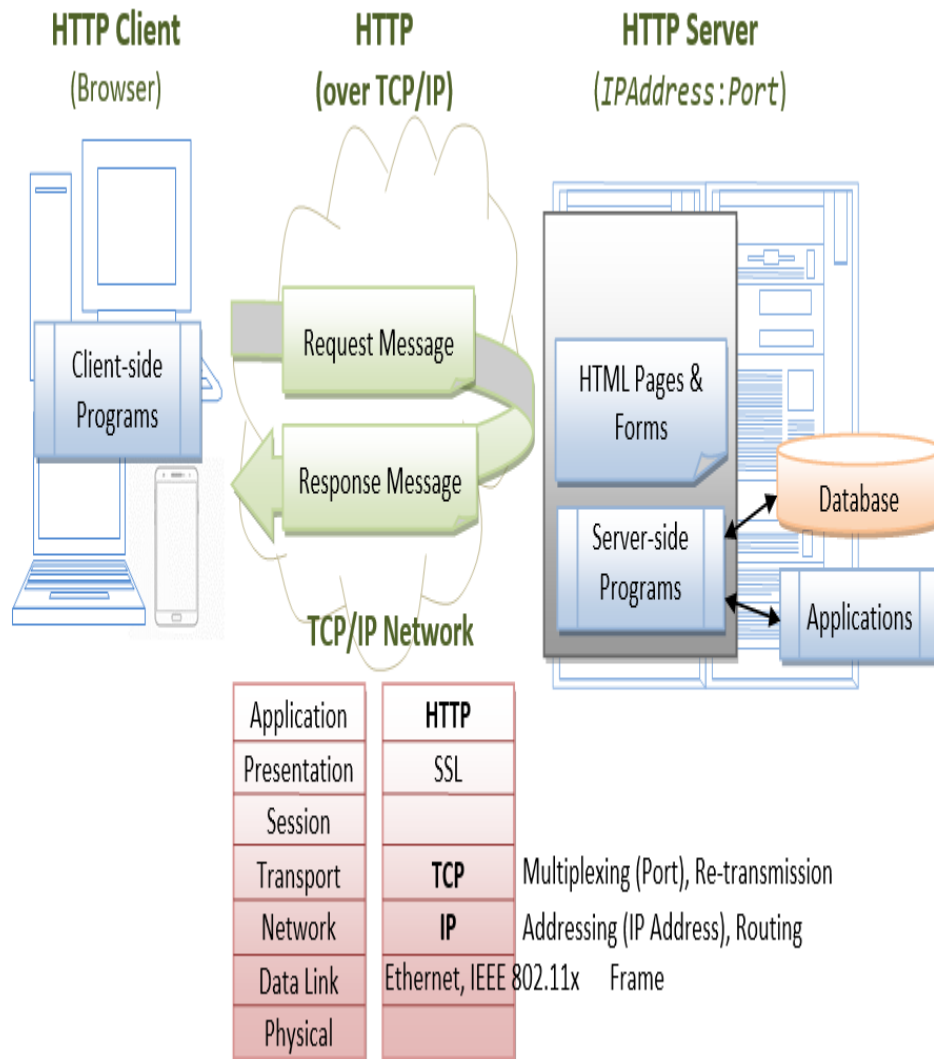
## Java Servlets

### 1. Introduction

In the early days, web servers deliver *static* contents that are indifferent to users' requests. Java servlets are *server-side programs* (running inside a web server) that handle clients' requests and return a *customized* or *dynamic response* for each request. The dynamic response could be based on user's input (e.g., search, online shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices).



Java servlets typically run on the HTTP protocol. HTTP is an *asymmetrical request-response protocol*. The client sends a *request message* to the server, and the server returns a *response message* as illustrated.



## Server-Side Technologies

There are many (competing) server-side technologies available: Java-based (servlet, JSP, JSF, Struts, Spring, Hibernate), ASP, PHP, CGI Script, and many others.

Java servlet is the *foundation* of the Java server-side technology, JSP (JavaServer Pages), JSF (JavaServer Faces), Struts, Spring, Hibernate, and others, are extensions of the servlet technology.

### Pre-requisites

HTML, Java Programming Language, HTTP and Apache Tomcat Server, SQL and MySQL Database System, and many others.

### Apache Tomcat Server

Servlets are server-side programs run inside a *Java-capable* HTTP server. Apache Tomcat Server (@ <http://tomcat.apache.org>) is the official Reference Implementation (RI) for Java servlet and JSP, provided free by open-source foundation Apache (@ <http://www.apache.org>).

You need to install Tomcat to try out Java servlets. Read "[How to Install Tomcat and Get Started Java Servlet Programming](#)".

I shall denote Tomcat's installed directory as <CATALINA\_HOME>, and assume that Tomcat server is running in port 8080.

Tomcat provides many excellent servlet examples in "<CATALINA\_HOME>\webapps\examples\servlets". You can run these examples by launching Tomcat and issuing URL <http://localhost:8080/examples>.

## Java Servlet Versions

Java Servlet has these versions: [TODO features and what is new]

- J2EE 1.2 (December 12, 1999) (**Java Servlet 2.2**, JSP 1.1, EJB 1.1, JDBC 2.0)
- J2EE 1.3 (September 24, 2001) (**Java Servlet 2.3**, JSP 1.2, EJB 2.0, JDBC 2.1)
- J2EE 1.4 (November 11, 2003) (**Java Servlet 2.4**, JSP 2.0, EJB 2.1, JDBC 3.0)
- Java EE 5 (May 11, 2006) (**Java Servlet 2.5**, JSP 2.1, JSTL 1.2, JSF 1.2, EJB 3.0, JDBC 3.0)
- Java EE 6 (December 10, 2009) (**Java Servlet 3.0**, JSP 2.2/EL 2.2, JSTL 1.2, JSF 2.0, EJB 3.1, JDBC 4.0)
- Java EE 7: expected in end of 2012.

The Java Servlets Home Page is @ <http://java.sun.com/products/servlet> (<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>). For developers, check out the Servlet Developers @ <http://java.net/projects/servlet/>.

Java Servlet is the *foundation* technology for Java server-side programming. You need to understand Servlet thoroughly before you could proceed to other Java server-side technologies such as JavaServer Pages (JSP) and JavaServer Faces (JSF).

## 2. Review of HTTP

A HTTP Servlet runs under the HTTP protocol. It is important to understanding the HTTP protocol in order to understand server-side programs (servlet, JSP, ASP, PHP, etc) running over the HTTP. Read "[HTTP Basics](#)", if needed.

In brief, HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The messages consists of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs.

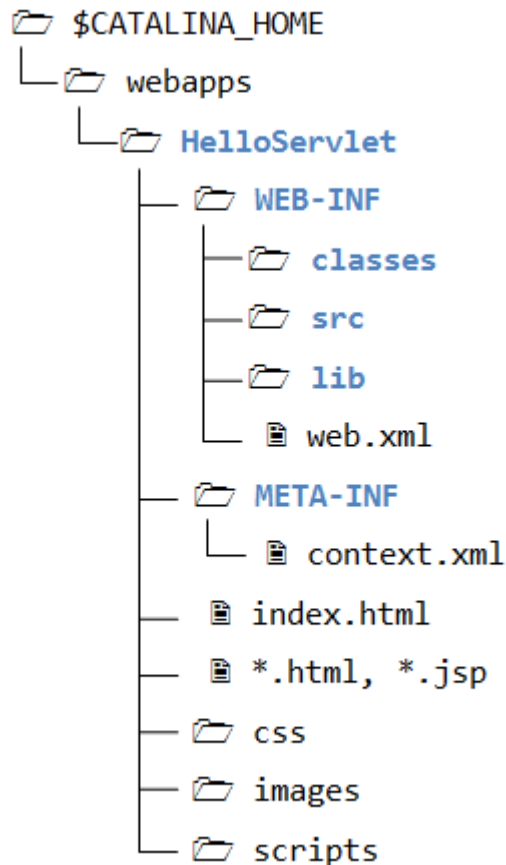
Read "[HTTP Request and Response Messages](#)" for the format, syntax of request and response messages, and examples.

## 3. First "Hello-world" Servlet

Let us begin by writing a servlet that says hello in response to a client's request. We shall use JDK and Tomcat to understand the basics, instead of IDE such as Eclipse/NetBeans. Once you understand the basics, you should use Eclipse/NetBeans to develop your webapp for better productivity.

### 3.1 Create a new Webapp "helloservlet"

We shall begin by defining a new webapp (web application) called "helloservlet" in Tomcat. A webapp, known as a *web context* in Tomcat, comprises a set of resources, such as HTML files, CSS, JavaScripts, images, programs and libraries.



A Java webapp has a *standardized directory structure* for storing various types of resources. Create a directory "helloservlet" under Tomcat's "webapps" directory (i.e., "<CATALINA\_HOME>\webapps\helloservlet", where <CATALINA\_HOME> denotes Tomcat's installed directory). Create sub-directories "WEB-INF" and "META-INF" under "helloservlet". Create sub-sub-directories "classes", "lib" and "src" under "WEB-INF". Take note that the directory names are case-sensitive.

The resources must be kept in the respective directories:

- **<CATALINA\_HOME>\webapps\helloservlet**: This directory is known as *context root* for the web context "helloservlet". It contains the resources that are *accessible by the clients*, such as HTML, CSS, Scripts and images. These resources will be delivered to the clients *as it is*. You could create sub-directories such as *images*, *css* and *scripts*, to further categories the resources.
- **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF**: This directory is NOT accessible by the clients directly. This is where you keep your application-specific configuration files (such as "web.xml"), and its sub-directories contain program classes, source files, and libraries.
  - **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF\src**: Keep the Java program source files. It is a good practice to separate the source files and classes to facilitate deployment.
  - **<CATALINA\_HOME>\webapps\helloservlet\WEB-INF\classes**: Keep the Java classes (compiled from the source codes). Classes defined in packages must be kept according to the package directory structure.

- `<CATALINA_HOME>\webapps\helloservlet\WEB-INF\lib`: keep the JAR files provided by external packages, available to this webapp only.
- `<CATALINA_HOME>\webapps\helloservlet\META-INF`: This directory is also NOT accessible by the clients. It keeps resources and configurations (e.g., "context.xml") related to the particular server (e.g., Tomcat, Glassfish). In contrast, "WEB-INF" is for resources related to this webapp, independent of the server.

