1. Write a Java program using JDBC to extract name of those students who study in class 12 and they are boy, assuming that student table has four attributes (ID, name, class and gender).

Solution :

```
import java.sql.*;

class MysqlCon{

public static void main(String args[])

{

try{

Class.forName("com.mysql.jdbc.Driver");

Connection con=DriverManager.getConnection(“jdbc:mysql://localhost:3306/oxford", "username", "password");

Statement stmt=con.createStatement();

String query = “select * from student where class = 12 and gender = ‘boy’”;

ResultSet rs = stmt.executeQuery(query);

System.out.println(“ID          Name   Class              Gender”);

while( rs.next() )

System.out.println(rs.getInt(1) + "  " +rs.getString(2) + "  " + rs.getString(3) +"  " + rs.getString(3));

con.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

}
```

2. Write a simple Java program that reads data from one file and writes the data to another file sequentially.

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class CopyFile {

    public static void main(String[] args) {
        try
        {
            boolean create=true;
            Scanner KB=new Scanner(System.in);
            System.out.print("Enter Source File Name:");
            String sfilename=KB.next();
            File srcfile=new File(sfilename);
            if(!srcfile.exists())
            {
                System.out.println("File Not Found..");
            }
            else
            {
                FileInputStream FI=new FileInputStream(sfilename);
                System.out.print("Enter Target File Name:");
                String tfilename=KB.next();
                File tfile=new File(tfilename);
                if(tfile.exists())
                {
                    System.out.print("File Already Exist OverWrite it..Yes/No?:");
                    String confirm=KB.next();
                    if(confirm.equalsIgnoreCase("yes"))
                    {
                        create=true;
```

```
                                        }
                                        else
                                        {
                                                create=false;
                                        }
                                }
                                if(create)
                                {
                                        FileOutputStream FO=new FileOutputStream(tfilename);
                                        int b;
                                        //read content and write in another file
                                        while((b=FI.read())!=-1)
                                        {
                                                FO.write(b);
                                        }
                                        System.out.println("\nFile Copied...");
                                }
                                FI.close();
                        }
                }
                catch(IOException e)
                {
                        System.out.println(e);
                }
        }
}
```

3. Write an object oriented program in Java to find factorial of given number

```
public class Factorial {
   public static void main(String[] args) {
      int num = 10;
      long factorial = 1;
      for(int i = 1; i <= num; ++i)
      {

         factorial *= i;       // factorial = factorial * i;
      }
      System.out.printf("Factorial of %d = %d", num, factorial);
```

```
    }
}
```

4. What is layout management? Discuss any three layout management classes with examples of each.

   The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

   java.awt.BorderLayout
   java.awt.FlowLayout
   java.awt.GridLayout
   java.awt.CardLayout
   java.awt.GridBagLayout
   javax.swing.BoxLayout
   javax.swing.GroupLayout
   javax.swing.ScrollPaneLayout
   javax.swing.SpringLayout etc.

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Example :

```
import java.awt.*;
import javax.swing.*;

public class Border {
JFrame f;
Border(){
```

```
    f=new JFrame();

    JButton b1=new JButton("NORTH");;
    JButton b2=new JButton("SOUTH");;
    JButton b3=new JButton("EAST");;
    JButton b4=new JButton("WEST");;
    JButton b5=new JButton("CENTER");;

    f.add(b1,BorderLayout.NORTH);
    f.add(b2,BorderLayout.SOUTH);
    f.add(b3,BorderLayout.EAST);
    f.add(b4,BorderLayout.WEST);
    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);
    f.setVisible(true);
  }
  public static void main(String[] args) {
    new Border();
  }
  }
```

Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Example :

```
import java.awt.*;
import javax.swing.*;

public class MyGridLayout{
JFrame f;
MyGridLayout(){
f=new JFrame();

JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");
JButton b6=new JButton("6");
JButton b7=new JButton("7");
JButton b8=new JButton("8");
```

```
JButton b9=new JButton("9");

f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
f.add(b6);f.add(b7);f.add(b8);f.add(b9);

f.setLayout(new GridLayout(3,3));
//setting grid layout of 3 rows and 3 columns

f.setSize(300,300);
f.setVisible(true);
}
public static void main(String[] args) {
new MyGridLayout();
}
}
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Example :

```
import java.awt.*;
import javax.swing.*;

public class MyFlowLayout{
JFrame f;
MyFlowLayout(){
f=new JFrame();

JButton b1=new JButton("1");
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");

f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
```

```
f.setLayout(new FlowLayout(FlowLayout.RIGHT));
//setting flow layout of right alignment

f.setSize(300,300);
f.setVisible(true);
}
public static void main(String[] args) {
new MyFlowLayout();
}
}
```

6. What is Ajax? How does Ajax work?

   Solution :

