B.SC III Year
Programming in JAVA
UNIT I

Syllabus:

Unit – 1: Introduction to Java

Basic Concepts: Introduction, Java features, Java Virtual Machine, Java Program Structure, Command Line Arguments, Type Casting.

Closer look at Methods and classes: Defining a Class, Creating Objects, Accessing class members, Constructors, Garbage Collection, finalize () method, Static Members, Final Variables and Methods, Final Classes, Abstract Methods and Classes, String Handling, Inheritance, Exceptions Handling.

Unit – 2: Packages, Interfaces & Multi-threading

Packages: Built-in Packages (java.awt, java.io, java.lang, java.math, java.sql, java.util), Creating User Defined Packages, Accessing a Package, Using a Package.

Interfaces: Defining Interfaces, Extending Interfaces, Implementing Interfaces, Accessing Interface Variables.

Threading: Introduction, creating threads, extending the thread class, Life cycle of thread, thread methods, threads priority, implementing thread using runnable interface.

Unit – 3: Java Server Pages:

JSP: Introduction, Architecture of JSP, Life Cycle of JSP, Scripting elements (Scriplets, JSP Declarations, JSP Expression),Directive Elements (page, include, taglib),

JSP Actions (include, setproperty, getproperty, forward, text), Implicit objects (request, response, out, page, Exception), including HTML in JSP

Unit -4: Interacting with Database:

Introduction to JDBC, Essential JDBC classes, Connecting to database, Inserting data in database, Retrieving data from database, deleting data in database, updating data in database, store image in the database, to retrieve image from database, to store file in database, retrieve file from database.

References:

1. Herbert Schildt, The Complete Reference Java2.0, Fifth edition, TATA McGraw-Hill Company.
2. Phil Hanna, JSP : Complete Reference, TATA McGraw-Hill Company
3. Debasish Jana, Java and Object-Oriented programming Paradigm, PHI.
4. Jana, Java and Object Oriented Programming Paradigm, PHI (2007).

B.SC III Year
Programming in JAVA
UNIT I

B.Sc. (Computer Science)

III - Year / V - Semester

PRACTICAL PAPER – VII

Programming in Java Lab

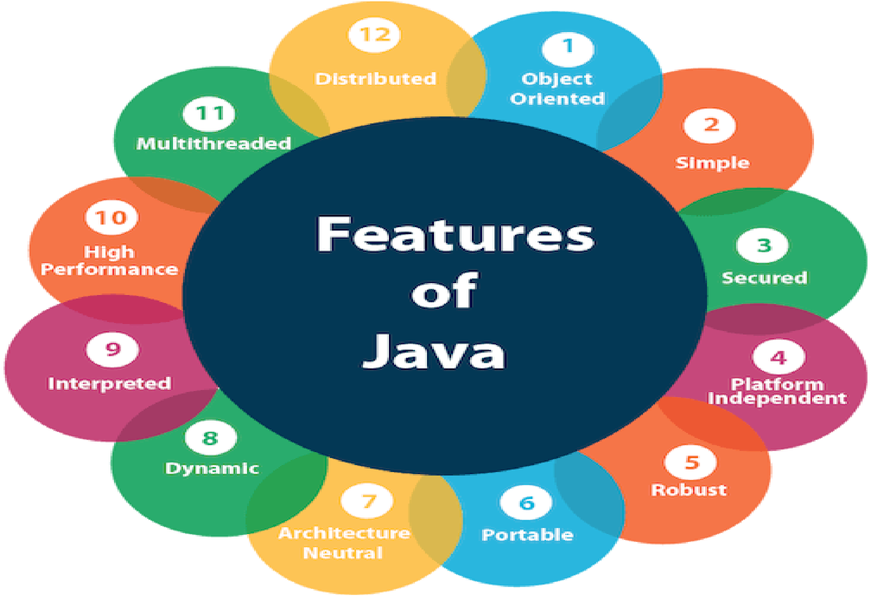
Subject Code: BS. 07.201.23.P

1. Write a Java Program to demonstrate constructors.
2. Write a Java program to practice using String class and its methods.
3. Implementing an exception called 'MarksOutOfBoundsException' that is thrown if entered marks greater than 100.
4. Write a java program to demonstrate Packages.
5. Write a program to demonstrate use of implementing interfaces.
6. Write a program to demonstrate use of extending interfaces.
7. Write a java program to demonstrate threads.
8. Write a java to implementing thread using runnable interface.
9. Installation of Tomcat Server.
10. Create a table which should contain at least the following fields: name, password, email-id, phone number Write a JSP to connect to that database and extract data from the tables and display them. Insert the details of the users who register with the web site, whenever a new user clicks the submit button in the registration page.
11. Create a table which should contain the following fields: name, password, email-id, phone number (these should hold the data from the registration form).Write JSP to connect to that database and extract data from the tables and display them.
12. Write a JSP which does the following job: Insert the details of the 3 or 4 users who register with the web site by using registration form. Authenticate the user when he submits the login form using the user name and password from the database and display the message "successfully logged in".
13. Write a JSP Program to delete data in database.
14. Write a JSP Program store image and file in the database.
15. Write a JSP Program to retrieve image and file from database.

B.SC III Year
Programming in JAVA
UNIT I

Features of Java

A list of most important features of Java language is given below.



1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

<u>Simple</u>

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

B.SC III Year
Programming in JAVA
UNIT I

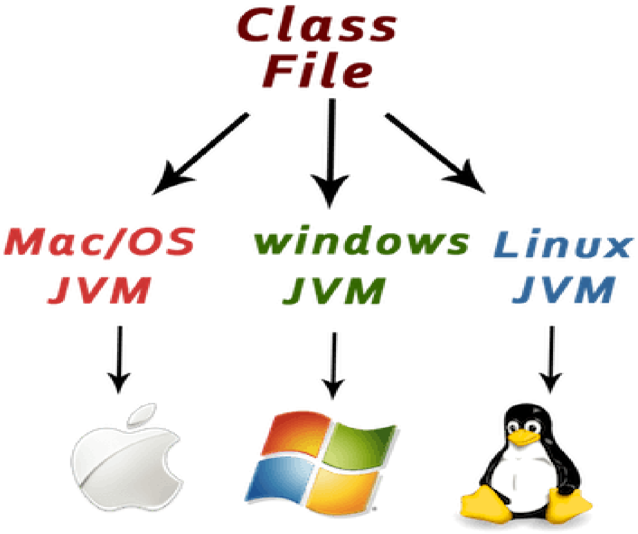
Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent



Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

Java code can be run on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere(WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer

- Java Programs run inside a virtual machine sandbox

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust:

Robust simply means strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral:

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

Portable: Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

High-performance:

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed: Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded:A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic: Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

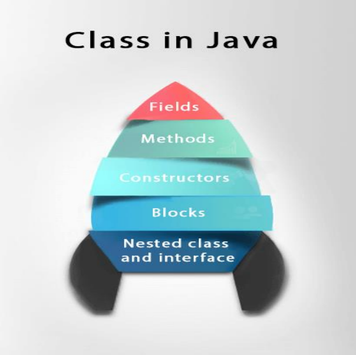
B.SC III Year
Programming in JAVA
UNIT I

Classes and Objects:

Classes

- A Class is a collection of data members and member functions. Data members are used to store information; member functions are used to perform operations on data.
- A class is a group of objects which have common properties.
- It is a template or blueprint from which objects are created.

A class in Java can contains:



Syntax to declare a class:

```
class   ClassName
{
  Fileds or Data members;
 Methods or member functions;
}
```

Field declarations Syntax:

```
Class classname
{
Datatype variablename1;
Datatype variablename2;
.
.
.
Datatype variablenamen;
}
```

Method declaration Syntax:

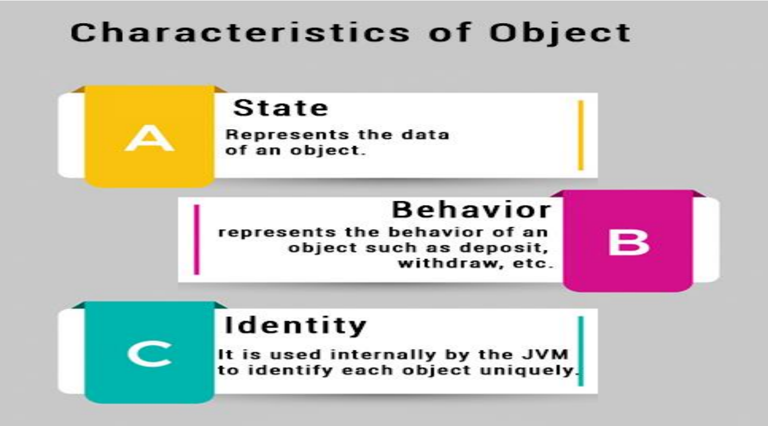
```
            Returntype   methodname(ArgumentsList)
            {
            Method body;
            }
```

Object:

An entity that has state and behaviour is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical.