### 3.2 Write a Hello-world Java Servlet - "HelloServlet.java"

Servlets are Java programs that runs inside a Java-capable HTTP server. A user can invoke a servlet by issuing a specific URL from the browser (HTTP client). In this example, we shall write a servlet called "HelloServlet.java" and compiled into "HelloServlet.class". A client can invoke "HelloServlet.class" by issuing URL `http://hostname:port/helloServlet/sayhello` (i.e., "sayhello" relative to the webapp). A servlet shall be kept inside a Java package (instead of the default *no-name* package) for proper deployment. Let's call our package "mypkg". Create a sub-directory called "mypkg" under "WEB-INF\src". Use a programming text editor to enter the following source codes, and save as "HelloServlet.java" in "`<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg`".

```
1// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\HelloServlet.java"
2package mypkg;
3
4import java.io.*;
5import javax.servlet.*;
6import javax.servlet.http.*;
7
8public class HelloServlet extends HttpServlet {
9    @Override
10    public void doGet(HttpServletRequest request, HttpServletResponse response)
11        throws IOException, ServletException {
12        // Set the response message's MIME type
13        response.setContentType("text/html;charset=UTF-8");
14        // Allocate a output writer to write the response message into the network socket
15        PrintWriter out = response.getWriter();
16
17        // Write the response message, in an HTML page
18        try {
19            out.println("<!DOCTYPE html>");
20            out.println("<html><head>");
21            out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
22            out.println("<title>Hello, World</title></head>");
23            out.println("<body>");
24            out.println("<h1>Hello, world!</h1>"); // says Hello
25            // Echo client's request information
26            out.println("<p>Request URI: " + request.getRequestURI() + "</p>");
27            out.println("<p>Protocol: " + request.getProtocol() + "</p>");
28            out.println("<p>PathInfo: " + request.getPathInfo() + "</p>");
29            out.println("<p>Remote Address: " + request.getRemoteAddr() + "</p>");
30            // Generate a random number upon each request
31            out.println("<p>A Random Number: <strong>" + Math.random() + "</strong></p>");
32            out.println("</body>");
```

```

33     out.println("</html>");
34 } finally {
35     out.close(); // Always close the output writer
36 }
37 }
38}

```

### Dissecting the Program:

- We define a Java class called `HelloServlet` (in Line 8). Line 2 places this class in a package called `mypkg`. Hence, we save the source file under "`mypkg`" of the "`helloservlet\WEB-INF\src`" directory, following the Java's standard package directory structure.
- We need the Servlet API library to compile this program. Servlet API is not part of JDK or Java SE (but belongs to Java EE). Tomcat provides a copy of servlet API called "`servlet-api.jar`" in "`<CATALINA_HOME>\lib`". You could copy "`servlet-api.jar`" from "`<CATALINA_HOME>\lib`" to "`<JAVA_HOME>\jre\lib\ext`" (the JDK Extension Directory), or include the Servlet JAR file in your `CLASSPATH`.
- To compile the program under JDK, we need to use the `-d` option to specify the output *destination* directory to place the compiled class in "`helloservlet\WEB-INF\classes\mypkg`" directory.

- `// Change directory to <CATALINA_HOME>\webapps\helloservlet\WEB-INF`
- `d:\...> cd <CATALINA_HOME>\webapps\helloservlet\WEB-INF`
- `// Compile the source file and place the class in the specified destination directory`

```

d:\<CATALINA_HOME>\webapps\helloservlet\WEB-INF> javac -d classes
src\mypkg\HelloServlet.java

```

The option "`-d classes`" specifies the output destination directory, relative to the current directory. The output is `<CATALINA_HOME>\webapps\helloservlet\WEB-INF\classes\mypkg\HelloServlet.class`. The compiler creates the package directory "`mypkg`" automatically.

- We don't write a servlet from scratch. Instead, we create a servlet by subclassing `javax.servlet.http.HttpServlet` (in Line 8).
- As mentioned, a servlet is invoked in response to a request URL issued by a client. Specifically, a client issues an HTTP request, the server routes the request message to the servlet for processing. The servlet returns a response message to the client.
- An HTTP request could use either GET or POST request methods, which will be processed by the servlet's `doGet()` or `doPost()` method, respectively.
- In the `HelloServlet`, we override the `doGet()` method (as denoted by the annotation `@Override`). The `doGet()` runs in response to an HTTP GET request issued by a user via an URL. `doGet()` takes two arguments, an `HttpServletRequest` object and an `HttpServletResponse` object, corresponding to the request and response messages.
- The `HttpServletRequest` object can be used to retrieve incoming HTTP *request headers* and *form data*. The `HttpServletResponse` object can be used to set the HTTP *response headers* (e.g., *content-type*) and the *response message body*.
- In Line 13, we set the "MIME" type of the response message to "`text/html`". The client need to know the message type in order to correctly display the data received. (Other MIME types include `text/plain`, `image/jpeg`, `video/mpeg`, `application/xml`, and many others.) In Line

15, we retrieve a `Writer` object called out for writing the response message to the client over the network. We then use the `out.println()` to print out a proper HTML page containing the message "Hello, world!". This servlet also echoes some of the clients's request information, and prints a random number for each request.

### 3.3 Configure the Application Deployment Descriptor - "web.xml"

A web user invokes a servlet, which is kept in the web server, by issuing a specific URL from the browser. In this example, we shall configure the following request URL to trigger the "HelloServlet":

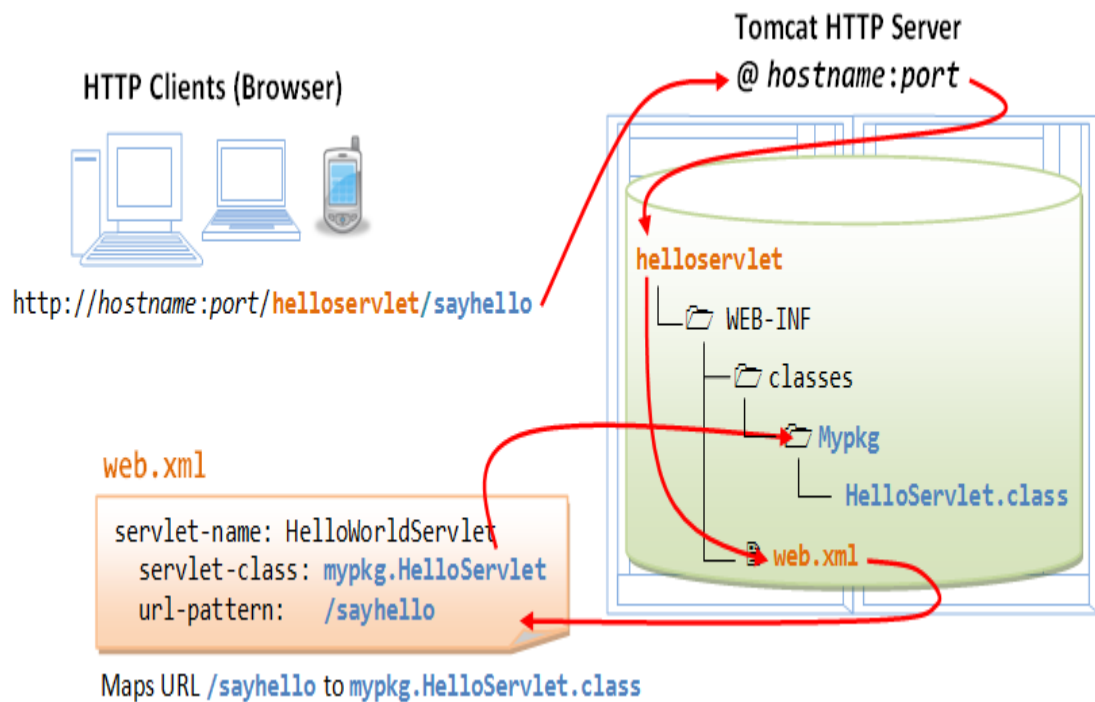
```
http://hostname:port/helloservlet/sayhello
```

Create a configuration file called "web.xml", and save it under "webapps\helloservlet\WEB-INF", as follows:

```
1<?xml version="1.0" encoding="ISO-8859-1"?>
2<web-app version="3.0"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee
6
7  <!-- To save as <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml -->
8
9  <servlet>
10    <servlet-name>HelloWorldServlet</servlet-name>
11    <servlet-class>mypkg.HelloServlet</servlet-class>
12  </servlet>
13
14  <!-- Note: All <servlet> elements MUST be grouped together and
15    placed IN FRONT of the <servlet-mapping> elements -->
16
17  <servlet-mapping>
18    <servlet-name>HelloWorldServlet</servlet-name>
19    <url-pattern>/sayhello</url-pattern>
20  </servlet-mapping>
21</web-app>
```

- The "web.xml" is called *web application deployment descriptor*. It provides the configuration options for that particular web application, such as defining the the *mapping* between URL and servlet class.
- The above configuration defines a servlet named "HelloWorldServlet", implemented in "mypkg.HelloServlet.class" (written earlier), and maps to URL "/sayhello", where "/" denotes the context root of this webapp "helloservlet". In other words, the absolute URL

for this servlet is `http://hostname:port/helloservlet/sayhello`.



- Take note that EACH servlet requires a pair of `<servlet>` and `<servlet-mapping>` elements to do the mapping, via an arbitrary but unique `<servlet-name>`. Furthermore, all the `<servlet>` elements must be grouped together and placed before the `<servlet-mapping>` elements (as specified in the XML schema).

### 3.4 Run the Hello-world Servlet

To run the servlet, first start the Tomcat server. Verify that the web context "helloservlet" has been deployed by observing the following messages in the Tomcat's console:

```
xxx x, xxxx xx:xx:xx xx org.apache.catalina.startup.HostConfig deployDirectory
INFO: Deploying web application directory helloservlet
.....
```

Start a web browser (Firefox, IE or Chrome), and issue the following URL (as configured in the "web.xml"). Assume that Tomcat is running in port number 8080.

```
http://localhost:8080/helloservlet/sayhello
```

We shall see the output "Hello, world!".

