



# JAVA

Version Dec-2019

## Contents

JAVA .....	1
Contents .....	2
<b>What is java?</b> .....	9
<b>Java Features</b> .....	9
<b>Why we need java?</b> .....	10
<b>What are J2se J2ee and J2me?</b> .....	10
<b>What is JVM?</b> .....	11
<b>What is JRE?</b> .....	11
<b>What is JDK?</b> .....	11
<b>Garbage Collection</b> .....	11
<b>Class Loader</b> .....	12
<b>Byte Code Verifier</b> .....	12
<b>Compiler Process of java</b> .....	12
Structure of java Program .....	13
<b>What is Class?</b> .....	14
<b>What is Encapsulation?</b> .....	14
<b>Information Hiding</b> .....	14
<b>What is Variable?</b> .....	16
<b>What is Constructor?</b> .....	18
Default Constructor.....	19
What is Object?.....	19
Using Command-Line Arguments .....	20
<b>What is Package?</b> .....	20
Importing Packages or how use packages .....	21
Java Data Types .....	22
<b>Java Keywords</b> .....	24
Lexical Issues.....	26
Pass by Value .....	28
<b>this Keyword.</b> .....	28
<b>Control Statements</b> .....	30
Java's Selection Statements .....	30
Arithmetic Operators .....	33
Unary Operators.....	34
Relational Operator.....	34
Conditional Operators.....	35
Type Comparison Operator.....	36
Bitwise and Bit Shift Operators .....	36

Casting Incompatible Types.....	37
<b>Using break with Label .....</b>	41
What is Array? .....	43
<b>One-dimensional array.....</b>	43
Arrays of arrays: .....	44
<b>Class Design.....</b>	45
<b>What is Inheritance?.....</b>	45
<b>The instanceof Operator.....</b>	54
<b>Casting Objects .....</b>	54
<b>The Object Class .....</b>	57
<b>The equals Method.....</b>	58
<b>The toString Method.....</b>	60
<b>What is Wrapper class?.....</b>	60
Features Of the Wrapper Classes .....	
Wrapper Classes : Methods with examples .....	
<b>Autoboxing of Primitive Types .....</b>	61
<b>Static Keyword .....</b>	62
<b>Final Keyword.....</b>	63
final classes .....	
final methods .....	
final variable.....	
<b>Enum Types .....</b>	64
<b>Static import .....</b>	64
<b>Abstract Class: .....</b>	65
<b>What is Interface?.....</b>	65
<b>Different between Interface and Abstract .....</b>	70
<b>Different between Overloading and Overriding .....</b>	70
<b>Object Serialization .....</b>	80
<b>DataInput/Output Streams .....</b>	81
<b>Files and File I/O.....</b>	82
<b>What is Exception? .....</b>	86
<b>Try and Catch statement .....</b>	87
<b>The finally Clause .....</b>	89
<b>Exception Categories .....</b>	91
The finalize( ) Method .....	94
What is Threading? .....	95
Thread Life cycle .....	95
<b>Creating Thread .....</b>	96

<b>Dead Lock .....</b>	105
<b>Thread Interaction.....</b>	105
<b>Thread State diagram with wait and notify.....</b>	106
<b>Painting in AWT.....</b>	125
<b>Swing .....</b>	136
<b>Event Handling in Java .....</b>	136
<b>Delegation Event Model .....</b>	136
<b>Event Adapters .....</b>	140
Why Use Nested Classes?.....	141
<b>Event Handling Using Anonymous Classes .....</b>	142
<b>JDBC Architecture.....</b>	144
<b>Web Technologies In Java .....</b>	151
Designing .....	151
Html .....	
<b>Client Side Scripting in Java.....</b>	
JavaScript.....	
Event .....	
Mouse Events.....	
JavaScript Validation .....	
<b>Introduction of Client Server Architecture .....</b>	
<b>HTTP Overview .....</b>	
<b>J2EE Architecture.....</b>	
<b>Web Components Development.....</b>	
Common Gateway Interface(CGI).....	
<b>Servlet Programming Process .....</b>	
<b>Servlets .....</b>	
Introduction .....	
Servlet versions .....	
Types of Servlet .....	
Creating Servlet.....	
Servlet Entry in Web.xml.....	
ServletConfig and ServletContext Interface.....	
ServletContext Interface .....	
ServletConfig Interface.....	
Web application Listener .....	
RequestDispatcher Interface.....	
<b>Filters .....</b>	
Why we are using Filter? .....	

Filter Interface.....	
FilterChain Interface .....	
<b>JSP (Java Server Pages) .....</b>	
JSP Page Translation .....	
JSP Life Cycle .....	
JSP Comments.....	
JSP Directives.....	
JSP Scriptlets .....	
JSP Expression .....	
JSP Declaration.....	
JSP Implicit Object.....	
JSP Action .....	
List of JSP Actions .....	
JSTL.....	
JSP Custom tags.....	
<b>Applicability to Industry .....</b>	198
Session Management .....	
What we need for Session Management.....	
Session Tracking Technique .....	
Using Cookies .....	
Using HttpSession.....	
<b>Design Patterns.....</b>	
• Distributed Applications in Java .....	
• Client-Server:.....	
<b>EJB (Enterprise Java Beans) .....</b>	
Web Services.....	
Types of Web-Service.....	
Soap Web Service .....	
RESTful Web Service .....	
Restful WebServices Introduction.....	
Restful Web Services Annotations .....	
@PUT .....	
@DELETE .....	
@Produces .....	
@Consumes .....	
Restful WebService Example.....	
<b>Firebase .....</b>	232

<b>Hibernate</b>	237
<b>Introduction</b>	
What is ORM?	
Hibernate Architecture	
Hibernate Configuration	
Hibernate Configuration Mapping	
Hibernate Bean Mapping	
<b>Hibernate Example</b>	
<b>ORM</b>	
One-to-One Tag Properly	
Many-to-One Tag Property	
One-to-Many Tag Property	
Many -to-Many Tag Property	
<b>Hibernate Query Language</b>	
<b>Pros of Using Spring</b>	
Inversion of Control (IoC) and Dependency Injection (DI)	
Aspect Oriented Programming (AOP)	
Spring Core Container	
Data Access/Integration	
Web	
Misc	
<b>Spring IOC Containers</b>	265
Difference between BeanFactory and the ApplicationContext	
Using BeanFactory	
Using ApplicationContext	
<b>Spring Hello World using IDE</b>	267
Create the Java Project	
Add spring jar files	
Create Java class	
Create the xml file	
Create the Main Class using BeanFactory	
Create the Main Class using ApplicationContaxt	
Output of the Program	
Spring Configuration Metadata	
<b>Spring Life Cycle</b>	
Program	
<b>Inheritance</b>	
Program	

<b>Bean Scope</b>	.....
The Singleton Scope	.....
Program	.....
<b>The Prototype Scope</b>	.....
Program	.....
<b>Spring Dependency Injections</b>	..... 278
<b>Dependency Injection By Constructor Based</b>	.....
Program	.....
Dependency Injection By Setter-Getter Based	.....
Program	.....
<b>Dependency Injection By Object Based</b>	.....
<b>Inner Beans Alias and IdRef</b>	.....
Program using Constructor	.....
Program using Setter-Getter	.....
<b>Collection and References</b>	.....
Program	.....
<b>Spring Auto-Wiring</b>	.....
Program - byName	.....
Program – byType	.....
Program - constructor	.....
<b>Spring ORM</b>	..... 300
<b>Step to Create Program</b>	.....
Program Based on Above Steps	.....
<b>Spring MVC</b>	..... 311
<b>Request flow of Spring MVC</b>	.....
<b>Framework Initiation/Configuration</b>	.....
<b>Spring MVC Hello World using IDE</b>	..... 314
<b>Spring configuration with Eclipse</b>	.....
<b>Step to create Spring MVC practical</b>	.....
Program	.....
<b>Spring Page Redirections</b>	.....
Spring Static Page Redirection	.....
<b>Spring MVC Web Forms</b>	..... 328
Spring Form Handling	.....
Spring Form Tags with Validation	.....
<b>Spring MVC with Session Management</b>	..... 342
<b>Spring CRUD Operation using JDBC</b>	.....
<b>Spring CRUD Operation using Hibernate ORM</b>	.....

<b>Spring CRUD Operation Output .....</b>	.....
Application Deployment on Cloud .....	..... 386
Cloud Computing Models .....	.....
Deployment of Cloud Services .....	.....
<b>Steps for Cloud Application Deployment.....</b>	.....
Start configuring and deploying in IDE.....	.....

## Java Language Fundamentals

### What is java?

- Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems' Java platform.
- The language derives much of its syntax from C and C++. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture.
- Java is a general-purpose, concurrent, class-based, object-oriented language that is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere."
- Java is currently one of the most popular programming languages in use, particularly for client-server web applications

### Java Features

**Simple:** - Java was developed by taking the best points from other programming languages, primarily C and C++. Java therefore utilizes algorithms and methodologies that are already proven. Error prone tasks such as pointers and memory management have either been eliminated or are handled by the Java environment automatically rather than by the programmer. Since Java is primarily a derivative of C++ which most programmers are conversant with, it implies that Java has a familiar feel rendering it easy to use.

1. **Object oriented:** - Java is fully Object-oriented programming language. & Object oriented concepts are following.
  - Class
  - Object
  - Inheritance
  - Encapsulation
  - Polymorphism
2. **Portable:-** The feature Write-once-run-anywhere makes the java language portable provided that the system must have interpreter for the JVM. Java also having the standard data size irrespective of operating system or the processor. This feature makes the java as a portable language.
3. **Distributed:-** The widely used protocols like HTTP and FTP are developed in java. Internet programmers can call functions on these protocols and can get access the files from any remote machine on the internet rather than writing codes on their local system.
4. **High performance:-** Java uses native code usage, and lightweight process called threads. In the beginning interpretation of bytecode resulted the performance slow but the advance version of JVM uses the adaptive and just in time compilation technique that improves the performance.
5. **Multithreaded:-** As we all know several features of Java like Secure, Robust, Portable, dynamic etc; you will be more delighted to know another feature of Java which is **Multithreaded**.

Java is also a Multi-threaded programming language. Multithreading means a single program having different threads executing independently at the same time. Multiple threads execute instructions according to the program code in a process or a program. Multithreading works the similar way as multiple processes run on one computer.

Multithreading programming is a very interesting concept in Java. In multithreaded programs not even a single thread disturbs the execution of other thread. Threads are obtained from the pool of available ready to run threads and they run on the system CPUs. This is how Multithreading works in Java which you will soon come to know in details in later chapters.

6. **Robust:-** Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. All of the above features make the java language robust.
7. **Dynamic:-** While executing the java program the user can get the required files dynamically from a local drive or from a computer thousands of miles away from the user just by connecting with the Internet.
8. **Secure:-** Java does not use memory pointers explicitly. All the programs in java are run under an area known as the sand box. Security manager determines the accessibility options of a class like reading and writing a file to the local disk. Java uses the public key encryption system to allow the java applications to transmit over the internet in the secure encrypted form. The bytecode Verifier checks the classes after loading.

## Why we need java?

- Java technology is a high-level programming and a platform independent language.
- Java is designed to work in the distributed environment on the Internet.
- Java is simple, easy to design, easy to write, and therefore easy to compile, debug, and learn than any other programming languages.
- Java has a GUI features that provides you better "look and feel" over the C++ language, moreover it is easier to use than C++ and works on the concept of object-oriented programming model.

## What are J2se J2ee and J2me?

### J2SE

It is the Java 2 Standard Edition that contains your basic core Java classes. This is the one that most people use to write your standard applets and applications.

### J2EE

It is the Java 2 Enterprise Edition and it contains classes that go above and beyond J2SE. In fact, you will need J2SE in order to use many of the classes in J2EE. Some of the things that J2EE provides are server-side classes such as Servlets and EJB's. It also contains Security API, Java Mail API, XML Parsers

etc., Java Messaging Service API, and a few others. A few of these API's have been included in the new J2SE 1.4 version and are now considered standard. The XML API is an example of this.

## J2ME

It is for developers that code to portable devices, such as a palm pilot or a cellular phone. Code on these devices needs to be small in size and take less memory.

## Java Virtual Machine

### What is JVM?

1. It is a platform-independent execution environment that converts the Java bytecode into machine language and executes it. Most programming languages compile source code directly into machine code that is designed to run on a specific operating system, such as Windows or UNIX.
- JVMs are available for many hardware and software platforms. The use of the same bytecode for all JVMs on all platforms allows Java to be described as a "compile once, run anywhere" programming language, as opposed to "write once, compile anywhere", which describes cross-platform compiled languages. Thus, the JVM is a crucial component of the Java platform.

## Java Run Time Environment

### What is JRE?

- Java Runtime Environment contains JVM, class libraries, and other supporting files. It does not contain any development tools such as compiler, debugger, etc.
- Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in JRE. If you want to run any java program, you need to have JRE installed in the system.

### What is JDK?

Java Developer Kit contains tools needed to develop the Java programs.

- **Appletviewer**:-(For viewing java applets) this tool can be used to run and debug Java applets without a web browser
- **Javac**:-Javac means Java Compiler, which converts source code into Java bytecode.
- **Java** :-( java interpreter) the loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler. Now a single launcher is used for both development and deployment. The
- old deployment launcher, jre, no longer comes with Sun JDK, and instead it has been replaced by this new java loader.
- **Javap** :-( java disassembler) the class file disassemble.
- **Javah** :-(produce header files) the C header and stub generator, used to write native methods
- **Java doc** :-(creating html document) the documentation generator, which automatically generates documentation from source code comments
- **Jdb**:-java debugger.

### Garbage Collection

- Allocated memory that is no longer needed should be de-allocated.
- In some languages, such as C++, dynamically allocated objects must be manually released by use of a **delete** operator. Java takes a different approach; it handles de-allocation for you automatically.

- The technique that accomplishes this is called *garbage collection*. It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
- Garbage collection only occurs sporadically (if at all) during the execution of your program. It will not occur simply because one or more objects exist that are no longer used.

Garbage collection has the following characteristics:

- Checks for and frees memory no longer needed.
- Is done automatically.
- Can vary dramatically across JVM implementations.

## Class Loader

1. Loads all classes necessary for the execution of a program.
2. Maintains classes of the local file system in separate *namespaces*

## Byte Code Verifier

1. The code adheres to the JVM specification.
2. The code does not violate system integrity.
3. The code causes no operand stack overflows or underflows.
4. No illegal data conversions have occurred.

## Compiler Process of java

### Create a source file

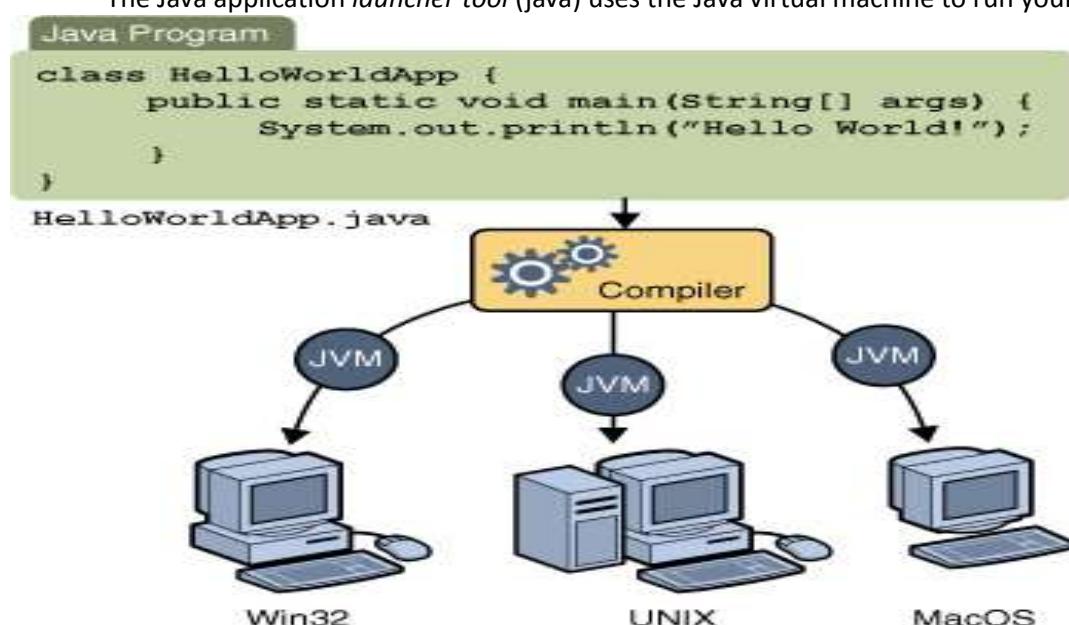
A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use any text editor to create and edit source files. (Or you can also use eclipse or NetBeans)

### Compile the source file into .class file

The Java programming language *compiler* (javac) takes your source file and translates its text into instructions that the Java virtual machine can understand. The instructions contained within this file are known as *bytecode*.

### Run the program

The Java application *launcher tool* (java) uses the Java virtual machine to run your application.



## First Java Program

1. Here name of the class is “HelloWorldApp”

### 2. **Public static void main(String args[])**

**Public**:-The public keyword is access specifier, which means that the content of the following block accessible from all other classes.

**Static**:-The keyword static allows main() to be called without having to instantiate a particular instance of a class.

**Void**:-The keyword void tells the compiler that main () does not return a value.

**main ()**:-main is the method called when a java application begins,  
Java is case-sensitive.

#### **String args []**:-

Declares a parameter named args, which is an array of instance of the class string.

args [] receives any command-line argument present when the program is executed.

**System**:-is a class in java.lang package. The java.lang package is by default imported for source code.

**out**:-is the object of OutputStream class static defines in System class which is responsible for write out put on to a console.

**println ()**:-display the string which is passed do it.

## Structure of java Program

### 1. Document Section :-

In Documentation section one can write author name, definition of class or a program , description of a program and how this algorithm works and other details, and this is OPTIONAL part and this part write in /\* \*/multiline comment.

### 2. Package Statement :-

In java program first statement one can write is a package statement. This statement declares a package name and tells the compiler that the classes included here belongs to this package; the package statement is optional in any java program

For ex:-package first;

### 3. Import Statement:-

Once a package is declared one can write any number of import statements. import statement can be write after package statement and before the class definition.

For ex:-import java.util.\*;

Import first.\*;

### 4. Class Declaration :-

In java program there may be more than one class. In this class definition statement we have to provide keyword class along with the class name. A class can include variable as well as methods.

### 5. Main Method :-

In a java program there may be more than one class one class can have main method. This is the essential part of java program. In main method one can create objects of different classes and with the use of there

objects one can call methods as well as variables of those classes. After reaching the end part of main method, program terminates and control passes back to the operating system.

## Java Classes and OOPS implementation

### What is Class?

A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object.

It intentionally focuses on the basics, showing how even simple classes can cleanly model state and behavior.

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model.

Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an *instance* of the *class of objects* known as bicycles.

A *class* is the blueprint from which individual objects are created.

### Syntax:-

```
<modifier>* class <class_name>
{
    <attribute_declaration>*
    <constructor_declaration>*
    <method_declaration>*
}
```

### Example:-

```
public class Vehicle
{
    private double maxLoad;
    public void setMaxLoad (double value)
    {
        maxLoad = value;
    }
}
```

### What is Encapsulation?

*Encapsulation* is the mechanism that binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

One way to think about encapsulation is as a protective wrapper that prevents the code and data from being arbitrarily accessed by other code defined outside the wrapper.

Access to the code and data inside the wrapper is tightly controlled through a well-defined interface.

Class is the example of encapsulation that bind code and data into single protecting wrapper.

### Information Hiding

It is a mechanism of protecting implementation details of class and providing interface to access properties of a class.

Following example explains you how it is important to use in application.

### Example:

```
class Account
{
    public double amount;
    public Account()
    {
        amount = 0.0;
    }
}
public class Main
{
    public static void main(String[] args)
    {
        Account acc = new Account();
        acc.amount = -100;
    }
}
```

In , above example Account class has amount properties which has public access .

So , in called class amount properties is directly accessed this is happen because we made it public and double type has wide range so it can take any number we can't restrict it .

### Solution:-

```
class Account
{
    private double amount;
    public Account()
    {
        Amount = 0.0;
    }
    public void setAmount(double amt)
    {
        if(amt>0)
        {
            amount = amt;
        }
        Else
        {
            System.out.println("Please, enter positive integer
values only");
        }
    }
    public double getAmount ()
    {
        return amount;
    }
}
```

Here, in solved example we make it amount as private so it will not be accessed outside of the class and provide public method to set an amount. In method we can write our logic and keeps safe , accurate and consistence data.

```
public class Main
{
    public static void main(String[] args)
    {
        Account acc = new Account();
        acc.amount = -100; // it cannot access .
        acc.setAmount(100);
    }
}
```

## Attribute

### What is Attributes?

#### Syntax:-

```
<modifier>* <type> <name> [= <initial_value>];
```

#### Examples:

```
public class Foo
{
    private int x;
    private float y = 10000.0F;
    private String name = "Bates Motel";
}
```

### What is Variable?

1. In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

#### Syntax:-

```
Type identifier [= value] [, identifier][=value].....;
```

#### 2. Instance Variables (Non-static fields):

In object oriented programming, objects store their individual states in the "non-static fields" that is declared without the static keyword. Each object of the class has its own set of values for these non-static variables so we can say that these are related to objects (instances of the class).

Hence these variables are also known as **instance variables**. These variables take default values if not initialized.

#### 3. Class Variables (Static fields):

These are collectively related to a class and none of the object can claim them its sole-proprietor. The variables defined with static keyword are shared by all objects.

Here Objects do not store an individual value but they are forced to share it among themselves. These variables are declared as "**static fields**" using the **static** keyword.

Always the same set of values is shared among different objects of the same class. So these variables are like **global variables** which are same for all objects of the class. These variables take default values if not initialized.

#### 4. Local Variables:

The variables defined in a method or block of code is called **local variables**.

These variables can be accessed within a method or block of code only. These variables don't take default values if not initialized. These values are required to be initialized before using them.

#### 5. Parameters:

Parameters or arguments are variables used in method declarations.

#### Declaring and defining variables

Before using variables you must declare the variables name and type. See the following example for variables declaration:

```
int num;  
        //represents that num is a variable that can store value of int type.  
  
String name;  
        //represents that name is a variable that can store string value.  
  
boolean bol;
```

```
        //represents that bol is a variable that can take boolean value  
        // (true/false);
```

You can assign a value to a variable at the declaration time by using an assignment operator (=).

```
int num = 1000;  
        // This line declares num as an int variable which holds value "1000".  
  
boolean bol = true;  
        // This line declares bol as boolean variable which sets value "true".
```

## Method

### What is Method?

A Java method is a series of statements that perform some repeated task. Instead of writing 10 lines of code we can put those ten lines in a method and just call it in one line.

## Syntax:-

```
[access keywords] [return type] [method name] ([parameters separated by commas])
{
    //statements
}
```

## Example:-

```
Public void Print ()
{
    //statements
}
```

**Modifiers:** - such as public, private, and others you will learn about later.

**The return type:** - The data type of the value returned by the method or void if the method does not return a value.

**The method name:** - The rules for field names apply to method names as well, but the convention is a little different.

**The parameter list:** - Comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, (). If there are no parameters, you must use empty parentheses.

**An exception list:** - To be discussed later.

**The method body:** - The method's code, including the declaration of local variables, goes here.

## Accessing Object Members

The *dot* notation is: **<object>.<member>**

- This is used to access object members, including attributes and methods.
- Examples of dot notation are: d.setWeight(42);  
d.weight = 42; // only permissible if weight is public

## Constructors and Destructors

### What is Constructor?

A **Constructor** initializes an object immediately upon creation.

It has the same name as the class in which it resides and is syntactically similar to a method.

Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes.

Constructors look a little strange because they have no return type, not even **void**.

## Syntax:-

```
[<modifier>] <class_name> (<argument>* )
{
    <statement>*
}
```

## Example:

```
Class Simplecls
{
    double height;
    double width;
    double depth;
    Simplecls()
```

```
{  
    System.out.println("Constructor");  
    width=10;  
    height=10;  
    depth=10;  
}  
double volume()  
{  
    return width*height*depth;  
}  
}  
Class usingconst  
{  
    public static void main(String args[])  
    {  
        Simplecls sd=new Simplecls();  
        double vol= sd.volume();  
        System.out.println("volume is:-"+vol);  
    }  
}
```

Here Simplecls() is constructor and it is called automatically when object of Simplecls is created.

## Default Constructor

1. There is always at least one constructor in every class.
2. If the writer does not supply any constructors, the default constructor is present automatically:
3. The default constructor takes no arguments
4. The default constructor body is empty
5. The default enables you to create object instances with new Xxx() without having to write a constructor.

## Constructor v/s Method

Constructor	Method
1. Constructor has a same name as that of the class.	2. Method is a member function of class which has its own name
3. It does not have any return type.	4. Method has return type specified
5. We have to call constructor by the time of object creation .	6. Method is invoked using dot (.) operator.

## What is Object?

An object is a instance of class. We can also say that object is blue print of the class .

**Syntax :-**

```
Classname objectname = new classname();
```

**Example:**

```
Vehicle v1=new Vehicle();
```

We use the all member of the class. Using object name But we can not use private member of class.

**Ex:-**

```
v1. setMaxLoad(55.55);
```

## Using Command-Line Arguments

Sometimes you will want to pass information into a program when you run it.

This is accomplished by passing *command-line arguments* to **main()**. A command-line argument is the information that directly follows the program's name on the command line when it is executed.

To access the command-line arguments inside a Java program is quite easy—they are stored as strings in the **String** array passed to **main()**.

**Example:**

```
class CommandLine
{
    public static void main(String args[])
    {
        for(int i=0; i<args.length; i++)
        {
            System.out.println("args[" + i + "]: " + args[i]);
        }
    }
}
```

Try executing this program, as shown here:

```
java CommandLine this is a test 100 -1
```

if you use eclipse then go the run menu and click on 'run configuration' and pass command line-argument.

**Output: -**

```
args[0]: this
args[1]: is
args[2]: a
args[3]: test
args[4]: 100
args[5]: -1
```

## Package

### What is Package?

- Java provides a mechanism for partitioning the class name space into more manageable chunks. This mechanism is the package.
- The package is both a naming and a visibility control mechanism. You can define classes inside a package that are not accessible by code outside that package. You can also define class members that are only exposed to other members of the same package. This allows your classes to have intimate knowledge of each other, but not expose that knowledge to the rest of the world.

## Defining a Package

This is the general form of the package statement:

```
package pkg;
```

Here, *pkg* is the name of the package. For example, the following statement creates a package called MyPackage

```
package MyPackage;
```

- More than one file can include the same package statement. The package statement simply specifies to which package the classes defined in a file belong.
- You can create a hierarchy of packages. To do so, simply separate each package name from the one above it by use of a period. The general form of a multilevelled package statement is shown here:

```
package pkg1[.pkg2[.pkg3]];
```

A package hierarchy must be reflected in the file system of your Java development system. For example, a package declared as

```
package java.awt.image;
```

## Importing Packages or how use packages

- In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the import statement:  

```
import pkg1[.pkg2].(classname|*);
```
- Here, *pkg1* is the name of a top-level package, and *pkg2* is the name of a sub ordinate package inside the outer package separated by a dot (.).
- There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Finally, you specify either an explicit *classname* or a star (\*), which indicates that the Java compiler should import the entire package.

### Example:

```
public class Balance
{
    String name;
    double bal;
    public Balance(String n, double b)
    {
        name = n;
        bal = b;
    }
    public void show()
```

```
{  
    if(bal<0)  
    {  
        System.out.print("--> ");  
    }  
    System.out.println(name + ": $" + bal);  
}  
}
```

- As you can see, the Balance class is now public. Also, its constructor and its show( ) method are public, too. This means that they can be accessed by any type of code outside the MyPack package. For example, here TestBalance imports MyPack and is then able to make use of the Balance class:

```
import MyPack.*;  
class TestBalance  
{  
    public static void main(String args[])  
    {  
        Balance test = new Balance("J. J. Jaspers", 99.88);  
        test.show(); // you may also call show()  
    }  
}
```

## Data Type, Identifiers, Keywords

### Java Data Types

#### Primitive Data Types

The primitive data types are **predefined** data types, which always hold the value of the same data type, and the values of a primitive data type don't share the **state** with other primitive values. These data types are named by a **reserved keyword** in Java programming language.

There are **eight primitive data types** supported by Java programming language:

#### byte:

The byte data type is an 8-bit signed two's complement integer. It ranges from -128 to 127 (inclusive). This type of data type is useful to save memory in large arrays. We can also use **byte** instead of **int** to increase the limit of the code. The syntax of declaring a byte type variable is shown as:

```
byte b = 5;
```

#### short:

The short data type is a 16-bit signed two's complement integer. It ranges from -32,768 to 32,767. **short** is used to save memory in large arrays. The syntax of declaring a short type variable is shown as: `short s = 2;`

## **int :**

The int data type is used to store the integer values not the fraction values. It is a 32-bit signed two's complement integer data type. It ranges from -2,147,483,648 to 2,147,483,647 that are more enough to store large number in your program. However for wider range of values use **long**. The syntax of declaring an int type variable is shown as: int n = 50;

## **long :**

The long data type is a 64-bit signed two's complement integer. It ranges from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Use this data type with larger range of values. The syntax of declaring a long type variable is shown as: long ln = 746L;

## **float :**

The float data type is a single-precision 32-bit IEEE 754 floating point. It ranges from 1.40129846432481707e-45 to 3.40282346638528860e+38 (positive or negative). Use a float (instead of double) to save memory in large arrays. We do not use this data type for the exact values such as currency. For that we have to use java.math.BigDecimal class. The syntax of declaring a float type variable is: float f = 105.65f;

## **double :**

This data type is a double-precision 64-bit IEEE 754 floating point. It ranges from 4.94065645841246544e-324d to 1.79769313486231570e+308d (positive or negative). This data type is generally the default choice for decimal values. The syntax of declaring a double type variable is shown as: double d = 6677.60;

## **char :**

The char data type is a single 16-bit, **unsigned** Unicode character. It ranges from 0 to 65,535. They are not integral data type like int, short etc. i.e. the char data type can't hold the numeric values. The syntax of declaring a char type variable is shown as: char c = 'c';

## **boolean:**

The boolean data type represents only two values: **true** and **false** and occupy is **1-bit** in the memory. These values are **keywords** in Java and represent the two boolean states: **on** or **off**, **yes** or **no**. We use **boolean** data type for specifying conditional statements as **if**, **while**, **do**, **for**. In Java, **true** and **false** are not the same as **True** and **False**. They are defined constants of the language. The syntax of declaring a boolean type variable is shown as: boolean r = true;

The ranges of these data types can be described with default values using the following table:

Data Type	Default Value (for fields)	Size (in bits)	Minimum Range	Maximum Range
Byte	0	8 bits	-128	127
Short	0	16 bits	-32768	32767
Int	0	32 bits	-2147483648	2147483647
Long	0L	64 bits	-9.22E+018	9.22E+018
Float	0.0f	32-bit	1.4012984643248170 7e-45	3.40E+038
Double	0.0d	64-bit	4.9406564584124654 4e-324d	1.79769313486231570e+308d
Char	'\u0000'	16-bit		0 to 65,535
boolean	FALSE	1-bit	NA	NA

## Identifiers

Identifiers have the following characteristics:

1. Are names given to a variable, class, or method
2. Can start with a Unicode letter, underscore (\_), or dollar sign (\$)
3. Are case-sensitive and have no maximum length

### Examples:

```
identifier
userName
user_name
_sys_var1
$change
```

## Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Reserved literal words: null, true, and false

## Reference Data Types

In Java a **reference data type** is a variable that can contain the reference or an address of dynamically created object. These types of data type are not predefined like **primitive** data type. The reference data types are **arrays**, **classes** and **interfaces** that are made and handle according to a programmer in a java program which can hold the three kinds of values as:

Class type:

As you know that Java is an object-oriented programming language where an object is a variable, associated with methods that is described by a class. The name of a class is treated as a **type** in a java program, so that you can declare a variable of an object-type, and a method which can be called using that object- type variable.

Whenever a variable is created, a reference to an object is also created using the name of a class for its type i.e. that variable can contain either **null** or a **reference** to an object of that class. It is not allowed to contain any other kinds of values. Such type is called **reference types** in Java. The object becomes an **instance** when the memory is allocated to that object using **new** keyword. In addition, **array types** are **reference types** because these are treated as objects in Java.

Array type:

An array is a special kind of **object** that contains values called **elements**. The java array enables the user to store the values of the same type in **contiguous** memory allocations. The elements in an array are identified by an **integer index** which initially starts from **0** and ends with **one less than number** of elements available in the array. All elements of an array must contain the same type of value i.e. if an array is a type of integer then all the elements must be of integer type. It is **reference data type** because the class named as **Array** implicitly extends **java.lang.Object**.

Interface Type:

Java provides another kind of **reference data type** or a mechanism to support **multiple inheritance** features called an **interface**. The name of an interface can be used to specify the type of a reference. A value is not allowed to be assigned to a variable declared using an interface type until the **object** implements the specified **interface**.

## Lexical Issues

### 1. Whitespace

Java is a free-form language. This means that you do not need to follow any special indentation rules. For example, the Example program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator. In java, whitespace is a space, tab, or new line.

### 2. Identifiers

Identifiers are used for class names, method names, and variable names. An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers or the underscore and dollar sign characters. They must not begin with a number, lest they be confused with a numeric literal. Again, java is case-sensitive, so **VALUE** is a different identifier than the **Value**.

### 3. Comments

- // single line comments  
All characters from // to the end of the line are ignored.
- /\* text \*/ Multiline comments  
All characters from /\* to \*/ are ignored.
- /\*\* text \*/ Documents section comments

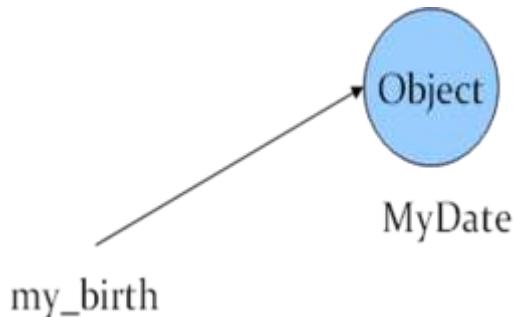
These comments are treated specially when they occur immediately before any declaration. They should not be used any other place in the code. These comments indicate that the enclosed text should be included in automatically generated documentation as a description of the declared item.

### 4. Literals

Literals in java are a sequence of characters(digits, letters and other characters)that represent constant values to be stored in variables.

- Integer Literals
- Character Literals
- Boolean Literals
- Floating point Literals
- String Literals

## Constructing and Initializing Objects



Calling new `Xyz()` performs the following actions:

- a. Memory is allocated for the object.
- b. Explicit attribute initialization is performed.
- c. A constructor is executed.
- d. The object reference is returned by the new operator.
  - The reference to the object is assigned to a variable.
  - An example is: `MyDate my_birth = new MyDate(22, 7, 1964);`

### Memory Allocation and Layout

A declaration allocate storage only for the reference:

```
MyDate my_birth=new MyDate(22,7,1964);
```

Use new operator to allocate space for MyDate:

```
MyDate my_birth=new MyDate(22,7,1964);
```

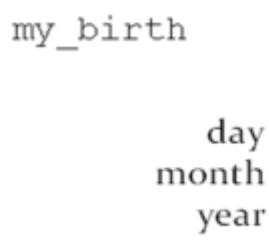
## Memory Allocation and Layout

A declaration allocates storage only for a reference:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

Use the new operator to allocate space for MyDate:

```
MyDate my_birth = new MyDate(22, 7,1964);
```



## Executing the Constructor

- Execute the matching constructor as follows:

```
MyDate my_birth = new MyDate(22, 7, 1964);
```

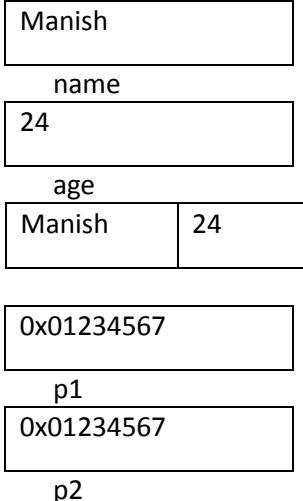
- In the case of an overloaded constructor, the first constructor can call another.

## Assigning References

Assign reference to other object then both object refer to the same point .

```
Person p1 = new Person("Manish".24) ;
```

```
Person p2 = p1;
```



## Pass by Value

In a single virtual machine, the Java programming language only passes arguments by value.

When an object instance is passed as an argument to a method, the value of the argument is a *reference* to the object.

The *contents* of the object can be changed in the called method, but the original object reference is never changed.

## this Keyword

this keyword refers to the current class reference.

Here are three main uses of this keyword :

- To resolve ambiguity between instance variable and parameter.
- To pass current object as a parameter to another method or constructor .
- To call a constructor from another constructor of same class .

### Example :-

```
public class MyDate
{
    private int day = 1;
    private int month = 1;
    private int year = 2000;

    public MyDate(int day, int month, int year)
    {
```

```
        this.day = day;
        this.month = month;
        this.year = year;
    }
    public MyDate(MyDate date)
    {
        this.day = date.day;
        this.month = date.month;
        this.year = date.year;
    }
    public MyDate addDays(int moreDays)
    {
        MyDate newDate = new MyDate(this);
        newDate.day = newDate.day + moreDays;
        // Not Yet Implemented: wrap around code...
        return newDate;
    }
    public String toString()
    {
        return "" + day + "-" + month + "-" + year;
    }
}

public class TestMyDate
{
    public static void main(String[] args)
    {
        MyDate my_birth = new MyDate(22, 7, 1964);
        MyDate the_next_week = my_birth.addDays(7);
        System.out.println(the_next_week);
    }
}
```

## Language Coding Conventions

- Packages:  
com.example.domain;
- Classes, interfaces, and enum types:  
SavingsAccount
- Methods:  
getAccount()
- Variables:  
currentCustomer
- Constants:  
HEAD\_COUNT
- Control structures:  
if ( *condition* ) {  
 *statement1*;  
} else {

- ```
    statement2;
}
• Spacing:
• Use one statement per line.
• Use two or four spaces for indentation.
• Comments:
• Use // to comment inline code.
• Use /** documentation */ for class members.
```

## Expressions and Flow control

### Control Statements

A programming language uses *control* statements to cause the flow of execution to advance and branch based on changes to the state of a program. Java's program control statements can be put into the following categories: selection, iteration, and jump.

#### Java's Selection Statements

Java supports two selection statements: **if** and **switch**. These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

- **If Statements**

This is a control statement to execute a single statement or a block of code, when the given condition is true and if it is false then it skips **if** block and rest code of program is executed .

The **if** statement is Java's conditional branch statement.

**Syntax:-**

```
if ( <boolean_expression> )
    <statement_or_block>
```

**Example:**

```
if ( x < 10 )
    System.out.println("Are you finished yet?");
```

- **If ,else Statements**

The "**if-else**" statement is an extension of if statement that provides another option when 'if' statement evaluates to "false" i.e. else block is executed. if "**if**" statement is false.

**Syntax :-**

```
if ( <boolean_expression> )
    <statement_or_block1>
else
    <statement_or_block2>
```

If the *condition* is true, then **statement\_or\_block1** is executed. Otherwise, **statement\_or\_block2** (if it exists) is executed. In no case will both statements be executed.

**Example:**

```
if ( x < 10 )
{
    System.out.println("Are you finished yet?");
}
else
```

```
{  
    System.out.println("Keep working...");  
}
```

- **If –else-if Ladder**

A common programming construct that is based upon a sequence of nested **ifs** is the **if-else-if ladder**.

**Syntax :-**

```
if ( <boolean_expression> )  
    <statement_or_block>  
else if ( <boolean_expression> )  
    <statement_or_block>  
else if ( <boolean_expression> )  
    <statement_or_block>  
. . .  
else  
    <statement_or_block>
```

**Example:**

```
class result  
{  
    //int marks;  
    public void marks(int marks)  
    {  
        if(marks>=70)  
        {  
            System.out.println("grade is a+");  
        }  
        else if(marks>=60)  
        {  
            System.out.println("grade is a");  
        }  
        else if(marks>=50)  
        {  
            System.out.println("grade is b");  
        }  
        else if(marks>=40)  
        {  
            System.out.println("grade is c");  
        }  
        else  
        {  
            System.out.println("fail");  
        }  
    }  
}
```

```
public class ifelseif
{
    public static void main(String args[])
    {
        result r1=new result();
        r1.marks(50);
    }
}
```

**Output:**

April is in the Spring.

## Variable Scope & Initialization

Local variables are:

1. Variables that are defined inside a method and are called *local, automatic, temporary, or stack* variables
2. Variables that are created when the method is executed are destroyed when the method is exited.

Variable initialization comprises the following:

- Local variables require explicit initialization.
- Instance variables are initialized automatically.

**Example :-**

```
public class ScopeExample
{
    private int i=1;
    public void firstMethod()
    {
        int i=4, j=5;
        this.i = i + j;
        secondMethod(7);
    }
    public void secondMethod(int i)
    {
        int j=8;
        this.i = i + j;
    }
}

public class TestScoping
{
    public static void main(String[] args)
    {
        ScopeExample scope = new ScopeExample();
        scope.firstMethod();
    }
}
```

## Java Operators

### Arithmetic Operators

| Operators | Description                                            |
|-----------|--------------------------------------------------------|
| +         | Additive operator (also used for String concatenation) |
| -         | Subtraction operator                                   |
| *         | Multiplication operator                                |
| /         | Division operator                                      |
| %         | Remainder operator                                     |

**Example:**

```
class ArithmeticOperator
{
    public void Addition(int a, int b)
    {
        System.out.println("Ans is:-"(a+b));
    }
    public void Subtract(int a, int b)
    {
        System.out.println("Ans is:-"(a-b));
    }
    public void Multiply(int a, int b)
    {
        System.out.println("Ans is:-"(a*b));
    }
    public void Divide(int a, int b)
    {
        System.out.println("Ans is:-"(a/b));
    }
    public static void main(String args[])
    {
        ArithmeticOperator a1=new ArithmeticOperator();
        a1.Addtion(10,20);
        a1.Subtract(20,10);
        a1.Multiply(10,10);
        a1.Divide(10,2);
    }
}
```

## Unary Operators

| Operators       | Description                                 |
|-----------------|---------------------------------------------|
| <code>++</code> | Increment operator; increments a value by 1 |
| <code>--</code> | Decrement operator; decrements a value by 1 |
| <code>!</code>  | Logical complement operator                 |

### Example :-

```

class UnaryOperation
{
    public int i=0;
    public void operate()
    {
        int i = +1;
        System.out.println(i);
        i--;
        System.out.println(i);
        i++;
        System.out.println(i);
        i = ++i;
        System.out.println(i);
        boolean success = false;
        System.out.println(!success);
        System.out.println(success);
        return 0;
    }
    public static void main(String args[])
    {
        UnaryOperation uo= new UnaryOperation();
        uo.operate();
    }
}

```

## Relational Operator

| Operators        | Description  |
|------------------|--------------|
| <code>= =</code> | Equal to     |
| <code>!=</code>  | Not equal to |

|     |                          |
|-----|--------------------------|
| >   | Greater than             |
| > = | Greater than or equal to |
| <   | Less than                |
| < = | Less than or equal to    |

### Example:

```

class RelationalOperator
{
    public void Equal(int val1,int val2)
    {
        if(val1 == val2)
        {
            System.out.println("both are equal");
        }
        if(val1!=val2)
        {
            System.out.println("both are not equal");
        }
        if(val1<val2)
        {
            System.out.println("Value1 is less then values2");
        }
        if(val1 > val2)
        {
            System.out.println(val1+" is greater then "+ val2);
        }
    }
    public static void main(String arg[])
    {
        RelationalOperator ro = new RelationalOperator();
        ro.Equal(10,20);
    }
}

```

### Conditional Operators

| Operators | Description                                    |
|-----------|------------------------------------------------|
| &&        | Conditional-AND                                |
|           | Conditional-OR                                 |
| ?:        | Ternary (shorthand for if-then-else statement) |

## Example:

```
class ConditionalOperator
{
    public static void main(String[] args)
    {
        int x = 5;
        int y = 10, result=0;
        boolean bl = true;
        if((x == 5) && (x < y))
        {
            System.out.println("value of x is "+x);
        }
        if((x == y) || (y > 1))
        {
            System.out.println("value of y is greater than the
                               value of x");
        }
        result = bl ? x : y;
        System.out.println("The returned value is "+result);
    }
}
```

## Type Comparison Operator

instanceOf    Compares an object to a specified type

## Bitwise and Bit Shift Operators

| Operators | Description              |
|-----------|--------------------------|
| ~         | Unary bitwise complement |
| <<        | Signed left shift        |
| >>        | Signed right shift       |
| >>>       | Unsigned right shift     |
| &         | Bitwise AND              |
| ^         | Bitwise exclusive OR     |
|           | Bitwise inclusive OR     |

## Casting

## Java's Automatic Conversions

When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:

- The two types are compatible.
- The destination type is larger than the source type.

When these two conditions are met, a *widening conversion* takes place. For example, the **int** type is always large enough to hold all valid **byte** values, so no explicit cast statement is required.

## Casting Incompatible Types

- Although the automatic type conversions are helpful, they will not fulfill all needs.
- For example, what if you want to assign an **int** value to a **byte** variable? This conversion will not be performed automatically, because a **byte** is smaller than an **int**. This kind of conversion is sometimes called a *narrowing conversion*, since you are explicitly making the value narrower so that it will fit into the target type.
- To create a conversion between two incompatible types, you must use a cast. A *cast* is simply an explicit type conversion. It has this general form:

### Syntax:-

```
(target-type) value
int a;
byte b;
b = (byte) a;
```

### Example :-

```
class Conversion
{
    public static void main(String args[])
    {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("\nConversion of int to byte.");
        b = (byte) i;
        System.out.println("i and b " + i + " " + b);
        System.out.println("\nConversion of double to int.");
        i = (int) d;
        System.out.println("d and i " + d + " " + i);
        System.out.println("\nConversion of double to byte.");
        b = (byte) d;
        System.out.println("d and b " + d + " " + b);
    }
}
```

### Output :-

```
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
```

```
d and i 323.142 323  
Conversion of double to byte.  
d and b 323.142 67
```

## Switch Statements

This is an easier implementation to the if-else statements. The keyword "**switch**" is followed by an expression that should evaluate to byte, short, char or int primitive data types , only. In a switch block there can be one or more labeled cases.

The expression that creates labels for the case must be unique. The switch expression is matched with each case label. Only the matched case is executed, if no case matches then the default statement (if present) is executed.The **switch** statement is Java's multi-way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

### Syntax:

```
switch ( <expression> )  
{  
    case <constant1>:  
        <statement_or_block>*  
        [break;]  
    case <constant2>:  
        <statement_or_block>*  
        [break;]  
    default:  
        <statement_or_block>*  
        [break;]  
}
```

### Example:-

```
int day = 5;  
switch (day)  
{  
    case 1: System.out.println("Monday");  
            break;  
    case 2: System.out.println("Tuesday");  
            break;  
    case 3: System.out.println("Wednesday");  
            break;  
    case 4: System.out.println("Thursday");  
            break;  
    case 5: System.out.println("Friday");  
            break;  
    case 6: System.out.println("Saturday");  
            break;  
    case 7: System.out.println("Sunday");  
            break;  
    default: System.out.println("Invalid entry");  
             break;  
}
```

## Looping Statements

Java's iteration statements are **for**, **while**, and **do-while**.

- **while Statements**

The **while** loop is Java's most fundamental looping statement. It repeats a statement or block while its controlling expression is true. Here is its general form:

**Syntax:**

```
while(condition)
{
    // body of loop
}
```

The *condition* can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true.

When *condition* becomes false, control passes to the next line of code immediately following the loop. The curly braces are unnecessary if only a single statement is being repeated.

**Example:**

```
int i = 0;
while ( i < 10 )
{
    System.out.println(i + " squared is " + (i*i));
    i++;
}
```

- **do-while Statements**

The **do-while** loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Its general form is

**Syntax:**

```
do
{
    // body of loop
}
while (condition);
```

Each iteration of the **do-while** loop first executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates

**Example:**

```
int i = 0;
do
{
    System.out.println(i + " squared is " + (i*i));
    i++;
}
while ( i < 10 );
```

- **For Loop**

**Syntax:**

```
for(initialization; condition; iteration)
{
    // body
```

```
}
```

When the loop first starts, the *initialization* portion of the loop is executed. Generally, this is an expression that sets the value of the *loop control variable*, which acts as a counter that controls the loop.

It is important to understand that the initialization expression is only executed once. Next, *condition* is evaluated. This must be a Boolean expression.

It usually tests the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates. Next, the *iteration* portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable.

The loop then iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression with each pass. This process repeats until the controlling expression is false.

**Example:**

```
class FindPrime
{
    public static void main(String args[])
    {
        int num;
        boolean isPrime = true;
        num = 14;
        for(int i=2; i <= num/2; i++)
        {
            if((num % i) == 0)
            {
                isPrime = false;
                break;
            }
        }
        if(isPrime)
            System.out.println("Prime");
        else
            System.out.println("Not Prime");
    }
}
```

## Break Statement

- In Java, the break statement has three uses. First, as you have seen, it terminates a statement sequence in a switch statement. Second, it can be used to exit a loop. Third, it can be used as a “civilized” form of go to.

### Using break to Exit a Loop

- By using break, you can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
- When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

**Example:**

```
public class breakex
{
    public static void main(String[] args)
```

```
{  
    int a;  
    for(a=0;a<10;a++)  
    {  
        if(a==5)  
        {  
            break;  
        }  
        System.out.println(a);  
    }  
}  
Output :  
0,1,2,3,4.
```

## Using break with Label

Most often, *label* is the name of a label that identifies a block of code. This can be a stand-alone block of code but it can also be a block that is the target of another statement.

When this form of **break** executes, control is transferred out of the named block. The labeled block must enclose the **break** statement, but it does not need to be the immediately enclosing block.

This means, for example, that you can use a labeled **break** statement to exit from a set of nested blocks. But you cannot use **break** to transfer control out of a block that does not enclose the **break** statement.

### Example:

```
class BreakLoop  
{  
    public static void main(String args[])  
    {  
        outer: for(int i=0; i<3; i++)  
        {  
            System.out.print("Pass " + i + ": ");  
            for(int j=0; j<100; j++)  
            {  
                if(j == 10) break outer; // exit both loops  
                System.out.print(j + " ");  
            }  
            System.out.println("This will not print");  
        }  
        System.out.println("Loops complete.");  
    }  
}
```

**Output :**  
Pass 0: 0 1 2 3 4 5 6 7 8 9 Loops complete.

## Continue Statement

Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop, but stop processing the remainder of the code in its body for this particular iteration.

**Example:**

```
public class breakex
{
    public static void main(String[] args)
    {
        int a;
        for(a=0;a<10;a++)
        {
            if(a==5)
            {
                continue;
            }
            System.out.println(a);
        }
    }
}
```

**Output :**

0,1,2,3,4,6,7,8,9.

**Using continue with Label**

As with the **break** statement, **continue** may specify a label to describe which enclosing loop to continue.

**Example:**

```
class ContinueLabel
{
    public static void main(String args[])
    {
        outer: for (int i=0; i<10; i++)
        {
            for(int j=0; j<10; j++)
            {
                if(j > i)
                {
                    System.out.println();
                    continue outer;
                }
                System.out.print(" " + (i * j));
            }
            System.out.println();
        }
    }
}
```

**Output :-**

0  
0 1  
0 2 4  
0 3 6 9

```
0 4 8 12 16  
0 5 10 15 20 25  
0 6 12 18 24 30 36  
0 7 14 21 28 35 42 49  
0 8 16 24 32 40 48 56 64  
0 9 18 27 36 45 54 63 72 81
```

## Array

### What is Array?

An *array* is a container object that holds a fixed number of values of a single type.

The length of an array is established when the array is created. After creation, its length is fixed. You've seen an example of arrays already, in the main method of the "Hello World!" application. This section discusses arrays in greater detail.

An array of ten elements

Each item in an array is called an *element*, and each element is accessed by its numerical *index*. As shown in the above illustration, numbering begins with 0. The 9th element, for example, would therefore be accessed at index 8.

### One-dimensional array

#### Syntax:

```
datatype arrayname[] = new datatype[size];
```

#### Example:

```
int myarr = new int[10];
```

Here declare array and name of the array is myarr and its size is 10

## Initializing Arrays

```
String[] names;  
names = new String[3];  
names[0] = "Georgia";  
names[1] = "Jen";  
names[2] = "Simon";
```

### OR

```
String[] names = {"Georgia", "Jen", "Simon"};
```

#### Example:

```
class Average  
{  
    public static void main(String args[])  
    {  
        double nums[] = {10.1, 11.2, 12.3, 13.4, 14.5};  
        double result = 0;  
        int i;  
        for(i=0; i<5; i++)  
            result = result + nums[i];  
        System.out.println("Average is " + result / 5);  
    }  
}
```

}

## Multi-dimensional array

### Syntax:

```
datatype arrayname[][] = new datatype[][];
```

### Example:-

```
int twoD[][] = new int[4][5];
```

This allocates a 4 by 5 array and assigns it to **twoD**. Internally this matrix is implemented as an *array of arrays of int*.

### Example:-

```
public class twodimex
{
    public static void main(String[] args)
    {
        int myarr[][]={{5,8,9},{8,6,9},{10,12,55}};

        for(int i=0;i<myarr.length;i++)
        {
            for(int j=0;j<myarr.length;j++)
            {
                System.out.println("value of "+ i + j +
elements is:-"+myarr[i][j]);
            }
        }
    }
}
```

## Arrays of arrays:

```
int[][] twoDim = new int[4][];
twoDim[0] = new int[5];
twoDim[1] = new int[5];
```

### Non-rectangular arrays of arrays:

```
twoDim[0] = new int[2];
twoDim[1] = new int[4];
twoDim[2] = new int[6];
twoDim[3] = new int[8];
```

## Array of Object

```
public Point[] createArray()
{
    Point[] p;
    p = new Point[10];

    for ( int i=0; i<10; i++ )
    {
        p[i] = new Point(i, i+1);
    }
    return p;
}
```

}

## Array Resizing

- You cannot resize an array.
- You can use the same reference variable to refer to an entirely new array, such as:  
int[] myArray = new int[6];  
myArray = new int[10];

## Copying Arrays

The System.arraycopy() method to copy arrays is:

```
//original array  
int[] myArray = { 1, 2, 3, 4, 5, 6 };  
// new larger array  
int[] hold = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };  
// copy all of the myArray array to the hold  
// array, starting with the 0th index  
System.arraycopy(myArray, 0, hold, 0, myArray.length);
```

## Class Design

### Class Design

Object-oriented programming (OOP) is at the core of Java. In fact, all Java programs are to at least some extent object-oriented. OOP is so integral to Java that it is best to understand its basic principles before you begin writing even simple Java programs. Therefore, this chapter begins with a discussion of the theoretical aspects of OOP.

To manage increasing complexity, the second approach, called object-oriented programming, was conceived. Object-oriented programming organizes a program around its data (that is, objects) and a set of well-defined interfaces to that data. An object-oriented program can be characterized as data controlling access to code. As you will see, by switching the controlling entity to data, you can achieve several organizational benefits.

### What is Inheritance?

Inheritance is the process by which object of one class can acquire properties of other class.

Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a *super class*. The class that does the inheriting is called a *subclass*. Therefore, a subclass is a specialized version of a super class. It inherits all of the instance variables and a method defined by the super class and adds its own, unique elements.

### Types of Inheritance:

- Single Inheritance
- Multiple Inheritance

Java supports only single inheritance and does not support multiple inheritance. The other types such as multi-level, hierarchical are not a basic type of inheritance we can achieve such types of inheritance by using different combinations of single and multiple inheritance.

**Single Inheritance:** - One class can acquire properties of one class.

**Example:**

```
Class A
{
    int i, j;
    void showij()
    {
        System.out.println("i and j: " + i + " " + j);
    }
}
// Create a subclass by extending class A.
class B extends A
{
    int k;
    void showk()
    {
        System.out.println("k: " + k);
    }
    void sum()
    {
        System.out.println("i+j+k: " + (i+j+k));
    }
}
class SimpleInheritance
{
    public static void main(String args[])
    {
        A superOb = new A();
        B subOb = new B();
        // The superclass may be used by itself.
        superOb.i = 10;
        superOb.j = 20;
        System.out.println("Contents of superOb: ");
        superOb.showij();
        System.out.println();
        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        System.out.println("Contents of subOb: ");
        subOb.showij();
        subOb.showk();
        System.out.println();
        System.out.println("Sum of i, j and k in subOb:");
        subOb.sum();
    }
}
```

## Output:

Contents of superOb:  
i and j: 10 20

Contents of subOb:

i and j: 7 8

k: 9

Sum of i, j and k in subOb:

i+j+k: 24

**Multiple Inheritance:** - One class can acquire properties of more than one classes.

**Multi-level Inheritance:** - In multilevel inheritance, the ladder of single inheritance increases.

### Example:

```
class Aves
{
    public void nature()
    {
        System.out.println("Generally, Aves fly");
    }
}
class Bird extends Aves
{
    public void eat()
    {
        System.out.println("Eats to live");
    }
}
public class Parrot extends Bird
{
    public void food()
    {
        System.out.println("Parrot eats seeds and fruits");
    }
    public static void main(String args[])
    {
        Parrot p1 = new Parrot();
        p1.food();      // calling its own
        p1.eat();       // calling super class Bird method
        p1.nature();   // calling super class Aves method
    }
}
```

**Hierarchical Inheritance:** - It will achieve using combination of single and multiple inheritance.

### Example:

```
class Aves
{
    public void fly()
    {
```

```
        System.out.println("Generally, aves fly");
    }
}
class Parrot extends Aves
{
    public void eat()
    {
        System.out.println("Parrot eats fruits and seeds");
    }
}
class Vulture extends Ave
{
    public void vision()
    {
        System.out.println("Vulture can see from high
altitudes");
    }
}
public class FlyingCreatures
{
    public static void main(String args[])
    { // all the following code is composition for FlyingCreatures
        Parrot p1 = new Parrot();
        p1.eat(); // calling its own member
        p1.fly(); // calling super class member by inheritance
        Vulture v1 = new Vulture();
        v1.vision(); // calling its own member
        v1.fly(); // calling super class member by inheritance
    }
}
```

## Access Control

| Access Modifiers | Same Class | Same Package | Subclass | Other Packages |
|------------------|------------|--------------|----------|----------------|
| Public           | Y          | Y            | Y        | Y              |
| protected        | Y          | Y            | Y        | N              |
| Default          | Y          | Y            | N        | N              |
| Private          | Y          | N            | N        | N              |

## Method Overriding

- When a method in a subclass has the same name and type signature as a method in its super class, then the method in the subclass is said to *override* the method in the super class.
- When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass. The version of the method defined by the super class will be hidden. Consider the following:

**Example:**

```
class A
{
    int i, j;
    A(int a, int b)
    {
        i = a;
        j = b;
    }
    void show()
    {
        System.out.println("i and j: " + i + " " + j);
    }
}
class B extends A
{
    int k;
    B(int a, int b, int c)
    {
        super(a, b);
        k = c;
    }
    void show()
    {
        System.out.println("k: " + k);
    }
}
class Override
{
    Public static void main(String args[])
    {
        B subOb = new B(1, 2, 3);
        subOb.show(); // this calls show() in B
    }
}
```

**Output:**

```
k: 3
```

When **show()** is invoked on an object of type **B**, the version of **show()** defined within **B** is used. That is, the version of **show()** inside **B** overrides the version declared in **A**. If you wish to access the super class version of an overridden function, you can do so by using **super**. For example, in

this version of **B**, the super class version of **show( )** is invoked within the subclass' version. This allows all instance variables to be displayed.

```
class B extends A
{
    int k;
    B(int a, int b, int c)
    {
        super(a, b);
        k = c;
    }
    void show()
    {
        super.show(); // this calls A's show()
        System.out.println("k: " + k);
    }
}
```

If you substitute this version of **A** into the previous program, you will see the

**Output:**

```
i and j: 1 2
k: 3
```

Here, **super.show( )** calls the super class version of **show()**. Method overriding occurs *only* when the names and the type signatures of the two methods are identical. If they are not, then the two methods are simply overloaded. For example, consider this modified version of the preceding example:

```
// Methods with differing type signatures are overloaded – not
// overridden.
```

```
class A
{
    int i, j;
    A(int a, int b)
    {
        i = a;
        j = b;
    }
    // display i and j
    void show()
    {
        System.out.println("i and j: " + i + " " + j);
    }
}
// Create a subclass by extending class A.
class B extends A
{
    int k;
    B(int a, int b, int c)
    {
        super(a, b);
```

```
k = c;  
}  
// overload show()  
void show(String msg)  
{  
    System.out.println(msg + k);  
}  
}  
class Override  
{  
    public static void main(String args[])  
    {  
        B subOb = new B(1, 2, 3);  
        subOb.show("This is k: "); // this calls show() in B  
        subOb.show(); // this calls show() in A  
    }  
}
```

**Output:**

```
This is k: 3  
i and j: 1 2
```

The version of **show( )** in **B** takes a string parameter. This makes its type signature different from the one in **A**, which takes no parameters. Therefore, no overriding (or name hiding) takes place.

## Invoking Overridden Method

Super keyword

- Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword **super**.
- **super** has two general forms.
- The first calls the super class' constructor.
- The second is used to access a member of the super class that has been hidden by a member of a subclass.
- A subclass can call a constructor method defined by its super class by use of the following form of **super**:

**Syntax:**

```
super (parameter-list);
```

Here, parameter-list specifies any parameters needed by the constructor in the super class.

**super( )** must always be the first statement executed inside a subclass' constructor.

- The second form of **super** acts somewhat like **this**, except that it always refers to the super class of the subclass in which it is used. This usage has the following general form:

**Example:**

```
class A  
{  
    int i;  
}  
class B extends A  
{
```

```
int i; // this i hides the i in A
B(int a, int b)
{
    super.i = a; // i in A
    i = b; // i in B
}
void show()
{
    System.out.println("i in superclass: " + super.i);
    System.out.println("i in subclass: " + i);
}
}
class UseSuper
{
    public static void main(String args[])
    {
        B subOb = new B(1, 2);
        subOb.show();
    }
}
Output:
i in super class: 1
i in subclass: 2
```

## Polymorphism

*Polymorphism* (from Greek, meaning “many forms”) is a feature that allows one interface to be used for a general class of actions. The specific action is determined by the exact nature.

### Types of Polymorphism:

- **Compile – time** Polymorphism  
Example: - **Method Overloading**
- **Run – time** Polymorphism  
Example: - **Method Overriding**

## Dynamic Method Dispatch Or Virtual Method Invocation

Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. Dynamic method dispatch is important because this is how Java implements run-time polymorphism.

Let’s begin by restating an important principle: a super class reference variable can refer to a subclass object. Java uses this fact to resolve calls to overridden methods at run time. Here is how. When an overridden method is called through a super class reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. When different types of objects are referred to, different versions of an overridden method will be called.

In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed. Therefore, if a super class contains a method that is overridden by a subclass, then when different types of objects are referred to through super class reference variable, different versions of the method are executed.

**Example:**

```
class X
{
    void my ()
    {
        System.out.println ("Inside X's my () method");
    }
}
class Y extends X
{
    // override my ()
    void my ()
    {
        System.out.println ("Inside Y's my () method");
    }
}
class Z extends X
{
    // override my()
    void my ()
    {
        System.out.println ("Inside Z's my () method");
    }
}
class VirtualMethodExample
{
    public static void main (String args[])
    {
        X x = new X(); // object of type X
        Y y = new Y(); // object of type Y
        Z z = new Z(); // object of type Z
        X m;          // obtain a reference of type X
        m = x;         // m refers to an X object
        m.my ();      // calls X's version of my ()
        m = y;         // m refers to a Y object
        m.my ();      // calls Y's version of my()
        m = z;         // m refers to a Z object
        m.my ();      // calls Z's version of my ()
    }
}
```

**Output:**

Inside X's my () method  
Inside Y's my () method  
Inside Z's my () method

## Polymorphic Arguments

Because a Manager is an Employee, the following is valid:

**Example:**

```
public class TaxService
```

```
{  
    public TaxRate findTaxRate(Employee e)  
    {  
        // calculate the employee's tax rate  
    }  
}  
// Meanwhile, elsewhere in the application class  
  
TaxService taxSvc = new TaxService();  
Manager m = new Manager();  
TaxRate t = taxSvc.findTaxRate(m);
```

## The instanceof Operator

```
public class Employee extends Object  
{  
    public class Manager extends Employee  
{  
        public class Engineer extends Employee  
{  
            public void doSomething(Employee e)  
            {  
                if ( e instanceof Manager )  
                {  
                    // Process a Manager  
                }  
                else if ( e instanceof Engineer )  
                {  
                    // Process an Engineer  
                }  
                else  
                {  
                    // Process any other type of Employee  
                }  
            }  
        }  
    }  
}
```

## Casting Objects

```
public void doSomething(Employee e)  
{  
    if ( e instanceof Manager )  
    {  
        Manager m = (Manager) e;  
        System.out.println("This is the manager of  
        "+m.getDepartment());  
    }  
    // rest of operation  
}
```

Use instanceof to test the type of an object.

- Restore full functionality of an object by casting.
- Check for proper casting using the following guidelines:
- Casts *upward* in the hierarchy are done implicitly.
- *Downward* casts must be to a subclass and checked by the compiler.
- The object type is checked at runtime when runtime errors can occur.

## Method Overloading

- In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be *overloaded*, and the process is referred to as *method overloading*.
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Thus, overloaded methods must differ in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method.
- When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

### Example:

```
class OverloadDemo
{
    void test()
    {
        System.out.println("No parameters");
    }
    void test(int a)
    {
        System.out.println("a: " + a);
    }
    void test(int a, int b)
    {
        System.out.println("a and b: " + a + " " + b);
    }
    double test(double a)
    {
        System.out.println("double a: " + a);
        return a*a;
    }
}
class Overload
{
    public static void main(String args[])
    {
        OverloadDemo ob = new OverloadDemo();
        double result;
        ob.test();
        ob.test(10);
```

```
        ob.test(10, 20);
        result = ob.test(123.25);
        System.out.println("Result of ob.test(123.25): " + result);
    }
}

Output:
a: 10
a and b: 10 20
double a: 123.25
Result of ob.test(123.25): 15190.5625
```

## Methods Using Variable Arguments

Methods using *variable arguments* permit multiple number of arguments in methods.

For example:

```
public class Statistics {
    public float average(int... nums) {
        int sum = 0;
        for (int x : nums) {
            sum += x;
        }
        return ((float) sum) / nums.length;
    }
}
```

- The *vararg* parameter is treated as an array. For example:

```
float gradePointAverage = stats.average(4, 3, 4);
float averageAge = stats.average(24, 32, 27, 18);
```

## Constructors Overloading

- In addition to overloading normal methods, you can also overload constructor same as methods.
- You can overload constructors as follow.

**Example:**

```
class Box
{
    double width;
    double height;
    double depth;
    Box()
    {
        width = -1; // use -1 to indicate
        height = -1; // an uninitialized
        depth = -1; // box
    }
    Box(double len)
    {
```

```
        width = height = depth = len;
    }
    Box(double w, double h, double d)
    {
        width = w;
        height = h;
        depth = d;
    }
    double volume()
    {
        return width * height * depth;
    }
}
class OverloadCons
{
    public static void main(String args[])
    {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);
        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
    }
}
```

**Output:**

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
```

## Constructing and Initializing Objects

Memory is allocated and default initialization occurs.

Instance variable initialization uses these steps recursively:

1. Bind constructor parameters.
2. If explicit this(), call recursively, and then skip to Step 5.
3. Call recursively the implicit or explicit super call, except for Object.
4. Execute the explicit instance variable initializers.
5. Execute the body of the current constructor.

## The Object Class

1. The Object class is the root of all classes in Java.
2. A class declaration with no extends clause implies extends Object. For example:

```
public class Employee
{
    ...
}

is equivalent to:
public class Employee extends Object
{
    ...
}
```

Two important methods are:

- 1) equals
- 2) toString

## The equals Method

1. The == operator determines if two references are identical to each other (that is, refer to the same object).
2. The equals method determines if objects are *equal* but not necessarily identical.
3. The Object implementation of the equals method uses the == operator.
4. User classes can override the equals method to implement a domain-specific test for equality.
5. Note: You should override the hashCode method if you override the equals method.

## Using Objects as Parameters

It is also possible to pass object as parameter as follow

**Example:**

```
// Objects may be passed to methods.
class Test
{
    int a, b;
    Test(int i, int j)
    {
        a = i;
        b = j;
    }
    // return true if o is equal to the invoking object
    boolean equals(Test o)
    {
        if(o.a == a && o.b == b)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
class PassOb
```

```
{  
    public static void main(String args[])  
    {  
        Test ob1 = new Test(100, 22);  
        Test ob2 = new Test(100, 22);  
        Test ob3 = new Test(-1, -1);  
        System.out.println("ob1 == ob2: " + ob1.equals(ob2));  
        System.out.println("ob1 == ob3: " + ob1.equals(ob3));  
    }  
}
```

## Output: -

```
ob1 == ob2: true  
ob1 == ob3: false
```

As you can see, the equals( ) method inside Test compares two objects for equality and returns the result. That is, it compares the invoking object with the one that it is passed. If they contain the same values, then the method returns true. Otherwise, it returns false. Notice that the parameter o in equals( ) specifies Test as its type. Although Test is a class type created by the program, it is used in just the same ways Java's built-in types

## Returning Objects

A method can return any type of data, including class types that you create.

For example, in the following program, the incrByTen( ) method returns an object in which the value of a is ten greater than it is in the invoking object.

### Example:

```
Class Test  
{  
    int a;  
    Test(int i)  
    {  
        a = i;  
    }  
    Test incrByTen()  
    {  
        Test temp = new Test(a+10);  
        return temp;  
    }  
}  
class RetOb  
{  
    public static void main(String args[])  
    {  
        Test ob1 = new Test(2);  
        Test ob2;  
        ob2 = ob1.incrByTen();  
        System.out.println("ob1.a: " + ob1.a);  
        System.out.println("ob2.a: " + ob2.a);  
    }  
}
```

```
        ob2 = ob2.incrByTen();
        System.out.println("ob2.a after second increase: "+
                           ob2.a);
    }
}

Output: -
ob1.a: 2
ob2.a: 12
ob2.a after second increase: 22
```

## The **toString** Method

The **toString** method has the following characteristics:

- This method converts an object to a String.
- Use this method during string concatenation.
- Override this method to provide information about a user-defined object in readable format.
- Use the wrapper class's **toString** static method to convert primitive types to a String.

## What is Wrapper class?

- Wrapper class is a wrapper around a primitive data type. It represents primitive data types in their corresponding class instances e.g. a boolean data type can be represented as a Boolean class instance.
- All of the primitive wrapper classes in Java are immutable i.e. once assigned a value to a wrapper class instance cannot be changed further.

Wrapper Classes are used broadly with Collection classes in the **java.util** package and with the classes in the **java.lang.reflect** reflection package.

- Following table lists the primitive types and the corresponding wrapper classes:

| Primitive      | Wrapper                    |
|----------------|----------------------------|
| <b>boolean</b> | <b>java.lang.Boolean</b>   |
| <b>Byte</b>    | <b>java.lang.Byte</b>      |
| <b>Char</b>    | <b>java.lang.Character</b> |
| <b>Double</b>  | <b>java.lang.Double</b>    |
| <b>Float</b>   | <b>java.lang.Float</b>     |
| <b>Int</b>     | <b>java.lang.Integer</b>   |

|              |                        |
|--------------|------------------------|
| <b>Long</b>  | <b>java.lang.Long</b>  |
| <b>Short</b> | <b>java.lang.Short</b> |
| <b>Void</b>  | <b>java.lang.Void</b>  |

## Features Of the Wrapper Classes

Some of the sound features maintained by the Wrapper Classes are as under :

- All the methods of the wrapper classes are static.
- The Wrapper class does not contain constructors.
- Once a value is assigned to a wrapper class instance it can not be changed, anymore.

## Wrapper Classes : Methods with examples

There are some of the methods of the Wrapper class which are used to manipulate the data. Few of them are given below:

`add(int, Object)`: Learn to insert an element at the specified position.

`add(Object)`: Learn to insert an object at the end of a list.

`addAll(ArrayList)`: Learn to insert an array list of objects to another list.

`get()`: Learn to retrieve the elements contained with in an `ArrayList` object.

`Integer.toBinaryString()`: Learn to convert the `Integer` type object to a `String` object.

`size()`: Learn to get the dynamic capacity of a list.

`remove()`: Learn to remove an element from a particular position specified by a index value.

`set(int, Object)`: Learn to replace an element at the position specified by a index value.

## Autoboxing of Primitive Types

Autoboxing has the following description:

- Conversion of primitive types to the object equivalent
- Wrapper classes not always needed
- Example:

```
int plnt = 420;
```

```
Integer wInt = pInt; // this is called autoboxing
```

```
int p2 = wInt; // this is called autounboxing
```

- Language feature used most often when dealing with collections
- Wrapped primitives also usable in arithmetic expressions
- Performance loss when using autoboxing

## Advance Class Design

### Static Keyword

- There will be times when you will want to define a class member that will be used independently of any object of that class.
- Normally a class member must be accessed only in conjunction with an object of its class. However, it is possible to create a member that can be used by itself, without reference to a specific instance.
- To create such a member, precede its declaration with the keyword **static**. When a member is declared **static**, it can be accessed before any objects of its class are created, and without reference to any object.
- You can declare both methods and variables to be **static**. The most common example of a **static** member is **main( )**. **main( )** is declared as **static** because it must be called before any objects exist.
- Instance variables declared as **static** are, essentially, global variables. When objects of its class are declared, no copy of a **static** variable is made. Instead, all instances of the class share the same **static** variable.

Methods declared as **static** have several restrictions:

- They can only call other **static** methods.
- They must only access **static** data.
- They cannot refer to **this** or **super** in any way.

#### Example:

```
class UseStatic
{
    static int a = 3;
    static int b;
    static void meth(int x)
    {
        System.out.println("x = " + x);
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
    static
    {
        System.out.println("Static block initialized.");
        b = a * 4;
    }
    public static void main(String args[])
}
```

```
{  
    meth(42);  
}  
}
```

## Final Keyword

In Java, final keyword is applied in various context.

The final keyword is a modifier means the final class can't be extended, a variable can't be modified, and also a method can't be override.

### final classes

The class declared as final can't be subclass or extend. The final class declared as follows :

```
public final class MyFinalClass
```

### final methods

The final method can be declare as follows:

```
public final String convertCurrency()
```

The final method can't be override in a subclass.

### final variable

The field declared as final behaves like constant. Means once it is declared, it can't be changed. Before compiling, only once it can be set; after that you can't change its value. Attempt to change in its value lead to exception or compile time error.

You can declare the final fields as :

```
public final double radius = 126.45;
```

```
public final int PI = 3.145;
```

#### Example:

```
class Another  
{  
    final int xyz;  
    public Another()  
    {  
        xyz=1000;  
    }  
    public void display()  
    {  
        System.out.println("XYZ="+xyz);  
    }  
}  
public class FinalTest
```

```
{  
    static final int y=3;  
    public FinalTest()  
    {  
    }  
    static  
    {  
        System.out.println("Just for fun...");  
    }  
    public static void main(String[] args)  
    {  
        final int x=5;  
        System.out.println("X="+x);  
        System.out.println("Y="+y);  
  
        Another ano=new Another();  
        ano.display();  
    }  
}
```

## Enum Types

- An enum type is a type whose fields consist of a fixed set of constants
- Any enum type is declared very much like a class, and an enum value behaves very much like an immutable object
- In Java, an enum value may have data members, and even code.  
enum Planet { MERCURY, VENUS, EARTH, MARS, JUPITER, SATURN, URANUS, NEPTUNE, PLUTO };  
Planet myPlanet = MARS;

## Static import

- A static import imports the static members from a class:  
import static <pkg\_list>.<class\_name>.<member\_name>;  
OR  
import static <pkg\_list>.<class\_name>.\*;
- A static import imports members individually or collectively:  
import static cards.domain.Suit.SPADES;  
OR  
import static cards.domain.Suit.\*;
- There is no need to qualify the static constants:  
PlayingCard card1 = new PlayingCard(SPADES, 2);
- Use this feature sparingly.

## What is Abstraction?

- Abstraction in the process of selecting important data sets for an Object in your software, and leaving out the insignificant ones.

- Once you have modeled your object using Abstraction, the same set of data could be used in different applications.

## Abstract Class:

- There are situations in which you will want to define a super class that declares the structure of a given abstraction without providing a complete implementation of every method.
- That is, sometimes you will want to create a super class that only defines a generalized form that will be shared by all of its subclasses, leaving it to each subclass to fill in the details. Such a class determines the nature of the methods that the subclasses must implement. One way this situation can occur is when a super class is unable to create a meaningful implementation for a method.
- main use of abstract class is provide a common definition or behavior for its sub class.
- if class declare as an abstract then we can't create object of that class
- if method in class is declare as abstract then class must be declare as abstract
- abstract methods does not have a body
- You must be override all abstract method in its sub class if u can't override then this (sub class) also declare as abstract
- abstract class contains the abstract methods and non-abstract methods
- we can declare a class abstract using 'abstract' keyword and same as methods

## Example:

// A Simple demonstration of abstract.