An object has three characteristics:

- ✓ State : Represents the data (value) of an object.
- ✓ Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- ✓ Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.


Characteristics of Object
A
State
Represents the data of an object.
Behavior
represents the behavior of an object such as deposit, withdraw, etc.
B
C
Identity
It is used internally by the JVM to identify each object uniquely.

Syntax to declare a object:

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

Programs: Programs on Classes and objects refer the class notes.

B.SC III Year
Programming in JAVA
UNIT I

Write briefly about Type Casting:

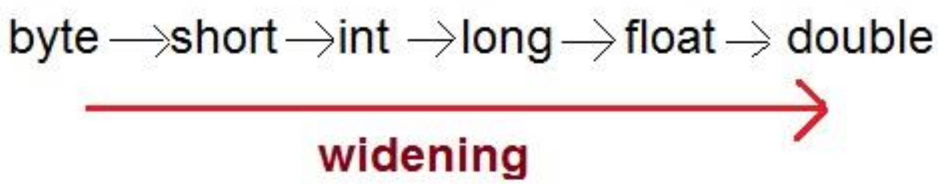
- ✓ Converting one data type variable value into another data type variable is called type casting.
  (OR)
- ✓ Assigning a value of one type to a variable of another type is known as **Type Casting**.

Example:

```
int x = 10;
byte y = (byte)x;
```

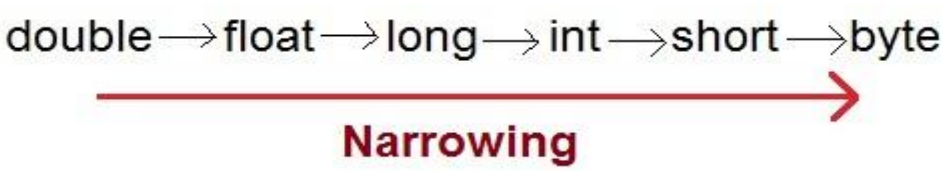
In Java, type casting is classified into two types,

- **Widening Casting(Implicit)**



Here we are converting byte to short, short to int, int to long, long to float and float to double.

- **Narrowing Casting (Explicitly done)**



Here we are converting Here we are converting double to float, float to long, long to int, int to short and short to byte.

Widening or Automatic type converion

Automatic Type casting take place when,

- The two types are compatible

- The target type is larger than the source type

Example :

```
class Test
{
    public static void main(String[] args)
    {
      int i = 100;
      long l = i;              //no explicit type casting required
      float f = l;        //no explicit type casting required
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
    }

}
```

OUTPUT:

```
            Int value 100
             Long value 100
            Float value 100.0
```

Narrowing or Explicit type conversion:

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

Example :

```
class Test

{
    public static void main(String[] args)
    {
      double d = 100.04;
      long l = (long)d;  //explicit type casting required
      int i = (int)l;     //explicit type casting required

      System.out.println("Double value "+d);
      System.out.println("Long value "+l);
      System.out.println("Int value "+i);
    }
}
```

OUTPUT:

```
Double value 100.04
Long value 100
Int value 100
```

Write briefly about Garbage Collection

- ✓ In Java destruction of object from memory is done automatically by the JVM.
- ✓ This technique is called **Garbage Collection**. This is accomplished by the JVM. Unlike C++ there is no explicit need to destroy object.

Advantages of Garbage Collection

1. Programmer doesn't need to worry about dereferencing an object.
2. It is done automatically by JVM.
3. Increases memory efficiency and decreases the chances for memory leak.

finalize() method

- ✓ Sometime an object will need to perform some specific task before it is destroyed such as closing an open connection or releasing any resources held. To handle such situation **finalize()** method is used.
- ✓ **finalize()** method is called by garbage collection before collecting object.

Syntax of **finalize()** method

```
protected void finalize()
{
            //finalize-code
}
```

Note:

1. finalize() method is defined in **java.lang.Object** class, therefore it is available to all the classes.
2. finalize() method is declare as **proctected/public** inside Object class.
3. finalize() method gets called only once by a thread named GC (Garbage Collector)thread.

gc() Method

gc() method is used to call garbage collector explicitly. It only requests the JVM for garbage collection. This method is present in **System** and **Runtime** class.

Example for gc() method.

```
public class Test
{

   public static void main(String[] args)
   {
      Test t = new Test();
      t=null;
      System.gc();
   }
   public void finalize()
   {
      System.out.println("Garbage Collected");
   }
}
```

OUTPUT: Garbage Collected

Can the Garbage Collection be forced explicitly ?

No, the Garbage Collection can not be forced explicitly. We may request JVM for **garbage collection** by calling **System.gc()** method.

<u>Abstract Methods and classes:</u>

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

<u>Ways to achieve Abstraction:</u>

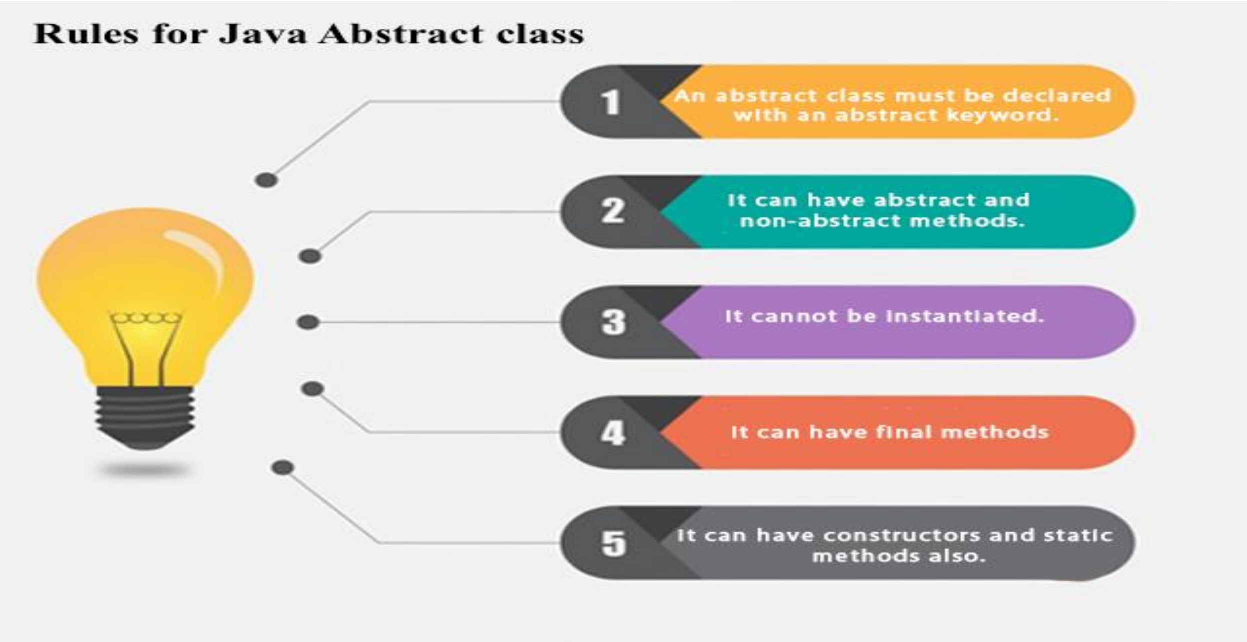
There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class in Java

A class which is declared as abstract is known as an **abstract class**. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

- ✓ An abstract class must be declared with an abstract keyword.
- ✓ It can have abstract and non-abstract methods.
- ✓ It cannot be instantiated.
- ✓ It can have constructors and static methods also.
- ✓ It can have final methods.


Rules for Java Abstract class
1
An abstract class must be declared with an abstract keyword.
2
It can have abstract and non-abstract methods.
3
It cannot be instantiated.
4
It can have final methods
5
It can have constructors and static methods also.

Syntax:

```
abstract class classname

{

Body of the abstract class

}
```

Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

Syntax:

Abstract returntype methodname (ArgumentsList); //no method body

Program: Write a Java Program on abstract methods and abstract classes

In this example, Bank is an abstract class that contains only one abstract method getRateOfInterset( ). Its implementation is provided by the SBI,PNB classes.