# Hello, world!

Request URI: /helloservlet/sayhello

Protocol: HTTP/1.1

PathInfo: null

Remote Address: 127.0.0.1

A Random Number: **0.4320795689818858**

Try selecting "View Source" in your browser, which produces these output:

```
<!DOCTYPE html>
<html><head>
<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
<title>Hello, World</title></head>
<body>
<h1>Hello, world!</h1>
<p>Request URI: /helloservlet/sayhello</p>
<p>Protocol: HTTP/1.1</p>
<p>PathInfo: null</p>
<p>Remote Address: 127.0.0.1</p>
<p>A Random Number: <strong>0.4320795689818858</strong></p>
</body>
</html>
```

It is important to take note that users receive the *output* of the servlet. User does not receive the servlet's program codes, which are kept under a hidden directory "WEB-INF" and not directly accessible by web users.

**Everything that can possibly go wrong will go wrong...** Read "[Common Error Messages](#)". The likely errors are "404 File Not Found" and "500 Internal Server Error".

## 4. Processing HTML Form Data

### 4.1 Write an HTML Form

HTML provides a `<form>...</form>` tag, which can be used to build a user input form containing elements such as text fields, password field, radio buttons, pull-down menu, checkboxes, text area, hidden field, submit and reset buttons. This allows web users to interact with the web server by submit data. For example,



**User Input Form**

**Personal Particular**

Name:

Password:

Gender: ☐ Male ☐ Female

Age:

**Languages**

☐ Java ☐ C/C++ ☐ C#

**Instruction**

Create the following HTML script, and save as "form\_input.html" under the context root "helloservlet".

```

1<!DOCTYPE html>
2<html>
3<head>
4  <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>
5  <title>User Input Form</title>
6</head>
7
8<body>
9<h2>User Input Form</h2>
10<form method="get" action="echo">
11  <fieldset>
12    <legend>Personal Particular</legend>
13    Name: <input type="text" name="username" /><br /><br />
14    Password: <input type="password" name="password" /><br /><br />
15    Gender: <input type="radio" name="gender" value="m" checked />Male
16            <input type="radio" name="gender" value="f" />Female<br /><br />
17    Age: <select name = "age">
18        <option value="1">&lt; 1 year old</option>
19        <option value="99">1 to 99 years old</option>

```

```

20     <option value="100">&gt; 99 years old</option>
21 </select>
22 </fieldset>
23
24 <fieldset>
25     <legend>Languages</legend>
26     <input type="checkbox" name="language" value="java" checked />Java
27     <input type="checkbox" name="language" value="c" />C/C++
28     <input type="checkbox" name="language" value="cs" />C#
29 </fieldset>
30
31 <fieldset>
32     <legend>Instruction</legend>
33     <textarea rows="5" cols="30" name="instruction">Enter your instruction here...</tex
34 </fieldset>
35
36 <input type="hidden" name="secret" value="888" />
37 <input type="submit" value="SEND" />
38 <input type="reset" value="CLEAR" />
39</form>
40</body>
41</html>

```

Start the tomcat server. Issue the following URL to request for the HTML page:

```
http://localhost:8080/helloservlet/form_input.html
```

### Explanation

- The `<fieldset>...</fieldset>` tag groups related elements and displays them in a box. The `<legend>...</legend>` tag provides the legend for the box.
- This HTML form (enclosed within `<form>...</form>`) contains the following types of input elements:
  1. Text field (`<input type="text">`): for web users to enter text.
  2. Radio buttons (`<input type="radio">`): choose any one (and possibly none).
  3. Pull-down menu (`<select>` and `<option>`): pull-down menu of options.
  4. Checkboxes (`<input type="checkbox">`): chose none or more.
  5. Text area (`<textarea>...<textarea>`): for web users to enter multi-line text. (Text field for single line only.)
  6. Hidden field (`<input type="hidden">`): for submitting hidden name=value pair.
  7. Submit button (`<input type="submit">`): user clicks this button to submit the form data to the server.
  8. Reset button (`<input type="reset">`): resets all the input field to their default value.

Each of the input elements has an attribute "name", and an optional attribute "value". If an element is selected, its "name=value" pair will be submitted to the server for processing.
- The `<form>` start-tag also specifies the URL for submission in the `action="url"` attribute, and the request method in the `method="get|post"` attribute.

For example, suppose that we enter "Alan Smith" in the text field, select "male", and click the "SEND" button, we will get a "404 page not found" error (because we have yet to write the processing script). BUT observe the destination URL:

```
http://localhost:8080/helloservlet/echo?username=Alan+Smith&gender=m&....
```

Observe that:

- The URL `http://localhost:8080/helloservlet/echo` is retrieved from the attribute `action="echo"` of the `<form>` start-tag. *Relative URL* is used in this example. The *base URL* for the current page `"form_input.html"` is `http://localhost:8080/helloservlet/`. Hence, the relative URL `"echo"` resolves into `http://localhost:8080/helloservlet/echo`.
- A `'?'` follows the URL, which separates the URL and the so-called *query string* (or *query parameters*, *request parameters*) followed.
- The query string comprises the `"name=value"` pairs of the *selected* input elements (i.e., `"username=Alan+Smith"` and `"gender=m"`). The `"name=value"` pairs are separated by an `'&'`. Also take note that the blank (in `"Alan Smith"`) is replaced by a `'+'`. This is because special characters are not permitted in the URL and have to be encoded (known as *URL-encoding*). Blank is encoded as `'+'` (or `%20`). Other characters are encoded as `%xx`, where `xx` is the ASCII code in hex. For example, `'&'` as `%26`, `'?'` as `%3F`.
- Some input elements such as checkboxes may trigger multiple parameter values, e.g., `"language=java&language=c&language=cs"` if all three boxes are checked.
- HTTP provides two request methods: GET and POST. For GET request, the query parameters are appended behind the URL. For POST request, the query string is sent in the request message's body. POST request is often preferred, as users will not see the strange string in the URL and it can send an unlimited amount of data. The amount of data that can be sent via the GET request is limited by the length of the URL. The request method is specified in the `<form method="get|post" ... >` start-tag. In this tutorial, we use the GET request, so that you can inspect the query string.

## 4.2 Write a Servlet to Process Form Data - "EchoServlet.java"

The form that we have written sends its data to a server-side program having relative URL of `"echo"` (as specified in the `action="url"` attribute of the `<form>` start-tag). Let us write a servlet called `EchoServlet`, which shall be mapped to the URL `"echo"`, to process the incoming form data. The servlet simply echoes the data back to the client.

Similar to the `"HelloServlet"`, we define the `"EchoServlet"` under package `"mypkg"`, and save the source file as `"<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\EchoServlet.java"`.

```
1// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\EchoServlet.java"
2package mypkg;
3
4import java.io.*;
5import javax.servlet.*;
6import javax.servlet.http.*;
7import java.util.*;
8
9public class EchoServlet extends HttpServlet {
10
11    @Override
```

```

12 public void doGet(HttpServletRequest request, HttpServletResponse response)
13     throws IOException, ServletException {
14     // Set the response message's MIME type
15     response.setContentType("text/html; charset=UTF-8");
16     // Allocate a output writer to write the response message into the network socket
17     PrintWriter out = response.getWriter();
18
19     // Write the response message, in an HTML page
20     try {
21         out.println("<!DOCTYPE html>");
22         out.println("<html><head>");
23         out.println("<meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>");
24         out.println("<title>Echo Servlet</title></head>");
25         out.println("<body><h2>You have enter</h2>");
26
27         // Retrieve the value of the query parameter "username" (from text field)
28         String username = request.getParameter("username");
29         // Get null if the parameter is missing from query string.
30         // Get empty string or string of white spaces if user did not fill in
31         if (username == null
32             || (username = htmlFilter(username.trim())).length() == 0) {
33             out.println("<p>Name: MISSING</p>");
34         } else {
35             out.println("<p>Name: " + username + "</p>");
36         }
37
38         // Retrieve the value of the query parameter "password" (from password field)
39         String password = request.getParameter("password");
40         if (password == null
41             || (password = htmlFilter(password.trim())).length() == 0) {
42             out.println("<p>Password: MISSING</p>");
43         } else {
44             out.println("<p>Password: " + password + "</p>");
45         }
46
47         // Retrieve the value of the query parameter "gender" (from radio button)
48         String gender = request.getParameter("gender");
49         // Get null if the parameter is missing from query string.
50         if (gender == null) {
51             out.println("<p>Gender: MISSING</p>");
52         } else if (gender.equals("m")) {
53             out.println("<p>Gender: male</p>");
54         } else {
55             out.println("<p>Gender: female</p>");
56         }
57
58         // Retrieve the value of the query parameter "age" (from pull-down menu)
59         String age = request.getParameter("age");
60         if (age == null) {

```

```

61         out.println("<p>Age: MISSING</p>");
62     } else if (age.equals("1")) {
63         out.println("<p>Age: &lt; 1 year old</p>");
64     } else if (age.equals("99")) {
65         out.println("<p>Age: 1 to 99 years old</p>");
66     } else {
67         out.println("<p>Age: &gt; 99 years old</p>");
68     }
69
70     // Retrieve the value of the query parameter "language" (from checkboxes).
71     // Multiple entries possible.
72     // Use getParameterValues() which returns an array of String.
73     String[] languages = request.getParameterValues("language");
74     // Get null if the parameter is missing from query string.
75     if (languages == null || languages.length == 0) {
76         out.println("<p>Languages: NONE</p>");
77     } else {
78         out.println("<p>Languages: ");
79         for (String language : languages) {
80             if (language.equals("c")) {
81                 out.println("C/C++ ");
82             } else if (language.equals("cs")) {
83                 out.println("C# ");
84             } else if (language.equals("java")) {
85                 out.println("Java ");
86             }
87         }
88         out.println("</p>");
89     }
90
91     // Retrieve the value of the query parameter "instruction" (from text area)
92     String instruction = request.getParameter("instruction");
93     // Get null if the parameter is missing from query string.
94     if (instruction == null
95         || (instruction = htmlFilter(instruction.trim())).length() == 0
96         || instruction.equals("Enter your instruction here...")) {
97         out.println("<p>Instruction: NONE</p>");
98     } else {
99         out.println("<p>Instruction: " + instruction + "</p>");
100     }
101
102     // Retrieve the value of the query parameter "secret" (from hidden field)
103     String secret = request.getParameter("secret");
104     out.println("<p>Secret: " + secret + "</p>");
105
106     // Get all the names of request parameters
107     Enumeration names = request.getParameterNames();
108     out.println("<p>Request Parameter Names are: ");
109     if (names.hasMoreElements()) {