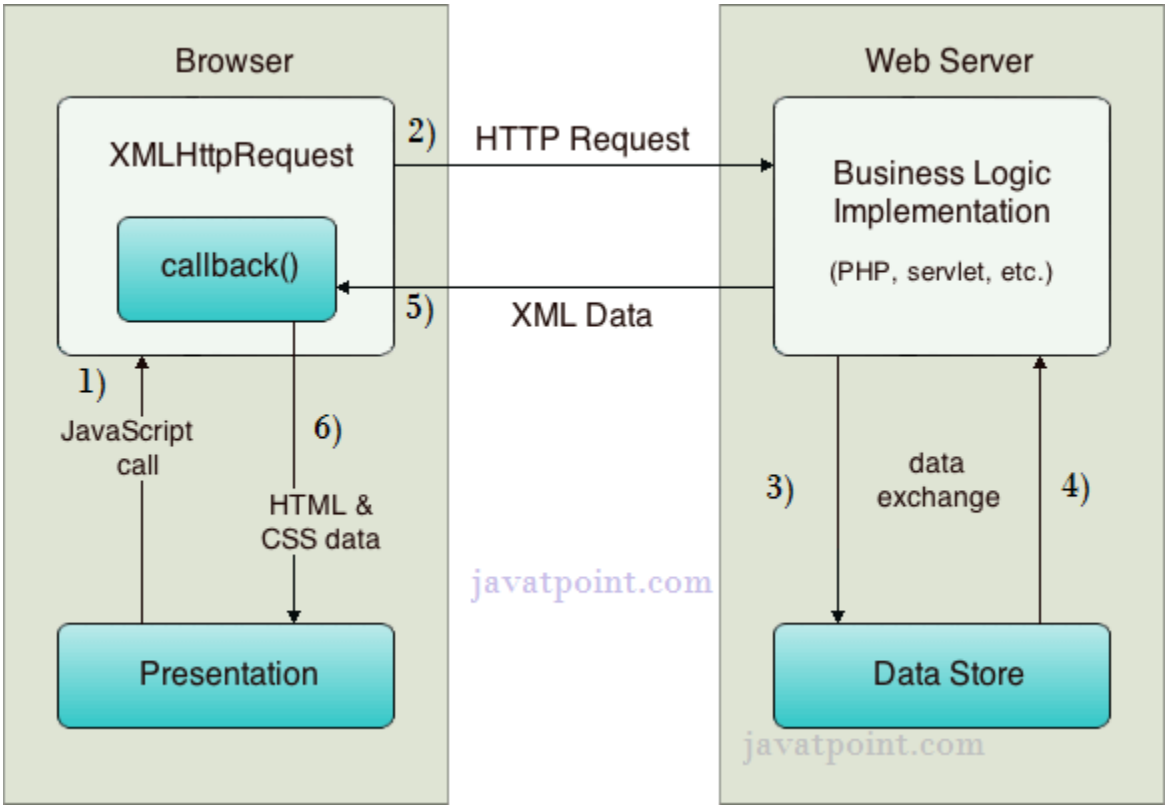
AJAX is an acronym for Asynchronous JavaScript and XML. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest etc.

AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.

AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

How AJAX works?

AJAX communicates with the server using XMLHttpRequest object. Let's try to understand the flow of ajax or how ajax works by the image displayed below.



As you can see in the above example, XMLHttpRequest object plays a important role.

1. User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
2. HTTP Request is sent to the server by XMLHttpRequest object.
3. Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
4. Data is retrieved.
5. Server sends XML data or JSON data to the XMLHttpRequest callback function.
6. HTML and CSS data is displayed on the browser.