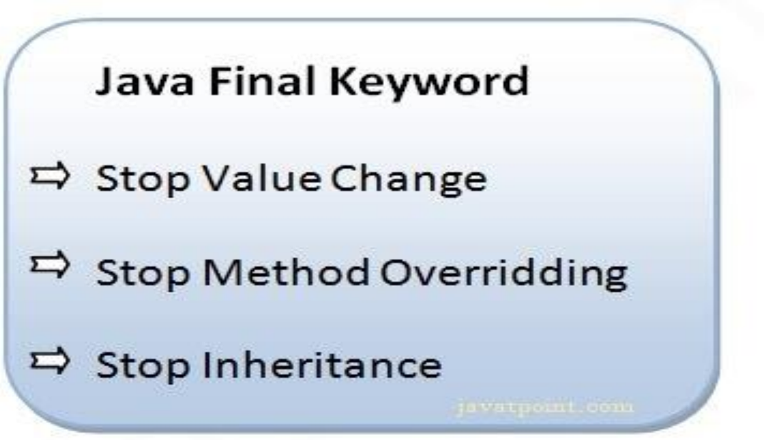
```
abstract class Bank
{
abstract int getRateOfInterest( );
}
class SBI extends Bank
{
int getRateOfInterest ()
{
return 7;
}
}
class PNB extends Bank
{
int getRateOfInterest()
{
return 8;
}
}

class TestBank
{
public static void main(String args[ ])
{
Bank b;
b=new SBI( );
System.out.println("Rate of Interest is: "+b.getRateOfInterest( ));
b=new PNB( );
System.out.println("Rate of Interest is: "+b.getRateOfInterest( ));
}
}
```

<u>Explain in detail about Final variables, Final Methods and Final Classes.</u>

- ✓ The **final keyword** in java is used to restrict the user.
- ✓ The java final keyword can be used in many contexts.

Final can be:

1. variable
2. method
3. class



Java Final Keyword

⇨ Stop Value Change

⇨ Stop Method Overridding

⇨ Stop Inheritance

javatpoint.com

<u>1) Java final variable</u>

- ✓ If you make any variable as final, you cannot change the value of final variable
- ✓ The final variables will be constant.

<u>Example of final variable:</u> There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike
{
final int speedlimit=90;//final variable
 void run()
{
speedlimit=400;
 }
public static void main(String args[])
{
Bike obj=new  Bike( );
obj.run();
}
}
```

2) Java final method

- ✓ If you make any method as final, you cannot override it.
- ✓ Final Methods are not used in derived classes.

Example of final method

```
class Bike
{
 final void run()
{
System.out.println("running");
}
}

class Honda extends Bike
{
  void run()
{
System.out.println("running safely with 100kmph");
}
 public static void main(String args[])
{
Honda h= new Honda();
h.run();
   }
}
```

Here in above example we are using final method run( ) in derived class. This is violating the rule of final methods. That is we cannot use same function in derived class. That means, we cannot override the methods.

3) Java final class

- ✓ If you make any class as final, you cannot extend it. That is, we cannot use final class as base class for other classes.
- ✓ This means, we cannot inherit properties from final class

Example of final class

```
final class Bike
{
Void run( )
{
System.out.println("running safely with 90kmph");
}
}

class Honda1  extends  Bike
{
void  run()
{
System.out.println("running safely with 100kmph");
}
 public static void main(String args[])
{
 Honda1 honda= new Honda1();
honda.run();
 }
}
```

JVM:

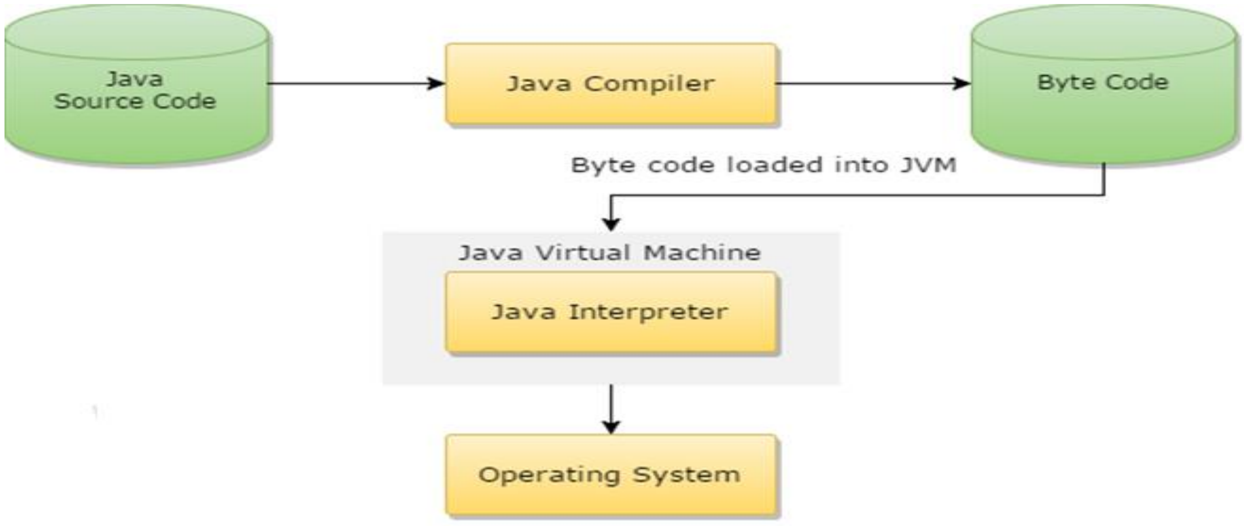


Diagram of JVM

JVM is a engine that provides runtime environment to drive the Java Code or applications. It converts Java bytecode into machines language. JVM is a part of JRE(Java Run Environment). It stands for Java Virtual Machine

4. First, Java code is complied into bytecode. This bytecode gets interpreted on different machines
5. Between host system and Java source, Bytecode is an intermediary language.
6. JVM is responsible for allocating memory space.

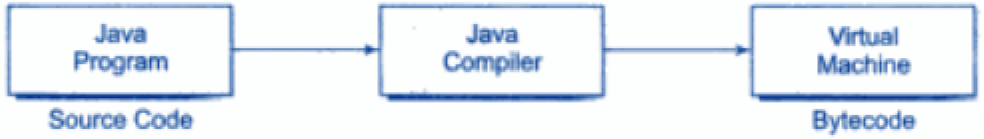


Fig. 3.6 *Process of compilation*

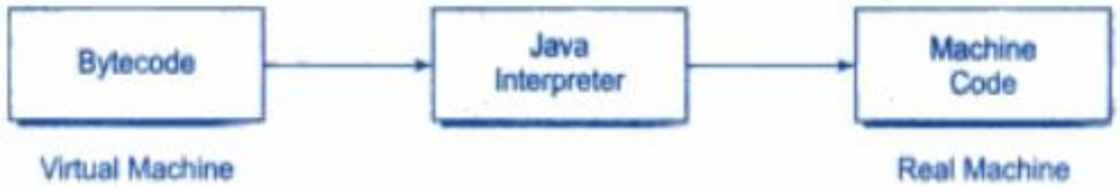


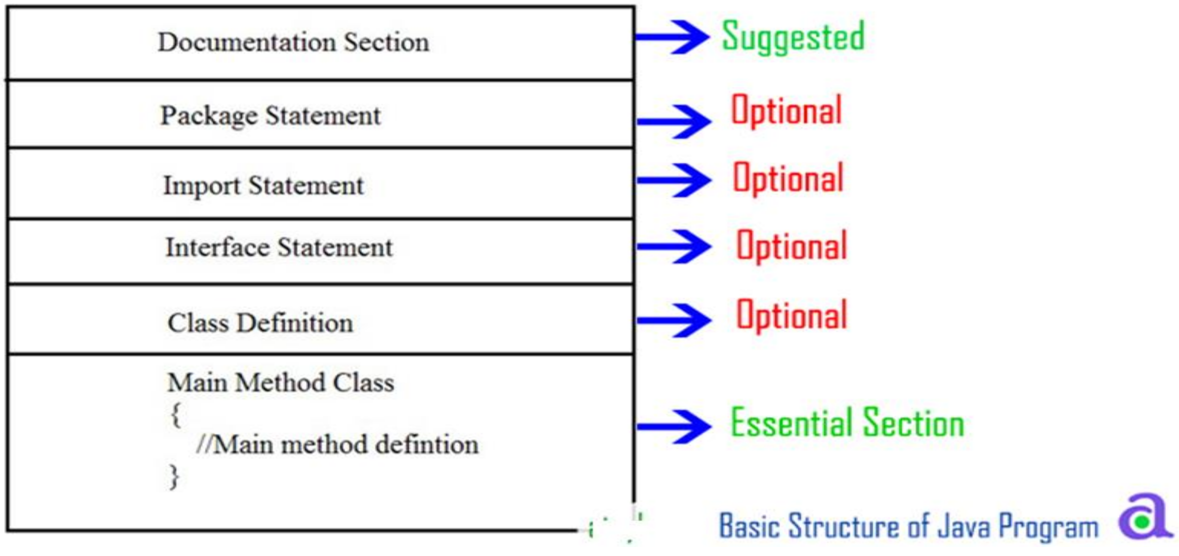
Fig. 3.7 *Process of converting bytecode into machine code*

Generating machine code is a two step process:

Step 1: The process of compiling a java program into bytecode which is also reffered to as virtual machine code.

Step 2: The virtual machine code is not machine specific. The machine specific code is generated by the java interpreter. The process of converting bytecode into machine code is performed by java Interpreter

Java Program Structure:



Basic Structure of Java Program

B.SC III Year
Programming in JAVA
UNIT I

Section	Description
Documentation Section	You can write a comment in this section. Comments are beneficial for the programmer because they help them understand the code. These are optional. Three types of comment line representation: • Single line (//) • Multi line (/* */) • Documentation Comments (/** */)
Import statements	This line indicates that if you want to use a class of another package, then you can do this by importing it directly into your program. Example: import java.io.*;
Interface statement	Interfaces are like a class that includes a group of method declarations. It's an optional section and can be used when programmers want to implement multiple inheritances within a program.
Class Definition	A Java program may contain several class definitions. Classes are the main and essential elements of any Java program.
Main Method Class	Every Java stand-alone program requires the main method as the starting point of the program. This is an essential part of a Java program. There may be many classes in a Java program, and only one class defines the main method. Methods contain data type declaration and executable statements

History of Java: The history of Java is very interesting. Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of java starts with Green Team. Java team members (also known as **Green Team**), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was suited for internet programming. Later, Java technology was incorporated by Netscape. The principles for creating Java programming were "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Architecture Neutral, Object-Oriented, Interpreted and Dynamic".

B.SC III Year
Programming in JAVA
UNIT I

Currently, Java is used in internet programming, mobile devices, games, e-business solutions, etc. There are given the significant points that describe the history of Java.
1) James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.
2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.
3) Firstly, it was called "Greentalk" by James Gosling, and file extension was .gt.
4) After that, it was called Oak and was developed as a part of the Green project.

Why Java named "Oak"?

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania, etc.
6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

Why Java Programming named "Java"?

7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA", etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
According to James Gosling, "Java was one of the top choices along with **Silk**". Since Java was so unique, most of the team members preferred Java than other names.
8) Java is an island of Indonesia where first coffee was produced (called java coffee).
9) Notice that Java is just a name, not an acronym.
10) Initially developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.
11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
12) JDK 1.0 released in(January 23, 1996).

UNIT-II

In java, programmers can create several classes &Interface. After creating these classes and interface, it is better if they are divided into some groups depending on their relationship. Thus, the classes and interface which handle similar or same task are put into the same directory or folder, which is also known as package.

Packages act as “containers” for classes. A package represents a directory that contain related group of classes & interface.

TYPES OF PACKAGES

There are basically only 2 types of java packages. They are as follow :

- System Packages or Java API/ Built -in Packages
- User Defined Packages.

SYSTEM PACKAGES OR JAVA API

As there are built in methods , java also provides inbuilt packages which contain lots of classes &interfaces. These classes inside the packages are already defined & we can use them by importing relevant package in our program. Java has an extensive library of packages, a programmer need not think about logic for doing any task.

JAVA SYSTEM PACKAGES & THEIR CLASSES

- **java.lang**

Language Support classes. These are classes that java compiler itself uses & therefore they are automatically imported. They include classes for primitive types, strings, maths function, threads

&exception.

- **java .util**

Language Utility classes such as vector, hash tables ,random numbers, date etc.

- **java.io**

Input /Output support classes. They provide facilities for the input & output of data

- **java.awt**

Set of classes for implementing graphical user interface. They include classes for windows, buttons, list, menus & so on.

- **java.net**

Classes for networking. They include classes for communicating with local computers as well as with internet servers.

- **java.applet**

Classes for creating & implementing applets.

<u>USER DEFINED PACKAGES :</u>

The users of the Java language can also create their own packages. They are called user-defined packages. User defined packages can also be imported into other classes & used exactly in the same way as the Built in packages.

Creating User Defined Packages

Syntax :