```
abstract class A
{
    abstract void callme();
    // concrete methods are still allowed in abstract classes
    void callmetoo()
    {
        System.out.println("This is a concrete method.");
    }
}
class B extends A
{
    void callme()
    {
        System.out.println("B's implementation of callme.");
    }
}
class AbstractDemo
{
    public static void main(String args[])
    {
        B b = new B();
        b.callme();
        b.callmetoo();
    }
}
```

## What is Interface?

- Using the keyword **interface**, you can fully abstract a class' interface from its implementation.

- That is, using **interface**, you can specify what a class must do, but not how it does it. Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are declared without any body.
- To implement an interface, a class must create the complete set of methods defined by the interface.
- However, each class is free to determine the details of its own implementation. By providing the **interface** keyword, Java allows you to fully utilize the “one interface, multiple methods” aspect of polymorphism.
  - we can not create a object of interface
  - interface define only non implemented methods
  - A class that implements an interface must implement all of the methods described in the interface.
  - all methods in interface is by default public (we can not use private and protected modifier in interface)
  - all variables are in interface is by default public static final

## Defining an Interface

An interface is defined much like a class. This is the general form of an interface:

### Syntax:

```
access interface name
{
    return-type method-name1(parameter-list);
    return-type method-name2(parameter-list);
    type final-varname1 = value;
    type final-varname2 = value;
    // ...
    return-type method-nameN(parameter-list);
        type final-varnameN = value;
}
```

## Implementing Interfaces

```
access class classname [extends superclass][implements interface [,interface...]]
{
    // class-body
}
```

### Example:

```
interface first
{
    public void show();
    public void print();
    public void display();
}

class second implements first
{
    public void show()
    {
        System.out.println("this is show method");
    }
    public void print()
```

```
{  
    System.out.println("this is print method");  
}  
public void display()  
{  
    System.out.println("this is display method");  
}  
  
}  
public class interfaceexample  
{  
    public static void main(String args[])  
{  
        second s1=new second();  
        s1.display();  
        s1.show();  
        s1.print();  
    }  
}
```

## Example for multiple Inheritance

```
interface test  
{  
    public void callme();  
    public void document();  
}  
interface demo  
{  
    public void hello();  
    public void welcome();  
}  
class demo2  
{  
}  
class testing extends demo2 implements test, demo  
{  
    public void hello()  
    {  
        System.out.println("this os hello method");  
    }  
    public void welcome()  
    {  
        System.out.println("this is welcome method");  
    }  
    public void callme()  
    {  
        System.out.println("this is callme method");  
    }  
    public void document()
```

```
{  
    System.out.println("this is document method");  
}  
}  
public class multipleinterface  
{  
    public static void main(String args[])  
    {  
        testing t1=new testing();  
        t1.callme();  
        t1.document();  
        t1.welcome();  
        t1.hello();  
    }  
}
```

## Interfaces Can Be Extended

- One interface can inherit another by use of the keyword **extends**. The syntax is the same as for inheriting classes.
- When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance chain.

### Example:

```
// One interface can extend another.  
interface A  
{  
    void meth1();  
    void meth2();  
}  
// B now includes meth1() and meth2() -- it adds meth3().  
interface B extends A  
{  
    void meth3();  
}  
// This class must implement all of A and B  
class MyClass implements B  
{  
    public void meth1()  
    {  
        System.out.println("Implement meth1().");  
    }  
    public void meth2()  
    {  
        System.out.println("Implement meth2().");  
    }  
    public void meth3()  
    {  
        System.out.println("Implement meth3().");  
    } }
```

```
class IFExtend
{
    public static void main(String arg[])
    {
        MyClass ob = new MyClass();
        ob.meth1();
        ob.meth2();
        ob.meth3();
    }
}
```

## Nested Interfaces

- An interface can be declared a member of a class or another interface. Such an interface is called a *member interface* or a *nested interface*.
- A nested interface can be declared as **public**, **private**, or **protected**. This differs from a top-level interface, which must either be declared as **public** or use the default access level, as previously described.
- When a nested interface is used outside of its enclosing scope, it must be qualified by the name of the class or interface of which it is a member.
- Thus, outside of the class or interface in which a nested interface is declared, its name must be fully qualified.

### Example:

```
// A nested interface example.
// This class contains a member interface.
class A {
    // this is a nested interface
    public interface NestedIF {
        boolean isNotNegative(int x);
    }
}
// B implements the nested interface.
class B implements A.NestedIF {
    public boolean isNotNegative(int x)
    {
        return x < 0 ? false : true;
    }
}
class NestedIFDemo {
    public static void main(String args[])
    {
        // use a nested interface reference
        A.NestedIF nif = new B();
        if(nif.isNotNegative(10))
            System.out.println("10 is not negative");
        if(nif.isNotNegative(-12))
            System.out.println("this won't be displayed");
    }
}
```

Notice that **A** defines a member interface called **NestedIF** and that it is declared **public**. Next, **B** implements the nested interface by specifying  
implements A.NestedIF

## Different between Interface and Abstract

- Main difference is methods of a Java interface are implicitly abstract and cannot have implementations. A Java abstract class can have instance methods that implements a default behavior.
- Variables declared in a Java interface is by default final. An abstract class may contain non-final variables.
- Members of a Java interface are public by default. A Java abstract class can have the usual flavors of class members like private, protected, etc..
- Java interface should be implemented using keyword “implements”; A Java abstract class should be extended using keyword “extends”.
- An interface can extend another Java interface only, an abstract class can extend another Java class and implement multiple Java interfaces.
- A Java class can implement multiple interfaces but it can extend only one abstract class.
- Interface is absolutely abstract and cannot be instantiated; A Java abstract class also cannot be instantiated, but can be invoked if a main() exists.
- In comparison with java abstract classes, java interfaces are slow as it requires extra indirection.

## Different between Overloading and Overriding

### In Overriding

- The argument list must exactly match that of the overridden method.
- The return type must exactly match that of the overridden method.
- The access level must not be more restrictive than that of the overridden method.
- The access level can be less restrictive than that of the overridden method.
- The overriding method must not throw new or broader checked exceptions than those declared by the overridden method.
- The overriding method can throw narrower or fewer exceptions. Just because an overridden method “takes risks” doesn’t mean that the overriding subclass’ exception takes the same risks. Bottom line: An overriding method doesn’t have to declare any exceptions that it will never throw, regardless of what the overridden method declares.
- You cannot override a method marked final.
- If a method can’t be inherited, you cannot override it.

### In Overloading

- Overloaded methods must change the argument list.
- Overloaded methods can change the return type.
- Overloaded methods can change the access modifier.
- Overloaded methods can declare new or broader checked exceptions.
  - A method can be overloaded in the same class or in a subclass.

## • I/O Stream

### Stream

A stream is the ordered sequence of data that uses a common characteristics shared by the entire input/output device. Stream is serried of byte that used to travel data from one place to another. It is common link between program and other device.

There are two types of stream

- 1) Byte Streams
- 2) Character Streams

### Types of stream

#### Byte Stream

- Byte streams allows a programmer to work with the binary data in a file. In this stream data are accessed as a sequence of bytes. all types other than text or character are dealt in this stream.
- Byte stream contains a different types of bytes stream
- Byte stream divided in to mainly 2 parts.

#### Input Stream

- The Input stream class defines the functionality that is available for all byte input streams
- The method provided by the inputstream class are.
- The three basic read methods are:

```
int read()
int read(byte[] buffer)
int read(byte[] buffer, int offset, int length)
```

- Other methods include:  
void close()  
int available()  
long skip(long n)  
boolean markSupported()  
void mark(int readlimit)  
void reset()

#### Output Stream

- The output stream class defines the functionality that is available for all byte output streams
- The method provided by the outputstream class are.
- The three basic write methods are:  
void write(int c)  
void write(byte[] buffer)  
void write(byte[] buffer, int offset, int length)
- Other methods include:  
void close()  
void flush()

**Example (File input and output streams):**

```
class wridata
{
    public wridata()
    {
        try
        {
            FileOutputStream
            FileOutputStream("D:\\\\writedata1.txt");
            //String s="this is 1 file";
            //byte []b=s.getBytes();
            for(int i=0;i<10;i++)
            {
                fos.write(i);
            }
            fos.flush();
            fos.close();
            System.out.println("successfully write");
        }
        catch(Exception e)
        {
            System.out.println("error in write");
        }
    }
}
class readdata
{
    public readdata()
    {
        try
        {
            FileInputStream fis=new
            FileInputStream("D:\\\\writedata1.txt");
            int i;
            while((i=fis.read())!=-1)
            {
                System.out.println(i);
            }
            fis.close();
            System.out.println("successfully read");
        }
        catch(Exception e)
        {
            System.out.println("error in read");
        }
    }
}
public class fileinputoutex
```

```
{  
    public static void main(String[] args)  
    {  
        writedata wd=new writedata();  
        readdata rd=new readdata();  
    }  
}  
Example (Buffered input and output):-  
class bufferwrite  
{  
    public bufferwrite()  
    {  
        try  
        {  
            FileOutputStream fos=new  
                FileOutputStream("D:\\file5.txt");  
            BufferedOutputStream bos=new  
                BufferedOutputStream(fos);  
  
            String s="this is buffered input stream example";  
            byte b[]={};  
            bos.write(b);  
            bos.flush();  
            bos.close();  
        }  
        catch(Exception e)  
        {  
            System.out.println("error in write");  
        }  
    }  
}  
class bufferread  
{  
    public bufferread()  
    {  
        try  
        {  
            FileInputStream fis=new  
                FileInputStream("D:\\file5.txt");  
            BufferedInputStream bis=new  
                BufferedInputStream(fis);  
            int i;  
            while((i=bis.read())!=-1)  
            {  
                System.out.print((char)i);  
            }  
            bis.close();  
        }  
    }  
}
```

```
        catch(Exception e)
        {
            System.out.println("error in read");
        }
    }
public class bufferinputex
{
    public static void main(String args[])
    {
        bufferwrite bw=new bufferwrite();
        bufferread br=new bufferread();
    }
}
```

## Character Stream

- Character stream classes allow you to read and write character and string.
- The character are stored and retrieved in a human readable form an input character Stream convert bytes to characters

## Reader

- Reader stream classes are used to read characters from the file. The reader class provides a functionality that is available for all character input stream.
- The Reader class provides a following types of methods
- The three basic read methods are:

```
int read()
int read(char[] cbuf)
int read(char[] cbuf, int offset, int length)
```
- Other methods include:

```
void close()
boolean ready()
long skip(long n)
boolean markSupported()
void mark(int readAheadLimit)
void reset()
```

## Writer

- The Writer stream classes are used to perform all output operation on files the Writer class is an abstract class which acts as base class for all the other writer stream classes.
- The writer class provides following types of methods.
- The basic write methods are:

```
void write(int c)
void write(char[] cbuf)
void write(char[] cbuf, int offset, int length)
void write(String string)
void write(String string, int offset, int length)
```

- Other methods include:

```
void close()  
void flush()
```

### **Example of (File Reader/Writer):**

```
import java.io.FileReader;  
import java.io.FileWriter;  
class writerdata  
{  
    public writerdata()  
    {  
        try  
        {  
            FileWriter fw=new FileWriter("D:\\writer.txt");  
            String s="this is file writer example";  
            fw.write(s);  
            fw.flush();  
            fw.close();  
            System.out.println("successfully write");  
        }  
        catch(Exception e)  
        {  
            System.out.println("error in write");  
        }  
    }  
    class readerdata  
    {  
        public readerdata()  
        {  
            try  
            {  
                FileReader fr=new FileReader("D:\\writer.txt");  
                int i;  
                while((i=fr.read())!=-1)  
                {  
                    System.out.print((char)i);  
                }  
                fr.close();  
                System.out.println("successfully read");  
            }  
            catch(Exception e)  
            {  
                System.out.println("error in read");  
            }  
        }  
    }  
}
```

```
    }
    public class filereaderwritrex
    {
        public static void main(String[] args)
        {
            writerdata wd=new writerdata();
            readerdata rd=new readerdata();
        }
    }
```

### **Example of CharacterArrayReader:**

```
import java.io.*;
public class CharArrayReaderDemo
{
    public static void main(String args[]) throws IOException
    {
        String tmp = "abcdefghijklmnopqrstuvwxyz";
        int length = tmp.length();
        char c[] = new char[length];
        tmp.getChars(0, length, c, 0);
        CharArrayReader input1 = new CharArrayReader(c);
        CharArrayReader input2 = new CharArrayReader(c, 0,
  5);
        int i;
        System.out.println("input1 is:");
        while((i = input1.read()) != -1)
        {
            System.out.print((char)i);
        }
        System.out.println();
        System.out.println("input2 is:");
        while((i = input2.read()) != -1)
        {
            System.out.print((char)i);
        }
        System.out.println();
    }
}
```

### **Output:**

```
input1 is:
abcdefghijklmnopqrstuvwxyz
input2 is:
abcde
```

### **Example of CharacterArrayWriter:**

```
import java.io.*;
class CharArrayWriterDemo
{
    public static void main(String args[]) throws IOException
    {
```

```
CharArrayWriter f = new CharArrayWriter();
String s = "This should end up in the array";
char buf[] = new char[s.length()];
s.getChars(0, s.length(), buf, 0);
f.write(buf);
System.out.println("Buffer as a string");
System.out.println(f.toString());
System.out.println("Into array");
char c[] = f.toCharArray();
for (int i=0; i<c.length; i++) {
    System.out.print(c[i]);
}
System.out.println("\nTo a FileWriter()");
FileWriter f2 = new FileWriter("test.txt");
f.writeTo(f2);
f2.close();
System.out.println("Doing a reset");
f.reset();
for (int i=0; i<3; i++)
    f.write('X');
System.out.println(f.toString());
    } }
```

### Example of Buffered Reader/Writer:

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
public class BufferReaderWriterDemo {
    public static void main(String[] args)
        {
            try
                {
                    FileReader fr = new FileReader("D:/Xyz/TestArgs.java");
                    BufferedReader br = new BufferedReader(fr);
                    FileWriter fw = new FileWriter("D:/Xyz/Newone.txt");
                    BufferedWriter bw = new BufferedWriter(fw);
                    String line="";
                    while((line = br.readLine()) != null)
                    {
                        bw.write(line);
                    }
                    br.close();
                    bw.close();
                }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
}
```

## BufferedOutputStream:

By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written. For example, consider writing to file: without buffer, it has to access the hard disk for it has to access the hard disk for every single byte, which is obviously slow.

To add buffering to your OutputStream's simply wrap them in a BufferedOutputStream.

Here is how that looks:

```
OutputStream output = new BufferedOutputStream(new FileOutputStream("c:\\data\\output-file.txt"))
```

## BufferedInputStream:

The BufferedInputStream class provides buffering to your input streams.

Buffering can speed up IO quite a bit. Rather than read one byte at a time from the network or disk, you read a larger block at a time.

This is typically much faster, especially for disk access and larger data amounts.

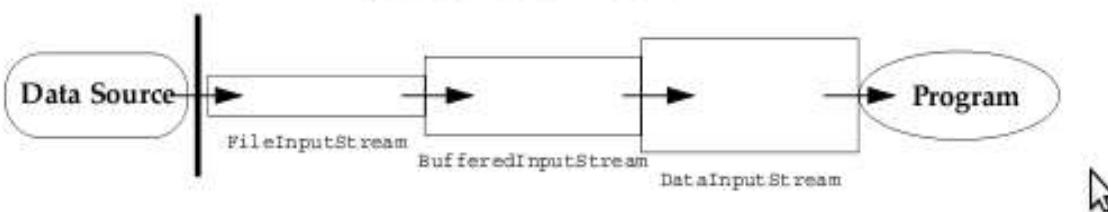
To add buffering to your InputStream's simply wrap them in a BufferedInputStream.

Here is how that looks:

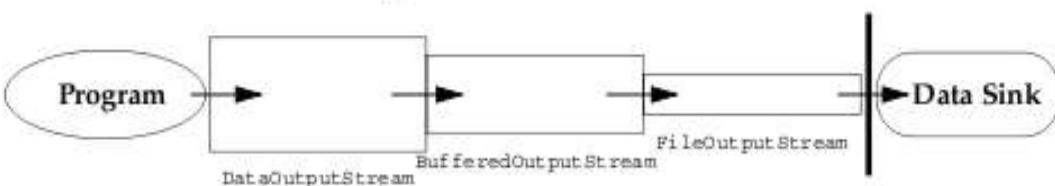
```
InputStream input = new BufferedInputStream(new FileInputStream("c:\\data\\input-file.txt"));
```

# I/O Stream Chaining

## Input Stream Chain



## Output Stream Chain



## Command line arguments

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class KeyboardInput {

    public static void main(String args[]) {
        String s;
        // Create a buffered reader to read
        // each line from the keyboard.
        InputStreamReader ir = new InputStreamReader(System.in);
        BufferedReader in = new BufferedReader(ir);
        System.out.println("Unix: Type ctrl-d to exit.");
        + "\nWindows: Type ctrl-z to exit");
        try {
            // Read each input line and echo it to the screen.
            s = in.readLine();
            while (s != null) {
                System.out.println("Read: " + s);
                s = in.readLine();
            }
            // Close the buffered reader.
            in.close();
        } catch (IOException e) { // Catch any IO exceptions.
            e.printStackTrace();
        }
    }
}
```

Any Java technology application can use command-line arguments.

These string arguments are placed on the command line to launch the Java interpreter, after the class name: java TestArgs arg1 arg2 "another arg" Each command-line argument is placed in the args array that is passed to the static main method: public static void main(String[] args)

## Console I/O

The variable *System.out* enables you to write to *standard output*. It is an object of type *PrintStream*. The variable *System.in* enables you to read from *standard input*.

It is an object of type *InputStream*. The variable *System.err* enables you to write to standard error.

It is an object of type *PrintStream*.

## Writing to Standard Output

The *println* methods print the argument and a newline character (\n). The *print* methods print the argument without a newline character. The *print* and *println* methods are overloaded for most primitive types (boolean, char, int, long, float, and double) and for *char[]*, *Object*, and *String*. The *print(Object)* and *println(Object)* methods call the *toString* method on the argument.

## Reading From Standard Input

## Object Serialization

Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

### Serializable:-

Only an object that implements the **Serializable** interface can be saved and restored by the serialization facilities. The **Serializable** interface defines no members. It is simply used to indicate that a class may be serialized. If a class is serializable, all of its subclasses are also serializable.

Variables that are declared as **transient** are not saved by the serialization facilities. Also, **static** variables are not saved.

#### Example:

```
class student implements Serializable
{
    private int rollno;
    private String name;
    private String lastname;
    public student(int rollno,String name,String lastname)
    {
        this.rollno=rollno;
        this.name=name;
        this.lastname=lastname;
    }
    @Override
    public String toString()
    {
        return "student [rollno=" + rollno + ", name=" + name +
               ", lastname="+lastname+"]";
    }
}
public class readwriteobject
{
    private student s;
    public static void main(String args[])
    {
        student s1=new student(101,"abishek", "patel");
        try
        {
            FileOutputStream fos=new
                FileOutputStream("file3.txt");
            ObjectOutputStream oos=new
                ObjectOutputStream(fos);
            oos.writeObject(s1);
            oos.flush();
            oos.close();
            System.out.println("success");
        }
    }
}
```

```
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    try
    {
        FileInputStream fis=new
            FileInputStream("file3.txt");
        ObjectInputStream ois=new
            ObjectInputStream(fis);
        s1=(student)ois.readObject();
        System.out.println(s1.toString());
        ois.close();
    }
    catch(Exception e)
    {
        e.printStackTrace();
    }
}
```

## DataInput/Output Streams

- The data input stream and data output stream use for write and read the different kinds of data
- for example you can read only int type data you can read only string type data using data input stream class it is possible
- And same for data output stream class
- The data input and data out stream class provides a different method for example readint(),readdouble() and writefloat()....etc

### Example:

```
class dataoutput
{
    public dataoutput()
    {
        try
        {
            FileOutputStream fos=new
                FileOutputStream("D:\\dataffile.txt");
            DataOutputStream dos=new
                DataOutputStream(fos);
            int a=10;
            String s="Hello";
            dos.writeInt(a);
            dos.writeBoolean(false);
            dos.writeChars(s);
            dos.flush();
            dos.close();
        }
    }
}
```

```
        System.out.println("succssfully write");
    }
    catch(Exception e)
    {
        System.out.println("error in writing");
    }
}
class datainput
{
    public datainput()
    {
        try
        {
            FileInputStream fis=new
                FileInputStream("D:\\dataffile.txt");
            DataInputStream dis=new DataInputStream(fis);
            System.out.println("int data:-"+dis.readInt());
            System.out.println("boolean data:
                "+dis.readBoolean());
            int i;
            while((i=dis.readChar())!=-1)
            {
                System.out.println("string data:-"(char)i);
            }
            dis.close();
            System.out.println("successfully read");
        }
        catch(Exception e)
        {
            System.out.println("error in read");
        }
    }
}
public class datainputoutput
{
    public static void main(String[] args)
    {
        dataoutput doa=new dataoutput();
        datainput di=new datainput();
    }
}
```

## Files and File I/O

- Files and directories are accessed and manipulated via the java.io.File class.

- The File class does **not** actually provide for input and output to files. It simply provides an *identifier* of files and directories.
- The file class provides methods those are related to files and directories Using file class we can create file, directories and also perform a different operations.

The following constructors can be used to create **File** objects:

```
File(String directoryPath)  
File(String directoryPath, String filename)  
File(File dirObj, String filename)  
File(URI uriObj)
```

## Creating new File object

```
File myFile;  
myFile = new File("myfile.txt");  
myFile = new File("MyDocs", "myfile.txt");
```

Directories are treated just like files in Java; the File class supports methods for retrieving an array of files in the directory, as follows:

```
File myDir = new File("MyDocs");  
myFile = new File(myDir, "myfile.txt");
```

## File Tests and utilities

The following are the methods of File class:

- **File information:**  
String getName()  
String getPath()  
String getAbsolutePath()  
String getParent()  
long lastModified()  
long length()
- **File modification:**  
boolean renameTo(File newName)  
boolean delete()
- **Directory utilities:**  
boolean mkdir()  
String[] list()
- **File tests:**  
boolean exists()  
boolean canWrite()  
boolean canRead()  
boolean isFile()  
boolean isDirectory()  
boolean isAbsolute();

### Example:

```
Try {
```

```
File file = new File("filename")
// Create file if it does not exist
boolean success = file.createNewFile();
if (success)
{
    // File did not exist and was created
}
else
{
    // File already exists
}
catch (IOException e)
{
}
```

## File Stream I/O

For file input:

Use the FileReader class to read characters. Use the BufferedReader class to use the readLine method.

For file output: Use the FileWriter class to write characters.

    Use the PrintWriter class to use the print and println methods.

## File Input Example

```
package com.handler;

import java.io.*;
public class ReadFile {
    public static void main(String[] args) {
        // Create file
        File file = new File(args[0]);
        try {
            // Create a buffered reader // to read each line from a file.
            BufferedReader in = new BufferedReader(new FileReader(file));
            String s;
            while (s != null) {
                System.out.println("Read: " + s);
                s = in.readLine();
            }
            // Close the buffered reader
            in.close();
        } catch (FileNotFoundException e1) {
            // If this file does not exist
            System.err.println("File not found: " + file);
        } catch (IOException e2) {
            // Catch any other IO exceptions.
            e2.printStackTrace();
        }
    }
}
```

## File Output Example

```
package com.handler;

import java.io.*;
public class WriteFile {

    public static void main(String[] args) {
        // Create file
        File file = new File(args[0]);
        try {
            // Create a buffered reader to read each line from standard in.
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader in = new BufferedReader(isr);
            // Create a print writer on this file.
            PrintWriter out = new PrintWriter(new FileWriter(file));
            String s;
            System.out.print("Enter file text. ");
            System.out.println("[Type ctrl-d to stop.]");
            // Read each input line and echo it to the screen.
            while ((s = in.readLine()) != null) {
                out.println(s);
            }
            // Close the buffered reader and the file print writer.
            in.close();
            out.close();
        } catch (IOException e) {
            // Catch any IO exceptions.
            e.printStackTrace();
        }
    }
}
```

## Using **FilenameFilter**

You will often want to limit the number of files returned by the **list( )** method to include only those files that match a certain filename pattern, or *filter*. To do this, you must use a second form of **list( )**, shown here:

```
String[ ] list(FilenameFilter FFObj)
```

In this form, *FFObj* is an object of a class that implements the **FilenameFilter** interface. **FilenameFilter** defines only a single method, **accept( )**, which is called once for each file in a list. Its general form is given here:

```
boolean accept(File directory, String filename)
```

The **accept( )** method returns **true** for files in the directory specified by *directory* that should be included in the list (that is, those that match the *filename* argument), and returns **false** for those files that should be excluded.

**Example:**

```
import java.io.*;
public class OnlyExt implements FilenameFilter
{
    String ext;
    public OnlyExt(String ext)
    {
        this.ext = "." + ext;
    }
    public boolean accept(File dir, String name)
    {
        return name.endsWith(ext);
    }
}
import java.io.*;
class DirListOnly
{
    public static void main(String args[])
    {
        String dirname = "/java";
        File f1 = new File(dirname);
        FilenameFilter only = new OnlyExt("html");
        String s[] = f1.list(only);
        for (int i=0; i < s.length; i++)
        {
            System.out.println(s[i]);
        }
    }
}
```

## Exceptional Handling

### What is Exception?

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code.
- When an exceptional condition arises, an object representing that exception is created and *thrown* in the method that caused the error.
- That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is *caught* and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code.
- Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method.
- Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.
- Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the **try** block, it is thrown. Your code can catch this exception (using **catch**) and handle it in some rational manner.
- System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword **throw**. Any exception that is thrown out of a method must be specified as such by a **throws** clause. Any code that absolutely must be executed before a method returns is put in a **finally** block.

**Syntax:**

```
try
{
    // block of code to monitor for errors
}
catch (ExceptionType1 exOb)
{
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb)
{
    // exception handler for ExceptionType2
}
// ...
finally
{
    // block of code to be executed before try block ends
}
```

## Try and Catch statement

- The try/catch statement encloses some code and is used to handle errors and exceptions that might occur in that code. Here is the general syntax of the try/catch statement:

**Syntax:**

```
try
{
    body-code
}
catch (exception-classname variable-name)
```

```
{  
    handler-code  
}
```

- The try/catch statement has four parts. The *body-code* contains code that might throw the exception that we want to handle.
- The *exception-classname* is the class name of the exception we want to handle. The *variable-name* specifies a name for a variable that will hold the exception object if the exception occurs.
- Finally, the *handler-code* contains the code to execute if the exception occurs. After the handler-code executes, execution of the thread continues after the try/catch statement.
- Here is an example of code that tries to create a file in a non-existent directory which results in an IOException.

**Example:**

```
String filename = "/nosuchdir/myfilename";  
try  
{  
    // Create the file  
    new File(filename).createNewFile();  
}  
catch (IOException e)  
{  
    // Print out the exception that occurred  
    System.out.println("Unable to create "+filename+":  
    "+e.getMessage());  
}  
// Execution continues here after the IOException handler is executed
```

## Multiple Catch Block

- It is possible to specify more than one exception handler in a try/catch statement.
- When an exception occurs, each handler is checked in order (i.e. top-down) and the handler that first matches is executed.
- The following example describe how use multiple catch block

**Example:**

```
public class Multi_Catch  
{  
    public static void main (String args[])  
    {  
        int array[]={20,10,30};  
        int num1=15,num2=0;  
        int res=0;  
        try  
        {  
            res = num1/num2;  
            System.out.println("The result is" +res);  
  
            for(int ct =2;ct >=0; ct--)  
            {
```

```
        System.out.println("The value of array are"
                           +array[ct]);
    }
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("Error?. Array is out of
                       Bounds");
}
catch (ArithmaticException e)
{
    System.out.println ("Can't be divided by Zero");
}
}
}
```

## The finally Clause

- The finally block *always* executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having cleanup code accidentally bypassed by a return, continue, or break.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

```
try {
    startFaucet();
    waterLawn();
} catch (BrokenPipeException e) {
    logProblem(e);
} finally {
    stopFaucet();
}
```

## Common Exceptions

- NullPointerException
- FileNotFoundException
- NumberFormatException
- ArithmaticException
- SecurityException

## Nested try Statement

- In Java we can have nested try and catch blocks. It means that, a try statement can be inside the block of another try.
- If an inner try statement does not have a matching catch statement for a particular exception, the control is transferred to the next try statement's catch handlers that are expected for a matching catch statement.

- This continues until one of the catch statements succeeds, or until all of the nested try statements are done in.
- If no one catch statements match, then the Java run-time system will handle the exception.

**Syntax:**

```
try
{
    try
    {
        // ...
    }
    catch (Exception1 e)
    {
        //statements to handle the exception
    }
}
catch (Exception 2 e2)
{
    //statements to handle the exception
}
```

**Example:**

```
import java.io.*;
public class NestedTry
{
    public static void main (String args[])throws IOException
    {
        int num=2,res=0;
        try
        {
            FileInputStream fis=null;
            fis = new FileInputStream (new File (args[0]));
            try
            {
                res=num/0;
                System.out.println("The result is"+res);
            }
            catch(ArithmaticException e)
            {
                System.out.println("divided by Zero");
            }
        }
        catch (FileNotFoundException e)
        {
            System.out.println("File not found!");
        }
        catch(ArrayIndexOutOfBoundsException e)
        {
            System.out.println("Array index is Out of bound!
Argument required");
        }
    }
}
```

```
        }
        catch(Exception e)
        {
            System.out.println("Error.."+e);
        }
    }
}
```

## Exception Categories

### Use of Throw

- Before you can catch an exception, some code somewhere must throw one. Any code can throw an exception: your code, code from a package written by someone else such as the packages that come with the Java platform, or the Java runtime environment.
- Regardless of what throws the exception, it's always thrown with the throw statement.
- All methods use the throw statement to throw an exception. The throw statement requires a single argument: a throwable object. Throwable objects are instances of any subclass of the Throwable class. Here's an example of a throw statement.  
*throw someThrowableObject;*
- Let's look at the throw statement in context. The following pop method is taken from a class that implements a common stack object.
- The method removes the top element from the stack and returns the object.

#### Example:

```
class MyException extends Exception
{
    public MyException(String msg)
    {
        super(msg);
    }
}
public class Test
{
    static int divide(int first,int second) throws MyException
    {
        if(second==0)
            throw new MyException("can't be divided by
                                zero");
        return first/second;
    }
    public static void main(String[] args)
    {
        try
        {
            System.out.println(divide(4,0));
        }
        catch (MyException exc)
        {
```

```
        exc.printStackTrace();
    }
}
}
```

## Use of Throws

- Whereas when we know that a particular exception may be thrown or to pass a possible exception then we use throws keyword.
- Point to note here is that the Java compiler very well knows about the exceptions thrown by some methods so it insists us to handle them.
- We can also use throws clause on the surrounding method instead of try and catch exception handler. For instance in the above given program we have used the following clause which will pass the error up to the next level

### Example:

```
class MyException extends Exception
{
    public MyException(String msg)
    {
        super(msg);
    }
}

public class Test
{
    static int divide(int first,int second) throws MyException
    {
        if(second==0)
            throw new MyException("can't be divided by zero");
        return first/second;
    }

    public static void main(String[] args)
    {
        try
        {
            System.out.println(divide(4,0));
        }
        catch (MyException exc)
        {
            exc.printStackTrace();
        }
    }
}
```

## Handle or declare a Rule

Use the handle or declare rule as follows:

Handle the exception by using the try-catch-finally block. Declare that the code causes an exception by using the throws clause.

```
void trouble() throws IOException { ... }  
void trouble() throws IOException, MyException { ... }
```

## Other Principles

You do not need to declare runtime exceptions or errors.

You can choose to handle runtime exceptions.

## Method overriding and exception

The overriding method can throw:

- No exceptions
- One or more of the exceptions thrown by the overridden method
- One or more subclasses of the exceptions thrown by the overridden method

The overriding method cannot throw:

- Additional exceptions not thrown by the overridden method
- Superclasses of the exceptions thrown by the overridden method

## The try with resources statement

The *try-with-resources* statement is a try statement that declares one or more resources

A *resource* is an object that must be closed after the program is finished with it.

```
static String readFirstLineFromFile(String path) throws IOException {  
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {  
        return br.readLine();  
    }  
    # the resource declared in the try-with-resources statement is a BufferedReader  
    # it will be closed regardless of whether the try statement completes normally or abruptly
```

## Creating Your Own Exceptions (Custom Exception)

- It is also possible to create your own exception but you must be extends Exception class.
- You can create your own exception as follow

### Example:

```
class ageexception extends Exception  
{  
    int age;  
    String msg="";  
    public ageexception()  
    {  
    }  
    public ageexception(String str)  
    {  
        super(str);  
    }  
    public String toString()  
    {  
        if(age<18 || age>60)  
        {
```

```
        msg="invalid age";
    }
    return msg;
}
}
public class userageexception
{
    public static void main(String args[])
    {
        userageexception uae=new userageexception();
        uae.test();
    }
    public void test()
    {
        int i=25;
        try
        {
            if(i<18 || i>60)
            {
                throw new ageexception();
            }
            else
            {
                System.out.println("valid age");
            }
        }
        catch(ageexception e)
        {
            System.out.println(e);
        }
    }
}
```

## The finalize( ) Method

- Sometimes an object will need to perform some action when it is destroyed.
- For example, if an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed.
- To handle such situations, Java provides a mechanism called *finalization*
- By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector.

### Syntax:

```
protected void finalize()
{
    // finalization code here
}
```

## Threading

### What is Threading?

- Thread is the feature of mostly languages including Java. Threads allow the program to perform multiple tasks simultaneously.
- Process speed can be increased by using threads because the thread can stop or suspend a specific running process and start or resume the suspended processes.
- Multitasking or multiprogramming is delivered through the running of multiple threads concurrently.
- If your computer does not have multi-processors then the multi-threads really do not run concurrently.
- This example illustrates how to create a thread and how to implement the thread. In this example we will see that the program prints numbers from 1 to 10 line by line after 5 seconds which has been declared in the sleep function of the thread class.
- Sleep function contains the sleeping time in millisecond and in this program sleep function has contained 5000 millisecond mean 5 second time.
- There is sleep function must caught by the InterruptedException. So, this program used the InterruptedException which tells something the user if thread is failed or interrupted.

### Example:

```
public class Threads
{
    public static void main(String[] args)
    {
        Thread th = new Thread();
        System.out.println("Numbers are printing line by line
                           after 5 seconds : ");

        try
        {
            for(int i = 1;i <= 10;i++)
            {
                System.out.println(i);
                th.sleep(5000);
            }
        }
        catch(InterruptedException e)
        {
            System.out.println("Thread interrupted!");
            e.printStackTrace();
        }
    }
}
```

### Thread Life cycle

#### New state:

After the creation of Thread instance the thread is in this state but before the start() method invocation. At this point, the thread is considered not alive.

### **Runnable (Ready-to-run) state:**

A thread starts its life from Runnable state. A thread first enters runnable state after the invoking of start() method but a thread can return to this state after either running, waiting, sleeping or coming back from blocked state also. On this state a thread is waiting for a turn on the processor.

### **Running state:**

A thread is in running state that means the thread is currently executing. There are several ways to enter in Runnable state but there is only one way to enter in

### **Running state:**

the scheduler selects a thread from runnable pool.

### **Dead state:**

A thread can be considered dead when its run() method completes. If any thread comes on this state that means it cannot ever run again.

Blocked - A thread can enter in this state because of waiting for the resources that are held by another thread.

## **Creating Thread**

- There are two main ways of creating a thread.
- The first is to extend the Thread class and the second is to implement the Runnable interface.

### **Extending Thread:**

```
public class MyThread extends Thread
{
    public void run()
    {
        System.out.println("Thread executed!");
    }
    public static void main(String[] args)
    {
        Thread thread = new MyThread();
        thread.start();
    }
}
```

### **Implementing the Runnable interface:**

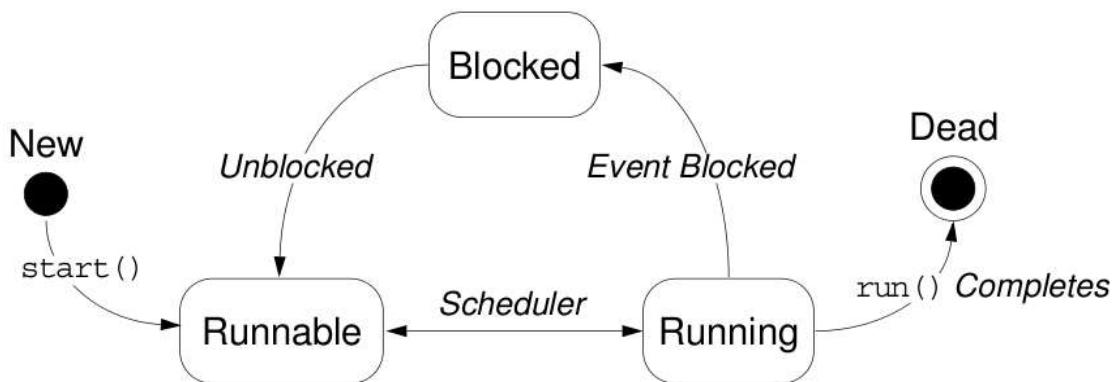
```
public class MyRunnable implements Runnable
{
    public void run()
    {
```

```

        System.out.println("Thread executed!");
    }
public static void main(String[] args)
{
    Thread thread = new Thread(new MyRunnable());
    thread.start();
}

```

## Thread Scheduling



}

### Basic control of Thread

Test threads:

    IsAlive()

Access thread priority:

    getPriority()

    SetPriority()

Put threads on hold:

    Thread.sleep() //static method

    join()

    Thread.yield() //static method

### **The join method**

# The join Method

```
public static void main(String[] args) {  
    Thread t = new Thread(new Runner());  
    t.start();  
    ...  
    // Do stuff in parallel with the other thread for a while  
    ...  
    // Wait here for the other thread to finish  
    try {  
        t.join();  
    } catch (InterruptedException e) {  
        // the other thread came back early  
    }  
    ...  
    // Now continue in this thread  
    ...  
}
```

## Thread Priority

In Java, thread scheduler can use the thread **priorities** in the form of **integer value** to each of its thread to determine the execution schedule of threads . Thread gets the **ready-to-run** state according to their priorities. The **thread scheduler** provides the CPU time to thread of highest priority during ready-to-run state.

Priorities are integer values from

- 1 (lowest priority given by the constant **Thread.MIN\_PRIORITY**)
- 10 (highest priority given by the constant **Thread.MAX\_PRIORITY**).
- The default priority is 5(**Thread.NORM\_PRIORITY**).

| Constant             | Description                                                |
|----------------------|------------------------------------------------------------|
| Thread.MIN_PRIORITY  | The maximum priority of any thread<br>(an int value of 10) |
| Thread.MAX_PRIORITY  | The minimum priority of any thread<br>(an int value of 1)  |
| Thread.NORM_PRIORITY | The normal priority of any thread (an int value of 5)      |

## Methods:

setPriority()

This method is used to set the priority of thread.

getPriority()

This method is used to get the priority of thread.

## Blocking a Thread

While the suspend( ), resume( ), and stop( ) methods defined by **Thread** class seem to be a perfectly reasonable and convenient approach to managing the execution of threads, they must not be used for new Java programs and obsolete in newer versions of Java.

The following example illustrates how the wait( ) and notify( ) methods that are inherited from Object can be used to control the execution of a thread.

This example is similar to the program in the previous section. However, the deprecated method calls have been removed. Let us consider the operation of this program.

The NewThread class contains a boolean instance variable named suspendFlag, which is used to control the execution of the thread. It is initialized to false by the constructor.

The run( ) method contains a synchronized statement block that checks suspendFlag. If that variable is true, the wait( ) method is invoked to suspend the execution of the thread. The mysuspend( ) method sets suspendFlag to true. The myresume( ) method sets suspendFlag to false and invokes notify( ) to wake up the thread. Finally, the main( ) method has been modified to invoke the mysuspend( ) and myresume( ) methods.

## Example:

```
class NewThread implements Runnable
{
    String name; // name of thread
    Thread t;
    boolean suspendFlag;
    NewThread(String threadname)
    {
        name = threadname;
        t = new Thread(this, name);
        System.out.println("New thread: " + t);
        suspendFlag = false;
        t.start(); // Start the thread
    }
    // This is the entry point for thread.
    public void run()
    {
        try
        {
```

```
for(int i = 15; i > 0; i--)
{
    System.out.println(name + ": " + i);
    Thread.sleep(200);
    synchronized(this)
    {
        while(suspendFlag)
        {
            wait();
        }
    }
}
catch (InterruptedException e)
{
    System.out.println(name + " interrupted.");
}
System.out.println(name + " exiting.");
}

void mysuspend()
{
    suspendFlag = true;
}
synchronized void myresume()
{
    suspendFlag = false;
    notify();
}
}

class SuspendResume
{
    public static void main(String args[])
    {
        NewThread ob1 = new NewThread("One");
        NewThread ob2 = new NewThread("Two");
        try
        {
            Thread.sleep(1000);
            ob1.mysuspend();
            System.out.println("Suspending thread One");
            Thread.sleep(1000);
            ob1.myresume();
            System.out.println("Resuming thread One");
            ob2.mysuspend();
            System.out.println("Suspending thread Two");
            Thread.sleep(1000);
            ob2.myresume();
            System.out.println("Resuming thread Two");
        }
    }
}
```

```
        }
        catch (InterruptedException e)
        {
            System.out.println("Main thread Interrupted");
        }
        // wait for threads to finish
        try
        {
            System.out.println("Waiting for threads to
                               finish.");
            ob1.t.join();
            ob2.t.join();
        }
        catch (InterruptedException e)
        {
            System.out.println("Main thread Interrupted");
        }
        System.out.println("Main thread exiting.");
    }
}
```

## Synchronization

- When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time.
- The process by which this is achieved is called *synchronization*. As you will see, Java provides unique, language-level support for it.
- Key to synchronization is the concept of the monitor (also called a *semaphore*).
- A *monitor* is an object that is used as a mutually exclusive lock, or *mutex*.
- Only one thread can *own* a monitor at a given time. When a thread acquires a lock, it is said to have *entered* the monitor.
- All other threads attempting to enter the locked monitor will be suspended until the first thread *exits* the monitor. These other threads are said to be *waiting* for the monitor.
- A thread that owns a monitor can reenter the same monitor if it so desires.

If you have worked with synchronization when using other languages, such as C or C++, you know that it can be a bit tricky to use. This is because most languages do not, themselves, support synchronization. Instead, to synchronize threads, your programs need to utilize operating system primitives. Fortunately, because Java implements synchronization through language elements, most of the complexity associated with synchronization has been eliminated. You can synchronize your code in either of two ways. Both involve the use of the **synchronized** keyword, and both are examined here.

**This program uses a synchronized block.**

```
class Callme
{
    void call(String msg)
    {
        System.out.print("[ " + msg);
    }
}
```

```
try
{
    Thread.sleep(1000);
}
catch (InterruptedException e)
{
    System.out.println("Interrupted");
}
System.out.println("]");

}

class Caller implements Runnable
{
    String msg;
    Callme target;
    Thread t;
    public Caller(Callme targ, String s)
    {
        target = targ;
        msg = s;
        t = new Thread(this);
        t.start();
    }
    // synchronize calls to call()
    public void run()
    {
        synchronized(target)
        {
            // synchronized block
            target.call(msg);
        }
    }
}
class Synch1
{
    public static void main(String args[])
    {
        Callme target = new Callme();
        Caller ob1 = new Caller(target, "Hello");
        Caller ob2 = new Caller(target, "Synchronized");
        Caller ob3 = new Caller(target, "World");
        // wait for threads to end
        try
        {
            ob1.t.join();
            ob2.t.join();
            ob3.t.join();
        }
    }
}
```

```
        catch(InterruptedException e)
        {
            System.out.println("Interrupted");
        }
    }
}
```

Here, the **call( )** method is not modified by **synchronized**. Instead, the **synchronized** statement is used inside **Caller's run( )** method. This causes the same correct output as the preceding example, because each thread waits for the prior one to finish before proceeding.

- A special type of error that you need to avoid that relates specifically to multitasking is *deadlock*, which occurs when two threads have a circular dependency on a pair of synchronized objects.
- For example, suppose one thread enters the monitor on object X and another thread enters the monitor on object Y. If the thread in X tries to call any synchronized method on Y, it will block as expected. However, if the thread in Y, in turn, tries to call any synchronized method on X, the thread waits forever, because to access X, it would have to release its own lock on Y so that the first thread could complete.
- Deadlock is a difficult error to debug for two reasons:
  - I. In general, it occurs only rarely, when the two threads time-slice in just the right way.
  - II. It may involve more than two threads and two synchronized objects. (That is, deadlock can occur through a more convoluted sequence of events than just described.)

**Example:**

```
class A
{
    synchronized void foo(B b)
    {
        String name = Thread.currentThread().getName();
        System.out.println(name + " entered A.foo");
        try
        {
            Thread.sleep(1000);
        }
        catch(Exception e)
        {
            System.out.println("A Interrupted");
        }
        System.out.println(name + " trying to call B.last()");
        b.last();
    }
    synchronized void last()
    {
        System.out.println("Inside A.last");
    }
}
class B
{
```

```
synchronized void bar(A a)
{
    String name = Thread.currentThread().getName();
    System.out.println(name + " entered B.bar");
    try
    {
        Thread.sleep(1000);
    }
    catch(Exception e)
    {
        System.out.println("B Interrupted");
    }
    System.out.println(name + " trying to call A.last()");
    a.last();
}

synchronized void last()
{
    System.out.println("Inside A.last");
}

class Deadlock implements Runnable
{
    A a = new A();
    B b = new B();
    Deadlock()
    {
        Thread.currentThread().setName("MainThread");
        Thread t = new Thread(this, "RacingThread");
        t.start();
        a.foo(b); // get lock on a in this thread.
        System.out.println("Back in main thread");
    }
    public void run()
    {
        b.bar(a); // get lock on b in other thread.
        System.out.println("Back in other thread");
    }
    public static void main(String args[])
    {
        new Deadlock();
    }
}
```

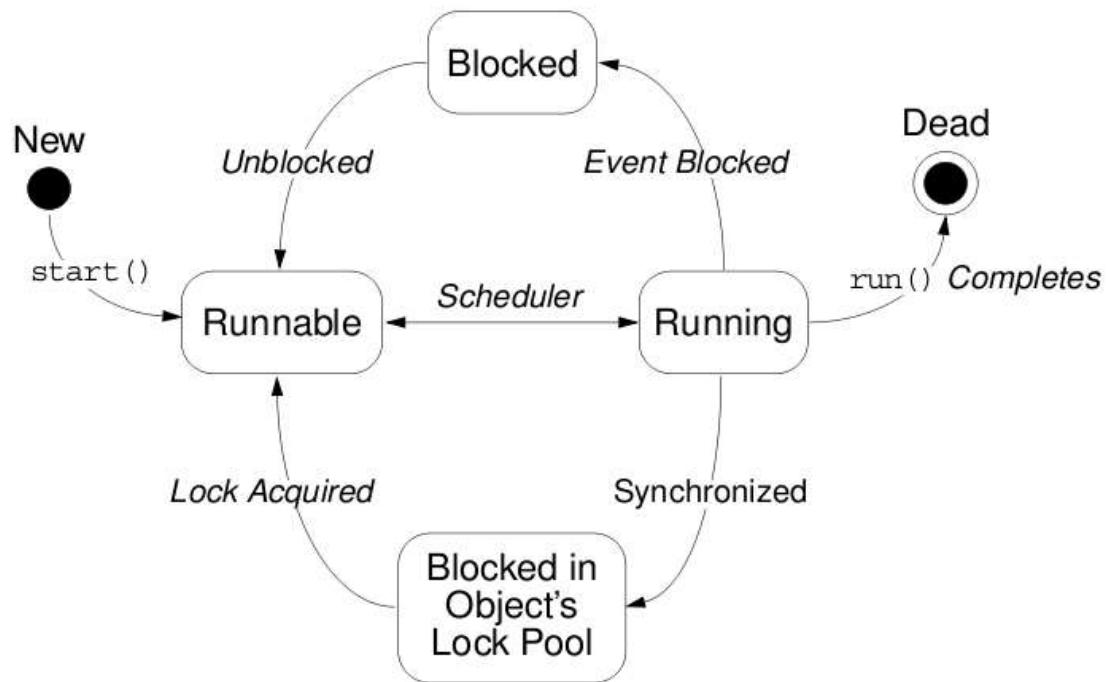
**Output:**

```
MainThread entered A.foo
RacingThread entered B.bar
MainThread trying to call B.last()
RacingThread trying to call A.last()
```

Because the program has deadlocked, you need to press CTRL-C to end the program. You can see a full thread and monitor cache dump by pressing CTRL-BREAK on a PC . You will see that **RacingThread** owns the monitor on **b**, while it is waiting for the monitor on **a**. At the same time, **MainThread** owns **a** and is waiting to get **b**. This program will never complete. As this example illustrates, if your multithreaded program locks up occasionally, deadlock is one of the first conditions that you should check for.

## Thread State Diagram with Synchronization

## Thread State Diagram With Synchronization



## Dead Lock

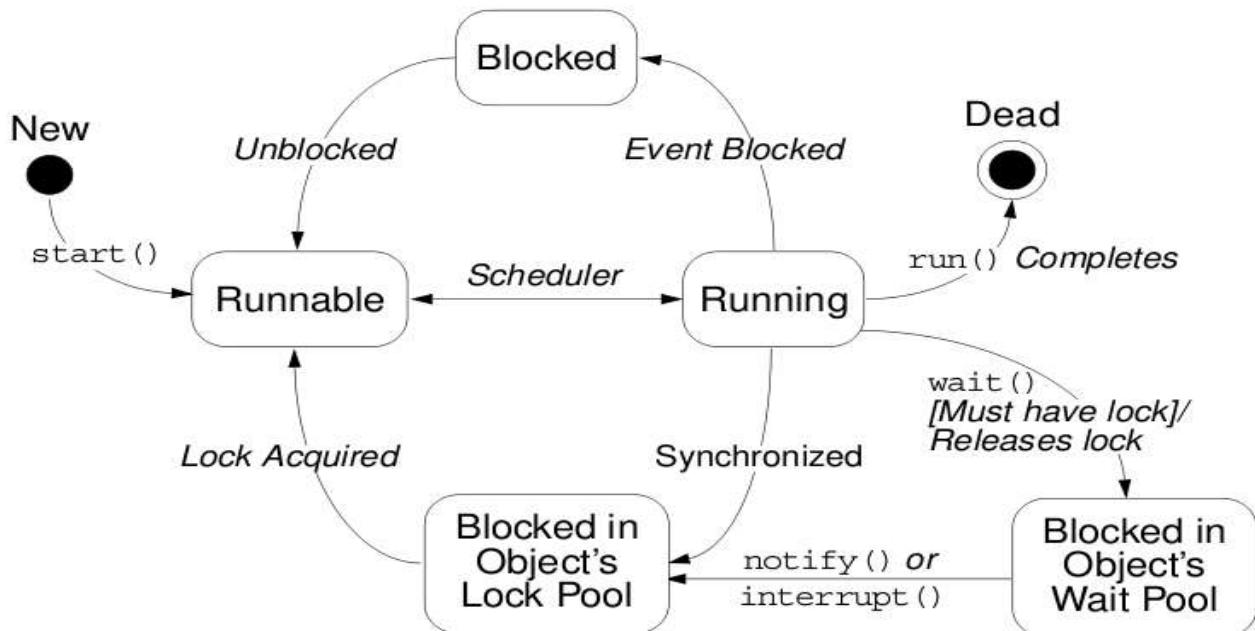
- When two threads or processes are waiting for each other to release the resource or object they holds and so are blocked forever. This situation is called deadlock.
- For example if one thread is holding the lock on some object that the other thread is waiting for and the other thread is holding lock on some object the first one is waiting for then they both will wait for each other to release the object they need to finish their operation but no one will release the hold object and so they will wait for each other forever.

## Thread Interaction

- Scenario: Consider yourself and a cab driver as two threads.
- The problem: How do you determine when you are at your destination?
- The solution: You notify the cab driver of your destination and relax. The driver drives and notifies you upon arrival at your destination.
- Thread interactions include:
- The wait and notify methods
  - The pools:
    - Wait pool
    - Lock pool

### Thread State diagram with wait and notify

## Thread State Diagram With wait and notify



### Monitor model for synchronization

- Leave shared data in a consistent state.
- Ensure programs cannot deadlock.
- Do not put threads expecting different notifications in the same wait pool.

### The Producer Class

```
package com.interthreadcommunication;
```

```
public class Producer implements Runnable {
SyncTest sync;
Producer(SyncTest sync) {
```

```
this.sync = sync;
new Thread(this, "Producer").start();
}
public void run() {
int i = 0;
while(true) {
sync.put(i++);
}
}
```

## The Consumer class

```
package com.interthreadcommunication;

public class SyncTest {
int n;
boolean valueSet = false;
synchronized int get() {
if (!valueSet)
try {
wait();
} catch (InterruptedException e) {
System.out.println("InterruptedException caught");
}
System.out.println("Got: " + n);
valueSet = false;
notify();
return n;
}
```

## The SyncStack Class

```
public class SyncStack {
private List<Character> buffer=new ArrayList<Character>(400);
public synchronized char pop()
{ //pop code here }
public synchronized char push(char c)
{ //push code here }
```

### The pop method

```
public synchronized char pop() {
char c;
while(buffer.size()==0){
try{ this.wait();
}catch(InterreptedException e){
}
c=buffer.remove(buffer.size()-1);
return c;
}
```

## The push method

```
public synchronized char push(char c)
{
    this.notify();
    buffer.add(c);
}
```

## The SyncTest Class

```
package com.interthreadcommunication;
public class SyncTest {
int n;
boolean valueSet = false;

synchronized int get() {
    if (!valueSet)
        try {
            wait();
        } catch (InterruptedException e) {
            System.out.println("InterruptedException caught");
        }
    System.out.println("Got: " + n);
    valueSet = false;
    notify();
    return n;
}
synchronized void put(int n) {
    if (valueSet)
        try {
            wait();
        } catch (InterruptedException e) {
            System.out.println("InterruptedException caught");
        }
    this.n = n;
    valueSet = true;
    System.out.println("Put: " + n);
    notify();
}
}
```

## Graphical User Interface in Java

As a platform-independent programming language, how is Java technology used to make the graphical user interface (GUI) platform-independent?

### AWT

The AWT contains numerous classes and methods that allow you to create and manage windows. It is also the foundation upon which Swing is built. The AWT is quite large and a full description would easily fill an entire book.

Therefore, it is not possible to describe in detail every AWT class, method, or instance variable. Although a common use of the AWT is in applets, it is also used to create stand-alone windows that run in a GUI environment, such as Windows.

## Exploring `java.awt` package

The AWT defines windows according to a class hierarchy that adds functionality and specificity with each level. The two most common windows are those derived from **Panel**, which is used by applets, and those derived from **Frame**, which creates a standard application window.

Much of the functionality of these windows is derived from their parent classes. Thus, a description of the class hierarchies relating to these two classes is fundamental to their understanding.

### Component:

- At the top of the AWT hierarchy is the **Component** class. **Component** is an abstract class that encapsulates all of the attributes of a visual component.
- All user interface elements that are displayed on the screen and that interact with the user are subclasses of **Component**.
- It defines over a hundred public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting.
- A **Component** object is responsible for remembering the current foreground and background colors and the currently selected text font.

### Container

- The **Container** class is a subclass of **Component**. It has additional methods that allow other **Component** objects to be nested within it.
- Other **Container** objects can be stored inside of a **Container** (since they are themselves instances of **Component**).
- This makes for a multileveled containment system. A container is responsible for laying out (that is, positioning) any components that it contains.

### Positioning Components

- The position and size of a component in a container is determined by a layout manager.
- You can control the size or position of components by disabling the layout manager.
- You must then use `setLocation()`, `setSize()`, or `setBounds()` on components to locate them in the container.

### Frame

- **Frame** encapsulates what is commonly thought of as a “window.” It is a subclass of **Window** and has a title bar, menu bar, borders, and resizing corners.
- If you create a **Frame** object from within an applet, it will contain a warning message, such as “Java Applet Window,” to the user that an applet window has been created.
- This message warns users that the window they see was started by an applet and not by software running on their computer. (An applet that could masquerade as a host-based application could be used to obtain passwords and other sensitive information without the user’s knowledge.)

- When a Frame window is created by a stand-alone application rather than an applet, a normal window is created.

## Panel

- The **Panel** class is a concrete subclass of **Container**. It doesn't add any new methods; it simply implements **Container**.
- A **Panel** may be thought of as a recursively nestable, concrete screen component. **Panel** is the superclass for **Applet**. When screen output is directed to an applet, it is drawn on the surface of a **Panel** object.
- In essence, a **Panel** is a window that does not contain a title bar, menu bar, or border. This is why you don't see these items when an applet is run inside a browser.
- When you run an applet using an applet viewer, the applet viewer provides the title and border.
- Other components can be added to a **Panel** object by its **add( )** method (inherited from **Container**).
- Once these components have been added, you can position and resize them manually using the **setLocation( )**, **setSize( )**, **setPreferredSize( )**, or **setBounds( )** methods defined by **Component**.

## Window:

- The **Window** class creates a top-level window. *A top-level window* is not contained within any other object; it sits directly on the desktop.
- Generally, you won't create **Window** objects directly. Instead, you will use a subclass of **Window** called **Frame**, described next.

## Canvas:

- Although it is not part of the hierarchy for applet or frame windows, there is one other type of window that you will find valuable: **Canvas**.
- **Canvas** encapsulates a blank window upon which you can draw.

## Dialog:

- Often, you will want to use a *dialog box* to hold a set of related controls.
- Dialog boxes are primarily used to obtain user input and are often child windows of a top-level window.

## Applet:

- *Applets* are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document.
- After an applet arrives on the client, it has limited access to resources so that it can produce a graphical user interface and run complex computations without introducing the risk of viruses or breaching data integrity.

## Label

The easiest control to use is a label. A *label* is an object of type **Label**, and it contains a string, which it displays. Labels are passive controls that do not support any interaction with the user.

**Label** defines the following constructors:

**Label( )** throws HeadlessException

The first version creates a blank label.

`Label(String str) throws HeadlessException`

The second version creates a label that contains the string specified by *str*.

`Label(String str, int how) throws HeadlessException`

This string is left-justified. The third version creates a label that contains the string specified by *str* using the alignment specified by *how*. The value of *how* must be one of these three constants:

`Label.LEFT, Label.RIGHT, or Label.CENTER.`

#### **Methods:**

`void setText(String str)`

`String getText( )`

`void setAlignment(int how)`

`int getAlignment( )`

## **Button**

Perhaps the most widely used control is the push button. A *push button* is a component that contains a label and that generates an event when it is pressed. Push buttons are objects of type **Button**.

**Button** defines following constructors:

`Button( ) throws HeadlessException`

The first version creates an empty button.

`Button(String str) throws HeadlessException`

The second creates a button that contains *str* as a label.

#### **Methods:**

`voidsetLabel(String str)`

`String getLabel( )`

## **Check Boxes**

A *check box* is a control that is used to turn an option on or off. It consists of a small box that can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group. Check boxes are objects of the **Checkbox** class.

**Check Box** defines following constructors:

`Checkbox( ) throws HeadlessException`

The first form creates a check box whose label is initially blank.

`Checkbox(String str) throws HeadlessException`

The second form creates a check box whose label is specified by *str*.

The state of the check box is unchecked.

`Checkbox(String str, boolean on) throws HeadlessException`

The third form allows you to set the initial state of the check box.

If *on* is **true**, the check box is initially checked; otherwise, it is cleared.

`Checkbox(String str, boolean on, CheckboxGroup cbGroup) throw HeadlessException`

`Checkbox(String str, CheckboxGroup cbGroup, boolean on) throws`

`HeadlessException`

The fourth and fifth forms create a check box whose label is specified

by *str* and whose group is specified by *cbGroup*. If this check box is not part of a group, then *cbGroup* must be **null**. (Check box groups are described in the next section.) The value of *on* determines the initial state of the check box.

## Methods:

```
boolean getState( )
void setState(boolean on)
String getLabel( )
void setLabel(String str)
```

## Checkbox Group

It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time. These check boxes are often called *radio buttons*, because they act like the station selector on a car radio—only one station can be selected at any one time. To create a set of mutually exclusive check boxes, you must first define the group to which they will belong and then specify that group when you construct the check boxes. Check box groups are objects of type **CheckboxGroup**.

**Checkbox Group** defines following constructors:

```
CheckboxGroup()
```

## Methods:

```
Checkbox getSelectedCheckbox( )
void setSelectedCheckbox(Checkbox which)
```

## Choice

The **Choice** class is used to create a *pop-up list* of items from which the user may choose. Thus, a **Choice** control is a form of menu. When inactive, a **Choice** component takes up only enough space to show the currently selected item. When the user clicks on it, the whole list of choices pops up, and a new selection can be made. Each item in the list is a string that appears as a left-justified label in the order it is added to the **Choice** object.

**Choice** defines following constructors:

```
Choice()
```

## Methods:

```
void add(String name)
String getSelectedItem( )
int getSelectedIndex( )
int getItemCount( )
void select(int index)
void select(String name)
String getItem(int index)
```

## List

The **List** class provides a compact, multiple-choice, scrolling selection list. Unlike the **Choice** object, which shows only the single selected item in the menu, a **List** object can be constructed to show any number of choices in the visible window. It can also be created to allow multiple selections.

**List** defines following constructors:

`List()` throws HeadlessException

The first version creates a **List** control that allows only one item to be selected at any one time.

`List(int numRows)` throws HeadlessException

In the second form, the value of *numRows* specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).

`List(int numRows, boolean multipleSelect)` throws HeadlessException

In the third form, if *multipleSelect* is **true**, then the user may select two or more items at a time. If it is **false**, then only one item may be selected.

**Methods:**

```
void add(String name)
void add(String name, int index)
String getSelectedItem()
int getSelectedIndex()
String[ ] getSelectedItems()
int[ ] getSelectedIndexes()
int getItemCount()
void select(int index)
String getItem(int index)
```

## Scroll Bar

**Scroll bars** are used to select continuous values between a specified minimum and maximum. Scroll bars may be oriented horizontally or vertically. A scroll bar is actually a composite of several individual parts. Each end has an arrow that you can click to move the current value of the scroll bar one unit in the direction of the arrow. The current value of the scroll bar relative to its minimum and maximum values is indicated by the *slider box* (or *thumb*) for the scroll bar. The slider box can be dragged by the user to a new position. The scroll bar will then reflect this value. In the background space on either side of the thumb, the user can click to cause the thumb to jump in that direction by some increment larger than 1. Typically, this action translates into some form of page up and page down. Scroll bars are encapsulated by the **Scrollbar** class.

**Scroll Bar** defines following constructors:

`Scrollbar()` throws HeadlessException

The first form creates a vertical scroll bar.

`Scrollbar(int style)` throws HeadlessException

The second and third forms allow you to specify the orientation of the scroll bar. If *style* is **Scrollbar.VERTICAL**, a vertical scroll bar is created. If *style* is **Scrollbar.HORIZONTAL**, the scroll bar is horizontal.

`Scrollbar(int style, int initialValue, int thumbSize, int min, int max)`

throws HeadlessException

In the third form of the constructor, the initial value of the scroll bar is passed in *initialValue*. The number of units represented by the height of the thumb is passed in *thumbSize*. The minimum and maximum values for the scroll bar are specified by *min* and *max*.

#### Methods:

```
void setValues(int initialValue, int thumbSize, int min, int max)
int getValue( )
void setValue(int newValue)
int getMinimum( )
int getMaximum( )
void setUnitIncrement(int newIncr)
void setBlockIncrement(int newIncr)
```

### Text Field

The **TextField** class implements a single-line text-entry area, usually called an *edit control*. Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections. **TextField** is a subclass of **TextComponent**.

**TextField** defines the following constructors:

TextField( ) throws HeadlessException The first version creates a default text field.  
TextField(int numChars) throws HeadlessException  
The second form creates a text field that is *numChars* characters wide.  
TextField(String str) throws HeadlessException  
The third form initializes the text field with the string contained in *str*.  
TextField(String str, int numChars) throws HeadlessException  
The fourth form initializes a text field and sets its width.

#### Methods:

```
String getText( )
void setText(String str)
String getSelectedText( )
void select(int startIndex, int endIndex)
boolean isEditable( )
void setEditable(boolean canEdit)
void setEchoChar(char ch)
boolean echoCharIsSet( )
char getEchoChar( )
```

### Text Area

Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called **TextArea**

**Text Area** defines following constructor:

TextArea( ) throws HeadlessException  
TextArea(int numLines, int numChars) throws HeadlessException  
TextArea(String str) throws HeadlessException  
TextArea(String str, int numLines, int numChars) throws HeadlessException

`TextArea(String str, int numLines, int numChars, int sBars)` throws HeadlessException

Here, `numLines` specifies the height, in lines, of the text area, and `numChars` specifies its width, in characters. Initial text can be specified by `str`. In the fifth form, you can specify the scroll bars that you want the control to have. `sBars` must be one of these values:

`SCROLLBARS_BOTH`  
`SCROLLBARS_NONE`  
`SCROLLBARS_HORIZONTAL_ONLY`  
`SCROLLBARS_VERTICAL_ONLY`

**Methods:**

```
void append(String str)
void insert(String str, int index)
void replaceRange(String str, int startIndex, int endIndex)
```

**Example 1:**

```
import java.awt.Frame;
import java.awt.TextField;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
public class KeyEventDemo extends Frame implements KeyListener
{
    TextField t1,t2,t3;

    public void KeyEventDemo()
    {
        t1 = new TextField();
        t2 = new TextField();
        t3 = new TextField();
        t3.setEditable(false);
        setTitle("Sample Frame");
        setVisible(true);
        setLayout(new FlowLayout());
        add(t1);
        add(t2);
        add(t3);

        t3.addKeyListener(this);
    }
    public void keyTyped(KeyEvent e)
    {
    }

    public void keyPressed(KeyEvent e)
    {

        if(e.getKeyCode()!=KeyEvent.VK_ESCAPE)
        {
            double x = Double.parseDouble(t1.getText());
        }
    }
}
```

```
        double y = Double.parseDouble(t2.getText());
        double z = x + y;
        t3.setText(String.valueOf(z));
    }
}

public void keyReleased(KeyEvent e)
{
}

public static void main(String[] args)
{
    KeyEventDemo key = new KeyEventDemo();
}
}
```

**Example 2:**

```
import java.awt.Choice;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

public class SampleChoiceExample extends Frame implements
    ItemListener
{
    private Label l;
    private Choice ch;

    public SampleChoiceExample()
    {
        ch = new Choice();
        addData();
        l = new Label("India");
        setTitle("Choice Demo");
        setLayout(new FlowLayout());
        add(ch);
        add(l);
        pack();
        setVisible(true);

        ch.addItemListener(this);
    }

    private void addData()
    {
        ch.add("India");
```

```
        ch.add("China");
        ch.add("Sri Lanka");
        ch.add("Japan");
    }
    public void itemStateChanged(ItemEvent e)
    {
        if(e.getSource() == ch)
        {
            String item = ch.getSelectedItem();
            l.setText(item);
        }
    }
    public static void main(String[] args)
    {
        SampleChoiceExample exam = new
            SampleChoiceExample();
    }
}
```

## Menubars and Menus

An application window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in the AWT by the following classes: **MenuBar**, **Menu**, and **MenuItem**. In general, a menu bar contains one or more **Menu** objects. Each **Menu** object contains a list of **MenuItem** objects. Each **MenuItem** object represents something that can be selected by the user. Since **Menu** is a subclass of **MenuItem**, a hierarchy of nested submenus can be created.

It is also possible to include checkable menu items. These are menu options of type **CheckboxMenuItem** and will have a check mark next to them when they are selected. To create a menu bar, first create an instance of **MenuBar**. This class only defines the default constructor. Next, create instances of **Menu** that will define the selections displayed on the bar.

**Menu** defines following constructors:

```
Menu( ) throws HeadlessException
Menu(String optionName) throws HeadlessException
Menu(String optionName, boolean removable) throws HeadlessException
    Here, optionName specifies the name of the menu selection. If
    removable is true, the menu can be removed and allowed to float free.
    Otherwise, it will remain attached to the menu bar. (Removable
    menus are implementation-dependent.) The first form creates an
    empty menu.
```

**MenuItem** defines following constructors:

```
MenuItem( ) throws HeadlessException
MenuItem(String itemName) throws HeadlessException
```

```
MenuItem(String itemName, MenuShortcut keyAccel) throws  
   HeadlessException
```

Here, *itemName* is the name shown in the menu, and *keyAccel* is the menu shortcut for this item.

**Methods:**

```
void setEnabled(boolean enabledFlag)  
boolean isEnabled()  
void setLabel(String newName)  
String getLabel()
```

You can create a checkable menu item by using a subclass of **MenuItem** called **CheckboxMenuItem**. It has these constructors:

```
CheckboxMenuItem() throws HeadlessException  
CheckboxMenuItem(String itemName) throws HeadlessException  
CheckboxMenuItem(String itemName, boolean on) throws  
   HeadlessException
```

Here, *itemName* is the name shown in the menu. Checkable items operate as toggles. Each time one is selected, its state changes. In the first two forms, the checkable entry is unchecked. In the third form, if *on* is **true**, the checkable entry is initially checked. Otherwise, it is cleared.

**Methods:**

```
boolean getState()  
void setState(boolean checked)
```

Once you have created a menu item, you must add the item to a **Menu** object by using **add( )**, which has the following general form:

```
MenuItem add(MenuItem item)
```

Once you have added all items to a **Menu** object, you can add that object to the menu bar by using this version of **add( )** defined by **MenuBar**:

```
Menu add(Menu menu)
```

**Example:**

```
import java.awt.FlowLayout;  
import java.awt.Frame;  
import java.awt.Label;  
import java.awt.Menu;  
import java.awtMenuBar;  
import java.awt.MenuItem;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;
```

```
public class MenuExample extends Frame implements ActionListener
{
    MenuBar menuBar;
    Menu file, edit, subMenu;
    MenuItem open, save, dash, sample, exit, cut, copy, paste;
    Label label;
    public MenuExample()
    {
        setTitle("My Menu Sample Frame");
        menuBar = new MenuBar();
        setMenuBar(menuBar);

        file = new Menu("File");

        open = new MenuItem("Open");
        save = new MenuItem("Save");
        dash = new MenuItem("- ");
        exit = new MenuItem("Exit");

        //This is for Sub Menu
        subMenu = new Menu("Sub Menu");
        sample = new MenuItem("Sample Sub Menu");
        subMenu.add(sample);

        file.add(open);
        file.add(save);
        file.add(dash);
        file.add(subMenu);
        file.add(exit);

        menuBar.add(file);

        edit = new Menu("Edit");

        cut = new MenuItem("Cut");
        copy = new MenuItem("Copy");
        paste = new MenuItem("Paste");

        edit.add(cut);
        edit.add(copy);
        edit.add(paste);

        menuBar.add(edit);

        setLayout(new FlowLayout());

        label = new Label("", Label.CENTER);
```

```
        add(label);
        setSize(300, 300);
        setVisible(true);

        open.addActionListener(this);
        save.addActionListener(this);
        sample.addActionListener(this);
        exit.addActionListener(this);
        cut.addActionListener(this);
        copy.addActionListener(this);
        paste.addActionListener(this);

    }
    public static void main(String[] args)
    {
        MenuExample example = new MenuExample();
    }
    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource().equals(open))
        {
            label.setText(open.getLabel()+" is clicked");
        }
        if(e.getSource().equals(save))
        {
            label.setText(save.getLabel()+" is clicked");
        }
        if(e.getSource().equals(sample))
        {
            label.setText(sample.getLabel()+" is clicked");
        }
        if(e.getSource().equals(exit))
        {
            label.setText(exit.getLabel()+" is clicked");
        }
        if(e.getSource().equals(cut))
        {
            label.setText(cut.getLabel()+" is clicked");
        }
        if(e.getSource().equals(copy))
        {
            label.setText(copy.getLabel()+" is clicked");
        }

        if(e.getSource().equals(paste))
        {
            label.setText(paste.getLabel()+" is clicked");
        }
    }
}
```

```
    }  
}
```

## Dialog Boxes

Often, you will want to use a *dialog box* to hold a set of related controls. Dialog boxes are primarily used to obtain user input and are often child windows of a top-level window. Dialog boxes don't have menu bars, but in other respects, they function like frame windows. (You can add controls to them, for example, in the same way that you add controls to a frame window.) Dialog boxes may be modal or modeless. When a *modal* dialog box is active, all input is directed to it until it is closed.

This means that you cannot access other parts of your program until you have closed the dialog box. When a *modeless* dialog box is active, input focus can be directed to another window in your program. Thus, other parts of your program remain active and accessible. Dialog boxes are of type **Dialog**.

**Dialog** defines following constructors:

```
Dialog(Frame parentWindow, boolean mode)  
Dialog(Frame parentWindow, String title, boolean mode)
```

### Example:

```
import java.awt.GridLayout;  
import java.awt.Label;  
import java.util.Random;  
import java.awt.Button;  
import java.awt.FileDialog;  
import java.awt.FlowLayout;  
import java.awt.Frame;  
import java.awt.Label;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
  
public class SampleDialog extends Dialog implements ActionListener  
{  
    private Label l,l1;  
    private Button ok, exit;  
    public SampleDialog(Frame parent, String title)  
    {  
        super(parent, title, false);  
        l = new Label("Press the button");  
        l1 = new Label();  
        ok = new Button("OK");  
        exit = new Button("Exit");  
        setLayout(new GridLayout(2, 2));  
        add(l);  
        add(ok);  
        add(l1);  
        add(exit);  
        setSize(600, 100);  
        ok.addActionListener(this);  
    }
```

```
        exit.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e)
    {
        if(e.getSource().equals(ok))
        {
            Random r = new Random();
            l1.setText(String.valueOf("Random Generated
Number is :- "+r.nextInt()));
        }
        else if(e.getSource().equals(exit))
        {
            setVisible(false);
        }
    }
}

public class TestDialog extends Frame implements ActionListener
{
    private Label l;
    private Button b;
    public TestDialog()
    {
        setTitle("Demo");
        setResizable(false);
        l = new Label("Press the button");
        b = new Button("Press");
        setLayout(new FlowLayout());
        add(l);
        add(b);
        b.addActionListener(this);
        addWindowListener(new WindowAdapter()
        {
            @Override
            public void windowClosing(WindowEvent e)
            {
                super.windowClosing(e);
                System.exit(0);
            }
        });
        setSize(300, 300);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
    {
        SampleDialog sd = new SampleDialog(this, "Dialog
Box");
    }
}
```

```
        sd.setVisible(true);
    }
    public static void main(String[] args)
    {
        TestDialog td = new TestDialog();
    }
}
```

## File Dialog

Java provides a built-in dialog box that lets the user specify a file. To create a file dialog box, instantiate an object of type **FileDialog**. This causes a file dialog box to be displayed. Usually, this is the standard file dialog box provided by the operating system. Here are three

**FileDialog** defines following constructors:

```
FileDialog(Frame parent)
FileDialog(Frame parent, String boxName)
FileDialog(Frame parent, String boxName, int how)
```

Here, *parent* is the owner of the dialog box. The *boxName* parameter specifies the name displayed in the box's title bar. If *boxName* is omitted, the title of the dialog box is empty. If *how* is **FileDialog.LOAD**, then the box is selecting a file for reading. If *how* is **FileDialog.SAVE**, the box is selecting a file for writing. If *how* is omitted, the box is selecting a file for reading. **FileDialog** provides methods that allow you to determine the name of the file and its path as selected by the user.