7. What is socket? Write a program to read from a socket.

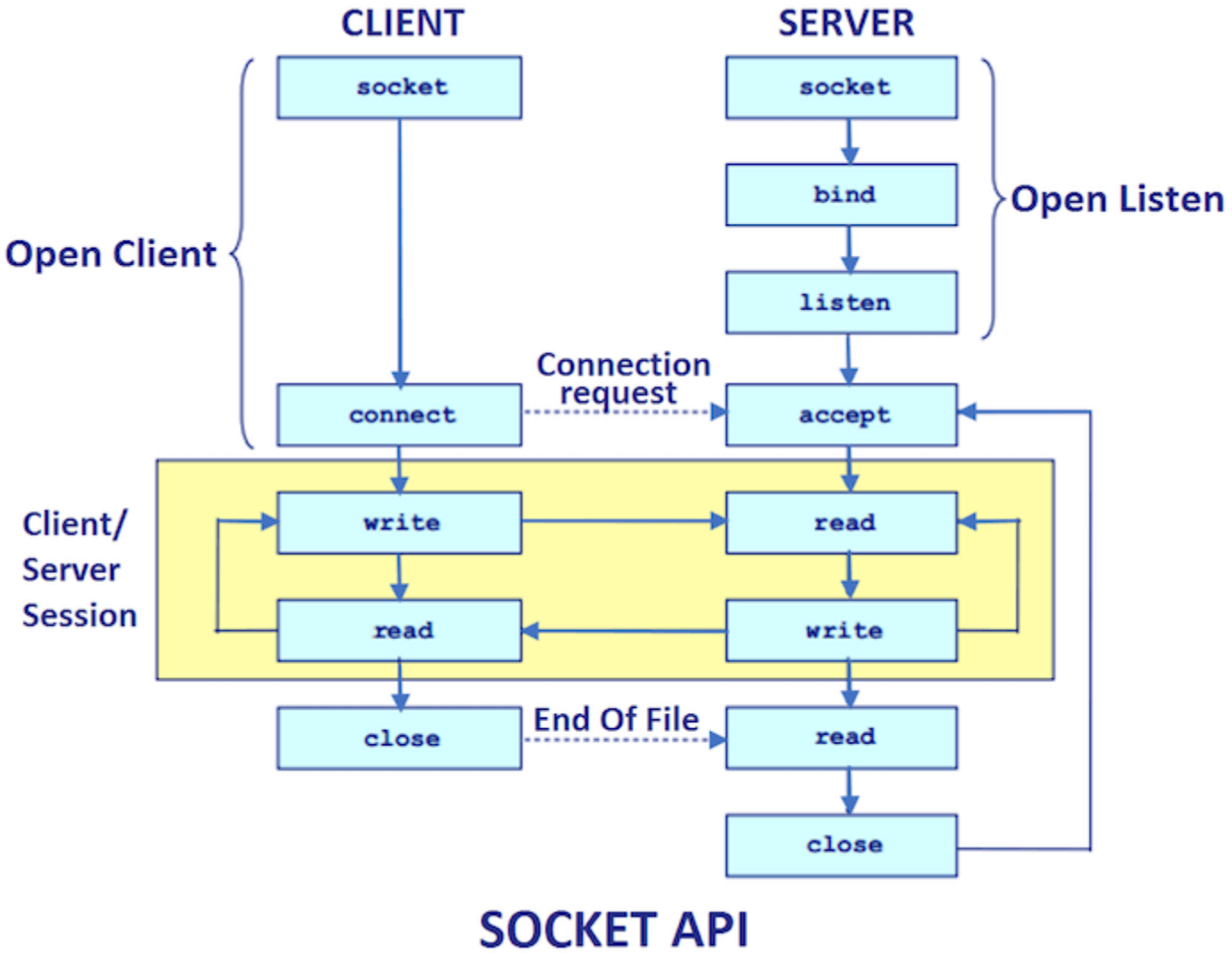
Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE.Java Socket programming can be connection-oriented or connection-less.Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