```
package packageName;

public class className

{

- - - - - - - - - - - - -

// Body of className

- - - - - - - - - - - -

}
```

We must first declare the name of the package using the package keyword followed by the package name. This must be the first statement in a Java source file. Then define a classes as normally as define a class.

Example :

package **myPackage**;

public class **class1**

{

- - - - - - - - - - - - -

// Body of class1

}

In the above example, **myPackage** is the name of the package. The class **class1** is now considered as a part of this package. This listing would be saved as a file called **class1.java** & located in a directory named **mypackag**e.

STEPS FOR CREATING PACKAGE : To create a user defined package the following steps

should be involved :-

1: Declare the package at the beginning of a file using the syntax :

package packageName;

2: Define the class that is to be put in the package & declare it public.

Java also supports the concept of package hierarchy. This is done by specifying multiple names in a package statement, seprated by dots (.).

Ex :- package firstPackage.secondPackage;

ACCESSING A PACKAGE

Java package can be accessed either using a fully qualifiedclass name or using a shortcut approach through the import statement.

Syntax :

import package1[.package2][.package3].classname;

Here, **package1** is the name of the top level package, **package2** is the name of the package that is inside the package & so on. We can have any number of packages in a package hierarchy. Finally the explicit classname is specified. The import statement must end with a **semicolon (;).** The import statement should appear before any class definitions in a source file. Multiple import statements are allowed.

Ex :

```
import firstpackage.secondPackage.Myclass;
```

or

```
import firstpackage.*;
```

Simple example of java package

The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile java package If you are not using any IDE, you need to follow the **syntax** given below:

```
javac -d directory javafilename
```

For **example**

```
javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

B.SC III Year
Programming in JAVA
UNIT I

How to run java package program

You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac –d . Simple.java
To Run: java mypack.Simple

Interfaces in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

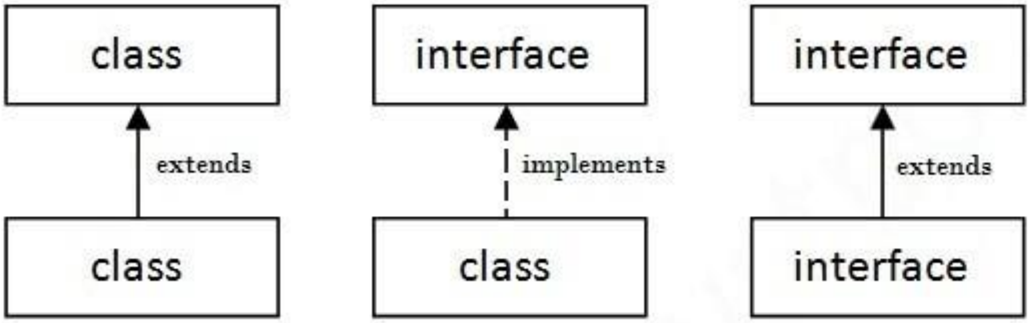
The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

How to declare an interface?An interface is declared by using the **interface** keyword. It provides total abstraction; means all the methods in an interface are declared with the **empty body**, and all the fields are **public, static and final** by default. A class that implements an interface must implement all the methods declared in the interface.

The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**



Extending Interfaces

When one interface inherits from another interface, that sub-interface inherits all the methods and constants that its super interface declared. In addition, it can also declare new abstract methods and constants. To extend an interface, you use the **extends keyword** just as you do in the class definition, an interface can directly extend multiple interfaces.