```
String getDirectory()
String getFile()
```

### Example:

```
import java.awt.Button;
import java.awt.FileDialog;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class TestDialog extends Frame implements ActionListener
{
    private Label l;
    private Button b;
    public TestDialog()
    {
        setTitle("Demo");
        setResizable(false);
        l = new Label("Press the button");
        b = new Button("Press");
    }
}
```

```
setLayout(new FlowLayout());
add(l);
add(b);
b.addActionListener(this);
addWindowListener(new WindowAdapter()
{
    @Override
    public void windowClosing(WindowEvent e)
    {
        super.windowClosing(e);
        System.exit(0);
    }
});
setSize(300, 300);
setVisible(true);
}
public void actionPerformed(ActionEvent e)
{
    FileDialog fd = new FileDialog(this, "File Dialog");
    fd.setVisible(true);
}
public static void main(String[] args)
{
    TestDialog td = new TestDialog();
}
}
```

## Save Dialog

```
import java.awt.Button;
import java.awt.FileDialog;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class TestDialog extends Frame implements ActionListener
{
    private Label l;
    private Button b;
    public TestDialog()
    {
        setTitle("Demo");
```

```
        setResizable(false);
        l = new Label("Press the button");
        b = new Button("Press");
        setLayout(new FlowLayout());
        add(l);
        add(b);
        b.addActionListener(this);
        addWindowListener(new WindowAdapter()
        {
            @Override
            public void windowClosing(WindowEvent e)
            {
                super.windowClosing(e);
                System.exit(0);
            }
        });
        setSize(300, 300);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e)
    {
        FileDialog fd = new FileDialog(this, "Save
   Dialog",FileDialog.SAVE);
        fd.setVisible(true);
    }

    public static void main(String[] args)
    {
        TestDialog td = new TestDialog();
    }
}
```

## Painting in AWT

The AWT supports a rich assortment of graphics methods. All graphics are drawn relative to a window. This can be the main window of an applet, a child window of an applet, or a stand-alone application window. The origin of each window is at the top-left corner and is 0,0. Coordinates are specified in pixels. All output to a window takes place through a graphics context. A *graphics context* is encapsulated by the **Graphics** class and is obtained in two ways:

- It is passed to an applet when one of its various methods, such as **paint( )** or **update( )**, is called.
- It is returned by the **getGraphics( )** method of **Component**.

For the sake of convenience the remainder of the examples in this chapter will demonstrate graphics in the main applet window. However, the same techniques will apply to any other window.

The **Graphics** class defines a number of drawing functions. Each shape can be drawn edge-only or filled. Objects are drawn and filled in the currently selected graphics color, which is black by default. When a graphics object is drawn that exceeds the dimensions of the window, output is automatically clipped. Let's take a look at several of the drawing methods.

## Methods:

`void drawLine(int startX, int startY, int endX, int endY)`

`drawLine( )` displays a line in the current drawing color that begins at *startX*, *startY* and ends at *endX*, *endY*.

`void drawRect(int top, int left, int width, int height)`

`void fillRect(int top, int left, int width, int height)`

The `drawRect( )` and `fillRect( )` methods display an outlined and filled rectangle, respectively.

`void drawRoundRect(int top, int left, int width, int height,int xDiam, int yDiam)`

`void fillRoundRect(int top, int left, int width, int height,int xDiam, int yDiam)`

To draw a rounded rectangle, use `drawRoundRect( )` or `fillRoundRect( )`

`void drawOval(int top, int left, int width, int height)`

`void fillOval(int top, int left, int width, int height)`

To draw an ellipse, use `drawOval( )`. To fill an ellipse, use `fillOval( )`

`void drawArc(int top, int left, int width, int height, int startAngle,int sweepAngle)`

`void fillArc(int top, int left, int width, int height, int startAngle,int sweepAngle)`

Arcs can be drawn with `drawArc( )` and `fillArc( )`

`void drawPolygon(int x[ ], int y[ ], int numPoints)`

`void fillPolygon(int x[ ], int y[ ], int numPoints)`

It is possible to draw arbitrarily shaped figures using `drawPolygon( )` and `fillPolygon( )`

## Example 1:

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Toolkit;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;

public class MouseCursorDemo extends Applet implements
                                MouseListener,MouseMotionListener
{
    private String msg;
    private int x, y;
```

```
@Override
public void init()
{
    Toolkit t = Toolkit.getDefaultToolkit();
    Dimension d = t.getScreenSize();
    setSize(d.width, d.height);
    msg = "";
    addMouseListener(this);
    addMouseMotionListener(this);
}
@Override
public void paint(Graphics g)
{
    g.setColor(Color.red);
    g.drawString(msg, x, y);
}
public void mouseClicked(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    msg = "Mouse Clicked at ["+x+","+y+"]";
    repaint();
}
public void mousePressed(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    msg = "Mouse Pressed at ["+x+","+y+"]";
    repaint();
}
public void mouseReleased(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    msg = "Mouse Released at ["+x+","+y+"]";
    repaint();
}
public void mouseEntered(MouseEvent e)
{
    x = e.getX();
    y = e.getY();
    msg = "Mouse Entered at ["+x+","+y+"]";
    repaint();
}
public void mouseExited(MouseEvent e)
{
}
public void mouseDragged(MouseEvent e)
```

```
{  
    x = e.getX();  
    y = e.getY();  
    msg = "Mouse Dragged at ["+x+","+y+"]";  
    repaint();  
}  
public void mouseMoved(MouseEvent e)  
{  
    x = e.getX();  
    y = e.getY();  
    msg = "Mouse Moved at ["+x+","+y+"]";  
    repaint();  
}  
}
```

**Example 2:**

```
import java.applet.Applet;  
import java.awt.Color;  
import java.awt.Dimension;  
import java.awt.Graphics;  
import java.awt.Toolkit;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;  
import java.util.Random;  
  
public class MouseCircle extends Applet  
{  
    private int x, y, red, green, blue;  
  
    @Override  
    public void init()  
    {  
        Toolkit t = Toolkit.getDefaultToolkit();  
        Dimension d = t.getScreenSize();  
        setSize(d.width, d.height);  
        addMouseListener(new MouseAdapter()  
        {  
            @Override  
            public void mouseClicked(MouseEvent me)  
            {  
                x = me.getX();  
                y = me.getY();  
                repaint();  
            }  
        });  
        addMouseMotionListener(new MouseAdapter()  
        {  
            @Override
```

```
public void mouseDragged(MouseEvent me)
{
    x = me.getX();
    y = me.getY();
    repaint();
}
});
}
@Override
public void paint(Graphics g)
{
    Random r = new Random();
    red = r.nextInt(255);
    green = r.nextInt(255);
    blue = r.nextInt(255);
    g.setColor(new Color(red, green, blue));
    g.fillOval(x-25, y-25, 50, 50);
}
}
```

## Creating Top-level Containers

Here, given examples to show how many ways we can create a top-level containers for our application.

### Example 1:

```
import java.awt.*;
public class FrameExample
{
    private Frame f;
    public FrameExample()
    {
        f = new Frame("Hello Out There!");
    }
    public void launchFrame()
    {
        f.setSize(170,170);
        f.setBackground(Color.blue);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        FrameExample guiWindow = new FrameExample();
        guiWindow.launchFrame();
    }
}
```

### Example 2:

```
import java.awt.*;
public class FrameExample extends Frame
```

```
{  
    public FrameExample()  
    {  
        launchFrame();  
    }  
    public void launchFrame()  
    {  
        setSize(170,170);  
        setBackground(Color.blue);  
        setVisible(true);  
    }  
    public static void main(String args[])  
    {  
        FrameExample guiWindow = new FrameExample();  
    }  
}
```

## Layout Managers

- **FlowLayout**
- **BorderLayout**
- **GridLayout**
- **CardLayout**
- **GridBagLayout**
- **Box Layout**

## FlowLayout Manager

**java.awt.FlowLayout** arranges components from left-to-right and top-to-bottom, centering components horizontally with a five pixel gap between them. When a container size is changed (eg, when a window is resized), FlowLayout recompute new positions for all components subject to these constraints.

### characteristics:

- Forms the default layout for the Panel class
- Adds components from left to right
- Alignment default is centered
- Uses components' preferred sizes
- Uses the constructor to tune behavior

**FlowLayout** defines following constructors:

**FlowLayout()**

Constructs a new FlowLayout with a centered alignment and a default 5-unit horizontal and vertical gap.

**FlowLayout(int align)**

Constructs a new FlowLayout with the specified alignment and a default 5-unit horizontal and vertical gap.

`FlowLayout(int align, int hgap, int vgap)`

Creates a new flow layout manager with the indicated alignment and the indicated horizontal and vertical gaps.

**Example:**

```
import java.awt.*;

public class LayoutExample
{
    private Frame f;
    private Button b1;
    private Button b2;

    public LayoutExample()
    {
        f = new Frame("GUI example");
        b1 = new Button("Press Me");
        b2 = new Button("Don't press Me");
    }

    public void launchFrame()
    {
        f.setLayout(new FlowLayout());
        f.add(b1);
        f.add(b2);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[])
    {
        LayoutExample guiWindow = new LayoutExample();
        guiWindow.launchFrame();
    }
}
```

## BorderLayout Manager

- The BorderLayout manager is the default layout for the Frame class.
- Components are added to specific regions.
- A border layout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center.
- Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.
- The resizing behavior is as follows:
  - North, South, and Center regions adjust horizontally
  - East, West, and Center regions adjust vertically

**BorderLayout** defines following constructor:

`BorderLayout()`

Constructs a new border layout with no gaps between components.

`BorderLayout(int hgap, int vgap)`

Constructs a border layout with the specified gaps between components.

**Example:**

```
import java.awt.*;

public class BorderExample
{
    private Frame f;
    private Button bn, bs, bw, be, bc;
    public BorderExample()
    {
        f = new Frame("Border Layout");
        bn = new Button("B1");
        bs = new Button("B2");
        bw = new Button("B3");
        be = new Button("B4");
        bc = new Button("B5");
    }
    public void launchFrame()
    {
        f.add(bn, BorderLayout.NORTH);
        f.add(bs, BorderLayout.SOUTH);
        f.add(bw, BorderLayout.WEST);
        f.add(be, BorderLayout.EAST);
        f.add(bc, BorderLayout.CENTER);
        f.setSize(200,200);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        BorderExample guiWindow2 = new BorderExample();
        guiWindow2.launchFrame();
    }
}
```

## GridLayout Manager

- Components are added from left to right, and from top to bottom.
- All regions are sized equally.
- The constructor specifies the rows and columns.

**GridLayout** defines following constructors:

## GridLayout()

Creates a grid layout with a default of one column per component, in a single row.  
GridLayout(int rows, int cols)

Creates a grid layout with the specified number of rows and columns.

## GridLayout(int rows, int cols, int hgap, int vgap)

Creates a grid layout with the specified number of rows and columns.

### Example:

```
import java.awt.*;

public class GridExample
{
    private Frame f;
    private Button b1, b2, b3, b4, b5, b6;

    public GridExample()
    {
        f = new Frame("Grid Example");
        b1 = new Button("1");
        b2 = new Button("2");
        b3 = new Button("3");
        b4 = new Button("4");
        b5 = new Button("5");
        b6 = new Button("6");
    }

    public void launchFrame()
    {
        f.setLayout (new GridLayout(3,2));
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.add(b4);
        f.add(b5);
        f.add(b6);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[])
    {
        GridExample grid = new GridExample();
        grid.launchFrame();
    }
}
```

## CardLayout Manager

- A CardLayout object is a layout manager for a container. It treats each component in the container as a card.
- Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.
- The ordering of cards is determined by the container's own internal ordering of its component objects.
- CardLayout defines a set of methods that allow an application to flip through these cards sequentially, or to show a specified card.
- The CardLayout add LayoutComponent method can be used to associate a string identifier with a given card for fast random access.

**CardLayout** defines following constructors:

CardLayout()

Creates a new card layout with gaps of size zero.

CardLayout(int hgap, int vgap)

Creates a new card layout with the specified horizontal and vertical gaps.

### Example:

```
Panel cardPanel , panel1, panel2 , panel3 ;  
Button but1, but2, but3 ;  
CardLayout cards ;  
  
public Card()  
{  
    Container c = getContentPane();  
  
    cardPanel = new Panel();  
    panel1 = new Panel();  
    panel2 = new Panel();  
    panel3 = new Panel();  
    but1 = new Button( "Button 1" ) ;  
    but2 = new Button( "Button 2" ) ;  
    but3 = new Button( "Button 3" ) ;  
    but1.addActionListener( this );  
    but2.addActionListener( this );  
    but3.addActionListener( this );  
  
    panel1.add( but1 ) ;  
    panel2.add( but2 ) ;  
    panel3.add( but3 ) ;  
  
    cards = new CardLayout();
```

```
cardPanel.setLayout(cards);
cardPanel.add( panel1, "First");
cardPanel.add( panel2, "Second");
cardPanel.add( panel3, "Third");

c.add( cardPanel );
setSize( 450, 200 );
show();

}
```

## GridBagLayout Manager

- The GridBagLayout class is a flexible layout manager that aligns components vertically, horizontally or along their baseline without requiring that the components be of the same size.
- Each GridBagLayout object maintains a dynamic, rectangular grid of cells, with each component occupying one or more cells, called its display area.
- Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints. The constraints object specifies where a component's display area should be located on the grid and how the component should be positioned within its display area.
- In addition to its constraints object, the GridBagLayout also considers each component's minimum and preferred sizes in order to determine a component's size.

**GridBagLayout** defines following constructors:

GridBagLayout()

Creates a grid bag layout manager.

**Example:**

```
public class SimpleGridBag
{
    public static void main(String[] args)
    {
        Frame f = new Frame();
        Panel p = new Panel();

        p.setLayout(new GridBagLayout());
        p.add(new JButton("Java"));
        p.add(new JButton("Source"));
        p.add(new JButton("and"));
        p.add(new JButton("Support."));

        WindowListener wndCloser = new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        }
    }
}
```

```
    };
    f.addWindowListener(wndCloser);

    f.getContentPane().add(p);
    f.setSize(600, 200);
    f.show();
}
}
```

## The BoxLayout

- The BoxLayout class puts components in a single row or column
- It respects the components' requested maximum sizes and also lets you align components

## Swing

Not long after Java's original release, it became apparent that the limitations and restrictions present in the AWT were sufficiently serious that a better approach was needed.

The solution was Swing. Introduced in 1997, Swing was included as part of the Java Foundation Classes (JFC). Swing was initially available for use with Java 1.1 as a separate library. However, beginning with Java 1.2, Swing (and the rest of the JFC) was fully integrated into Java.

## AWT v/s Swing

| AWT                                                             | Swing                                                                           |
|-----------------------------------------------------------------|---------------------------------------------------------------------------------|
| ➤ Made of Partial Java Classes                                  | ➤ Made of Pure Java Classes                                                     |
| ➤ Its component are heavyweight due to depends on native peers. | ➤ Its components are lightweight because they doesn't depends on native peers . |
| ➤ AWT doesn't support a <i>pluggable look and feel</i>          | ➤ Swing supports a <i>pluggable look and feel</i>                               |

## Event Handling in Java

Event handling is fundamental to Java programming because it is integral to the creation of applets and other types of GUI-based programs. As explained here, applets are event-driven programs that use a graphical user interface to interact with the user. Furthermore, any program that uses a graphical user interface, such as a Java application written for Windows, is event driven. Thus, you cannot write these types of programs without a solid command of event handling. Events are supported by a number of packages, including **java.util**, **java.awt**, and **java.awt.event**.

Most events to which your program will respond are generated when the user interacts with a GUI-based program. They are passed to your program in a variety of ways, with the specific method dependent upon the actual event. There are several types of events, including those generated by the mouse, the keyboard, and various GUI controls, such as a push button, scroll bar, or check box.

## Delegation Event Model

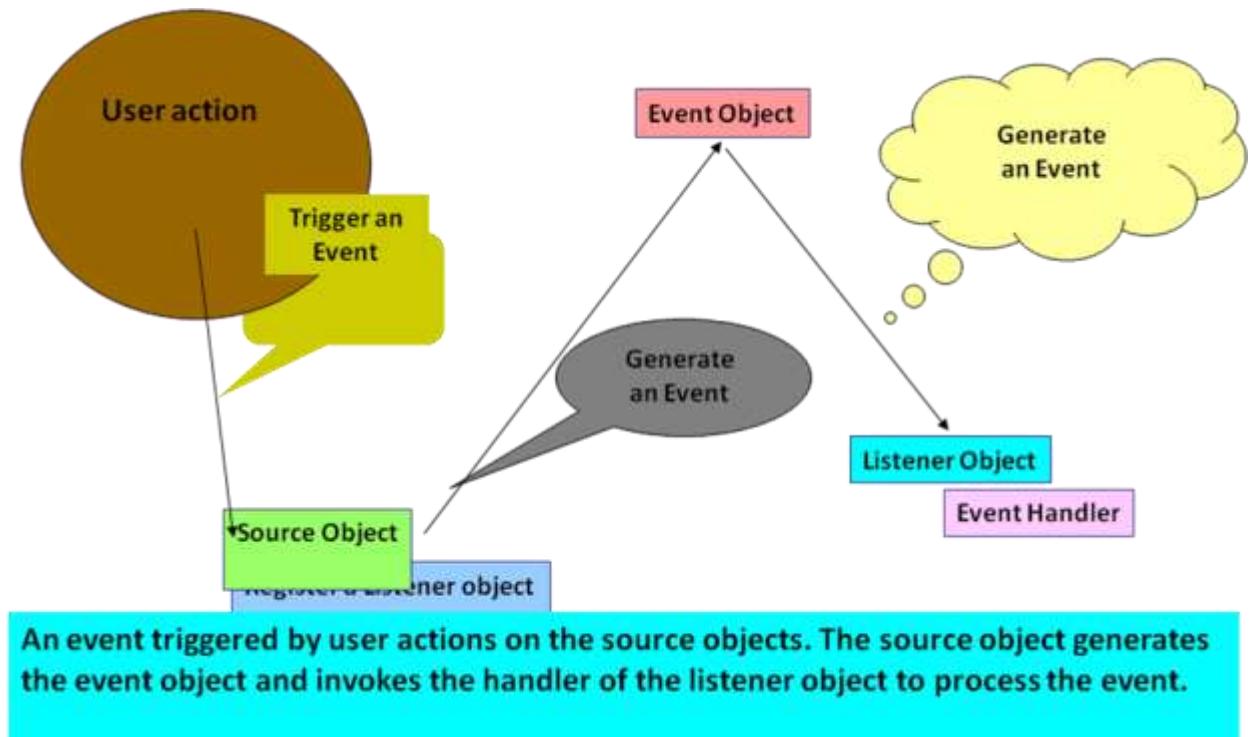
The modern approach to handling events is based on the *delegation event model*, which defines standard and consistent mechanisms to generate and process events. Its concept is quite simple: a *source* generates an event and sends it to one or more *listeners*. In this scheme, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.

The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events. A user interface element is able to “delegate” the processing of an event to a separate piece of code.

In the delegation event model, listeners must register with a source in order to receive an event notification. This provides an important benefit: notifications are sent only to listeners that want to receive them. This is a more efficient way to handle events than the design used by the old Java 1.0 approach.

Previously, an event was propagated up the containment hierarchy until it was handled by a component. This required components to receive events that they did not process, and it wasted valuable time. The delegation event model eliminates this overhead.

## Event Registration, Listening and Handling



### Event

- In the delegation model, an *event* is an object that describes a state change in a source.
- It can be generated as a consequence of a person interacting with the elements in a graphical user interface.
- Some of the activities that cause events to be generated are pressing a button, entering a character via the keyboard, selecting an item in a list, and clicking the mouse.
- Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, a counter exceeds a value, a software or hardware failure occurs, or an operation is completed.

- You are free to define events that are appropriate for your application.

## Event Sources

- A *source* is an object that generates an event. This occurs when the internal state of that object changes in some way.
- Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event.
- Each type of event has its own registration method. Here is the general form:

```
public void addTypeListener(TypeListener el)
```

- Here, *Type* is the name of the event, and *el* is a reference to the event listener.

For example,

the method that registers a keyboard event listener is called **addKeyListener( )**. The method that registers a mouse motion listener is called **addMouseMotionListener( )**.

When an event occurs, all registered listeners are notified and receive a copy of the event object. This is known as *multicasting* the event. In all cases, notifications are sent only to listeners that register to receive them.

## Event Listeners

- A *listener* is an object that is notified when an event occurs.
- It has two major requirements.
  - i. First, it must have been registered with one or more sources to receive notifications about specific types of events.
  - ii. Second, it must implement methods to receive and process these notifications.
- The methods that receive and process events are defined in a set of interfaces found in **java.awt.event**.
- For example, the **MouseMotionListener** interface defines two methods to receive notifications when the mouse is dragged or moved.
- Any object may receive and process one or both of these events if it provides an implementation of this interface.

## Event Categories

| Event Class     | Description                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| ActionEvent     | Generated when a button is pressed, a list item is double-clicked, or a menu item is selected.                                                      |
| AdjustmentEvent | Generated when a scroll bar is manipulated.                                                                                                         |
| ComponentEvent  | Generated when a component is hidden, moved, resized, or becomes visible.                                                                           |
| ContainerEvent  | Generated when a component is added to or removed from a container.                                                                                 |
| FocusEvent      | Generated when a component gains or loses keyboard focus.                                                                                           |
| InputEvent      | Abstract superclass for all component input event classes.                                                                                          |
| ItemEvent       | Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected. |
| KeyEvent        | Generated when input is received from the keyboard.                                                                                                 |

|                 |                                                                                                                                       |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| MouseEvent      | Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component. |
| MouseWheelEvent | Generated when the mouse wheel is moved.                                                                                              |
| TextEvent       | Generated when the value of a text area or text field is changed.                                                                     |
| WindowEvent     | Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.                                   |

## Event Sources

| Event Source    | Description                                                                                                                       |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Button          | Generates action events when the button is pressed.                                                                               |
| Check box       | Generates item events when the check box is selected or deselected.                                                               |
| Choice          | Generates item events when the choice is changed.                                                                                 |
| List            | Generates action events when an item is double-clicked; generates item event when an item is selected or deselected.              |
| Menu Item       | Generates action events when a menu item is selected; generates item events when a checkable menu item is selected or deselected. |
| Scroll bar      | Generates adjustment events when the scroll bar is manipulated                                                                    |
| Text Components | Generates text events when the user enters a character.                                                                           |
| Window          | Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.                 |

## Methods, Events and Listeners

| Event        | Listener            | Methods of Listeners                                                                                                                     |
|--------------|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Action       | ActionListener      | actionPerformed(ActionEvent)                                                                                                             |
| Item         | ItemListener        | itemStateChanged(ItemEvent)                                                                                                              |
| Mouse        | MouseListener       | mousePressed(MouseEvent)<br>mouseReleased(MouseEvent)<br>mouseEntered(MouseEvent)<br>mouseExited(MouseEvent)<br>mouseClicked(MouseEvent) |
| Mouse Motion | MouseMotionListener | mouseDragged(MouseEvent)<br>mouseMoved(MouseEvent)                                                                                       |
| Key          | KeyListener         | keyPressed(KeyEvent)<br>keyReleased(KeyEvent)<br>keyTyped(KeyEvent)                                                                      |
| Focus        | FocusListener       | focusGained(FocusEvent)<br>focusLost(FocusEvent)                                                                                         |
| Adjustment   | AdjustmentListener  | adjustmentValueChanged(AdjustmentEvent)                                                                                                  |
| Component    | ComponentListener   | componentMoved(ComponentEvent)<br>componentHidden(ComponentEvent)<br>componentResized(ComponentEvent)<br>componentShown(ComponentEvent)  |

|           |                   |                                                                                                                                                                                                                          |
|-----------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Window    | WindowListener    | windowClosing(WindowEvent)<br>windowOpened(WindowEvent)<br>windowIconified(WindowEvent)<br>windowDeiconified(WindowEvent)<br>windowClosed(WindowEvent)<br>windowActivated(WindowEvent)<br>windowDeactivated(WindowEvent) |
| Container | ContainerListener | componentAdded(ContainerEvent)<br>componentRemoved(ContainerEvent)                                                                                                                                                       |
| Text      | TextListener      | textValueChanged(TextEvent)                                                                                                                                                                                              |

## Event Adapters

- The listener classes that you define can extend adapter classes and override only the methods that you need.
- There are some event listeners that have multiple methods to implement. That is some of the listener interfaces contain more than one method.
- For instance, the **MouseListener** interface contains five methods such as **mouseClicked**, **mousePressed**, **mouseReleased** etc. If you want to use only one method out of these then also you will have to implement all of them. Thus, the methods which you do not want to care about can have empty bodies. To avoid such thing, we have **adapter class**.

### Example:

```

import java.awt.*;
import java.awt.event.*;

public class MouseBeeper extends MouseAdapter
{
    //here we can override only mouse clicked event .
    public void mouseClicked(MouseEvent evt)
    {
        Toolkit.getDefaultToolkit().beep();
    }
}

```

## Event handling using Inner class

- The Java programming language allows you to define a class within another class. Such a class is called *a nested class or inner class* and is illustrated here:

```

class OuterClass
{
    ...
    class NestedClass
    {
    }
}

```

```
    ...
}
}
```

- A nested class is a member of its enclosing class. Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.
- Static nested classes do not have access to other members of the enclosing class. As a member of the OuterClass, a nested class can be declared private, public, protected, or *package private*.

```
import java.awt.*;
import java.awt.event.*;
public class TestInner {
    Frame frame;
    TextField text;

    public TestInner() {
        frame = new Frame();
        text = new TextField(30);
        Label l = new Label("Click and drag the mouse");
        frame.add(l, BorderLayout.NORTH);
        frame.add(text, BorderLayout.SOUTH);
        // frame.addMouseListener();
        frame.addMouseMotionListener(new MyMouseMotionListener());
        frame.setSize(200, 200);
        frame.setVisible(true);
    }
    class MyMouseMotionListener extends MouseMotionAdapter {
        public void mouseDragged(MouseEvent e) {
            String drag = "Mouse dragging::x= " + e.getX()
                + " Mouse dragging ::y= " + e.getY();
            text.setText(drag);
        }
    }
    public static void main(String[] args) {
        TestInner t = new TestInner();
    }
}
```

(Recall that outer classes can only be declared public or *package private*.)

## Why Use Nested Classes?

There are several compelling reasons for using nested classes, among them:

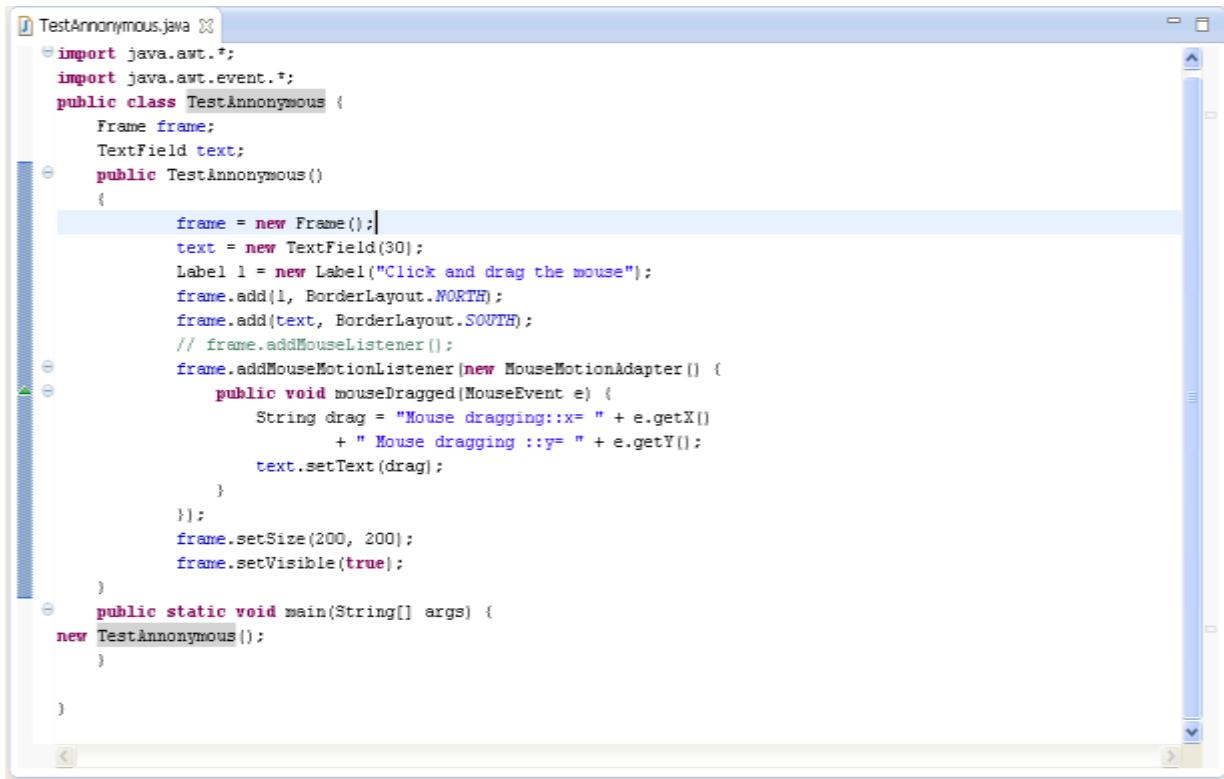
- It is a way of logically grouping classes that are only used in one place.
- It increases encapsulation.
- Nested classes can lead to more readable and maintainable code.

## Example:

```
import java.awt.*;
import java.awt.event.*;
public class TestInner
```

```
{  
    private Frame f;  
    private TextField tf; // used by inner class  
    public TestInner()  
    {  
        f = new Frame("Inner classes example");  
        tf = new TextField(30);  
    }  
    class MyMouseMotionListener extends MouseMotionAdapter  
    {  
        public void mouseDragged(MouseEvent e)  
        {  
            String s = "Mouse dragging: X = " + e.getX() + " Y  
                      = " + e.getY();  
            tf.setText(s);  
        }  
    }  
    public void launchFrame()  
    {  
        Label label = new Label("Click and drag the mouse");  
        f.add(label, BorderLayout.NORTH);  
        f.add(tf, BorderLayout.SOUTH);  
        f.addMouseListener(new  
                         MyMouseMotionListener());  
        f.setSize(300, 200);  
        f.setVisible(true);  
    }  
    public static void main(String args[])  
    {  
        TestInner obj = new TestInner();  
        obj.launchFrame();  
    }  
}
```

## Event Handling Using Anonymous Classes



The screenshot shows a Java code editor window titled "TestAnonymous.java". The code implements an anonymous inner class to handle mouse dragging events. The code is as follows:

```
import java.awt.*;
import java.awt.event.*;
public class TestAnonymous {
    Frame frame;
    TextField text;
    public TestAnonymous() {
        frame = new Frame();
        text = new TextField(30);
        Label l = new Label("Click and drag the mouse");
        frame.add(l, BorderLayout.NORTH);
        frame.add(text, BorderLayout.SOUTH);
        // frame.addMouseListener();
        frame.addMouseMotionListener(new MouseMotionAdapter() {
            public void mouseDragged(MouseEvent e) {
                String drag = "Mouse dragging::x= " + e.getX()
                    + " Mouse dragging ::y= " + e.getY();
                text.setText(drag);
            }
        });
        frame.setSize(200, 200);
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        new TestAnonymous();
    }
}
```

```
import java.awt.*;
import java.awt.event.*;

public class TestAnonymous
{
    private Frame f;
    private TextField tf;

    public TestAnonymous()
    {
        f = new Frame("Anonymous classes example");
        tf = new TextField(30);
    }
    public static void main(String args[])
    {
        TestAnonymous obj = new TestAnonymous();
        obj.launchFrame();
    }
    public void launchFrame()
    {
        Label label = new Label("Click and drag the mouse");
        f.add(label, BorderLayout.NORTH);
        f.add(tf, BorderLayout.SOUTH);
    }
}
```

```
f.addMouseListener(new MouseMotionAdapter() {  
    public void mouseDragged(MouseEvent e)  
    {  
        String s = "Mouse dragging: X = "+  
            e.getX()+ " Y = " + e.getY();  
        tf.setText(s);  
    }  
});  
f.setSize(300, 200);  
f.setVisible(true);  
}  
}
```

## Database Programming using JDBC

### What is JDBC

Java Database Connectivity in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

JDBC has been developed under the Java Community Process that allows multiple implementations to exist and be used by the

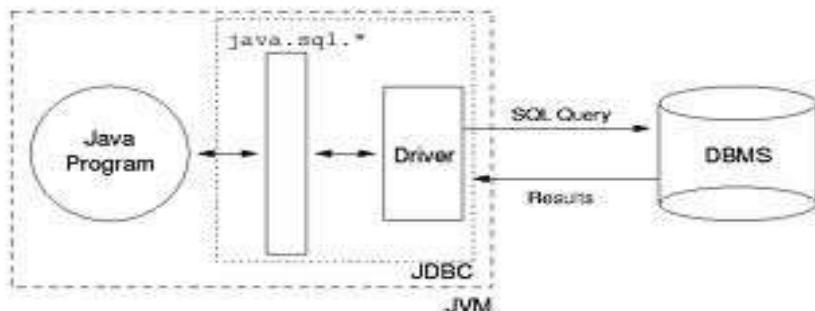
same application. JDBC provides methods for querying and updating the data in Relational Database Management system such as SQL, Oracle etc.

The Java application programming interface provides a mechanism for dynamically loading the correct Java packages and drivers and registering them with the JDBC Driver Manager that is used as a connection factory for creating JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE. Driver Manager is the backbone of the jdbc architecture.

Generally all Relational Database Management System supports SQL and we all know that Java is platform independent, so JDBC makes it possible to write a single database application that can run on different platforms and interact with different Database Management Systems.

Java Database Connectivity is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language.

### JDBC Architecture



In the two-tier model, a Java application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source being accessed. A user's commands are delivered to the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration, with the user's machine as the client, and the machine housing the data source as the server. The network can be an intranet, which, for example, connects employees within a corporation, or it can be the Internet.

In the three-tier model, commands are sent to a "middle tier" of services, which then sends the commands to the data source. The data source processes the commands and sends the results back to the middle tier, which then sends them to the user. MIS directors find the three-tier model very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. Another advantage is that it simplifies the deployment of applications. Finally, in many cases, the three-tier architecture can provide performance advantages.

Until recently, the middle tier has often been written in languages such as C or C++, which offer fast performance. However, with the introduction of optimizing compilers that translate Java bytecode into efficient machine-specific code and technologies such as Enterprise JavaBeans™, the Java platform is fast becoming the standard platform for middle-tier development. This is a big plus, making it possible to take advantage of Java's robustness, multithreading, and security features.

With enterprises increasingly using the Java programming language for writing server code, the JDBC API is being used more and more in the middle tier of a three-tier architecture. Some of the features that make JDBC a server technology are its support for connection pooling, distributed transactions, and disconnected rowsets. The JDBC API is also what allows access to a data source from a Java middle tier.

## JDBC Driver and Driver Types

JDBC Driver can be broadly categorized into 4 categories--

### **JDBC-ODBC BRIDGE DRIVER(TYPE 1)**

Features

- 1.Convert the query of JDBC Driver into the ODBC query, which in return pass the data.
- 2.JDBC-ODBC is native code not written in java.
- 3.The connection occurs as follows -- Client -> JDBC Driver -> ODBC Driver -> Database .

Pros-->

A type-1 driver is easy to install and handle

Cons-->

- 1.Extra channels in between database and application made performance overhead.
- 2.Needs to be installed on client machine.
- 3.Not suitable for applet , due to the installation at clients end.

### **Native-API Type-2 Driver**

Features

The type 2 driver need libraries installed at client site. For example, we need "mysqlconnector.jar" to be copied in library of java kit. It is not written in java entirely because the non-java interface have the direct access to database.

Pros--

1. Type 2 driver has additional functionality and better performance than Type 1.
2. Has faster performance than type 1,3 &4,since it has separate code for native APIS.

Cons--

1. library needs to be installed on the client machine .
2. Due to the client side software demand, it can't be used for web based application.
3. platform dependent.
4. It doesn't support "Applets".

## Network-Protocol Type 3 driver

Features

- 1.It has a 3-tier architecture.
- 2.It can interact with multiple database of different environment.
- 3.The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
- 4.The connection occurs as follows--Client -> JDBC Driver -> Middleware-Net Server -> Any Database.

Pros--

- 1.The client driver to middleware communication is database independent.
- 2.Can be used in internet since there is no client side software needed.

Cons--

- 1.Needs specific coding for different database at middleware.
- 2.Due to extra layer in middle can result in time-delay.

## Native Protocol Type 4 Driver

Features

Also known as Direct to Database Pure Java Driver .It is entirely in java. It interacts directly with database generally through socket connection. It is platform independent. It directly converts driver calls into database protocol calls.

Pros--

- 1.Improved performance because no intermediate translator like JDBC or middleware server.
- 2.All the connection is managed by JVM, so debugging is easier.

## JDBC Data Types

| JDBC Type     | Java Type |
|---------------|-----------|
| BIT           | boolean   |
| TINYINT       | byte      |
| SMALLINT      | short     |
| INTEGER       | int       |
| BIGINT        | long      |
| REAL          | float     |
| FLOAT         | double    |
| DOUBLE        |           |
| BINARY        | byte[]    |
| VARBINARY     |           |
| LONGVARBINARY |           |
| CHAR          |           |
| VARCHAR       | String    |
| LONGVARCHAR   |           |

| JDBC Type   | Java Type                  |
|-------------|----------------------------|
| NUMERIC     | BigDecimal                 |
| DECIMAL     |                            |
| DATE        | java.sql.Date              |
| TIME        | java.sql.Timestamp         |
| TIMESTAMP   |                            |
| CLOB        | Clob*                      |
| BLOB        | Blob*                      |
| ARRAY       | Array*                     |
| DISTINCT    | mapping of underlying type |
| STRUCT      | Struct*                    |
| REF         | Ref*                       |
| JAVA_OBJECT | underlying Java class      |

\*SQL3 data type supported in JDBC 2.0

## Seven basic steps in using JDBC

- Loading Driver
- Establishing Connection
- Executing Statements
- Getting Results
- Closing Database Connection

Before explaining you the JDBC Steps for making connection to the database and retrieving the employee from the tables, we will provide you the structure of the database and sample data.

Here is the sql script to create table and populate the table with data:

```
-- Table structure for table `employee`  
CREATE TABLE `employee` (  
`employee_name` varchar(50) NOT NULL,  
PRIMARY KEY (`employee_name`)  
);  
INSERT INTO `employee`(`employee_name`) VALUES  
('Deepak Kumar'),  
('Harish Joshi'),  
('Rinku roy'),  
('Vinod Kumar');
```

Explanation of JDBC Steps:

- Loading Driver

Loading Database driver is very first step towards making JDBC connectivity with the database. It is necessary to load the JDBC drivers before attempting to connect to the database. The JDBC drivers automatically register themselves with the JDBC system when loaded. Here is the code for loading the JDBC driver:

```
Class.forName(driver).newInstance();
```

- Establishing Connection

In the above step we have loaded the database driver to be used. Now its time to make the connection with the database server. In the Establishing Connection step we will logon to the database with user name and password. Following code we have used to make the connection with the database:

```
con = DriverManager.getConnection(url+db, user, pass);
```

- Executing Statements

In the previous step we established the connection with the database, now its time to execute query against database. You can run any type of query against database to perform database operations. In this example we will select all the rows from employee table. Here is the code that actually execute the statements against database:

```
ResultSet res = st.executeQuery( "SELECT * FROM employee" );
```

- Getting Results

In this step we receives the result of execute statement. In this case we will fetch the employees records from the recordset object and show on the console. Here is the code:

```
while (res.next()) {
```

```
String employeeName = res.getInt( " employee_name " );
System.out.println( employeeName );
}
```

- **Closing Database Connection**

Finally it is necessary to disconnect from the database and release resources being used. If you don't close the connection then in the production environment your application will fail due to hanging database connections. Here is the code for disconnecting the application from database:

```
con.close();
```

```
import java.sql.*;
public class RetriveAllEmployees{
    public static void main(String[] args) {
        System.out.println("Getting All Rows from employee table!");
        Connection con = null;
        String url = "jdbc:mysql://localhost:3306/";
        String db = "jdbc";
        String driver = "com.mysql.jdbc.Driver";
        String user = "root";
        String pass = "root";
        try{
            Class.forName(driver);
            con = DriverManager.getConnection(url+db, user, pass);
            Statement st = con.createStatement();
            ResultSet res = st.executeQuery("SELECT * FROM employee");
            System.out.println("Employee Name: " );
            while (res.next()) {
                String employeeName = res.getString("employee_name");
                System.out.println(employeeName );
            }
            con.close();
        }
        catch (ClassNotFoundException e){
            System.err.println("Could not load JDBC driver");
            System.out.println("Exception: " + e);
            e.printStackTrace();
        }
        catch(SQLException ex){
            System.err.println("SQLException information");
            while(ex!=null) {
                System.err.println ("Error msg: " + ex.getMessage());
                System.err.println ("SQLSTATE: " + ex.getSQLState());
                System.err.println ("Error code: " + ex.getErrorCode());
                ex.printStackTrace();
                ex = ex.getNextException();
            }
        }
    }
}
```

In this section you learnt about the JDBC Steps necessary for performing database operations.

## Retrieving data from a ResultSet

JDBC ResultSet is an interface of java.sql package. It is a table of data representing a database query result, which is obtained by executing the execute method of statement. A ResultSet object points the cursor to the first row of the data. Initially the cursor point before the fist row of the data. to move the cursor next() method is called. If there is no any row present in the result then it returns the false value.

The default ResultSet object is not updateable therefore the cursor moves only forward from the first row to the last row only once. It is possible to make ResultSet object that is updateable. to make such type of ResultSet object you need to write the following code.

```
Statement stmt = con.createStatement(  
ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_UPDATABLE);  
ResultSet rs = stmt.executeQuery("SELECT * FROM Student");
```

The ResultSet object also provides a getter method that gets the value from the ResultSet object table. You need to specify only the Column name or index no of the table. For example-  
resultSet.getString("ColumnName");, resultSet.getInt("ColumnName");,

You can update the database table from the resultSet object as -

```
Statement statement = connection.createStatement(  
ResultSet.TYPE_SCROLL_INSENSITIVE,  
ResultSet.CONCUR_UPDATABLE);  
  
resultSet.first();  
// Setting the updating String  
resultSet.updateString("Name", "vnay");  
// Updating the first row  
resultSet.updateRow();
```

An Example given below illustrate the above explanations, At first create database named student and then create a table student in the student database as

```
CREATE TABLE student (  
RollNo int(9) PRIMARY KEY NOT NULL,  
Name tinytext NOT NULL,  
Course varchar(25) NOT NULL,  
Address text  
);
```

Then insert the value into it as

```
INSERT INTO student VALUES(1, 'Ram', 'B.Tech', 'Delhi') ;  
INSERT INTO student VALUES(2, 'Syam', 'M.Tech', 'Mumbai') ;
```

```
JDBCResultSetExample.java  
package com.topsint;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
public class JDBCResultSetExample {  
    Connection connection = null;
```

```
public JDBCResultSetExample() {
    try {
        // Loading the driver
        Class.forName("com.mysql.jdbc.Driver");
    } catch (ClassNotFoundException e) {
        System.out.println(e.toString());
    }
}
public Connection createConnection() {
    Connection con = null;
    if (connection != null) {
        System.out.println("Cant create a connection");
    } else {
        try {
            // Creating a Connection to the Student database
            con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/student", "root",
                "root");
            System.out.println("Connection created Successfully");
        } catch (SQLException e) {
            System.out.println(e.toString());
        }
    }
    return con;
}
public static void main(String[] args) throws SQLException {
    JDBCResultSetExample resultSetExample = new JDBCResultSetExample();
    Connection connection = resultSetExample.createConnection();
    try {
        // creating a statement object
        Statement statement = connection.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_UPDATABLE);
        String query = "SELECT * FROM student";
        // executing a query string and storing it into the resultSet object
        ResultSet resultSet = statement.executeQuery(query);
        System.out.println("Before Updating.....\n");
        while (resultSet.next()) {
            // Printing results to the console
            System.out.println("Roll No- " + resultSet.getInt("RollNo")
                + ", Name- " + resultSet.getString("Name")
                + ", Course- " + resultSet.getString("Course")
                + ", Address- " + resultSet.getString("Address"));
        }
        // setting the row to we have to update
        resultSet.first();
        // Setting the updating String
        resultSet.updateString("Name", "vnay");
    }
}
```

```
// Updating the first row
resultSet.updateRow();
System.out.println("\n\n After Updatig.....\n");
while (resultSet.next()) {
    // Printing results to the console
    System.out.println("Roll No- " + resultSet.getInt("RollNo")
        + ", Name- " + resultSet.getString("Name")
        + ", Course- " + resultSet.getString("Course")
        + ", Address- " + resultSet.getString("Address"));
}
} catch (Exception e) {
    System.out.println(e.toString());
} finally {
    connection.close();
}
```

When you run this application it will display message as shown below:

Connection created Successfully

Before Updating.....

Roll No- 1, Name- Ram, Course- B.Tech, Address- Delhi

Roll No- 2, Name- Syam, Course- M.Tech, Address- Mumbai

After Updating.....

Roll No- 2, Name- Syam, Course- BCA, Address- Mumbai

## Web Technologies In Java

### Designing

#### Html

#### HTML Tag List

- HTML offers authors several mechanisms for specifying lists of information.
- All lists must contain one or more list elements. Lists may contain:
  - Unordered information.
  - Ordered information.
  - Definitions

#### Ordered List

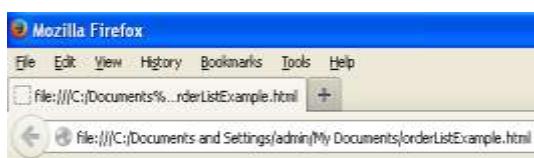
| List type  | Tag  | Attribute | value                 | Description                                                   | Example         |
|------------|------|-----------|-----------------------|---------------------------------------------------------------|-----------------|
| Order List | <ol> | reversed  |                       | Specifies that the list order should be descending (9,8,7...) | <ol reversed>   |
|            |      | start     | number                | Specifies the start value of an ordered list                  | <ol start="10"> |
|            |      | type      | 1<br>A<br>a<br>I<br>i | Specifies the kind of marker to use in the list               | <ol type="A">   |

| List Type      | Tag  | Attribute | value                    | description                                          | example                        |
|----------------|------|-----------|--------------------------|------------------------------------------------------|--------------------------------|
| Unordered List | <ul> | type      | disc<br>square<br>circle | Use to Specifies the kind of bullet use in the list. | <ul type="disc circle square"> |

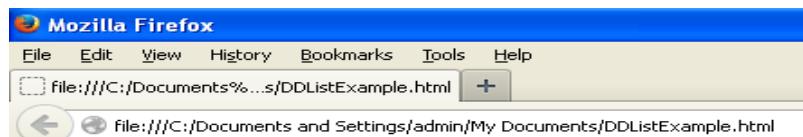
```

<ol>
<li>Ahmedabad</li>
<li>Baroda</li>
<li>Rajkot</li>
<li>Surat</li>
</ol>
<ol start="10">
<li>Ahmedabad</li>
<li>Baroda</li>
<li>Rajkot</li>
<li>Surat</li>
</ol>
<ol type="A">
<li>Ahmedabad</li>
<li>Baroda</li>
<li>Rajkot</li>
<li>Surat</li>
</ol>

```



1. Ahmedabad  
 2. Baroda  
 3. Rajkot  
 4. Surat  
 10. Ahmedabad  
 11. Baroda  
 12. Rajkot  
 13. Surat  
 A. Ahmedabad  
 B. Baroda  
 C. Rajkot  
 D. Surat



Ahmedabad  
 Famous for IIM  
 Baroda  
 Famous for MS

```

<ul>
<li>Ahmedabad</li>
<li>Baroda</li>
<li>Rajkot</li>
<li>Surat</li>
</ul>
<ul type="square">
<li>Ahmedabad</li>
<li>Baroda</li>
<li>Rajkot</li>
<li>Surat</li>
</ul>
<ul type="circle">
<li>Ahmedabad</li>

```

| List Type       | Tag  | description                | example                                                                                      |
|-----------------|------|----------------------------|----------------------------------------------------------------------------------------------|
| Definition List | <dl> | Defines a description list | <dl> <dt>Ahmedabad</dt> <dd>Famous for IIM</dd> <dt>Baroda</dt> <dd>Famous for MS</dd> </dl> |
|                 | <dt> | Defines terms/names.       |                                                                                              |
|                 | <dd> | describes each term/name   |                                                                                              |

```

<dl>
<dt>Ahmedabad</dt>
<dd>Famous for IIM</dd>
<dt>Baroda</dt>
<dd>Famous for MS</dd>

```

## HTML hyperlink

| Tag | Description                                                                 | attribute                                    |                                                 | example                                                            |
|-----|-----------------------------------------------------------------------------|----------------------------------------------|-------------------------------------------------|--------------------------------------------------------------------|
| <a> | This tag defines hyperlink, which is used to link from one page to another. | href                                         | Specifies the URL of the page the link goes to. | <a href="http://www.google.com" >Visit Google!</a>                 |
|     |                                                                             | target values (_blank, _parent, _self, _top) | Specifies where to open the linked document     | <a href="http://www.google.com" target="_parent">Visit google!</a> |

## HTML Image tag

- To add image in html page we can use <img> tag.

| attribute | Description                                                                                              | value                                    | Example                                                         |
|-----------|----------------------------------------------------------------------------------------------------------|------------------------------------------|-----------------------------------------------------------------|
| Align     | places the graphic image at a specified position, in relation either to the page margins or to the text. | top<br>bottom<br>middle<br>left<br>right |                            |
| alt       | Specifies an alternate text for an image                                                                 | text                                     |                              |
| height    | Specifies the height of an image                                                                         | pixels                                   |  |
| width     | Specifies the width of an image                                                                          | pixels                                   |  |
| src       | Specifies the URL of an image                                                                            | URL                                      |                                          |

```

```

## HTML Table tag

- The HTML <table> element inserts a table in the document.
- This element is the main container of a table, but many other elements are needed to define a table correctly.



| attribute   | value                              | example                                                     |
|-------------|------------------------------------|-------------------------------------------------------------|
| align       | left<br>center<br>right            | <table border="1" align="right">                            |
| bgcolor     | rgb(x,x,x)<br>#xxxxxx<br>colorname | <table border="1" bgcolor="#00FF00">                        |
| border      | ""                                 | <table border="1" border="1" bgcolor="#00FF00">             |
| cellpadding | pixels                             | <table border="1" cellpadding="10" bgcolor="#00FF00">       |
| cellspacing | pixels                             | <table border="1" cellspacing="10" bgcolor="#00FF00">       |
| width       | pixels,<br>%                       | <table border="1" width="400" border="1" bgcolor="#00FF00"> |

| <b>Tag</b> | <b>Attribute</b>                                                                         | <b>Values</b>                         | <b>description</b>                                        |
|------------|------------------------------------------------------------------------------------------|---------------------------------------|-----------------------------------------------------------|
| <th>       | Defines a header cell in a table                                                         |                                       |                                                           |
|            | <td>Left, right, center, justify, char</td> <td>Aligns the content in a header cell</td> | Left, right, center, justify, char    | Aligns the content in a header cell                       |
|            | bgcolor                                                                                  | <i>rgb(x,x,x), #xxxxxx, colorname</i> | Specifies the background color of a header cell           |
|            | colspan                                                                                  | <i>number</i>                         | Specifies the number of columns a header cell should span |
|            | Height                                                                                   | <i>Pixels, %</i>                      | Sets the height of a header cell                          |
|            | rowspan                                                                                  | <i>number</i>                         | Specifies the number of rows a header cell should span    |
|            | Width                                                                                    | <i>pixels %</i>                       | Specifies the width of a header cell                      |

```

<h4>Table headers:</h4>
<table border="1">
<tr><th bgcolor="green" width=100>Name</th>
<th bgcolor="green" width=100>Telephone</th></tr>
<tr><td>Tops Technologies</td>
<td>123456789</td>
</tr>
</table>
<h4>Vertical headers:</h4>
<table border="1">
<tr>
<th height=30>First Name:</th>
<td>Tops Technology</td>
</tr>
<tr>

```

#### Table Row tag

| <b>Tag</b> | <b>Attribute</b>                                                                       | <b>Value</b>                          | <b>description</b>                               |
|------------|----------------------------------------------------------------------------------------|---------------------------------------|--------------------------------------------------|
| <tr>       | <td>Right, left, center, justify, char</td> <td>Aligns the content in a table row</td> | Right, left, center, justify, char    | Aligns the content in a table row                |
|            | bgcolor                                                                                | <i>rgb(x,x,x), #xxxxxx, colorname</i> | Specifies a background color for a table row     |
|            | char                                                                                   | <i>character</i>                      | Aligns the content in a table row to a character |
|            | valign                                                                                 | Top, middle, bottom, baseline         | Vertical aligns the content in a table row       |

#### Table data tag

| <b>Tag</b>           | <b>Attribute</b>                                                                  | <b>Value</b>                       | <b>description</b>           |
|----------------------|-----------------------------------------------------------------------------------|------------------------------------|------------------------------|
| <td> (contains data) | <td>Left, right, center, justify, char</td> <td>Aligns the content in a cell</td> | Left, right, center, justify, char | Aligns the content in a cell |

# TOPS Technologies

|  |         |                                       |                                          |
|--|---------|---------------------------------------|------------------------------------------|
|  | bgcolor | <i>rgb(x,x,x), #xxxxxx, colorname</i> | Specifies the background color of a cell |
|--|---------|---------------------------------------|------------------------------------------|

Cell that spans two columns:

|                   |                       |
|-------------------|-----------------------|
| Name              | Telephone             |
| Tops Technologies | 123456789   987654321 |

Cell that spans two rows:

|             |                       |
|-------------|-----------------------|
| First Name: | Tops Technologies     |
| Telephone:  | 123456789   987654321 |

|  |         |                                      |                                                    |
|--|---------|--------------------------------------|----------------------------------------------------|
|  | colspan | <i>Number</i>                        | Specifies the number of columns a cell should span |
|  | height  | <i>Pixels, %</i>                     | Sets the height of a cell                          |
|  | rowspan | <i>Number</i>                        | Sets the number of rows a cell should span         |
|  | valign  | <i>Top, middle, bottom, baseline</i> | Vertical aligns the content in a cell              |
|  | width   | <i>Pixels, %</i>                     | Specifies the width of a cell                      |

```
<h4>Cell that spans two columns:</h4>
<table border="1">
<tr>
<th>Name</th><th colspan="2">Telephone</th>
</tr>
<tr>
<td>Tops Technologies</td><td>123456789</td><td>987654321</td>
</tr>
</table>
<h4>Cell that spans two rows:</h4>
<table border="1">
<tr>
<th>First Name:</th><td>Tops Technologies</td>
</tr>
<tr>
<th rowspan="2">Telephone:</th><td>123456789</td>
</tr>
<tr><td>987654321</td></tr>
</table>
```

## HTML Form tag

- HTML Forms are used to select different kinds of user input.

Attribute	Value	Description
action	URL	Specifies where to send the form-data when a form is submitted
enctype	application/x-www-form-urlencoded, multipart/form-data , text/plain	Specifies how the form-data should be encoded when submitting it to the server (only for method="post")
method	Get , post	Specifies the HTTP method to use when sending form-data
name	Text	Specifies the name of a form
target	_blank, _self, _parent, _top	Specifies where to display the response that is received after submitting the form

Tags	Description
<input>	The <input> tag specifies an input field where the user can enter data.
<textarea>	The <textarea> tag defines a multi-line text input control.
<button>	The <button> tag defines a clickable button.
<select>	The <select> element is used to create a drop-down list.
<option>	The <option> tags inside the <select> element define the available options in the list.
<optgroup>	The <optgroup> is used to group related options in a drop-down list.
<fieldset>	The <fieldset> tag is used to group related elements in a form.
<label>	The <label> tag defines a label for an <input> element.

## Form input tag

Tag <input>	Attribute	Description	values
	type	Specifies the type of control.	Text, password, checkbox, radio, submit, reset, file, Hidden, image, button
	Name	Assigns a name to the input control.	
	value	Specifies the initial value for the control. Note: If type="checkbox" or type="radio" this attribute is required.	
	Size	Specifies the width of the control. If type="text" or type="password" this refers to the width in characters. Otherwise it's in pixels.	
	maxlength	Specifies the maximum number of characters that the user can input. This can be bigger than the value indicated in the size attribute.	
	checked	If type="radio" or type="checkbox" it will already be selected when the page loads.	

	Src	If type="image", this attribute specifies the location of the image.	
--	-----	----------------------------------------------------------------------	--

```

<form action="example.html">
<fieldset>
    <legend>Personal information</legend>
    <p>Name:<input type="text" name="pname" /></p>
    <p>Address: <input type="text" name="paddress" /></p>
    <p>Phone:<input type="text" name="pphone" /></p>
    <p><label>Gender <input type="radio" name="gender" value="male" />Male</label><label><input type="radio" name="gender" value="female" />Female</label><br /></p>
</fieldset>
<fieldset>
    <legend>Work information</legend>
    <p>Address: <input type="text" name="waddress" /></p>
    <p>Phone: <input type="text" name="wphone" /></p>
    <p>Select City <select name="city">
        <option value="ahm">Ahmedabad</option>
        <option value="baroda">Baroda</option>
        <option value="surat">Surat</option>
        <option value="rajkot">Rajkot</option></select>      </p>
    <p><label><input name="OK" type="submit" id="OK" value="Submit" />
        </label><label><input type="reset" name="Submit2" value="Reset" /></label></p>
</fieldset></form>

```

The screenshot shows a web browser window with the following details:

- Page Title:** Personal information
- Form Fields (Personal Information):**
  - Name: [Text Input]
  - Address: [Text Input]
  - Phone: [Text Input]
  - Gender:  Male  Female
- Form Fields (Work Information):**
  - Address: [Text Input]
  - Phone: [Text Input]
  - Select City: Ahmedabad (dropdown menu)
- Buttons:** Submit, Reset

## CSS (Cascading Style Sheet)

- **CSS Introduction**
  - CSS stands for Cascading Style Sheets
  - Styles define how to display HTML elements
  - Styles were added to HTML 4.0 to solve a problem
  - External Style Sheets can save a lot of work
  - External Style Sheets are stored in CSS files.
- **CSS Syntax**
  - With html : <body bgcolor="#FF0000">
  - With CSS : body {background-color: #FF0000;}
- **Types of CSS**
  - Inline Styles
  - Internal Styles
  - External Styles

### Inline style

- One way to apply CSS to HTML is by using the HTML attribute style. Make html with the red background color.

```
<html><head><title>Example</title>
</head>
<body style="background-color: #FF0000;">
```

### Internal (the tag style)

- Another way is to include the CSS codes using the HTML tag <style>.

```
<html><head><title>Example</title>
<style type="text/css">
body {background-color: #FF0000;}
</style>
```

### External (link to a style sheet)

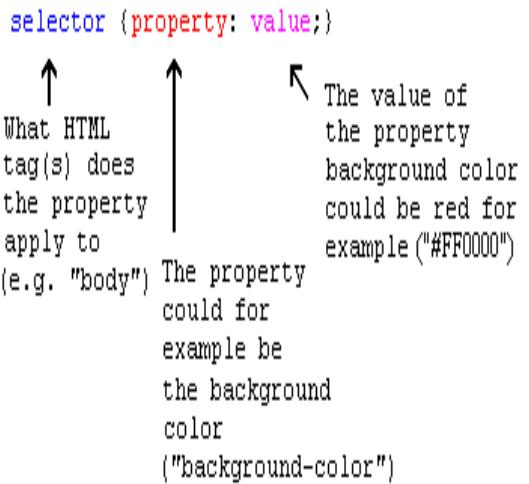
- The recommended method is to link to a so-called external style sheet.
- An external style sheet is simply a text file with the extension .css.
- Like any other file, you can place the style sheet on your web server or hard disk.

```
<html><head>
<title>My document</title>
<link rel="stylesheet" type="text/css" href="style/style.css" />
```

## Pseudo Classes

### Styles for Special Cases :

- Although primarily intended to add styles to particular elements created using HTML tags, there are several cases where we can use CSS to style content on the page that is not specifically set off by HTML tags or to create a dynamic style in reaction to something that your Web site visitor is doing on the screen.
- These are known as pseudo-elements and pseudo-classes:



- **Link pseudo-classes:** Used to style hypertext links. Although primarily associated with color, you can actually use any CSS property to set off links and provide user feedback during interaction.
- **Dynamic pseudo-classes:** Used to style any element on the screen depending on how the user is interacting with it.
- **Pseudo-elements:** Used to style the first letter or first line in a block of text.
- **Link States:** All hypertext links have four “states” that can be styled in reaction to a user action.

## Headings and paddings

Links are typically the blue, underlined, hypertext links that are used to move between web pages and web sites over the web. A link's state can be:

### Link colors and link states

a link state when there has been no action.	a:link {color:grey;} /* unvisited link */
A hover state when the mouse cursor is over it.	a:visited {color:red;} /* visited link */
An active state when the user clicks it.	a:hover {color:yellow;} /* mouse over link */
A visited state when the user returns after having visited the linked-to .	a:active {color:blue;} /* selected link */

## Margins and padding

Margin and padding are the two most commonly used properties for spacing-out elements.

- A margin is the space outside something, whereas padding is the space inside something.

## Margin

```
body { margin-top: 100px; margin-right: 40px; margin-bottom: 10px; margin-left: 70px; }
```

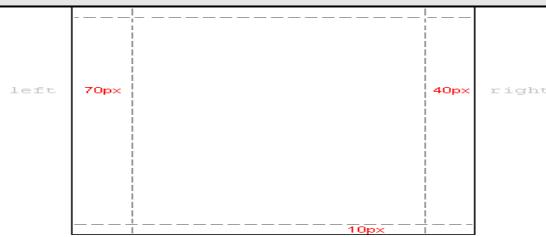
OR

```
body { margin: 100px 40px 10px 70px; }
```

Property	Description
margin	A shorthand property for setting the margin properties in one declaration
margin-bottom	Sets the bottom margin of an element
margin-left	Sets the left margin of an element
margin-right	Sets the right margin of an element
margin-top	Sets the top margin of an element

## Padding

```
h1 {background: yellow;padding: 20px 20px 20px 80px; }
h2 { background: orange;padding-left: 120px; }
```



<b>Property</b>	<b>Description</b>
padding	A shorthand property for setting all the padding properties in one declaration
padding-bottom	Sets the bottom padding of an element
padding-left	Sets the left padding of an element
padding-right	Sets the right padding of an element
padding-top	Sets the top padding of an element

## CSS background

- CSS background properties are used to define the background effects of an element.

<b>Property</b>	<b>Description</b>	<b>Example</b>
background-color	Sets the background color of an element	background-color:yellow;
background-image	Sets the background image for an element	background-image:url('paper.gif');
background-repeat	Sets how a background image will be repeated.(By default, the background-image property repeats an image both horizontally and vertically.)	background-repeat:repeat-y;(repeat only vertically)
background-position	Sets the starting position of a background image	background-position:center;
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page	background-attachment:fixed;
background	Set all the background properties in one declaration Syntax : background: color position size repeat origin clip attachment image;	background: #00ff00 url('smiley.gif') no-repeat fixed center;

## Background Color property

<b>Property</b>	<b>Value</b>	<b>Description</b>	<b>Example</b>
background-color	Color	Specifies the background color.	background-color:red; background-color:rgb(255,130,255)
	transparent	Specifies that the background color should be transparent. This is default	background-color:transparent;
	Inherit	Specifies that the background color should be inherited from the parent element	background-color:inherited;

## Background position property

<b>Property</b>	<b>value</b>	<b>Description</b>	<b>Example</b>
background-position	left top left center left bottom right top	If you only specify one keyword, the other value will be "center"	background-position: top;

	right center right bottom center top center center center bottom		
	<i>x%</i> <i>y%</i>	The first value is the horizontal position and the second value is the vertical.	background-position: 25% 75%;
	<i>xpos</i> <i>ypos</i>	The first value is the horizontal position and the second value is the vertical.	background-position: right 20px bottom 20px;
	<i>inherit</i>	Specifies that the setting of the background-position property should be inherited from the parent element	

#### Background repeat property

Property	value	Description	Example
background-repeat	repeat	The background image will be repeated both vertically and horizontally. This is default	background-repeat: repeat
	repeat-x	The background image will be repeated only horizontally	background-repeat: repeat-x
	repeat-y	The background image will be repeated only vertically	background-repeat: repeat-y
	no-repeat	The background-image will not be repeated	background-repeat: no-repeat
	inherit		background-repeat: inherit

#### Background image property

Property	value	Description	Example
background-image	url('URL')	The URL to the image. To specify more than one image, separate the URLs with a comma	background-image: url(http://www.example.com/images/bck.png)
	none	No background image will be displayed. This is default	background-image: none
	inherit	Specifies that the background image should be inherited from the parent element	background-image: inherit

#### Background attachment property

Property	value	Description	Example
background-attachment	scroll	The background scrolls along with the element. This is default	background-attachment: scroll;
	fixed	The background is fixed with regard to the viewport	background-attachment: fixed;
	local	The background scrolls along with the element's contents	background-attachment: local;

## CSS using ID & class

- Setting a style for a HTML element, CSS allows you to specify your own selectors called "id" and "class".

Id	class
Use id to identify elements that there will only be a single instance of on a page.	Use class to group elements that all behave a certain way.
an ID selector is a name preceded by a <b>hash character</b> ("#").	a class selector is a name preceded by a <b>full stop</b> (".")
#top { background-color: #ccc; padding: 20px }	.intro { color: red; font-weight: bold; }

```
<div id="top">
<h1>Chocolate curry</h1>
<p class="intro">This is my recipe for making curry purely with chocolate</p>
```

## Client Side Scripting in Java

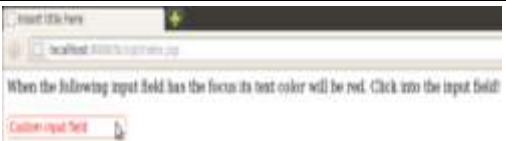
JavaScript

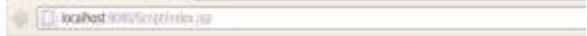
### Event

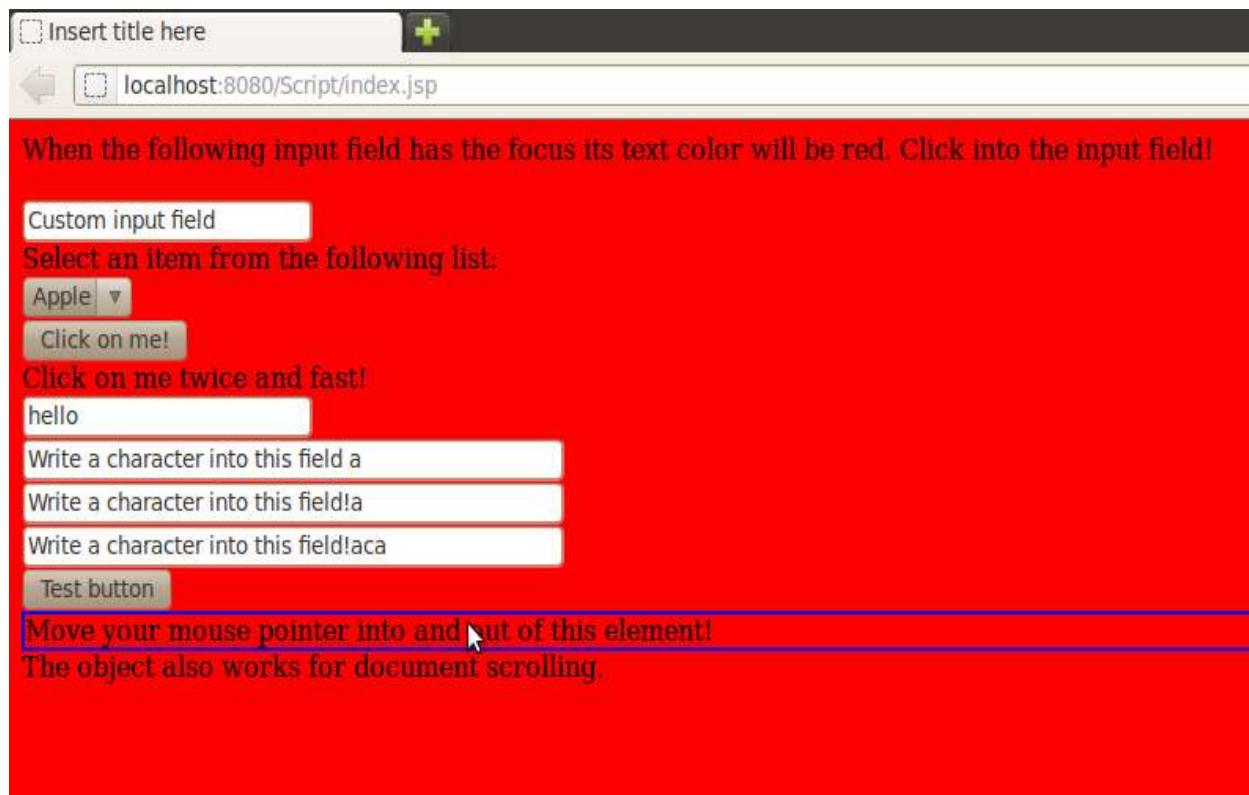
Events	
onBlur	User has left the focus of the object. For example, they clicked away from a text field that was previously selected.
onChange	User has changed the object, then attempts to leave that field.
onClick	User clicked on the object.
onDblClick	User clicked twice on the object.
onFocus	User brought the focus to the object.
onkeydown	A key was pressed over an element.
onkeyup	A key was released over an element.
onkeypress	A key was pressed over an element then released
onLoad	The object has loaded.

## Mouse Events

<i>Mouse Event:</i>	
<i>onMousedown</i>	<i>The cursor moved over the object and mouse/pointing device was pressed down.</i>
<i>onMouseup</i>	<i>The mouse/pointing device was released after being pressed down.</i>
<i>onMouseover</i>	<i>The cursor moved over the object.</i>
<i>onMousemove</i>	<i>The cursor moved while hovering over an object.</i>
<i>onMouseout</i>	<i>The cursor moved off the object.</i>

<b>onBlur</b>	<pre>function OnFocusInput(input)     {input.style.color = "red";} function OnBlurInput(input) {     input.style.color = "";</pre>	 <p>When the following input field has the focus its text color will be red. Click into the input field!</p> <p>Custom input field</p>
<b>onChange</b>	<pre>function OnSelectionChange (select) {     var selectedOption =         select.options[select.selectedIndex];     alert ("The selected option is "         +selectedOption.value); } &lt;select onchange="OnSelectionChange (this)"&gt; &lt;option value="Apple" /&gt;Apple &lt;option value="Pear" /&gt;Pear &lt;option value="Peach" /&gt;Peach &lt;/select&gt;</pre>	 <p>When the following input field has the focus its text color will be red. Click into the input field!</p> <p>Custom input field</p> <p>Select an item from the following list:</p> <p>Apple</p> <p>The selected option is Apple</p> <p>OK</p>
<b>onClick</b>	<pre>function OnClickButton () {     alert ("You clicked on the button!"); } &lt;button onclick="OnClickButton ()&gt;Click on me!&lt;/button&gt;</pre>	
<b>onDblClick</b>	<pre>&lt;script type="text/javascript"&gt;     function OnDblClickSpan () {         alert ("You have double- clicked on the text!");</pre>	

	<pre>         }       &lt;/script&gt;       &lt;span         ondblclick="OnDoubleClickSpan ()&gt;Click on         me twice and fast!&lt;/span&gt;     </pre>	
<b>onFocus</b>	<pre> function OnFocusInput (input) {     input.style.color = "blue"; } &lt;input type="text"   onfocus="OnFocusInput (this)"   Value="Change color on focus "/&gt; </pre>	 <p>When the following input field has the focus its text color will be red. Click into the input field!</p> <p>Custom input field Select an item from the following list: Apple Click on me Click on me twice and fast! Hello</p>
<b>onKeydown</b>	<pre> function GetChar (event){     var keyCode = ('which' in       event) ? event.which : event.keyCode;     alert ("The Unicode key       code is: " + keyCode); } &lt;input size="40" value="Write a   character into this field!"   onkeydown="GetChar (event);"/&gt; </pre>	 <p>When the following input field has the focus its text color will be red. Click into the input field!</p> <p>Custom input field Select an item from the following list: Apple Click on me Click on me twice and fast! Hello Write a character into this field!</p> <p>The Unicode key code is: 65</p>
<b>onkeyup</b>	<pre> function GetChar (event){     var keyCode = ('which' in       event) ? event.which : event.keyCode;     alert ("The Unicode key       code is: " + keyCode); } &lt;input size="40" value="Write a   character into this field!"   onkeyup="GetChar (event);"/&gt; </pre>	
<b>onKeyPress</b>	<pre> function GetChar (event){     var keyCode = ('which' in       event) ? event.which : event.keyCode;     alert ("The Unicode key       code is: " + keyCode); } &lt;input size="40" value="Write a   character into this field!"   onKeyPress="GetChar (event);"/&gt; </pre>	



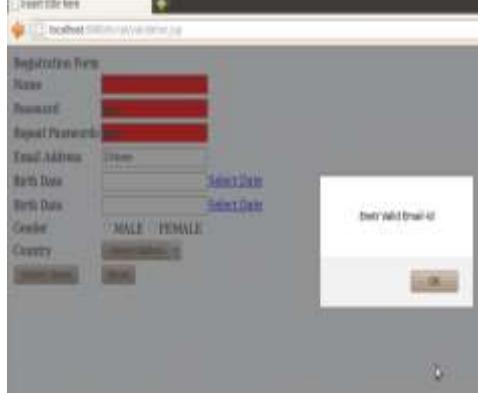
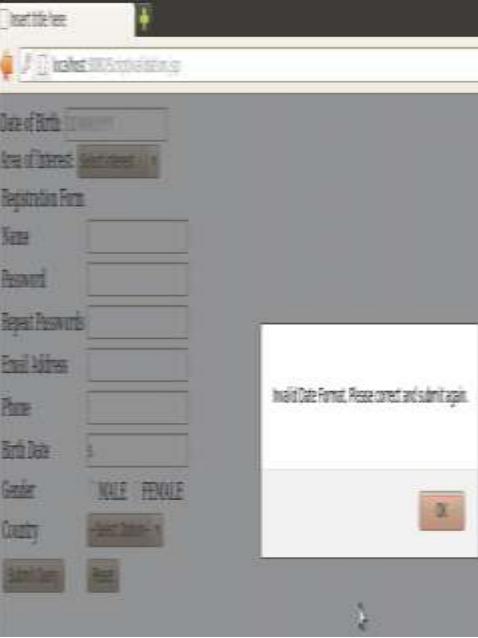
	<p><i>character into this field!"</i>          onkeypress="GetChar (event);"/&gt;</p>	
<b>onLoad</b>	<pre>function OnLoadDocument () {     alert ("The document has been           loaded."); document.body.style.backgroundColor =     "red"; &lt;body onload="OnLoadDocument             ();&gt;</pre>	

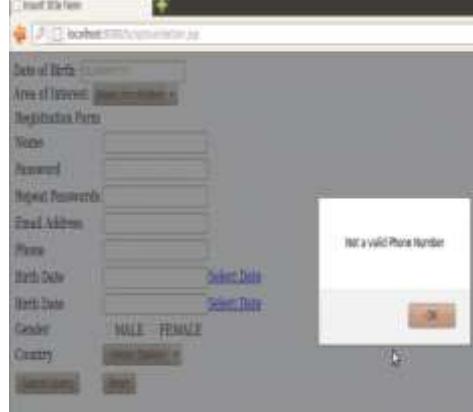
## Mouse Event

```
function OnButtonDown (button) {button.style.color = "#ff0000"; }
function OnButtonUp (button) { button.style.color = "#000000"; }
function OnMouseIn (elem) { elem.style.border = "2px solid blue"; }
function OnMouseOut (elem) { elem.style.border = ""; }
function UpdateFlyingObj (event) {
    var mouseX = Math.round (event.clientX);
    var mouseY = Math.round (event.clientY);
    var flyingObj = document.getElementById ("obj");
    flyingObj.style.left = mouseX + "px";
    flyingObj.style.top = mouseY + "px";
}
```

## JavaScript Validation

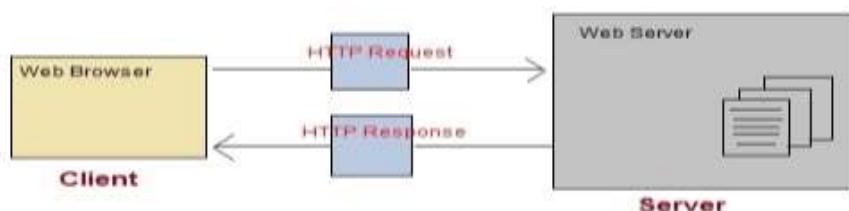
Name Validation	<pre>         function check_uname() {             var uname = document.getElementById("uname");             var User = /^[a-zA-Z ]+\$/;             if (uname.value == "") {                 alert("Enter Value");                 uname.style.background = 'red';             } else if (!User.test(uname.value)) {                 alert("Enter Valid Character");                 uname.style.background = 'red';             }         }  &lt;input type="text" name="uname" id="uname" onBlur="check_uname()"/&gt;</pre>	
Password Validation	<pre>         function check_pass() {             var pass = document.getElementById("new_pass");             var Password = /^[a-zA-Z0-9_?]+\$/;             if (pass.value == "") {                 alert("Enter Value");                 pass.style.background = 'red';             } else if (!Password.test(pass.value)) {                 alert("Enter Valid Character");                 pass.style.background = 'red';             }         }          function reppass() {             var pass = document.getElementById("new_pass");             var rep = document.getElementById("rep");             if (rep.value == "") {                 alert("Enter Password");                 rep.style.background = 'red';             } else if (pass.value != rep.value) {                 alert("Enter Same Password");                 rep.style.background = 'red';             }         }</pre>	

Email validation	<pre> function emailid() {     var ep =         document.getElementById("email");     var ei = /^[a-zA-Z0-9.-_]+@[a-zA-Z0- 9]+\.[a-zA-Z]{2,4}\$/;     if (ep.value == "") {         alert("Enter Email-id");         email.style.background = 'red';     } else if (!ei.test(ep.value)) {         alert("Enter Valid Email-id");         email.style.background = 'red';     } } </pre>	
Date Validation	<pre> function checkdate(input) {     var validformat = /^(\d{2})/(\d{2})/(\d{4})\$/     ;     var returnval = false;     if (!validformat.test(input.value))         alert("Invalid Date Format. Please correct and submit again.");     else {         var monthfield =             input.value.split("/")[0];         var dayfield =             input.value.split("/")[1];         var yearfield =             input.value.split("/")[2];         var dayobj = new Date(yearfield,             monthfield - 1, dayfield);         if ((dayobj.getMonth() + 1 != monthfield)    (dayobj.getDate() != dayfield)                (dayobj.getFullYear() != yearfield))             alert("Invalid Day, Month, or Year range detected. Please correct and submit again.");         else             returnval = true;     }     if (returnval == false)         input.select();     return returnval; } &lt;td&gt;Birth Date&lt;/td&gt; </pre>	

	<td><input type="text" name="firstinput" onblur="return checkdate(this);"></td></tr>	
Phone Number Validation	<pre>function phonenumber(inputtxt) {     var phoneno = /^(\d{10})\$/;     if (inputtxt.value.match(phoneno)) {         return true;     } else {         alert("Not a valid Phone Number");         return false;     } }</pre>	

## Introduction of Client Server Architecture

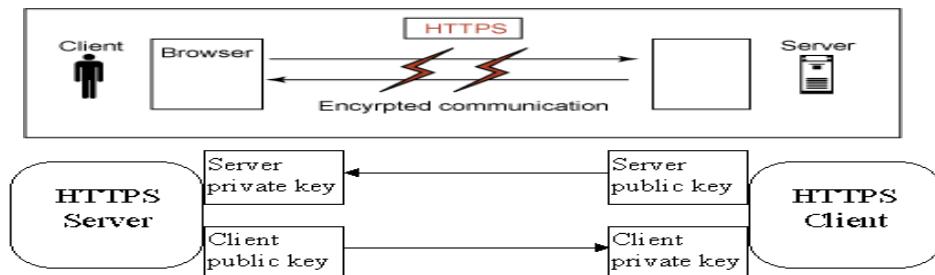
- The type of computing system in which one powerful workstation serves the requests of other systems, is an example of client server technology. A computer network is an interconnection of computers which share various resources.
- Clients are the individual components which are connected in a network. They have a basic configuration. Client sends a request/query to server and server responds accordingly. Please note that the client doesn't share any of its resources. They are subordinates to servers, and their access rights are defined by servers only. They have localized databases.
- The client-server model is a distributed application structure in computing that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.
- HTTP**
  - The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.
  - Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.
  - Short for HyperText Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter a



URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

- **HTTPS**

- Hypertext Transfer Protocol Secure (HTTPS) is a communications protocol for secure communication over a computer network, with especially wide deployment on the Internet.
- Hyper Text Transfer Protocol Secure (HTTPS) is a secure version of the Hyper Text Transfer Protocol (http). HTTPS allows secure ecommerce transactions, such as online banking.
- Web browsers such as Internet Explorer and Firefox display a padlock icon to indicate that the website is secure, as it also displays https:// in the address bar.