```

```

110         out.print(htmlFilter(names.nextElement().toString()));
111     }
112     do {
113         out.print(", " + htmlFilter(names.nextElement().toString()));
114     } while (names.hasMoreElements());
115     out.println("</p>");
116
117     // Hyperlink "BACK" to input page
118     out.println("<a href='form_input.html'>BACK</a>");
119
120     out.println("</body></html>");
121 } finally {
122     out.close(); // Always close the output writer
123 }
124 }
125
126 // Redirect POST request to GET request.
127 @Override
128 public void doPost(HttpServletRequest request, HttpServletResponse response)
129     throws IOException, ServletException {
130     doGet(request, response);
131 }
132
133 // Filter the string for special HTML characters to prevent
134 // command injection attack
135 private static String htmlFilter(String message) {
136     if (message == null) return null;
137     int len = message.length();
138     StringBuffer result = new StringBuffer(len + 20);
139     char aChar;
140
141     for (int i = 0; i < len; ++i) {
142         aChar = message.charAt(i);
143         switch (aChar) {
144             case '<': result.append("&lt;"); break;
145             case '>': result.append("&gt;"); break;
146             case '&': result.append("&amp;"); break;
147             case '"': result.append("&quot;"); break;
148             default: result.append(aChar);
149         }
150     }
151     return (result.toString());
152 }
153 }

```

## Dissecting the Program

- The query string comprises name=value pairs. We can retrieve the query parameters from the request message (captured in `doGet()`'s argument `HttpServletRequest request`) via one of the following methods:

- `request.getParameter("paramName")`
- `// Returns the parameter value in a String.`
- `// Returns null if parameter name does not exist.`
- `// Returns the first parameter value for a multi-value parameter.`
- 
- `request.getParameterValues("paramName")`
- `// Return all the parameter values in a String[].`
- `// Return null if the parameter name does not exist.`
- 
- `request.getParameterNames()`
- `// Return all the parameter names in a java.util.Enumeration, possibly empty.`

- Take note that the parameter name is case sensitive.
- We use `request.getParameter("paramName")` to retrieve the parameter value for most of the single-value input elements (such as text field, radio button, text area, etc). If the parameter is present (not null), we `trim()` the returned string to remove the leading and trailing white spaces.
- We also replace the special HTML characters (>, <, &, ") with the HTML escape sequences in the input strings, before we echo them back to the client via `out.println()`. This step is necessary to prevent the so-called *command-injection attack*, where user enters a script into the text field. The replacement is done via a static helper method `htmlFilter()`. [Rule of thumb: Any text string taken from the client and echoing back via `out.println()` needs to be filtered!]
- If the parameter could possess multiple values (e.g., checkboxes), we use `request.getParameterValues()`, which returns an array of `String` or null if the parameter does not exist.
- One of the nice features of Java servlet is that all the form data decoding (i.e., *URL-decoding*) is handled automatically. That is, '+' will be decoded to blank, %xx decoded into the corresponding character.

### 4.3 Configure the Servlet URL mapping in "web.xml"

Our `<form>`'s action attribute refers to relative URL "echo", which has to be mapped to the `EchoServlet.class` in the web application deployment descriptor file "WEB-INF/web.xml":

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">

  <!-- To save as <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml -->

  <servlet>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>mypkg.HelloServlet</servlet-class>
  </servlet>
```

```

<servlet>
    <servlet-name>EchoServletExample</servlet-name>
    <servlet-class>mypkg.EchoServlet</servlet-class>
</servlet>

<!-- Note: All <servlet> elements MUST be grouped together and
      placed IN FRONT of the <servlet-mapping> elements -->

<servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/sayhello</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>EchoServletExample</servlet-name>
    <url-pattern>/echo</url-pattern>
</servlet-mapping>
</web-app>

```

#### 4.4 Run the EchoServlet

Start the Tomcat server. Issue URL `http://localhost:8080/helloservlet/form_input.html`. Fill up the form, click the submit button to trigger the servlet. Alternatively, you could issue a URL with query string.

#### 4.5 Form-Data Submission Methods: GET/POST

Two request methods, GET and POST, are available for submitting form data, to be specified in the `<form>`'s attribute `"method=GET|POST"`. GET and POST performs the same basic function. That is, gather the name-value pairs of the selected input elements, URL-encode, and pack them into a query string. However, in a GET request, the query string is appended behind the URL, separated by a `'?'`. Whereas in a POST request, the query string is kept in the request body (and not shown in the URL). The length of query string in a GET request is limited by the maximum length of URL permitted, whereas it is unlimited in a POST request. I recommend POST request for production, as it does not show the strange looking query string in the URL, even if the amount of data is limited. In this tutorial, I use GET method, so that you can inspect the query string on the URL. To try out the POST request, modify the `"form_input.html"`:

```

<form method="post" action="echo">
    .....
</form>

```

Inside the servlet, GET request is processed by the method `doGet()`, while POST request is processed by the method `doPost()`. Since they often perform identical operations, we redirect `doPost()` to `doGet()` (or vice versa), as follows:

```

public class MyServlet extends HttpServlet {
    // doGet() handles GET request
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        .....
        .....
    }
}

```



```

    }

    // doPost() handles POST request
    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        doGet(request, response); // call doGet()
    }
}

```

## 5. Request Header and Response Header

HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The request and response messages consists of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs. Read "[HTTP Request and Response Messages](#)" for the format, syntax of request and response messages.

### 5.1 *HttpServletRequest*

The request message is encapsulated in an `HttpServletRequest` object, which is passed into the `doGet()` methods. `HttpServletRequest` provides many methods for you to retrieve the headers:

- General methods: `getHeader(name)`, `getHeaders(name)`, `getHeaderNames()`.
- Specific methods: `getContentType()`, `getCookies()`, `getAuthType()`, etc.
- URL related: `getRequestURI()`, `getQueryString()`, `getProtocol()`, `getMethod()`.

Example: Read "[Request Header Example](#)".

### 5.2 *HttpServletResponse*

The response message is encapsulated in the `HttpServletResponse`, which is passed into `doGet()` by reference for receiving the servlet output.

- `setStatusCode(int statuscode)`, `sendError(int code, String message)`, `sendRedirect(url)`.
- `response.setHeader(String headerName, String headerValue)`.
- `setContentType(String mimeType)`, `setContentLength(int length)`, etc.

Example: [TODO]

## 6. Session Tracking

HTTP is a *stateless* protocol. In other words, the current request does not know what has been done in the previous requests. This creates a problem for applications that runs over many requests, such as online shopping (or shopping cart). You need to maintain a so-called *session* to pass data among the multiple requests.

You can maintain a session via one of these three approaches:

4. **Cookie:** A cookie is a small text file that is stored in the client's machine, which will be send to the server on each request. You can put your session data inside the cookie. The biggest problem in using cookie is clients may disable the cookie.
5. **URL Rewriting:** Passes data by appending a short text string at the end of every URL, e.g., `http://host/path/file.html;jsessionid=123456`. You need to rewrite all the URLs (e.g., the "action" attribute of `<form>`) to include the session data.

6. Hidden field in an HTML form: pass data by using hidden field tag (<input type="hidden" name="session" value="...." />). Again, you need to include the hidden field in all the pages.

For detailed information, read "[HTTP state and session management](#)".

## 6.1 HttpSession

Programming your own session tracking (using the above approaches) is tedious and cumbersome. Fortunately, Java Servlet API provides a session tracking facility, via an interface called `javax.servlet.http.HttpSession`. It allows servlets to:

- View and manipulate information about a session, such as the session identifier, creation time, and last accessed time.
- Bind objects to sessions, allowing user information to persist across multiple user requests.

The procedure is as follows:

21. Check if a session already exists. If so, use the existing session object; otherwise, create a new session object. Servlet API automates this step via the `getSession()` method of `HttpServletRequest`:

```
22. // Retrieve the current session. Create one if not exists
23. HttpSession session = request.getSession(true);
24. HttpSession session = request.getSession(); // same as above
25.
26. // Retrieve the current session.
27. // Do not create new session if not exists but return null
```

```
HttpSession session = request.getSession(false);
```

The first statement returns the existing session if exists, and create a new `HttpSession` object otherwise. Each session is identified via a session ID. You can use `session.getID()` to retrieve the session ID string. `HttpSession`, by default, uses cookie to pass the session ID in all the client's requests within a session. If cookie is disabled, `HttpSession` switches to URL-rewriting to append the session ID behind the URL. To ensure robust session tracking, all the URLs emitted from the server-side programs should pass thru the method `response.encodeURL(url)`. If cookie is used for session tracking, `encodeURL(url)` returns the `url` unchanged. If URL-rewriting is used, `encodeURL(url)` encodes the specified `url` by including the session ID.

28. The session object maintains data in the form of *key-value* pairs. You can use `session.getAttribute(key)` to retrieve the *value* of an existing *key*, `session.setAttribute(key, value)` to store new *key-value* pair, and `session.removeAttribute(key)` to remove an existing *key-value* pair. For example,

```
29. // Allocate a shopping cart (assume to be a list of String)
30. List<String> shoppingCart = new ArrayList<>();
31. // Populate the shopping cart
32. shoppingCart.add("Item 1");
33. ....
34. // Retrieve the current session, create one if not exists
35. HttpSession session = request.getSession(true);
36. // Place the shopping cart inside the session
37. synchronized (session) { // synchronized to prevent concurrent updates
```

```
38. session.setAttribute("cart", shoppingCart);
39. }
```

```
.....
```

Any page within the session can retrieve the shopping cart:

```
// Retrieve the current session, do not create new session
HttpSession session = request.getSession(false);
if (session != null) {
    List<String> theCart = (List<String>)session.getAttribute("cart");
    if (theCart != null) { // cart exists?
        for (String item : theCart) {
            .....
        }
    }
}
```

40. You can use `session.invalidate()` to terminate and remove a session. You can use `setMaxInactiveInterval()` and `getMaxInactiveInterval()` to set and get the inactive interval from the last client request, before the server invalidate the session.

## 6.2 Example

The following servlet demonstrates the use of session, by counting the number of accesses within this session from a particular client. We also use `getSessionID()` to retrieve the session ID, `getSessionCreationTime()` and `getSessionLastAccessedTime()` to get the session creation and last accessed times.

`SessionServlet.java`