This can be done using the **following syntax:** interface **InterfaceName** extends interfacel[, interface2, , interfaceN]

Example Program on Extending Interface:

```
import java.io.*;

interface student

{

public void store(int a,String b);

public void display();

}

interface marks extends student

{

public void read(int a,int b,int c);
public void compute();
}
class Demo5 implements marks
{
int rno;
String name;
int m1,m2,m3;
public void store(int a,String b)
{
rno=a;
name=b;
}
public void display()
{
System.out.println("The Student Roll Number is "+rno);
System.out.println("The Student Name  is "+name);
}
public void read(int x, int y, int z)
{
m1=x;
m2=y;
m3=z;
}
```

```
public void compute()
{
int tot=m1+m2+m3;
float avg=(tot)/3;
System.out.println("The total is "+tot);
System.out.println("The average is "+avg);
}
public static void main(String args[])
{
Demo5 d=new Demo5();
d.store(01,"Aman");
d.display();
d.read(40,50,65);
d.compute();
}
}
```

Implementing Interfaces

To declare a class that implements an interface, you include an implements clause in the class declaration. Your class can implement more than one interface, so the implements keyword is followed by a comma-separated list of the interfaces implemented by the class.

A class that implements an interface must implement all the methods declared in the interface. The methods must have the exact same signature (name + parameters) as declared in the interface

All variables in an interface are public, even if you leave out the public keyword in the variable declaration.

B.SC III Year
Programming in JAVA
UNIT I

Threads in Java

To achieve multiple tasks parallel, Programmer uses threads. Multithreading gives Java the ability to achieve multiple tasks in parallel. One task does not wait for another to complete. That is, without completing one task, another task can start and also can execute.

MULTITHREADING REALTIME EXAMPLES

- Background jobs like running application servers like Oracle application server, Web servers like Tomcat etc which will come into action whenever a request comes.
- Typing MS Word document while listening to music.
- Railway ticket reservation system where multiple customers accessing the server.

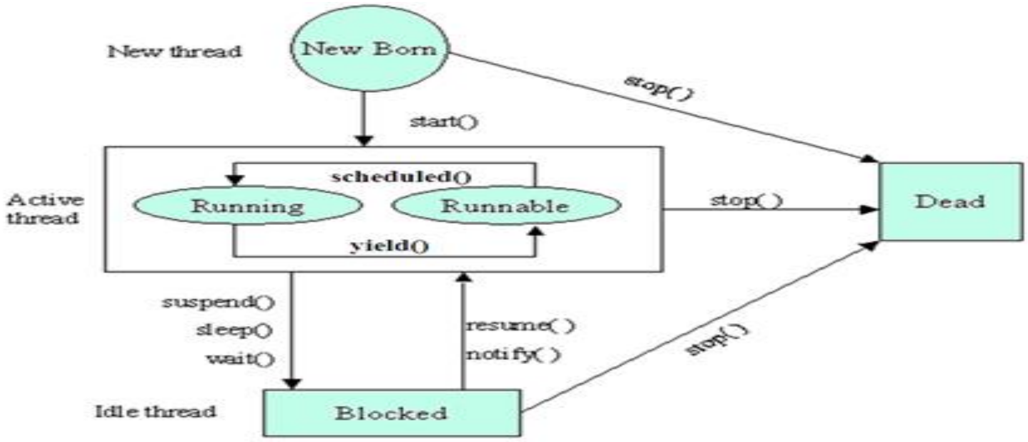
Multithreading in Java

- Multithreading in java is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation, etc

  Every thread in Java is created and controlled by the **java.lang.Thread** class.

A thread can be in one of the following states,

1. New born state(New)
2. Ready to run state (Runnable)
3. Running state(Running)
4. Blocked state
5. Dead state



<u>New Born State:</u>--

- The thread enters the new born state as soon as it is created. The thread is created using the new operator.
- From the new born state the thread can go to ready to runnable mode or dead state.

<u>Ready to run mode (Runnable Mode)</u>:--

- If the thread is ready for execution but waiting for the CPU the thread is said to be in ready to running mode.
- All the events that are waiting for the processor are queued up in the ready to run mode and are served in FIFO manner or priority scheduling.

<u>Running State</u>:--

- If the thread is in execution then it is said to be in running state.
- The thread can finish its work and end normally.
- The thread can also be forced to give up the control when one of the following conditions arise
- A thread can be suspended by suspend( ) method. A suspended thread can be revived by using the resume() method.
- A thread can be made to sleep for a particular time by using the sleep(milliseconds) method.

<u>Blocked State</u>:--

- ✓ A thread is said to be in blocked state if it prevented from entering into the runnable state and so the running state.
- ✓ The thread enters the blocked state when it is suspended, made to sleep or wait.

<u>Dead State</u>:--

- ✓ The running thread ends its life when it has completed executing the run( ) method which is called natural dead.
- ✓ The thread can also be killed at any stage by using the stop( ) method.

Extending Thread Class

```
class Multi extends Thread {       // Create a class by extending Thread class

public void run(){

System.out.println("thread is running...");    // Defining run( ) in above class

}

public static void main(String args[]){

Multi t1=new Multi();                        // Creating an object for class

t1.start();                             // Call the start( ) by using object

 }

}
```

Explain various Thread Methods:

- ✓ We have various methods which can be called on Thread class object.
- ✓ These methods are very useful when writing a multithreaded application.
- ✓ Thread class has following important methods.

Method Signature	**Description**
String getName()	Retrieves the name of running thread.
void setName(String name)	It is used to set name for thread
void start()	This method will start a new thread of execution by calling run() method of Thread/runnable object.

void run()	This method is the entry point of the thread. Execution of thread starts from this method.
void sleep (intsleeptime)	This method stops the thread for mentioned time duration in argument (sleeptime in ms)
void yield()	By invoking this method the current thread pause its execution temporarily and allow other threads to execute.
void join()	This method used to queue up a thread in execution. Once called on thread, current thread will wait till calling thread completes its execution
booleanisAlive()	This method will check if thread is alive or dead
intgetPriority()	To retrieve priority of Thread
void setPriority(priorityConstant)	It is used to set the priority value. There are 3 priorities. MIN_PRIORITY MAX_PRIORITY NORM_PRORITY
currentThread()	It is used to know currently executed Thread status.

<u>Explain about Priority of a Thread (Thread Priority):</u>

✓ Each thread has a priority.

✓ Priorities are represented by a number between 1 and 10.

✓ In most cases, thread scheduler schedules the threads according to their priority.

The Thread class defines 3priority constants.

- ✓ **MIN_PRIORITY=1**
- ✓ **NORM_PRIORITY=5**
- ✓ **MAX_PRIORITY=10**

✓ Default priority of a thread is **5 (NORM_PRIORITY).**

✓ The value of MIN_PRIORITY is 1 and the value **of MAX_PRIORITY is 10.**

<u>Program: Write a JAVA Program on Thread Priorities.</u>

```
    import java.io.*;

        classTestPriority extends Thread

        {

        public void run()

        {

    System.out.println("running thread name is:"+Thread.currentThread().getName());
System.out.println("running thread priorityis:"+Thread.currentThread().getPriority());

        }

        }

        class Demo2

        {

            public static void main(String args[])

            {

            TestPriority m1=new TestPriority();
        TestPriority m2=new TestPriority();

        m1.setPriority(Thread.MIN_PRIORITY);

        m2.setPriority(Thread.MAX_PRIORITY);

        m1.start();

        m2.start();

        }

}
```

Write a JAVA Program on Implementing Thread Using Runnable Interface:

```
Class  Multi3 implements Runnable{

public void run(){

System.out.println("thread is running...");

}


public static void main(String args[]){

Multi3 m1=new Multi3();

Thread t1 =new Thread(m1);

t1.start();

 }

}
```

<u>Important Note</u>:

If you are **not extending the Thread class**, your class object would not be treated as a thread object. So you need to explicitly create **Thread class object**. We are passing the object of your class that implements Runnable so that your class run() method may execute.

UNIT-III

What is a web application?

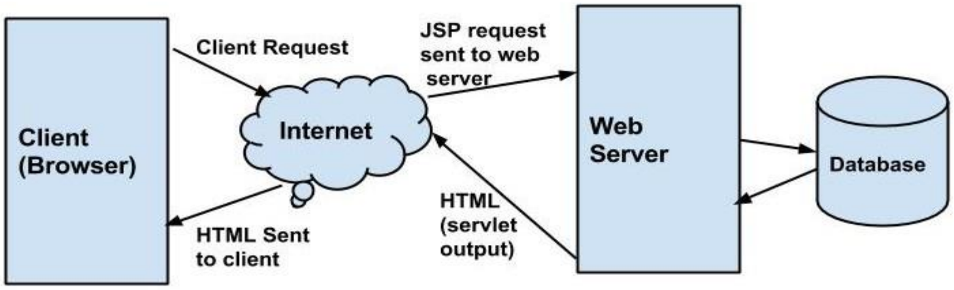
A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript.