- **HTTPD**

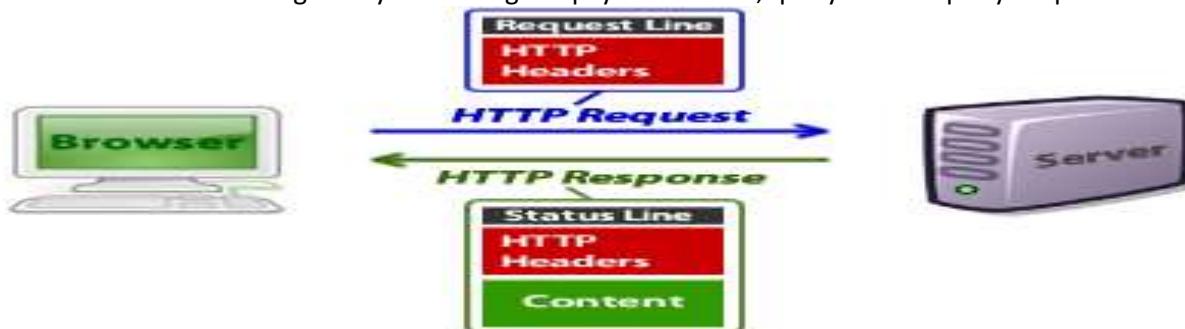
- **Hypertext Transfer Protocol daemon (HTTPD)** On the Web, each server has an *HTTPD* or Hypertext Transfer Protocol daemon that waits in attendance for requests to come in from the rest of the Web. A *daemon* is a program that is "an attendant power or spirit" (Webster's). It sits waiting for requests to come in and then forwards them to other processes as appropriate.

## HTTP Overview

- HTTP is connectionless: The HTTP client ie. browser initiates an HTTP request and after a request is made, the client disconnects from the server and waits for a response. The server processes the request and re-establishes the connection with the client to send response back.
- HTTP is media independent: This means, any type of data can be sent by HTTP as long as both the client and server know how to handle the data content. This is required for client as well as server to specify the content type using appropriate MIME-type.
- HTTP is stateless: As mentioned above, HTTP is a connectionless and this is a direct result that HTTP is a stateless protocol. The server and client are aware of each other only during a current request. Afterwards, both of them forget about each other. Due to this nature of the protocol, neither the client nor the browser can retain information between different requests across the web pages.
- HTTP makes use of the Uniform Resource Identifier (URI) to identify a given resource and to establish a connection. Once connection is established, messages are passed in a format similar to that used by Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045].
- HTTP is based on **client-server architecture model** and a **stateless request/response protocol** that operates by exchanging messages across a reliable connection.
- An HTTP "**client**" is a program (Web browser or any other client) that establishes a connection to a server for the purpose of sending one or more HTTP requests.
- An HTTP "**server**" is a program (generally a web service like Apache Web Server or Internet Information Services IIS etc.) that accepts connections in order to service HTTP requests by sending HTTP responses.

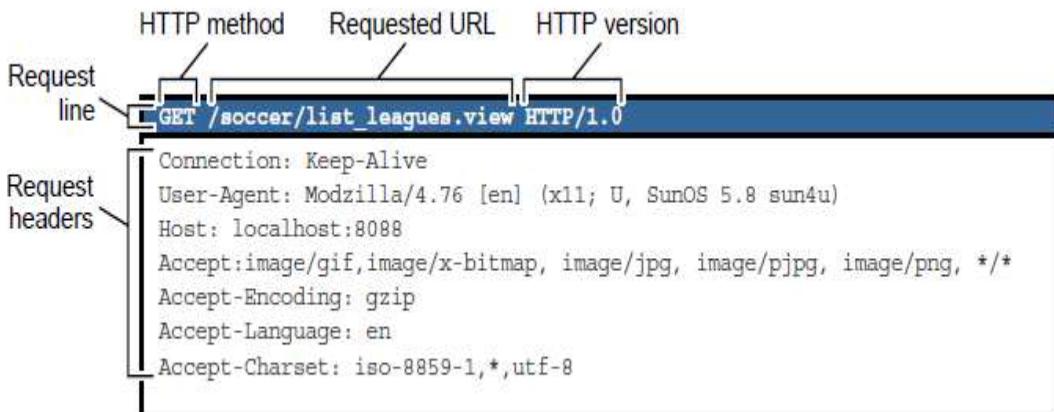
## Client Request Message

- A client sends an HTTP request to a server in the form of a request message which includes following format:
  - Request-line that includes a method, URI, and protocol version
  - Header fields containing request modifiers, client information, and representation metadata
  - An empty line to indicate the end of the header section
  - A message body containing the payload like file, query data or query output.



## Server Response Message

- A server responds to a client's request by sending one or more HTTP response messages having the following format:
  - Status line that includes the protocol version, a success or error code, and textual reason phrase.
  - Header fields containing server information, resource metadata, and representation metadata
  - An empty line to indicate the end of the header section
  - A message body containing the payload like file, query data or query output.



Header	Uses
Accept	The Mime types the client can receive
Host	The internet host and port number of the resources being requested
Refere	The address from which the Request-Universal Resource identifier(URI) was obtained
User-Agent	The information about the client originating the request

## Example Conversation

The following example illustrates a typical message exchange for a GET request on the URI "<http://www.example.com/hello.txt>":

- **Client request:**

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
    • Server response:
HTTP/1.1 200 OK Date: Mon, 27 October 2013 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 October 2013 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
```

## The GET Method

Note that query strings (name/value pairs) is sent in the URL of a GET request:

/test/demo\_form.asp?name1=value1&name2=value2

- **GET requests:**

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data

- GET requests have length restrictions
- GET requests should be used only to retrieve data

## The Post Method

Note that query strings (name/value pairs) is sent in the HTTP message body of a POST request:

POST /test/demo\_form.asp HTTP/1.1

Host: w3schools.com

name1=value1&name2=value2

- Some other notes on POST requests:
  - POST requests are never cached
  - POST requests do not remain in the browser history
  - POST requests cannot be bookmarked
  - POST requests have no restrictions on data length

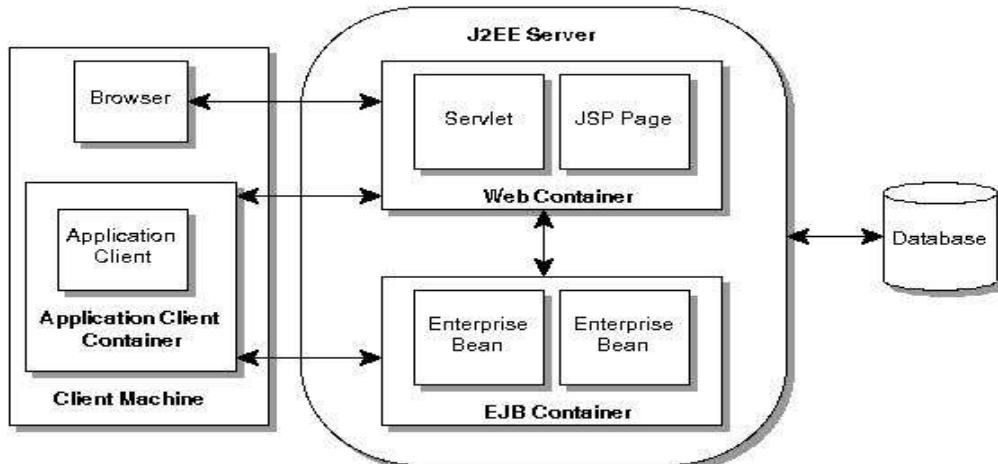
## J2EE Architecture

### J2EE Server

- The J2EE server provides the following services:
- Naming and Directory - allows programs to locate services and components through the Java Naming and Directory Interface(JNDI) API
- Authentication - enforces security by requiring users to log in
- HTTP - enables Web browsers to access servlets and Java Server Pages(JSP) files
- EJB - allows clients to invoke methods on enterprise beans

### EJB Container

- Enterprise bean instances run within an EJB container. The container is a runtime environment that controls the enterprise beans and provides them with important system-level services. Since



you don't have to develop these services yourself, you are free to concentrate on the business methods in the enterprise beans.

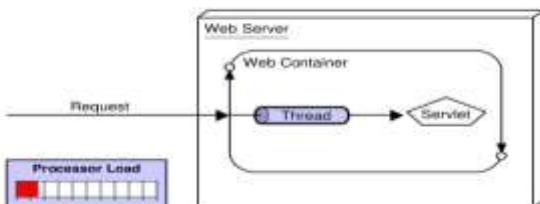
## Web Components Development

### Common Gateway Interface(CGI)

- The CGI connects Web servers to external applications.

# TOPS Technologies

How servlets work with one request:



- CGI is the interface between server programs and other software.
- CGI can do two things.
  - It can gather information sent from a web browser to a web server, and make the information available to an external program.

- CGI can send the output of a program to a Web browser that request it.

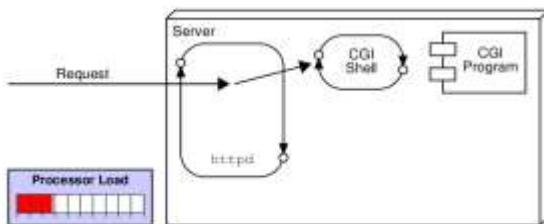
## CGI Process



## CGI Programming Process

- Phase 1: Create.
- Phase 2: Request/Execute.
- Phase 3: Respond/Display.

How CGI works with one request:



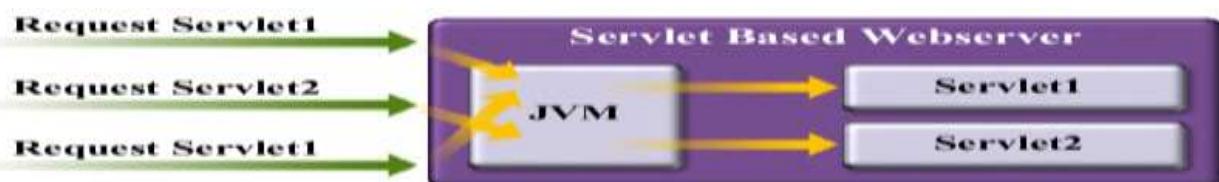
## Execution of CGI Programs

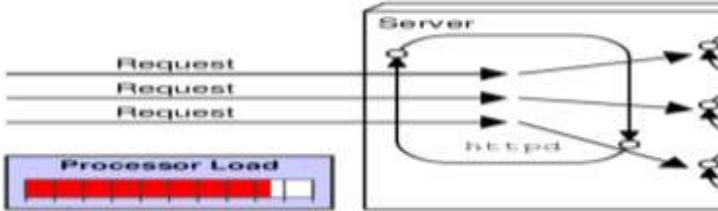
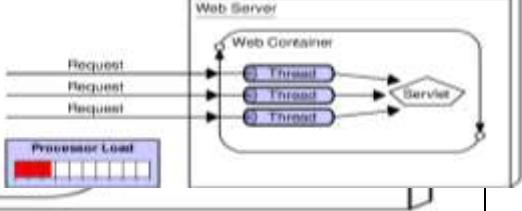
### Advantages and Disadvantages of CGI Programs

- CGI Advantages
  - Written in a variety of languages
  - Relatively easy for a web designer to reference
- CGI Disadvantages
  - If number of clients increases, it takes more time for sending response.
  - For each request, it starts a process and Web server is limited to start processes.
  - It uses platform dependent language e.g. C, C++, perl.

## Servlet Programming Process

- A **Servlet** is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications



CGI	Servlet
<b>How CGI works with many requests:</b>	<b>How servlets work with many requests:</b>
	
Request Request Request  Processor Load	Request Request Request  Processor Load
Written in C, C++, Visual Basic and Perl	Written in Java
Difficult to maintain, non-scalable, non-manageable	Powerful, reliable, and efficient
Prone to security problems of programming language	Improves scalability, reusability (component based)
Resource intensive and inefficient	Leverages built-in security of Java programming language
Platform and application-specific	Platform independent and portable

hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.

- The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.
- This chapter focuses on writing servlets that generate responses to HTTP requests.
- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

## Servlet Process

### Execution of Servlet Programs

#### CGI vs Servlet

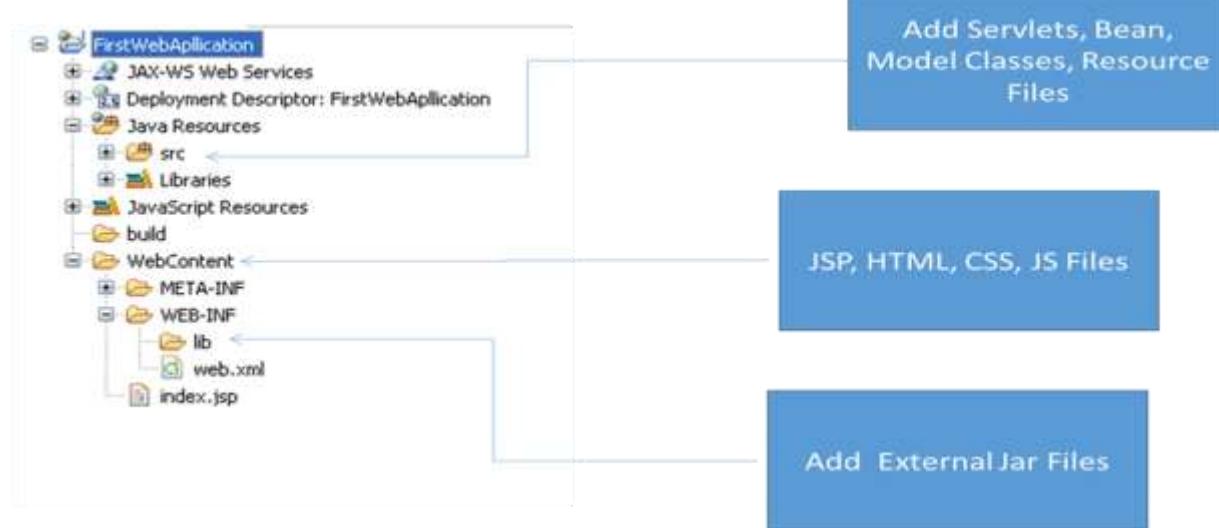
#### Servlet Advantages Over CGI

There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a

common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

- **Better performance:** because it creates a thread for each request not process.
- **Portability:** because it uses java language.
- **Robust:** Servlets are managed by JVM so no need to worry about momory leak, garbage collection etc.
- **Secure:** because it uses java language..

## Web Application Structure



## Servlets

### Introduction

#### What is Servlet

Servlets are Java technology's answer to CGI programming. They are programs that run on a Web server and build Web pages. Building Web pages on the fly is useful (and commonly done) for a number of reasons:

- **The Web page is based on data submitted by the user.** For example the results pages from search engines are generated this way, and programs that process orders for e-commerce sites do this as well.
- **The data changes frequently.** For example, a weather-report or news headlines page might build the page dynamically, perhaps returning a previously built page if it is still up to date.
- **The Web page uses information from corporate databases or other such sources.** For example, you would use this for making a Web page at an on-line store that lists current prices and number of items in stock.

#### Advantages of JAVA Servlet

Java servlets are more efficient, easier to use, more powerful, more portable, and cheaper than traditional CGI and than many alternative CGI-like technologies. (More importantly, servlet developers get paid more than Perl programmers :-).

- **Efficient.**
  - With traditional CGI, a new process is started for each HTTP request.
  - If the CGI program does a relatively fast operation, the overhead of starting the process can dominate the execution time.

- With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process.
- Similarly, in traditional CGI, if there are  $N$  simultaneous requests to the same CGI program, then the code for the CGI program is loaded into memory  $N$  times.
- With servlets, however, there are  $N$  threads but only a single copy of the servlet class.
- Servlets also have more alternatives than do regular CGI programs for optimizations such as caching previous computations, keeping database connections open, and the like.
- **Convenient.**
  - Besides the convenience of being able to use a familiar language, servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.
- **Powerful.**
  - Java servlets let you easily do several things that are difficult or impossible with regular CGI.
  - For one thing, servlets can talk directly to the Web server (regular CGI programs can't).
  - This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement.
  - They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- **Portable.**
  - Servlets are written in Java and follow a well-standardized API.
  - Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar.
  - Servlets are supported directly or via a plugin on almost every major Web server.
- **Inexpensive.**
  - There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites.
  - However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive.
  - Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

## Servlet versions

- **Servlet 2.1**
  - **ConsistentLogging**
  - **Removed Redundancy**
  - **Easy Initialisation**
  - **Simple To getSession**
- **Servlet 2.2**
  - The new feature included in servlet 2.2 represents a single web application, it is a collection of servlet, Java Server Pages (JSP), HTML, Images and other web resources.
  - It is very simple to install the Web Application, installation. you have to place only war file in a specific directory.
  - Here the WEB-INF directory is special, it contains all the java classes of a web application and a xml file for mapping these java classes.

- The directory META-INF contains the meta information about archive contents.
- **Servlet 2.3**
  - Servlet have now dependant on J2SDK 1.2 or JDK 1.2 from its 2.3 specification. This specification have added many new features into the servlet.
  - Some of the important feature are -
    1. Filter
    2. Lifecycle Events
- **Servlet 2.4**
  - There are many feature added in servlet 2.4. This specification enhanced many classes and interfaces.
  - The new feature added in **ServletRequest** interface
  - New feature has been added in **RequestDispatcher** interface
  - Single Thread model have been deprecated in Servlet 2.4 specification.
  - New methods HttpSession.logout() has been added in Session interface. and now session allows to accept zero and negative value.
- **Servlet 2.5**

To use this feature you must have jdk 1.5 version or above than this. The servlet 2.5 doesn't supports the lower version than jdk 1.5. It has added many features in servlet 2.4

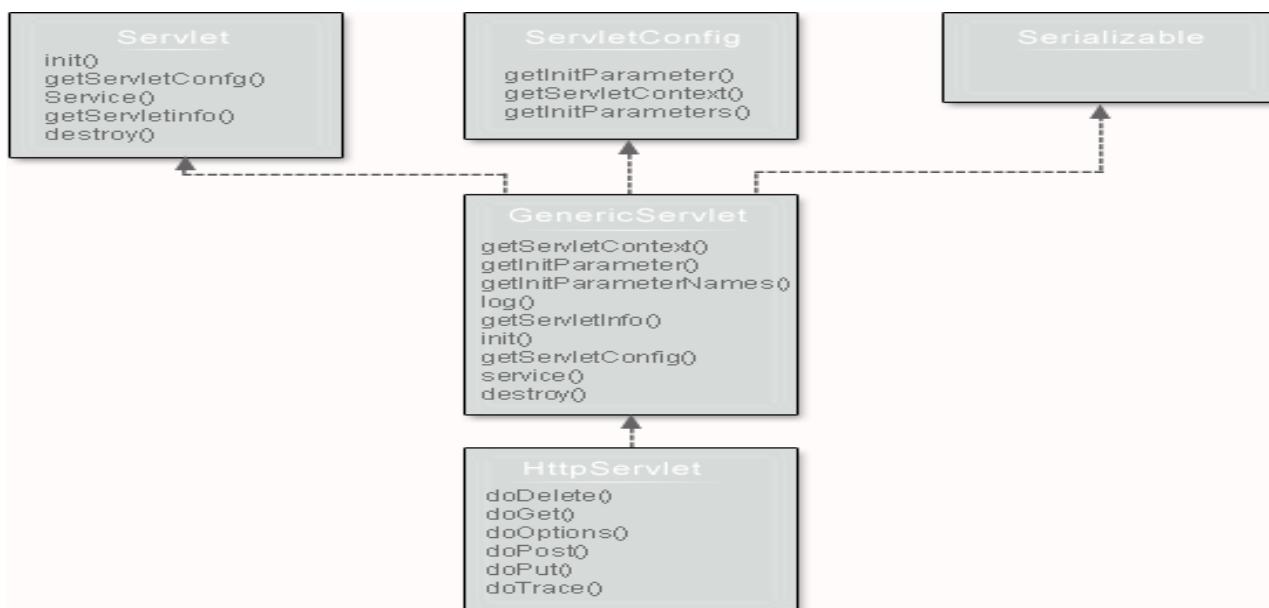
  - **Added Annotations**- The annotations provides a mechanism with metadata for to decorate java codings. They actually mark code in such a way that processor may alter their behavior.
  - **Added convenience web.xml** - This specification added many small changes in web.xml. This makes it more convenient for developers. for example we do in filter mapping as
  - **A Handful of removed restrictions**- Servlet 2.5 have added some restrictions around session tracking and error handling.
  - It removed the restriction that error page could not call the setStatus() method to alter the error code.
  - In session tracking it has made a convenient rule that the servlet called by RequestDispatcher() can not set the response header.
- **Servlet 3.0**
  - Ease of development
  - Pluggability and extensibility
  - Asynchronous support
  - Security enhancements
  - Other miscellaneous changes
- **Servlet 3.1**
  - Non blocking I/O
  - HTTP upgrade
  - Async Servlet
  - Security
  - File Uploading
- **Servlet 4.0**
  - HTTP/2 upgrade

## Types of Servlet

There are mainly two types of servlets -

- **Generic Servlet** - Generic servlet is protocol independent servlet. It implements the Servlet and ServletConfig interface. It may be directly extended by the servlet. Writing a servlet in in GenericServlet is very easy. It has only init() and destroy() method of ServletConfig interface in its life cycle. It also implements the log method of ServletContext interface.
- **Http Servlet** - HttpServlet is HTTP (Hyper Text Transfer Protocol ) specific servlet. It provides an abstract class HttpServlet for the developers for extend to create there own HTTP specific servlets. The sub class of HttpServlet must overwrite at least one method given below-
  - doGet()
  - doPost()
  - doPut()
  - doDelete()
  - init()
  - destroy()
  - getServiceInfo()

There is no need to override service() method. All the servlet either Generic Servlet or Http Servlet passes there config parameter to the Servlet interface.



## Difference between HttpServlet and GenericServlet

### Generic Servlet:

GenericServlet class is direct subclass of Servlet interface.

- Generic Servlet is protocol independent. It handles all types of protocol like http, smtp, ftp etc.
- Generic Servlet only supports service() method. It handles only simple request Public void service(ServletRequest req,ServletResponse res ).
- Generic Servlet only supports service() method.

### HttpServlet

- HttpServlet class is the direct subclass of Generic Servlet.
- HttpServlet is protocol dependent. It handles only http protocol.
- HttpServlet supports public void service(ServletRequest req,ServletResponse res ) and protected void service(HttpServletRequest req,HttpServletResponse res ).

- HttpServlet supports also doGet(), doPost(), doPut(), doDelete(), doHead(), doTrace(), doOptions(), etc.

## Servlet Life Cycle

The life cycle of the servlet is controlled by the servlet container. The servlet container is responsible for doing following task::

- When the first request is made for servlet the container loads the servlet class and initiates it.
- Then make an instance of servlet class.
- Initializes the servlet class by calling init() method.
- Call the service method passes the request response object to the service method.

*If container need to remove the servlet then call the destroy() method to finalize the servlet.*



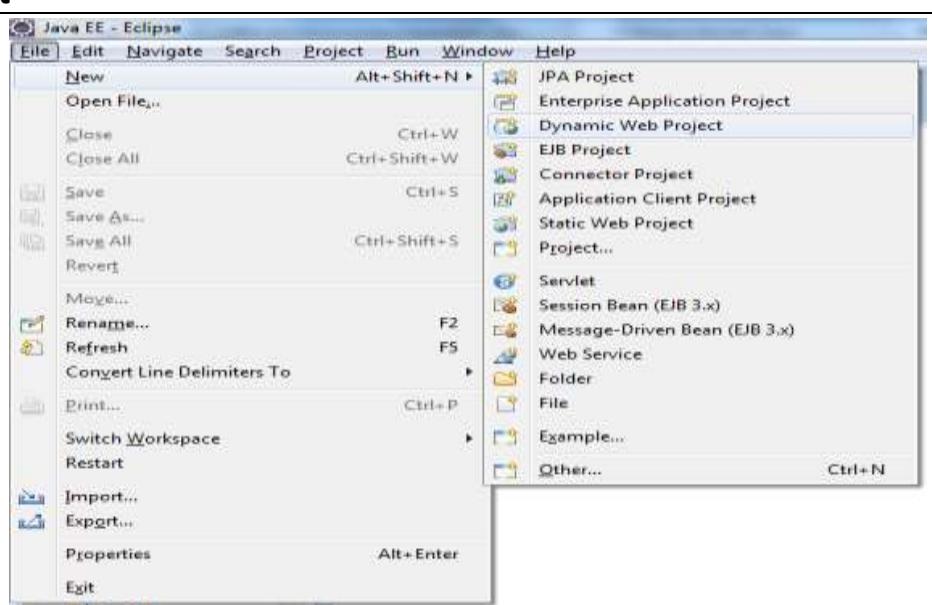
Basically there are three methods in servlet life cycle which servlet calls

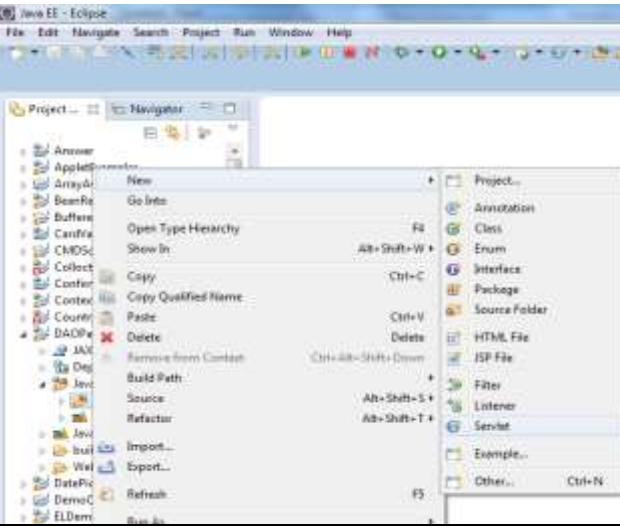
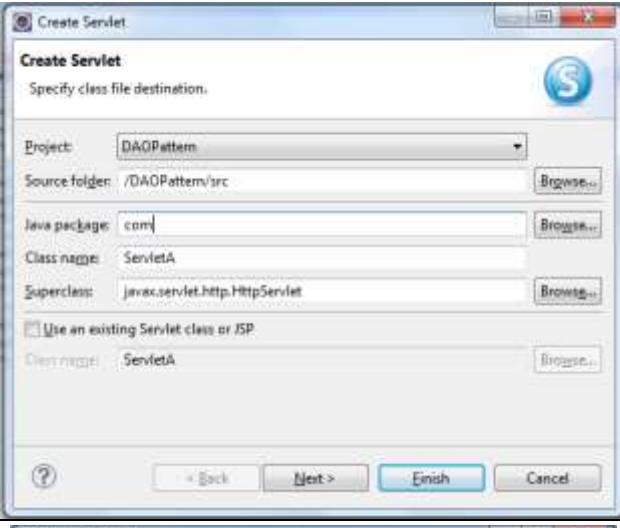
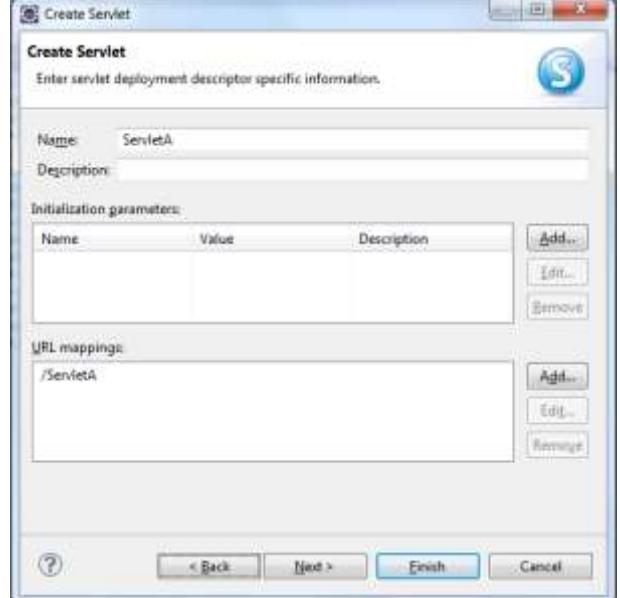
1. **init()**- This method is called to initialize a servlet. In servlet 2.4 specification if you do not call init() method in your servlet program then the container call it implicitly.
2. **service()**- This method is called to execute the business logic written in servlet. This is the method which you must have to override, when you are writing a servlet.
3. **destroy()**- This method is called to finalize the servlet.

**Note-** The servlet call init() and destroy() method only once called during its life cycle.

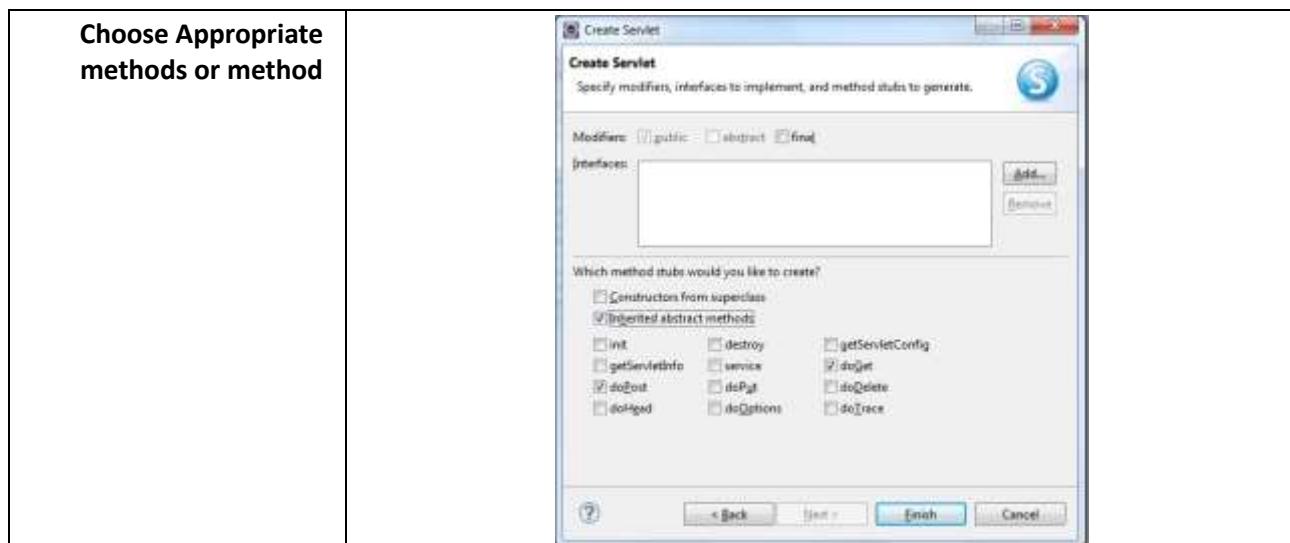
## Creating Servlet

Create a dynamic web project click on File Menu -> New -> dynamic web project



<p><b>Create a servlet explore the project by clicking the (+) icon -&gt; explore the Java Resources -&gt; right click on src -&gt; New -&gt; servlet -&gt; write your servlet name e.g. Hello -&gt; uncheck all the checkboxes except doGet() -&gt; next -&gt; Finish.</b></p>	
<p><b>Give Servlet name and package name</b></p>	
<p><b>Give Appropriate URL pattern (URL Pattern is used to recognize which servlet is to invoke at run time by the container)</b></p>	

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<servlet>
    <servlet-name>abc</servlet-name>
    <servlet-class>com.ServletA</servlet-class>
</servlet>
<servlet>
    <servlet-name>xyz</servlet-name>
    <servlet-class>com.ServletB</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>abc</servlet-name>
    <url-pattern>/ServletA</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>xyz</servlet-name>
    <url-pattern>ServletB</url-pattern>
</servlet-mapping>
</web-app>
```



## Servlet Entry in Web.xml

- You can put servlet mapping into either web.xml file or through annotation.
- In web.xml put servlet name in servlet tag with its class name.
- And mapping file would be in servlet-mapping tag with its servlet name.
- Example of web.xml is mentioned below

## ServletConfig and ServletContext Interface

### ServletContext Interface

Context parameters are the application level parameter. It means these parameters are the one per application. This interface is used to pass parameter to whole application. Once set this parameter can be accessed through out application (In any page, any servlet). It is applicable only within a single Java Virtual Machine. If a web application is distributed between multiple JVM this will not work.

**To set context parameter in web.xml consider following example**

```
<context-param>
<param-name>globalVariable</param-name>
<param-value>tops-int.com</param-value>
</context-param>
```

**To access context parameter in servlet**

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws  
IOException {  
    PrintWriter pw = response.getWriter();  
    pw.println(getServletContext().getInitParameter("globalvariable"));
```

Technology  
Show Servlet

## ServletConfig Interface

ServletConfig is implemented by the servlet container **to initialize a single servlet** using init(). That is, you can pass initialization parameters to the servlet using the web.xml deployment descriptor. This parameter is individual for different servlets. Each Servlet has its specific Config parameter. For understanding, this is similar to a constructor in a java class.

```
<servlet>  
    <servlet-name>ServletConfigTest</servlet-name>  
    <servlet-class>com.ServletA</servlet-class>  
    <init-param>  
        <param-name>mail</param-name>  
        <param-value>admin@gmail.com</param-value>
```

## To access parameter in servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse response) throws  
IOException {  
    PrintWriter pw = response.getWriter();  
    pw.println(getServletConfig().getInitParameter("mail"));
```

## Web application Listener

- Servlet Listener is used for listening to events in a web containers, such as when you create a session, or place an attribute in a session or if you passivate and activate in another container, to subscribe to these events.
- The listeners are like triggers that can be attached to events in your app server with listeners you can track application-level, session-level, life-cycle changes, attribute changes etc.
- You can configure listener in web.xml, for example HttpSessionListener.

```
<context-param>  
    <param-name>url</param-name>  
    <param-value>jdbc:mysql://localhost/student</param-value>  
</context-param>  
<context-param>  
    <param-name>uname</param-name><param-value>root</param-value>  
</context-param>  
<context-param>  
    <param-name>pword</param-name><param-value></param-value>  
</context-param>
```

The implemented interfaces are **javax.servlet.Listener interface.**

```
public class ListenerClass implements ServletContextListener{  
    public void contextDestroyed(ServletContextEvent arg0) { }  
    public void contextInitialized(ServletContextEvent arg0) {  
        ServletContext sc = arg0.getServletContext();  
        String url = sc.getInitParameter("url");  
        String uname = sc.getInitParameter("uname");  
        String pword = sc.getInitParameter("pword");  
        DbConnection db = new DbConnection(url, uname, pword);  
    }  
}
```

## RequestDispatcher Interface

- Defines an object that receives requests from the client and sends them to any resource (such as a servlet, HTML file, or JSP file) on the server.
- The servlet container creates the RequestDispatcher object, which is used as a wrapper around a server resource located at a particular path or given by a particular name.
- This interface is intended to wrap servlets, but a servlet container can create RequestDispatcher objects to wrap any type of resource.
- This interface is used to forward request to another page (HTML or JSP) or to another servlet.

There are main two methods of requestdispatcher interface

- Forward
- Include

The forward() method intended for use in **forwarding** the request, meaning after the response of the calling servlet has been committed. You cannot merge response output using this method.

The include() method merges the response written by the calling servlet, and the activated servlet. This way you can achieve "server side includes" using the include().

**Example of forward method:**

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
    PrintWriter out = response.getWriter();  
    out.print("Tops");  
    out.print("Technology");  
    RequestDispatcher rd = request.getRequestDispatcher("show");  
    rd.forward(request, response);  
}
```

**Example of Include Method**

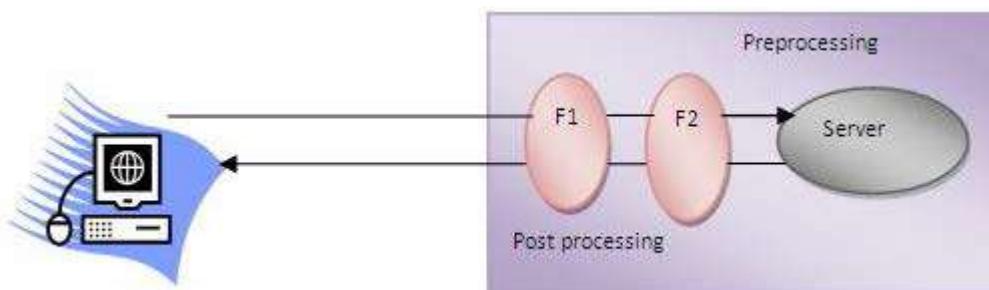
```
protected void doGet(HttpServletRequest request, HttpServletResponse response)  
throws ServletException, IOException {  
    PrintWriter out = response.getWriter();  
    out.print("Tops");  
    out.print("Technology");  
    RequestDispatcher rd = request.getRequestDispatcher("show");  
    rd.include(request, response);  
}
```

Show Servlet

Forward method is just passing request to resource. (No response of current page is included)  
Include method is used to pass request to resource with response of current page.

## Filters

- Servlet Filters are Java classes that can be used in Servlet Programming for the following purposes:
  - To intercept requests from a client before they access a resource at back end.
  - To manipulate responses from server before they are sent back to the client.
- Filter offers a useful way of performing filtered functionality in a Java web application. Typically, filters do not generate content themselves.
- A filter is typically used to perform a particular piece of functionality either before or after the primary functionality of a web application is performed.
- As an example, if a request is made for a particular resource such as a servlet and a filter is used, the filter code may execute and then pass the user on to the servlet.
- As a further example, the filter might determine that the user does not have permissions to access a particular servlet, and it might send the user to an error page rather than to the requested resource.



- A filter is an object that is used to perform filtering tasks such as conversion, log maintain, compression, encryption and decryption, input validation etc.
- A filter is invoked at the preprocessing and post-processing of a request. It is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So it will be easier to maintain the web application.

### Why we are using Filter?

Filters are not servlet; they don't actually create a response. They are preprocessors of the request before it reaches a servlet, and/or postprocessors of the response leaving a servlet. As you'll see later in the examples, a filter can:

- Intercept a servlet's invocation before the servlet is called
- Examine a request before a servlet is called
- Modify the request headers and request data by providing a customized version of the request object that wraps the real request
- Modify the response headers and response data by providing a customized version of the response object that wraps the real response

- Intercept a servlet's invocation after the servlet is called.
- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

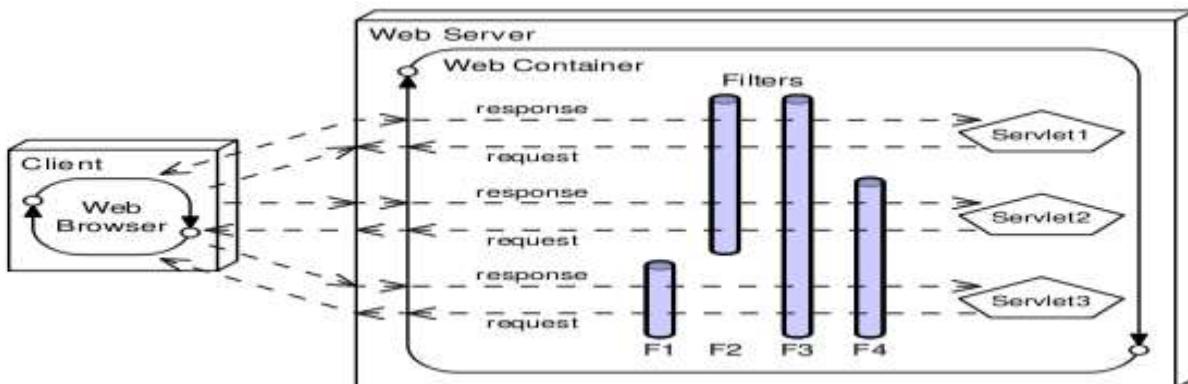
You can configure a filter to act on a servlet or group of servlets. Zero or more filters can filter one or more servlets.

### Advantages of Filter

- A filter can intercept a servlet's invocation before the servlet is called.
- Can examine a request before a servlet is called.
- Can modify the request headers and request data by providing a customized version of the request object that wraps the real request.
- Can modify the response headers and response data by providing a customized version of the response object that wraps the real response.
- Intercept a servlet's invocation after the servlet is called.
- Filter is pluggable.
- One filter don't have dependency onto another resource.

### Applying Filter

- A filter intercepts the request before it gets to the requested resources.
- A response is returned to the client through the filter.
- Multiple filters can intercept a given request.
- This provides for modularity and reuse of code.
- Filter can be applied to different requests in different combinations.
- Filter can be applied to internal dispatch, such as a request forward or include.
- This behavior is determined by the information in the deployment descriptor.



**Filter API:**

- Filter
- FilterChain
- FilterConfig
- Filters are deployed in the deployment descriptor file **web.xml** and then map to either servlet names or URL patterns in your application's deployment descriptor.
- When the web container starts up your web application, it creates an instance of each filter that you have declared in the deployment descriptor.
- The filters execute in the order that they are declared in the deployment descriptor.

## Filter Interface

For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

- **public void init(FilterConfig config) throws ServletException**
  - init() method is invoked only once it is used to initialize the filter.

Called by the web container to indicate to a filter that it is being placed into service. The servlet container calls the init method exactly once after instantiating the filter. The init method must complete successfully before the filter is asked to do any filtering work. The web container cannot place the filter into service if the init method either:

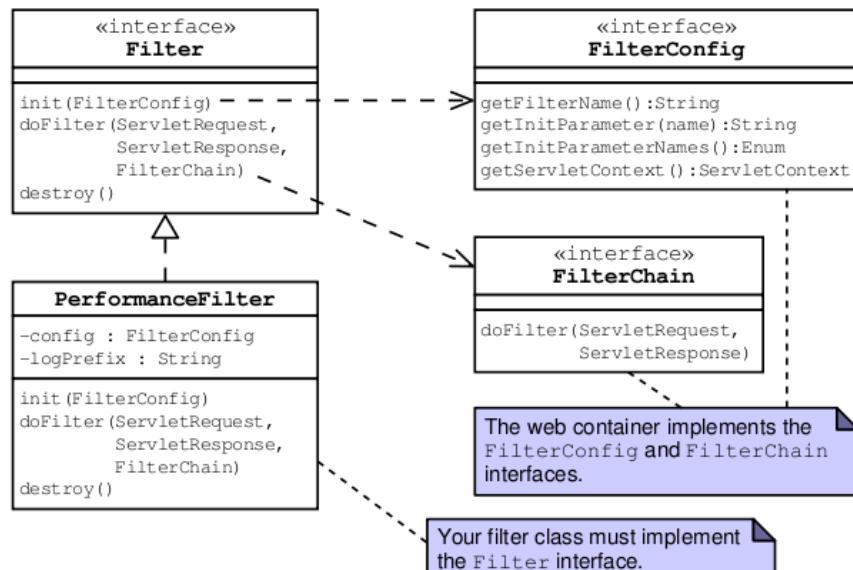
  1. Throws a ServletException
  2. Does not return within a time period defined by the web container
- **public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException**

- **doFilter()** method is invoked every time when user request to any resource, to which the filter is mapped. It is used to perform filtering tasks.

- The **doFilter()** method of the Filter is called by the container each time a request/response pair

is passed through the chain due to a client request for a resource at the end of the chain. The FilterChain passed in to this method allows the Filter to pass on the request and response to the next entity in the chain.

A typical implementation of this method would follow the following pattern:



Examine the request

Optionally wrap the request object with a custom implementation to filter content or headers for input filtering

Optionally wrap the response object with a custom implementation to filter content or headers for output filtering

Either invoke the next entity in the chain using the FilterChain object(chain.doFilter()), or not pass on the request/response pair to the next entity in the filterchain to block the request processing

Directly set headers on the response after invocation of the next entity inthe filter chain.

### **public void destroy()**

This is invoked only once when filter is taken out of the service.

Called by the web container to indicate to a filter that it is being taken out of service. This method is only called once all threads within the filter's doFilter method have exited or after a timeout period has passed. After the web container calls this method, it will not call the doFilter() method again on this instance of the

filter. This method gives the filter an opportunity to cleanup any resources that are being held (for example, memory, file handles, threads) and make sure that any persistent state is synchronized with the filter's current state in memory.

### **FilterChain Interface**

The object of FilterChain is responsible to invoke the next filter or resource in the chain. This object is passed in the doFilter() method of Filter interface. A FilterChain is an object provided by the servlet container to the developer giving a view into the invocation chain of a filtered request for a resource.

Filters use the FilterChain to invoke the next filter in the chain, or if the calling filter is the last filter in the chain, to invoke the resource at the end of the chain.

The FilterChain interface contains only one method:

- **public void doFilter(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException**
  - It passes the control to the next filter or resource.
  - Causes the next filter in the chain to be invoked, or if the calling filter is the last filter in the chain, causes the resource at the end of the chain to be invoked.

### **FilterConfig Interface**

- A filter configuration object used by a servlet container to pass information to a filter during initialization.

### **Methods of FilterConfig**

<b>public String getFilterName()</b>	<i>Returns the filter-name of this filter as defined in the deployment descriptor.</i>
<b>public String getInitParameter(String name)</b>	<i>Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.</i>
<b>public Enumeration getInitParameterNames ()</b>	<i>Returns the names of the filter's initialization parameters as an Enumeration of a String objects, or an empty Enumeration if the filter has no</i>

	<i>initialization parameters.</i>
<b>public ServletContext getServletContext ()</b>	Returns a reference to the ServletContextin which the caller is executing.

[Click Me!](#)

### **Index.jsp**

```
<form action="FirstServlet">
    <input type="submit" value="Click Me!!"/>
</form>
```

### **Web.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>cookieTest</display-name>
    <filter>
        <filter-name>Filter2</filter-name>
        <filter-class>com.filter.Filter2</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Filter2</filter-name>
        <url-pattern>/FirstServlet</url-pattern>
    </filter-mapping>
    <filter>
        <filter-name>Filter1</filter-name>
        <filter-class>com.filter.Filter1</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>Filter1</filter-name>
        <url-pattern>/FirstServlet</url-pattern>
    </filter-mapping>
    <servlet>
        <servlet-name>FirstServlet</servlet-name>
        <servlet-class>com.servlet.FirstServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>FirstServlet</servlet-name>
        <url-pattern>/FirstServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

### **FirstServlet.java**

**Filter1.java**

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Filter1 implements Filter {
    public void destroy() {
        System.out.println("DestoryFilter.....");
    }
    public void doFilter(ServletRequest request, ServletResponse response,
                         FilterChain chain) throws IOException, ServletException
    {
        System.out.println("Send Request....");
        chain.doFilter(request, response);
        System.out.println("Response changes...");
    }
    public void init(FilterConfig fConfig) throws ServletException {
        System.out.println("Init filter ....");
    }
}
```

**Filter2.java**

```
import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Filter2 implements Filter {
    public void destroy() {
        System.out.println("DestoryFilter2.....");
    }
}
```

## JSP (Java Server Pages)

- **Java Server Pages** is a server side programming language.
- It is an object oriented language that uses the Java servlets technology.
- JSP provides the flexibility to handle large amount of dynamic data, databases with performance and stability it has the ability to integrate with HTML very easily to enhance the presentation of a page.
- JSP pages helps to differentiate the design from the programming logic of a web page. Since it is platform independent it can be used with any server.
- To use Java Server Pages, both JDK and Tomcat server have to be installed.

## JSP Page Translation

- A java servlet file is generated from the JSP source file.
- This is the first step in its tedious multiple phase life cycle.
- In the translation phase, the container validates the syntactic correctness of the JSP pages and tag files.
- The container interprets the standard directives and actions, and the custom actions referencing tag libraries used in the page.

## JSP Life Cycle

- The generated java servlet file is compiled into a java servlet class.

Note: The translation of a JSP source page into its implementation class can happen at any time between initial deployment of the JSP page into the JSP container and the receipt and processing of a client request for the target JSP page.

- *Class Loading:* The java servlet class that was compiled from the JSP source is loaded into the container.
- *Execution phase:* In the execution phase the container manages one or more instances of this class in response to requests and other events.  
*The interface JspPage contains jspInit() and jspDestroy().*
- *The JSP specification has provided a special interface HttpJspPage for JSP pages serving HTTP requests and this interface contains \_jspService()*
  - *Initialization:* jspInit() method is called immediately after the instance was created. It is called only once during JSP life cycle.
  - *JspService () execution:* This method is called for every request of this JSP during its life cycle.
  - *This is where it serves the purpose of creation. Oops! it has to pass through all the above steps to reach this phase. It passes the request and the response objects. \_jspService() cannot be overridden.*
  - *JspDestroy () execution:* This method is called when this JSP is destroyed. With this call the servlet serves its purpose and submits itself to heaven (garbage collection). This is the end of jsp life cycle.

*JspInit (), \_jspService () and jspDestroy () are called the life cycle methods of the JSP.*

## JSP Comments

- Since JSP is built on top of HTML, we can write comments in JSP file like html comments as
  - <-- This is HTML Comment -->
  - These comments are sent to the client and we can look it with view source option of browsers.
- We can put comments in JSP files as:
  - <%-- This is JSP Comment--%>

- This comment is suitable for developers to provide code level comments because these are not sent in the client response.

## JSP Directives

There are 3 types of directives in JSP

- Import
- Page
- Include

It is possible to use "import" statements in JSPs, but the syntax is a little different from normal Java. Try the following example:

```
<%@ page import="java.util.*" %>
<HTML><BODY>
<%
System.out.println( "Evaluating date now" );
%>
```

characters.

- This one is a "page directive". The page directive can contain the list of all imported packages. To import more than one item, separate the package names by commas, e.g.
- <%@ page import="java.util.\* , java.text.\*" %>
- There are a number of JSP directives, besides the page directive. Besides the page directives, the other most useful directives are include and taglib. We will be covering taglib separately.
- The include directive is used to physically include the contents of another file. The included file can be HTML or JSP or anything else -- the result is as if the original JSP file actually contained the included text. To see this directive in action, create a new JSP

```
<HTML>
<BODY>
Going to include hello.jsp...<BR>
<%@ include file="hello.jsp" %>
</BODY>
```

- View this JSP in your browser, and you will see your original hello.jsp get included in the new JSP.

## JSP Scriptlets

- It is difficult to do much programming just by putting Java expressions inside HTML.
- JSP also allows you to write blocks of Java code inside the JSP. You do this by placing your Java code between <% and %> characters.
- This block of code is known as a "scriptlet". By itself, a scriptlet doesn't contribute any HTML (though it can, as we will see down below.)
- A scriptlet contains Java code that is executed every time the JSP is invoked.
- Here is a modified version of our JSP adding in a scriptlet.

```
<HTML>
<BODY>
<%
    // This is a scriptlet. Notice that the "date"
    // variable we declare here is available in the
    // embedded expression later on.
System.out.println( "Evaluating date now" );
java.util.Date date = new java.util.Date();
%>
```

- If you run the above example, you will notice the output from the "System.out.println" on the server log. This is a convenient way to do simple debugging.
- By itself a scriptlet does not generate HTML.
- If a scriptlet wants to generate HTML, it can use a variable called "out". This variable does not need to be declared. It is already predefined for scriptlets, along with some other variables. The following example shows how the scriptlet can generate HTML output.

```
<HTML>
<BODY>
<%
    // This scriptlet declares and initializes "date"
System.out.println( "Evaluating date now" );
java.util.Date date = new java.util.Date();
%>
Hello! The time is now
<%
    // This scriptlet generates HTML output
out.println( String.valueOf( date ) );
```

- Another very useful pre-defined variable is "request". It is of type `javax.servlet.http.HttpServletRequest`
- A "request" in server-side processing refers to the transaction between a browser and the server. When someone clicks or enters a URL, the browser sends a "request" to the server for that URL, and shows the data returned.
- As a part of this "request", various data is available, including the file the browser wants from the server, and if the request is coming from pressing a SUBMIT button, the information the user has entered in the form fields.
- The JSP "request" variable is used to obtain information from the request as sent by the browser.
- For instance, you can find out the name of the client's host (if available, otherwise the IP address will be returned.) Let us modify the code as shown:
- A similar variable is "response".
- This can be used to affect the response being sent to the browser.

- For instance, you can call `response.sendRedirect( anotherUrl );` to send a response to the browser that it should load a different URL.
- This response will actually go all the way to the browser.
- The browser will then send a different request, to "anotherUrl".
- This is a little different from some other JSP mechanisms we will come across, for including another page or forwarding the browser to another page.

## JSP Expression

- Since most of the times we print dynamic data in JSP page using `out.print()` method, there is a shortcut to do this through JSP Expressions.
- JSP Expression starts with `<%=` and ends with `%>`.
  - `<% out.print("TOPS"); %>` can be written using JSP Expression as `<%= "TOPS" %>`
  - Notice that anything between `<%= %>` is sent as parameter to `out.print()` method.
- Also notice that scriptlets can contain multiple java statements and always ends with semicolon `(;)` but expression doesn't end with semicolon.

## JSP Declaration

- The JSP you write turns into a class definition.
- All the scriptlets you write are placed inside a single method of this class.
- You can also add variable and method declarations to this class.
- You can then use these variables and methods from your scriptlets and expressions.
  - To add a declaration, you must use the `<%!` and `%>` sequences to enclose your declarations, as shown below.

```
<%@ page import="java.util.*" %>
<HTML><BODY>
<%!
    Date theDate = new Date();
    Date getDate()
    {
        System.out.println( "In getDate() method" );
        return theDate;
    }
}
```

## JSP Implicit Object

- JSP Implicit objects are created by the web container.
- These implicit objects are Java objects that implement interfaces in the Servlet and JSP API.
- Scripting elements in a JSP page can make use of these JSP implicit objects.
- There are nine (9) JSP implicit objects as follows:

### 1. request implicit object

- The JSP implicit request object is an instance of a java class that implements the `javax.servlet.http.HttpServletRequest` interface.
- It represents the request made by the client.

- The request implicit object is generally used to get request parameters, request attributes, header information and query string values.

## 2. response implicit object

- The JSP implicit response object is an instance of a java class that implements the javax.servlet.http.HttpServletResponse interface.
- It represents the response to be given to the client.
- The response implicit object is generally used to set the response content type, add cookie and redirect the response.

```
<HTML>
<BODY>
<% // This scriptlet declares and initializes "date"
System.out.println( "Evaluating date now" );
java.util.Date date = new java.util.Date();
%>
Hello! The time is now
<%
out.println( date );
out.println( "<BR>Your machine's address is " );
out.println( request.getRemoteHost() );
```

## 3. out implicit object

- The JSP implicit out object is an instance of the javax.servlet.jsp.JspWriter class.
- It represents the output content to be sent to the client.
- The out implicit object is used to write the output content.

## 4. session implicit object

- The JSP implicit session object is an instance of a java class that implements the javax.servlet.http.HttpSession interface.
- It represents a client specific conversation.
- The session implicit object is used to store session state for a single user.

## 5. application implicit object

- The JSP implicit application object is an instance of a java class that implements the javax.servlet.ServletContext interface.
- It gives facility for a JSP page to obtain and set information about the web application in which it is running

## 6. exception implicit object

- The JSP implicit exception object is an instance of the java.lang.Throwable class.
- It is available in JSP error pages only. It represents the occurred exception that caused the control to pass to the JSP error page.

## 7. config implicit object

- The JSP implicit config object is an instance of the java class that implements javax.servlet.ServletConfig interface.
- It gives facility for a JSP page to obtain the initialization parameters available.

## 8. page implicit object

- The JSP implicit page object is an instance of the java.lang.Object class.

- It represents the current JSP page.
- That is, it serves as a reference to the java servlet object that implements the JSP page on which it is accessed.
- It is not advisable to use this page implicit object often as it consumes large memory.

## 9. pageContext implicit object

- The JSP implicit pageContext object is an instance of the javax.servlet.jsp.PageContext abstract class.
- It provides useful context information.
- That is it provides methods to get and set attributes in different scopes and for transferring requests to other resources.
- Also it contains the reference to to implicit objects.

## JSP Action

- Servlet container provides many built in functionality to ease the development of the applications. Programmers can use these functions in JSP applications.
- The JSP Actions tags enables the programmer to use these functions.
- The JSP Actions are XML tags that can be used in the JSP page.
- JSP Tags or Action Elements represent dynamic actions that occur at runtime.

### List of JSP Actions

Action Tags	Description
<b>jsp:include</b>	<ul style="list-style-type: none"><li>• The jsp:include action work as a subroutine, the Java servlet temporarily passes the request and response to the specified JSP/Servlet.</li><li>• Control is then returned back to the current JSP page.</li></ul>
<b>jsp:param</b>	<ul style="list-style-type: none"><li>• The jsp:param action is used to add the specific parameter to current request.</li><li>• The jsp:param tag can be used inside a jsp:include, jsp:forward or jsp:params block.</li></ul>
<b>jsp:forward</b>	<ul style="list-style-type: none"><li>• The jsp:forward tag is used to hand off the request and response to another JSP or servlet.</li><li>• In this case the request never return to the calling JSP page.</li></ul>
<b>jsp:plugin</b>	<ul style="list-style-type: none"><li>• In older versions of Netscape Navigator and Internet Explorer; different tags is used to embed applet.</li><li>• The jsp:plugin tag actually generates the appropriate HTML code the embed the Applets correctly.</li></ul>
<b>jsp:fallback</b>	<ul style="list-style-type: none"><li>• The jsp:fallback tag is used to specify the message to be shown on the browser if applets is not supported by browser.<ul style="list-style-type: none"><li>• Example: <pre>&lt;jsp:fallback&gt; &lt;p&gt;Unable to load applet&lt;/p&gt; &lt;/jsp:fallback&gt;</pre></li></ul></li></ul>
<b>jsp:getProperty</b>	<ul style="list-style-type: none"><li>• The jsp:getProperty is used to get specified property from the JavaBean object.</li></ul>
<b>jsp:setProperty</b>	<ul style="list-style-type: none"><li>• The jsp:setProperty tag is used to set a property in the JavaBean object.</li></ul>

<b>jsp:useBean</b>	<ul style="list-style-type: none"> <li>• The <code>jsp:useBean</code> tag is used to instantiate an object of Java Bean or it can re-use existing java bean object.</li> </ul>
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## JSTL

- JSP Standard Tag Library (JSTL) was introduced to ease the programming in JSP by storing generic tasks in tag libraries under four different categories based on conditional processing and looping, XML processing, Internationalization and formatting, database access and a set of expression language functions.

### Syntax:

```
<%@ taglib uri="URIToTagLibrary" prefix="tagPrefix"%>
```

- In the above example the "uri" points to the tag library "matlib.tld", the prefix used in the taglib directive as well as in the coding is same that is "mtaglib".
- The following table lists the URI for the JSTL 1.1 libraries.

<b>Library</b>	<b>URI</b>	<b>Prefix</b>
Core	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	<u>c</u>
XML Processing	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	<u>x</u>
I18N formatting	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	<u>fmt</u>
Database access	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	<u>sql</u>
Functions	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	<u>fn</u>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<% @ taglib uri="http://java.sun.com/jsp/jstl/functions"
prefix="fn"%>
<html>
<head><title>JSTL Functions</title></head>
<form>
USERNAME:<input type="text" name="usr" size="25">
<p></p>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
```

- In the above example first taglib directive points to the JSTL core tag library with prefix "c". So the "<c:out>" statement works fine.
- For the functions like "{\$fn:length(param.usr)}" that can be invoked using the JSP Expression Language we include the JSTL function library with prefix "fn".
- The above example outputs the length of the username entered.

## JSP Custom tags

- Custom tags are mainly used to customize the usage of java in a JSP page.
- Usually these tags define different objects and classes, so that it can be used in a JSP page with a simple syntax.
- Custom tags have a unique "prefix" to refer a particular tag library file.
- The following is the directory structure that can help you understand where the files, folders are located.

- Under Application folder create WEB-INF folder.
- Inside WEB-INF Folder create classes and tlds folder.
- In classes folder we will have the java class, servlets and packages.
- In tlds class we will have tld files which contains tag information.

All the classes and packages are inside the "classes" folder.

## Applicability to Industry

### Session Management

- When we are using multi-user application, there is a need to identify the user, which one is currently processing. In regardance of that we use HttpSession in java.
- HttpSession is interface from javax.servlet.http package.
- It helps us in keeping track of the user.
- As well it provides a way to identify a user across more than one page request or visit to a website and to store info about the user.
- The servlet container uses this interface to create a session between HttpClient and HttpServer.
- Session is a conversational state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.
- This interface 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 connections
- When container migrates a session between VMs in a distributed container setting, all session attributes implementing the HttpSessionActivationListener interface are notified.

## What we need for Session Management

- HttpSession object is used to entire session with a specific client.
- We can store, retrieve and remove attributes from HttpSession object.
- Any session can have access to HttpSession object through the getSession() method of the HttpServletRequest object.

Creating a session object	
<b>HttpSession session = request.getSession();</b>	getSession() method returns a session. If the session already exist, it will return the existing session else create a new session object.
<b>HttpSession session = request.getSession(true);</b>	getSession(true) will always create a new session object and returns it.
<b>HttpSession session = request.getSession(false);</b>	getSession(false) will never create a new session it will returns an existing session object.
Destroying a session object	
<b>session.invalidate();</b>	This will destroy existing session object.

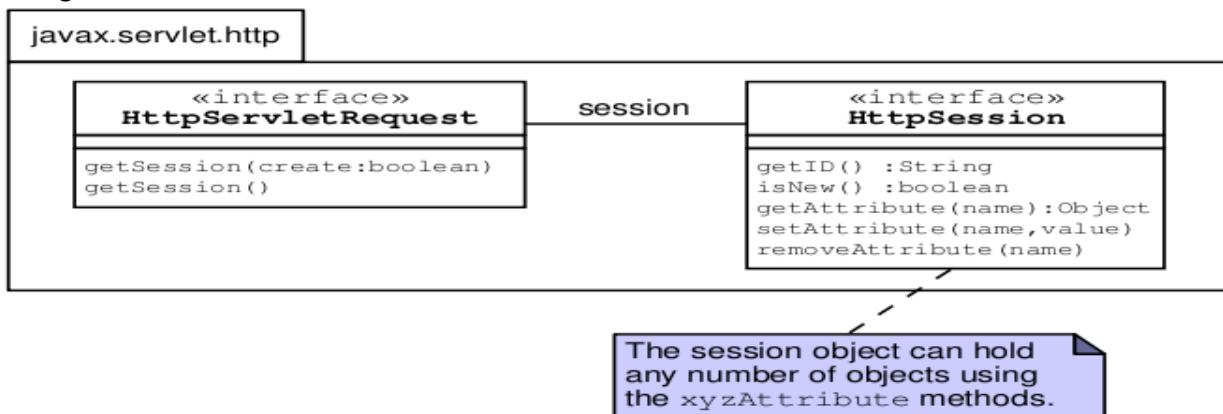
## Session Tracking Technique

There are several ways through which we can provide unique identifier in request and response.

- **User Authentication** – This is the very common way where we user can provide authentication credentials from the login page and then we can pass the authentication information between server and client to maintain the session. This is not very effective method because it won't work if the same user is logged in from different browsers.
- **HTML Hidden Field** – We can create a unique hidden field in the HTML and when user starts navigating, we can set its value unique to the user and keep track of the session. This method can't be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field. Also it's not secure because we can get the hidden field value from the HTML source and use it to hack the session. By using hidden form fields we can insert information in the web pages and this information will be sent to the server. These fields are not visible directly to the user, but can be viewed using view source option from the browsers. The hidden form fields are as given below:
 

```
<input type='hidden' name='site Name' value='hidden Value'/>
```
- **URL Rewriting** – We can append a session identifier parameter with every request and response to keep track of the session. This is very tedious because we need to keep track of this parameter in every response and make sure it's not clashing with other parameters. With this method, the information is carried through url as request parameters. In general added parameter will be sessionid, userid.
- **Cookies** – You can use HTTP cookies to store information. Cookies will be stored at browser side. Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. We can maintain a session with cookies but if the client disables the cookies, then it won't work.
- **Session Management API** – Session Management API is built on top of above methods for session tracking. Some of the major disadvantages of all the above methods are:
  - Most of the times we don't want to only track the session, we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.
  - All the above methods are not complete in themselves; all of them won't work in a particular scenario. So we need a solution that can utilize these methods of session tracking to provide session management in all cases.
  - Using HttpSession, we can store information at server side. HttpSession provides methods to handle session related information.

That's why we need **Session Management API** and J2EE Servlet technology comes with session management API that we can use.



## Using Cookies

- A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- Package javax.servlet.http.Cookie
- Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
- A cookie's value can uniquely identify a client, so cookies are commonly used for session management.
- A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
- Some Web browsers have bugs in how they handle the optional attributes, so use them sparingly to improve the interoperability of your servlets.

### Constructors of Cookie class:

**Cookie(String name, String value):** Constructs a cookie with a specified name and value.

### Commonly used Methods of Cookie class:

There are given some commonly used methods of the Cookie class.

- **public void setMaxAge(int expiry):** Sets the maximum age of the cookie in seconds.
- **public String getName():** Returns the name of the cookie. The name cannot be changed after creation.
- **public String getValue():** Returns the value of the cookie.

### Advantages of Cookies:

- Simplest technique of maintaining the state.
- Cookies are maintained at client side.

### Disadvantages of Cookies:

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

## **Index.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<body>
    <form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
</body>
</html>
```

## **web.xml**

```
<web-app version="2.5"
    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_2_5.xsd">
<servlet>
    <servlet-name>Servlet1</servlet-name>
    <servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>SecondServlet</servlet-name>
    <servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Servlet1</servlet-name>
    <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SecondServlet</servlet-name>
```

## **FirstServlet.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse
response){
    try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String n=request.getParameter("userName");
        out.print("Welcome "+n);
        Cookie ck=new Cookie("uname",n);//creating cookie object
        response.addCookie(ck);//adding cookie in the response
        //creating submit button
        out.print("<form action='servlet2' method='post'>");
        out.print("<input type='submit' value='go'>");
        out.print("</form>");
        out.close();
    }catch(Exception e){System.out.println(e); } }
```

## **SecondServlet.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SecondServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse
response){
    try{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        Cookie ck[]=request.getCookies();
        out.print("Hello "+ck[0].getValue());
```

## **Using HttpSession**

### **Creating session object:**

Servlet API provides Session management through HttpSession interface. We can get session from HttpServletRequest object using following methods. HttpSession allows us to set objects as attributes that can be retrieved in future requests.

- **HttpSession getSession()** – This method always returns a HttpSession object. It returns the session object attached with the request, if the request has no session attached, then it creates a new session and return it.

- **HttpSession getSession(boolean flag)** – This method returns HttpSession object if request has session else it returns null.

Some of the important methods of HttpSession are:

- **String getId()** – Returns a string containing the unique identifier assigned to this session.
- **Object getAttribute(String name)** – Returns the object bound with the specified name in this session, or null if no object is bound under the name. Some other methods to work with Session attributes are getAttributeNames(),removeAttribute(String name) and setAttribute(String name, Object value).
- **long getCreationTime()** – Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. We can get last accessed time with getLastAccessedTime() method.
- **setMaxInactiveInterval(int interval)** – Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. We can get session timeout value from getMaxInactiveInterval() method.
- **ServletContext getServletContext()** – Returns ServletContext object for the application.
- **boolean isNew()** – Returns true if the client does not yet know about the session or if the client chooses not to join the session.
- **void invalidate()** – Invalidates this session then unbinds any objects bound to it.

Name:

## Index.jsp

```
<html> <body>
<form action="create">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form> </body> </html>
```

## Web.xml

```
<web-app version="2.5"
    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_2_5.xsd">
    <servlet> <servlet-name>createSession</servlet-name>
    <servlet-class>tops.sessionExample.createSession</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>createSession</servlet-name>
        <url-pattern>/createSession</url-pattern>
    </servlet-mapping>
    <servlet> <servlet-name>checkSession</servlet-name>
    <servlet-class>tops.sessionExample.checkSession</servlet-class>
    </servlet> <servlet-mapping>
        <servlet-name>checkSession</servlet-name>
        <url-pattern>/checkSession</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
```

**createSession.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class createSession extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response){
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String n=request.getParameter("userName");
            out.print("Welcome "+n);
            HttpSession session=request.getSession();
            session.setAttribute("uname",n);
            session.setMaxInactiveInterval(2*60*60);
            //Session will terminate automatically after 2*60*60 seconds
            out.print("<a href='checkSession'>visit</a>");
            out.close();
        }catch(Exception e){System.out.println(e);}
    }
}
```

**checkSession.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class checkSession extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        try{
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            HttpSession session=request.getSession(false);
```

**Design Patterns**

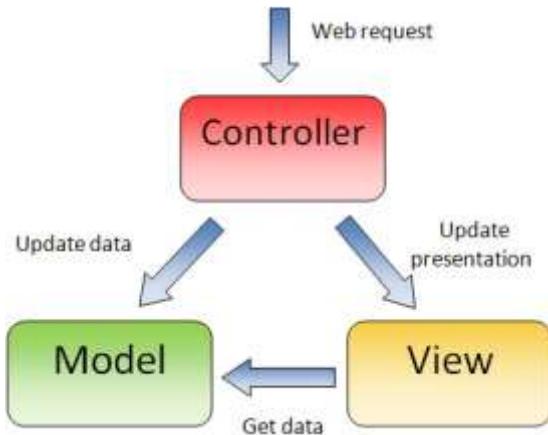
- When we are creating any application sometimes it is becoming tedious for us to maintain each and every part of our application.
- User interfaces are especially prone to change requests.
- Different user place conflicting requirements on the user interface.

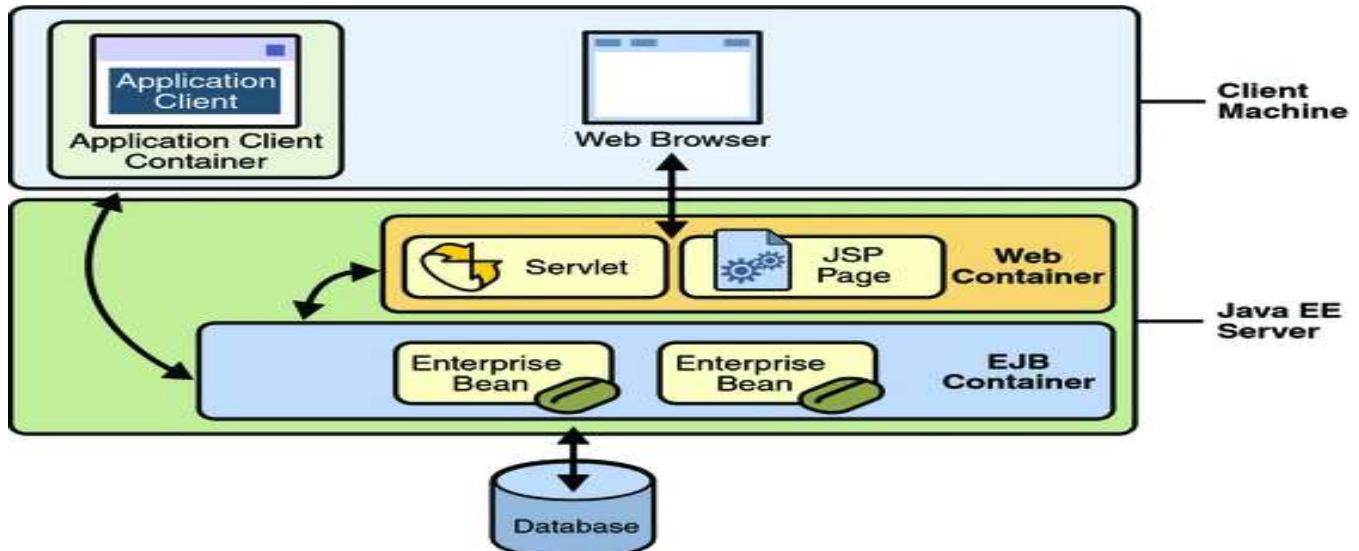
- Building a system with the flexibility is expensive and error-prone if the user interface is tightly interwoven with the functional core.
- To overcome this problems we are using design patterns that are easy to use and access.
- Design patterns represent solutions to problems that arise when developing software within a particular context.
- In software engineering, a design pattern is a general reusable solution to a commonly occurring problem within a given context in software design.
- It is a description or template for how to solve a problem that can be used in many different situations
- Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Model View controller is a classical design pattern used in applications who needs a clean separation between their business logic and view who represents data. MVC design pattern isolates the application logic from the user interface and permitted the individual development, testing and maintenance for each component.

This design pattern is divided into three parts.

- **Model** - This component manages the information and notify the observers when the information changes. It represents the data when on which the application operates. The model provides the persistent storage of data, which manipulated by the controller.
- **View** - The view displays the data, and also takes input from user. It renders the model data into a form to display to the user. There can be several view associated with a single model. It is actually representation of model data.
- **Controller** - The controller handles all requests coming from the view or user interface. The data flow to whole application is controlled by controller. It forwarded the request to the appropriate handler. Only the controller is responsible for accessing model and rendering it into various UIs.





## Example

### **Index.jsp**

```
<form action="ControllerServlet" method="post">
    Name:<input type="text" name="name"><br>
    Password:<input type="password" name="password"><br>
    <input type="submit" value="login">
</form>
```

### **login-error.jsp**

```
<p>Sorry! username or password error</p>
<% @ include file="index.jsp" %>
```

### **LoginBean.java**

```
package com.bean;
public class LoginBean {
    private String name,password;
    //Generate Getter and Setter method
    ...
```

## Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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_3_0.xsd"
  id="WebApp_ID" version="3.0">

  <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>com.example.ControllerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>s1</servlet-name>
    <url-pattern>/ControllerServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

## ControllerServlet.java

```
package com.example;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ControllerServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String name=request.getParameter("name");
        String password=request.getParameter("password");
        LoginBean bean=new LoginBean();
        bean.setName(name);
        bean.setPassword(password);
        request.setAttribute("bean",bean);
        boolean status=bean.validate();
        if(status){
            RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
            rd.forward(request, response);
        }
        else{
            RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
            rd.forward(request, response);
        }
    }
}
```

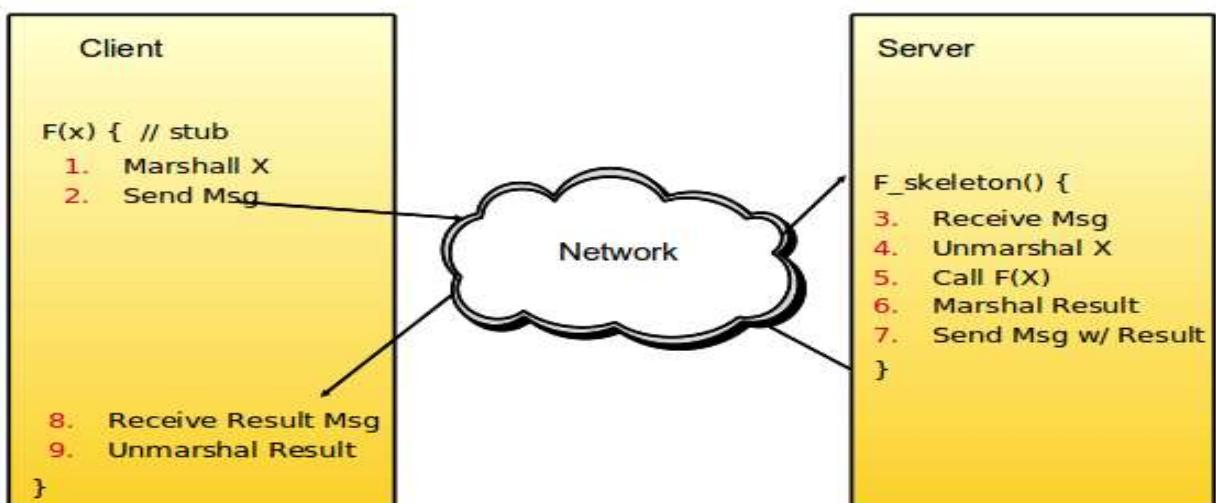
- Distributed Applications in Java
- Client-Server:

The client is the entity accessing the remote resource and the server provides access to the resource. Operationally, the client is the caller and the server is the callee.

- In Java terms:
  - The client is the invoker of the method and the server is the object implementing the method.
- The client and the server can be heterogeneous:
  - Different implementation languages
  - Different operating systems
- The roles can be transient
  - The definition is with respect to a particular interaction.
- Client and Server refer both to the code and the system on which the code is running

## RMI (Remote Method Invocation)

- Java RMI allowed programmer to execute remote function class using the same semantics as local function calls.
- To transfer data from one host to another.
- RMI is a specification that enables one JVM to invoke methods in an object located in another JVM.
- These JVMs could be running on different computers or running as separate processes on the same computer
- The server must first bind its name to the registry
- The client looks up the server name in the registry to establish remote references.
- The Stub serializes the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.



## EJB (Enterprise Java Beans)

Enterprise JavaBeans is a specification for creating server-side scalable, transactional, multi-user secure enterprise-level applications. It provides a consistent component architecture framework for creating distributed n-tier middleware. It would be fair to call a bean written to EJB spec a Server Bean.

A typical EJB Architecture consists of an EJB server, EJB containers that runs on these servers, EJBs that run in these containers, EJB clients and other auxiliary systems like the Java Naming and Directory Interface (JNDI ) and the Java Transaction Service (JTS).

They contain business logic that operates on the enterprise's data.

They depend on a container environment to supply life-cycle services for them. EJB instances are created and maintained by the container.

They can be customized at deployment time by editing the deployment descriptor.

System-level services, such as transaction management and security, are described separately from the enterprise bean.