```
// To save as "<CATALINA_HOME>\webapps\helloservlet\WEB-INF\src\mypkg\SessionServlet.java"
package mypkg;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class SessionServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // Set the response message's MIME type
        response.setContentType("text/html;charset=UTF-8");
        // Allocate a output writer to write the response message into the network
        socket
        PrintWriter out = response.getWriter();

        // Return the existing session if there is one. Create a new session
        otherwise.
        HttpSession session = request.getSession();
        Integer accessCount;
        synchronized(session) {
```

```

        accessCount = (Integer)session.getAttribute("accessCount");
        if (accessCount == null) {
            accessCount = 0;    // autobox int to Integer
        } else {
            accessCount = new Integer(accessCount + 1);
        }
        session.setAttribute("accessCount", accessCount);
    }

    // Write the response message, in an HTML page
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head><meta    http-equiv='Content-Type'    content='text/html; charset=UTF-8'>");
        out.println("<title>Session Test Servlet</title></head><body>");
        out.println("<h2>You have access this site " + accessCount + " times in this session.</h2>");
        out.println("<p>(Session ID is " + session.getId() + ")</p>");

        out.println("<p>(Session creation time is " +
            new Date(session.getCreationTime()) + ")</p>");
        out.println("<p>(Session last access time is " +
            new Date(session.getLastAccessedTime()) + ")</p>");
        out.println("<p>(Session max inactive interval is " +
            session.getMaxInactiveInterval() + " seconds)</p>");

        out.println("<p><a            href='" + request.getRequestURI() +
            "'>Refresh</a>");
        out.println("<p><a    href='" + response.encodeURL(request.getRequestURI()) +
            "'>Refresh with    URL rewriting</a>");
        out.println("</body></html>");
    } finally {
        out.close();    // Always close the output writer
    }
}
}

```

web.xml

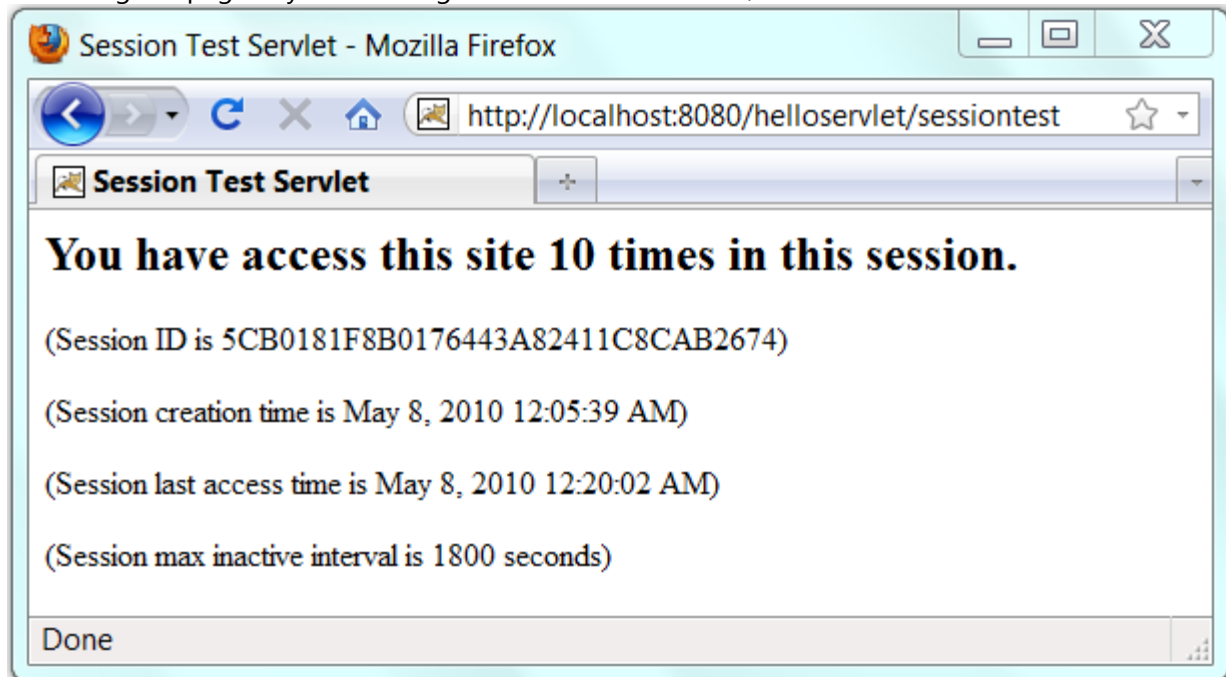
```

.....
<servlet>
    <servlet-name>SessionTestServlet</servlet-name>
    <servlet-class>mypkg.SessionServlet</servlet-class>
</servlet>
.....
.....
<servlet-mapping>
    <servlet-name>SessionTestServlet</servlet-name>
    <url-pattern>/sessiontest</url-pattern>
</servlet-mapping>

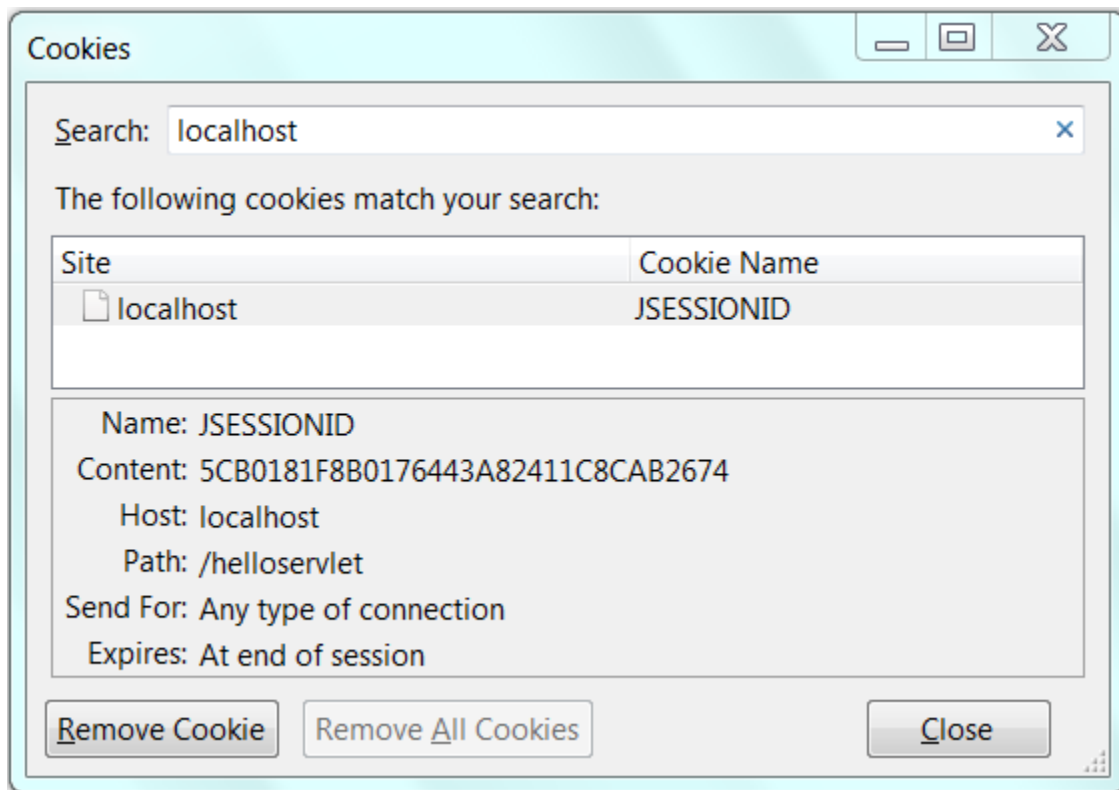
```

## Running the Servlet

You can use URL <http://localhost:8080/helloservlet/sessiontest> to access this servlet. Try refreshing the page. Try also closing and restart the browser, and issue the URL.



Under Firefox, a cookie named `jsessionid` is created for this session. The value of the cookie is the same as the return value of `session.getID()`. By default, Servlet API uses a cookie for managing session, but will automatically switch into URL rewriting if cookie is disabled. To ensure robust session tracking, all the URLs emitted from the server-side programs should pass thru the method `response.encodeURL(url)`. If cookie is used for session tracking, `encodeURL(url)` returns the `url` unchanged. If URL-rewriting is used, `encodeURL(url)` encodes the specified `url` by including the session ID. The session data are kept in the server, only a session ID is passed to the client.



Try disabling the cookie, and use (a) the refresh button (F5), (b) refresh and clear cache (Ctrl-F5), (c) the refresh link, and (d) the refresh with URL re-writing, to refresh the page.

## 7. ServletConfig and ServletContext

### ServletConfig

ServletConfig is a servlet configuration object used by a servlet container (e.g., Tomcat, GlassFish) to pass information to a servlet during *initialization*. It is passed as the argument in the `init()` method. The init parameters are declared in the application-specific deployment descriptor "web.xml". You can retrieve the init parameters via `ServletConfig.getInitParam("paramName")` method. For example, suppose the application's "web.xml" declares these initialization parameters about database connection:

```
<web-app ...>
...
<servlet>
...
  <init-param>
    <param-name>databaseURL</param-name>
    <param-value>jdbc:mysql://localhost:3306/ebookshop</param-value>
  </init-param>
  <init-param>
    <param-name>user</param-name>
    <param-value>myuser</param-value>
  </init-param>
  <init-param>
    <param-name>password</param-name>
```

```

        <param-value>xxxx</param-value>
    </init-param>
</servlet>
...
</web-app>

```

You can retrieve the init parameters in the servlet's `init()` method, as follow:

```

@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    // Read the init params and save them in web context for use by
    // servlets and JSP within this web app.
    ServletContext context = config.getServletContext();
    context.setAttribute("databaseURL", config.getInitParameter("databaseURL"));
    context.setAttribute("user", config.getInitParameter("user"));
    context.setAttribute("password", config.getInitParameter("password"));
    .....
}

```

## ServletContext

Each *webapp* is represented in a single *context* within the servlet container (such as Tomcat, Glassfish). In Servlet API, this context is defined in `javax.servlet.ServletContext` interface (a better name is probably `WebappContext`). A webapp may use many servlets. Servlets deployed in the same webapp can share information between them using the shared `ServletContext` object. There is one `ServletContext` per webapp (or web context). It can be retrieved via `ServletConfig.getServletContext()`. A servlet can use it to communicate with its servlet container (e.g., Tomcat, Glassfish), for example, to get the MIME type of a file, dispatch requests, or write to a log file. `ServletContext` has an "application" scope, and can also be used to pass information between servlets and JSPs within the same application, via methods `setAttribute("name", object)` and `getAttribute("name")`.

Example [TODO]

## 8. Developing and Deploying Web Applications using IDE

It is a lot more productive and efficient to use an IDE (such as Eclipse or NetBeans) to develop your web application. You could start/stop your servers from IDE directly. You could debug your web application in IDE, like debugging standalone application.

**NetBeans:** Read "[Developing and Deploying Web Applications in NetBeans](#)".

**Eclipse:** Read "[Developing and Deploying Web Applications in Eclipse](#)".

## 9. Tomcat's Servlet Examples

Tomcat provides a number of *excellent* servlet examples in "`<CATALINA_HOME>\webapps\examples`". The servlet source files are kept under "`<CATALINA_HOME>\webapps\examples\WEB-INF\classes`", together with the compiled classes. To run the examples, start Tomcat server and issue URL `http://localhost:8080/examples`.

I strongly encourage you to study the examples, Read "[Tomcat's Java Servlet Examples Explained](#)".

## 10. Database Servlet

Read "[Java Servlet Case Study](#)" and "[Java Servlet Case Study Continue](#)".

## 11. Servlet API – A Deeper Look

A servlet is a Java web component, managed by a servlet container (such as Apache Tomcat or Glassfish), which generates dynamic content in response to client's request. A servlet container (or servlet engine) is a web server extension which provides servlet functionality. A servlet container contains and manages servlets throughout their life cycle.

### 11.1 Interface Servlet

The Servlet interface is the central abstraction of the Java servlet API. `HttpServlet` - the most commonly servlet which handles HTTP requests, is a subclass of `GenericServlet` which implements Servlet interface.

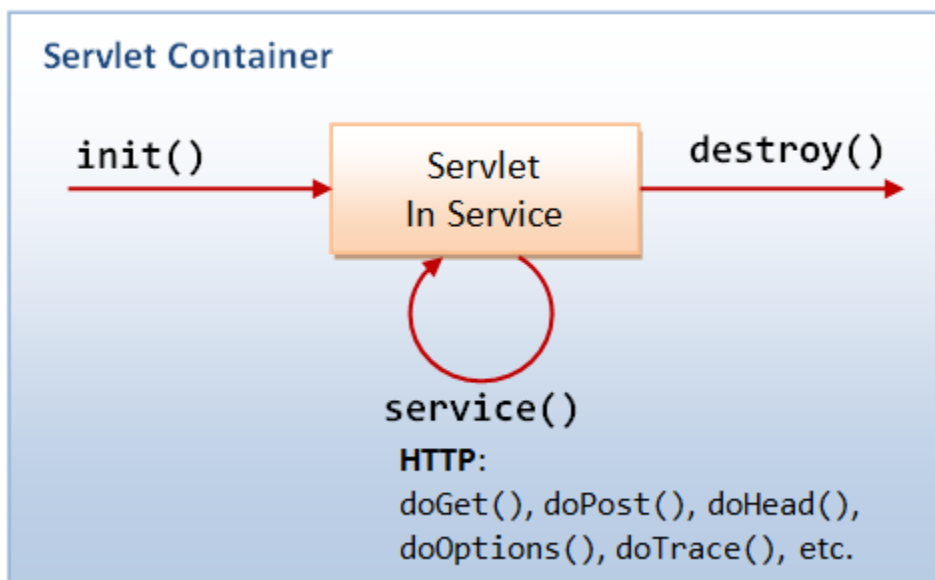
The Servlet interface declares these abstract methods:

```
// Servlet's lifecycle
void init(ServletConfig config)
void destroy()
void service(ServletRequest request, ServletResponse response)

// Servlet configuration and information
ServletConfig getServletConfig()
String getServletInfo()
```

### 11.2 A Servlet's Life cycle

A servlet's life cycle is managed via the `init()`, `service()` and `destroy()` methods.



#### Loading and Initialization

Servlet container (e.g., Tomcat or Glassfish) is responsible for loading and instantiating servlets. It may load and instantiate servlets when it is started, or delay until it determines that the servlet is needed to service a request (usually at the first request to the servlet).



The servlet container invokes the `init(ServletConfig)` method of the servlet, providing a `ServletConfig` object as an argument. `init()` runs only once. It is usually used to read persistent configuration data and initialize costly resource.

This `ServletConfig` object allows the servlet to access *initialization parameters* for this particular servlet. These parameters are defined in the *web application deployment descriptor file* (i.e., "web.xml"), under the servlet's name, as follows:

```
<servlet>
  <servlet-name>ServletName</servlet-name>
  <servlet-class>ServletClassFile</servlet-class>
  <init-param>
    <param-name>initParam1</param-name>
    <param-value>initParam1Value</param-value>
  </init-param>
  <init-param>
    <param-name>initParam2</param-name>
    <param-value>initParam2Value</param-value>
  </init-param>
</servlet>
```

The `ServletConfig` interface defines these methods to retrieve the initialization parameters for this servlet.

```
String getInitParameter(String name)
java.util.Enumeration getInitParameterNames()
```

For example,

```
public void init(ServletConfig config) throws ServletException {
    // Read all the init parameters for this servlet
    Enumeration e = config.getInitParameterNames();
    while (e.hasMoreElements()) {
        String initParamName = (String)e.nextElement();
        String initParamValue = config.getInitParameter(initParamName);
        .....
    }
}
```

The `ServletConfig` interface is implemented by `HTTPServlet` and `GenericServlet`. Hence, the `getInitParameter()` and `getInitParameterNames()` method can be called directly within `init()` or `service()`.

The `ServletConfig` also gives servlet access to a `ServletContext` object that provides information about this web context (aka web application). `ServletContext` will be discussed later.

### In Service

Once a servlet is initialized, the servlet container invokes its `service()` method to handle client requests. This method is called once for each request. Generally, the servlet container handle concurrent request to the same servlet by running `service()` on different threads (unless `SingleThreadModel` interface is declared).

For `HttpServlet`, `service()` dispatches `doGet()`, `doPost()`, `doHead()`, `doOptions()`, `doTrace()`, etc, to handle HTTP GET, POST, HEAD, OPTIONS, TRACE, etc, request respectively.

The `service()` method of an `HttpServlet` takes two arguments, an `HttpServletRequest` object and an `HttpServletResponse` object that corresponds to the HTTP request and response messages respectively.

End of Service

When the servlet container decides that a servlet should be removed from the container (e.g., shutting down the container or time-out, which is implementation-dependent), it calls the `destroy()` method to release any resource it is using and save any persistent state. Before the servlet container calls the `destroy()`, it must allow all `service()` threads to complete or time-out.

### 11.3 Interface *ServletContext*

The `ServletContext` interface defines a servlet's view of the webapp (or web context) in which it is running (a better name is actually `ApplicationContext`). Via the `ServletContext` object, a servlet can communicate with the container, e.g., write to event log, get the URL reference to resources, and get and set attributes that other servlets in the same context can access.

There is one `ServletContext` object for each web application deployed into a container. You can specify initialization parameters for a web context (that are available to all the servlet under the web context) in the web application deployment descriptor, e.g.,

```
<web-app .....>
  <context-param>
    <param-name>jdbcDriver</param-name>
    <param-value>com.mysql.jdbc.Driver</param-value>
  </context-param>
  <context-param>
    <param-name>databaseUrl</param-name>
    <param-value>jdbc:mysql://localhost/eshop</param-value>
  </context-param>
  .....
</web-app>
```

Servlets under this web context can access the context's initialization parameters via the `ServletConfig`'s methods:

```
// ServletConfig
String getInitParameter(String name)
java.util.Enumeration getInitParameterNames()
```

A servlet can bind an attribute of name-value pair into the `ServletContext`, which will then be available to other servlet in the same web application. The methods available are:

```
// ServletContext
Object getAttribute(String name)
void setAttribute(String name, Object value)
void removeAttribute(String name)
java.util.Enumeration getAttributeNames()
```

Other methods in `ServletContext` are:

```
// Write message to event log
```

```

void log(String message)
// Get container info
String getServerInfo()
int getMajorVersion()
int getMinorVersion()

```

The ServletContext provides direct access to static content of the web application (such as HTML, GIF files), via the following methods:

```

java.net.URL getResource(String path)
java.io.InputStream getResourceAsStream(String path)

```

## 11.4 Dispatch Request - RequestDispatcher

When building a web application, it is often useful to forward a request to another servlet, or to include the output of another servlet in the response. The RequestDispatcher interface supports these. The RequestDispatcher can be obtained via ServletContext:

```

// ServletContext
RequestDispatcher getRequestDispatcher(String servletPath)
RequestDispatcher getNamedDispatcher(String servletName)

```

Once the servlet obtained a RequestDispatcher of another servlet within the same web application, it could include or forward the request to that servlet, e.g.,

```

RequestDispatcher rd = context.getRequestDispatcher("/test.jsp?isbn=123");
rd.include(request, response);
// or
rd.forward(request, response);

```

## 11.5 Filtering

A filter is a reusable piece of code that can transform the content of HTTP requests, responses, and header information. Examples of filtering components are:

- Authentication filters
- Logging and auditing filters
- Image conversion filters
- Data compression filters
- Encryption filters
- Tokenizing filters
- Filters that trigger resource access events
- XSL/T filters that transform XML content
- MIME-type chain filters
- Caching filters

[TODO] more

## 12. Web Application Deployment Descriptor "web.xml"

---

The "web.xml" contains the web application *deployment descriptors*. Tomcat's has a system-wide (global) "web.xml" in "<CATALINA\_HOME>\conf". Each web application has its own "web.xml" in "ContextRoot\WEB-INF", which overrides the global settings. Tomcat monitors web.xml for all web applications and reloads the web application when web.xml changes, if reloadable is set to true.

## 12.1 A Sample "web.xml"