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.
A server(generally referred to as application or web server) supports the Java Server Pages. This server will act as a mediator between the client browser and a database. The following diagram shows the **JSP architecture**.

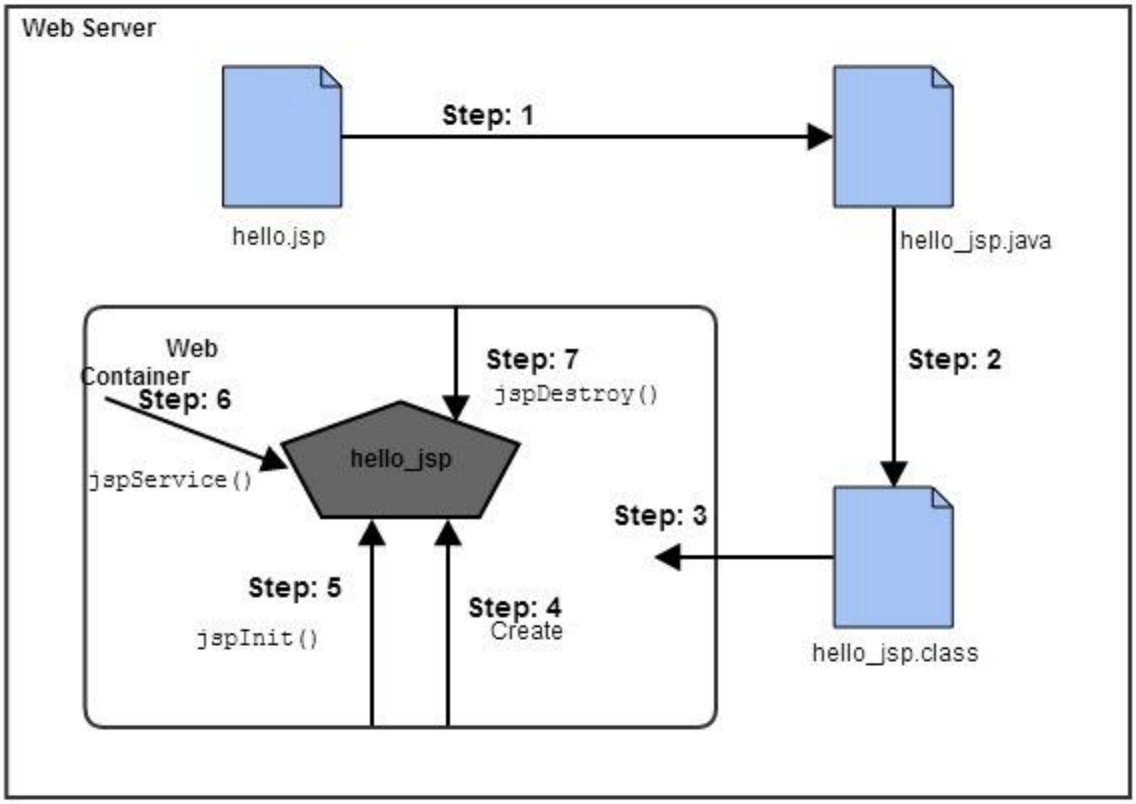
JSP Architecture Flow

1. The user goes to a JSP page and makes the request via internet in user’s web browser.
2. The JSP request is sent to the Web Server.
3. Web server accepts the requested .jsp file and passes the JSP file to the JSP Servlet Engine.
4. If the JSP file has been called the first time then the JSP file is parsed otherwise servlet is instantiated. The next step is to generate a servlet from the JSP file. The generated servlet output is sent via the Internet form web server to users web browser.
5. Now in last step, HTML results are displayed on the users web browser.



Explain in detail about Lifecycle of JSP
A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code.
The Following are the **JSP Lifecycle steps**:

1. Converting JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class into memory.
4. Creating servlet instance.
5. Initialization by calling jspInit() method
6. Request Processing by calling _jspService() method
7. Destroying by calling jspDestroy() method



Web Container translates JSP code into a **servlet class source(.java) file in step 1**,

then in **step 2** , compiles that into a java servlet class.

In the **step 3**, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.

The initialized servlet can now service request. For each request the **Web Container** call the **_jspService()** method. When the Container removes the servlet instance from service, it calls the **jspDestroy()** method to perform any required clean up.

JSP Scripting Elements

In JSP there are three types of scripting elements:

- **JSP Expressions**: It is a small java code which you can include into a JSP page. The syntax is "<%= some java code %>"
- **JSP Scriptlet**: The syntax for a scriptlet is "<% some java code %>". You can add 1 to many lines of Java code in here.
- **JSP Declaration**: The syntax for declaration is "<%! Variable or method declaration %>", in here you can declare a variable or a method for use later in the code.

JSP Expressions

Using the JSP Expression you can compute a small expression, always a single line, and get the result included in the HTML which is returned to the browser. Using the code we have previously written, let's explore expressions.

Eg Code:

```
The time on the server is <%= new java.util.Date() %>
```

Output:

```
The time on the server is Thursday January 21 07:21:43 GMT 2016.
```

Explanation

Here the "new java.util.Date()" is processed into the actual date and time shown through HTML on the browser. Let's explore expressions through a couple of more examples.

Examples

- In the first example we are going to see an expression for converting a string from lower case to upper case. Here is the code:

  The Expression: Converting a string to uppercase <%= new String("Hello World").toUpperCase() %>

  Here we are creating a "String" object with "Hello World" set as the value for object. Following that we are calling a Java function ".toUpperCase" to convert the string from lower case to upper case.

  The HTML: Converting a string to uppercase: HELLO WORLD

JSP Scriptlets

This JSP Scripting Element allows you to put in a lot of Java code in your HTML code. This Java code is processed top to bottom when the page is the processed by the web server. Here the result of the code isn't directly combined with the HTML rather you have to use "out.println()" to show what you want to mix with HTML. The syntax is pretty much the same only you don't have to put in an equal sign after the opening % sign.
Let's take a look at the code:

Code:

```
<h2> Hello World</h2>

<%

for(inti=0; i<= 5; i++)

{

out.println("<br/> I really love counting: " + i);

}

%>
```

Output:

```
I really love counting: 1
I really love counting: 2
I really love counting: 3
I really love counting: 4
```

Explanation:

In this example we have set up a basic h2 heading and following that we have a "for loop" in the scriptlet. Just to remember println means print line. In every iteration of the loop we print the "I really love counting" and appends it with the integer value of the "i" printed through the HTML. Just try to make sure that you don't put in a lot of code in a scriptlet in JSP. This will make it readable and easy to manage.

JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations −

```
<%! declaration; [ declaration; ]+ ... %>
```

You can write the XML equivalent of the above syntax as follows −

```
<jsp:declaration>
   code fragment
</jsp:declaration>
```

Following is an example for JSP Declarations −

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.

The entire JSP page process is controlled by this directive tags.JSP
Directives has been categorized into three types as follows.
1) Page directive
2) include directive
3) taglib directive

Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

import

The import attribute is used to import class,interface or all the members of a package.

It is similar to import keyword in java class or interface.

Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

The include directive is used to include the contents of any resource it may be jsp file, html file or text file.