A client never accesses an enterprise bean directly; the container environment mediates access for the client. This provides component-location transparency.

## Web Services

- This is the most important benefit of Web Services. Web Services typically work outside of private networks, offering developers a non-proprietary route to their solutions. Web Services also let developers use their preferred programming languages. In addition, thanks to the use of standards-based communications methods, Web Services are virtually platform-independent.
- Data is isolated between applications creating 'silos'. Web Services act as glue between these and enable easier communications within and across organisations.

## Why use WebServices

- Exposing the function on to network: A Web service is a unit of managed code that can be remotely invoked using HTTP, that is, it can be activated using HTTP requests. So, Web Services allows you to expose the functionality of your existing code over the network.
- Once it is exposed on the network, other application can use the functionality of your program.
- Connecting Different Applications: Web Services allows different applications to talk to each other and share data and services among themselves. Other applications can also use the services of the web services.
- For example VB or .NET application can talk to java web services and vice versa. So, Web services is used to make the application platform and technology independent.

## Types of Web-Service

1. SOAP Web Service

2. RESTful Web Service

### Soap Web Service

- Simple Object Access Protocol (SOAP) is a standard protocol specification for message exchange based on XML.
- Communication between the web service and client happens using XML messages. SOAP defines the rules for communication like what are all the tags that should be used in XML and their meaning.

### RESTful Web Service

- RESTful web service uses architectures that use HTTP or similar protocols by restricting the interface to use standard operations like GET, POST, PUT, DELETE for HTTP.

- Based on my experience RESTful is easier to develop. I know this statement will invite wrath of SOAP lovers.

## Restful WebServices Introduction

- In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.
- The response from server is considered as the representation of the resources. This representation can be generated from one resource or more number of resources.



- HTTP methods :

- RESTful web services use HTTP protocol methods for the operations they perform. Methods are:
- GET : It defines a reading access of the resource without side-effects. This operation is idempotent i.e. they can be applied multiple times without changing the result
- PUT : It creates a new resource. It must also be idempotent.
- DELETE : It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
- POST : It updates an existing resource or creates a new resource.

## Restful Web Services Annotations

### @Path()

- Its a Class & Method level of annotation
- This will check the path next to the base URL

### Syntax :

Base URL :

`http://localhost:(port)/<YourApplicationName>/<UrlPattern In Web.xml>/<path>`

Here `<path>` is the part of URI, and this will be identified by `@path` annotation at class/method level, you will be able to understand in the next RESTful hello world tutorial.

### @PUT

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP PUT request only.

### @DELETE

Its a method level of annotation, this annotation indicates that the following method should respond to the HTTP DELETE request only.

## @Produces

Its a method or field level annotation, This tells which MIME type is delivered by the method annotated with @GET. I mean when e

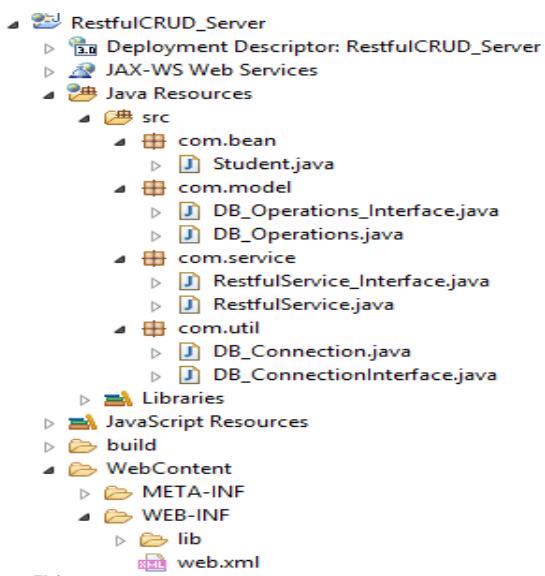
## @Consumes

This is a class and method level annotation, this will define which MIME type is consumed by the particular method. I mean in which format the method can accept the input from the client.

- Every we send a HTTP GET request to our RESTful service, it will invokes particular method and produces the output in different formats. There you can specifies in what are all formats (MIME) your method can produce the output, by using @produces annotation.

**Remember:** We will use @Produces annotation for GET requests only.

## Restful WebService Example



## Server Creation

### WEB.XML Using Jersey Framework

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
  app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>RestfulCRUD_Server</display-name>
  <welcome-file-list>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <servlet>
```

```
<servlet-name>MyRestCrud</servlet-name>
<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
<init-param>
<param-name>jersey.config.server.provider.packages</param-name>
<param-value>com.service</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>MyRestCrud</servlet-name>
<url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>
```

### **Student.java**

```
package com.bean;
public class Student
{
private int id;
private String fname, lname, emailid;
//getters/setters
```

### **DB\_ConnectionInterface.java**

```
package com.util;
import java.sql.Connection;
public interface DB_ConnectionInterface
{
    public Connection getConnection();
}
```

### **DB\_Connection.java**

```
package com.util;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class DB_Connection implements DB_ConnectionInterface
{
    public Connection getConnection()
    {
        Connection con=null;
        String host, db, url, uname, pass;
        host = "jdbc:mysql://localhost:3306/";
        db = "restful_crud";
        url = host+db;
        uname = "root";
        pass = "";
        try {
```

```
Class.forName("com.mysql.jdbc.Driver");
    con=DriverManager.getConnection(url,uname,pass);
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (SQLException e) {
    e.printStackTrace();
}

return con;
}
}
```

## **DB\_Operations\_Interface.java**

```
package com.model;
import java.util.List;
import com.bean.Student;
public interface DB_Operations_Interface
{
    public int insertData(Student student);
    public int deleteData(int id);
    public int updateData(Student student);
    public List<Student> viewData();

}
```

## **DB\_Operations.java**

```
package com.model;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import com.bean.Student;
import com.util.DB_Connection;
public class DB_Operations implements DB_Operations_Interface
{
    Connection con=null;
    PreparedStatement pst=null;
    DB_Connection db=new DB_Connection();
    @Override
    public int insertData(Student student)
    {
        int result=0;
        con=db.getConnection();
        String sql="insert into student(fname, lname, emailid) values(?, ?, ?)";

```

```
try {
    pst=con.prepareStatement(sql);
    pst.setString(1, student.getFname());
    pst.setString(2, student.getLname());
    pst.setString(3, student.getEmailId());
    result=pst.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
return result;
}
@Override
public int deleteData(int id)
{
    int result=0;
    con=db.getConnection();
    String sql="delete from student where id=?";
    try {
        pst=con.prepareStatement(sql);
        pst.setInt(1, id);
        result=pst.executeUpdate();
        result=pst.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return result;
}
@Override
public int deleteData(int id)
{
    int result=0;
    con=db.getConnection();
    String sql="delete from student where id=?";
    try {
        pst=con.prepareStatement(sql);
        pst.setInt(1, id);
        result=pst.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return result;
}
@Override
public int updateData(Student student)
{
    int result=0;
```

```
con=db.getConnection();
String sql="update student set fname=?, lname=?, emailid=? where id=?";
try {
    pst=con.prepareStatement(sql);
    pst.setString(1, student.getFname());
    pst.setString(2, student.getLname());
    pst.setString(3, student.getEmailId());
    pst.setInt(4, student.getId());
    result=pst.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
}
return result;
}
@Override
public List<Student> viewData()
{
    List<Student> list=new ArrayList<Student>();
    con=db.getConnection();
    String sql="select * from student";
    try {
        pst=con.prepareStatement(sql);
        ResultSet rs=pst.executeQuery();
        while(rs.next()){
            Student student=new Student();
            student.setFname(rs.getString("fname"));
            student.setLname(rs.getString("lname"));
            student.setEmailId(rs.getString("emailid"));
            student.setId(rs.getInt("id"));
            list.add(student);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return list;
}
}
```

## **RestfulService\_Interface.java**

```
package com.service;
public interface RestfulService_Interface
{
    String addStudent(String student);
    String updateStudent(String student);
    String deleteStudent(int id);
    String viewAllStudents();
```

}

## **RestfulService.java**

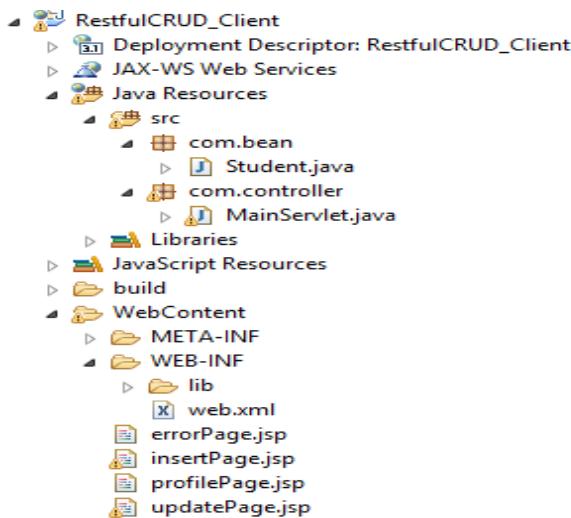
```
package com.service;
import java.util.List;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;
import com.bean.Student;
import com.google.gson.Gson;
import com.model.DB_Operations;
import com.model.DB_Operations_Interface;
@Path("student")
public class RestfulService implements RestfulService_Interface
{
    Gson gson=new Gson();
    DB_Operations_Interface dao=new DB_Operations();
    @Path("add")
    @Produces(MediaType.TEXT_PLAIN)
    @Consumes(MediaType.APPLICATION_JSON)
    @POST
    @Override
    public String addStudent(String student)
    {
        Student s=gson.fromJson(student, Student.class);
        int result=dao.insertData(s);
        if(result>0)
        {
            return "Data Inserted Successfully";
        }else
        {
            return "Error In Adding Data Into Table";
        }
    }
    @Path("update")
    @Produces(MediaType.TEXT_PLAIN)
    @Consumes(MediaType.APPLICATION_JSON)
    @POST
    @Override
    public String updateStudent(String student)
    {
        Student s=gson.fromJson(student, Student.class);
```

```

int result=dao.updateData(s);
if(result>0)
{
    return "Data Updated Successfully";
}else
{
    return "Error In Updating Data ";
}
@Path("delete")
@Produces(MediaType.TEXT_PLAIN)
@GET
@Override
public String deleteStudent(@QueryParam("studentid") int id) {
    int result= dao.deleteData(id);
    if(result>0){
        return "Data Deleted Successfully";
    }else{
        return "Error In Deleting Data ";
    }
}
@Path("viewall")
@Produces(MediaType.APPLICATION_JSON)
@GET
@Override
public String viewAllStudents() {
    List<Student> list=dao.viewData();
    return gson.toJson(list);
}
}

```

## Client Creation



**Student.java**

```
package com.bean;
public class Student
{
    private int id;
    private String fname, lname, emailid;
    //getters/setters
```

**insertPage.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Insertion Page</h1><br><br>
<form action="MainServlet" method="post">
First Name :- <input type="text" name="fname" placeholder="First Name"><br>
Last Name :- <input type="text" name="lname" placeholder="Last Name"><br>
Email ID :- <input type="text" name="emailid" placeholder="Email ID"><br>
<input type="submit" value="Insert" name="action">
</form>
</body>
</html>
```

**profilePage.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h4 align="center">${error}</h4>
<br>
<table border=1 cellpadding="5" align="center">
<tr>
```

```
<th>First Name</th>
<th>Last Name</th>
<th>Email ID</th>
<th>Edit</th>
<th>Delete</th>
</tr>
```

## **profilePage.jsp**

```
<c:forEach var="stud" items="${students}">
<tr>
<td>${stud.fname}</td>
<td>${stud.lname}</td>
<td>${stud.emailid}</td>
<td><a href="updatePage.jsp?sid=${stud.id}">Edit</a></td>
<td><a href="MainServlet?sid=${stud.id}&action=delete">Delete</a></td>
</tr>
</c:forEach>
</table>
</body>
</html>
```

## **updatePage.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Updation Page</h1><br><br>
<form action="MainServlet" method="post">
First Name :- <input type="text" name="fname" placeholder="First Name"><br>
Last Name :- <input type="text" name="lname" placeholder="Last Name"><br>
Email ID :- <input type="text" name="emailid" placeholder="Email ID"><br>
<input type="text" name="id" value="${param.sid}">
<input type="submit" value="Update" name="action">
</form>
</body>
</html>
```

## **errorPage.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
${msg }
</body>
</html>
```

## MainServlet.java

```
package com.controller;
import java.io.IOException;
import java.util.List;
import javax.servlet.DispatcherType;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.Response;
import org.glassfish.jersey.client.ClientConfig;
import com.bean.Student;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
@WebServlet("/MainServlet")
public class MainServlet extends HttpServlet {
MainServlet.java
private static final long serialVersionUID = 1L;
String WebServiceURI="http://localhost:8080/RestfulCRUD_Server/";
ClientConfig clientConfig=null;
Client client=null;
WebTarget webTarget=null;
Gson gson=null;
public MainServlet() {
    super();
}
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
doProcess(request, response);
```

```
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
doProcess(request, response);
}
MainServlet.java
protected void doProcess(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
clientConfig = new ClientConfig();
client = ClientBuilder.newClient(clientConfig);
webTarget = client.target(WebServiceURI).path("rest");
gson= new Gson();
String action=request.getParameter("action");
if(action.equalsIgnoreCase("insert"))
{
Student student= new Student();
String fname=request.getParameter("fname");
String lname=request.getParameter("lname");
String emailid=request.getParameter("emailid");
student.setEmailid(emailid);
student.setFname(fname);
student.setLname(lname);
String s=gson.toJson(student);
MainServlet.java
Response res=webTarget.path("student").path("add").request().post(Entity.json(s));
String result=res.readEntity(String.class);System.out.println(result);
RequestDispatcher view=null;
if(res.getStatus()==200)
{
    showStudent(request,response);
}
else
{
    request.setAttribute("msg", result);
    view= request.getRequestDispatcher("errorPage.jsp");
    view.forward(request, response);
}
System.out.println(result);
}
MainServlet.java
else if(action.equalsIgnoreCase("update"))
{
    RequestDispatcher rd=null;
    String fname, lname, emailid;
    fname = request.getParameter("fname");
    lname = request.getParameter("lname");
}
```

```
emailid = request.getParameter("emailid");
int id = Integer.parseInt(request.getParameter("id"));

Student student = new Student();
student.setEmailid(emailid);
student.setFname(fname);
student.setLname(lname);
student.setId(id);
String stud=gson.toJson(student);
Response rs=webTarget.path("student").path("update").request().post(Entity.json(stud));
String result=rs.readEntity(String.class);
if(rs.getStatus()==200){
    showStudent(request,response);
}
else{
    request.setAttribute("msg", result);
    rd=request.getRequestDispatcher("errorPage.jsp");
    rd.forward(request, response);
}
System.out.println(result);
}
else if(action.equalsIgnoreCase("delete")){
    RequestDispatcher rd=null;
    String id = request.getParameter("sid");
    Response rs = webTarget.path("student").path("delete").queryParam("studentid",
    id).request().get();
    String result = rs.readEntity(String.class);
    if(rs.getStatus()==200)
    {
        showStudent(request,response);
    }
}
else
{
    request.setAttribute("msg", result);
    rd=request.getRequestDispatcher("errorPage.jsp");
    rd.forward(request, response);
}
System.out.println(result);
}
}

protected void showStudent(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
    Response rs=webTarget.path("student").path("viewall").request().get();
    String result=rs.readEntity(String.class);
if(rs.getStatus()==200)
```

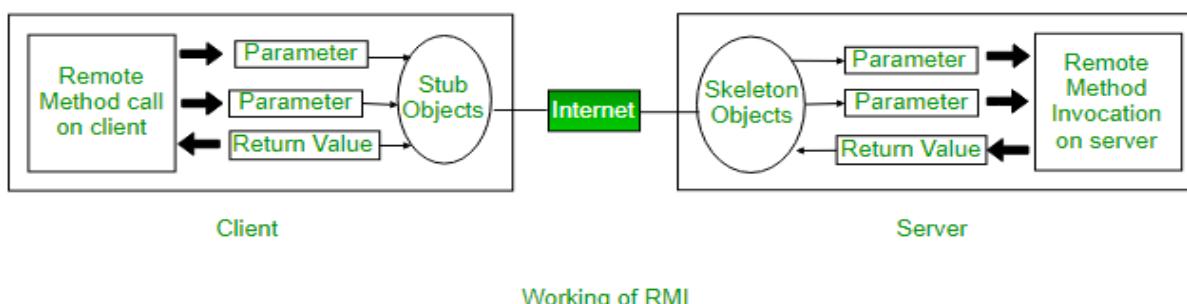
```

{
    List<Student> studentList=gson.fromJson(result,new
TypeToken<List<Student>>().get());
    request.setAttribute("students", studentList);
}
else
{
    request.setAttribute("error", "Error In Fetching All Students");
}
System.out.println(result);
RequestDispatcher view=request.getRequestDispatcher("profilePage.jsp");
view.forward(request, response);
}
}

```

## RMI

- **Remote Method Invocation (RMI)** is an API which allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine.
- Through RMI, object running in a JVM present on a computer (Client side) can invoke methods on an object present in another JVM (Server side).
- The RMI provides remote communication between the applications using two objects *stub* and *skeleton*.
- The communication between client and server is handled by using two intermediate objects:
  1. Stub object (on client side)
  2. Skeleton object (on server side).



## STUB

- The stub object on the client machine builds an information block and sends this information to the server.

The block consists of :

1. An identifier of the remote object to be used
2. Method name which is to be invoked
3. Parameters to the remote JVM

## SKELETON

- The skeleton is an object, acts as a gateway for the server side object.
- All the incoming requests are routed through it. When the skeleton receives the incoming request.

It does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

## Create RMI Interface Steps

1. Create the remote interface
2. Provide the implementation of the remote interface
3. Compile the implementation class and create the stub and skeleton objects using the rmic tool
4. Start the registry service by rmiregistry tool
5. Create and start the remote application
6. Create and start the client application
7. Provide the implementation of the remote interface
8. Compile the implementation class and create the stub and skeleton objects using the rmic tool
9. Start the registry service by rmiregistry tool
10. Create and start the remote application
11. Create and start the client application

## Create Remote Interface

```
import java.rmi.*;
public interface Adder extends Remote
{
    public int add(int x,int y) throws RemoteException;
}
```

## Implementation of Remote Interface

```
import java.rmi.*;
import java.rmi.server.*;
public class AdderRemote extends UnicastRemoteObject implements Adder
{
    AdderRemote() throws RemoteException
    {
        super();
    }
    public int add(int x,int y){return x+y;}
}
```

Create the stub and skeleton objects using the rmic tool.

## Create And Run the Server Application

```
import java.rmi.*;
import java.rmi.registry.*;
public class MyServer{
public static void main(String args[])
{
try{
    Adder stub=new AdderRemote();
    Naming.rebind("rmi://localhost:5000/sonoo",stub);
}
catch(Exception e)
{
    System.out.println(e);
}
}
```

## Create And Run the Client Application

```
import java.rmi.*;
public class MyClient{
public static void main(String args[]){
try{
    Adder stub=(Adder)Naming.lookup("rmi://localhost:5000/so
noo");
    System.out.println(stub.add(34,4));
}
catch(Exception e){}
}
}
```

The image contains two side-by-side screenshots of a Windows command prompt window. Both windows have the title bar 'C:\Windows\system32\cmd.exe - rmiregistry 5000' and show the Microsoft Windows [Version 6.1.7600] Copyright (c) 2009 Microsoft Corporation. All rights reserved.

**Top Window:** The user is in the directory 'D:\Sonoo\Programs\core\rmi\1'. They run the command 'rmic AdderRemote' which generates the stub and skeleton files for the 'Adder' interface.

```
C:\Users\Sonoo>d:
D:\>cd "Sonoo\Programs\core\rmi\1"
D:\Sonoo\Programs\core\rmi\1>javac *.java
D:\Sonoo\Programs\core\rmi\1>rmic AdderRemote
D:\Sonoo\Programs\core\rmi\1>rmiregistry 5000
```

**Bottom Window:** The user is also in the directory 'D:\Sonoo\Programs\core\rmi\1'. They run the command 'java MyServer' to start the server application.

```
C:\Users\Sonoo>d:
D:\>cd "Sonoo\Programs\core\rmi\1"
D:\Sonoo\Programs\core\rmi\1>java MyServer
```



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Sonoo>d:
D:\>cd "Sonoo Programs\core\rmi\1"
D:\Sonoo Programs\core\rmi\1>java MyClient
38
D:\Sonoo Programs\core\rmi\1>
```

Use 3 Different Command prompt for execution as given above

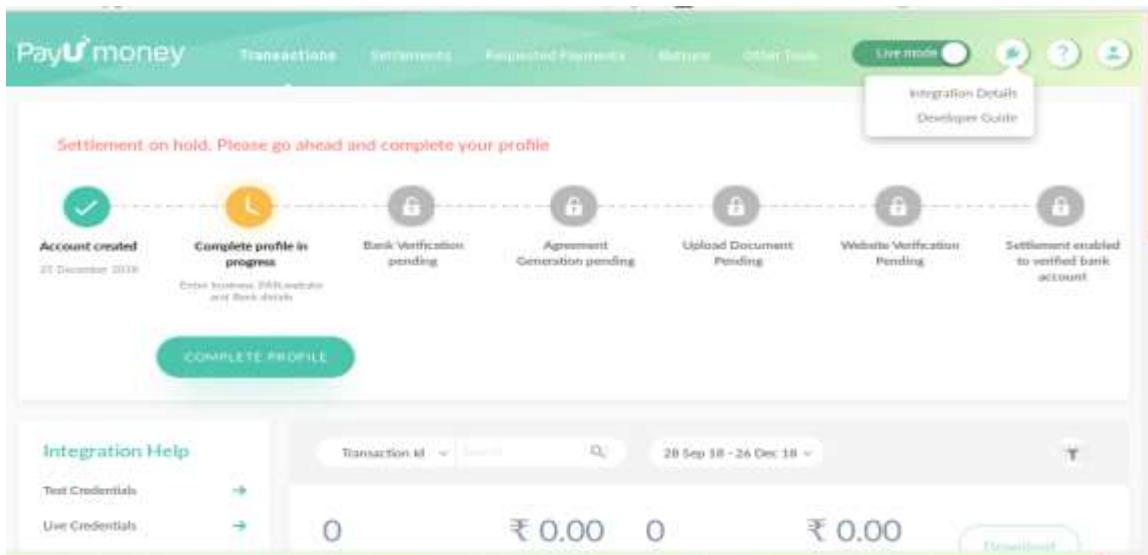
**Note:** please don't close any one of them otherwise it will generate error

## PayUMoney Integration

- First Go to this url for a payumoney signup for a merchant user with a proper name and mobile number.
- <https://www.payumoney.com/>



After Signup click on a Integration



You need to note down Merchant Key and Merchant Salt

The screenshot shows the PayUmoney Integration Details page. At the top, it says "Everything you need for Integration!". Below that, it says "Here are your Integration Credentials". It is divided into two sections: "Live Credentials" and "Test Credentials".

Live Credentials	Test Credentials
Merchant Key bh0oeJBh	Test Key bh0oeJBh
Merchant Salt ri4yv1skBR	Test Salt ri4yv1skBR
Auth Header you can click here to <a href="#">generate header</a>	Test Auth Header you can click here to <a href="#">generate header</a>

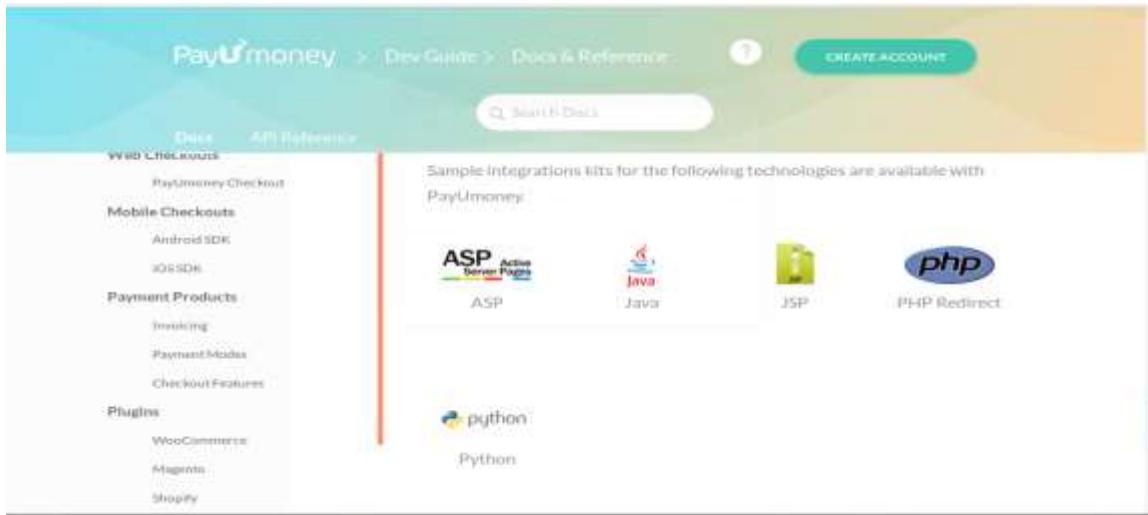
You will also be given Test credentials.

Then choose how you want to integrate(Java) and proceed

The screenshot shows the PayUmoney Integration Details page. It asks if the user's business is set up on leading Shopping Carts like Shopify, OpenCart, etc. It says "Just enter few details for quick setup of PayUmoney on all leading shopping carts like Shopify, OpenCart, etc. No coding required." There is a "Proceed" button.

Below that, it asks if the user is building a custom website on Java, PHP, etc. It says "Choose this to integrate payment gateway easily on your website on JAVA, PHP, Python, Node.js. Use sample integration kits here." There is another "Proceed" button.

Then from Integration kits available option click on Java that will download PUM-Java-Master.zip



Open pom.xml file.

Also open pom.xml file from downloaded kit and copy from <dependencies> to </dependencies> and paste to your projects pom.xml file before <build> tag and save it.

That will download all the necessary library from the internet

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>PayUMoney</groupId>
  <artifactId>PayUMoney</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.3.5</version>
      <type>jar</type>
    </dependency>
    <dependency>
      <groupId>commons-httpclient</groupId>
      <artifactId>commons-httpclient</artifactId>
      <version>3.1</version>
      <type>jar</type>
    </dependency>
    <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.2.3</version>
    </dependency>
  </dependencies>

```

```

<type>jar</type>
</dependency>
<dependency>
    <groupId>javax</groupId>
    <artifactId>javaee-web-api</artifactId>
    <version>6.0</version>
    <scope>provided</scope>
</dependency>
</dependencies>
<build>
    <sourceDirectory>src</sourceDirectory>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.0</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.2.1</version>
            <configuration>
                <warSourceDirectory>WebContent</warSourceDirectory>
            </configuration>
        </plugin>
    </plugins>
</build>
</project>
```

Now open index.html and replace key with your payumoney merchant key that you get from payumoney site.

```

<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
    <head>
    </head>
    <body>

        <h1>PayUForm </h1>

        <form action="myServlet" name="payuform" method=POST>
            <input type="hidden" name="key" value="IqrM8G13" />
            <input type="hidden" name="hash_string" value="" />
            <input type="hidden" name="hash" />

            <input type="hidden" name="txnid"/>
```

```

<table>
    <tr>
        <td><b>Mandatory Parameters</b></td>
    </tr>
    <tr>
        <td>Amount: </td>
        <td><input name="amount" /></td>
        <td>First Name: </td>
        <td><input name="firstname" id="firstname" /></td>
    </tr>
    <tr>

```

Then open JavaIntegrationKit.java and put merchant key and salt key as shown in the screen.

```

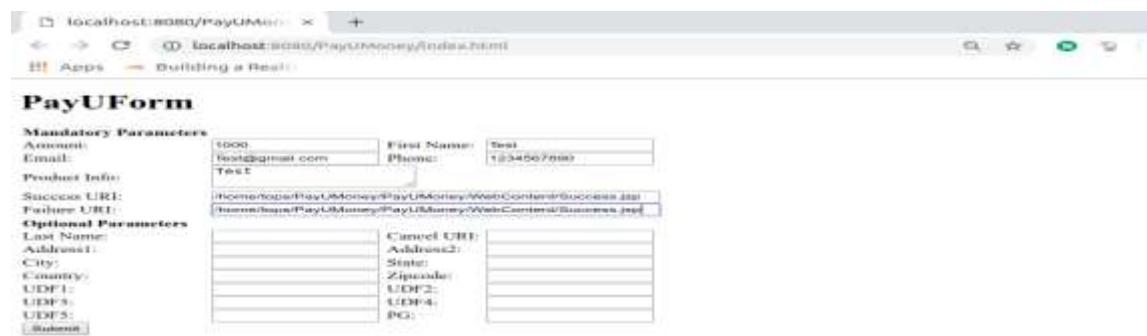
protected Map<String, String> hashCalMethod(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    String key = "IqrM8G13";
    String salt = "8XwJFKMnPm";
    String action1 = "https://sandboxsecure.payu.in/_payment";
    String base_url = "https://sandboxsecure.payu.in";
    error = 0;
    String hashString = "";
    Enumeration paramNames = request.getParameterNames();
    Map<String, String> params = new HashMap<String, String>();
    Map<String, String> urlParams = new HashMap<String, String>();
    while (paramNames.hasMoreElements()) {
        String paramName = (String) paramNames.nextElement();
        String paramValue = request.getParameter(paramName);
        params.put(paramName, paramValue);
    }
    String txnid = "";
    if (empty(params.get("txnid"))) {
        Random rand = new Random();
        String rndm = Integer.toString(rand.nextInt()) +
(System.currentTimeMillis() / 1000L);
        txnid = rndm;
        params.remove("txnid");
        params.put("txnid", txnid);
        txnid = hashCal("SHA-256", rndm).substring(0, 20);
    } else {
        txnid = params.get("txnid");
    }
}

```

Now create success.jsp in WebApp and just print Hello for the success url.  
Note down the path of success.jsp that we need while running the app.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Success</h1>
</body>
</html>
```

Now run index.html, fill necessary fields and in fail and success url part put success.jsp's path for temporary output.



You will be given testing credentials for education purpose use this to make payment and you will be done.

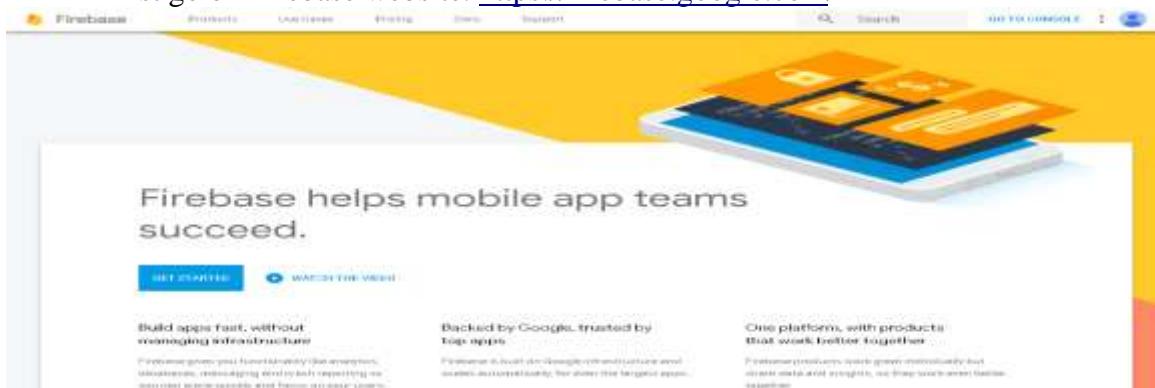


## Firebase

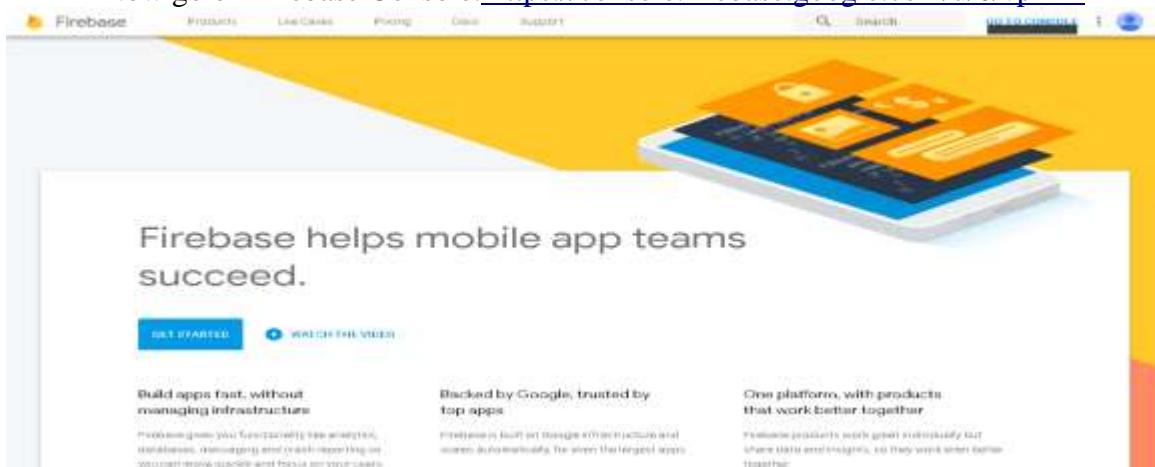
Firebase evolved from Envolve, a prior startup founded by James Tamplin and Andrew Lee in 2011. Envolve provided developers an API that enables the integration of online chat functionality into their websites. After releasing the chat service, Tamplin and Lee found that it was being used to pass application data that weren't chat messages. Developers were using Envolve to sync application data such as game state in real time across their users. Tamplin and Lee decided to separate the chat system and the real-time architecture that powered it. They founded Firebase as a separate company in April 2012.

Firebase Inc. raised seed funding in May 2012. The company further raised Series A funding in June 2013. In October 2014, Firebase was acquired by Google. In October 2015, Google acquired Divshot to merge it with the Firebase team. Since the acquisition, Firebase has grown inside Google and expanded their services to become a unified platform for mobile developers. Firebase now integrates with various other Google services to offer broader products and scale for developers. In January 2017, Google acquired Fabric and Crashlytics from Twitter to join those services to the Firebase team. Firebase launched Cloud Firestore, a Document Database, in October 2017

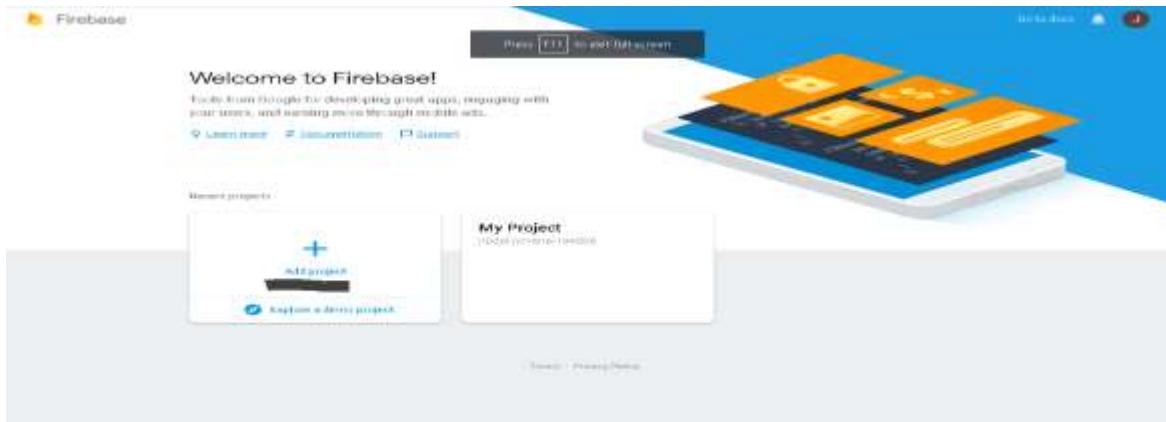
- First go on Firebase website. <https://firebase.google.com/>



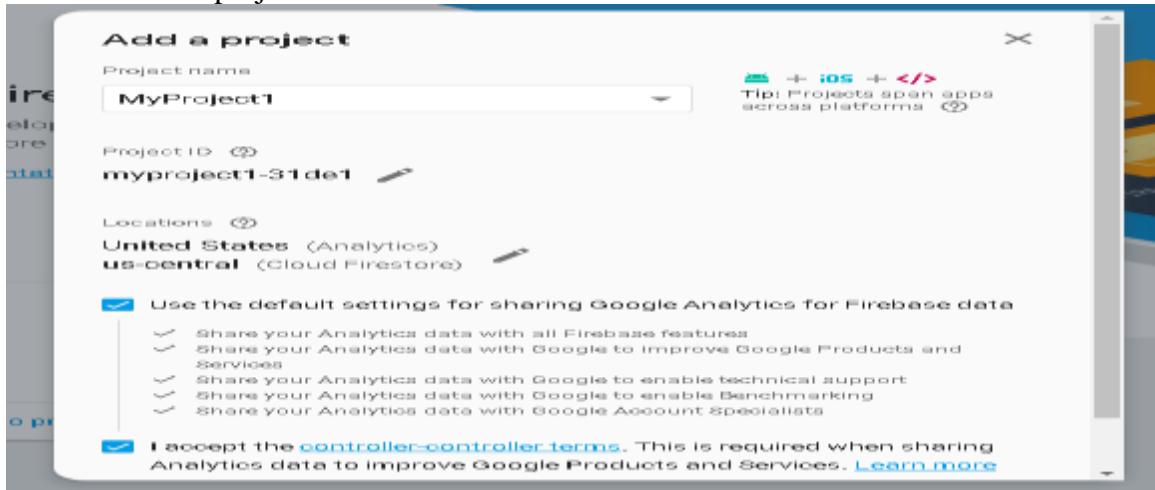
- Now go on Firebase Console. <https://console.firebaseio.google.com/u/0/?pli=1>



- Now add one project on console.

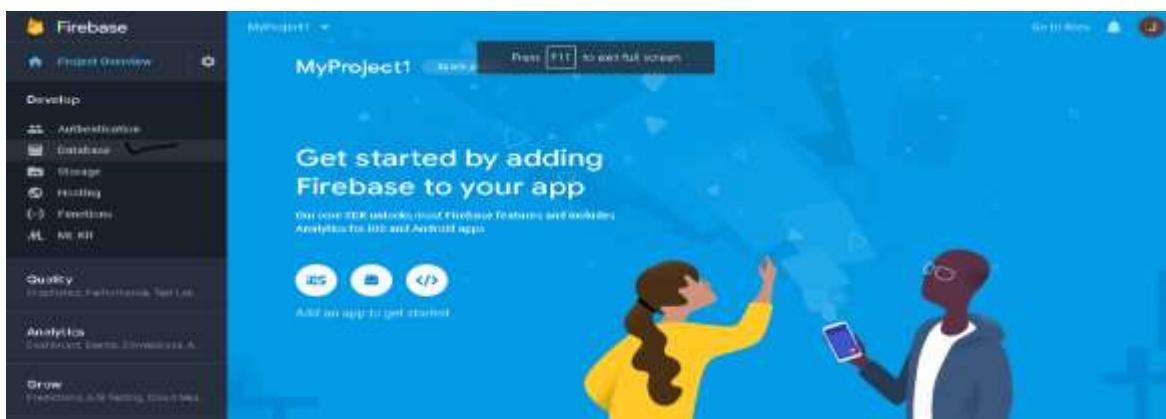


Now create one project

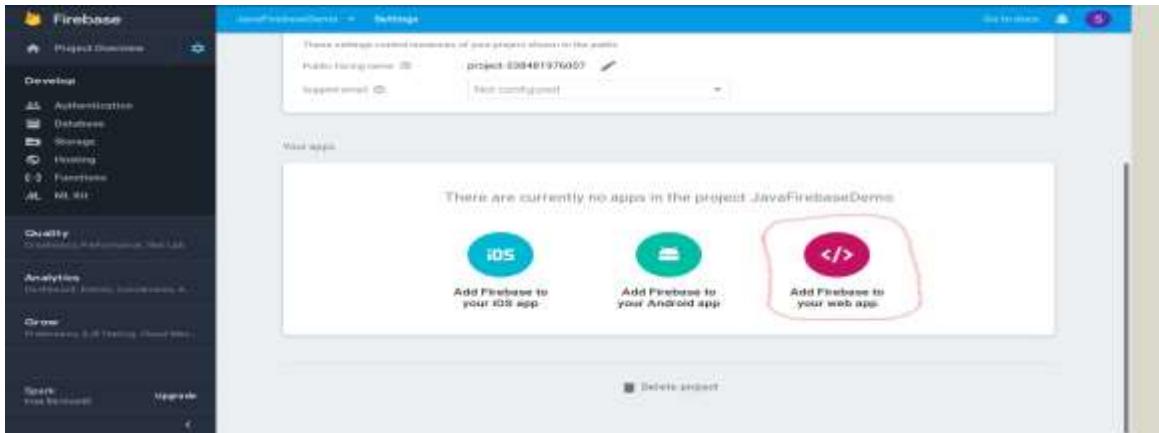


Now click on Developer Tab. In Developer tab click on Database .

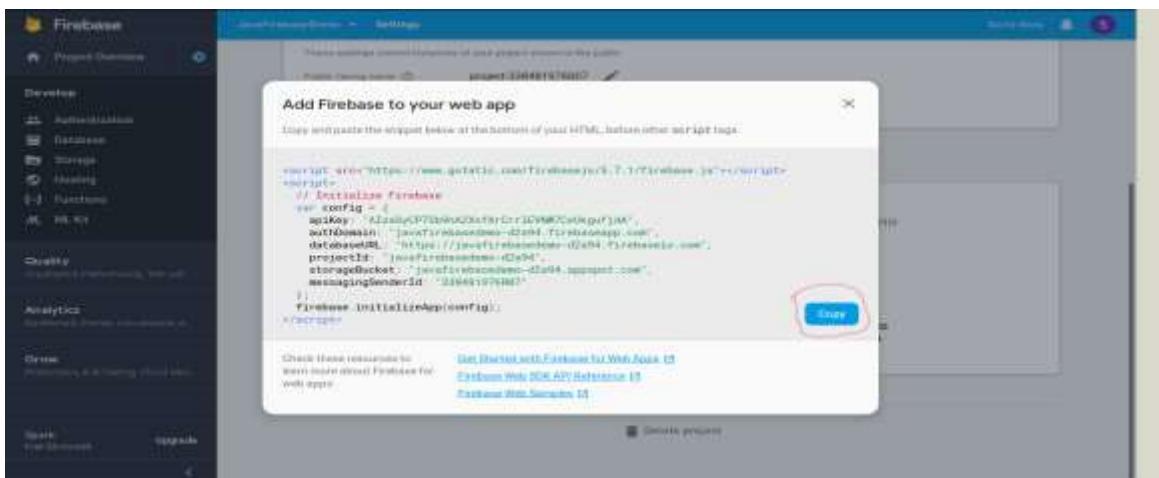
<https://console.firebaseio.google.com/u/0/project/myproject1-34bf5/overview>



- Now you want configuration of Firebase for your project.



- Go to project setting and copy the configuration details.



- Now Create Dynamic Web Project in Eclipse.
- Create one JSP file in Web Content.

Now add Firebase Configuration details between the Script Tag<script></script>.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>

<script src="https://www.gstatic.com/firebasejs/5.5.9.firebaseio.js"></script>
<script>
// Initialize Firebase
var config = {
  apiKey: "AIzaSyCP7Sb9nA2XsfArCrrlEVNW7CeUkgufjdA",
  authDomain: "javafirebasedemo-d2a94.firebaseio.com",
  databaseURL: "https://javafirebasedemo-d2a94.firebaseio.com",
  projectId: "javafirebasedemo-d2a94",
  storageBucket: "javafirebasedemo-d2a94.appspot.com",
  messagingSenderId: "384691976887"
}>
```

```
messagingSenderId: "338481976007"
};

firebase.initializeApp(config);
</script>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
```

Now create one table in Body Tag.

```
<body>

<table>
<tr>
<td>Id: </td>
<td><input type="text" name="id" id="user_id" /></td>
</tr>
<tr>
<td>User Name: </td>
<td><input type="text" name="user_name" id="user_name" /></td>
</tr>
<tr>
<td colspan="2">
<input type="button" value="Save" onclick="save_user();"/>
<input type="button" value="Update" onclick="update_user();"/>
<input type="button" value="Delete" onclick="delete_user();"/>
</td>
</tr>
</table>
```

<h3>Users List</h3>

```
<table id="tbl_users_list" border="1">
<tr>
<td>#ID</td>
<td>NAME</td>
</tr>
</table>
```

```
<script>
```

```
var tblUsers = document.getElementById('tbl_users_list');
var databaseRef = firebase.database().ref('users/');
var rowIndex = 1;

databaseRef.once('value', function(snapshot) {
  snapshot.forEach(function(childSnapshot) {
    var childKey = childSnapshot.key;
    var childData = childSnapshot.val();

    var row = tblUsers.insertRow(rowIndex);
    var cellId = row.insertCell(0);
    var cellName = row.insertCell(1);
    cellId.appendChild(document.createTextNode(childKey));
    cellName.appendChild(document.createTextNode(childData.user_name));
```

```
RowIndex = rowIndex + 1;
});

});

function save_user(){
var user_name = document.getElementById('user_name').value;

var uid = firebase.database().ref().child('users').push().key;

var data = {
  user_id: uid,
  user_name: user_name
}

var updates = {};
updates['/users/' + uid] = data;
firebase.database().ref().update(updates);

alert('The user is created successfully!');
reload_page();
}

function update_user(){
var user_name = document.getElementById('user_name').value;
var user_id = document.getElementById('user_id').value;

var data = {
  user_id: user_id,
  user_name: user_name
}

var updates = {};
updates['/users/' + user_id] = data;
firebase.database().ref().update(updates);

alert('The user is updated successfully!');

reload_page();
}

function delete_user(){
var user_id = document.getElementById('user_id').value;

firebase.database().ref().child('/users/' + user_id).remove();
alert('The user is deleted successfully!');
reload_page();
}

function reload_page(){
  window.location.reload();
}

</script>

</body>
```

</html>

Output :



**Id:**

**User Name:**

**Save**

**Update**

**Delete**

## Users List

#ID	NAME
-LRuzp9wLw3bmFnKEap-	SUNITJHA
-LSAGQiHkmvblWqlt4k4	TanyaJha
-LSDfVFWV2OgCD-f1053	margi

## Hibernate

### Introduction

- Hibernate was started in 2001 by Gavin King as an alternative to using EJB2-style entity beans.
- Its mission back then was to simply offer better persistence capabilities than offered by EJB2 by simplifying the complexities and allowing for missing features.
- Early in 2003, the Hibernate development team began Hibernate2 releases which offered many significant improvements over the first release.

- JBoss, Inc. (now part of Red Hat) later hired the lead Hibernate developers and worked with them in supporting Hibernate. Hibernate is part of JBoss (a division of Red Hat) Enterprise Middleware System (JEMS) suite of products.
- Hibernate is an Object-relational mapping (ORM) tool. Object-relational mapping or ORM is a programming method for mapping the objects to the relational model where entities/classes are mapped to tables, instances are mapped to rows and attributes of instances are mapped to columns of table.

## What is Hibernate?

- Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.
- Hibernate facilitates the storage and retrieval of Java domain objects via Object/Relational Mapping.

## Task of Hibernate

- Map Java class to database tables & vice versa
- Data query and retrieval facility
- Generate the SQL query based on the underline DB. and attempts to relieve the developer from manual result set handling and object conversion.
- Make application portable to all relational DB.
- Enhance performance by providing the different levels of cache(First, Second and Query level).

## What is HQL(Hibernate Query Language)

- It is SQL inspired language provided by hibernate.
- Developer can write SQL like queries to work with data objects.

## Supported Databases:

- Hibernate supports almost all the major RDBMS.
- Following is list of few of the database engines supported by Hibernate.
  - HSQL Database Engine
  - DB2/NT
  - MySQL
  - PostgreSQL
  - FrontBase
  - Oracle
  - Microsoft SQL Server Database
  - Sybase SQL Server
  - Informix Dynamic Server

## Pros and Cons of JDBC

Pros	Cons
Clean and simple SQL processing	Complex if it is used in large projects
Good performance with large data	Large programming overhead
Very good for small applications	No encapsulation
Simple syntax so easy to learn	Hard to implement MVC concept Query is DBMS specific

Why Object Relational Mapping (ORM)?

- When we work with an object-oriented systems, there's a mismatch between the object model and the relational database.
- RDBMSs represent data in a tabular format whereas object-oriented languages, such as Java or C# represent it as an interconnected graph of objects.
- Consider the following Java Class with proper constructors and associated public function:

### **Employee.java**

```
public class Employee {
    private int id;
    private String first_name;
    private String last_name;
    private int salary;
```

Consider above objects need to be stored and retrieved into the following RDBMS table:

```
create table EMPLOYEE (
    id INT NOT NULL auto_increment,
    first_name VARCHAR(20) default NULL,
    last_name VARCHAR(20) default NULL,
    salary INT default NULL,
```

First problem, what if we need to modify the design of our database after having developed few pages or our application? Second, Loading and storing objects in a relational database exposes us to the following five mismatch problems.

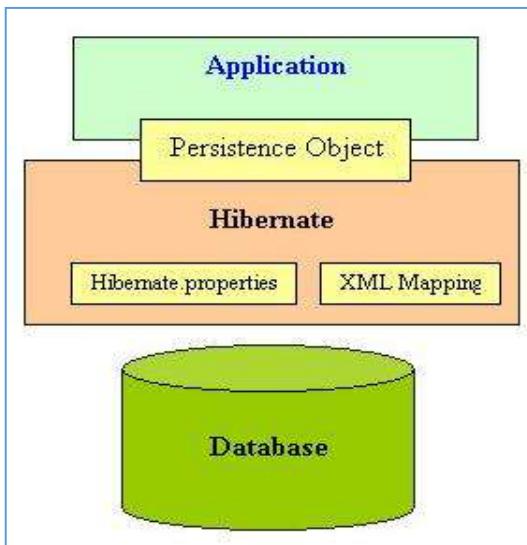
<b>Mismatch</b>	<b>Description</b>
<i>Granularity</i>	<i>Sometimes you will have an object model which has more classes than the number of corresponding tables in the database.</i>
<i>Inheritance</i>	<i>RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.</i>
<i>Identity</i>	<i>A RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (<math>a==b</math>) and object equality (<math>a.equals(b)</math>).</i>
<i>Associations</i>	<i>Object-oriented languages represent associations using object references where as am RDBMS represents an association as a foreign key column.</i>
<i>Navigation</i>	<i>The ways you access objects in Java and in a RDBMS are fundamentally different.</i>

### **What is ORM?**

- The **Object-Relational Mapping (ORM)** is the solution to handle all the above impedance mismatches.
- ORM stands for **Object-Relational Mapping (ORM)** is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.

- An ORM system has following advantages over plain JDBC

S.N.	Advantages
1	<i>Lets business code access objects rather than DB tables.</i>
2	<i>Hides details of SQL queries from OO logic.</i>



3	<i>Based on JDBC 'under the hood'</i>
4	<i>No need to deal with the database implementation.</i>
5	<i>Entities based on business concepts rather than database structure.</i>
6	<i>Transaction management and automatic key generation.</i>
7	<i>Fast development of application.</i>

### Advantages of ORM

- Lets business code access objects rather than DB tables.
- Hides details of SQL queries from OO logic.
- Based on JDBC 'under the hood'
- No need to deal with the database implementation.
- Entities based on business concepts rather than database structure.
- Transaction management and automatic key generation.
- Fast development of application.

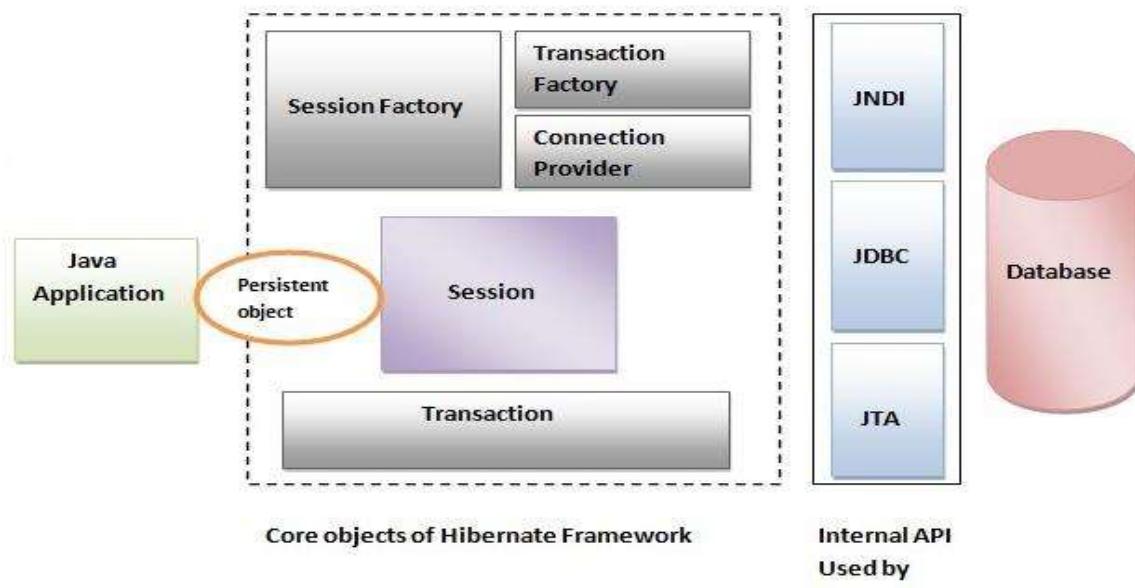
### Hibernate is a persistence framework

**Persistence** is a process of storing the data to some permanent medium and retrieving it back at any point of time even after the application that had created the data ended.

### Hibernate Architecture

- The above diagram shows minimal architecture of Hibernate. It creates a layer between Database and the Application.

- It loads the configuration details like Database connection string, entity classes, mappings etc.
- Diagram shows that Hibernate is using the database and configuration data to provide persistence services (and persistent objects) to the application.
- To use Hibernate, it is required to create Java classes that represents the table in the database and then map the instance variable in the class with the columns in the database.
- Then Hibernate can be used to perform operations on the database like select, insert, update and delete the records in the table. Hibernate automatically creates the query to perform these operations.
- Hibernate creates persistent objects which synchronize data between application and database.
- In order to persist data to a database, Hibernate create an instance of entity class (Java class mapped with database table).
- This object is called Transient object as they are not yet associated with the session or not yet persisted to a database.
- To persist the object to database, the instance of **SessionFactory** interface is created. SessionFactory is a singleton instance which implements Factory design pattern. SessionFactory loads **hibernate.cfg.xml** file (Hibernate configuration file).
- More details in following section) and with the help of TransactionFactory and ConnectionProvider implements all the configuration settings on a database.
- Each database connection in Hibernate is created by creating an instance of Session interface. Session represents a single connection with database.
- Session objects are created from SessionFactory object.



- ***SessionFactory (org.hibernate.SessionFactory)***

- A thread-safe, immutable cache of compiled mappings for a single database.
- The SessionFactory is a factory of session and client of ConnectionProvider.
- It is factory of JDBC connections. It abstracts the application from DriverManager or DataSource.
- It holds second level cache(optional) of data.
- The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

## Configuration

- An instance of Configuration allows the application to specify properties and mapping documents to be used when creating a SessionFactory.
- Usually an application will create a single Configuration, build a single instance of SessionFactory and then instantiate Sessions in threads servicing client requests.
- The Configuration is meant only as an initialization-time object. SessionFactory's are immutable and do not retain any association back to the Configuration.
- A new Configuration will use the properties specified in hibernate.properties by default.

## ***Session (org.hibernate.Session)***

- A single-threaded, short-lived object representing a conversation between the application and the persistent store.
- The session object provides an interface between the application and data stored in the database.
- It is a short-lived object and wraps the JDBC connection.
- It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data.
- The org.hibernate.Session interface provides method to insert(save), update and delete the object.
- It also provides factory method for transaction, Query and Criteria.

## ***Transaction (org.hibernate.Transaction)***

- (Optional) A single-threaded, short-lived object used by the application to specify atomic units of work. It abstracts the application from the underlying JDBC, JTA or CORBA transaction. A org.hibernate.Session might span several org.hibernate.Transactions in some cases.
- The transaction object specifies the atomic unit of work. It is optional.
- The org.hibernate.Transaction interface provides methods for transaction management.

## ***ConnectionProvider (org.hibernate.connection.ConnectionProvider)***

- (Optional) A factory for, and pool of, JDBC connections. It abstracts the application from underlying javax.sql.DataSource or java.sql.DriverManager.

## ***TransactionFactory (org.hibernate.TransactionFactory)***

- (Optional) A factory for org.hibernate.Transaction instances. It is not exposed to the application, but it can be extended and/or implemented by the developer.

## ***Query***

- An object-oriented representation of a Hibernate query.
- A Query instance is obtained by calling **Session.createQuery()**.
- This interface exposes some extra functionality beyond that provided by **Session.iterate()** and **Session.find()**
- a particular page of the result set may be selected by calling setMaxResults(), setFirstResult()
- named query parameters may be used
- the results may be returned as an instance of ScrollableResults
- Queries are executed by calling **list()**, **scroll()** or **iterate()**. A query may be re-executed by subsequent invocations.
- Its lifespan is, however, bounded by the lifespan of the Session that created it.

## ***Criteria***

- Criteria is a simplified API for retrieving entities by composing Criterion objects.
- Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables. One of the methods is Criteria API which allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.

- The Hibernate **Session** interface provides **createCriteria()** method which can be used to create a **Criteria** object that returns instances of the persistence object's class when your application executes a criteria query.

## Types

<b>Mapping type</b>	<b>Java type</b>	<b>ANSI SQL Type</b>
<i>Integer</i>	<i>int or java.lang.Integer</i>	<i>INTEGER</i>
<i>Long</i>	<i>long or java.lang.Long</i>	<i>BIGINT</i>
<i>Short</i>	<i>short or java.lang.Short</i>	<i>SMALLINT</i>
<i>Float</i>	<i>float or java.lang.Float</i>	<i>FLOAT</i>
<i>Double</i>	<i>double or java.lang.Double</i>	<i>DOUBLE</i>
<i>big_decimal</i>	<i>java.math.BigDecimal</i>	<i>NUMERIC</i>
<i>Character</i>	<i>java.lang.String</i>	<i>CHAR(1)</i>
<i>String</i>	<i>java.lang.String</i>	<i>VARCHAR</i>
<i>Text</i>	<i>java.lang.String</i>	<i>CLOB</i>
<i>Byte</i>	<i>byte or java.lang.Byte</i>	<i>TINYINT</i>
<i>boolean</i>	<i>boolean or java.lang.Boolean</i>	<i>BIT</i>
<i>yes/no</i>	<i>boolean or java.lang.Boolean</i>	<i>CHAR(1) ('Y' or 'N')</i>
<i>true/false</i>	<i>boolean or java.lang.Boolean</i>	<i>CHAR(1) ('T' or 'F')</i>
<i>Date</i>	<i>java.util.Date or java.sql.Date</i>	<i>DATE</i>
<i>Time</i>	<i>java.util.Date or java.sql.Time</i>	<i>TIME</i>
<i>Timestamp</i>	<i>java.util.Date or java.sql.Timestamp</i>	<i>TIMESTAMP</i>
<i>Calendar</i>	<i>java.util.Calendar</i>	<i>TIMESTAMP</i>
<i>calendar_date</i>	<i>java.util.Calendar</i>	<i>DATE</i>

## Hibernate Configuration

- Hibernate requires to know in advance where to find the mapping information that defines how your Java classes relate to the database tables. Hibernate also requires a set of configuration settings related to database and other related parameters. All such information is usually supplied

as a standard Java properties file called **hibernate.properties**, or as an XML file named **hibernate.cfg.xml**.

- consider XML formatted file **hibernate.cfg.xml** to specify required Hibernate properties in my examples. Most of the properties take their default values and it is not required to specify them in the property file unless it is really required. This file is kept in the root directory of your application's classpath.

## Hibernate Properties

Hibernate Properties	Description
<b>hibernate.dialect</b>	This property makes Hibernate generate the appropriate SQL for the chosen database.
<b>hibernate.connection.driver_class</b>	The JDBC driver class
<b>hibernate.connection.url</b>	The JDBC URL to the database instance.
<b>hibernate.connection.username</b>	The database username.
<b>hibernate.connection.password</b>	The database password.
<b>hibernate.hbm2ddl.auto</b>	It validate   update   create   create-drop the tables
<b>Hibernate.show_sql</b>	Write all SQL statements to console.
<b>hibernate.connection.pool_size</b>	Limits the number of connections waiting in the Hibernate database connection pool.

## Hibernate Properties with MySQL

Hibernate Properties	Description
<b>hibernate.dialect</b>	org.hibernate.dialect.MySQLDialect
<b>hibernate.connection.driver_class</b>	com.mysql.jdbc.Driver
<b>hibernate.connection.url</b>	jdbc:mysql://localhost:3306/hibernate
<b>hibernate.connection.username</b>	root
<b>hibernate.connection.password</b>	root   (nothing)
<b>hibernate.hbm2ddl.auto</b>	validate   update   create   create-drop
<b>Hibernate.show_sql</b>	true   false
<b>hibernate.connection.pool_size</b>	5, 10, 20, 40 (Number format)

## Hibernate Configuration Mapping

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<!-- Hibernate Configuration Mapping with MySQL -->
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernate</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password"></property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>

```

## Hibernate Bean Mapping

### User.java

```
public class User {  
    private int userid;  
    private String username;  
    private String password;  
    //Generate getter and setter of the fields}
```

### User.hbm.xml

```
<?xml version="1.0"?>  
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
<class name="com.bean.User" table="USER">  
<id name="userid" type="int">  
<column name="USERID" />  
<generator class="increment" /></id>  
<property name="username" type="java.lang.String">  
<column name="USERNAME" /></property>  
<property name="password" type="java.lang.String">  
<column name="PASSWORD" /></property>  
</class></hibernate-mapping>
```

## Generator

- <generator /> is one of main element we are using in the hibernate framework [in the mapping file], let us see the concept behind this generators.
- Up to now in our hibernate mapping file, we used to write <generator /> in the id element scope, actually this is default like whether you write this assigned generator or not hibernate will takes automatically
- In fact this assigned means hibernate will understand that, while saving any object hibernate is not responsible to create any primary key value for the current inserting object, user has to take the response.
- The thing is, while saving an object into the database, the generator informs to the hibernate that, how the primary key value for the new record is going to generate
- hibernate using different primary key generator algorithms, for each algorithm internally a class is created by hibernate for its implementation.
- hibernate provided different primary key generator classes and all these classes are implemented from **org.hibernate.id.IdentifierGeneratar** Interface while configuring <generator /> element in mapping file, we need to pass parameters if that generator class need any parameters, actually one sub element of <generator /> element is <param />, will talk more about this

## Example of :

```
<generator class="">  
    <param name=""> value </param>
```

## List of Generator

- Assigned - default generator by default 0(Zero).
- Increment - assign primary key and increment by default
- Sequence - create data as sequentially in database
- identity – support to id column
- hilo – high and low algorithm
- native – uses identity, sequence, hilo
- foreign – associated the object to the id
- uuid.hex – 128-bit UUID algorithm in hexadecimal
- uuid.string – 128-bit UUID algorithm in string

## How to create Hibernate Application?

- Requirements for creating Hibernate Application
  - Hibernate Jar libraries
    - antlr
    - Cglib
    - Log4j
    - Commons-io, commons-logging, commons-collection
    - Sl4j – api and sl4j-log4j
    - Dom4j
    - Hibernate-core
  - Hibernate configuration file (hibernate.cfg.xml)
  - A POJO class (bean.java)
  - A hibernate-mapping file (bean.hbm.xml)
  - A Main class, who have insert, update, delete, select methods

## Hibernate Example

### Employee.java

```
public class Employee {  
    private int id;  
    private String firstName;  
    private String lastName;  
    private int salary;  
    //Generate getter and setter field with default and parametrize constructor }
```

### Employee.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<!DOCTYPE hibernate-mapping PUBLIC  
"-//Hibernate/Hibernate Mapping DTD//EN"  
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
<class name="Employee" table="EMPLOYEE">  
<meta attribute="class-description">  
    This class contains the employee detail. </meta>  
<id name="id" type="int" column="id">  
<generator class="increment"/>  
</id>  
<property name="firstName" column="first_name" type="string"/>  
<property name="lastName" column="last_name" type="string"/> </class> </hibernate-mapping>
```

**ManageEmployee.java**

```
Public class ManageEmployee {  
    public static void main(String[] args) {  
        ManageEmployee ME = new ManageEmployee();  
        /* Add few employee records in database */  
        Integer empID1 = ME.addEmployee("Vishal", "Ali", 1000);  
        Integer empID2 = ME.addEmployee("Ketan", "Shah", 500000);  
        Integer empID3 = ME.addEmployee("Cartton", "Paul", 10000);  
        /* List down all the employees */  
        ME.listEmployees();  
        /* Update employee's records */  
        ME.updateEmployee(empID1, 5000);  
        /* Delete an employee from the database */  
        ME.deleteEmployee(empID2);  
        /* List down new list of the employees */  
        ME.listEmployees(); } }  
addEmployee Method/* Method to CREATE an employee in the database */  
public Integer addEmployee(String fname, String lname, int salary){  
    Session session = factory.openSession();  
    Transaction tx = null;  
    Integer employeeID = null;  
    try{ tx = session.beginTransaction();  
        Employee employee = new Employee(fname, lname, salary);  
        employeeID = (Integer) session.save(employee);  
        tx.commit();  
    }catch (HibernateException e) {  
        if (tx!=null) tx.rollback(); e.printStackTrace();  
    }finally {session.close(); }  
    return employeeID; }  
listEmployees Method/* Method to READ all the employees */  
public void listEmployees( ){  
    Session session = factory.openSession(); Transaction tx = null;  
    try{ tx = session.beginTransaction();  
        List employees = session.createQuery("FROM Employee").list();  
        for (Iterator iterator =employees.iterator(); iterator.hasNext();){  
            Employee employee = (Employee) iterator.next();  
            System.out.print("First Name: " + employee.getFirstName());  
            System.out.print("Last Name: " + employee.getLastName());  
            System.out.println("Salary: " + employee.getSalary()); }  
        tx.commit();  
    }catch (HibernateException e) {  
        if (tx!=null) tx.rollback();  
        e.printStackTrace();  
    }finally {session.close(); }  
} } updateEmployee Method/* Method to UPDATE salary for an employee */  
public void updateEmployee(Integer EmployeeID, int salary ){  
    Session session = factory.openSession(); Transaction tx = null;  
    try{ tx = session.beginTransaction();  
        Employee employee = (Employee)session.get(Employee.class, EmployeeID);  
        employee.setSalary( salary );  
        session.update(employee); tx.commit();  
    }catch (HibernateException e) {  
        if (tx!=null) tx.rollback();  
        e.printStackTrace();  
    }finally {session.close(); }  
}
```

```
deleteEmployee Method /* Method to DELETE an employee from the records */
public void deleteEmployee(Integer EmployeeID){
    Session session = factory.openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        Employee employee = (Employee)session.get(Employee.class, EmployeeID);
        session.delete(employee);
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
```

## ORM

### Mappings Association

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the cardinality of the relationship between the objects can be expressed. An association mapping can be unidirectional as well as bidirectional.

### Collection Mapping

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java. Hibernate can persist instances of **java.util.Map**, **java.util.Set**, **java.util.SortedMap**, **java.util.SortedSet**, **java.util.List**, and any **array** of persistent entities or values.

<b>Classes</b>	<b>Description</b>
<i>Java.util.Set</i> <i>Java.util.SortedSet</i>	<i>This is mapped with a &lt;set&gt; element and initialized with java.util.HashSet</i>
<i>Java.util.List</i>	<i>This is mapped with a &lt;list&gt; element and initialized with java.util.ArrayList</i>
<i>Java.util.Collection</i>	<i>This is mapped with a &lt;bag&gt; or &lt;ibag&gt; element and initialized with java.util.ArrayList</i>
<i>Java.util.Map</i> <i>Java.util.SortedMap</i>	<i>This is mapped with a &lt;map&gt; element and initialized with java.util.HashMap</i>

### Class Tag Property

```
<class name="ClassName" table="tableName" />
```

<b>Property</b>	<b>Description</b>
name (optional)	the name of the property.
table (optional - defaults to the unqualified class name)	the name of its database table.

## Id Tag property

```
<id name="propertyName" type="typename" column="column_name" >
    <generator class="generatorClass"/>
</id>
```

<b>Property</b>	<b>Description</b>
name (optional)	the name of the identifier property.
type (optional)	a name that indicates the Hibernate type.
column (optional - defaults to the property name)	the name of the primary key column.
generator	<generator> child element names a Java class used to generate unique identifiers for instances of the persistent class.

## One-to-One Tag Properly

```
<one-to-one name="propertyName" class="ClassName" cascade="cascade_style"
constrained="true|false" fetch="join|select" property-
ref="propertyNameFromAssociatedClass" access="field|property|ClassName"
```

<b>Property</b>	<b>Description</b>
name	the name of the property.
class (optional - defaults to the property type determined by reflection)	the name of the associated class.
cascade (optional)	specifies which operations should be cascaded from the parent object to the associated object.
constrained (optional)	specifies that a foreign key constraint on the primary key of the mapped table and references the table of the associated class. This option affects the order in which save() and delete() are cascaded, and determines whether the association can be proxied. It is also used by the schema export tool.
fetch (optional - defaults to select)	chooses between outer-join fetching or sequential select fetching.
property-ref (optional)	the name of a property of the associated class that is joined to the primary key of this class. If not specified, the primary key of the associated class is used.
access (optional - defaults to property)	the strategy Hibernate uses for accessing the property value.
formula (optional)	almost all one-to-one associations map to the primary key of the owning entity. If this is not the case, you can specify

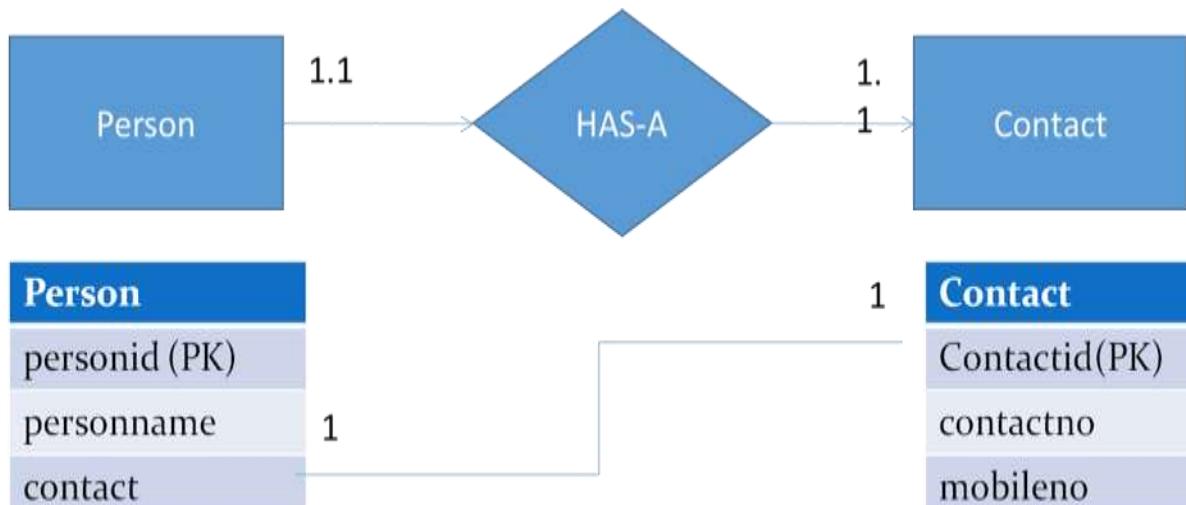
	another column, columns or expression to join on using an SQL formula. See org.hibernate.test.onetooneformula for an example.
<b>lazy (optional - defaults to proxy)</b>	by default, single point associations are proxied. lazy="no-proxy" specifies that the property should be fetched lazily when the instance variable is first accessed. It requires build-time bytecode instrumentation. lazy="false" specifies that the association will always be eagerly fetched. Note that if constrained="false", proxying is impossible and Hibernate will eagerly fetch the association.
<b>entity-name (optional)</b>	the entity name of the associated class.

**Example:**

```

<class name="Person">
    <id name="id" column="personId">
        <generator class="increment"/>
    </id>
    <many-to-one name="address" column="addressId" unique="true" not-null="true"/>
</class>
<class name="Address">
    <id name="id" column="addressId">
        <generator class="increment"/>
    </id>
</class>
<class name="Person">
    <id name="id" column="personId">
        <generator class="increment"/>
    </id>
</class>
<class name="Address">
    <id name="id" column="personId">
        <generator class="foreign">
            <param name="property">person</param>
        </generator>
    </id>
</class>
<one-to-one name="person" constrained="true"/>

```



**Person.java**

```
package com.bean;
public class Person {
    private int personid;
    private String personname;
    private Contact contact;
    // Generate getter and setter
}
```

**Person.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.bean.Person" table="PERSON">
    <id name="personid" type="int" column="PERSONID">
        <generator class="increment" />
    </id>
    <property name="personname" type="java.lang.String"
column="PERSONNAME"/>
    <many-to-one name="contact" column="CONTACT"
class="com.bean.Contact" fetch="join" unique="true"/>
</class>
</hibernate-mapping>
```

**Contact.java**

```
package com.bean;
public class Contact {
    private int contactid;
    private long contactno;
    private long mobileno;
    // Generate getter and setter
}
```

**Contact.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.bean.Contact" table="CONTACT">
```

## **MainClass.java**

```

package com.bean;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class MainClass {

    public static void main(String[] args) {
        Person p = new Person();
        p.setPersonname("Vishal");

        Contact c = new Contact();
        c.setContactno(123456789);
        c.setMobileno(321567985);

        p.setContact(c);

        SessionFactory sf = new Configuration().configure().buildSessionFactory();

        Session session = sf.openSession();
        Transaction tr = session.beginTransaction();
        session.save(p);
    }
}

```

## **Many-to-One Tag Property**

```

<many-to-one name="propertyName" column="column_name" class="ClassName"
cascade="cascade_style" fetch="join|select" update="true|false" insert="true|false"
property-ref="propertyNameFromAssociatedClass" access="field|property|ClassName"
unique="true|false" not-null="true|false" optimistic-lock="true|false" lazy="proxy|no-
proxy|false" not-found="ignore|exception" entity-name="EntityName"

```

Property	Description
<b>name</b>	the name of the property.
<b>column(optional)</b>	the name of the foreign key column. This can also be specified by nested <column> element(s).
<b>class(optional - defaults to the property type determined by reflection)</b>	the name of the associated class.
<b>cascade(optional)</b>	specifies which operations should be cascaded from the parent object to the associated object.
<b>fetch(optional - defaults to</b>	chooses between outer-join fetching or sequential select

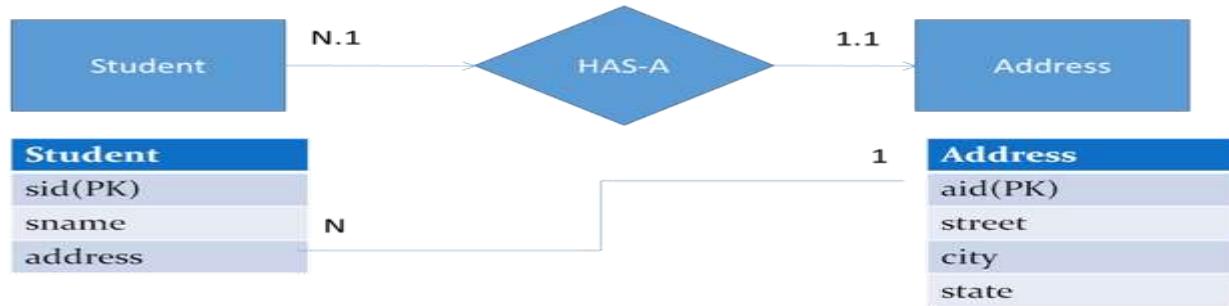
<b>select</b>	fetching.
<b>update, insert (optional - defaults to true)</b>	specifies that the mapped columns should be included in SQL UPDATE and/or INSERT statements. Setting both to false allows a pure "derived" association whose value is initialized from another property that maps to the same column(s), or by a trigger or other application.
<b>property-ref (optional)</b>	the name of a property of the associated class that is joined to this foreign key. If not specified, the primary key of the associated class is used.
<b>access (optional - defaults to property)</b>	the strategy Hibernate uses for accessing the property value.
<b>unique (optional)</b>	enables the DDL generation of a unique constraint for the foreign-key column. By allowing this to be the target of a property-ref, you can make the association multiplicity one-to-one.
<b>not-null (optional)</b>	enables the DDL generation of a nullability constraint for the foreign key columns.
<b>optimistic-lock (optional - defaults to true)</b>	specifies that updates to this property do or do not require acquisition of the optimistic lock. In other words, it determines if a version increment should occur when this property is dirty.
<b>lazy (optional - defaults to proxy)</b>	by default, single point associations are proxied. lazy="no-proxy" specifies that the property should be fetched lazily when the instance variable is first accessed. This requires build-time bytecode instrumentation. lazy="false" specifies that the association will always be eagerly fetched.
<b>not-found (optional - defaults to exception)</b>	specifies how foreign keys that reference missing rows will be handled. ignore will treat a missing row as a null association.
<b>entity-name (optional)</b>	the entity name of the associated class.
<b>formula (optional)</b>	an SQL expression that defines the value for a computed foreign key.

Example:

```

<class name="Person">
    <id name="id" column="personId">
        <generator class="increment"/>
    </id>
    <many-to-one name="address" column="addressId" not-null="true"/>
</class>
<class name="Address">
    <id name="id" column="addressId">

```



### Student.java

```

package com;
public class Student{
    int sid;
    String sname;
    Address address;
  
```

### Student.hbm.xml

```

<?xml version="1.0"?>
<hibernate-mapping>
<class name="com.Student" table="STUDENT">
<id name="sid" type="int">
<column name="SID"/>
<generator class="increment" /></id>
<property name="sname" type="java.lang.String">
<column name="SNAME" /></property>
<many-to-one name="address" class="com.Address" cascade="save-update">
<column name="ADDRESS" /></many-to-one></class></hibernate-mapping>
  
```

### Address.java

```

package com;
public class Address{
    int aid;
    String Street;
    String city;
    String state;
    //Generate getter and setter }
  
```

### Address.hbm.xml

```

<hibernate-mapping>
<class name="com.Address" table="ADDRESS">
<id name="aid" type="int">
<column name="AID"/>
<generator class="increment" /></id>
<property name="Street" type="java.lang.String">
<column name="STREET" /></property>
<property name="city" type="java.lang.String">
  
```

## MainClass.java

```

public class MainClass {
    public static void main(String[] args) {
        SessionFactory sf=new Configuration().configure().buildSessionFactory();
        Session session=sf.openSession();
        Transaction t=session.beginTransaction();
        Address address=new Address();
        address.setStreet("Rajpur");
        address.setCity("Dehradun");
        address.setState("Uttarakhand");
        Student s=new Student();
        s.setSname("Mohit");
        s.setAddress(address);
        Student s1=new Student();
        s1.setSname("Rohit");
        s1.setAddress(address);
        session.save(s);
        session.save(s1);
        t.commit();
        session.flush();
        session.close();
    }
}

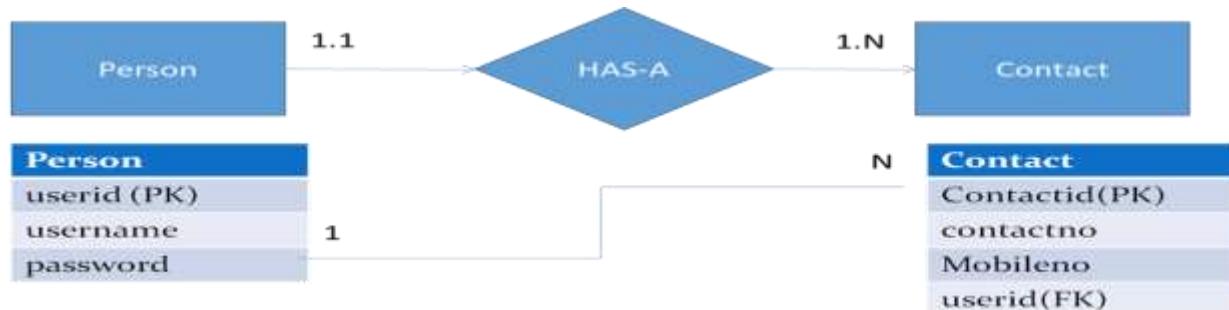
```

## One-to-Many Tag Property

```

<class name="Person">
    <id name="id" column="personId">
        <generator class="increment"/>
    </id>
    <set name="addresses">
        <key column="personId" not-null="true"/>
        <one-to-many class="Address"/>
    </set>
</class>
<class name="Address">
    <id name="id" column="addressId">
        <generator class="increment"/>
    </id>
</class>

```



**Person.java**

```
package com.bean;
public class Person {
    private int userid;
    private String username;
    private String password;
    private Set<Contact> contact = new HashSet<>();
    // Generate getter and setter
}
```

**Person.hbm.xml**

```
<hibernate-mapping>
<class name="com.bean.Person" table="PERSON_OM">
<id name="userid" type="int" column="USERID">
<generator class="increment" /></id>
<property name="username" type="java.lang.String">
<column name="USERNAME" /></property>
<property name="password" type="java.lang.String">
<column name="PASSWORD" /></property>
<set name="contact" inverse="false" table="CONTACT_OM" lazy="true" >
<key><column name="USERID" /></key>
<one-to-many class="com.bean.Contact" /></set></class>
</hibernate-mapping>
```

**Contact.java**

```
package com.bean;
public class Contact {
    private int contactid;
    private long contactno;
    private long mobileno;
    //Generate getter and setter
}
```

**Contact.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://.hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.bean.Contact" table="CONTACT_OM">
<id name="contactid" type="int">
<column name="CONTACTID" />
<generator class="increment" /></id>
<property name="contactno" type="long">
<column name="CONTACTNO" /></property>
<property name="lanlineno" type="long">
<column name="LANLINENO" /></property></class>
</hibernate-mapping>
```

**MainClass.java**

```

public class MainClass {
    public static void main(String[] args) {
        Contact contact = new Contact();
        contact.setContactno(123456789);
        contact.setLanlineno(446541231);
        Contact contact1 = new Contact();
        contact1.setContactno(789475646);
        contact1.setLanlineno(874646);
        Set<Contact> contacts = new HashSet<>();
        contacts.add(contact);
        contacts.add(contact1);
        Person p = new Person();
        p.setUsername("Vishal");
        p.setPassword("vishal...");
        p.setContact(contacts);
        Configuration cfg = new Configuration();
        SessionFactory sf = cfg.configure().buildSessionFactory();
        Session session = sf.openSession();
        Transaction tr = session.beginTransaction();
        session.save(p);
        session.save(contact);
        session.save(contact1);
        tr.commit();
        session.close();
        sf.close();
    }
}

```

Property	Description
<b>column (optional)</b>	the name of the element foreign key column.
<b>formula (optional)</b>	an SQL formula used to evaluate the element foreign key value.
<b>class (required)</b>	the name of the associated class.
<b>fetch (optional - defaults to join)</b>	enables outer-join or sequential select fetching for this association. This is a special case; for full eager fetching in a single SELECT of an entity and its many-to-many relationships to

	other entities, you would enable join fetching, not only of the collection itself, but also with this attribute on the <many-to-many> nested element.
<b>unique (optional)</b>	enables the DDL generation of a unique constraint for the foreign-key column. This makes the association multiplicity effectively one-to-many.
<b>not-found (optional - defaults to exception)</b>	specifies how foreign keys that reference missing rows will be handled: ignore will treat a missing row as a null association.
<b>entity-name (optional)</b>	the entity name of the associated class, as an alternative to class.
<b>property-ref (optional)</b>	the name of a property of the associated class that is joined to this foreign key. If not specified, the primary key of the associated class is used.