Example

```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
```

```
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

8. What is applet? Discuss its life cycle

An applet is any small application that performs one specific task that runs within the scope of a dedicated widget engine or a larger program, often as a plug-in. The term is frequently used to refer to a Java applet, a program written in the Java programming language that is designed to be placed on a web page.

Even though, the methods are called automatically by the browser, the programmer should know well when they are called and what he can do with the methods. Following is the schematic representation of the methods.

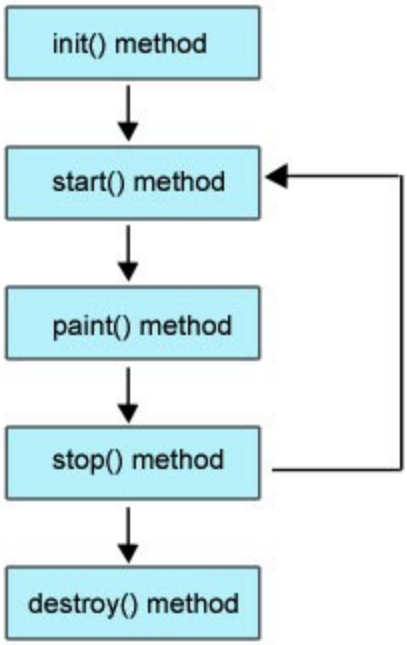


Figure: Life cycle of Applet

Brief Description of Life Cycle Methods

Following is the brief description of the above methods.

1. **init():** The applet's voyage starts here. In this method, the applet object is created by the browser. Because this method is called before all the other methods, programmer can utilize this method to instantiate objects, initialize variables, setting background and foreground colors in GUI etc.; the place of a constructor in an application. It is equivalent to **born state** of a thread.
2. **start():** In init() method, even through applet object is created, it is in **inactive** state. An inactive applet is not eligible for microprocessor time even though the microprocessor is idle. To make the applet active, the init() method calls start() method. In start() method, applet becomes active and thereby eligible for processor time.
3. **paint():** This method takes a **java.awt.Graphics** object as parameter. This class includes many methods of drawing necessary to draw on the applet window. This is the place where the programmer can write his code of what he expects from applet like animation etc. This is equivalent to **runnable state** of thread.
4. **stop():** In this method the applet becomes temporarily inactive. An applet can come any number of times into this method in its life cycle and can go back to the active state (paint() method) whenever would like. It is the best place to have cleanup code. It is equivalent to the **blocked state** of the thread.
5. **destroy():** This method is called just before an applet object is garbage collected. This is the end of the life cycle of applet. It is the best place to have cleanup code. It is equivalent to the **dead state** of the thread.

9. What is Event Handling? Discuss any three event classes in Java.

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener

AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

The Java ActionListener is notified whenever you click on the button or menu item. It is notified against ActionEvent. The ActionListener interface is found in java.awt.event package. It has only one method: actionPerformed().

actionPerformed() method

The actionPerformed() method is invoked automatically whenever you click on the registered component.

1. **public abstract void** actionPerformed(ActionEvent e);

Java MouseListener Interface

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. **public abstract void** mouseClicked(MouseEvent e);
2. **public abstract void** mouseEntered(MouseEvent e);
3. **public abstract void** mouseExited(MouseEvent e);
4. **public abstract void** mousePressed(MouseEvent e);
5. **public abstract void** mouseReleased(MouseEvent e);

Java KeyListener Interface

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

The signature of 3 methods found in KeyListener interface are given below:

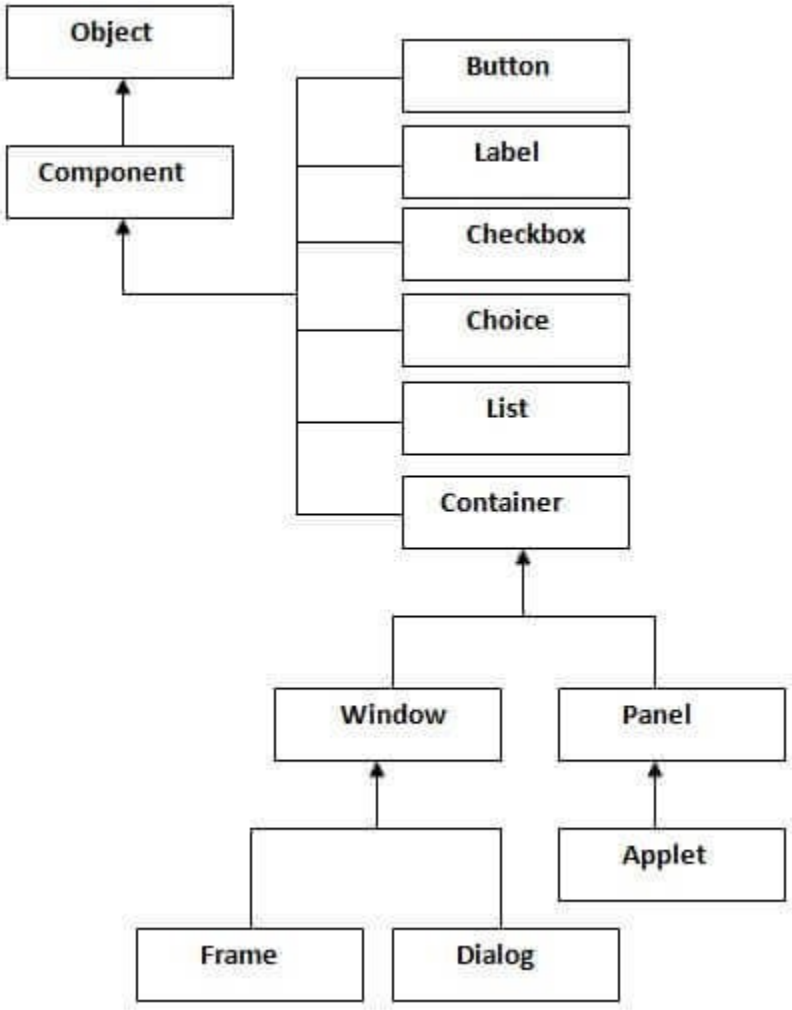
1. **public abstract void** keyPressed(KeyEvent e);
2. **public abstract void** keyReleased(KeyEvent e);
3. **public abstract void** keyTyped(KeyEvent e);

10. What is AWT? Explain different controls of AWT in detail.

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container
The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window
The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel
The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Button

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

Label

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

TextField

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

TextArea

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

Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

11. What is RMI? How is it different from CORBA.

RMI is an abbreviation of Remote Method Invocation. This technology was released with Java 1.1, actually available since JDK 1.02, and it lets Java developers invoke object methods and allows them to be executed on remote JVMs or Java Virtual Machines. Its implementation is rather easy particularly if you know Java very well. It's just like calling a process locally; however, its calls are limited to Java only.