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage

Advantage of Include directive

Code Reusability

Syntax of include directive

```
<%@ include file="resourceName" %>
```

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
```

```
</html>
```

JSP Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax JSP Taglib directive

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

JSP Actions

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.
The action tags are used to control the flow between pages and to use Java Bean.

jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Example of jsp:forward action tag

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

<jsp:include> Action:

- ✓ The include action is used to insert the files into the current page.
- ✓ The syntax of the include action:

```
<jsp: include page = " URL" />
```

Here page is an attribute is used to specify the address of the included page in the current page.

Example:

```
<jsp: include page="header.jsp" />
```

The <jsp:text> Action:

The **text** action can be used to write the template text in JSP pages and documents.

The **syntax** of the include action:

```
<jsp:text>Template data</jsp:text>
```

- ✓ **Example:**

```
<jsp:text>Wecome to JSP Applications</jsp:text>
```

<jsp:setProperty> Action

- This setProperty action tag is used to set the property of a Bean(class). While using this action tag, you may need to specify the Bean's(class) unique name.
- The **syntax** of the setProperty action:

```
            <jsp: useBean  name=”bean name” class=”classname”>

<jsp:setPropertyname="bean name"property="propertyname"value=”valueofproperty”/>
```

- Example:

```
<jsp:useBean name=”test”  class=”Geometry”>
<jsp:setProperty name = "test"  property = "message" value = "Hello JSP..." />
```

<jsp:getProperty> Action

- It is used to retrieve or fetch the value of Bean's (class) property.
- The **syntax** of the getProperty action:

```
<jsp: useBean  name=”bean name” class=”classname”>

<jsp:getPropertyname="bean name"property="property_name"/>
```

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.

Jsp Implicit Objects

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within Scriptlet without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Objects and their corresponding classes:

out	javax.servlet.jsp.JspWriter
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse

session	javax.servlet.http.HttpSession
application	javax.servlet.ServletContext
exception	javax.servlet.jsp.JspException
page	java.lang.Object
pageContext	javax.servlet.jsp.PageContext
config	javax.servlet.ServletConfig

The output which needs to be sent to the client (browser) is passed through this object. In simple words out implicit object is used to write content to the client.

Methods of OUT Implicit Object

void print()
void println()
void newLine()
void clear()
void clearBuffer()
void flush()
boolean isAutoFlush()
int getBufferSize()
int getRemaining()

1)void print(): This method writes the value which has been passed to it. below

Example:

```
        out.print("WELCOME");
```

void println(): This method is similar to the print() method, the only difference between print and println is that the println() method adds a new line character at the end.

Example:

```
out.print("hi");

 out.print("hello");
```

output :

```
hi hello
```

println()

Example:

```
 out.println(“hi”);
  out.println(“hello”);
```

output:

```
hi
hello
```

3) void newLine(): This method adds a new line to the output. Example –

Example:

```
out.print(“This will write content without a new line”);
    out.newLine();
    out.print(“I’m just an another print statement”);
```

Output:

```
This will write content without a new line
I’m just an another print statement
```

4)void clear( ): It clears the output buffer without even letting it write the buffer content to the client.

Example:

```
    out.clear();
```

5)void clearBuffer(): This method is similar to the clear() method. The only difference between them is that when we invoke out.clear() on an already flushed buffer it throws an exception, however out.clearBuffer() doesn’t.

6)boolean isAutoFlush() : It returns a Boolean value true/false. It is used to check whether the buffer is automatically flushed or not.

7)int getBufferSize(): This method returns the size of output buffer in bytes.

Example:

index.jsp

```
        <body>
        <%
        out.print( "print statement " );
        out.println( "println" );
        out.print("Another print statement");
        %>
      </body>
```

Request: The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purposes.

1) **getParameter (String name) :**
   - ✓ This method is used to get the value of a request's parameter.
   - ✓ Example at login page user enters user-id and password and once the credentials are verified the login page gets redirected to user information page, then using request.getParameter we can get the value of user-id and password which user has input at the login page.

String Uid= request.getParameter("user-id");

String Pass= request.getParameter("password");

2) **getCookies( ) :**
   It returns an array of cookie objects received from the client. This method is mainly used when dealing with cookies in JSP.
   request.getCookies( );
3) **getRequestURI( ) :**
   This method (request.getRequestURI()) returns the URL of current JSP page.
   request.getRequestURI( );
4) **getMethod() :**
   It returns HTTP request method. request.getMethod(). For example it will return GET for a Get request and POST for a Post Request.
   request.getMethod( );

Example of JSP request implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Response Object:

It is basically used for modifying or delaying with the response which is being sent to the client (browser) after processing the request.

Methods of request Implicit Object

1)void setContentType(String type) – This method tells browser, the type of response data by setting up the MIME type

Example –

```
response.setContentType("text/html");

response.setContentType("image/gif");

response.setContentType("image/png");

response.setContentType("application/pdf");
```

2**) void sendRedirect(String address) –** It redirects the control to a new JSP page. For e.g. When the browser would detect the below statement, it would be redirected to the josephscollege.ac.in from the current JSP page.

```
response.sendRedirect("http://josephscollege.ac.in");
```

3) **void addCookie(Cookie cookie)** –

This method adds a cookie to the response. The below statements would add 2 Cookies Author and Siteinfo to the response.

```
response.addCookie(Cookie Author);
response.addCookie(Cookie Siteinfo);
```

4) void sendError(int status_code, String message) –

It is used to send error response with a code and an error message. For example –

```
response.sendError(404, "Page not found error");
```

5)void setStatus(int statuscode) – This method is used to set the HTTP status to a given value. For e.g. the below statement would set HTTP response code to 404 (Page not found).

```
response.setStatus(404);
```

Example of response implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

4)Exception Object:

- ✓ Exception implicit object is used in exception handling for displaying the error messages.
- ✓ This object is only available to the JSP pages, which has isErrorPage set to true.

Example:

```
<%@ page isErrorPage="true" %>
<html>
<body>
Sorry following exception occured:<%= exception %>
</body>
</html>
```

5) page implicit object:

- ✓ In JSP, page is an implicit object of type Object class.
- ✓ This object is assigned to the reference of auto generated servlet class. It is written as:

```
Object page=this;
```

For example:

<% this.log("message"); %>

UNIT IV
JAVA Database Connectivity

1) Introduction to JDBC

Java Database Connectivity(JDBC) is an **Application Programming Interface(API)** used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

JDBC Driver

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or **JDBC-ODBC bridge**
- **Type-2 Driver** or **Native API Partly Java Driver**
- **Type-3 Driver** or **Network Protocol Driver**
- **Type-4 Driver** or **Thin Driver**

1) Essential JDBC Classes

JDBC API is available in two packages java.sql, core API and javax.sql JDBC optional packages. Following are the important classes and interfaces of JDBC.

Class/interface	Description
DriverManager	This class manages the JDBC drivers. You need to register your drivers to this.

Class/interface	Description
	It provides methods such as registerDriver() and getConnection().
Driver	This interface is the Base interface for every driver class i.e. If you want to create a JDBC Driver of your own you need to implement this interface. If you load a Driver class (implementation of this interface), it will create an instance of itself and register with the driver manager.
Statement	This interface represents a static SQL statement. Using the Statement object and its methods, you can execute an SQL statement and get the results of it. It provides methods such as execute(), executeBatch(), executeUpdate() etc. To execute the statements.
PreparedStatement	This represents a precompiled SQL statement. An SQL statement is compiled and stored in a prepared statement and you can later execute this multiple times. You can get an object of this interface using the method of the Connection interface named prepareStatement(). This provides methods such as executeQuery(), executeUpdate(), and execute() to execute the prepared statements and getXXX(), setXXX() (where XXX is the datatypes such as long int float etc..) methods to set and get the values of the bind variables of the prepared statement.
CallableStatement	Using an object of this interface you can execute the stored procedures. This returns single or multiple results. It will accept input parameters too. You can create a CallableStatement using the prepareCall() method of the Connection interface. Just like Prepared statement, this will also provide setXXX() and getXXX() methods to pass the input parameters and to get the output parameters of the procedures.

Class/interface	Description
Connection	This interface represents the connection with a specific database. SQL statements are executed in the context of a connection. This interface provides methods such as close(), commit(), rollback(), createStatement(), prepareCall(), prepareStatement(), setAutoCommit() setSavepoint() etc.
ResultSet	This interface represents the database result set, a table which is generated by executing statements. This interface provides getter and update methods to retrieve and update its contents respectively.
ResultSetMetaData	This interface is used to get the information about the result set such as, number of columns, name of the column, data type of the column, schema of the result set, table name, etc It provides methods such as getColumnCount(), getColumnName(), getColumnType(), getTableName(), getSchemaName() etc.

3) Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

```
import java.sql.*;
class ConnectProg{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();

//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e)
{
 System.out.println(e);}

}
}
```

The next() method of the ResultSet interface moves the pointer of the current (ResultSet) object to the next row, from the current position.

i.e., on calling the next() method for the first time the result set pointer/cursor will be moved to the 1st row (from default position).

And on calling the next() method for the second time the result set cursor will be moved to the 2nd row.

4)**Inserting into database**

```
import java.sql.*;
class JdbcInsert1 {
 public static void main (String[] args)
 {
       try {
           String url = "jdbc:msql://200.210.220.1:1114/Demo";
           Connection conn = DriverManager.getConnection(url,"","");
           Statement st = conn.createStatement();
           st.executeUpdate("INSERT INTO Customers " +
               "VALUES (1001, 'Simpson', 'Mr.', 'Springfield', 2001)");
```

```
          st.executeUpdate ("INSERT INTO Customers " +
             "VALUES (1002, 'McBeal', 'Ms.', 'Boston', 2004)");
          st.executeUpdate("INSERT INTO Customers " +
             "VALUES (1003, 'Flinstone', 'Mr.', 'Bedrock', 2003)");
          st.executeUpdate("INSERT INTO Customers " +
             "VALUES (1004, 'Cramden', 'Mr.', 'New York', 2001)");
          conn.close();
       }
 catch (Exception e) {
          System.err.println("Got an exception! ");
          System.err.println(e.getMessage());
      }
  }
}
```

5) Explain the procedure to retrieve data from database.

Step 1: Register with JDBC Driver

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

Step 2: Get the connection from database

```
Connection conn=DriverManager.getConnection (url, username, password);
Here, username=”system”; password=”admin”;
url=”jdbc:oracle:thin:@localhost:1521:XE”;
```

Step 3: Create Statement object

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

Step 4:

Execute the Query by using executeQuery() and store the result in ResultSet object.