## Example

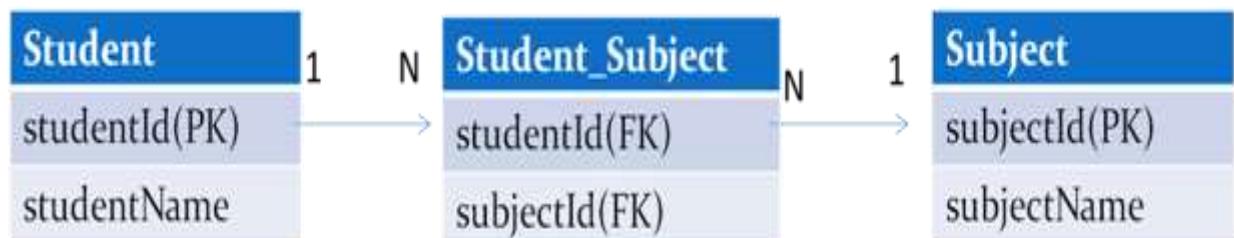
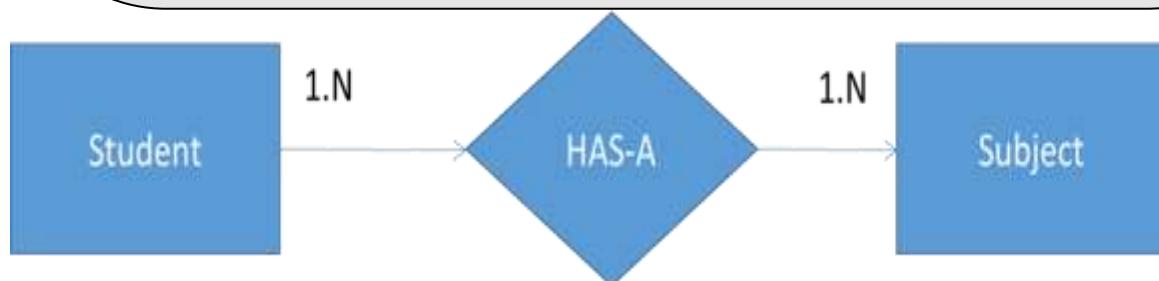
```
<class name="Person">
    <id name="id" column="personId">
        <generator class="increment"/>
    </id>
    <set name="addresses" table="Person_Address">
        <key column="personId"/>
        <many-to-many column="addressId"
            class="Address"/>
    </set>
</class>
<class name="Address">
    <id name="id" column="addressId">
        <generator class="increment"/>
    </id>
</class>
```

**Student.java**

```
package com.bean;
public class Student {
    public int studentId;
    public String studentName;
    private Set<Subject> subjects = new HashSet<Subject>();
    // Generate getter and setter
}
```

**Student.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.bean.Student" table="STUDENT">
<id name="studentId" type="int">
<column name="STUDENTID" />
<generator class="increment" />
</id>
<property name="studentName" type="java.lang.String">
<column name="STUDENTNAME" />
</property>
```



**Subject.java**

```
package com.bean;
public class Subject {
    private int subjectId;
    private String subjectName;
    //Generate getter and setter
}
```

**Subject.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.bean.Subject" table="SUBJECT">
<id name="subjectId" type="int">
<column name="SUBJECTID" />
<generator class="increment" />
</id>
<property name="subjectName" type="java.lang.String">
<column name="SUBJECTNAME" />
</property>
</class>
</hibernate-mapping>
```

**MainClass.java**

```
public class MaiClass {
static SessionFactory sf = new Configuration().configure().buildSessionFactory();
public static void main(String[] args) {
    Subject sb = new Subject();
    sb.setSubjectName("Maths");
    addSubject(sb);

    Subject sb1 = new Subject();
    sb1.setSubjectName("Science");
    addSubject(sb1);

    Subject sb2 = new Subject();
    sb2.setSubjectName("History");
    addSubject(sb2);

    //Get Subjects from DB
    List<Subject> subjects = getSubjects();
    Student s = new Student();
    s.setStudentName("Hello 32");

    Set<Subject> subjects1 = new HashSet<Subject>();
    subjects1.add(subjects.get(0));
    subjects1.add(subjects.get(1));
    s.setSubjects(subjects1);
    addStudent(s);

    //2nd student
```

```

public static void addStudent(Student s){
    Session session = sf.openSession();
    Transaction tr = session.beginTransaction();
    session.save(s);
    tr.commit();
    session.close();
}
public static void addSubject(Subject sb){
    Session session = sf.openSession();
    Transaction tr = session.beginTransaction();
    session.save(sb);
    tr.commit();
    session.close();
}
public static List<Subject> getSubjects(){
    Session session = sf.openSession();

```

### **Hibernate Query Language**

- Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries which in turns perform action on database.
- Although you can use SQL statements directly with Hibernate using Native SQL but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.
- Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table and column names are case sensitive in HQL.

### **HQL using Criteria**

- Hibernate provides alternate ways of manipulating objects and in turn data available in RDBMS tables. One of the methods is Criteria API which allows you to build up a criteria query object programmatically where you can apply filtration rules and logical conditions.
- The Hibernate **Session** interface provides **createCriteria()** method which can be used to create a **Criteria** object that returns instances of the persistence object's class when your application executes a criteria query.
- Following is the simplest example of a criteria query is one which will simply return every object that corresponds to the Employee class.

<b>Using HQL</b>	<b>Using Criteria</b>
<b>FROM Clause using HQL</b> <pre>String hql = "FROM Employee"; Query query = session.createQuery(hql); List results = query.list();</pre>	<b>FROM Clause using Criteria</b> <pre>Criteria criteria = session.createCriteria(Employee.class); List results = criteria .list();</pre>

<b>Where Clause using HQL</b> <pre>String hql = "FROM Employee E WHERE E.salary =               2000 "; Query query = session.createQuery(hql); List results = query.list();</pre>	<b>Restrictions Clause using Criteria</b> <pre>Criteria criteria = session.createCriteria(Employee.class); criteria add(Restrictions.eq("salary", 2000)); List results = criteria .list();</pre>
<b>Where and AS Clause using HQL</b> <pre>String hql = "FROM Employee E WHERE E.salary =               2000 "; Query query = session.createQuery(hql); List results = query.list();</pre>	<b>Restrictions Clause using Criteria</b> <pre>Criteria criteria = session.createCriteria(Employee.class); criteria add(Restrictions.eq("salary", 2000)); List results = criteria .list();</pre>

## Overview on Spring Frameworks

Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly. It was developed by Rod Johnson in 2003. Spring framework makes the easy development of JavaEE application.

Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems.

The spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC, Security, Roo etc. We will learn these modules in next page. Let's understand the IOC and Dependency Injection first.

### Pros of Using Spring

- **Lightweight:** because of its POJO implementation. The Spring Framework doesn't force the programmer to inherit any class or implement any interface. That is why it is said non-invasive.
- **Predefine Templates:** It provides templates for JDBC, Hibernate, JPA etc. technologies. So there is no need to write too much code. It hides the basic steps of these technologies.
- **Loose Coupling:** because of dependency injection and handles injecting dependent components without a component knowing where they came from Inversion of Control (IoC).
- **Easy to test:** The Dependency Injection makes easier to test the application. The EJB or Struts application require server to run the application but Spring framework doesn't require server.
- **Fast Development:** The Dependency Injection feature of Spring Framework and it support to various frameworks makes the easy development of JavaEE application.
- **Powerful abstraction:** It provides powerful abstraction to JavaEE specifications such as JMS, JDBC, JPA and JTA.
- **Declarative supports:** It provides declarative support for caching, validation, transactions and formatting.
- **Portable:** can use server-side in web/ejb app, client-server in swing app, business logic is completely portable.
- **Cross-cutting behavior:** Resource Management is cross-cutting concern, easy to copy-and-paste everywhere.

## Inversion of Control (IoC) and Dependency Injection (DI)

Dependency Injection (DI) flavor of Inversion of Control. The Inversion of Control (IoC) is a general concept, and it can be expressed in many different ways and Dependency Injection is merely one concrete example of Inversion of Control.

What is dependency injection exactly? - Let's look at these two words separately. Here the dependency part translates into an association between two classes. For example, class A is dependent on class B. Now, let's look at the second part, injection. All this means is that class B will get injected into class A by the IoC.

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while doing unit testing. Dependency Injection helps in gluing these classes together and same time keeping them independent.

Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods. As Dependency Injection is the heart of Spring Framework.

These are the design patterns that are used to remove dependency from the programming code. They make the code easier to test and maintain. Let's understand this with the following code:

```
class Employee{  
    Address address;  
    Employee(){  
        address=new Address();  
    }  
}
```

In such case, there is dependency between the Employee and Address (tight coupling). In the Inversion of Control scenario, we do this something like this:

```
class Employee{  
    Address address;  
    Employee(Address address){  
        this.address=address;  
    }  
}
```

Thus, IOC makes the code loosely coupled. In such case, there is no need to modify the code if our logic is moved to new environment.

In Spring framework, IOC container is responsible to inject the dependency. We provide metadata to the IOC container either by XML file or annotation.

## Aspect Oriented Programming (AOP)

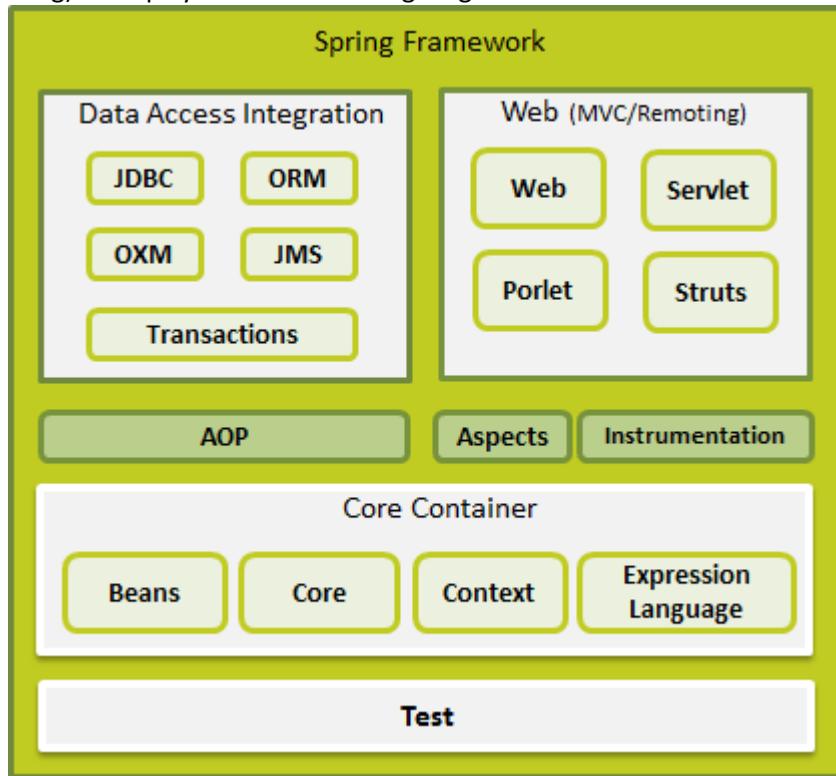
One of the key components of Spring is the Aspect oriented programming (AOP) framework. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects including logging, declarative transactions, security, and caching etc. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.

Whereas DI helps you decouple your application objects from each other, AOP helps you decouple cross-cutting concerns from the objects that they affect.

The AOP module of Spring Framework provides aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

## Introduction of Spring Framework Architecture

The Spring framework comprises of many modules such as core, beans, context, expression language, AOP, Aspects, Instrumentation, JDBC, ORM, OXM, JMS, Transaction, Web, Servlet, Struts etc. These modules are grouped into Test, Core Container, AOP, Aspects, Instrumentation, Data Access / Integration, Web (MVC / Remoting) as displayed in the following diagram.



## Spring Core Container

The Spring Core container contains core, beans, context and expression language (EL) modules.

- **Core**: These modules provide IOC and Dependency Injection features.
- **Beans**: These modules provides BeanFactory and ApplicationContext, which is sophisticated implementation of the factory pattern
- **Context**: This module supports internationalization (I18N), EJB, JMS, Basic Remoting.
- **Expression Language**: This module provides a powerful expression language for querying and manipulating an object at runtime. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, and retrieval of objects by name etc.

## Data Access/Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

- **JDBC**: This module provides a JDBC-abstraction layer that removes the need to do tedious JDBC related coding.
- **ORM**: This module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.

- **OXM:** This module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- **JMS(Java Messaging Service):** This module contains features for producing and consuming messages.
- **Transaction:** module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.

## Web

The Web layer consists of the Web, Web-Servlet, Web-Struts, and Web-Portlet modules.

- **Web:** This module provides basic web-oriented integration features such as multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context.
- **Web-Servlet:** This module contains Spring's model-view-controller (MVC) implementation for web applications.
- **Web-Struts:** This module contains the support classes for integrating a classic Struts web tier within a Spring application.
- **Web-Portlet:** This module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.

## Misc

There are few other important modules like AOP, Aspects, Instrumentation, Web and Test modules.

- **AOP:** This module provides aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.
- **Aspects:** This module provides integration with AspectJ which is again a powerful and mature aspect oriented programming (AOP) framework.
- **Instrumentation:** This module provides class instrumentation support and class loader implementations to be used in certain application servers.
- **Test:** This module supports the testing of Spring components with JUnit or TestNG frameworks.

## Spring IOC Containers

The IoC container is responsible to instantiate, configure and assemble the objects. The IoC container gets information's from the XML file and works accordingly. The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete lifecycle from creation till destruction. The Spring container uses dependency injection (DI) to manage the components that make up an application. The main tasks performed by IoC container are:

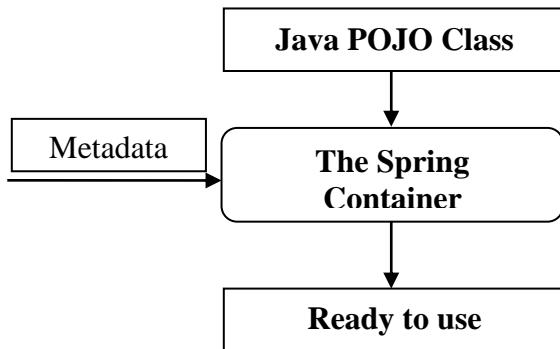
- to instantiate the application class
- to configure the object
- to assemble the dependencies between the objects

There are two types of IoC containers. They are:

- BeanFactory

- ApplicationContext

The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.



### Difference between BeanFactory and the ApplicationContext

The `org.springframework.beans.factory.xml.XmlBeanFactory`, `org.springframework.context.support.ClassPathXmlApplicationContext` and the `org.springframework.context.support.FileSystemXmlApplicationContext` as the IoC container. The `ApplicationContext` interface is built on top of the `BeanFactory` interface. It adds some extra functionality than `BeanFactory` such as simple integration with spring's AOP, message resource handling (for I18N), event propagation, application layer specific context (e.g. `WebApplicationContext`) for web application. So it is better to use `ApplicationContext` than `BeanFactory`.

### Using BeanFactory

The `XmlBeanFactory` is the implementation class for the `BeanFactory` interface. To use the `BeanFactory`, we need to create the instance of `XmlBeanFactory` class, which is deprecated form the spring 3.0 jar files as given below:

```
XmlBeanFactory factory = new XmlBeanFactory(new ClassPathResource("Beans.xml"));
```

The constructor of `XmlBeanFactory` class receives the `Resource` object so we need to pass the `Resource` object to create the object of `BeanFactory`.

### Using ApplicationContext

The `ClassPathXmlApplicationContext` class is the implementation class of `ApplicationContext` interface. We need to instantiate the `ClassPathXmlApplicationContext` class to use the `ApplicationContext` as given below:

```
ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
```

-----Or-----

```
ApplicationContext context = new FileSystemXmlApplicationContext
```

("D:/Vishal/Workspace/Spring/HelloWorld/src/Beans.xml");

The constructor of ClassPathXmlApplicationContext or FileSystemXmlApplicationContext class receives string, so we can pass the name of the xml file to create the instance of ApplicationContext.

## Spring Hello World using IDE

Here, we are going to create a simple application of spring framework using eclipse IDE. Let's see the simple steps to create the spring application in Eclipse IDE.

1. Create the java project
2. Add or configure spring jar files
3. Create the class
4. Create the xml file to provide the values
5. Create the main class

### Create the Java Project

Go to File menu -> New -> project -> Java Project. Write the project name e.g. HelloWorld -> Finish. Now the java project is created.

### Add spring jar files

There are mainly three jar files required to run this application.

- org.springframework.aop-3.1.0.RELEASE
- org.springframework.asm-3.1.0.RELEASE
- org.springframework.aspects-3.1.0.RELEASE
- org.springframework.beans-3.1.0.RELEASE
- org.springframework.context.support-3.1.0.RELEASE
- org.springframework.context-3.1.0.RELEASE
- org.springframework.core-3.1.0.RELEASE
- org.springframework.expression-3.1.0.RELEASE
- antlr-2.7.2
- commons-logging-1.1.1

To load the jar files in eclipse IDE, Right click on your project -> Build Path -> Add external archives - > select all the required jar files -> finish

### Create Java class

In such case, we are simply creating the Student class have name property. The name of the student will be provided by the xml file. It is just a simple example not the actual use of spring. We will see the actual use in Dependency Injection chapter.

To create the java class, Right click on src -> New -> class -> Write the class name e.g. HelloWorld -> finish.  
Write the following code:

```
package com.vishal.spring.ioc;
```

```
public class HelloWorld {
```

```
private String message;  
  
public String getMessage() {  
    return message;  
}  
  
public void setMessage(String message) {  
    this.message = message;  
}  
  
}
```

This is simple bean class, containing only one property name with its getters and setters method.

### Create the xml file

To create the xml file click on src -> new -> file -> give the file name such as Beans.xml -> finish. Open the Beans.xml file, and write the following code:

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
  
<bean id="hello" class="com.vishal.spring.ioc.HelloWorld">  
    <property name="message" value="Vishal Shah"></property>  
</bean>  
  
</beans>
```

### Create the Main Class using BeanFactory

Create the java class e.g. Test. Here we are getting the object of Student class from the IOC container using the getBean() method of BeanFactory. Let's see the code of test class.

```
import org.springframework.beans.factory.xml.XmlBeanFactory;  
import org.springframework.core.io.ClassPathResource;
```

```
public class RunBeanFactoryMain {  
  
    public static void main(String[] args) {  
  
        XmlBeanFactory factory = new XmlBeanFactory(new ClassPathResource(  
                "Beans.xml"));  
  
        HelloWorld obj = (HelloWorld) factory.getBean("hello");  
        System.out.println(">>" + obj.getMessage());  
  
    }  
}
```

## Create the Main Class using ApplicationContext

Create the java class e.g. Test. Here we are getting the object of Student class from the IOC container using the getBean() method of BeanFactory. Let's see the code of test class.

```
package com.vishal.spring.ioc;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class RunApplicationContextMain {

    public static void main(String[] args) {
        //ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        ApplicationContext context = new FileSystemXmlApplicationContext
            ("D:/Vishal/Workspace/Spring/HelloWorld/src/Beans.xml");

        HelloWorld obj = (HelloWorld) context.getBean("hello");

        System.out.println(">>> " + obj.getMessage());
    }
}
```

## Output of the Program

```
Nov 05, 2014 6:00:26 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@134ce4a: startup date [Wed Nov 05 18:00:26 IST 2014]; root of context hierarchy
Nov 05, 2014 6:00:26 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 05, 2014 6:00:27 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1cf850b: definition
>>> Vishal Shah
```

## Spring Bean Definition

The objects that form the backbone of your application and that are managed by the Spring IoC container are called beans. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container, for example, in the form of XML <bean/> definitions

The bean definition contains the information called configuration metadata which is needed for the container to know the followings:

- How to create a bean
- Bean's lifecycle details

- Bean's dependencies

All the above configuration metadata translates into a set of the following properties that make up each bean definition.

<b>Properties</b>	<b>Description</b>
<b>Class</b>	This attribute is mandatory and specify the bean class to be used to create the bean.
<b>Name</b>	This attribute specifies the bean identifier uniquely. In XML-based configuration metadata, you use the id and/or name attributes to specify the bean identifier(s).
<b>Scope</b>	This attribute specifies the scope of the objects created from a particular bean definition and it will be discussed in bean scopes chapter.
<b>constructor-arg</b>	This is used to inject the dependencies and will be discussed in next chapters.
<b>Properties</b>	This is used to inject the dependencies and will be discussed in next chapters.
<b>autowiring mode</b>	This is used to inject the dependencies and will be discussed in next chapters.
<b>lazy-initialization mode</b>	A lazy-initialized bean tells the IoC container to create a bean instance when it is first requested, rather than at startup.
<b>initialization method</b>	A callback to be called just after all necessary properties on the bean have been set by the container. It will be discussed in bean life cycle chapter.
<b>destruction method</b>	A callback to be used when the container containing the bean is destroyed. It will be discussed in bean life cycle chapter.

## Spring Configuration Metadata

Spring IoC container is totally decoupled from the format in which this configuration metadata is actually written. There are following three important methods to provide configuration metadata to the Spring Container:

- XML based configuration file.
- Annotation-based configuration
- Java-based configuration

## Spring Life Cycle

When bean is initialized it might require to perform some activity before it can come into use able state (State in which application can use it) and when bean is getting destroyed there might be some cleanup activity required for given bean. These activities are known as bean Lifecycle.

- Container will contain beans as long as they are required by Application.
- Beans created outside Spring container can also be registered with AC(Application Context).
- BeanFactory is root interface for accessing the bean container. Other interfaces are also available for specific purpose.

- BeanFactory is a central registry of application components (Beans).
- These component (Beans) have lifecycle interfaces and methods which will be invoked in some order before Bean can be handed over to application and before Bean is getting destroyed.

Through, there are lists of the activities that take place behind the scenes between the time of bean Instantiation and its destruction. To define setup and teardown for a bean, we simply declare the <bean> with **init-method** and/or **destroy-method** parameters.

- The init-method attribute specifies a method that is to be called on the bean immediately upon instantiation.
- Similarly, destroy-method specifies a method that is called just before a bean is removed from the container.

## Program

### Bean.java

```
package com.vishal.spring.lifecycle;

public class Bean {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public void init() {
        // TODO Auto-generated method stub
        System.out.println("Initialize the Bean");
    }

    public void destroy() {
        // TODO Auto-generated method stub
        System.out.println("Destroy the Bean");
    }
}
```

### Bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
<bean id="hello" class="com.vishal.spring.lifecycle.Bean" init-method="init" destroy-
method="destroy">
    <property name="message" value="Welcome, to the World!!!"></property>
</bean>

</beans>
RunLifeCycleMain.java
package com.vishal.spring.lifecycle;

import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunLifeCycleMain {

    public static void main(String[] args) {

        AbstractApplicationContext context = new
ClassPathXmlApplicationContext("Beans.xml");

        Bean bean = (Bean)context.getBean("hello");

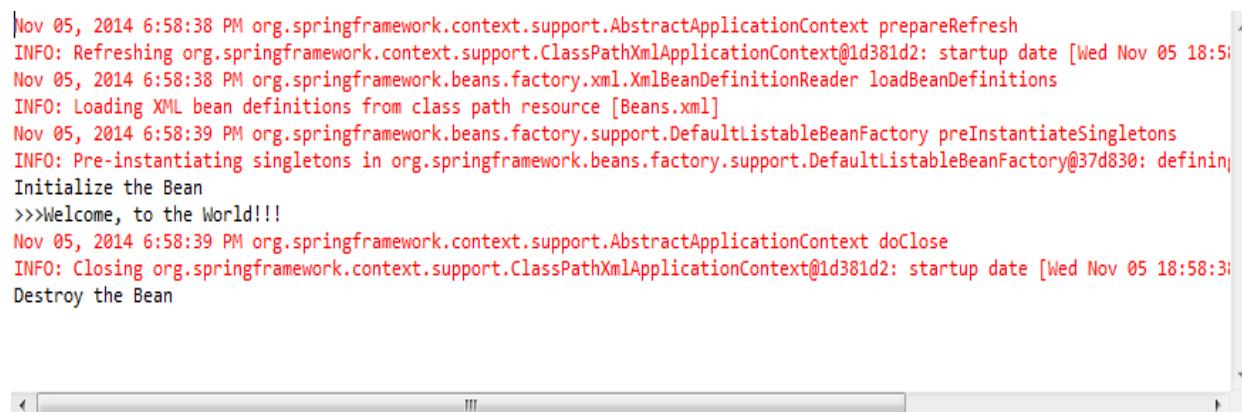
        System.out.println(">>>" + bean.getMessage());

        context.registerShutdownHook(); // This will ensures a graceful shutdown and calls the
relevant destroy methods.
    }

}
```

## Output:

```
Nov 05, 2014 6:58:38 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d381d2: startup date [Wed Nov 05 18:58:38 IST 2014]
Nov 05, 2014 6:58:38 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 05, 2014 6:58:39 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@37d830: defining beans [hello]; root of factory hierarchy
Initialize the Bean
>>>Welcome, to the World!!!
Nov 05, 2014 6:58:39 PM org.springframework.context.support.AbstractApplicationContext doClose
INFO: Closing org.springframework.context.support.ClassPathXmlApplicationContext@1d381d2: startup date [Wed Nov 05 18:58:38 IST 2014]
Destroy the Bean
```



Inheritance

A bean definition can contain a lot of configuration information, including constructor arguments, property values, and container-specific information such as initialization method, static factory method name, and so on.

A child bean definition inherits configuration data from a parent definition. The child definition can override some values, or add others, as needed.

Spring Bean definition inheritance has nothing to do with Java class inheritance but inheritance concept is same. You can define a parent bean definition as a template and other child beans can inherit required configuration from the parent bean.

When you use XML-based configuration metadata, you indicate a child bean definition by using the parent attribute, specifying the parent bean as the value of this attribute.

## Program

### Customer.java

```
package com.vishal.spring.inheritance;

public class Customer {

    private int type;
    private String action;
    private String country;
    public int getType() {
        return type;
    }
    public void setType(int type) {
        this.type = type;
    }
    public String getAction() {
        return action;
    }
    public void setAction(String action) {
        this.action = action;
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
    public String toString() {
        return "Customer [type=" + type + ", action=" + action + ", country="
            + country + "]";
    }
}
```

### Bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
<!-- Inheritance without abstract -->

<!-- <bean id="customer" class="com.vishal.spring.inheritance.Customer">
    <property name="country" value="India"></property>
</bean>

<bean id="countryBean" parent="customer" >
    <property name="type" value="0"></property>
    <property name="action" value="Buy and Sell"></property>
</bean> -->

<!-- inheritance using abstract - pure abstract -->

<bean id="customer" class="com.vishal.spring.inheritance.Customer" abstract="true">
    <property name="country" value="India"></property>
</bean>

<bean id="countryBean" parent="customer" >
    <property name="country" value="USA"></property>
    <property name="type" value="0"></property>
    <property name="action" value="Buy and Sell"></property>
</bean>

</beans>
```

## RunBeanInheritanceMain.java

```
package com.vishal.spring.inheritance;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class RunBeanInheritanceMain {
```

```
    public static void main(String[] args) {
```

```
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
```

```
        Customer customer = (Customer)context.getBean("countryBean");
        System.out.println(customer);
    }
```

```
}
```

## Output

```
Nov 07, 2014 8:42:59 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@134ce4a: startup date [Fri Nov 07 20:42:59 IST 2014]; root of context hierarchy
Nov 07, 2014 8:42:59 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 07, 2014 8:43:00 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@66db21: defining beans [Customer]; root of factory hierarchy
Customer [type=0, action=Buy and Sell, country=USA]
```

## Bean Scope

To force Spring to produce a new bean instance each time one is needed, you should declare the bean's scope attribute to be prototype. Similar way if you want Spring to return the same bean instance each time one is needed, you should declare the bean's scope attribute to be singleton.

The Spring Framework supports following five scopes, three of which are available only if you use a web-aware ApplicationContext.

Scope	Description
Singleton	This scopes the bean definition to a single instance per Spring IoC container (default).
Prototype	This scopes a single bean definition to have any number of object instances.
Request	This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
Session	This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
global-session	This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.

## The Singleton Scope

If scope is set to singleton, the Spring IoC container creates exactly one instance of the object defined by that bean definition. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached object.

## Program

### Bean.java

```
package com.vishal.spring.scope;

public class Bean {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

```
}

}

Beans.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="hello" class="com.vishal.spring.scope.Bean" scope="singleton">
        </bean>
    </beans>
RunSingletonScopeMain.java
package com.vishal.spring.scope;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunSingletonScopeMain {

    public static void main(String[] args) {

        //This scopes the bean definition to a single instance per Spring IoC container (default).

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
        Bean bean = (Bean)context.getBean("hello");

        bean.setMessage("A");
        System.out.println("Bean Instatnce 1>>>" + bean.getMessage());

        Bean bean1 = (Bean)context.getBean("hello");
        //bean1.setMessage("A");
        System.out.println("Bean Instatnce 2>>>" + bean1.getMessage());

    }
}
```

## Output

```
Nov 05, 2014 7:10:19 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@f01771: startup date [Wed Nov 05 19:10:19 IST 2014]; root of context hierarchy
Nov 05, 2014 7:10:20 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 05, 2014 7:10:20 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@154c72c: definition for singleton Bean Instatnce 1>>>A
Bean Instatnce 2>>>A
```

## The Prototype Scope

If scope is set to prototype, the Spring IoC container creates new bean instance of the object every time a request for that specific bean is made. As a rule, use the prototype scope for all state-full beans and the singleton scope for stateless beans.

### Program

#### Bean.java

```
package com.vishal.spring.scope;

public class Bean {

    private String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

#### Beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
    <bean id="helloproto" class="com.vishal.spring.scope.Bean" scope="prototype">
    </bean>
```

```
</beans>
```

#### RunPrototypeScopeMain.java

```
package com.vishal.spring.scope;

import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunPrototypeScopeMain {
    public static void main(String[] args) {

        //Spring IoC container creates new bean instance of the object every time a request for
        that specific bean is made.

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
        Bean bean = (Bean)context.getBean("helloproto");

        bean.setMessage("Tops Technologies");
        System.out.println(">>>" + bean.getMessage());

        Bean bean1 = (Bean)context.getBean("helloproto");
        System.out.println(">>>" + bean1.getMessage());

    }
}
```

## Output

```
|Nov 05, 2014 7:14:32 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@933bcb: startup date [Wed Nov 05 19:14
Nov 05, 2014 7:14:32 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 05, 2014 7:14:32 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@5cccd3: definin
>>>Tops Technologies
>>>null
```

## Spring Dependency Injections

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled. Every java based application has a few objects that work together to present what the end-user sees as a working application. When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while doing unit testing. To understand the DI better, Let's understand the Dependency Lookup (DL) first:

The Dependency Lookup is an approach where we get the resource after demand. There can be various ways to get the resource for example:

**T obj = new TImpl();**

In such way, we get the resource (instance of T class) directly by new keyword. Another way is factory method:

**T obj = T.getT();**

This way, we get the resource (instance of T class) by calling the static factory method `getT()`.

Alternatively, we can get the resource by JNDI (Java Naming Directory Interface) as:

```
Context ctx = new InitialContext();
Context environmentCtx = (Context) ctx.lookup("java:comp/env");
A obj = (A)environmentCtx.lookup("T");
```

The Dependency Injection is a design pattern that removes the dependency of the programs. In such case we provide the information from the external source such as XML file. It makes our code loosely coupled and easier for testing. In such case we write the code as:

```
class Employee{
Address address;

Employee(Address address){
this.address=address;
}
public void setAddress(Address address){
this.address=address;
}

}
```

In such case, instance of Address class is provided by external source such as XML file either by constructor or setter method.

Two ways to perform Dependency Injection in Spring framework:

- By Constructor
- By Setter method

## Dependency Injection By Constructor Based

We can inject the dependency by constructor. The `<constructor-arg>` sub element of `<bean>` is used for constructor injection. Here we are going to inject

- primitive and String-based values
- Dependent object (contained object)
- Collection values and objects

## Program

**Person.java**

```
package com.vishal.spring.di.counstructor;
```

```
public class Person {

    private String name;
    private String no;
    private int age;

    public Person(String name, String no, int age) {
        this.name = name;
```

```
        this.no = no;
        this.age = age;
    }

    public void callPhone() {
        // TODO Auto-generated method stub
        System.out.println("Call to " + name + " on " + no);
    }

    public void showAge() {
        // TODO Auto-generated method stub
        System.out.println( name+"s age is :" + age);
    }

}

Bean.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="person" class="com.vishal.spring.di.counstructor.Person">
        <constructor-arg name="name" value="Vishal Shah"></constructor-arg>
        <constructor-arg name="no" value="9876543210"></constructor-arg>
        <constructor-arg index="2" value="23" type="int"></constructor-arg>
    </bean>

</beans>
RunConstructorDIMain.java
package com.vishal.spring.di.counstructor;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunConstructorDIMain {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        Person person = (Person)context.getBean("person");

        person.callPhone();

        person.showAge();
    }
}
```

```
}
```

```
}
```

## Output

```
Nov 06, 2014 11:47:02 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d381d2: startup date [Thu Nov 06 11:47:02 IST 2014]; root of context hierarchy
Nov 06, 2014 11:47:02 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 11:47:03 AM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1f4ba0d: definition for singleton Bean
Call to Vishal Shah on 9876543210
Vishal Shah's age is :23
```

## Dependency Injection By Setter-Getter Based

We can inject the dependency by setter method also. The <property>subelement of <bean> is used for setter injection. Here we are going to inject

- primitive and String-based values
- Dependent object (contained object)
- Collection values and objects.

## Program

### Person.java

```
package com.vishal.spring.di.setter;

public class Person {

    private Phone phone;

    public Phone getPhone() {
        System.out.println("getting Phone by getter method");
        return phone;
    }

    public void setPhone(Phone phone) {
        System.out.println("setting Phone by setter method");
        this.phone = phone;
    }

    public void callPhone() {
        // TODO Auto-generated method stub
        System.out.println("Call Phone");
        phone.call();
    }
}
```

**Phone.java**

```
package com.vishal.spring.di.setter;

public class Phone {

    private String phoneNumber;

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public Phone() {
        // TODO Auto-generated constructor stub
        System.out.println("Inside Phone Constructor");
    }

    public void call() {
        // TODO Auto-generated method stub
        System.out.println("Call to " + phoneNumber);
    }
}
```

**Bean.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="person" class="com.vishal.spring.di.setter.Person">
        <property name="phone" ref="phone"></property>
    </bean>

    <bean id="phone" class="com.vishal.spring.di.setter.Phone">
        <property name="phoneNumber" value="7898654521"></property>
    </bean>

</beans>
```

**RunSetterDIMain.java**

```
package com.vishal.spring.di.setter;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunSetterDIMain {
```

```
public static void main(String[] args) {  
  
    ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");  
  
    Person person = (Person)context.getBean("person");  
  
    person.callPhone();  
  
}  
}
```

## Output

```
|Nov 06, 2014 12:20:47 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh  
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d381d2: startup date [Thu Nov 06 12:20:47 IST 2014  
Nov 06, 2014 12:20:48 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions  
INFO: Loading XML bean definitions from class path resource [Beans.xml]  
Nov 06, 2014 12:20:48 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons  
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@bc5aea: defining beans [pers  
Inside Phone Constructor  
setting Phone by setter method  
Call Phone  
Call to 7898654521
```



## Dependency Injection By Object Based

### Person.java

```
package com.vishal.spring.di.counstructor;  
  
public class Person {  
  
    private Phone phone;  
  
    public Person(Phone phone) {  
        // TODO Auto-generated constructor stub  
        System.out.println("Inside Person Constructor");  
        this.phone = phone;  
    }  
  
    public void callPhone() {  
        // TODO Auto-generated method stub  
        System.out.println("Call Phone");  
        phone.call();  
    }  
}
```

### Phone.java

```
package com.vishal.spring.di.counstructor;
```

```
public class Phone {  
  
    public Phone() {  
        // TODO Auto-generated constructor stub  
        System.out.println("Inside Phone Constructor");  
    }  
  
    public void call() {  
        // TODO Auto-generated method stub  
        System.out.println("Calling...");  
    }  
}  
Bean.xml  
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
  
    <bean id="person" class="com.vishal.spring.di.counstructor.Person">  
        <constructor-arg ref="phone"></constructor-arg>  
    </bean>  
  
    <bean id="phone" class="com.vishal.spring.di.counstructor.Phone"></bean>  
  
</beans>  
RunConstructorDIMain.java  
package com.vishal.spring.di.counstructor;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class RunConstructorDIMain {  
  
    public static void main(String[] args) {  
  
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");  
  
        Person person = (Person)context.getBean("person");  
  
        person.callPhone();  
  
    }  
}
```

## Output

```
Nov 06, 2014 11:18:11 AM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@179953c: startup date [Thu Nov 06 11:18:11 IST 2014]; root of context hierarchy
Nov 06, 2014 11:18:11 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 11:18:11 AM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@61a2e7: defining beans; root of context hierarchy
Inside Phone Constructor
Inside Person Constructor
Call Phone
Calling...
```

## Inner Beans Alias and IdRef

Java inner classes are defined within the scope of other classes, similarly, inner beans are beans that are defined within the scope of another bean. Thus, a <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean.

## Program using Constructor

### Person.java

```
package com.vishal.spring.di.counstructor;

public class Person {

    private Phone phone;

    public Person(Phone phone) {
        // TODO Auto-generated constructor stub
        System.out.println("Inside Person Constructor");
        this.phone = phone;
    }

    public void callPhone() {
        // TODO Auto-generated method stub
        System.out.println("Call Phone");
        phone.call();
    }
}
```

### Phone.java

```
package com.vishal.spring.di.counstructor;

public class Phone {

    public Phone() {
        // TODO Auto-generated constructor stub
        System.out.println("Inside Phone Constructor");
    }
}
```

```
public void call() {  
    // TODO Auto-generated method stub  
    System.out.println("Calling...");  
}  
}  
Bean.xml  
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
  
    <bean id="person" class="com.vishal.spring.di.counstructor.Person">  
        <constructor-arg>  
            <bean id="phone" class="com.vishal.spring.di.counstructor.Phone"></bean>  
        </constructor-arg>  
    </bean>  
  
</beans>  
RunConstructorDIMain.java  
package com.vishal.spring.di.counstructor;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class RunConstructorDIMain {  
  
    public static void main(String[] args) {  
  
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");  
  
        Person person = (Person)context.getBean("person");  
  
        person.callPhone();  
  
    }  
}
```

## Output

```
Nov 06, 2014 1:55:03 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@134ce4a: startup date [Thu Nov 06 13:55:03 IST 2014]
Nov 06, 2014 1:55:03 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 1:55:03 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@7fc85d: defining beans [pers
Inside Phone Constructor
Inside Person Constructor
Call Phone
Calling...
```

## Program using Setter-Getter

### Person.java

```
package com.vishal.spring.di.setter;

public class Person {

    private Phone phone;

    public Phone getPhone() {
        System.out.println("getting Phone by getter method");
        return phone;
    }

    public void setPhone(Phone phone) {
        System.out.println("setting Phone by setter method");
        this.phone = phone;
    }

    public void callPhone() {
        // TODO Auto-generated method stub
        System.out.println("Call Phone");
        phone.call();
    }
}
```

### Phone.java

```
package com.vishal.spring.di.setter;

public class Phone {

    private String phoneNumber;

    public String getPhoneNumber() {
        return phoneNumber;
    }
}
```

```
public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}

public Phone() {
    // TODO Auto-generated constructor stub
    System.out.println("Inside Phone Constructor");
}

public void call() {
    // TODO Auto-generated method stub
    System.out.println("Calling to " + phoneNumber);
}
}

Bean.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="person" class="com.vishal.spring.di.setter.Person">
        <property name="phone">
            <bean id="phone" class="com.vishal.spring.di.setter.Phone">
                <property name="phoneNumber" value="9876543210"></property>
            </bean>
        </property>
    </bean>

</beans>
RunSetterDIMain.java
package com.vishal.spring.di.setter;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunSetterDIMain {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        Person person = (Person)context.getBean("person");

        person.callPhone();

    }
}
```

```
}
```

## Output

```
Nov 06, 2014 2:10:30 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@1d381d2: startup date [Thu Nov 06 14:10:30 IST 2014]
Nov 06, 2014 2:10:30 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 2:10:30 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@37d830: defining beans [pers
Inside Phone Constructor
setting Phone by setter method
Call Phone
Calling to 9876543210
```



## Collection and References

You have seen how to configure primitive data type using value attribute and object references using ref attribute of the <property> tag in your Bean configuration file. Both the cases deal with passing singular value to a bean.

Now what about if you want to pass plural values like Java Collection types List, Set, Map, and Properties. To handle the situation, Spring offers four types of collection configuration elements which are as follows:

Element	Description
<list>	This helps in wiring ie injecting a list of values, allowing duplicates.
<set>	This helps in wiring a set of values but without any duplicates.
<map>	This can be used to inject a collection of name-value pairs where name and value can be of any type.
<props>	This can be used to inject a collection of name-value pairs where the name and value are both Strings.

## Program

### Person.java

```
package com.vishal.spring.di.collection;

public class Person {

    private String name;
    private String address;
    private int age;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

```
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "Person [name=" + name + ", address=" + address + ", age=" + age
               + "]";
    }
}

Customer.java
package com.vishal.spring.di.collection;

import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.Set;

public class Customer {

    private List<Object> lists;
    private Set<Object> sets;
    private Map<Object, Object> maps;
    private Properties pros;

    public List<Object> getLists() {
        return lists;
    }

    public void setLists(List<Object> lists) {
        this.lists = lists;
    }

    public Set<Object> getSets() {
        return sets;
    }

    public void setSets(Set<Object> sets) {
        this.sets = sets;
    }

    public Map<Object, Object> getMaps() {
        return maps;
    }
}
```

```
}

public void setMaps(Map<Object, Object> maps) {
    this.maps = maps;
}

public Properties getPros() {
    return pros;
}

public void setPros(Properties pros) {
    this.pros = pros;
}

@Override
public String toString() {
    return "Customer [lists=" + lists + ", sets=" + sets + ", maps=" + maps
        + ", pros=" + pros + "]";
}

}

Bean.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="customer" class="com.vishal.spring.di.collection.Customer">
        <property name="lists">
            <list>
                <value>1</value>
                <ref bean="person" />
                <bean class="com.vishal.spring.di.collection.Person">
                    <property name="name" value="Vishal" />
                    <property name="address" value="Satellite" />
                    <property name="age" value="23" />
                </bean>
            </list>
        </property>

        <property name="sets">
            <set>
                <value>1</value>
                <ref bean="person" />
                <bean class="com.vishal.spring.di.collection.Person">
                    <property name="name" value="Vishal" />
                    <property name="address" value="Satellite" />
```

```

        <property name="age" value="23" />
    </bean>
</set>
</property>

<property name="maps">
    <map>
        <entry key="Key 1" value="1" />
        <entry key="Key 2" value-ref="person" />
        <entry key="Key 3">
            <bean class="com.vishal.spring.di.collection.Person">
                <property name="name" value="Vishal" />
                <property name="address" value="Satellite" />
                <property name="age" value="23" />
            </bean>
        </entry>
    </map>
</property>

<property name="pros">
    <props>
        <prop key="admin">admin@vishal.com</prop>
        <prop key="support">support@vishal.com</prop>
    </props>
</property>
</bean>

<bean id="person" class="com.vishal.spring.di.collection.Person">
    <property name="name" value="Vishal Shah" />
    <property name="address" value="CGRoad" />
    <property name="age" value="24" />
</bean>

```

</beans>

## RunConstructorDIMain.java

**package** com.vishal.spring.di.collection;

**import** org.springframework.context.ApplicationContext;  
**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public class** RunConstructorDIMain {

**public static void** main(String[] args) {

        ApplicationContext context = **new** ClassPathXmlApplicationContext("Beans.xml");

        Customer customer = (Customer)context.getBean("customer");

```

    //System.out.println(customer);
    System.out.println("List : " + customer.getLists());
    System.out.println("Set : " + customer.getSets());
    System.out.println("Map : " + customer.getMaps());
    System.out.println("Props : " + customer.getPros());
}

}

```

### **Output**

```

Nov 06, 2014 12:00:28 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@afa68a: startup date [Thu Nov 06 12:00:28
Nov 06, 2014 12:00:28 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 12:00:29 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1c96492: defining b
List : [1, Person [name=Vishal Shah, address=CGRoad, age=24], Person [name=Vishal, address=Satellite, age=23]]
Set : [1, Person [name=Vishal Shah, address=CGRoad, age=24], Person [name=Vishal, address=Satellite, age=23]]
Map : {Key 1=1, Key 2=Person [name=Vishal Shah, address=CGRoad, age=24], Key 3=Person [name=Vishal, address=Satellite, age=23]}
Props : {admin=admin@admin@vishal.com, support=support@vishal.com}

```



### **Spring Auto-Wiring**

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only. The Spring container can autowire relationships between collaborating beans without using <constructor-arg> and <property> elements which helps cut down on the amount of XML configuration you write for a big Spring based application.

**Advantage:** It requires the less code because we don't need to write the code to inject the dependency.

**Disadvantage:** No control of programmer and It can't be used for primitive and string values.

There are following autowiring modes which can be used to instruct Spring container to use autowiring for dependency injection. You use the autowire attribute of the <bean/> element to specify autowire mode for a bean definition.

Mode	Description
No	It is the default autowiring mode. It means no autowiring bydefault.
byname	The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
byType	The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
Constructor	The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.
Autodetect	Spring first tries to wire using autowire by constructor, if it does not work, Spring tries to autowire by byType. It is deprecated since Spring 3

### Program - byName

**Person.java**

```
package com.vishal.spring.auto.byname;

public class Person {

    private String name;
    private Phone phone;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Phone getPhone() {
        return phone;
    }

    public void setPhone(Phone phone) {
        this.phone = phone;
    }

    public void callPhone(){
        System.out.println("Call to "+name);
        phone.call();
    }

}
```

**Phone.java**

```
package com.vishal.spring.auto.byname;

public class Phone {

    public Phone() {
        // TODO Auto-generated constructor stub
        System.out.println("Phone Constructor");
    }

    private String phoneNumber;

    public String getPhoneNumber() {
        return phoneNumber;
    }

}
```

```
public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}

public void call() {
    // TODO Auto-generated method stub
    System.out.println("Call on " + phoneNumber);
}

}

Bean.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="person" class="com.vishal.spring.auto.byname.Person" autowire="byName">
        <property name="name" value="Vishal Shah" />
        <!-- <property name="phone" ref="phone" /> --><!-- No need to write property auto
wire by the same name -->
    </bean>

    <bean id="phone" class="com.vishal.spring.auto.byname.Phone">
        <property name="phoneNumber" value="9876543210"></property>
    </bean>

</beans>
RunAutoByNameMain.java
package com.vishal.spring.auto.byname;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunAutoByNameMain {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "Beans.xml");

        Person person = (Person) context.getBean("person");

        person.callPhone();

    }
}
```

```
}
```

## Output

```
Nov 06, 2014 2:20:57 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@179953c: startup date [Thu Nov 06 14:20:57 IST 2014]
Nov 06, 2014 2:20:57 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 2:20:57 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1ba0fd: defining beans [per
Phone Constructor
Call to Vishal Shah
Call on 9876543210
```



## Program – byType

### Person.java

```
package com.vishal.spring.auto.bytype;

public class Person {

    private String name;
    private Phone phone;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Phone getPhone() {
        return phone;
    }

    public void setPhone(Phone phone) {
        this.phone = phone;
    }

    public void callPhone(){
        System.out.println("Call to Phone");
        phone.call();
    }
}
```

### Phone.java

```
package com.vishal.spring.auto.bytype;
```

```
public class Phone {  
  
    public Phone() {  
        // TODO Auto-generated constructor stub  
        System.out.println("Phone Constructor");  
    }  
  
    private String phoneNumber;  
  
    public String getPhoneNumber() {  
        return phoneNumber;  
    }  
  
    public void setPhoneNumber(String phoneNumber) {  
        this.phoneNumber = phoneNumber;  
    }  
  
    public void call() {  
        // TODO Auto-generated method stub  
        System.out.println("Call on " + phoneNumber);  
    }  
}
```

## Bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">  
  
    <bean id="person" class="com.vishal.spring.auto.bytype.Person" autowire="byType">  
        <!-- <property name="name" value="Vishal Shah" /> --> <!-- No need to write property  
        auto wire by the same name -->  
        <property name="phone" ref="phone" />  
    </bean>  
  
    <bean id="phone" class="com.vishal.spring.auto.bytype.Phone">  
        <property name="phoneNumber" value="9876543210"></property>  
    </bean>  
  
</beans>
```

## RunAutoByTypeMain.java

```
package com.vishal.spring.auto.bytype;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class RunAutoByTypeMain {  
  
    public static void main(String[] args) {  
  
        ApplicationContext context = new ClassPathXmlApplicationContext(  
            "Beans.xml");  
  
        Person person = (Person) context.getBean("person");  
  
        person.callPhone();  
  
    }  
}
```

## Output

```
|Nov 06, 2014 2:26:52 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh  
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@179953c: startup date [Thu Nov 06 14:26:52 IST 2014]  
Nov 06, 2014 2:26:53 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions  
INFO: Loading XML bean definitions from class path resource [Beans.xml]  
Nov 06, 2014 2:26:53 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons  
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1e2b9d1: defining beans [per  
Phone Constructor  
Call to Phone  
Call on 9876543210
```

## Program - constructor

### Person.java

```
package com.vishal.spring.auto.byconstructor;
```

```
public class Person {  
  
    private String name;  
    private Phone phone;  
  
    public Person(String name, Phone phone) {  
        System.out.println("Person Constructor");  
        this.name = name;  
        this.phone = phone;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public Phone getPhone() {  
        return phone;  
    }  
}
```

```
public void callPhone(){
    System.out.println("Call to "+name);
    phone.call();
}

}
Phone.java
package com.vishal.spring.auto.byconstructor;

public class Phone {

    private String phoneNumber;

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public Phone(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public void call() {
        // TODO Auto-generated method stub
        System.out.println("Call on " + phoneNumber);
    }

}
Bean.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="person" class="com.vishal.spring.auto.byconstructor.Person"
          autowire="constructor">
        <constructor-arg value="Vishal Shah" />
        <!-- <constructor-arg ref="phone" /> --> <!-- if you are not autowired by
constructor[autowire="constructor"] by writting
                this breacket code.. then you can write -->
    </bean>

    <bean id="phone" class="com.vishal.spring.auto.byconstructor.Phone">
        <constructor-arg value="9876543210" />
    </bean>
```

```
</beans>
RunAutoByTypeConstructor.java
package com.vishal.spring.auto.byconstructor;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RunAutoByTypeConstructor {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext(
                "Beans.xml");

        Person person = (Person) context.getBean("person");

        person.callPhone();

    }

}
Output
Nov 06, 2014 2:29:19 PM org.springframework.context.support.AbstractApplicationContext prepareRefresh
INFO: Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@179953c: startup date [Thu Nov 06 14:29:19 IST 2014]
Nov 06, 2014 2:29:19 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [Beans.xml]
Nov 06, 2014 2:29:19 PM org.springframework.beans.factory.support.DefaultListableBeanFactory preInstantiateSingletons
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@1dd1a31: defining beans [per
Person Constructor
Call to Vishal Shah
Call on 9876543210
```

## Spring ORM

Spring provides API to easily integrate Spring with ORM frameworks such as Hibernate, JPA(Java Persistence API), JDO(Java Data Objects), Oracle Toplink and iBATIS. We can simply integrate hibernate application with spring application. In hibernate frameworks, we provide all the database information hibernate.cfg.xml file. But if we are going to integrate the hibernate application with spring, we don't need to create the hibernate.cfg.xml file. We can provide all the information in the Bean.xml (applicationContext.xml) file.

Advantage of ORM Frameworks with Spring:

The Spring framework provides HibernateTemplate class, so you don't need to follow so many steps like create Configuration, BuildSessionFactory, Session, beginning and committing transaction etc. So it saves a lot of code.

- **Less coding is required:** By the help of Spring framework, you don't need to write extra codes before and after the actual database logic such as getting the connection, starting transaction, committing transaction, closing connection etc.
- **Easy to test:** Spring's IoC approach makes it easy to test the application.
- **Better exception handling:** Spring framework provides its own API for exception handling with ORM framework.
- **Integrated transaction management:** By the help of Spring framework, we can wrap our mapping code with an explicit template wrapper class or AOP style method interceptor.

## Step to Create Program

Let's see what the simple steps are for hibernate and spring integration:

1. Create Database.
2. Configure Jar files of Spring and Hibernate
  - a. org.springframework.aop-3.1.0.RELEASE
  - b. org.springframework.asm-3.1.0.RELEASE
  - c. org.springframework.aspects-3.1.0.RELEASE
  - d. org.springframework.beans-3.1.0.RELEASE
  - e. org.springframework.context.support-3.1.0.RELEASE
  - f. org.springframework.context-3.1.0.RELEASE
  - g. org.springframework.core-3.1.0.RELEASE
  - h. org.springframework.expression-3.1.0.RELEASE
  - i. org.springframework.jdbc-3.1.0.RELEASE
  - j. org.springframework.orm-3.1.0.RELEASE
  - k. org.springframework.oxm-3.1.0.RELEASE
  - l. org.springframework.transaction-3.1.0.RELEASE
  - m. antlr-2.7.2
  - n. commons-logging-1.1.1
  - o. mysql-connector-java-5.1.22-bin
  - p. commons-collections-3.1
  - q. commons-io-2.0.1
  - r. dom4j-1.6.1
  - s. hibernate-core- 3.3.1.GA
  - t. javassist-3.9.0.GA
  - u. log4j-1.2.15
  - v. slf4j-api-1.4.2
  - w. slf4j-log4j12-1.4.2
  - x. jta-1.3.1
3. Create Bean.xml file it contains to import DataSource.xml, Hibernate.xml, Person.xml.
  - a. DataSource.xml contains the Datasource coding
  - b. Hibernate.xml contains the Hibernate Configuration properties with Session Factory and Resource Mapping

- c. Person.xml contains bean mapping of the Data Access Object (DAO) and Business Object (BO)
- 4. Create Person.java file It is the persistent class
- 5. Create person.hbm.xml file It is the mapping file of the persistence class.
- 6. Create PersonDAO.java file is interface for declaring the methods. It implementing to PersonDAOImpl file. It is the DAOImpl class that uses HibernateTemplate by extending the HibernateDaoSupport.
- 7. Create PersonBO.java file is interface for decalring the methods. It implementing to PersonBOImpl.java file. It is the BOImpl class that uses PersonDAOImpl class methods.
- 8. Create AddPerson.java, FindPerson.java, ListPerson.java ,DeletePerson.java UpdatePerson.java. These files call the method of PersonBO interface.

## Program Based on Above Steps

### Bean.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<import resource="DataSource.xml"/>
<import resource="Hibernate.xml"/>
<import resource="Person.xml"/>
</beans>
```

### DataSource.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql://localhost:3306/spring"></property>
    <property name="username" value="root"></property>
    <property name="password" value=""></property>
</bean>
```

```
</beans>
```

### Hibernate.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>

    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>

    <property name="mappingResources">
        <list>
            <value>com/vishal/spring/hibernate/inte/Person.hbm.xml</value>
        </list>
    </property>
</bean>
```

```
</beans>
```

## Person.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <bean id="personBo" class="com.vishal.spring.hibernate.inte.PersonBoImpl">
        <property name="personDAO" ref="personDAO"></property>
    </bean>

    <bean id="personDAO" class="com.vishal.spring.hibernate.inte.PersonDAOImpl">
        <property name="sessionFactory" ref="sessionFactory"></property>
    </bean>
</beans>
```

## Person.java

```
package com.vishal.spring.hibernate.inte;

import java.io.Serializable;

public class Person implements Serializable{

    private int pid;
    private String pname;
    private String paddress;
    public int getPid() {
```

```

        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    public String getPaddress() {
        return paddress;
    }
    public void setPaddress(String paddress) {
        this.paddress = paddress;
    }
}

Person.hbm.xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated Oct 29, 2014 4:13:18 PM by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="com.vishal.spring.hibernate.inte.Person" table="PERSON">
        <id name="pid" type="int">
            <column name="PID" />
            <generator class="increment" />
        </id>
        <property name="pname" type="java.lang.String">
            <column name="PNAME" />
        </property>
        <property name="paddress" type="java.lang.String">
            <column name="PADDRESS" />
        </property>
    </class>
</hibernate-mapping>
PersonDAO.java
package com.vishal.spring.hibernate.inte;

import java.util.List;

public interface PersonDAO {

    public void add(Person person);
    public void update(Person person);
    public void delete(int pid);
}

```

```
public Person find(int pid);
public List<Person> list();

}

PersonDAOImpl.java
package com.vishal.spring.hibernate.inte;

import java.util.List;

import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

public class PersonDAOImpl extends HibernateDaoSupport implements PersonDAO{

    @Override
    public void add(Person person) {
        // TODO Auto-generated method stub
        getHibernateTemplate().save(person);
    }

    @Override
    public void update(Person person) {
        // TODO Auto-generated method stub
        getHibernateTemplate().update(person);
    }

    @Override
    public void delete(int pid) {
        // TODO Auto-generated method stub
        HibernateTemplate template = getHibernateTemplate();
        Person person = (Person)template.get(Person.class, pid);
        template.delete(person);
    }

    @Override
    public Person find(int pid) {
        // TODO Auto-generated method stub
        Person person = (Person)getHibernateTemplate().get(Person.class, pid);
        return person;
    }

    @Override
    public List<Person> list() {
        // TODO Auto-generated method stub

        List<Person> persons = getHibernateTemplate().find("from Person");

        return persons;
    }
}
```

```
}

}

PersonBO.java
package com.vishal.spring.hibernate.inte;

import java.util.List;

public interface PersonBo {

    public void add(Person person);
    public void update(Person person);
    public void delete(int pid);
    public Person find(int pid);
    public List<Person> list();

}

PersonBOImpl.java
package com.vishal.spring.hibernate.inte;

import java.util.List;

public class PersonBolmpl implements PersonBo {

    private PersonDAO personDAO;

    public PersonDAO getPersonDAO() {
        return personDAO;
    }

    public void setPersonDAO(PersonDAO personDAO) {
        this.personDAO = personDAO;
    }

    @Override
    public void add(Person person) {
        // TODO Auto-generated method stub
        this.personDAO.add(person);
    }

    @Override
    public void update(Person person) {
        // TODO Auto-generated method stub
        this.personDAO.update(person);
    }

    @Override
    public void delete(int pid) {
```

```
// TODO Auto-generated method stub
this.personDAO.delete(pid);
}

@Override
public Person find(int pid) {
    // TODO Auto-generated method stub

    return this.personDAO.find(pid);
}

@Override
public List<Person> list() {
    // TODO Auto-generated method stub
    return this.personDAO.list();
}

}

AddPerson.java
package com.vishal.spring.hibernate.inte.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.vishal.spring.hibernate.inte.Person;
import com.vishal.spring.hibernate.inte.PersonBo;

public class AddPerson {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        PersonBo personBo = (PersonBo)context.getBean("personBo");

        Person person = new Person();
        person.setPaddress("Satellite");
        person.setPname("Vish");

        personBo.add(person);

        System.out.println("Add Person Data... ");

    }

}
```

## Output

```
|log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select max(PID) from PERSON
Hibernate: insert into PERSON (PNAME, PADDRESS, PID) values (?, ?, ?)
Add Person Data...
```

## FindPerson.java

```
package com.vishal.spring.hibernate.inte.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.vishal.spring.hibernate.inte.Person;
import com.vishal.spring.hibernate.inte.PersonBo;

public class FindPerson {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        PersonBo personBo = (PersonBo)context.getBean("personBo");

        Person person = personBo.find(1);

        System.out.println("id : " + person.getPid() + " name : " + person.getPname() + " address :
" + person.getPaddress());
    }
}
```

## Output

```
|log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select person0_.PID as PID0_0_, person0_.PNAME as PNAME0_0_, person0_.PADDRESS as PADDRESS0_0_ from PERSON person
id : 1 name : Vishal Shah address : Satellite
```

## UpdatePerson.java

```
package com.vishal.spring.hibernate.inte.main;

import org.springframework.context.ApplicationContext;
```

```
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.vishal.spring.hibernate.inte.Person;
import com.vishal.spring.hibernate.inte.PersonBo;

public class UpdatePerson {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        PersonBo personBo = (PersonBo)context.getBean("personBo");

        Person person = new Person();
        person.setPaddress("CGRoad");
        person.setPname("VKS");
        person.setPid(2);
        personBo.update(person);

        System.out.println("Update Person Data...");

    }

}
```

## Output

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: update PERSON set PNAME=?, PADDRESS=? where PID=?
Update Person Data...
```



```
ListPerson.java
package com.vishal.spring.hibernate.inte.main;

import java.util.List;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.vishal.spring.hibernate.inte.Person;
import com.vishal.spring.hibernate.inte.PersonBo;

public class ListPerson {

    public static void main(String[] args) {
```

```
ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

PersonBo personBo = (PersonBo) context.getBean("personBo");
List<Person> persons = personBo.list();
for (Person person : persons)
    System.out.println("id : " + person.getPid() + " name : "
        + person.getPname() + " address : " + person.getPaddress());
}

}
```

**Output**

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select person0_.PID as PID0_, person0_.PNAME as PNAME0_, person0_.PADDRESS as PADDRESS0_ from PERSON person0_
id : 1 name : Vishal Shah address : Satellite
id : 2 name : VKS address : CGRoad
id : 3 name : Test Test address : Testing
id : 4 name : Vish address : Satellite
```

**DeletePerson.java**

```
package com.vishal.spring.hibernate.inte.main;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.vishal.spring.hibernate.inte.PersonBo;

public class DeletePerson {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        PersonBo personBo = (PersonBo)context.getBean("personBo");

        personBo.delete(3);

        System.out.println("Delete Person Data...");

    }
}
```

**Output**

```
log4j:WARN No appenders could be found for logger (org.springframework.core.env.StandardEnvironment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select person0_.PID as PID0_0_, person0_.PNAME as PNAME0_0_, person0_.PADDRESS as PADDRESS0_0_ from PERSON person
Hibernate: delete from PERSON where PID=?
Delete Person Data...
```

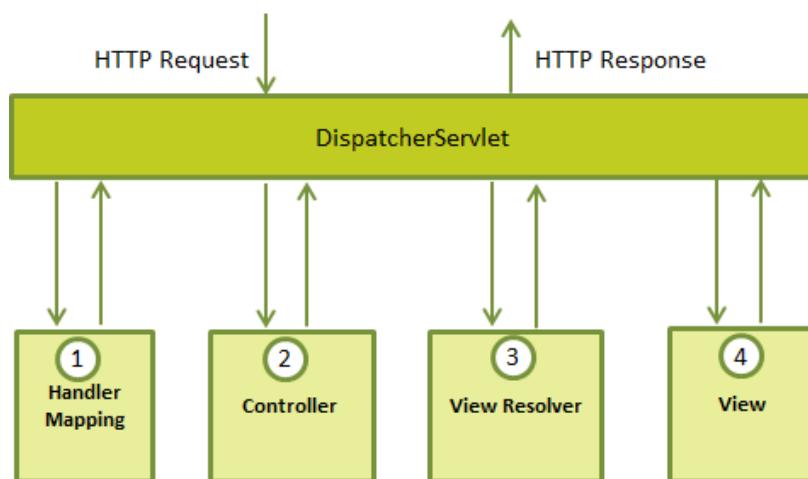
## Spring MVC

The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

- The **Model** encapsulates the application data and in general they will consist of POJO.
- The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
- The **Controller** is responsible for processing user requests and building appropriate model and passes it to the view for rendering.

## Request flow of Spring MVC

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that handles all the HTTP requests and responses. The request processing workflow of the Spring Web MVC DispatcherServlet is illustrated in the following diagram:



1. After receiving an HTTP request, DispatcherServlet advice the HandlerMapping to call the appropriate Controller.
2. The Controller takes the request and calls the appropriate service methods based on used GET or POST method.

- a. The service method will set model data based on defined business logic and returns view name to the DispatcherServlet.
  - b. The service method will set ModelAndView object to define the model data, view name and commander name of the form data.
3. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request.
  4. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the web browser.

Basically, the entire incoming request is intercepted by the DispatcherServlet that works as the front controller. The DispatcherServlet gets entry of handler mapping from the xml file and forwards the request to the controller. The controller returns an object of ModelAndView (model data with returning view name and command name). The DispatcherServlet checks the entry of view resolver in the xml file and invokes the specified view component.

## Framework Initiation/Configuration

You need to configure jar file in lib folder of J2EE application, if you are using Ant based project. And whenever you are using Maven based project that time you need to configure pom dependency in pom.xml file. You have to configure list of the jar files/pom dependencies are: antlr-2.7.2, commons-logging-1.1.1, jstl-1.2, org.springframework-3.1.0.RELEASE package jar files.

You need to map requests that you want the DispatcherServlet to handle, by using a URL mapping in the web.xml file. The following is an example to show declaration and mapping for SpringHelloWorldWeb example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
    id="WebApp_ID" version="2.5">
    <display-name>SpringHelloWorldWeb</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <!-- Servlet Dispatcher Configuration -->

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-servlet.xml</param-value>
    </init-param>
    </servlet>

    <servlet-mapping>
```

```
<servlet-name>spring</servlet-name>
<url-pattern>*.vishal</url-pattern>
</servlet-mapping>
```

```
</web-app>
```

In web.xml, you have to declare DispatcherServlet class as front (single) controller. As well as you have to optional to define init-param tag with param-name and param-value, whenever spring configuration file name is spring-servlet.xml. If you are modified this file name then you must be declare init-param tag with param-name as contextConfigLocation and param-value is to location of the spring-servlet.xml file. If you don't want to declare this init-param, then you will get page not found (404 errors) or mapping errors. In servlet-mapping, url-pattern of the incoming HTTP request and outgoing HTTP responses ending with ".vishal". You can modify as per your convenient for eg. \*.jsp, \*.htm, /, \*.spring. Alternative way to when you are not declaring the init-param tag inside servlet that time, you can define load-on-startup tag as value 1 when you put the context-param tag to define for param-name as contextConfigLocation and param-value is to location of the spring-servlet.xml file.

Now, we have to define the spring-servlet.xml file and placed near to web.xml file it means inside WEB-INF folder. Here below code of spring-servlet.xml is:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.web.mvc" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

Following are the important points about spring-servlet.xml file:

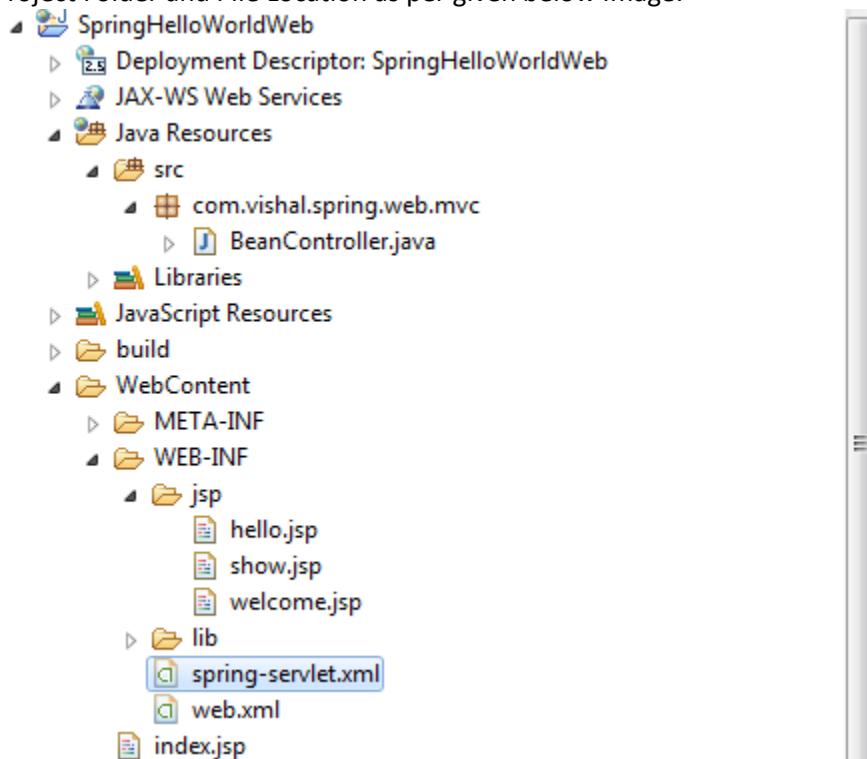
- The [servlet-name]-servlet.xml file will be used to create the beans defined, overriding the definitions of any beans defined with the same name in the global scope.
- The <mvc:annotation-driven/> tag will be use to support conversation service, number formatting of the field by @NumberFormat, as well as formatting Date, Calendar, Long using

@DateTimeFormat. Also validating the @Controller request field by @Valid /@Validator annotations. Support XML and JSON reading and writing the content in controller methods.

- The <context:component-scan...> tag will be used to activate Spring MVC annotation scanning capability which allows to make use of annotations like @Controller and @RequestMapping etc.
- The InternalResourceViewResolver will have rules defined to resolve the view names. As per the above defined rule, a logical view named hello is delegated to a view implementation located at/WEB-INF/jsp/hello.jsp .

Next section will show you how to create your actual components ie. Model, View and Controller.

Project Folder and File Location as per given below Image:



## Spring MVC Hello World using IDE

Let's see the simple example of spring 3 web MVC. There are given 7 steps for creating the spring MVC application. The steps are as follows:

1. Create the request page (optional)
2. Create the controller class
3. Provide the entry of controller in the web.xml file
4. Define the bean in the xml file
5. Display the message in the JSP page
6. Load the spring core and mvc jar files
7. Start server and deploy the project

## Spring configuration with Eclipse

The Spring IDE is an open-source project that provides a set of plugins for the Eclipse IDE. These plugins make the Eclipse IDE Spring-aware. After installing the Spring IDE plugins, your IDE understands your projects from the perspective of the Spring framework and provides you with a wide variety of additional features that make it easier and more convenient to work with Spring projects in Eclipse.

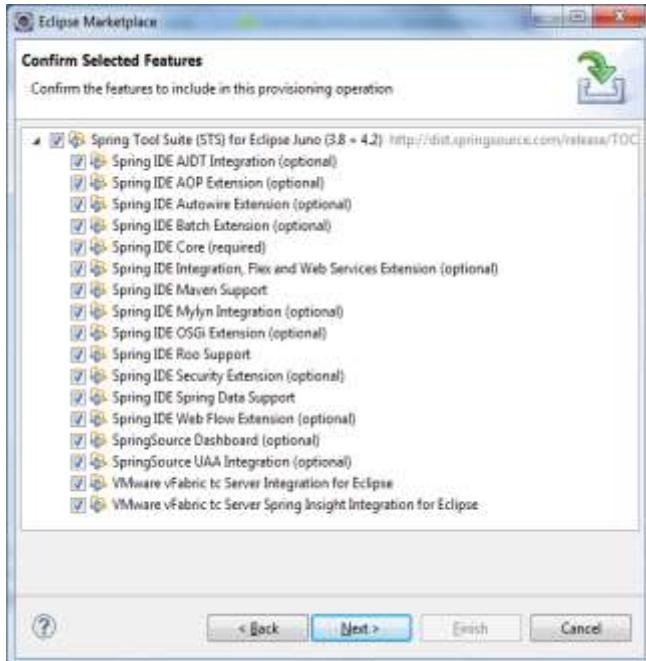
The IDE provides various wizards for creating Spring projects and getting started with Spring if you're a beginner. Of course, it can also be used on existing Spring projects. Once you have a project configured to be a Spring project, you can let the IDE know which Spring configuration files your projects uses (either those XML config files or your Spring JavaConfig classes that use the @Configuration annotation). From there on, you can benefit from the Spring-specific tooling support, including:

- Spring-specific content-assist and validation for your spring config files
- Spring-specific refactoring support for your spring config files
- Graphical visualization of your beans and their dependencies
- Graphical diagram-like editors for Spring Integration, Spring Batch, and Spring Webflow
- Advanced support for Spring aspects including pointcut visualization and navigation
- Integration with AJDT and Eclipse Mylyn tooling
- Support for Spring bean profiles and validation inside profiles
- Direct integration of the 'getting started' guides from <http://spring.io/guides>
- A number of example and reference applications to learn from
- Support for creating and using Spring Boot projects right within your IDE
- and more...