Having mentioned about RMI's Java-centric characteristic, the only way to integrate codes in other languages into the RMI distribution system is to use an interface. This interface is called the Java native-code interface. However, it can be extremely complex and, more often than not, results to fragile codes.

RMI has major features that CORBA doesn't have,

1. most notably, the ability to send new objects ,code and data across a network,
2. for remote virtual machines to faultlessly handle new objects

Difference between RMI and CORBA

CORBA isn't tied to one language; Java RMI is tied to Java. The down side to this for CORBA, is that developers also have to learn to write interfaces in yet another language, IDL.

CORBA doesn't allow executable code to be sent to the remote system; RMI does.

CORBA allows remote method invocations as well as the exchange of primitive data types and structures. RMI allows full objects to be passed and returned.

RMI provides simple interface which is easy to use where as CORBA is harder to use.

RMI provides object oriented communication whereas CORBA supports Structured Communication.

RMI has dynamic codebase Whereas CORBA is static Codebase.

RMI performance is not as good depending on application whereas performance is better in CORBA.

12. What are the benefits of using swing components? Explain

Swing provides a richer set of components than AWT. They are 100% Java-based.

AWT on the other hand was developed with the mind set that if a component or capability of a component wasn't available on one platform, it won't be available on any platform. Something quickly portable from platform x, to y, to z. Due to the peer-based nature of AWT, what might work on one implementation might not work on another, as the peer-integration might not be as robust. Many of the original AWT problems were traceable to differences in peer implementations.

This is not to say that there are less bugs in Swing, though most are out these days. Its just that if a bug exists in Swing, its the same problem on all platforms, which was not the case with AWT.
There are a few other advantages to Swing over AWT:

- *Larger set of components such as tables, trees, sliders, spinners, progress bars,internal frames and text components.*
- *Components can be bound to keyboard events to react to keystrokes.*
- *Better support for multi-threading*
- *Swing provides both additional components and added functionality to AWT-replacement components*
- *Swing components can change their appearance based on the current "look and feel" library that's being used. You can use the same look and feel as the platform you're on, or use a different look and feel*
- *Swing components follow the Model-View-Controller paradigm (MVC), and thus can provide a much more flexible UI.*
- *Swing provides "extras" for components, such as:*
    - *Icons on many components*
    - *Decorative borders for components*
    - *Tooltips for components*
- *Swing components are lightweight (less resource intensive than AWT)*
- *Swing provides built-in double buffering*
- *Swing provides paint debugging support for when you build your own components*

13. What is JDBC? Discuss different driver types on JDBC

Check notes of web technologies 2

14. What is exception handling? Discuss exception handling in detail with suitable example.

Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions.

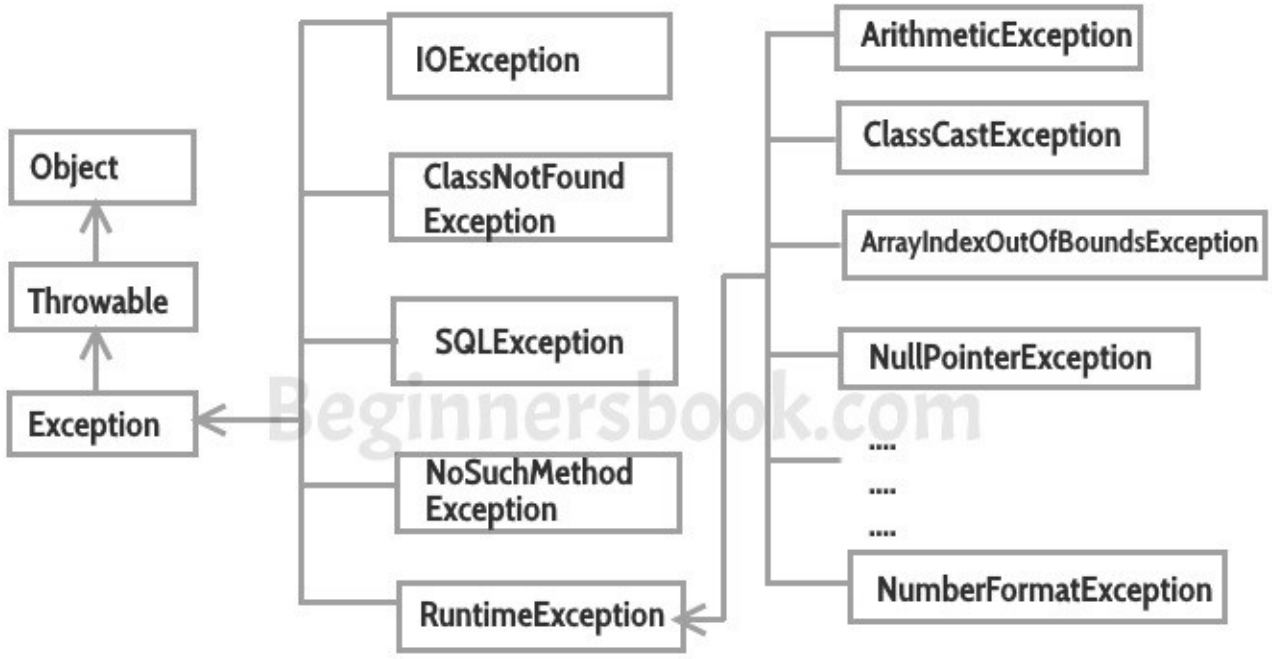
What is an exception?

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

Exceptions are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions. Few examples:
NullPointerException – When you try to use a reference that points to null.
ArithmeticException – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
ArrayIndexOutOfBoundsException – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.



Flow of control in try/catch/finally blocks:

1. If exception occurs in try block's body then control immediately transferred(**skipping rest of the statements in try block**) to the catch block. Once catch block finished execution then finally block and after that rest of the program.
2. If there is no exception occurred in the code which is present in try block then first, the try block gets executed completely and then control gets transferred to finally block (**skipping catch blocks**).
3. If a return statement is encountered either in try or catch block. In this case **finally block runs**. Control first jumps to finally and then it returned back to **return statement.**