```
1<?xml version="1.0" encoding="ISO-8859-1"?>
2<web-app version="3.0"
3  xmlns="http://java.sun.com/xml/ns/javaee"
4  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee
6
7  <!-- General Description of the web application -->
8    <display-name>Workshop Continue</display-name>
9    <description>We shall continue our e-bookstore...</description>
10
11  <!-- Context initialization parameters -->
12  <!-- Provide the database related parameters -->
13    <context-param>
14      <param-name>jdbcDriver</param-name>
15      <param-value>com.mysql.jdbc.Driver</param-value>
16    </context-param>
17    <context-param>
18      <param-name>databaseUrl</param-name>
19      <param-value>jdbc:mysql://localhost/eshop</param-value>
20    </context-param>
21
22  <!-- Define servlets -->
23    <servlet>
24      <servlet-name>BookQuery</servlet-name>
25      <servlet-class>BookQueryServlet</servlet-class>
26      <init-param>
27        <param-name>popularAuthor</param-name>
28        <param-value>Kelvin Jones</param-value>
29      </init-param>
30    </servlet>
31
32  <!-- Define servlet's URL mapping -->
33    <servlet-mapping>
34      <servlet-name>BookQuery</servlet-name>
35      <url-pattern>/query</url-pattern>
36    </servlet-mapping>
37
38    <session-config>
39      <session-timeout>30</session-timeout>
40    </session-config>
41
42    <mime-mapping>
```

```

43     <extension>pdf</extension>
44     <mime-type>application/pdf</mime-type>
45 </mime-mapping>
46
47 <!-- For directory request -->
48 <welcome-file-list>
49     <welcome-file>index.jsp</welcome-file>
50     <welcome-file>index.html</welcome-file>
51     <welcome-file>index.htm</welcome-file>
52 </welcome-file-list>
53
54 <error-page>
55     <error-code>404</error-code>
56     <location>/404.html</location>
57 </error-page>
58</web-app>

```

## 12.2 Syntax for "web.xml"

### Servlets 3.0 "web.xml" Syntax

Tomcat 7 and Glassfish 3.1 supports Servlet 3.0.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    metadata-complete="true">
    .....
</web-app>

```

### Servlets 2.5 "web.xml" Syntax

Tomcat 6 and Glassfish 3 supports Servlets 2.5, JSP 2.1 and JSF 2.0.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    .....
</web-app>

```

### Servlets 2.4 "web.xml" Syntax

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"

```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
.....
</web-app>
```

## 12.3 Servlet Deployment Descriptor

To deploy a servlet, you need to write one pair of `<servlet>` and `<servlet-mapping>` elements, with a matching (but arbitrary and unique) `<servlet-name>`. The `<servlet-class>` specifies the fully-qualified name of the servlet class. The `<url-pattern>` specifies the URL. For example,

```
<web-app ...>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servlet-class>mypkg.MyServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ServletName</servlet-name>
    <url-pattern>/MyURL</url-pattern>
  </servlet-mapping>
</web-app>
```

The resultant URL is `http://hostname:port/WebContext/MyURL`.

You can use wildcard `'*'` in the `<url-pattern>` for pattern matching. For example, `/MyURL.*` (which is matched by `/MyURL.html` and etc.), `/MyURL/*` (which is matched by `/MyURL/test`, and etc.)

Always use a custom URL for servlet, as you could choose a short and meaningful URL and include initialisation, parameters, filter, security setting in the deployment descriptor (see the next section).

## 12.4 Servlet Initialization Parameters

You can pass *initialization parameters* in the form of *name-value* pairs into a particular servlet from "web.xml". For example,

```
<web-app ...>
  <servlet>
    <servlet-name>ServletName</servlet-name>
    <servlet-class>mypkg.MyServlet</servlet-class>

    <init-param>
      <param-name>debug</param-name>
      <param-value>>false</param-value>
    </init-param>
    <init-param>
      <param-name>listing</param-name>
      <param-value>>true</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
```

```

    <servlet-name>ServletName</servlet-name>
    <url-pattern>/MyURL</url-pattern>
</servlet-mapping>
</web-app>

```

Inside the servlet, you can retrieve the init parameters via the `ServletConfig` object:

```

package mypkg;
public class MyServlet extends HttpServlet {

    private boolean debug = false, listing = false;

    @Override
    public void init() {
        ServletConfig config = getServletConfig();
        String strDebug = config.getInitParameter("debug");
        if (strDebug.equals("true")) debug = true;
        String strListing = config.getInitParameter("listing");
        if (strListing.equals("true")) listing = true;
    }
    .....
}

```

## 12.5 Application Initialization Parameters

Specified in webapp's "WEB-INF\web.xml", and available to all the servlets under this webapp. You can use the `getInitParameter()` method of `ServletContext` object to retrieve the init parameters.

```

<web-app .....>
    <context-param>
        <param-name>email</param-name>
        <param-value>query@abcde.com</param-value>
    </context-param>
    .....
</web-app>

```

## 12.6 Server-wide Initialization Parameters

Similar to application init parameters, but defined in the global "<CATALINA\_HOME>\conf\web.xml".

```

<context-param>
    <param-name>email</param-name>
    <param-value>query@abcde.com</param-value>
</context-param>

```

Use the `getInitParameter()` method of `ServletContext` object to retrieve the init parameters.

## 12.7 Welcome Page

Specifies the page to be displayed for request to web context root. For example,

```
<web-app ...>

.....

<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>test/index.html</welcome-file>
</welcome-file-list>
</web-app>
```

## 13. Servlet 3.0

Servlet API 3.0 introduces these annotations to simplify deployment in `javax.servlet.annotation` package:

- `@WebServlet`: Define a servlet component
- `@WebInitParam`: Define initialization parameters for a servlet
- `@WebListener`: Define a listener
- `@WebFilter`: Define a filter
- `@MultipartConfig`: For multipart file upload

For example,

```
@WebServlet(
    name = "HelloServletExample",
    urlPatterns = {"/sayhello"},
    initParams = {
        @WebInitParam(name = "param1", value = "value1"),
        @WebInitParam(name = "param2", value = "value2")}
)
public class HelloServlet extends HttpServlet { ..... }
```

The above is equivalent to the following configuration in "web.xml" prior to Servlet 3.0. The web application deployment descriptor "web.xml" has become optional in Servlet 3.0. Instead, the container at run time will process the annotations of the classes in WEB-INF/classes and JAR files in lib directory.

```
// web.xml
<servlet>
  <servlet-name>HelloServletExample</servlet-name>
  <servlet-class>hello.HelloServlet</servlet-class>
  <init-param>
    <param-name>param1</param-name>
    <param-value>value1</param-value>
  </init-param>
  <init-param>
    <param-name>param2</param-name>
    <param-value>value2</param-value>
  </init-param>
```



```

</servlet>

<servlet-mapping>
  <servlet-name>HelloServletExample</servlet-name>
  <url-pattern>/sayhello</url-pattern>
</servlet-mapping>

```

### 13.1 @WebServlet

@WebServlet defines a servlet component and its metadata, with the following attributes:

- `String[] urlPatterns`: An array of `String` declaring the `url-pattern` for `servlet-mapping`. Default is an empty array `{}`.
- `String value`: `urlPatterns`.
- `String name`: `servlet-name`, default is empty string `""`.
- `loadOnStartup`: The load-on-startup order of the servlet, default is `-1`.
- `WebInitParam[] initParams`: The init parameters of the servlet, default is an empty array `{}`.
- `boolean asyncSupported`: Declares whether the servlet supports asynchronous operation mode, default is `false`.
- `String smallIcon`, `String largeIcon`, `String description`: icon and description of the servlet.

Example:

```

@WebServlet("/sayHello")
public class Hello1Servlet extends HttpServlet { ..... }
// One URL pattern

@WebServlet(urlPatterns = {"/sayhello", "/sayhi"})
public class Hello2Servlet extends HttpServlet { ..... }
// More than one URL patterns

```

### 13.2 @WebInitParam

@WebInitParam is Used to declare init params in servlet, with the following attributes:

- `String name` and `String value` (required): Declare the name and value of the init parameter.
- `String description` (optional) description, default empty string `""`.

See the above example.

### 13.3 @WebFilter

@WebFilter defines a filter (which implements `javax.servlet.Filter` interface).

For example, the following filter log the request time for all the requests (`urlPattern="/*"`).

```

1package mypkg;
2
3import java.io.*;
4import java.util.logging.Logger;
5import javax.servlet.*;
6import javax.servlet.annotation.*;
7import javax.servlet.http.*;
8

```

```

9@WebFilter(urlPatterns={"//*"})
10public class RequestTimerFilter implements Filter {
11    private static final Logger logger
12        = Logger.getLogger(RequestTimerFilter.class.getName());
13
14    @Override
15    public void init(FilterConfig config) throws ServletException {
16        logger.info("RequestTimerFilter initialized");
17    }
18
19    @Override
20    public void doFilter(ServletRequest request, ServletResponse response,
21        FilterChain chain)
22        throws IOException, ServletException {
23        long before = System.currentTimeMillis();
24        chain.doFilter(request, response);
25        long after = System.currentTimeMillis();
26        String path = ((HttpServletRequest)request).getRequestURI();
27        logger.info(path + ": " + (after - before) + " msec");
28    }
29
30    @Override
31    public void destroy() {
32        logger.info("RequestTimerFilter destroyed");
33    }
34}

```

### 13.4 @WebListener

@WebListener defines a listener (which extends ServletContextListener, ServletRequestListener or HttpSessionListener). For example,