```
String sql=”select * from employee”;
ResultSet rs=Stmt.executeQuery (sql);
```

Use the next() to retrieve data from table.

```
while(rs.next())
{
int eid=rs.getInt(eid);
System.out.println(eid);
String name=rs.getString(name);
System.out.println(name);
}
```

Step 5: Close the Connection

```
            conn.close ();
```

Program:

```
import java.sql.*;
class RetrievingData
{
public static void main(String args[])
{
try
{
String username=”system”;
String password=”admin”;
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
DriverManager.registerDriver (new oracle.jdbc.driver.oracleDriver ());
Connection conn=DriverManager.getConnection (url, username, password);
Statement stmt=conn.createStatement ();
String sql=”select * from employee”;
ResultSet rs=Stmt.executeQuery (sql);
while(rs.next())
{
int eid=rs.getInt(eid);
System.out.println(eid);
String name=rs.getString(name);
System.out.println(name);
}
conn.close ();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

Note:

Here,

next() is used to move cursor or resultset object (rs) to next row in a table.

getInt() is used to retrieve integer data from database table.

getString() is used to retrieve String data from database table.

6) Explain the procedure to Store an image in database

To store image into the database first we have to create table with two columns that is name and photo in database as follows:

```
Create table student (Name varchar (20), Photo BLOB);
```

BLOB=A BLOB is binary large object that can hold a variable amount of data with a maximum length of 65535 characters.

These are used to store large amounts of binary data, such as images or other types of files.

The various steps involved in JDBC to store image in database

Step 1: Register with JDBC Driver

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

Step 2: Get the connection from database

```
Connection conn=DriverManager.getConnection (url, username, password);
Here, username=”system”; password=”admin”;
url=”jdbc:oracle:thin:@localhost:1521:XE”;
```

Step 3: Create PreparedStatement object by passing sql query

```
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
```

Why use PreparedStatement?

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Step 4: Now we have to store data of two columnsas follows:

a) Storing first column value using setString()

```
ps.setString(1,”Sailaja”);
```

b) Storing second column value as follows:

```
FileInputStream fin=new FileInputStream("d:\\Sailaja.jpg");
ps.setBinaryStream(2,fin,fin.available());
ps.executeUpdate();
```

Step 5: Close the Connection

```
conn.close ();
```

Program:

```
import java.sql.*;
class StoringImage
{
public static void main(String args[])
{
try
{
String username=”system”;
String password=”admin”;
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
DriverManager.registerDriver(new oracle.jdbc.driver.oracleDriver ());
Connection conn=DriverManager.getConnection (url, username, password);
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
ps.setString(1,”Sailaja”);
FileInputStream fin=new FileInputStream("d:\\Sailaja.jpg");
ps.setBinaryStream(2,fin,fin.available());
ps.executeUpdate();
conn.close ();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

7) Explain the procedure to store a file in database

To store file into the database first we have to create table with two columns that is name and profile in database as follows:

```
Create table student (Name varchar (20), Profile CLOB);
```

The various steps involved in JDBC to store file in database

Step 1: Register with JDBC Driver

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

Step 2: Get the connection from database

```
Connection conn=DriverManager.getConnection (url, username, password);
```

```
Here, username=”system”; password=”admin”;
url=”jdbc:oracle:thin:@localhost:1521:XE”;
```

Step 3: Create PreparedStatement object by passing sql query

```
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
```

Step 4: Now we have to store data of two columns as follows:

a) Storing first column value using setString()

```
ps.setString(1,”Sailaja”);
```

b) Storing second column value as follows:

```
File f=new File("d:\\myfile.txt");
FileReader fr=new FileReader(f);
ps.setCharacterStream(2,fr);
ps.executeUpdate();
```

Step 5: Close the Connection

```
conn.close ();
```

Program:

```
import java.sql.*;
class StoringFile
{
public static void main(String args[])
{
try
{
String username=”system”;
String password=”admin”;
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
DriverManager.registerDriver (new oracle.jdbc.driver.oracleDriver ());
Connection conn=DriverManager.getConnection (url, username, password);
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
ps.setString(1,”Sailaja”);
File f=new File("d:\\myfile.txt");
FileReader fr=new FileReader(f);
ps.setCharacterStream(2,fr);
ps.executeUpdate();
conn.close ();
}
catch(Exception e)
{
```

```
System.out.println(e);
}
}
}
```

8) Explain the procedure to retrieve an image from database

The various steps involved in JDBC to retrieve image from database

Step 1: Register with JDBC Driver

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

Step 2: Get the connection from database

```
        Connection conn=DriverManager.getConnection (url, username, password);
        Here, username=”system”; password=”admin”;
        url=”jdbc:oracle:thin:@localhost:1521:XE”;
```

Step 3: Create PreparedStatement object by passing sql query

```
        String sql=”select * from student”;
        PreparedStatement ps=conn.prepareStatement (sql);
```

Step 4: create ResultSet Object using executeQuery() and use next() to retrieve data from database table.

```
ResultSet rs=ps.executeQuery();
if(rs.next())                         //now on 1st row
{
Blob b=rs.getBlob(2); //2 means 2nd column data
byte i=b.getBytes();

FileOutputStream fout=new FileOutputStream();
fout.write(i);
}
```

Step 5: Close the Connection

```
        conn.close ();
```

<u>Program:</u>

```
import java.sql.*;
class StoringImage
{
public static void main(String args[])
{
try
{
String username=”system”;
String password=”admin”;
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
DriverManager.registerDriver (new oracle.jdbc.driver.oracleDriver ());
Connection conn=DriverManager.getConnection (url, username, password);
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
ResultSet rs=ps.executeQuery();
if(rs.next())                          //now on 1st row
{
Blob b=rs.getBlob(2); //2 means 2nd column data
byte i=b.getBytes();

FileOutputStream fout=new FileOutputStream();
fout.write(i);
}
conn.close ();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

9) Explain the procedure to retrieve a file from database

The various steps involved in JDBC to retrieve a file from database

Step 1: Register with JDBC Driver

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

Step 2: Get the connection from database

```
Connection conn=DriverManager.getConnection (url, username, password);
```

Here, username=”system”; password=”admin”;
url=”jdbc:oracle:thin:@localhost:1521:XE”;

Step 3: Create PreparedStatement object by passing sql query

```
String sql=”select * from student”;
PreparedStatement ps=conn.prepareStatement (sql);
```

Step 4: create ResultSet Object using executeQuery() and use next() to retrieve data from database table.

```
ResultSet rs=ps.executeQuery();
if(rs.next())                    //now on 1st row
{
Clob b=rs.getClob(2); //2 means 2nd column data
Reader i=b.getCharacterStream();

FileWriter f=new FileWriter();
f..write(i);
}
```

Step 5: Close the Connection

```
        conn.close ();
```

Program:

```
import java.sql.*;
class StoringImage
{
public static void main(String args[])
{
try
{
String username=”system”;
String password=”admin”;
String url=”jdbc:oracle:thin:@localhost:1521:XE”;
DriverManager.registerDriver (new oracle.jdbc.driver.oracleDriver ());
Connection conn=DriverManager.getConnection (url, username, password);
String sql=”insert into student values(?,?)”;
PreparedStatement ps=conn.prepareStatement (sql);
ResultSet rs=ps.executeQuery();
if(rs.next())                    //now on 1st row
{
Clob b=rs.getClob(2); //2 means 2nd column data
Reader i=b.getCharacterStream();

FileWriter f=new FileWriter();
f..write(i);
}
conn.close ();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
```

-----END----