```
class TestExceptions {
```

```
  static void myMethod(int testnum) throws Exception {
    System.out.println ("start - myMethod");
    if (testnum == 12)
      throw new Exception();
    System.out.println("end - myMethod");
    return;
  }
  public static void main(String  args[]) {
    int testnum = 12;
    try {
      System.out.println("try - first statement");
      myMethod(testnum);
      System.out.println("try - last statement");
    }
    catch ( Exception ex) {
      System.out.println("An Exception");
    }
    finally {
      System. out. println( "finally") ;
    }
    System.out.println("Out of try/catch/finally - statement");
  }
}
```

15. What is Java bean? How is it different from other Java classes

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Difference between java bean and java classes.

- Any Java class that adheres to certain conventions regarding property and event interface definitions can be a JavaBean. Beans are Java classes that can be manipulated in a visual builder

tool and composed into applications.

- Introspection, the process by which a builder tool analyzes how a Bean works, differentiates Beans from typical Java classes. Because Beans are coded with predefined patterns for their method signatures and class definitions, tools that recognize these patterns can "look inside" a Bean and determine its properties and behavior.

- Introspection allows a Bean's state to be manipulated at design time that is, at the time it is being assembled as a part within a larger application. For this to work, method signatures within Beans must follow a certain pattern so that introspection tools recognize how Beans can be manipulated, both at design time and at run time.

- In effect, Beans publish their attributes and behaviors through special method signature patterns that are recognized by beans-aware application construction tools. However, these construction tools are not required to build or test your beans. The pattern signatures are easily recognized by human readers, as well as builder tools. One of the first things you'll learn when building beans is how to recognize and construct methods that adhere to these patterns.