```

@WebListener()
public class MyContextListener extends ServletContextListener { ..... }

```

### 13.5 @MultipartConfig

For uploading file using multipart/form-data POST Request. Read "[Uploading Files in Servlet 3.0](#)".

## Unit : 4 JSP( JAVA SERVER PAGES)

### Introduction to JSP

#### Introduction

- It stands for **Java Server Pages**.
- It is a server side technology.
- It is used for creating web application.

- It is used to create dynamic web content.
- In this JSP tags are used to insert JAVA code into HTML pages.
- It is an advanced version of Servlet Technology.
- It is a Web based technology helps us to create dynamic and platform independent web pages.
- In this, Java code can be inserted in HTML/ XML pages or both.
- JSP is first converted into servlet by JSP container before processing the client's request.

### **JSP pages are more advantageous than Servlet:**

- They are easy to maintain.
- No recompilation or redeployment is required.
- JSP has access to entire API of JAVA .
- JSP are extended version of Servlet.

### **Features of JSP**

- **Coding in JSP is easy** :- As it is just adding JAVA code to HTML/XML.
- **Reduction in the length of Code** :- In JSP we use action tags, custom tags etc.
- **Connection to Database is easier** :-It is easier to connect website to database and allows to read or write data easily to the database.
- **Make Interactive websites** :- In this we can create dynamic web pages which helps user to interact in real time environment.
- **Portable, Powerful, flexible and easy to maintain** :- as these are browser and server independent.
- **No Redeployment and No Re-Compilation** :- It is dynamic, secure and platform independent so no need to re-compilation.
- **Extension to Servlet** :- as it has all features of servlets, implicit objects and custom tags

### **JSP syntax**

Syntax available in JSP are following

1. **Declaration Tag** :-It is used to declare variables.

**Syntax:-**

```
<%! Dec var %>
```

**Example:-**

```
<%! int var=10; %>
```

2. **Java Scriptlets** :- It allows us to add any number of JAVA code, variables and expressions.

**Syntax:-**

```
<% java code %>
```

3. **JSP Expression** :- It evaluates and convert the expression to a string.

**Syntax:-**

```
<%= expression %>
```

**Example:-**

```
<% num1 = num1+num2 %>
```

4. **JAVA Comments** :- It contains the text that is added for information which has to be ignored.

**Syntax:-**

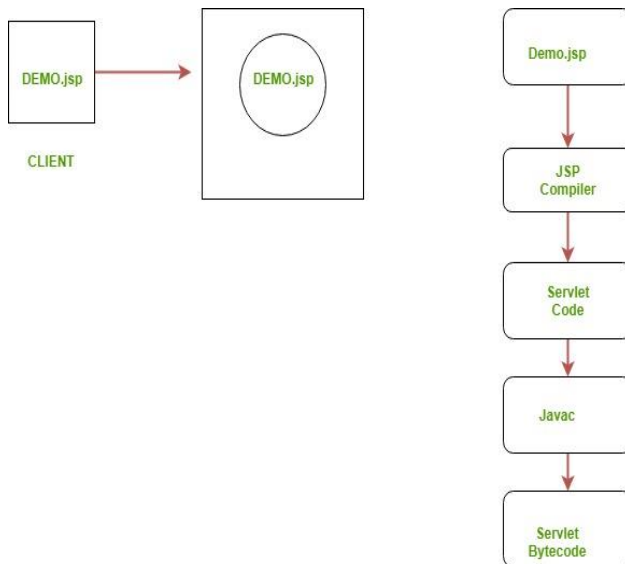
```
<% -- JSP Comments %>
```

### **Process of Execution**

Steps for Execution of JSP are following:-

- Create html page from where request will be sent to server eg try.html.

- To handle to request of user next is to create .jsp file Eg. new.jsp
- Create project folder structure.
- Create XML file eg my.xml.
- Create WAR file.
- Start Tomcat
- Run Application



### Example of Hello World

We will make one .html file and .jsp file

#### demo.jsp

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World - JSP tutorial</title>
</head>
<body>
    <%= "Hello World!" %>
</body>
</html>

```

### Advantages of using JSP

- It does not require advanced knowledge of JAVA
- It is capable of handling exceptions
- Easy to use and learn
- It can tags which are easy to use and understand
- Implicit objects are there which reduces the length of code
- It is suitable for both JAVA and non JAVA programmer

### Disadvantages of using JSP

- Difficult to debug for errors.
- First time access leads to wastage of time
- It's output is HTML which lacks features.

- Http protocol is a stateless protocol, that means that it can't persist the data. Http treats each request as a new request so every time you will send a request you will be considered as a new user.

In session management whenever a request comes for any resource, a unique token is generated by the server and transmitted to the client by the response object and stored on the client machine as a cookie. We can also say that the process of managing the state of a web based client is through the use of session IDs. Session IDs are used to uniquely identify a client browser, while the server side processes are used to associate the session ID with a level of access. Thus, once a client has successfully authenticated to the web application, the session ID can be used as a stored authentication voucher so that the client does not have to retype their login information with each page request. Now whenever a request goes from this client again the ID or token will also be passed through the request object so that the server can understand from where the request is coming.

Session management can be achieved by :

1. Cookies: cookies are small bits of textual information that a web server sends to a browser and that browser returns the cookie when it visits the same site again. In cookie the information is stored in the form of a name, value pair. By default the cookie is generated. If the user doesn't want to use cookies then it can disable them browser setting.

2. URL rewriting: In URL rewriting we append some extra information on the end of each URL that identifies the session. This URL rewriting can be used where a cookie is disabled. It is a good practice to use URL rewriting. In this session ID information is embedded in the URL, which is received by the application through Http GET requests when the client clicks on the links embedded with a page.

Example : <http://www.techfaq360.com/answers.jsp?sname=test>

3. Hidden form fields: In hidden form fields the html entry will be like this : `<input type="hidden" name="name" value="">`. This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and

submitted with the Http POST method.

4.HttpSessionobject: javax.servlet.http.HttpSession is an interface that provides a way to identify a user across more than one page request or visit to a web site. This is the way mainly used in webapplication. HttpSession object maintain session for you. You don't need to do any sessionmanagement.

```
session.setAttribute("name",name);  
Stringname=session.getAttribute("name");  
you will get the same value which you have set.
```

- 

## Maintaining Session Between Web Client And Server

Let us now discuss a few options to maintain the session between the Web Client and the Web Server –

### Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

### Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows –

```
<input type = "hidden" name = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** or the **POST** data. Each time the web browser sends the request back, the **session\_id** value can be used to keep the track of different web browsers.

This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

### URL Rewriting

You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.

For example, with **http://tutorialspoint.com/file.htm;sessionid=12345**, the session identifier is attached as **sessionid = 12345** which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

## The session Object

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

- a one page request or
- visit to a website or
- store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows –

```
<%@ page session = "false" %>
```

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Here is a summary of important methods available through the session object –

S.No.	Method & Description
1	<b>public Object getAttribute(String name)</b> This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	<b>public Enumeration getAttributeNames()</b> This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	<b>public long getCreationTime()</b> This method returns the time when this session was created, measured in milliseconds since

	midnight January 1, 1970 GMT.
4	<b>public String getId()</b> This method returns a string containing the unique identifier assigned to this session.
5	<b>public long getLastAccessedTime()</b> This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	<b>public int getMaxInactiveInterval()</b> This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.
7	<b>public void invalidate()</b> This method invalidates this session and unbinds any objects bound to it.
8	<b>public boolean isNew()</b> This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	<b>public void removeAttribute(String name)</b> This method removes the object bound with the specified name from this session.
10	<b>public void setAttribute(String name, Object value)</b> This method binds an object to this session, using the name specified.
11	<b>public void setMaxInactiveInterval(int interval)</b> This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

## Session Tracking Example

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
<%@ page import = "java.io.*,java.util.*" %>
<%
    // Get session creation time.
    Date createTime = new Date(session.getCreationTime());
```



```

// Get last access time of this Webpage.
Date lastAccessTime = new Date(session.getLastAccessedTime());

String title = "Welcome Back to my website";
Integer visitCount = new Integer(0);
String visitCountKey = new String("visitCount");
String userIDKey = new String("userID");
String userID = new String("ABCD");

// Check if this is new comer on your Webpage.
if (session.isNew() ){
    title = "Welcome to my website";
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey, visitCount);
}
visitCount = (Integer)session.getAttribute(visitCountKey);
visitCount = visitCount + 1;
userID = (String)session.getAttribute(userIDKey);
session.setAttribute(visitCountKey, visitCount);
%>

<html>
<head>
    <title>Session Tracking</title>
</head>

<body>
    <center>
        <h1>Session Tracking</h1>
    </center>

    <table border = "1" align = "center">
        <tr bgcolor = "#949494">
            <th>Session info</th>
            <th>Value</th>
        </tr>
        <tr>
            <td>id</td>
            <td><% out.print( session.getId()); %></td>
        </tr>
        <tr>
            <td>Creation Time</td>
            <td><% out.print(createTime); %></td>
        </tr>
        <tr>
            <td>Time of Last Access</td>
            <td><% out.print(lastAccessTime); %></td>
        </tr>
        <tr>
            <td>User ID</td>
            <td><% out.print(userID); %></td>
        </tr>
    </table>

```

```

        <tr>
            <td>Number of visits</td>
            <td><% out.print(visitCount); %></td>
        </tr>
    </table>

</body>
</html>

```

Now put the above code in **main.jsp** and try to access **<http://localhost:8080/main.jsp>**. Once you run the URL, you will receive the following result –

Welcome to my website

### Session Information

Session info	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Now try to run the same JSP for the second time, you will receive the following result.

Welcome Back to my website

### Session Information

info type	value
id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010

Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

## Deleting Session Data

When you are done with a user's session data, you have several options –

- **Remove a particular attribute** – You can call the ***public void removeAttribute(String name)*** method to delete the value associated with the a particular key.
- **Delete the whole session** – You can call the ***public void invalidate()*** method to discard an entire session.
- **Setting Session timeout** – You can call the ***public void setMaxInactiveInterval(int interval)*** method to set the timeout for a session individually.
- **Log the user out** – The servers that support servlets 2.4, you can call **logout** to log the client out of the Web server and invalidate all sessions belonging to all the users.
- **web.xml Configuration** – If you are using Tomcat, apart from the above mentioned methods, you can configure the session time out in web.xml file as follows.

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

The timeout is expressed as minutes, and overrides the default timeout which is 30 minutes in Tomcat.

The **getMaxInactiveInterval( )** method in a servlet returns the timeout period for that session in seconds. So if your session is configured in web.xml for 15 minutes, **getMaxInactiveInterval( )** returns 900.