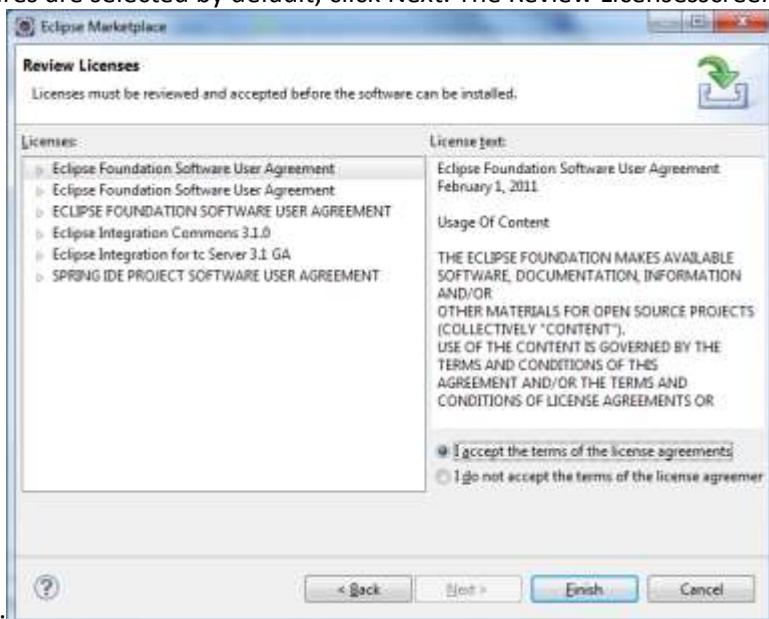
Classic way to configure, Eclipse IDE, click "Help" -> "Install New Software...". Type "<http://springide.org/updateSite>" to access the Spring IDE update site. **OR** Click Help > Eclipse Marketplace... from Eclipse's main menu. The Eclipse Marketplace dialog appears, type Spring Tool Suite or STS into the Find textfield and hit Enter. Eclipse will send query to its server and display results as shown below:



Select the version that matches your Eclipse's version and click Install button. Here we select Spring Tool Suite (STS) for Eclipse Juno (3.8 + 4.2). It takes a while for Eclipse to fetch the details and show the features of STS as shown below:

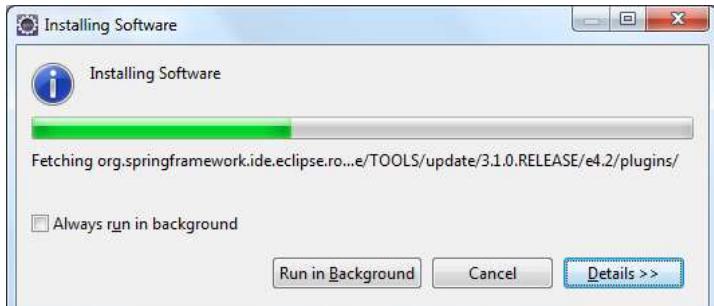


All features are selected by default, click Next. The Review Licenses screen

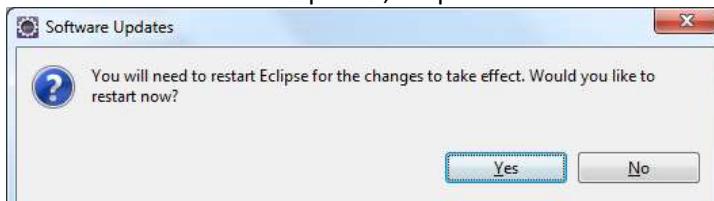


appears:

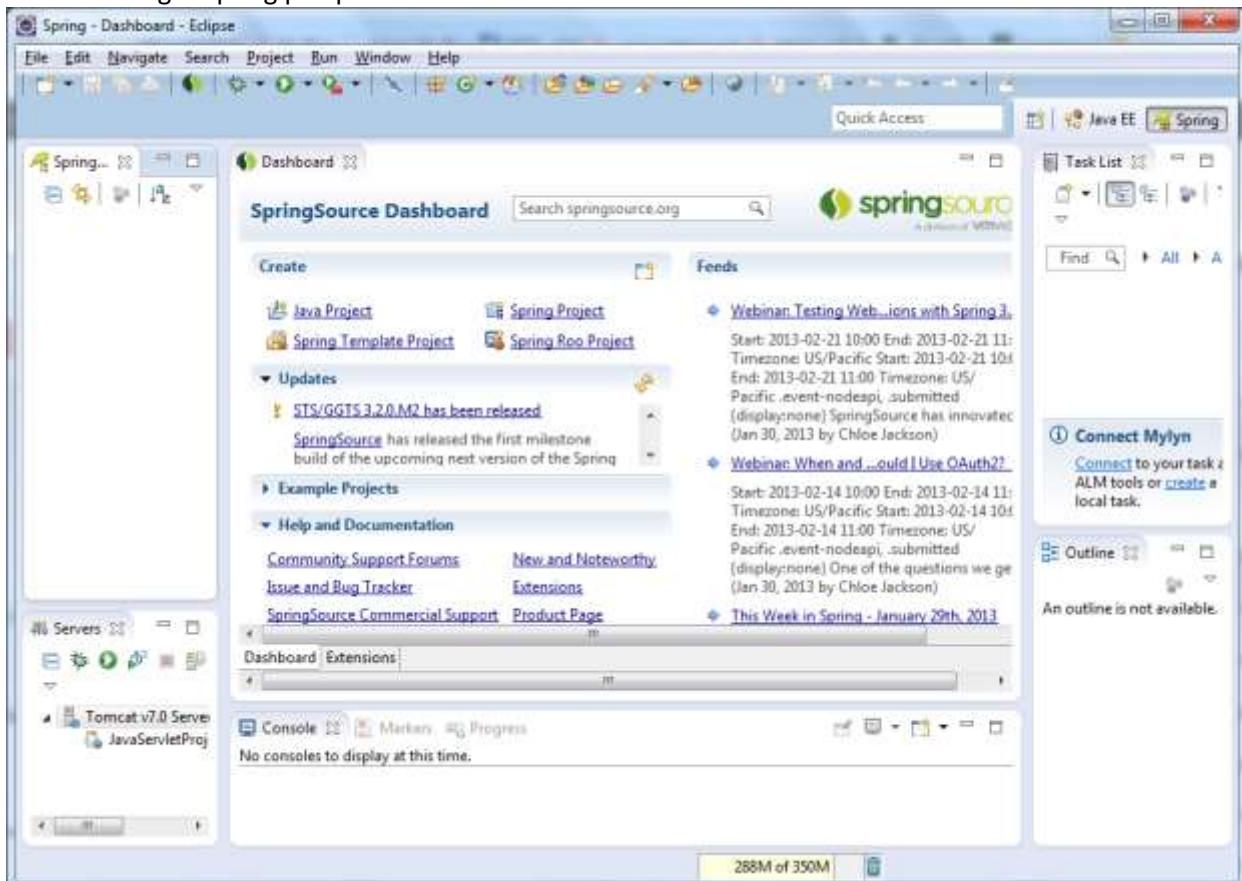
Select I accept the terms of the license agreements, and then click Finish. Eclipse will install STS and display the progress:



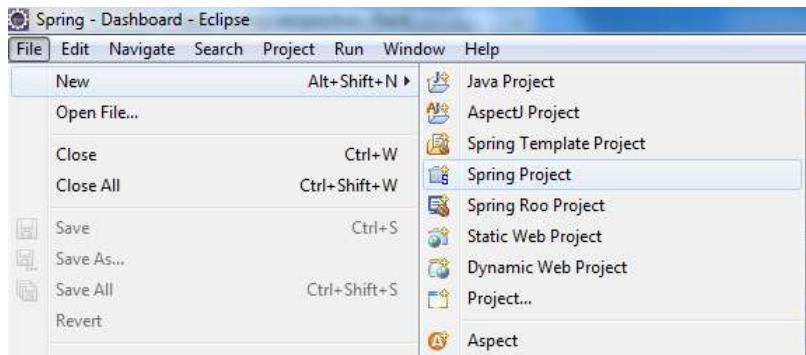
After the installation completed, Eclipse asks to restart the IDE:



Click Yes to restart the IDE. When Eclipse restarted, you will see some Spring natures are added to the IDE. The first thing is Spring perspective:

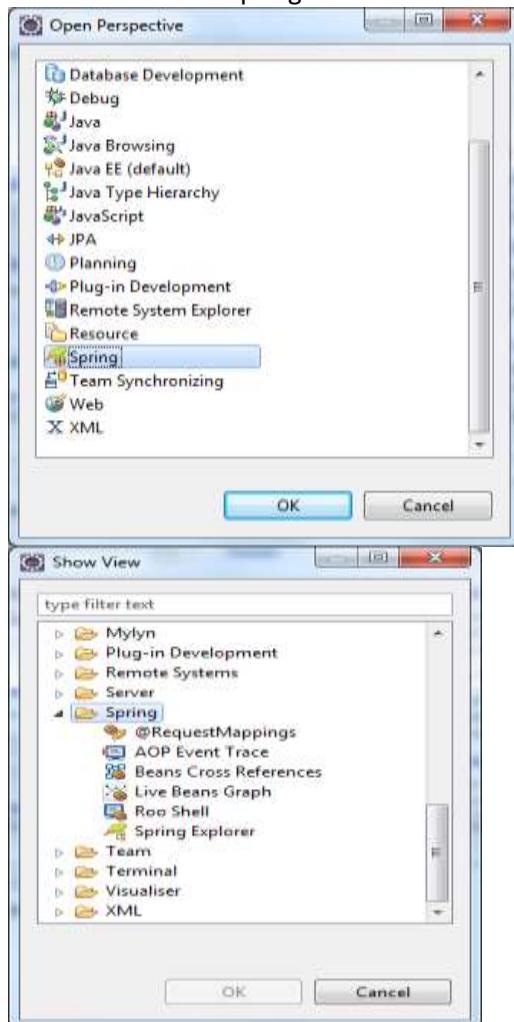


The menu File > New now comes with some Spring projects:

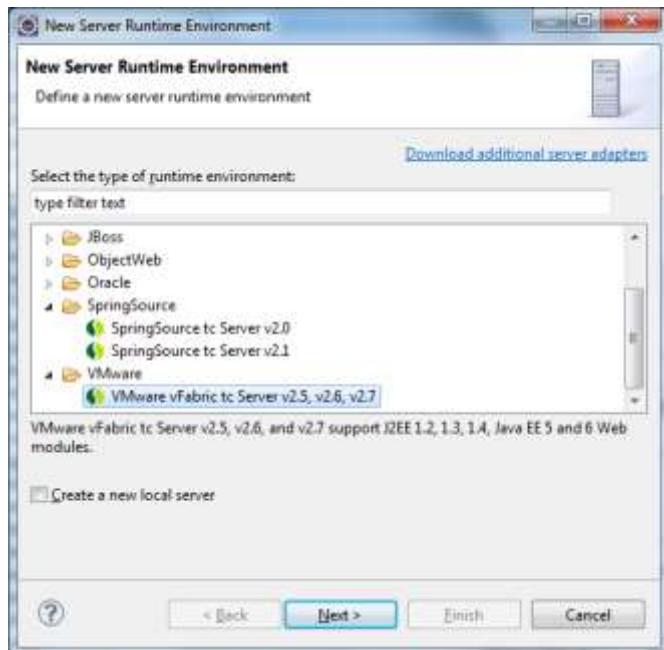


The menu Window > Open Perspective now has Spring perspective:

We can also show Spring views from the menu Window > Show View:



And new options in the New Server Runtime Environment dialog:



## Step to create Spring MVC practical

1. Load spring-core and spring-web jar file inside the jar folder of WEB-INF
2. Add DispatcherServlet in web.xml as servlet tag define with url-pattern
3. Add <servlet-name>-servlet.xml file eg. spring-servlet.xml declare with the InternalResourceViewResolver with prefix and suffix values.
4. Add mvc:annotation-driven and context:component-scan with base package where you are putting the controllers
5. Create index.jsp page with anchor tag for linking the url with proper suffix of given in url-pattern of servlet-mapping tag in web.xml file.
6. Create Controller with @Controller annotation and define the method with incoming request mapping by @RequestMapping("/xxx").
7. Method can return string value, which name is view page name or ModelAndView object with view page name, commander name with its object.
8. Create view page (hello.jsp and welcome.jsp) inside the given InternalResourceViewResolver's prefix location with appropriate message. This message show by using expression language(EL).

## Program

Based on the given above Framework Configuration topic you have to create jsp folder inside WEB-INF and put all the jsp pages inside the jsp folder because of our DispatcherServlet find this content inside the WEB-INF/jsp folde path because of we are define InternalResourceViewResolver in spring-servlet.xml file with prefix of "/WEB-INF/jsp/" with suffix ".jsp".

Now define the controller inside the package of "com.vishal.spring.web.mvc" due to this declare package name inside the scanning the component as base-package.

### BeanController.java

```
package com.vishal.spring.web.mvc;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class BeanController {

    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String control(ModelMap model) {
        // TODO Auto-generated method stub

        System.out.println("Control Method Call");

        model.addAttribute("message", "Hello World!!!");

        return "hello";
    }

    @RequestMapping(value = "/welcome", method = RequestMethod.GET)
    private String welcome(ModelMap model) {
        // TODO Auto-generated method stub

        System.out.println("Welcome Page Call");

        model.addAttribute("welcome", "Welcome to the Spring MVC");

        return "welcome";
    }
}

index.jsp (WebContent/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>
<a href="hello.vishal">Hello</a>
```

```
<a href="welcome.vishal">Welcome</a>

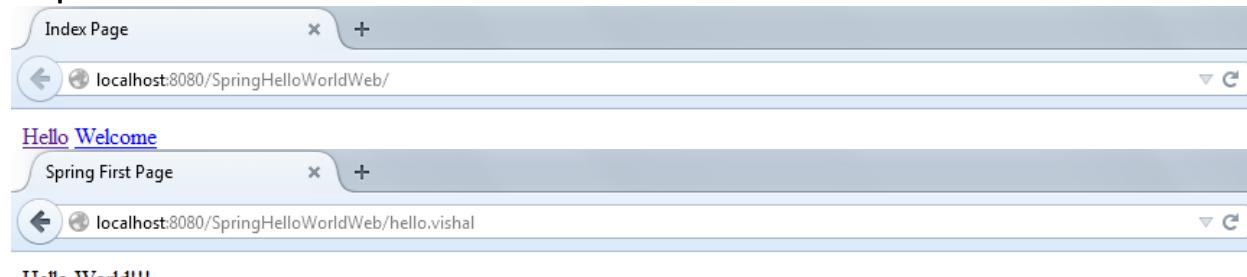
</body>
</html>
hello.jsp (WebContent /WEB-INF/jsp/hello.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring First Page</title>
</head>
<body>

${message}

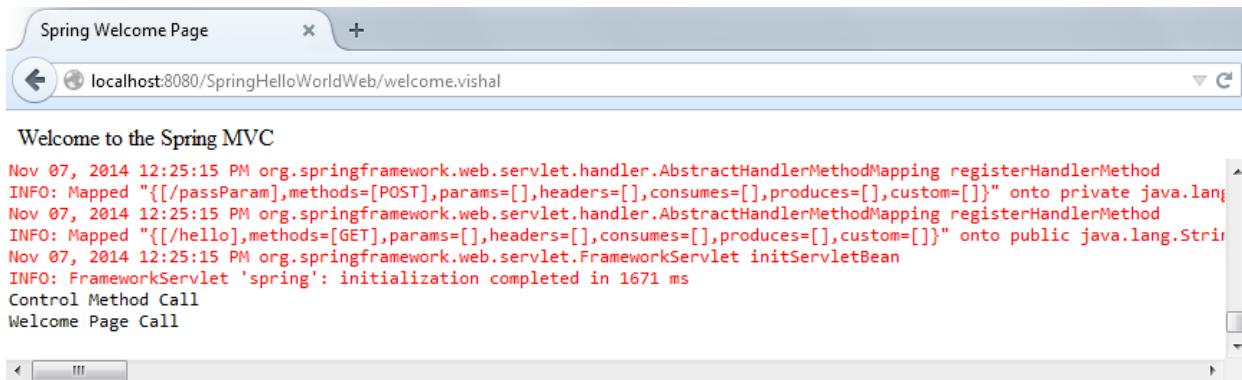
</body>
</html>
welcome.jsp(WebContent/WEB-INF/jsp/welcome.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Welcome Page</title>
</head>
<body>

${welcome}

</body>
</html>
```

**Output:**

Hello World!!!



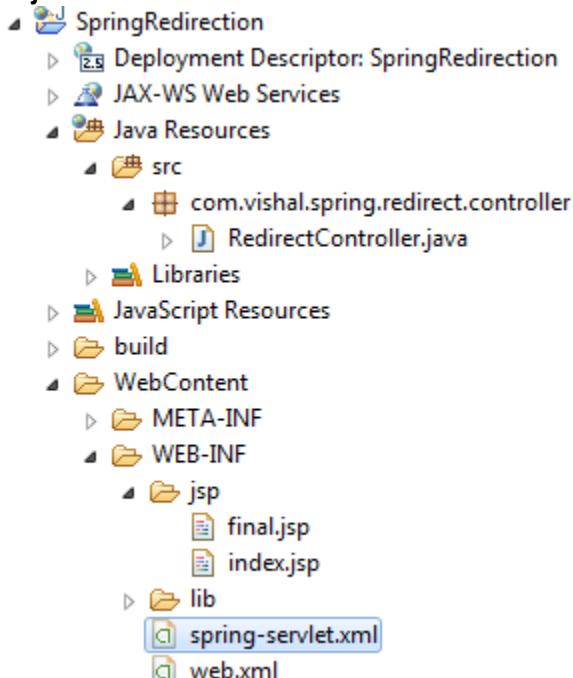
Welcome to the Spring MVC

```
Nov 07, 2014 12:25:15 PM org.springframework.web.servlet.handler.AbstractHandlerMapping registerHandlerMethod
INFO: Mapped "[{/passParam},methods=[POST],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto private java.lang.String org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping$HandlerWrapper.handleInternal(HttpServletRequest, HttpServletResponse)
Nov 07, 2014 12:25:15 PM org.springframework.web.servlet.handler.AbstractHandlerMapping registerHandlerMethod
INFO: Mapped "[{/hello},methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public java.lang.String org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping$HandlerWrapper.handleInternal(HttpServletRequest, HttpServletResponse)
Nov 07, 2014 12:25:15 PM org.springframework.web.servlet.FrameworkServlet initServletBean
INFO: FrameworkServlet 'spring': initialization completed in 1671 ms
Control Method Call
Welcome Page Call
```

## Spring Page Redirections

The following example show how to write a simple web based application which makes use of redirect to transfer a http request to another page. To start with it, let us have working Eclipse IDE in place and follow the following steps to develop a Dynamic Form based Web Application using Spring Web Framework:

### Project Folder and File Structure:



### web.xml (WebContent/WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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" id="WebApp_ID" version="2.5">
  <display-name>SpringRedirection</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
```

```
<servlet-name>spring</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>spring</servlet-name>
<url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
spring-servlet.xml (WebContent/WEB-INF/ spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.redirect.controller" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
Index.jsp (WebContent/WEB-INF/jsp/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>

<a href="redirect">Redirect Page Call</a>
```

```
</body>
</html>
RedirectController.java
package com.vishal.spring.redirect.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class RedirectController {

    @RequestMapping(value = "/", method = RequestMethod.GET)
    private String index() {
        // TODO Auto-generated method stub
        System.out.println("Index Page Call");
        return "index";
    }

    @RequestMapping(value = "/redirect", method = RequestMethod.GET)
    private String redirect() {
        // TODO Auto-generated method stub
        System.out.println("Redirecting to Final Page");
        return "redirect:finalCall";
    }

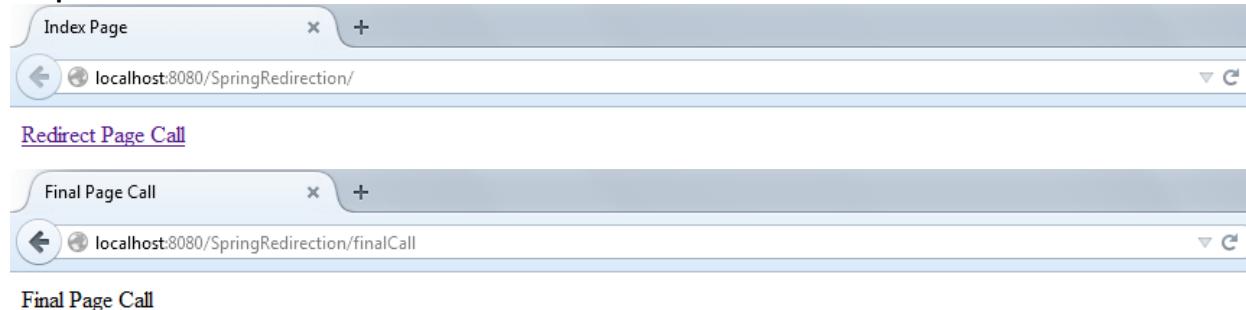
    @RequestMapping(value = "/finalCall", method = RequestMethod.GET)
    private String finalCall() {
        // TODO Auto-generated method stub
        System.out.println("Final Page Call");
        return "final";
    }
}

final.jsp (WebContent/WEB-INF/jsp/final.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Final Page Call</title>
</head>
<body>
```

Final Page Call

```
</body>
</html>
```

## Output

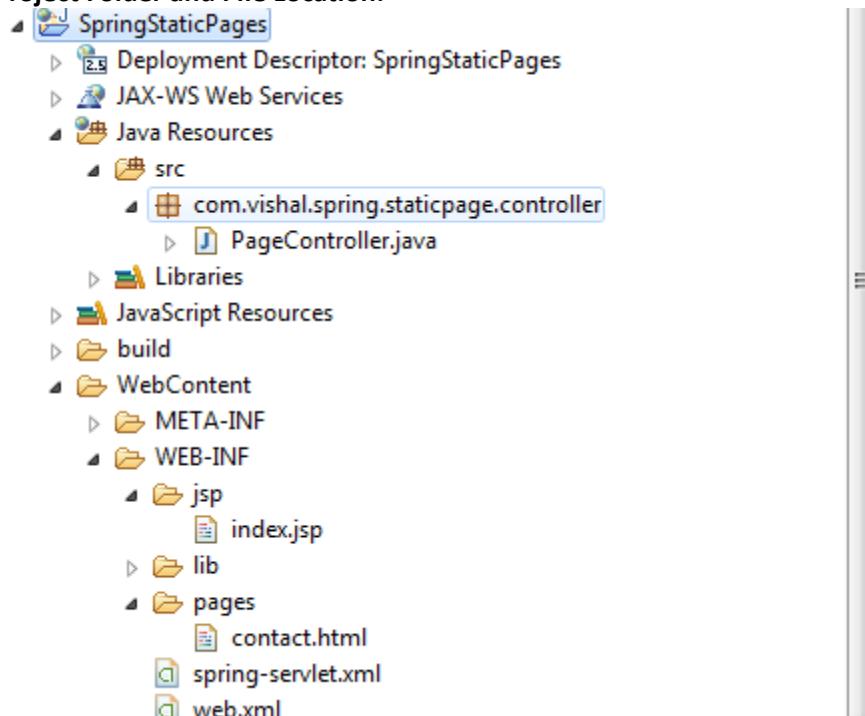


The screenshot shows a web browser with two tabs. The first tab is titled "Index Page" and displays the URL "localhost:8080/SpringRedirection/". The second tab is titled "Redirect Page Call" and displays the URL "localhost:8080/SpringRedirection/finalCall". Both tabs show the content "Final Page Call".

## Spring Static Page Redirection

The following example show how to write a simple web based application using Spring MVC Framework, which can access static pages along with dynamic pages with the help of <mvc:resources> tag. To start with it, let us have working Eclipse IDE in place and follow the following steps to develope a Dynamic Form based Web Application using Spring Web Framework:

### Project Folder and File Location:



### web.xml (WebContent/WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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/web-
    app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>SpringRedirection</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
```

```
</welcome-file-list>

<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>

</web-app>
spring-servlet.xml (WebContent/WEB-INF/ spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.staticpage.controller" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <!-- THis below line replacing the values as per the contents. This notation
        for registering the content of the HTML pages and pages(folder) to redirecting
        the pages by controller. -->

    <mvc:resources mapping="/pages/**" location="/WEB-INF/pages/" />

</beans>
Index.jsp (WebContent/WEB-INF/jsp/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page Call</title>
</head>
<body>

<a href="contact">Static Page Call</a>

</body>
</html>
```

## PageController.java

```
package com.vishal.spring.staticpage.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class PageController {

    @RequestMapping(value="/", method=RequestMethod.GET)
    private String index() {
        // TODO Auto-generated method stub

        return "index";
    }

    @RequestMapping(value="/contact", method=RequestMethod.GET)
    private String contact() {
        // TODO Auto-generated method stub
        System.out.println("Contact Us Page");
        return "redirect:/pages/contact.html";
    }

}

contact.html (WebCompoent/WEB-INF/pages/contact.html)
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Contact Us</title>
</head>
<body>
```

Address: 903, Samedh Complex,  
<br />

Next to Associated Petrol Pump,  
<br />  
CG Road, Ahmedabad, Gujarat 380009  
<br />  
Phone:099747 55006

```
</body>
</html>
```

## Output

The screenshot shows a Java IDE interface with two browser windows and a terminal window.

- Index Page Call:** A browser window titled "Index Page Call" showing the URL "localhost:8080/SpringStaticPages/". Below it is a link labeled "Static Page Call".
- Contact Us:** A browser window titled "Contact Us" showing the URL "localhost:8080/SpringStaticPages/pages/contact.html".
- Terminal:** A terminal window displaying Spring framework logs. The logs include:
  - INFO: Mapped "[/{contact}],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]" onto private java.lang.String org.springframework.web.servlet.handler.AbstractHandlerMethodMapping.registerHandlerMethod()
  - INFO: Mapped "[/],methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]" onto private java.lang.String org.springframework.web.servlet.handler.AbstractUrlHandlerMapping.registerHandler()
  - INFO: Mapped URL path [/pages/\*\*] onto handler 'org.springframework.web.servlet.resource.ResourceHttpRequestHandler#0'
  - INFO: FrameworkServlet 'spring': initialization completed in 3964 ms

## Spring MVC Web Forms

The following example show how to write a simple web based application which makes use of HTML forms using Spring Web MVC framework. To start with it, let us have working Eclipse IDE in place and follow the following steps to develop a Dynamic Form based Web Application using Spring Web Framework:

### Spring Form Handling

#### BeanController.java

```
package com.vishal.spring.web.mvc;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
```

```
@Controller
public class BeanController {

    @RequestMapping(value = "/formCall", method = RequestMethod.POST)
    private ModelAndView callForm(HttpServletRequest request,
                                  HttpServletResponse response) {

        String name = request.getParameter("name");
        String address = request.getParameter("address");

        System.out.println("Form Page Call");
        System.out.println("Name : " + name + " Address : " + address);

        ModelAndView model = new ModelAndView("show", "name", name);

        return model;
    }

    /**
     * You can use either callForm method by passing the formCall.vishal request or
     * getMultiParam method by passing passParam.vishal request. You can use any of way
     */
    @RequestMapping(value = "/passParam", method = RequestMethod.POST)
    private String getMultiParam(HttpServletRequest request, HttpServletResponse response
        ,ModelMap model) {
        // TODO Auto-generated method stub

        String name = request.getParameter("name");
        String address = request.getParameter("address");

        System.out.println("Multiple Paramters : Name : " + name + " Address : " + address);

        model.addAttribute("name", name);
        model.addAttribute("address", address);

        return "show";
    }

}

index.jsp (WebContent/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
```

```
<title>Index Page</title>
</head>
<body>
<a href="hello.vishal">Hello</a>
<a href="welcome.vishal">Welcome</a>

<form action="passParam.vishal" method="post">
    <table>
        <tr>
            <td>Name : </td>
            <td>
                <input type="text" name="name" />
            </td>
        </tr>
        <tr>
            <td>Address : </td>
            <td>
                <textarea cols="30" rows="5" name="address"></textarea>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <input type="submit" name="submit" value="Add" />
            </td>
        </tr>
    </table>
</form>

</body>
</html>
show.jsp (WebContent /WEB-INF/jsp/show.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring Show Page</title>
</head>
<body>

Welcome, ${name}
<br />

You are leaving area is : ${address}
</body>
</html>
```

## Output

**Index Page**

localhost:8080/SpringHelloWorldWeb/

Hello Welcome

Name :

Address :

Add

**Index Page**

localhost:8080/SpringHelloWorldWeb/

Hello Welcome

Name : Vishal Shah

Tops Technologies @Satellite and @CgRoad

Address :

Add

**Spring Show Page**

localhost:8080/SpringHelloWorldWeb/passParam.vishal

Welcome, Vishal Shah

You are leaving area is : Tops Technologies @Satellite and @CgRoad

```

Nov 07, 2014 12:25:15 PM org.springframework.web.servlet.handler.AbstractHandlerMethodMapping registerHandlerMethod
INFO: Mapped "[{/hello},methods=[GET],params=[],headers=[],consumes=[],produces=[],custom=[]]" onto public java.lang.String
Nov 07, 2014 12:25:15 PM org.springframework.web.FrameworkServlet initServletBean
INFO: FrameworkServlet 'spring': initialization completed in 1671 ms
Control Method Call
Welcome Page Call
Multiple Parameters : Name : Vishal Shah Address : Tops Technologies @Satellite and @CgRoad
Multiple Parameters : Name : Vishal Shah Address : Tops Technologies @Satellite and @CgRoad

```

### Note:

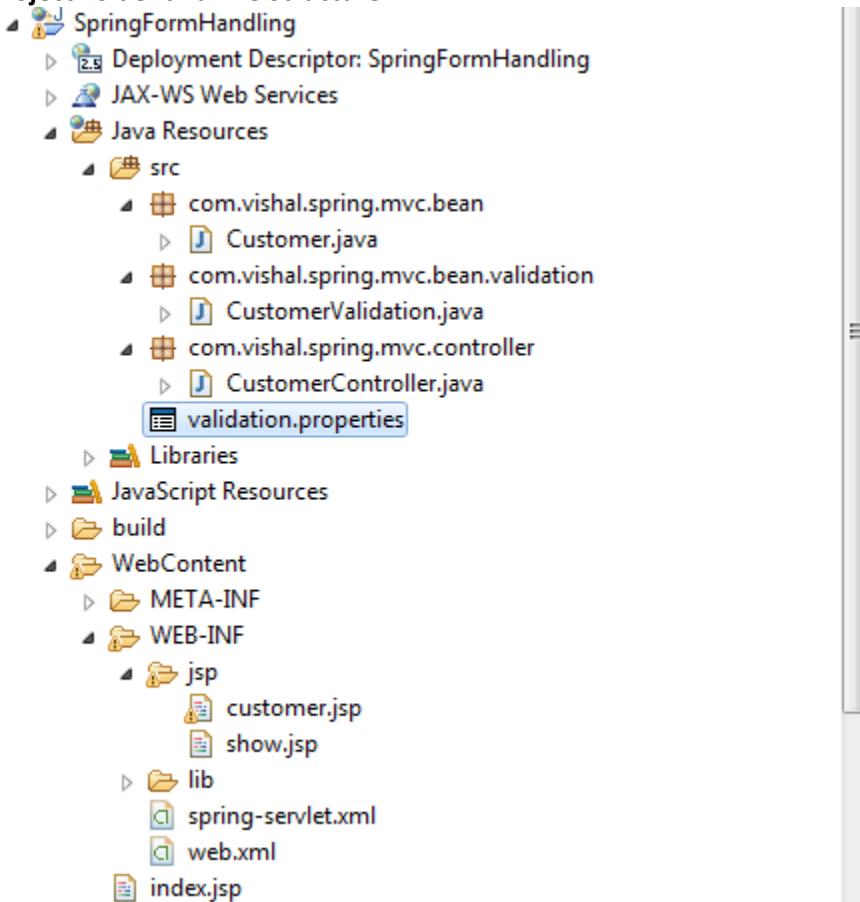
- In BeanController we are define 2 different method for two different way to passing the parameters using forms. You can use any of way.
  - First way is ModelAndView technique to return the object of ModelAndView by the passing view name, commander name, and its object in constructor.
  - Second way to pass multiple primitive and object values by using the Model interface object by setting the attributes and return the view name as string parameter.

### Spring Form Tags with Validation

The following example show how to write a simple web based application which makes use of Form tag library with dynamic data populating and static data using Spring Web MVC framework. To start with it,

let us have working Eclipse IDE in place and follow the following steps to develop a Dynamic Form based Web Application using Spring Web Framework:

## Project Folder and File Structure



### web.xml (WebContent/WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
    id="WebApp_ID" version="2.5">
    <display-name>SpringFormHandling</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
        <!-- <init-param>
            <param-name>contextConfigLocation</param-name>
```

```
<param-value>/WEB-INF/spring-servlet.xml</param-value>
</init-param> -->
</servlet>
<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
spring-servlet.xml (WebContent/WEB-INF/spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.mvc.controller" />

    <bean id="customerValidation"
class="com.vishal.spring.mvc.bean.validation.CustomerValidation"></bean>

    <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
        <property name="basename" value="validation"></property>
    </bean>

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
index.jsp (WebContent/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
</head>
<body>

<a href="customer.html">Add Customer</a>

</body>
</html>
Customer.java
package com.vishal.spring.mvc.bean;

public class Customer {

    private String name;
    private String address;
    private String[] language;
    private String gender;
    private String password;
    private String degree;
    private int id;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getDegree() {
        return degree;
    }

    public void setDegree(String degree) {
        this.degree = degree;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

public String[] getLanguage() {
    return language;
}

public void setLanguage(String[] language) {
    this.language = language;
}

public String getGender() {
    return gender;
}

public void setGender(String gender) {
    this.gender = gender;
}

}

CustomerController.java
package com.vishal.spring.mvc.controller;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.validation.BindingResult;
import org.springframework.validation.annotation.Validated;
```

```
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.support.SessionStatus;
import org.springframework.web.servlet.ModelAndView;

import com.vishal.spring.mvc.bean.Customer;
import com.vishal.spring.mvc.validation.CustomerValidation;

@Controller
public class CustomerController {

    private CustomerValidation customerValidation;

    public CustomerValidation getCustomerValidation() {
        return customerValidation;
    }

    @Autowired
    public void setCustomerValidation(CustomerValidation customerValidation) {
        this.customerValidation = customerValidation;
    }

    @RequestMapping(value = "/customer", method = RequestMethod.GET)
    private ModelAndView customer() {
        // TODO Auto-generated method stub

        Customer customer = new Customer();
        customer.setId(1);

        ModelAndView model = new ModelAndView("customer", "customer", customer);

        return model;
    }

    @ModelAttribute("customerDynamic")
    private Map<String, List<String>> dynamic(){

        Map<String, List<String>> map = new HashMap<String, List<String>>();

        List<String> languages = new ArrayList<String>();
        languages.add("Java");
        languages.add("Testing");
        languages.add("Asp.Net");
        languages.add("PHP");
        languages.add("Liferay");
    }
}
```

```
languages.add("Android");
languages.add("I-Phone");

map.put("languages", languages);

List<String> educations = new ArrayList<String>();
educations.add(" ");
educations.add("BE");
educations.add("ME");
educations.add("MCA");
educations.add("BCA");
educations.add("Phd");

map.put("degrees", educations);

return map;
}

@RequestMapping(value = "/addCustomer", method = RequestMethod.POST)
private String addCustomer(@ModelAttribute("customer") @Validated Customer customer,
ModelMap model, BindingResult result, SessionStatus status) {
    // TODO Auto-generated method stub

    String returnVal = "show";

    customerValidation.validate(customer, result);

    if(result.hasFieldErrors()){

        returnVal = "customer";

    }else{

        status.setComplete();
        model.addAttribute("customer",customer);
    }

    return returnVal;
}

}

CustomerValidation.java
package com.vishal.spring.mvc.bean.validation;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
```

```
import com.vishal.spring.mvc.bean.Customer;

public class CustomerValidation implements Validator{

    @Override
    public boolean supports(Class<?> clz) {
        // TODO Auto-generated method stub
        return Customer.class.equals(clz);
    }

    @Override
    public void validate(Object object, Errors errors) {
        // TODO Auto-generated method stub

        Customer customer = (Customer)object;

        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "valid.name");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "valid.password");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address", "valid.address");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "gender", "valid.gender");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "degree", "valid.degree");

        int passwordLength = customer.getPassword().length();
        if(passwordLength < 6 && passwordLength >12){

            errors.rejectValue("password", "valid.password.range");
        }

        int selectedLanguages = customer.getLanguage().length;
        if(selectedLanguages <1){
            errors.rejectValue("language", "valid.language");
        }
    }
}
```

**customer.jsp (WEBContent/WEB-INF/jsp/customer.jsp)**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Customer Data Entry</title>
<style type="text/css">
```

```
.error{
    color:#ff0000;
}

.errorBlock{
    color: #000;
    background-color: #ffeeee;
    border: 3px solid #ff0000;
    padding: 8px;
    margin: 16px
}


```

```
</style>
</head>
<body>
    <form:form method="POST" action="addCustomer.html" commandName="customer">
        <form:errors path="*" cssClass="errorBlock" element="div"></form:errors>

        <table>
            <tr>
                <td><form:label path="name">Name :</form:label></td>
                <td><form:input path="name" /></td>
                <td><form:errors path="name" cssClass="error" /></td>
            </tr>
            <tr>
                <td><form:label path="password">Password :</form:label></td>
                <td><form:password path="password" /></td>
                <td><form:errors path="password" cssClass="error" /></td>
            </tr>
            <tr>
                <td><form:label path="address">Address :</form:label></td>
                <td><form:textarea cols="30" rows="5" path="address" /></td>
                <td><form:errors path="address" cssClass="error" /></td>
            </tr>
            <tr>
                <td><form:label path="language">Technologies :</form:label></td>
                <td>
                    <form:checkboxes items="${customerDynamic.languages}" path="language"/>
                </td>
                <td><form:errors path="language" cssClass="error" /></td>
            </tr>
            <tr>
                <td><form:label path="gender">Gender :</form:label></td>
                <td>
                    <form:radiobutton path="gender" value="Male"/> Male
                    <form:radiobutton path="gender" value="Female"/> Female
                </td>
            </tr>
        </table>
    </form:form>
</body>
```

```

        <td><form:errors path="gender" cssClass="error" /></td>
    </tr>
    <tr>
        <td><form:label path="degree">Education :</form:label></td>
        <td>
            <form:select path="degree"
items="${customerDynamic.degrees}" />
        </td>
        <td><form:errors path="degree" cssClass="error" /></td>
    </tr>

    <tr>
        <td colspan="3">
            <form:hidden path="id" />
            <input type="submit" name="submit" value="Add Customer" />
        </td>
    </tr>
</table>
</form:form>
</body>
</html>

```

## **show.jsp (WebContent/WEB-INF/jsp/show.jsp)**

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Customer Data Show</title>
</head>
<body>

```

```

Id : ${customer.id }
<br />
Name : ${customer.name }
<br />
Password : ${customer.password }
<br />
Address : ${customer.address }
<br />
Gender : ${customer.gender }
<br />
Technologies :
<c:forEach var="lang" items="${customer.language}">
    <c:out value="${lang}" />

```

```
</c:forEach>
<br />
Education : ${customer.degree} 

</body>
</html>
validation.properties(JavaResources/validation.properties)
valid.name=Customer Name is Required!!!
valid.gender=Customer Gender is Required!!!
valid.degree=Customer Education is Required!!!
valid.language=Customer Language is Required atleast 1!!!
valid.password=Customer Password is Required!!!
valid.address=Customer Address is Required!!!
valid.password.range=Customer Password Range must be in between 6 to 12 characters!!!

Output
```



[Add Customer](#)

A screenshot of a web browser window titled "Customer Data Entry". The address bar shows the URL "localhost:8080/SpringFormHandling/addCustomer.html".

The page displays the following validation errors:

- Customer Name is Required!!!
- Customer Password is Required!!
- Customer Address is Required!!
- Customer Gender is Required!!
- Customer Education is Required!!
- Customer Language is Required atleast !!!

Below the errors, there are input fields and their corresponding validation messages:

- Name: [Input Field] Customer Name is Required!!
- Password: [Input Field] Customer Password is Required!!
- Address: [Input Field] Customer Address is Required!!
- Technologies: [checkboxes] Java, Testing, ASP.NET, PHP, Liferay, Android, I-Phone Customer Language is Required atleast !!!
- Gender: [radio buttons] Male, Female Customer Gender is Required!!
- Education: [dropdown menu] Customer Education is Required!!

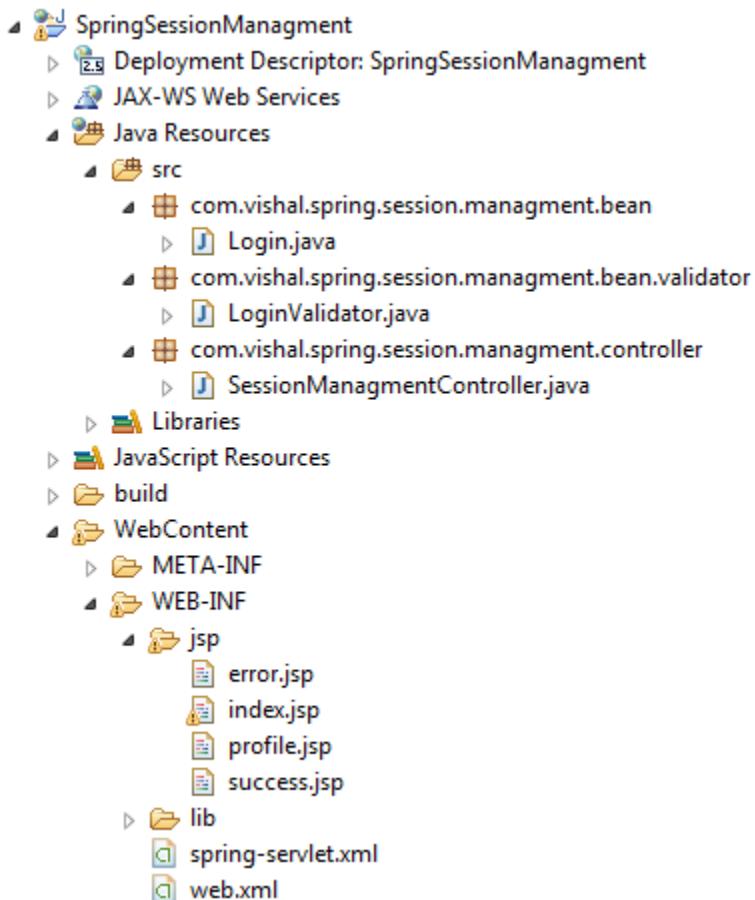
A "AddCustomer" button is located at the bottom left.

The screenshot shows two browser windows side-by-side. The left window is titled 'Customer Data Entry' and has the URL 'localhost:8080/SpringFormHandling/customer.html'. It contains a form with fields for Name (Vishal Shah), Password (\*\*\*\*\*), Address (Satellite), and a list of Technologies (Java, Testing, Liferay, Android). It also includes gender (Male selected) and education (ME selected) dropdowns. A 'Add Customer' button is at the bottom. The right window is titled 'Customer Data Show' and has the URL 'localhost:8080/SpringFormHandling/addCustomer.html'. It displays the same customer data: Id: 1, Name: Vishal Shah, Password: vishal, Address: Satellite, Gender: Male, Technologies: Java, Testing, Liferay, Android, and Education: ME.

## Spring MVC with Session Management

Session management is one of the essential parts for each web application. Since Spring MVC is a powerful framework for web development, it has its own tools and API for the interaction with sessions. Today I intend to show you the basic ways of session processing within Spring MVC application. That is: how to process forms, add objects into a session, and display objects from the session on JSP. This is a simple Spring MVC controller with the one extra `@SessionAttributes` annotation. It indicates that in the controller's methods some values can be assigned to the arguments of the annotation. In this example I have declared just one session attribute with the name "login". That means I can put some object into `ModelAndView` using the  `addObject()` method, and it will be added to the session if the name of the object will be the same as the name of the argument in `@SessionAttributes`.

Project Folder and File Structure:



## web.xml (WebContent/WEB-INF/web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
    id="WebApp_ID" version="2.5">
    <display-name>SpringSessionManagement</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

```
</web-app>
spring-servlet.xml(WebContent/WEB-INF/spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.session.managment.controller" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>

    <bean id="loginValidator"
        class="com.vishal.spring.session.managment.bean.validator.LoginValidator"></bean>

</beans>
Login.java
package com.vishal.spring.session.managment.bean;

public class Login {

    private String name;
    private String password;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

```
}

index.jsp (WebContent/WEB-INF/jsp/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Login Page</title>
</head>
<body>
    <a href="profile">Check Profile</a>

    <br />

    <f:form method="post" commandName="login" action="login">

        <table>
            <c:if test="${logout !=null}">
                <tr>
                    <td colspan="3">
                        <c:out value="${logout}"></c:out>
                    </td>
                </tr>
            </c:if>
            <c:if test="${error!=null}">
                <tr>
                    <td colspan="3">
                        <c:out value="${error}"></c:out>
                    </td>
                </tr>
            </c:if>
            <tr>
                <td><f:label path="name">Name :</f:label></td>
                <td><f:input path="name" /></td>
                <td><f:errors path="name" /></td>
            </tr>
            <tr>
                <td><f:label path="password">Password :</f:label></td>
                <td><f:password path="password" /></td>
                <td><f:errors path="password" /></td>
            </tr>
            <tr>
```

```
<td colspan="3"><input type="submit" name="submit"
    value="Login" /></td>
</tr>

</table>

</f:form>

</body>
</html>
SessionManagementController.java
package com.vishal.spring.session.managment.controller;

import javax.servlet.http.HttpSession;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.SessionAttributes;
import org.springframework.web.servlet.ModelAndView;

import com.vishal.spring.session.managment.bean.Login;
import com.vishal.spring.session.managment.bean.validator.LoginValidator;

@Controller
@SessionAttributes("login")
public class SessionManagementController {

    LoginValidator loginValidator;

    public LoginValidator getLoginValidator() {
        return loginValidator;
    }

    @Autowired
    public void setLoginValidator(LoginValidator loginValidator) {
        this.loginValidator = loginValidator;
    }

    @RequestMapping(value = { "/" }, method = RequestMethod.GET)
    private ModelAndView index() {
        // TODO Auto-generated method stub
    }
}
```

```
ModelAndView model = new ModelAndView("index", "login", new Login());  
  
        return model;  
    }  
  
    @RequestMapping(value = { "/index" }, method = RequestMethod.GET)  
    private ModelAndView auth() {  
        // TODO Auto-generated method stub  
  
        ModelAndView model = new ModelAndView("index", "login", new Login());  
        model.addObject("error", "You are Fake User. Session is destroyed!!! If you not then  
please go through Login Phase.");  
        return model;  
    }  
  
    @RequestMapping(value = "/login", method = RequestMethod.POST)  
    private String login(@ModelAttribute Login login, Model model,  
        BindingResult result) {  
        // TODO Auto-generated method stub  
  
        loginValidator.validate(login, result);  
  
        if (result.hasErrors()) {  
  
            return "index";  
        } else {  
            model.addAttribute("login", login);  
  
            return "success";  
        }  
    }  
  
    @RequestMapping(value = "/profile", method = RequestMethod.GET)  
    private String profile() {  
        // TODO Auto-generated method stub  
  
        return "profile";  
    }  
  
    @RequestMapping(value = "/logout", method = RequestMethod.GET)  
    private String logout HttpSession session){  
  
        System.out.println("Logout");  
        // This is another techniques for session management.... back tracking is also perfect after  
expiring the session.  
        session.removeAttribute("login");  
        session.invalidate();  
    }
```

```
        return "redirect:/";
    }

    @ExceptionHandler(Exception.class)
    private String handleException(Model model, Exception e) {
        // TODO Auto-generated method stub

        model.addAttribute("exception", e);
        return "error";
    }
    //Now Exception Problem is solved.....
}

LoginValidator.java
package com.vishal.spring.session.managment.bean.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.vishal.spring.session.managment.bean.Login;

public class LoginValidator implements Validator{

    @Override
    public boolean supports(Class<?> clz) {
        // TODO Auto-generated method stub
        return Login.class.equals(clz);
    }

    @Override
    public void validate(Object object, Errors errors) {
        // TODO Auto-generated method stub
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required.name", "Enter
Name");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "required.password",
"Enter Password");
    }

}

profile.jsp (WebContent/WEB-INF/jsp/profile.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
<title>Profile Page</title>  
</head>  
<body>  
  
<c:if test="\${sessionScope.login != null}">  
  
    \${sessionScope.login.name } is Activated!!!!  
  
    <br />  
  
    <a href="logout">Logout</a>  
  
</c:if>  
<c:if test="\${sessionScope.login.name ==null || sessionScope.login.name ==''}">  
  
    <c:redirect url="index"></c:redirect>  
</c:if>  
  
</body>  
</html>  
success.jsp (WebContent/WEB-INF/jsp/success.jsp)  
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"  
    pageEncoding="ISO-8859-1"%>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">  
<title>Success Page</title>  
</head>  
<body>
```

Success Login Detail:

```
<br />  
<hr />  
  
Name : \${login.name} <br />  
Password : \${login.password }  
  
<br />  
<hr />
```

Session Value Check :

```
<br/>
Name : ${sessionScope.login.name } <br />
Password : ${sessionScope.login.password }
```

```
<br />
```

```
<a href="profile">Go to Profile</a>
```

```
<br />
```

```
<a href="logout">Logout</a>
```

```
</body>
```

```
</html>
```

**error.jsp (WebContent/WEB-INF/jsp/error.jsp)**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Error Page</title>
</head>
<body>
```

Message : \${exception}

```
<br />
```

```
<a href="index">Go to Login Page</a>
```

```
</body>
```

**Output**

The screenshot shows a web browser window with the following details:

- Title Bar:** Shows "Login Page" as the active tab.
- Address Bar:** Displays the URL "localhost:8080/SpringSessionManagement/".
- Content Area:** Shows a form with the following structure:
  - Section Title:** Check Profile
  - Name:**
  - Password:**
  - Login Button:**

# TOPS Technologies

The screenshot shows a sequence of three browser windows illustrating session management:

- First Window:** Title: Login Page. URL: localhost:8080/SpringSessionManagement/index. Content: "Check Profile" link, message "You are Fake User. Session is destroyed!!! If you not then please go through Login Phase.", Name: [text input], Password: [text input], Login button.
- Second Window:** Title: Login Page. URL: localhost:8080/SpringSessionManagement/login. Content: "Check Profile" link, Name: [text input] "Enter Name", Password: [text input] "Enter Password", Login button.
- Third Window:** Title: Success Page. URL: localhost:8080/SpringSessionManagement/login. Content: "Success Login Detail:" followed by Name: tops and Password: 132456, "Session Value Check:" followed by Name: tops and Password: 132456, and links for Go to Profile and Logout.

tops is Activated!!!!

[Logout](#)

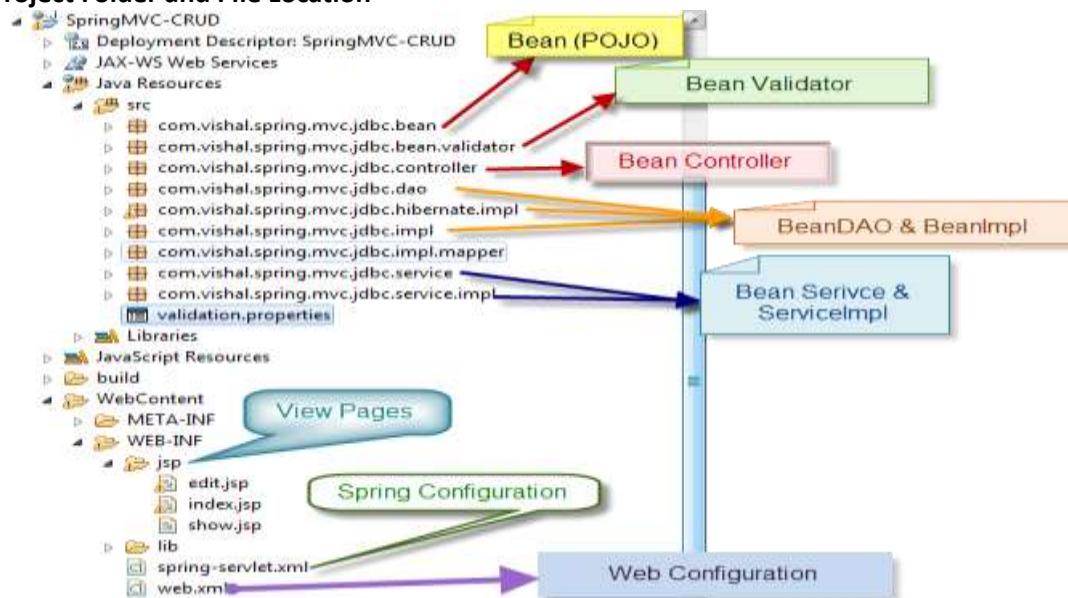
## Spring MVC CRUD Operation

In Spring MVC, we are performing CRUD (Create, Read , Update, Delete) operation by using JDBCCTemplate and HibernateTemaplte. So We can performing to integration of JDBC API and Hibernate. Spring CRUD Operation Steps

1. To add DispatcherServlet servlet configuration in web.xml file
2. To add spring-servlet.xml file with Bean configuration of InternalResourceViewResolver. component scanning and mvc annotation driver.
3. Create Bean POJO Class with properties and their getter and setter.

4. If you are using hibernate them create bean.hbm.xml file.
5. Create Controller with index method.
6. Create index.jsp page with Adding the entry form.
7. Create Validation of the Bean with Validator Class
8. Create validator properties file
9. To add validator bean mapping and ResourceBundleMessageSource for properties configuration in spring-servlet.xml file.
10. Create DAO and Service interface with declaring the methods.
11. Create DAOImpl and ServiceImpl classes for implementing the interfaces.
12. To add DAO and Service bean mapping in spring-servlet.xml file.
13. To add datasource mapping in spring-servlet.xml file.
14. If you are using hibernate then you have to configure session factory using LocalSessionFactoryBean with hibernate properties and resource mapping files.
15. To add method in controller for inserting data in database via service calling after validating form bean data.
16. Redirect to control to show method for setting attribute list data values of bean. It will show the data entries in show.jsp page.
17. To click on delete button to call delete method of controller for deleting the entries via service method calling. Afterword redirect control to show.jsp via show method of controller for setting the list attribute.
18. To click on edit link to call edit method of controller for fetching specific bean object via service method and set ModelAndView with commander object. Show entries in edit.jsp page for updating entries.
19. To update entries in database by update method of controller. This update method call through edit.jsp page form. After redirect to show.jsp via call show method through update method.

## Project Folder and File Location



## Spring CRUD Operation using JDBC

```
web.xml(WebContent/WEB-INF/web.xml)
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
    id="WebApp_ID" version="2.5">
    <display-name>SpringMVC-CRUD</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
spring-servlet.xml(WebContent/WEB-INF/spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.mvc.jdbc.controller" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
```

```
<bean id="studentValidator"
class="com.vishal.spring.mvc.jdbc.bean.validator.StudentValidator"></bean>

<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource" >
    <property name="basename" value="validation"></property>
</bean>

<bean id="studentDao" class="com.vishal.spring.jdbc.impl.StudentImpl"></bean>

<bean id="studentService"
class="com.vishal.spring.jdbc.service.impl.StudentServiceImpl"></bean>

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql://localhost:3306/springmvc"></property>
    <property name="username" value="root"></property>
    <property name="password" value=""></property>
</bean>

</beans>
```

## Student.java

```
package com.vishal.spring.mvc.jdbc.bean;

public class Student {

    private int id;
    private String name;
    private String address;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

```
}

}

StudentDAO.java
package com.vishal.spring.mvc.jdbc.dao;

import java.util.List;

import com.vishal.spring.jdbc.bean.Student;

public interface StudentDAO {

    public void addStudent(Student student);
    public void updateStudent(Student student);
    public void deleteStudent(int id);
    public Student findStudent(int id);
    public List<Student> getStudents();

}

StudentImpl.java
package com.vishal.spring.jdbc.impl;

import java.util.List;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;

import com.vishal.spring.jdbc.bean.Student;
import com.vishal.spring.jdbc.dao.StudentDAO;
import com.vishal.spring.jdbc.impl.mapper.StudentMapper;

public class StudentImpl implements StudentDAO {

    @Autowired
    DataSource dataSource;

    @Override
    public void addStudent(Student student) {
        // TODO Auto-generated method stub
        JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

        String sql = "insert into student(name, address) values(?,?)";

        Object[] parameterValues = new Object[] { student.getName(),
            student.getAddress() };
    }
}
```

```
        jdbcTemplate.update(sql, paramterValues);

    }

@Override
public void updateStudent(Student student) {
    // TODO Auto-generated method stub
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    String sql = "update student set name=?, address=? where id=?";

    Object[] paramterValues = new Object[] { student.getName(),
  student.getAddress(), student.getId() };

    jdbcTemplate.update(sql, paramterValues);
}

@Override
public void deleteStudent(int id) {
    // TODO Auto-generated method stub
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    String sql = "delete from student where id=?";

    Object[] paramterValues = new Object[] { id };

    jdbcTemplate.update(sql, paramterValues);
}

@Override
public Student findStudent(int id) {
    // TODO Auto-generated method stub
    JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

    String sql = "select * from student where id=?";

    Object[] parameterValues = new Object[]{id};

    Student student = jdbcTemplate.queryForObject(sql, parameterValues, new
StudentMapper());

    return student;
}

@Override
public List<Student> getStudents() {
    // TODO Auto-generated method stub
```

```
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);

String sql = "select * from student";
List<Student> students = jdbcTemplate.query(sql , new StudentMapper());

return students;
}

}

StudentMapper.java
package com.vishal.spring.mvc.jdbc.impl.mapper;

import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.vishal.spring.jdbc.bean.Student;

public class StudentMapper implements RowMapper<Student> {

    @Override
    public Student mapRow(ResultSet rs, int rowCount) throws SQLException {
        // TODO Auto-generated method stub

        Student student = new Student();
        student.setId(rs.getInt("id"));
        student.setName(rs.getString("name"));
        student.setAddress(rs.getString("address"));

        return student;
    }
}

StudentService.java
package com.vishal.spring.mvc.jdbc.service;

import java.util.List;

import com.vishal.spring.jdbc.bean.Student;

public interface StudentService {

    public void addStudent(Student student);
    public void updateStudent(Student student);
    public void deleteStudent(int id);
    public Student findStudent(int id);
```

```
public List<Student> getStudents();  
}  
StudentServiceImpl.java  
package com.vishal.spring.mvc.jdbc.service.impl;  
  
import java.util.List;  
  
import org.springframework.beans.factory.annotation.Autowired;  
  
import com.vishal.spring.jdbc.bean.Student;  
import com.vishal.spring.jdbc.dao.StudentDAO;  
import com.vishal.spring.jdbc.service.StudentService;  
  
public class StudentServiceImpl implements StudentService {  
  
    @Autowired  
    StudentDAO studentDAO;  
  
    @Override  
    public void addStudent(Student student) {  
        // TODO Auto-generated method stub  
        studentDAO.addStudent(student);  
    }  
  
    @Override  
    public void updateStudent(Student student) {  
        // TODO Auto-generated method stub  
        studentDAO.updateStudent(student);  
    }  
  
    @Override  
    public void deleteStudent(int id) {  
        // TODO Auto-generated method stub  
        studentDAO.deleteStudent(id);  
    }  
  
    @Override  
    public Student findStudent(int id) {  
        // TODO Auto-generated method stub  
        return studentDAO.findStudent(id);  
    }  
  
    @Override  
    public List<Student> getStudents() {  
        // TODO Auto-generated method stub  
        return studentDAO.getStudents();  
    }  
}
```

```
}
```

**StudentController.java**

```
package com.vishal.spring.mvc.jdbc.controller;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Controller;
```

```
import org.springframework.validation.BindingResult;
```

```
import org.springframework.web.bind.annotation.ModelAttribute;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
```

```
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.servlet.ModelAndView;
```

```
import com.vishal.spring.mvc.jdbc.bean.Student;
```

```
import com.vishal.spring.jdbc.bean.validator.StudentValidator;
```

```
import com.vishal.spring.jdbc.service.StudentService;
```

```
@Controller
```

```
public class StudentController {
```

```
    StudentValidator studentValidator;
```

```
    public StudentValidator getStudentValidator() {
```

```
        return studentValidator;
```

```
    }
```

```
    @Autowired
```

```
    public void setStudentValidator(StudentValidator studentValidator) {
```

```
        this.studentValidator = studentValidator;
```

```
    }
```

```
    @Autowired
```

```
    StudentService studentService;
```

```
    @RequestMapping(value = { "/", "/index" }, method = RequestMethod.GET)
```

```
    private ModelAndView index(@ModelAttribute Student student) {
```

```
        // TODO Auto-generated method stub
```

```
        ModelAndView model = new ModelAndView("index", "student", student);
```

```
        return model;
```

```
    }
```

```
    @RequestMapping(value = "/add", method = RequestMethod.POST)
```

```
    private String add(@ModelAttribute Student student, BindingResult result) {
```

```
studentValidator.validate(student, result);

if(result.hasErrors()){

    return "index";

} else{
    studentService.addStudent(student);
    System.out.println("Add Student");
    return "redirect:/show";
}

}

@RequestMapping(value = { "/edit", "/find" }, method = { RequestMethod.GET,
    RequestMethod.POST })
private ModelAndView edit(@RequestParam int id,
    @ModelAttribute Student student) {

    student = studentService.findStudent(id);
    ModelAndView model = new ModelAndView("edit", "student", student);

    return model;
}

@RequestMapping(value = "/update", method = RequestMethod.POST)
private String update(@ModelAttribute Student student, BindingResult result) {
    // TODO Auto-generated method stub

    studentValidator.validate(student, result);

    if(result.hasErrors()){

        return "edit";

} else{
    studentService.updateStudent(student);
    System.out.println("Update Student");

    return "redirect:/show";
}

}

@RequestMapping(value = "/delete", method = { RequestMethod.GET,
    RequestMethod.POST })
private String delete(@RequestParam int id) {
```

```
// TODO Auto-generated method stub
studentService.deleteStudent(id);
System.out.println("Delete Student");
return "redirect:/show";
}

@RequestMapping(value = "/show")
private ModelAndView show() {

    List<Student> students = studentService.getStudents();
    ModelAndView model = new ModelAndView("show", "students", students);

    return model;
}

}
StudentValidator.java
package com.vishal.spring.mvc.jdbc.bean.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.vishal.spring.mvc.jdbc.bean.Student;

public class StudentValidator implements Validator{

    @Override
    public boolean supports(Class<?> clz) {
        // TODO Auto-generated method stub
        return Student.class.equals(clz);
    }

    @Override
    public void validate(Object object, Errors errors) {
        // TODO Auto-generated method stub
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "valid.name");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address", "valid.address");
    }
}

validation.properties (JavaResources/validation.properties)
valid.name=Name is Required\!\!
valid.address=Address is Required\!\!
index.jsp (WebContent/WEB-INF/jsp/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
```

```
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
<style type="text/css">
.error{
    color: #ff0000;
}
</style>
</head>
<body>

<a href="show">Show Students</a>
<br />
<f:form method="post" commandName="student" action="add">
    <table>
        <tr>
            <td colspan="3">
                <f:errors path="*" cssClass="error" />
            </td>
        </tr>
        <tr>
            <td><f:label path="name">Name :</f:label></td>
            <td><f:input path="name"/></td>
            <td><f:errors path="name" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td><f:label path="address">Address :</f:label></td>
            <td><f:textarea path="address" rows="5" cols="25" /></td>
            <td><f:errors path="address" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td colspan="3">
                <input type="submit" name="submit" value="Add Student" />
            </td>
        </tr>
    </table>
</f:form>
</body>
</html>
edit.jsp(WEBContent/WEB-INF/jsp/edit.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
<style type="text/css">
.error{
    color: #ff0000;
}
</style>
</head>
<body>

<a href="show">Show Students</a>
<br />

<f:form method="post" commandName="student" action="update">
    <table>
        <tr>
            <td colspan="3">
                <f:errors path="*" cssClass="error" />
            </td>
        </tr>
        <tr>
            <td><f:label path="name">Name :</f:label></td>
            <td><f:input path="name" /></td>
            <td><f:errors path="name" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td><f:label path="address">Address :</f:label></td>
            <td><f:textarea path="address" rows="5" cols="25" /></td>
            <td><f:errors path="address" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td colspan="3">
                <input type="hidden" name="id" value="${student.id}" />
                <input type="submit" name="submit" value="Update Student" />
            </td>
        </tr>
    </table>
</f:form>
</body>
</html>
show.jsp(WEBContent/WEB-INF/jsp/show.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form"%>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Show List of Students</title>
</head>
<body>

<a href="index">Add Student</a>
<br />
<table>
    <tr>
        <td>NO</td>
        <td>ID</td>
        <td>NAME</td>
        <td>ADDRESS</td>
        <td>EDIT</td>
        <td>DELETE</td>
    </tr>
    <c:forEach var="student" varStatus="studentStatus" items="${students}">
        <tr>
            <td>${studentStatus.count}</td>
            <td>${student.id}</td>
            <td>${student.name}</td>
            <td>${student.address}</td>
            <td>
                <a href="edit?id=${student.id}">EDIT</a>
            </td>
            <td>
                <f:form action="delete" method="post">
                    <input type="hidden" name="id" value="${student.id}" />
                    <input type="submit" name="submit" value="DELETE" />
                </f:form>
            </td>
        </tr>
    </c:forEach>
</table>

</body>
</html>
```

## Spring CRUD Operation using Hibernate ORM

```
web.xml(WebContent/WEB-INF/web.xml)
<?xml version="1.0" encoding="UTF-8"?>
<web-app 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"
    id="WebApp_ID" version="2.5">
    <display-name>SpringMVC-CRUD</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
spring-servlet.xml(WebContent/WEB-INF/spring-servlet.xml)
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context-3.0.xsd">

    <mvc:annotation-driven />

    <context:component-scan base-package="com.vishal.spring.mvc.jdbc.controller" />

    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/" />
        <property name="suffix" value=".jsp" />
    </bean>
```

```
<bean id="studentValidator"
class="com.vishal.spring.mvc.jdbc.bean.validator.StudentValidator"></bean>

<bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource" >
    <property name="basename" value="validation"></property>
</bean>

<!-- <bean id="studentDao" class="com.vishal.spring.jdbc.impl.StudentImpl"></bean> -->

<bean id="studentService"
class="com.vishal.spring.jdbc.service.impl.StudentServiceImpl"></bean>

<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
    <property name="url" value="jdbc:mysql://localhost:3306/springmvc"></property>
    <property name="username" value="root"></property>
    <property name="password" value=""></property>
</bean>

<!-- Hibernate Integration -->

<bean id="studentDao" class="com.vishal.spring.jdbc.hibernate.impl.StudentImpl">
    <property name="sessionFactory" ref="sessionFactory"></property>
</bean>

<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
    <property name="mappingResources">
        <list>
            <value>com/vishal/spring/mvc/jdbc/bean/Student.hbm.xml</value>
        </list>
    </property>
</bean>

</beans>
```

**Student.java**

```
package com.vishal.spring.mvc.jdbc.bean;
```

```
public class Student {
```

```
    private int id;
    private String name;
    private String address;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
}
```

**StudentDAO.java**

```
package com.vishal.spring.mvc.jdbc.dao;

import java.util.List;

import com.vishal.spring.mvc.jdbc.bean.Student;

public interface StudentDAO {
```

```
    public void addStudent(Student student);
    public void updateStudent(Student student);
    public void deleteStudent(int id);
    public Student findStudent(int id);
    public List<Student> getStudents();
```

```
}
```

**StudentImpl.java**

```
package com.vishal.spring.mvc.jdbc.hibernate.impl;
```

```
import java.util.List;
```

```
import org.springframework.orm.hibernate3.HibernateTemplate;
```

```
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import com.vishal.spring.mvc.jdbc.bean.Student;
import com.vishal.spring.mvc.jdbc.dao.StudentDAO;

public class StudentImpl extends HibernateDaoSupport implements StudentDAO{

    @Override
    public void addStudent(Student student) {
        // TODO Auto-generated method stub
        getHibernateTemplate().save(student);
        System.out.println("Hibernate Save");
    }

    @Override
    public void updateStudent(Student student) {
        // TODO Auto-generated method stub
        getHibernateTemplate().update(student);
        System.out.println("Hibernate Update");
    }

    @Override
    public void deleteStudent(int id) {
        // TODO Auto-generated method stub
        HibernateTemplate template = getHibernateTemplate();
        Student student = template.get(Student.class, id);
        template.delete(student);
        System.out.println("Hibernate Delete");
    }

    @Override
    public Student findStudent(int id) {
        // TODO Auto-generated method stub
        return getHibernateTemplate().get(Student.class, id);
    }

    @Override
    public List<Student> getStudents() {
        // TODO Auto-generated method stub
        List<Student> students = getHibernateTemplate().find("from Student");
        return students;
    }

}

StudentService.java
package com.vishal.spring.mvc.jdbc.service;

import java.util.List;
```

```
import com.vishal.spring.mvc.jdbc.bean.Student;

public interface StudentService {

    public void addStudent(Student student);
    public void updateStudent(Student student);
    public void deleteStudent(int id);
    public Student findStudent(int id);
    public List<Student> getStudents();

}

StudentServiceImpl.java
package com.vishal.spring.mvc.jdbc.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import com.vishal.spring.mvc.jdbc.bean.Student;
import com.vishal.spring.jdbc.dao.StudentDAO;
import com.vishal.spring.jdbc.service.StudentService;

public class StudentServiceImpl implements StudentService {

    @Autowired
    StudentDAO studentDAO;

    @Override
    public void addStudent(Student student) {
        // TODO Auto-generated method stub
        studentDAO.addStudent(student);
    }

    @Override
    public void updateStudent(Student student) {
        // TODO Auto-generated method stub
        studentDAO.updateStudent(student);
    }

    @Override
    public void deleteStudent(int id) {
        // TODO Auto-generated method stub
        studentDAO.deleteStudent(id);
    }

    @Override
    public Student findStudent(int id) {
```

```
// TODO Auto-generated method stub
    return studentDAO.findStudent(id);
}

@Override
public List<Student> getStudents() {
    // TODO Auto-generated method stub
    return studentDAO.getStudents();
}

}
StudentController.java
package com.vishal.spring.mvc.jdbc.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import com.vishal.spring.mvc.jdbc.bean.Student;
import com.vishal.spring.mvc.jdbc.bean.validator.StudentValidator;
import com.vishal.spring.mvc.jdbc.service.StudentService;

@Controller
public class StudentController {

    StudentValidator studentValidator;

    public StudentValidator getStudentValidator() {
        return studentValidator;
    }

    @Autowired
    public void setStudentValidator(StudentValidator studentValidator) {
        this.studentValidator = studentValidator;
    }

    @Autowired
    StudentService studentService;

    @RequestMapping(value = { "/", "/index" }, method = RequestMethod.GET)
    private ModelAndView index(@ModelAttribute Student student) {
```

```
// TODO Auto-generated method stub

ModelAndView model = new ModelAndView("index", "student", student);

return model;
}

@RequestMapping(value = "/add", method = RequestMethod.POST)
private String add(@ModelAttribute Student student, BindingResult result) {

    studentValidator.validate(student, result);

    if(result.hasErrors()){

        return "index";

    }else{
        studentService.addStudent(student);
        System.out.println("Add Student");
        return "redirect:/show";
    }
}

@RequestMapping(value = { "/edit", "/find" }, method = { RequestMethod.GET,
    RequestMethod.POST })
private ModelAndView edit(@RequestParam int id,
    @ModelAttribute Student student) {

    student = studentService.findStudent(id);
    ModelAndView model = new ModelAndView("edit", "student", student);

    return model;
}

@RequestMapping(value = "/update", method = RequestMethod.POST)
private String update(@ModelAttribute Student student, BindingResult result) {
    // TODO Auto-generated method stub

    studentValidator.validate(student, result);

    if(result.hasErrors()){

        return "edit";

    }else{
        studentService.updateStudent(student);
        System.out.println("Update Student");
    }
}
```

```
        return "redirect:/show";
    }

}

@RequestMapping(value = "/delete", method = { RequestMethod.GET,
    RequestMethod.POST })
private String delete(@RequestParam int id) {
    // TODO Auto-generated method stub
    studentService.deleteStudent(id);
    System.out.println("Delete Student");
    return "redirect:/show";
}

@RequestMapping(value = "/show")
private ModelAndView show() {

    List<Student> students = studentService.getStudents();
    ModelAndView model = new ModelAndView("show", "students", students);

    return model;
}

}
StudentValidator.java
package com.vishal.spring.mvc.jdbc.bean.validator;

import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;

import com.vishal.spring.jdbc.bean.Student;

public class StudentValidator implements Validator{

    @Override
    public boolean supports(Class<?> clz) {
        // TODO Auto-generated method stub
        return Student.class.equals(clz);
    }

    @Override
    public void validate(Object object, Errors errors) {
        // TODO Auto-generated method stub
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "valid.name");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "address", "valid.address");
    }
}
```

```
}

}

index.jsp (WebContent/WEB-INF/jsp/index.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
<style type="text/css">
.error{
    color: #ff0000;
}
</style>
</head>
<body>

<a href="show">Show Students</a>
<br />
<f:form method="post" commandName="student" action="add">
    <table>
        <tr>
            <td colspan="3">
                <f:errors path="*" cssClass="error" />
            </td>
        </tr>
        <tr>
            <td><f:label path="name">Name :</f:label></td>
            <td><f:input path="name"/></td>
            <td><f:errors path="name" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td><f:label path="address">Address :</f:label></td>
            <td><f:textarea path="address" rows="5" cols="25" /></td>
            <td><f:errors path="address" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td colspan="3">
                <input type="submit" name="submit" value="Add Student" />
            </td>
        </tr>
    </table>
</f:form>
</body>
```

```
</html>
edit.jsp(WEBContent/WEB-INF/jsp/edit.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Index Page</title>
<style type="text/css">
.error{
    color: #ff0000;
}
</style>
</head>
<body>

<a href="show">Show Students</a>
<br />

<f:form method="post" commandName="student" action="update">
    <table>
        <tr>
            <td colspan="3">
                <f:errors path="*" cssClass="error" />
            </td>
        </tr>
        <tr>
            <td><f:label path="name">Name :</f:label></td>
            <td><f:input path="name" /></td>
            <td><f:errors path="name" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td><f:label path="address">Address :</f:label></td>
            <td><f:textarea path="address" rows="5" cols="25" /></td>
            <td><f:errors path="address" cssClass="error"></f:errors></td>
        </tr>
        <tr>
            <td colspan="3">
                <input type="hidden" name="id" value="${student.id}" />
                <input type="submit" name="submit" value="Update Student" />
            </td>
        </tr>
    </table>
</f:form>
</body>
```

```
</html>
show.jsp(WEBContent/WEB-INF/jsp/show.jsp)
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="f" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Show List of Students</title>
</head>
<body>

<a href="index">Add Student</a>
<br />
<table>
    <tr>
        <td>NO</td>
        <td>ID</td>
        <td>NAME</td>
        <td>ADDRESS</td>
        <td>EDIT</td>
        <td>DELETE</td>
    </tr>
    <c:forEach var="student" varStatus="studentStatus" items="${students}">
        <tr>
            <td>${studentStatus.count}</td>
            <td>${student.id}</td>
            <td>${student.name}</td>
            <td>${student.address}</td>
            <td>
                <a href="edit?id=${student.id}">EDIT</a>
            </td>
            <td>
                <f:form action="delete" method="post">
                    <input type="hidden" name="id" value="${student.id}" />
                    <input type="submit" name="submit" value="DELETE" />
                </f:form>
            </td>
        </tr>
    </c:forEach>
</table>

</body>
</html>
```

## Spring CRUD Operation Output

**Index Page**

localhost:8080/SpringMVC-CRUD/add

Show Students

Name is Required!!

Address is Required!!

Name :  Name is Required!!

Address :  Address is Required!!

**Add Student**

**Index Page**

localhost:8080/SpringMVC-CRUD/

Show Students

Name :  CGRoad

Address :

**Add Student**

**Show List of Students**

localhost:8080/SpringMVC-CRUD/show

Add Student

NO	ID	NAME	ADDRESS	EDIT	DELETE
1	1	vishal shah	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
2	4	vish	ahmedabad	<a href="#">EDIT</a>	<a href="#">DELETE</a>
3	6	vishal	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
4	7	vishal hibernate	hibernate jboss	<a href="#">EDIT</a>	<a href="#">DELETE</a>
5	8	done	hibernate	<a href="#">EDIT</a>	<a href="#">DELETE</a>
6	9	Aakash	CGRoad	<a href="#">EDIT</a>	<a href="#">DELETE</a>

**Index Page**

localhost:8080/SpringMVC-CRUD/edit?id=8

Show Students

Name :  Baroda

Address :

**Update Student**

Index Page x +

localhost:8080/SpringMVC-CRUD/update ▼ C

Show Students

Address is Required!!

Name :

Address :  Address is Required!!