- Beans should not be used indiscriminately. Beans are best suited to software components intended to be visually manipulated within builder tools. Some functionality, such as the JDBC API, is still best provided through a programmatic or textual interface, rather than a visual manipulation interface.

- A JavaBean is basically a Java class which satisfies a few simple rules (it must have a public no-arg constructor, no public instance variables, properties have getters and setters; actually, you can get around the last rule by supplying a BeanInfo object for your bean). JavaBeans were originally conceived for graphical application builder tools, but that emphasis has shifted considerably and they're now used almost everywhere.
- A Bean can have associated support classes: a BeanInfo class providing information about the bean, its properties and its events, and also property editors and a graphical customizer.
- A Bean is a Java class, but a Java class does not have to be a bean. In other words a bean in a specific Java class and has rules that have to be followed before you have a bean.

16. What is Multithreading? Discuss java thread model.

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.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

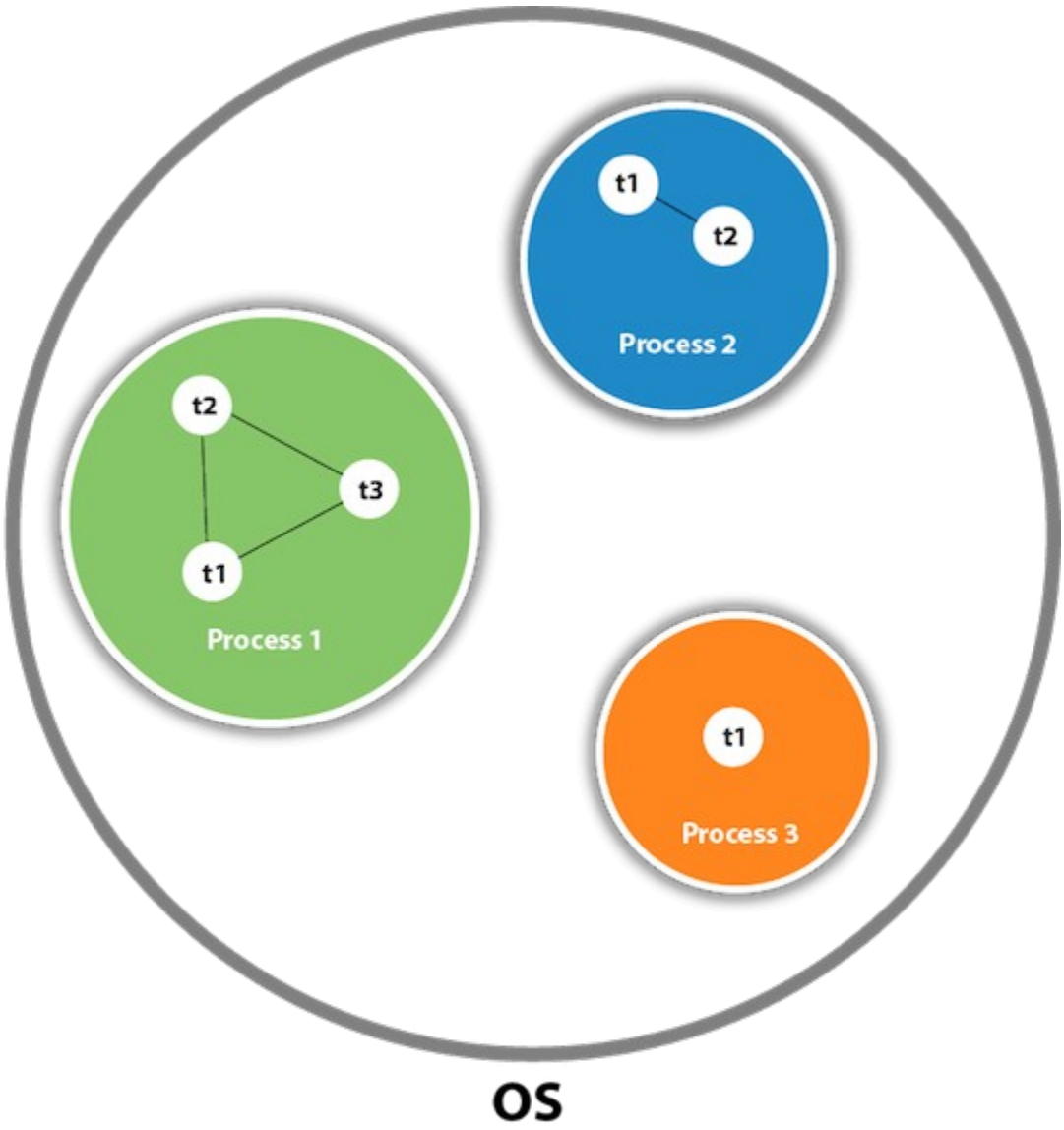
2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

What is Thread in java?

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

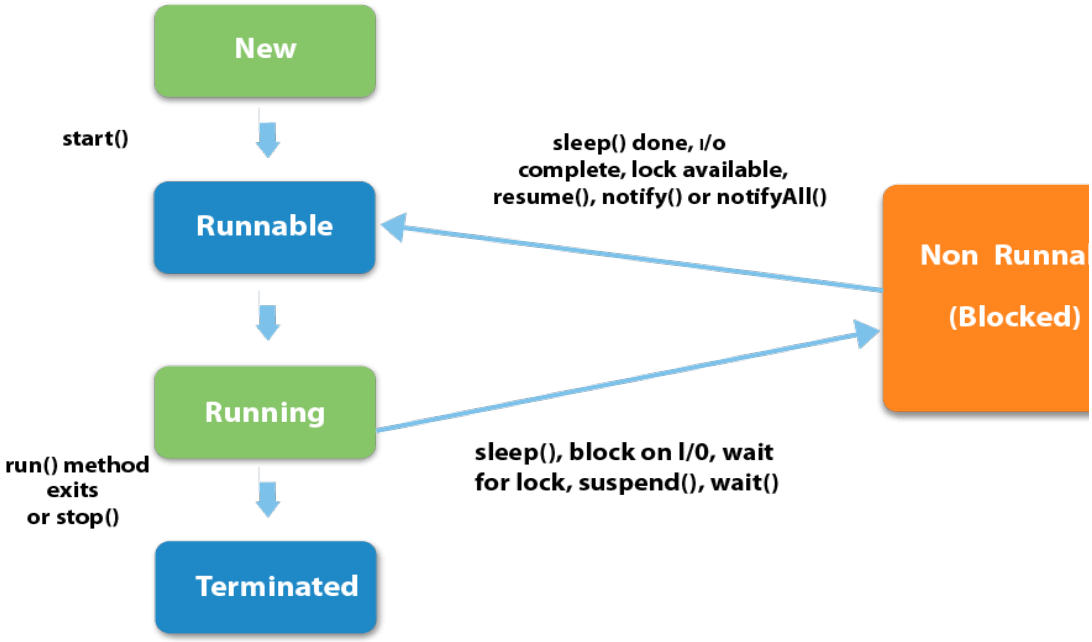
Life cycle of a Thread (Thread States)

*A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.*

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. *New*
2. *Runnable*
3. *Running*
4. *Non-Runnable (Blocked)*
5. *Terminated*



1) New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

17. Write an applet program to load and display images