Show List of Students x +

localhost:8080/SpringMVC-CRUD/show ▼ C

NO	ID	NAME	ADDRESS	EDIT	DELETE
1	1	vishal shah	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
2	4	vish	ahmedabad	<a href="#">EDIT</a>	<a href="#">DELETE</a>
3	6	vishal	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
4	7	vishal hibernate	hibernate jboss	<a href="#">EDIT</a>	<a href="#">DELETE</a>
5	8	Narayan	Baroda	<a href="#">EDIT</a>	<a href="#">DELETE</a>
6	9	Aakash	CGRoad	<a href="#">EDIT</a>	<a href="#">DELETE</a>

Show List of Students x +

localhost:8080/SpringMVC-CRUD/show ▼ C

NO	ID	NAME	ADDRESS	EDIT	DELETE
1	1	vishal shah	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
2	4	vish	ahmedabad	<a href="#">EDIT</a>	<a href="#">DELETE</a>
3	6	vishal	satellite	<a href="#">EDIT</a>	<a href="#">DELETE</a>
4	7	vishal hibernate	hibernate jboss	<a href="#">EDIT</a>	<a href="#">DELETE</a>
5	8	Narayan	Baroda	<a href="#">EDIT</a>	<a href="#">DELETE</a>
6	9	Aakash	CGRoad	<a href="#">EDIT</a>	<a href="#">DELETE</a>



## Spring Boot

- Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework.
- It is used to create stand alone spring based application that you can just run because it needs very little spring configuration.
- Spring Boot does not generate code and there is absolutely no requirement for XML configuration.
- It uses convention over configuration software design paradigm that means it decrease the effort of developer.

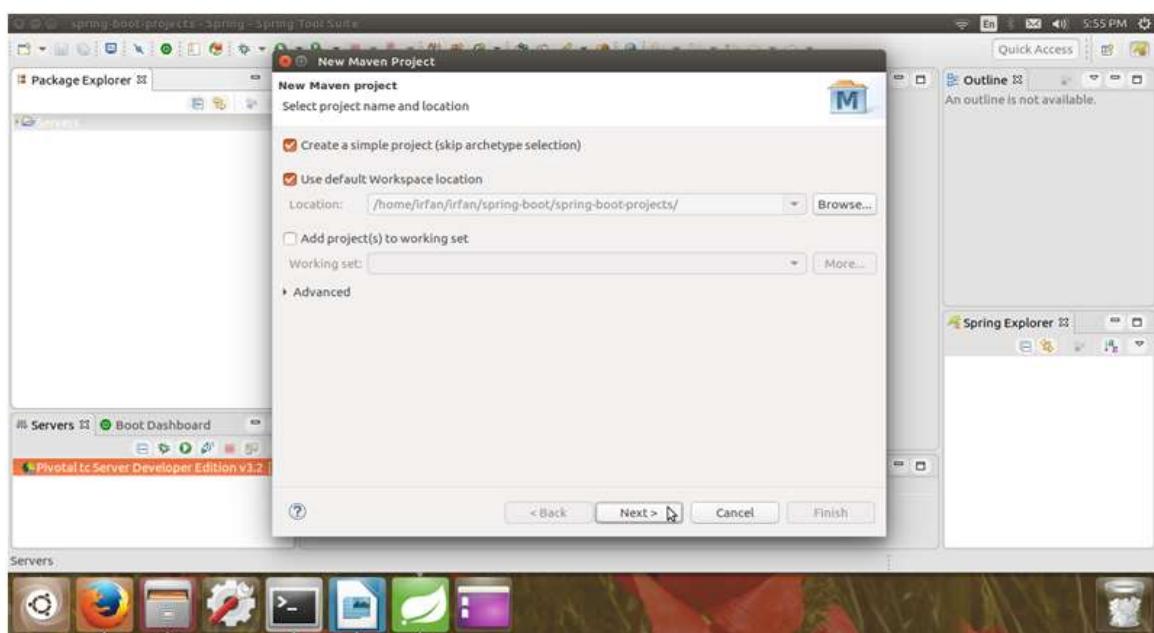
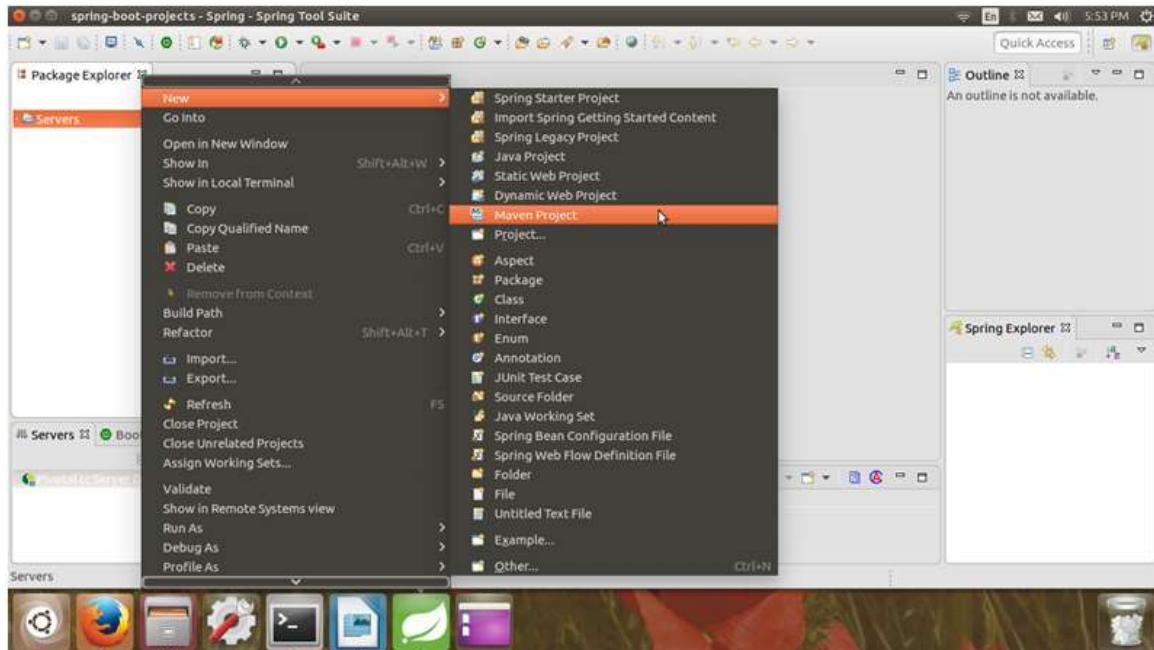
### **Advantages**

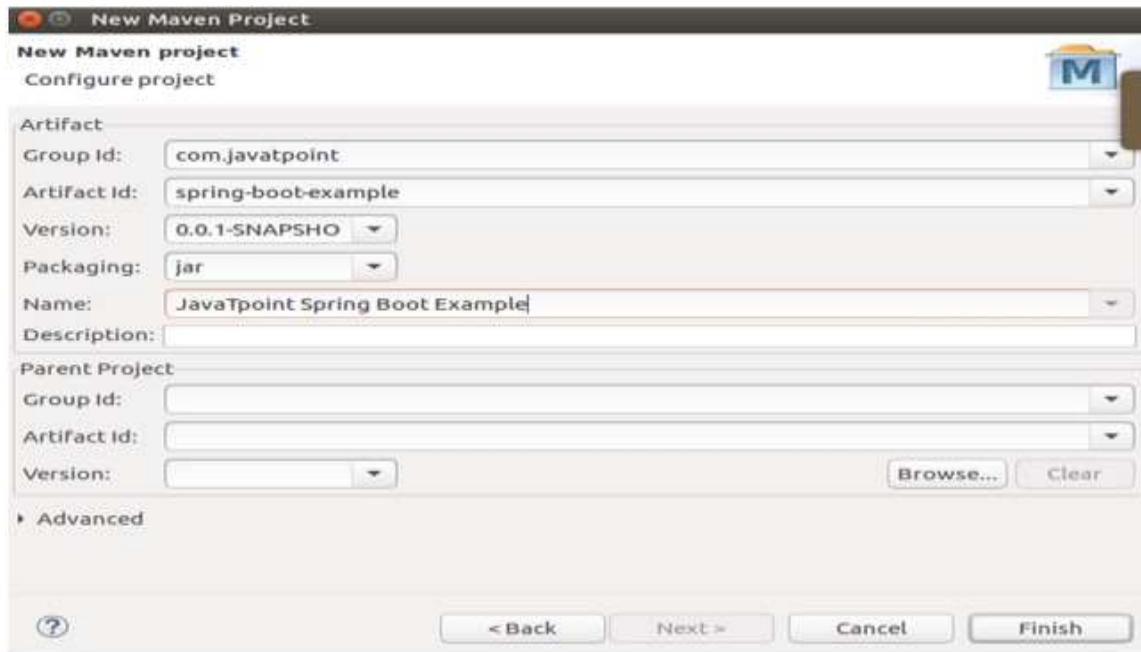
- Create stand-alone Spring applications that can be started using java -jar.
- Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
- It provides opinionated 'starter' POMs to simplify your Maven configuration.
- It automatically configure Spring whenever possible.
- It provides production-ready features such as metrics, health checks and externalized configuration.
- Absolutely no code generation and no requirement for XML configuration.

### **Spring Boot Project**

- There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.
- Spring Maven Project
- Spring Starter Project Wizard
- Spring Initializr
- Spring Boot CLI
- Creating Spring Boot project by creating maven project.
- It includes the following steps.
  - 1) Select project type.

# TOPS Technologies





We need to configure it in order to make it a Spring Boot project. Here, we are adding parent to our Maven project. It is used to declare that our project is a child to this parent project.

```
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>1.4.2.RELEASE</version>
</parent>
```

After that add the following dependency to the pom.xml file. Here, we are adding web dependency by adding spring-boot-starter-web.

Note - Maven project will add web dependency to the project by downloading the jar.

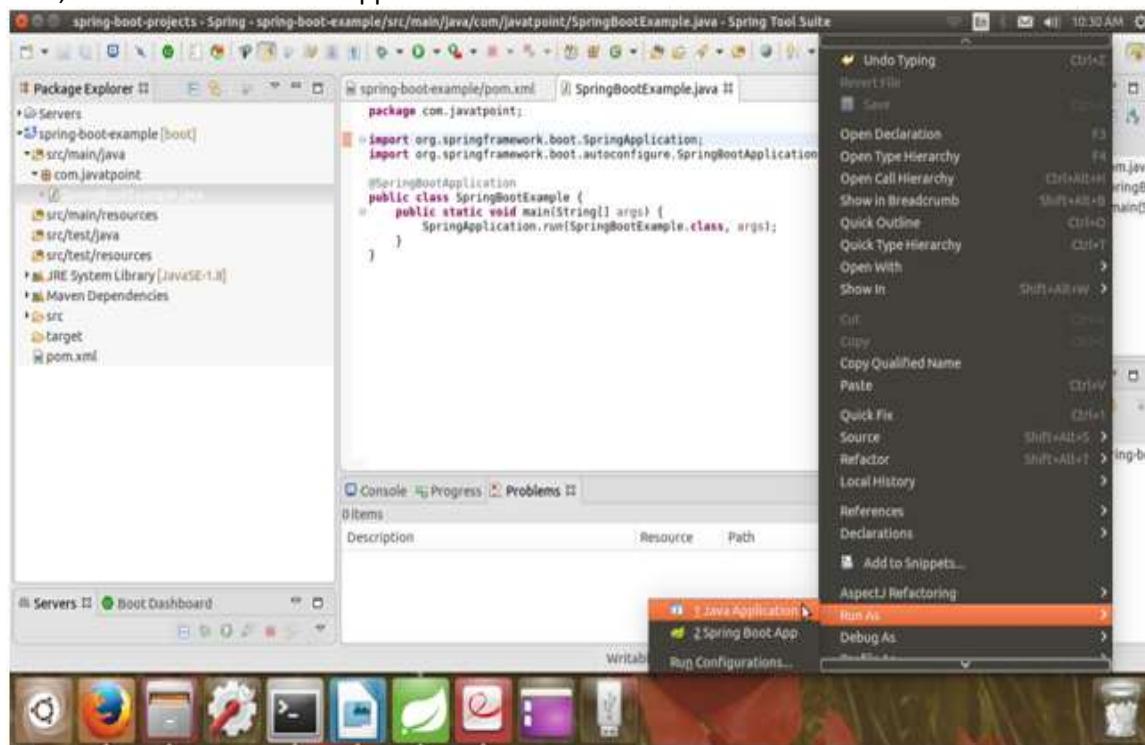
```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
</dependencies>
```

After creating class call the static run method of SpringApplication class. In the following code, we are calling run method and passing class name as argument.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootExample {
    public static void main(String[] args) {
        SpringApplication.run(SpringBootExample.class, args);
    }
}
```

Now, run this class as a Java Application.



It displays the following output.

```
\\ Spring Boot :: (v1.4.2.RELEASE)

2017-03-26 10:32:40.549 INFO 5735 --- [           main] com.javatpoint.SpringBootExample      : Starting SpringBootExample on irfan-
2017-03-26 10:32:40.579 INFO 5735 --- [           main] com.javatpoint.SpringBootExample      : No active profile set, falling back
2017-03-26 10:32:41.003 INFO 5735 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.
2017-03-26 10:32:47.462 INFO 5735 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 808
2017-03-26 10:32:47.542 INFO 5735 --- [           main] o.apache.catalina.core.StandardService : Starting service Tomcat
2017-03-26 10:32:47.546 INFO 5735 --- [           main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomc
2017-03-26 10:32:48.003 INFO 5735 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[]/ : Initializing Spring embedded WebAppl
2017-03-26 10:32:48.005 INFO 5735 --- [ost-startStop-1] o.s.web.context.ContextLoader       : Root WebApplicationContext: initiali
2017-03-26 10:32:48.828 INFO 5735 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet'
2017-03-26 10:32:48.849 INFO 5735 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'characterEncodingF
2017-03-26 10:32:48.851 INFO 5735 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'hiddenHttpMethodFil
2017-03-26 10:32:48.858 INFO 5735 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'httpPutFormContentF
2017-03-26 10:32:48.859 INFO 5735 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean  : Mapping filter: 'requestContextFilt
2017-03-26 10:32:50.131 INFO 5735 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.s
2017-03-26 10:32:50.363 INFO 5735 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[/{error},produces=[text/htm
2017-03-26 10:32:50.366 INFO 5735 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "[/{error}]" onto public org.
2017-03-26 10:32:50.509 INFO 5735 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/**] onto han
2017-03-26 10:32:50.515 INFO 5735 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**] onto handler c
2017-03-26 10:32:50.657 INFO 5735 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/**/favicon.ico] on
2017-03-26 10:32:51.261 INFO 5735 --- [           main] o.s.j.e.a.AnnotationBeanExporter        : Registering beans for JMX exposure c
2017-03-26 10:32:51.504 INFO 5735 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (htt
2017-03-26 10:32:51.594 INFO 5735 --- [           main] com.javatpoint.SpringBootExample      : Started SpringBootExample in 14.047
```

## Spring boot hibernate mysql example

Tools and Technologies Used

Spring Boot — 2.0.4.RELEASE

**JDK — 1.8 or later**

**Spring Framework — 5.0.8 RELEASE**

**Hibernate — 5.2.17.Final**

**JPA**

**Maven — 3.2+**

**IDE — Eclipse or Spring Tool Suite (STS)**

**Creating and Importing a Project**

There are many ways to create a [Spring Boot application](#). The simplest way is to use [Spring Initializr](#) <http://start.spring.io/>, which is an online Spring Boot application generator.

The screenshot shows the Spring Initializr interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below that, there's a search bar with "Generate a Maven Project with Java and Spring Boot 2.0.5".  
  
On the left, under "Project Metadata", there are fields for "Group" (set to "net.guides.springboot2") and "Artifact" (set to "springboot2-jpa-crud-example").  
  
On the right, under "Dependencies", there's a search bar with "Web, Security, JPA, Actuator, DevTools...". Below it, under "Selected Dependencies", are buttons for "Web", "DevTools", "MySQL", and "JPA".  
  
At the bottom center is a large green button labeled "Generate Project alt + e". Below the button, a note says "Don't know what to look for? Want more options? [Switch to the full version](#)".  
  
At the very bottom of the page, it says "start.spring.io is powered by Spring Initializr and Cloud Foundry Web Services".

Look at the From the above diagram, we have specified the following details:

**Generate: Maven Project**

**Java Version: 1.8 (Default)**

**Spring Boot:2.0.4**

**Group: net.guides.springboot2**

**Artifact: springboot2-jpa-crud-example**

**Name: springboot2-jpa-crud-example**

**Description: Rest API for a Simple Employee Management Application**

**Package Name : net.guides.springboot2.springboot2jpacrudexample**

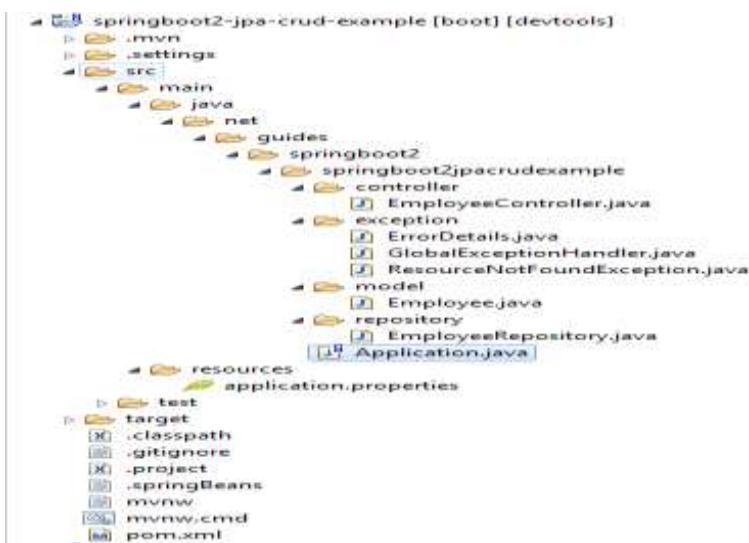
**Packaging: jar (This is the default value)**

**Dependencies: Web, JPA, MySQL, DevTools**

Once, all the details are entered, click on Generate Project button will generate a spring boot project and downloads it. Next, Unzip the downloaded zip file and import it into your favorite IDE.

**Packaging Structure**

The following is the packing structure of our Employee Management



Create JPA Entity — Employee.java

```
package net.guides.springboot2.springboot2jpacrudexample.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "employees")
public class Employee
{
    private long id;
    private String firstName;
    private String lastName;
    private String emailId;
    public Employee()
    {
    }
    public Employee(String firstName, String lastName, String emailId) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.emailId = emailId;
    }
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public long getId() {
        return id;
    }
    public void setId(long id) {
```

```
this.id = id;
}
@Column(name = "first_name", nullable = false)
public String getFirstName() {
    return firstName;
}
public void setFirstName(String firstName) {
    this.firstName = firstName;
}
@Column(name = "last_name", nullable = false)
public String getLastname() {
    return lastName;
}
public void setLastName(String lastName) {
    this.lastName = lastName;
}
@Column(name = "email_address", nullable = false)
public String getEmailId() {
    return emailId;
}
public void setEmailId(String emailId) {
    this.emailId = emailId;
}
@Override
public String toString() {
    return "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", emailId=" +
        emailId
    + "]";
}
}

Create Spring Data Repository — EmployeeRepository.java
package net.guides.springboot2.springboot2jpacrudexample.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import net.guides.springboot2.springboot2jpacrudexample.model.Employee;
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>
{
}

Create Spring Rest Controller — EmployeeController.java
@RestController
@RequestMapping("/api/v1")
public class EmployeeController
{
    @Autowired
    private EmployeeRepository employeeRepository;
    @GetMapping("/employees")
    public List<Employee> getAllEmployees() {
```

```
return employeeRepository.findAll();
}

@GetMapping("/employees/{id}")
public ResponseEntity<Employee> getEmployeeById(@PathVariable(value = "id") Long
employeeId) throws ResourceNotFoundException
{
Employee employee = employeeRepository.findById(employeeId)
.orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));
return ResponseEntity.ok().body(employee);
}
@PostMapping("/employees")
public Employee createEmployee(@Valid @RequestBody Employee employee)
{
return employeeRepository.save(employee);
}
@PutMapping("/employees/{id}")
public ResponseEntity<Employee> updateEmployee(@PathVariable(value = "id") Long employeeId,
@Valid @RequestBody Employee employeeDetails) throws ResourceNotFoundException {
Employee employee = employeeRepository.findById(employeeId)
.orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));
employee.setEmailId(employeeDetails.getEmailId());
employee.setLastName(employeeDetails.getLastName());
employee.setFirstName(employeeDetails.getFirstName());
final Employee updatedEmployee = employeeRepository.save(employee);
return ResponseEntity.ok(updatedEmployee);
}
@DeleteMapping("/employees/{id}")
public Map<String, Boolean> deleteEmployee(@PathVariable(value = "id") Long employeeId) throws
ResourceNotFoundException
{
Employee employee = employeeRepository.findById(employeeId)
.orElseThrow(() -> new ResourceNotFoundException("Employee not found for this id :: " + employeeId));
employeeRepository.delete(employee);
Map<String, Boolean> response = new HashMap<>();
response.put("deleted", Boolean.TRUE);
return response;
}
}
```

## Running Application

This spring boot application has an entry point Java class called *SpringBootCrudRestApplication.java* with the *public static void main(String[] args)* method, which you can run to start the application.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication
@SpringBootApplication
public class Application
{
public static void main(String[] args) {
```

```
SpringApplication.run(Application.class, args);
}
}
```

Testing REST APIs via Postman Client

1. Create Employee REST API

HTTP Method: POST

Request URL: <http://localhost:8080/api/v1/employees>

2. Get Employee by ID REST API

HTTP Method: GET

Request URL: <http://localhost:8080/api/v1/employees/11>

3. Get all Employees REST API

HTTP Method: GET Request URL: <http://localhost:8080/api/v1/employees>

4. Update Employee REST API

HTTP Method: GET

Request URL: <http://localhost:8080/api/v1/employees/7>

5. Delete Employee REST API

HTTP Method: DELETE

Request URL: <http://localhost:8080/api/v1/employees/11>

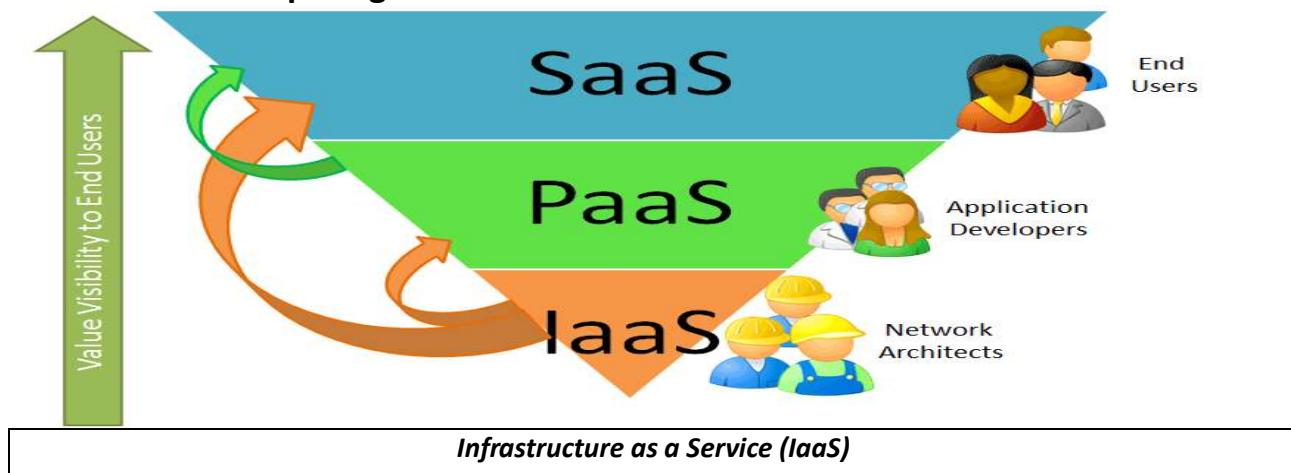
## Application Deployment on Cloud

### Introduction

#### What is Cloud Computing?

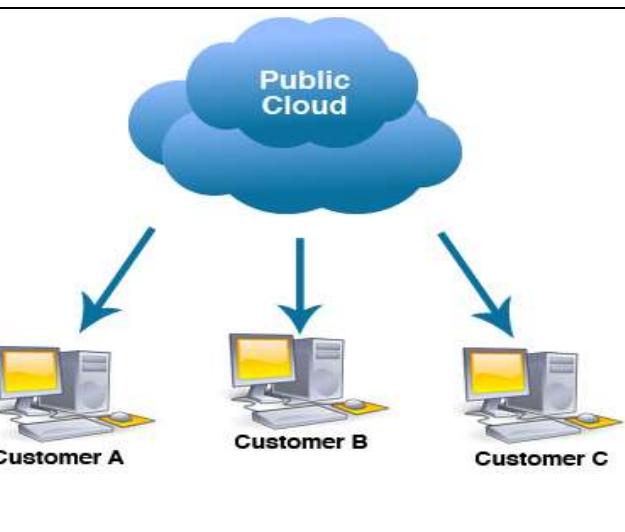
- *The “cloud” in cloud computing can be defined as the set of hardware, networks, storage, services, and interfaces that combine to deliver aspects of computing as a service.*
- *Cloud services include the delivery of software, infrastructure, and storage over the Internet (either as separate components or a complete platform) based on user demand.*

### Cloud Computing Models



<ul style="list-style-type: none"> <li>The IaaS layer offers storage and compute resources that developers and IT organizations can use to deliver business solutions</li> </ul>	
<b>Platform as a Service (PaaS)</b>	
<ul style="list-style-type: none"> <li>The PaaS layer offers black-box services with which developers can build applications on top of the compute infrastructure.</li> <li>This might include developer tools that are offered as a service to build services, or data access and database services, or billing services.</li> </ul>	
<b>Software as a Service (SaaS)</b>	
<ul style="list-style-type: none"> <li>In the SaaS layer, the service provider hosts the software so you don't need to install it, manage it, or buy hardware for it.</li> <li>All you have to do is connect and use it. SaaS Examples include customer relationship management as a service.</li> </ul>	

## Deployment of Cloud Services

Public Cloud	
<ul style="list-style-type: none"> <li>A public cloud is one in which the services and infrastructure are provided off-site over the Internet.</li> <li>These clouds offer the greatest level of efficiency in shared resources; however, they are also more vulnerable than private clouds. <ul style="list-style-type: none"> <li>Your standardized workload for applications is used by lots of people, such as e-mail.</li> </ul> </li> <li>You need to test and develop application code.</li> <li>You have SaaS (Software as a Service) applications from a vendor who has a well-implemented security strategy.</li> </ul>	

<ul style="list-style-type: none"> <li>• You need incremental capacity (the ability to add computer capacity for peak times).</li> <li>• You're doing collaboration projects.</li> <li>• You're doing an ad-hoc software development project using a Platform as a Service (PaaS) offering cloud.</li> </ul>	
<b>Private Cloud</b>	
<ul style="list-style-type: none"> <li>• A private cloud is one in which the services and infrastructure are maintained on a private network.</li> <li>• These clouds offer the greatest level of security and control, but they require the company to still purchase and maintain all the software and infrastructure, <ul style="list-style-type: none"> <li>• which reduces the cost savings</li> </ul> </li> <li>• Your business is your data and your applications. Therefore, control and security are paramount.</li> <li>• Your business is part of an industry that must conform to strict security and data privacy issues.</li> <li>• Your company is large enough to run a next generation cloud data center efficiently and effectively on its own.</li> </ul>	
<b>Hybrid Cloud</b>	
<ul style="list-style-type: none"> <li>• A hybrid cloud includes a variety of public and private options with multiple providers. By spreading things out over a hybrid cloud, you keep each aspect at your business in the most efficient environment possible.</li> <li>• The downside is that you have to keep track of multiple different security platforms and ensure that all aspects of your business can communicate with each other.</li> <li>• Your company wants to use a SaaS application but is concerned about security. Your SaaS vendor can create a private cloud just for your company inside their firewall. They provide you with a virtual private network (VPN) for additional security.</li> <li>• Your company offers services that are tailored for different vertical markets. You can use a public cloud to interact with the</li> </ul>	

*clients but keep their data secured within a private cloud.*

## Steps for Cloud Application Deployment

*Create an "Application" in OpenShift (With the command-line or via their IDE)*

*Code the application (in Vi, TextMate, Eclipse, Visual Studio, or whatever)*

*Push the application code to OpenShift (again, with the command-line or from their IDE)*



**1. Sign Up Open shift account and verify the account with your e-mail**

**2. Login The Account with Your Username and Password**

**3. To create an application from the Create Application- Menu Item**

The screenshot shows the 'CREATE AN APPLICATION' interface. At the top, there are three steps: 1. Choose a type of application (highlighted with a red box), 2. Configure and deploy the application, and 3. Next steps. Below this, a section titled 'Create your first application now!' displays a search bar and a browse by tag dropdown. To the right, there are two sections: 'Cartridge' (a managed runtime) and 'QuickStart' (a quick way to try out new technologies). Under 'Featured', there are cards for 'JBoss Enterprise Application Platform 6.0' and 'Zend Server 5.6'. In the 'Instant App' section, there are cards for Jenkins Server 1.4, Drupal 7, WordPress 3.x, PHP (with a green box around it and the text 'Click on the Image'), Instant App (PHP 5.3), CakePHP, and Reveal.js. On the right, under the 'Java' heading, there are cards for JBoss Application Server 7, Tomcat 6 (JBoss EWS 1.0), Tomcat 7 (JBoss EWS 2.0) (highlighted with a blue box), CapeDwarf, JEE Full Profile on JBoss, and Spring Framework on JBoss EAP6.

1 Choose a type of application      2 Configure and deploy the application      3 Next steps

**Based On** Tomcat 7 (JBoss EWS 2.0) Cartridge

JBoss Enterprise Web Server is the enterprise-class Java web container for large-scale lightweight web applications based on Tomcat 7. Build and deploy JSPs and Servlets in the cloud.  
<http://www.redhat.com/products/jbossenterprisemiddleware/web-server/>

OpenShift maintained  
 JAVA    JBOSS    TOMCAT    TOMCAT7

**Public URL** http:// Application Name -vish7090.rhcloud.com Insert App Name

OpenShift will automatically register this domain name for your application. You can add your own domain name later.

**Source Code** Default Change

We'll create a Git code repository in the cloud, and populate it with a set of reasonable defaults.

**Gears** Small

Gears are the application containers running your code. For most applications, the 'small' gear size provides plenty of resources.

**Cartridges** Tomcat 7 (JBoss EWS 2.0)

**Public URL** http:// test -vish7090.rhcloud.com

OpenShift will automatically register this domain name for your application. You can add your own domain name later.

**Source Code** Default Change

We'll create a Git code repository in the cloud, and populate it with a set of reasonable defaults.

**Gears** Small

Gears are the application containers running your code. For most applications, the 'small' gear size provides plenty of resources.

**Cartridges** Tomcat 7 (JBoss EWS 2.0)

Applications are composed of cartridges - each of which exposes a service or capability to your code. All applications must have a web cartridge.

**Scaling** No scaling Change

OpenShift automatically routes web requests to your web gear. If you allow your application to scale, we'll set up a load balancer and allocate more gears to handle traffic as you need it.

Back Create Application

Waiting for openshift.redhat.com...

1 Choose a type of application      2 Configure and deploy the application      3 Next steps

Your application has been created. If you're new to OpenShift check out these tips for where to go next.

**Accessing your application**

Your application has one or more cartridges that expose a public URL. Click the link below to see your application.

**Open URL**

<http://test-vish7090.rhcloud.com/>

The application overview page provides a summary of your application and its cartridges.

**Making code changes**

OpenShift uses the Git version control system to manage the code of your application. Each cartridge has a single Git repository that you'll use to check in changes to your application. When you push a change to your Git repository we'll automatically deploy your code and restart your application if necessary.

Install the Git client for your operating system, and from your command line run

```
git clone ssh://51d134065973cad0b60000e8@test-vish7090.rhcloud.com/~/.git/test.git/
```

**Adding capabilities**

Cartridges are the components of an OpenShift application, and include databases, build systems, and management capabilities. Adding a cartridge such as MySQL or MongoDB to an application provides the desired capability without forcing you to administrate or update that feature.

Add a cartridge to your application now

**Managing your application**

**RHC Client Tools**

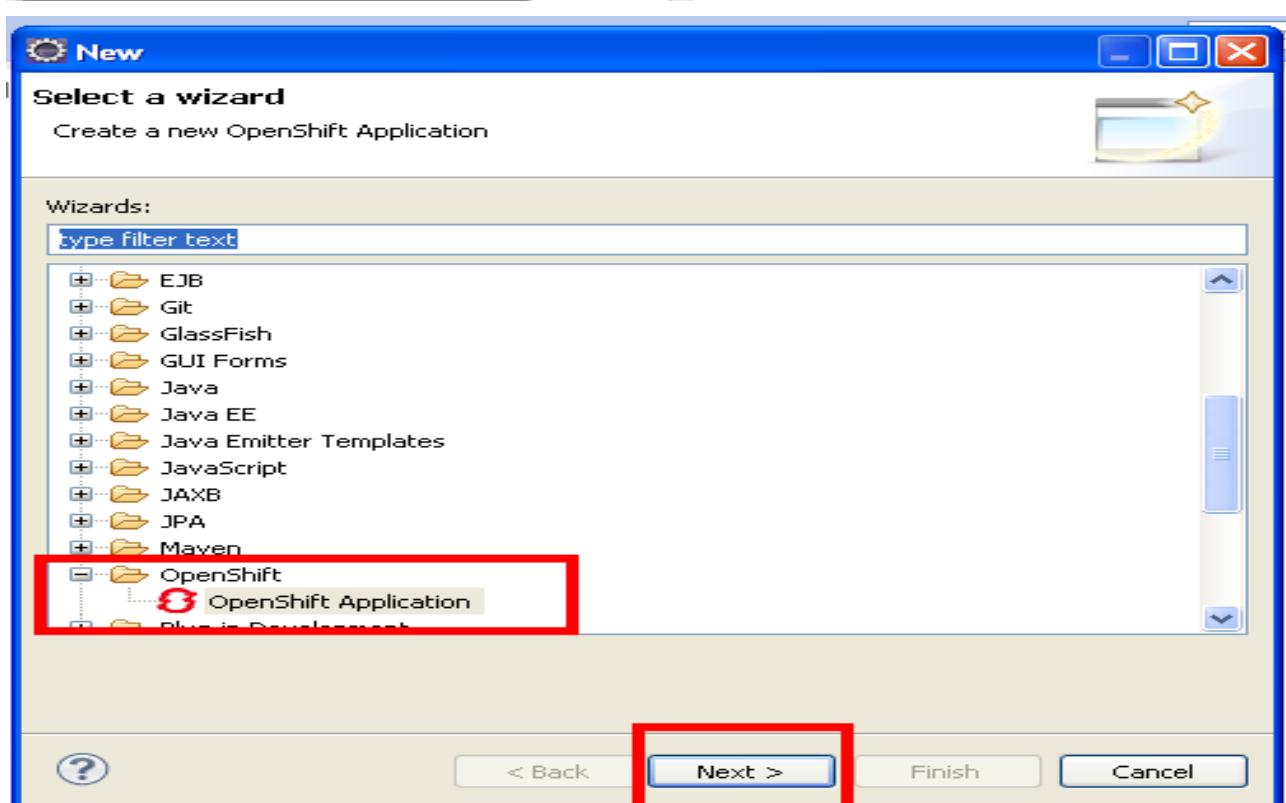
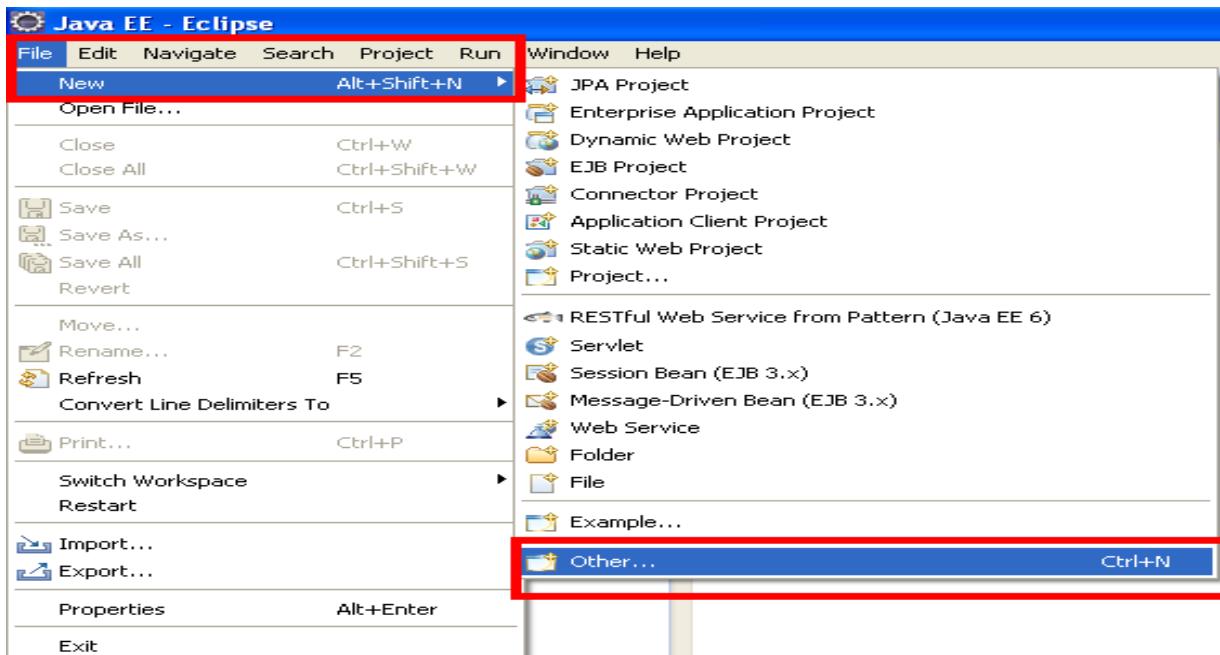
Most of the capabilities of OpenShift are exposed through our command line tool, rhc. Whether it's adding cartridges, checking uptime, or pulling log files from the server, you can quickly put a finger on the pulse of your application. Follow these steps to install the client on Linux, Mac OS X, or Windows.

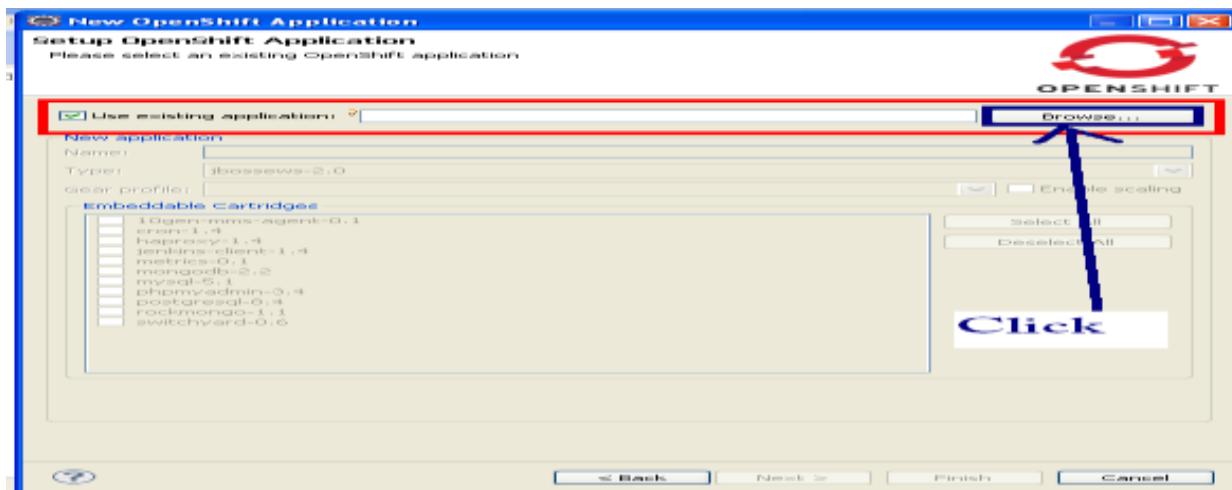
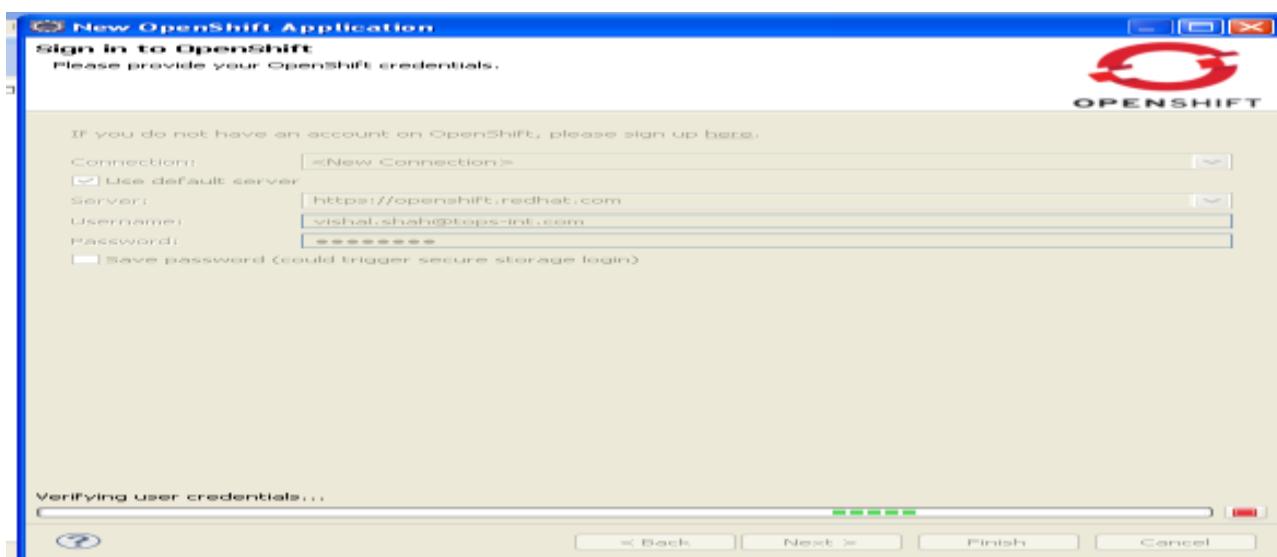
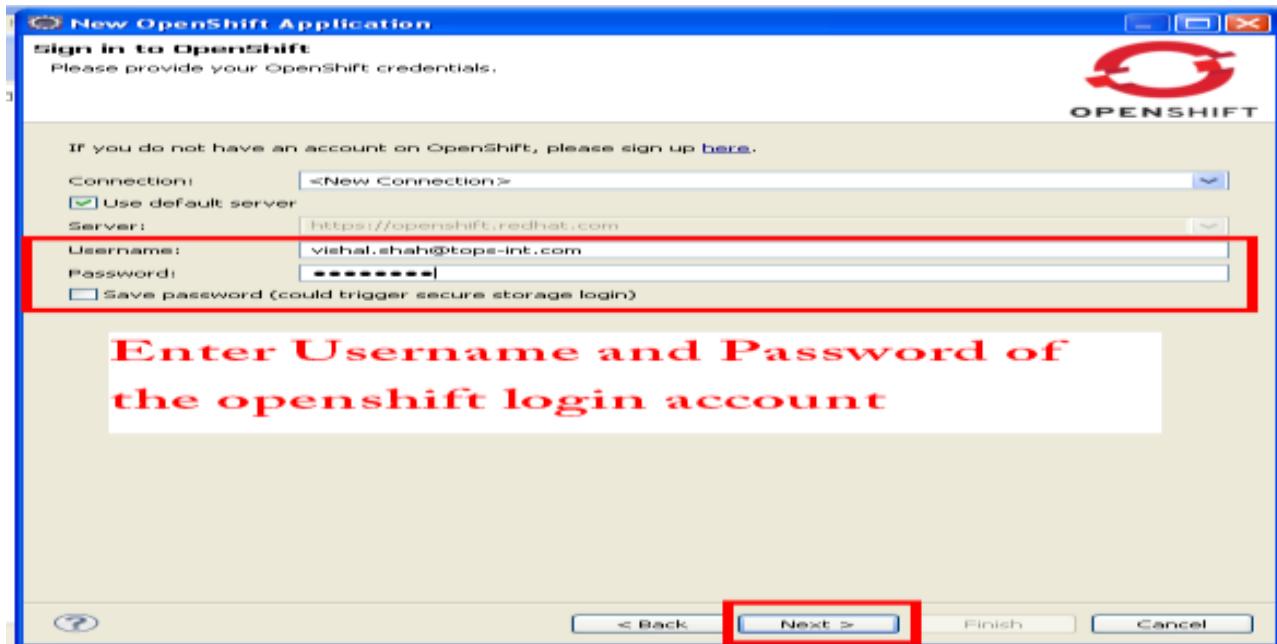
After installing the command line tool read more on how to manage your application from the command line in

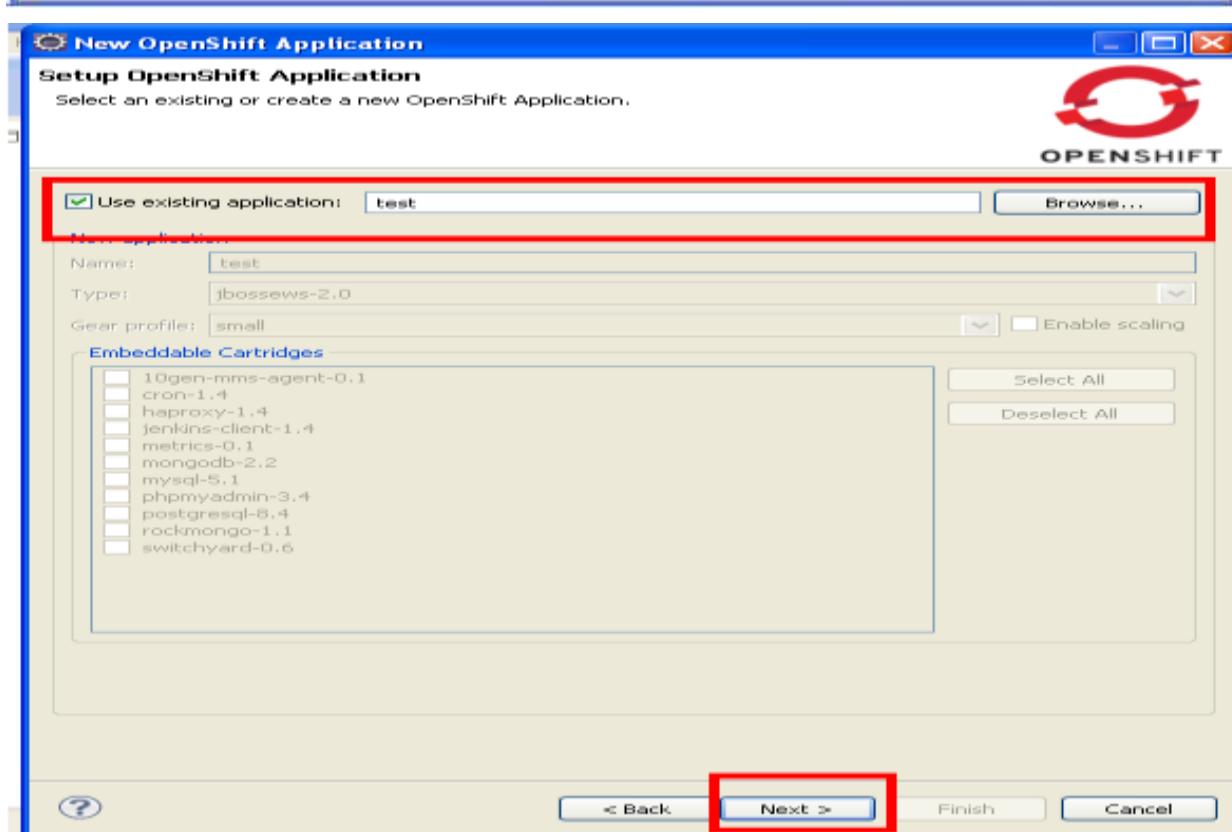
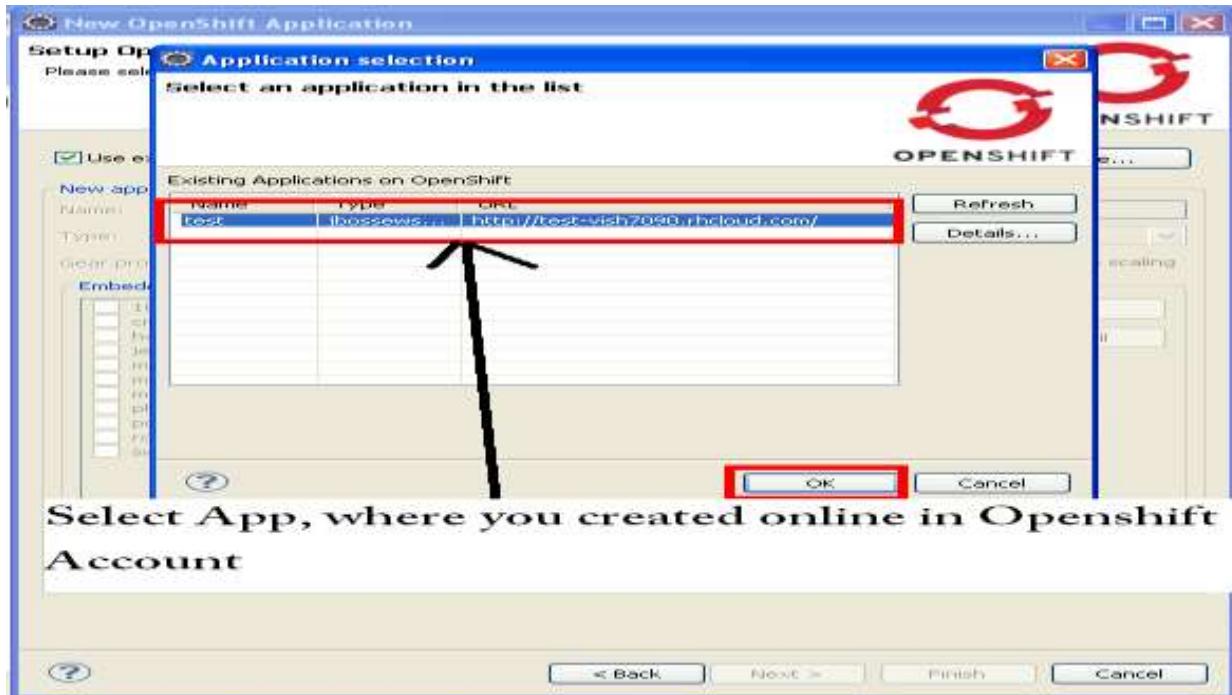
## Start configuring and deploying in IDE

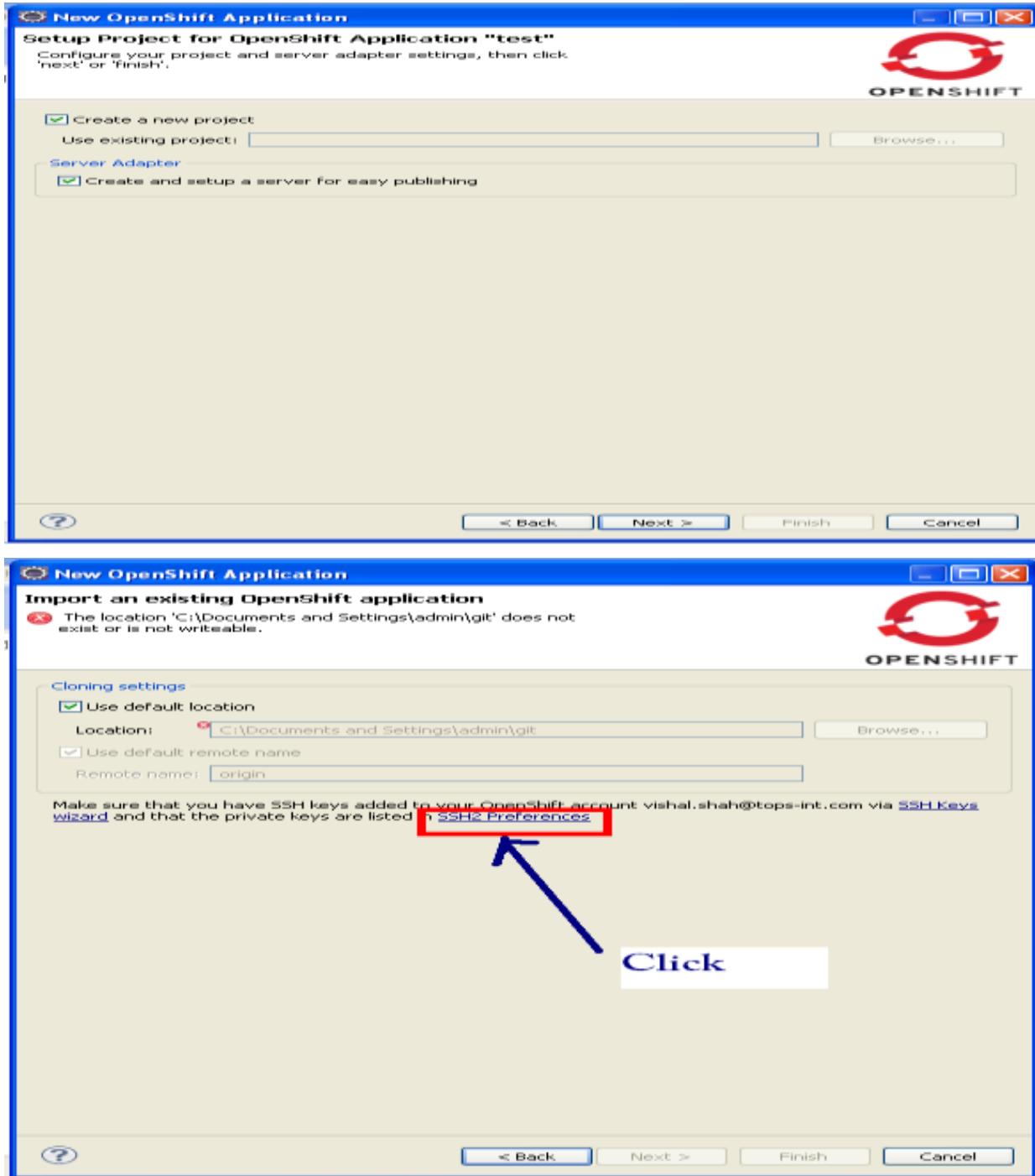
- Install Software Plugin in Eclipse :
  - JBoss - <http://download.jboss.org/jbosstools/updates/stable/juno/>

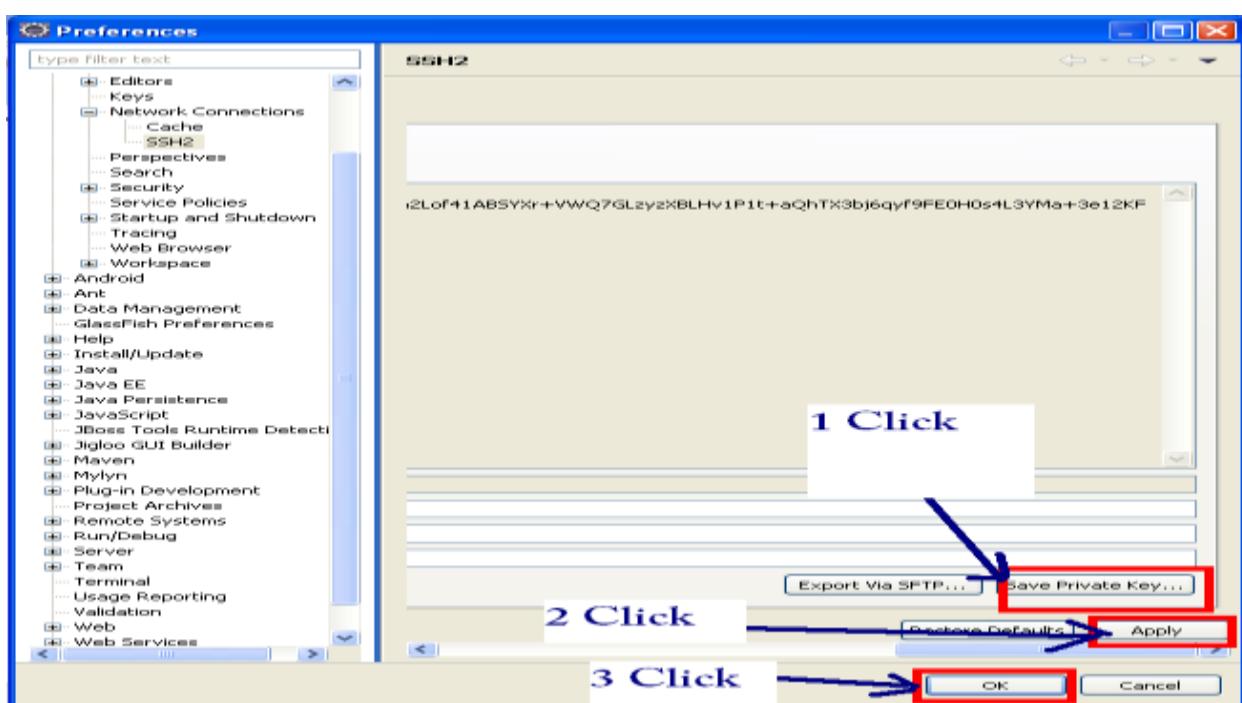
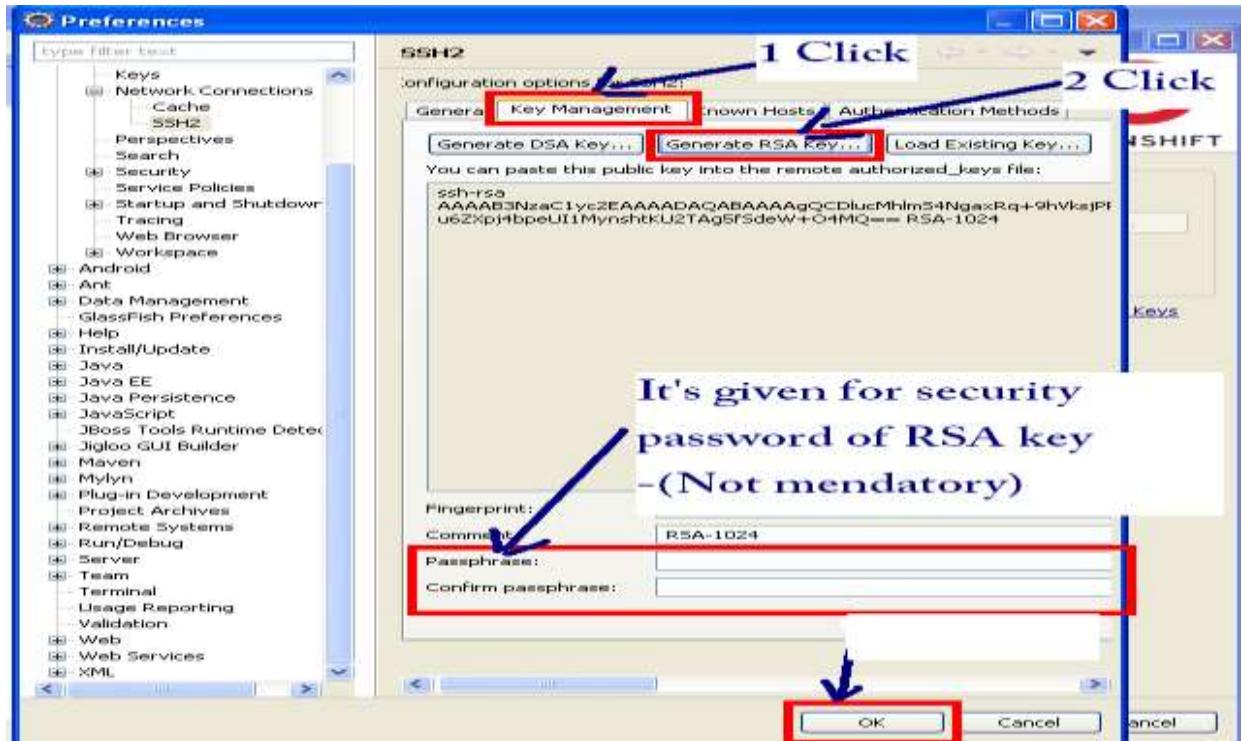
- *Cloud Development Tool > Jboss Openshift Tool*
- *New > Projects > Openshift > Openshift Application*

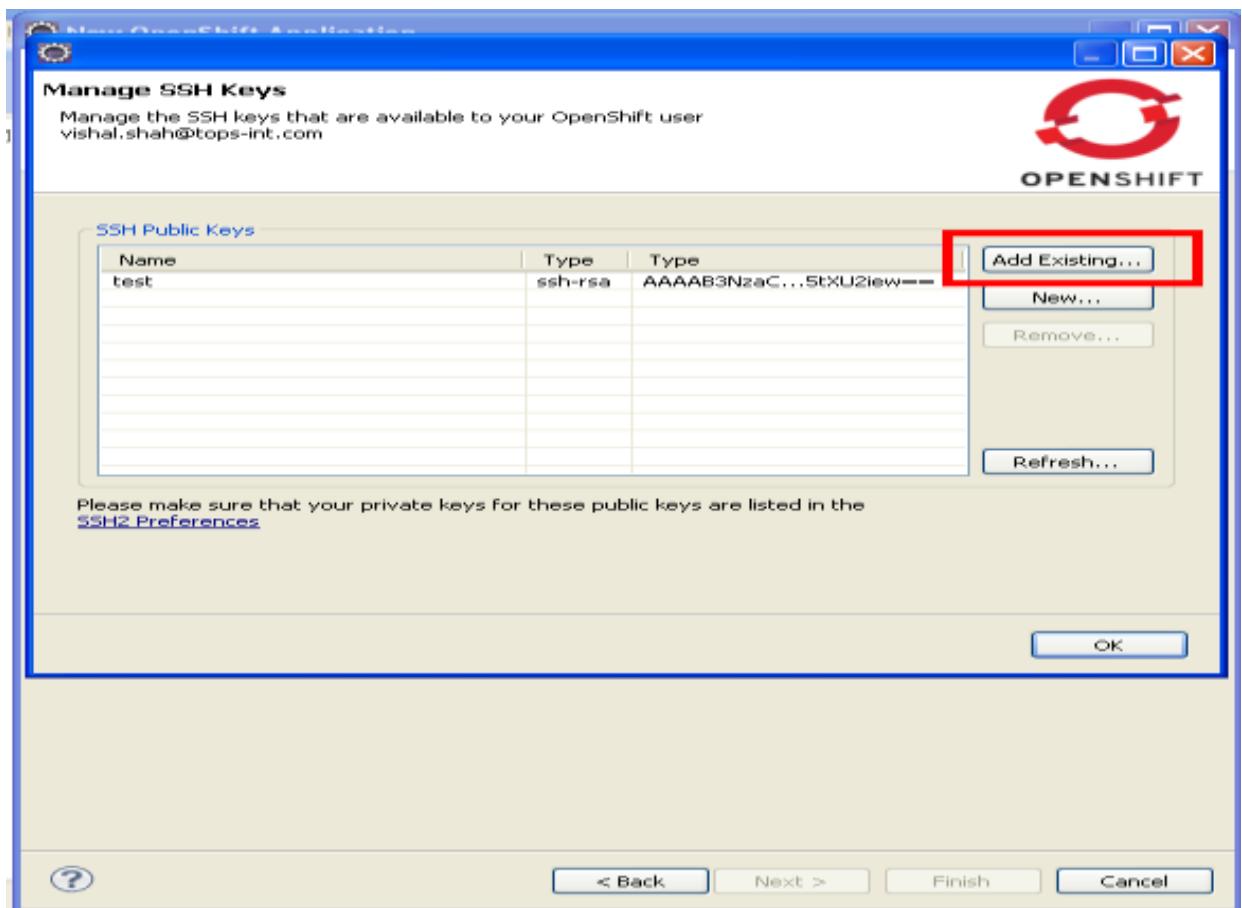
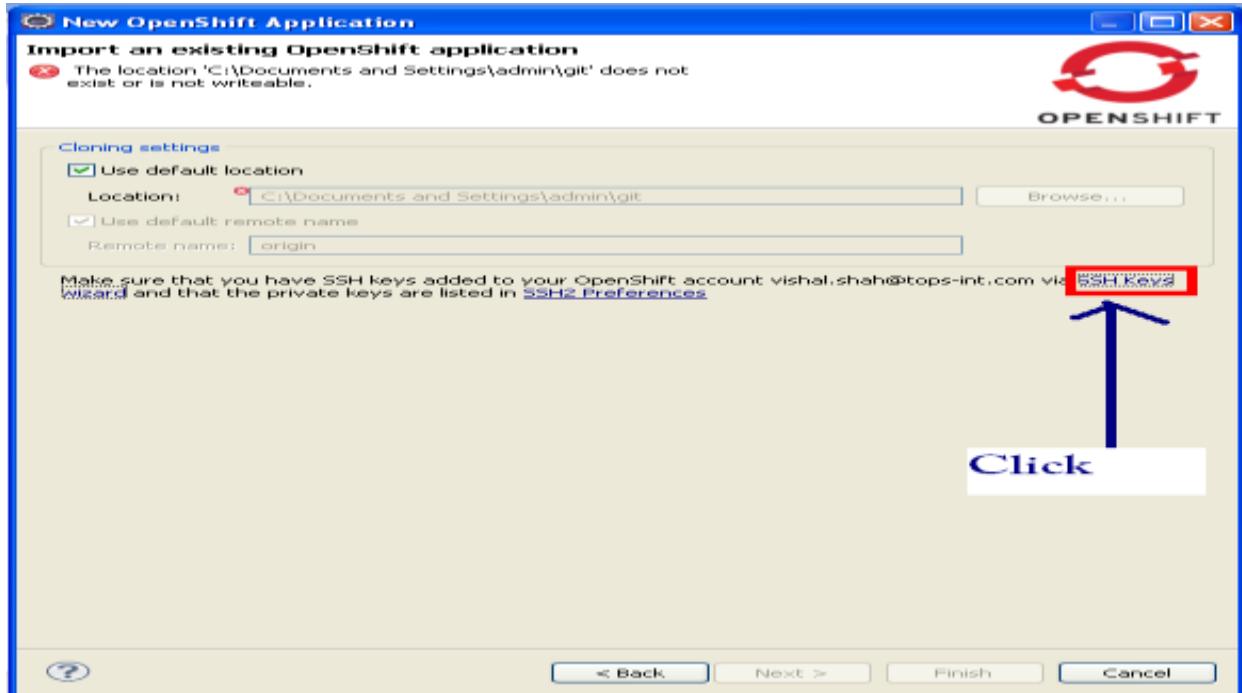












The screenshot shows the 'Add existing SSH Key' dialog box. It has a warning message at the top: 'Could not find the private key for your public key in the preferences. Make sure it is listed in the ssh2 preferences.' Below this, there is a form with a 'Name:' field containing 'testing' and a 'Public Key:' field containing the path 'C:\Documents and Settings\admin\Desktop\gitapp'. A red box highlights the 'Browse...' button next to the public key field. A blue arrow points from the text 'Browse created public key of RSA generated key' to this button. Another blue arrow points from the text 'Click' to the 'Finish' button, which is also highlighted with a red box. The 'Cancel' button is also visible.

**Browse created public key of RSA generated key**

**Click**

**Finish**

**Cancel**

**OK**

The screenshot shows the 'Manage SSH Keys' dialog box. It displays a table of 'SSH Public Keys' with two entries: 'test' and 'testing'. Both entries have 'ssh-rsa' as the type and a long string of characters as the key. A red box highlights the entire table row for 'testing'. On the right side of the dialog, there are buttons for 'Add Existing...', 'New...', 'Remove...', and 'Refresh...'. At the bottom, there is a note: 'Please make sure that your private keys for these public keys are listed in the [SSH2 Preferences](#)'. The 'OK' button is highlighted with a red box.

**Manage SSH Keys**

Manage the SSH keys that are available to your OpenShift user  
vishal.shah@tops-int.com

**SSH Public Keys**

Name	Type	Type
test	ssh-rsa	AAAAAB3NzaC1...5tXU2iew==
testing	ssh-rsa	AAAAAB3NzaC1...jZ2t5HWW==

**Add Existing...**

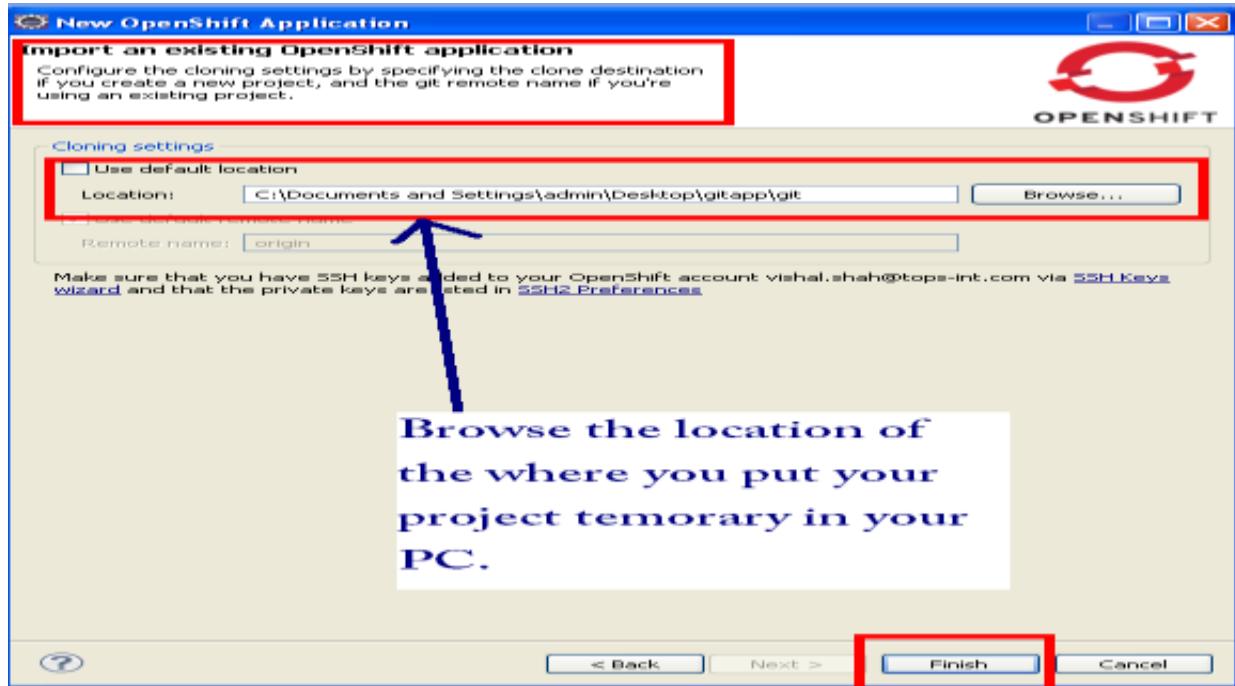
**New...**

**Remove...**

**Refresh...**

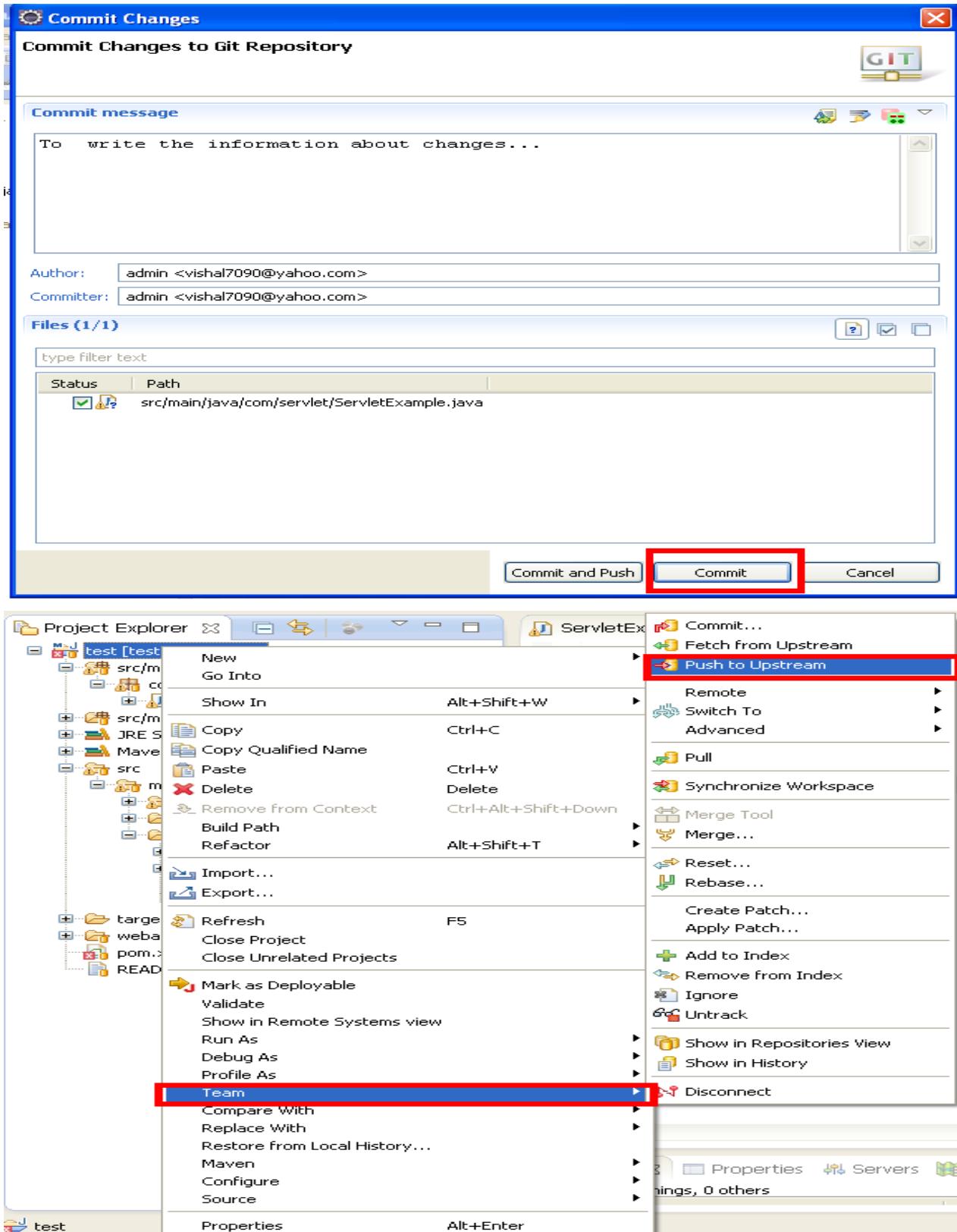
Please make sure that your private keys for these public keys are listed in the [SSH2 Preferences](#)

**OK**



To upload the application and application's jsp pages and servlets and all other files in the cloud by using the commit and put to upstream.





**WebApp JSP Snoop page**

**JVM Memory Monitor**

**Memory MXBean**

Heap Memory Usage	init = 67108864(65536K) used = 67235672(65659K) committed = 120782848(117952K) max = 238616576(233024K)
Non-Heap Memory Usage	init = 19136512(18688K) used = 15936416(15562K) committed = 19136512(18688K) max = 157286400(15360C)

**Memory Pool MXBeans**

Code Cache	
Type	Non-heap memory
Usage	init = 2359296(2304K) used = 1086016(1060K) committed = 2359296(2304K) max = 50331648(49152K)
Peak Usage	init = 2359296(2304K) used = 1096320(1070K) committed = 2359296(2304K) max = 50331648(49152K)
Collection Usage	null

PS Eden Space	
Type	Heap memory
Usage	init = 16842752(16448K) used = 42783824(41781K) committed = 66977792(65408K) max = 70975488(69312K)

## Interview Question:

1. What gives Java its 'write once and run anywhere' nature?
2. Which two method you need to implement for key Object in HashMap ?
3. What is immutable object? Can you write immutable object?
4. What is the difference between creating String as new() and literal?
5. What is difference between StringBuffer and StringBuilder in Java ?
6. What is the difference between ArrayList and Vector ?
7. Can we make array volatile in Java?
8. The difference between sleep and wait in Java?
9. What is the right data type to represent a price in Java?
10. Can we cast an int value into byte variable? what will happen if the value of int is larger than byte?
11. Is ++ operator is thread-safe in Java?
12. Difference between a = a + b and a += b ?
13. Which one will take more memory, an int or Integer?
14. What is constructor chaining in Java?
15. What is the size of int in 64-bit JVM?
16. What are the difference between JRE, JDK, JVM and JIT?
17. What is the difference between stack and heap in Java?
18. Difference between final, finalize and finally?
19. The difference between abstract class and interface in Java?
20. How can we create deadlock condition on our servlet?
21. How can we refresh servlet on client and server side automatically?
22. What are the main steps in java to make JDBC connectivity?

23. What are different types of Statement?
24. Difference between extends Thread vs implements Runnable in Java?
25. Can we have return statement in finally clause? What will happen?
26. Can you override static method in Java?
27. What is difference between Checked and Unchecked Exception in Java?
28. What are the advantages of Hibernate over JDBC?
29. Can a constructor be made final?
30. What is static block?
31. What is final class?
32. What are the ways in which a thread can enter the waiting state?
33. What is a transient variable?
34. What is runtime polymorphism or dynamic method dispatch?
35. What is Dynamic Binding(late binding)?
36. Which method must be implemented by all threads?
37. What is the difference between the size and capacity of a Vector?
38. Can we create Thread without extending thread and Implementing Runnable ?
39. Can abstract class have constructor in Java?
40. What is difference between hide comment and output comment?
41. What is EL in JSP?
42. Can an interface be implemented in the jsp file ?
43. What is interceptor?
44. Is there any difference between nested classes and inner classes?
45. What is MySQL default port number?
46. How many columns can you create for an index?
47. What if I write static public void instead of public static void?
48. What will be the initial value of an object reference which is defined as an instance variable?
49. Is constructor inherited?
50. Why main method is static?