```
import java.awt.*;
import java.applet.*;


public class DisplayImage extends Applet {

Image picture;

public void init() {
picture = getImage(getDocumentBase(),"sonoo.jpg");
}

public void paint(Graphics g) {
g.drawImage(picture, 30,30, this);
}

}
```

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

myapplet.html

```
<html>
<body>
```

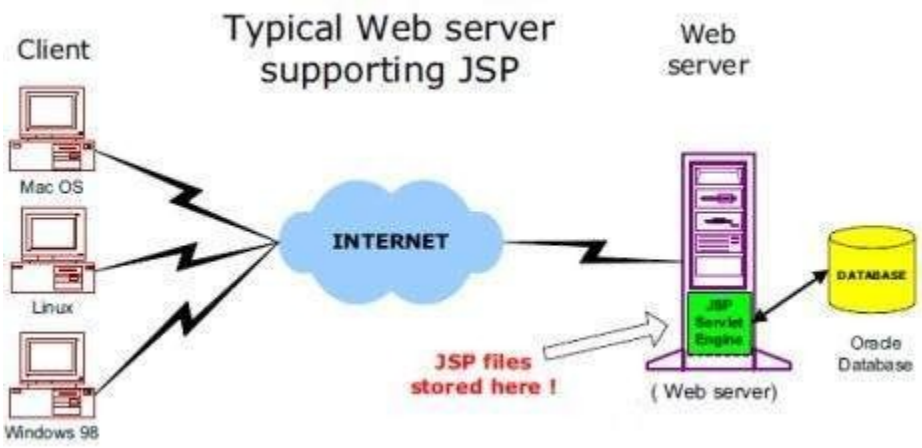
```
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

18. What is JSP? Discuss JSP structure in detail.

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.



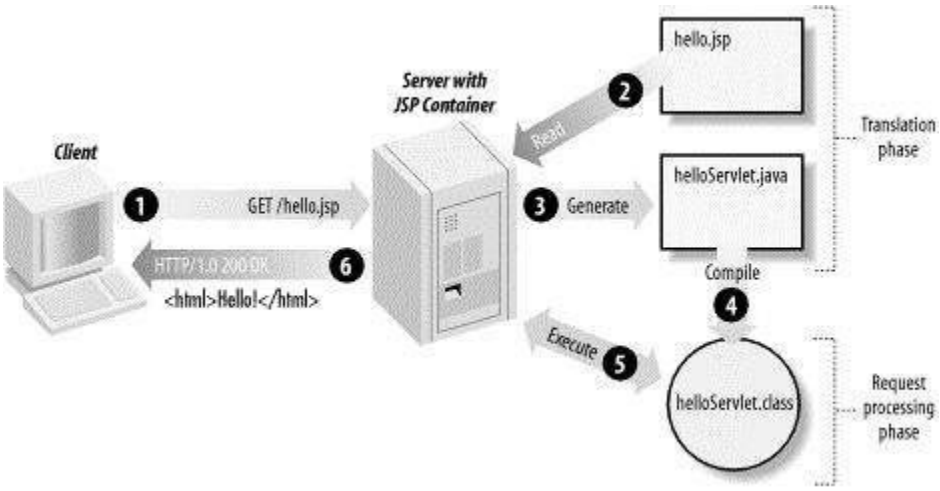
JSP Processing

The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.

- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

19. What is JDBC? Discuss JDBC Architecture in detail.

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

JDBC Architecture

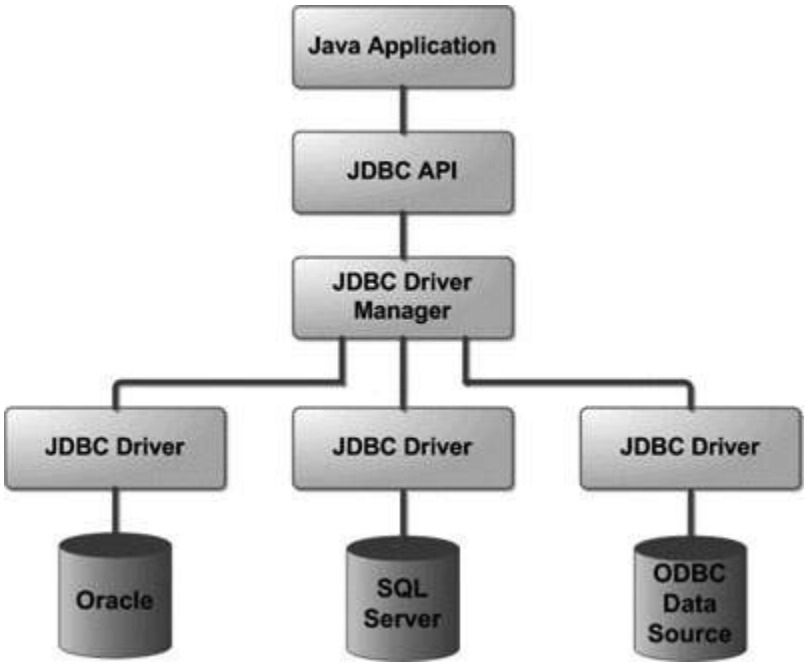
The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers −

- **JDBC API:** This provides the application-to-JDBC Manager connection.
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application −



Common JDBC Components

The JDBC API provides the following interfaces and classes −

- **DriverManager:** This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver:** This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement:** You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException:** This class handles any errors that occur in a database application.

20. Short notes on :

Synchronization:

Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section. Processes' access to critical section is controlled by using synchronization techniques. When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes. If proper synchronization techniques[1] are not applied, it may cause a race condition where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

Other than mutual exclusion, synchronization also deals with the following:

- deadlock, which occurs when many processes are waiting for a shared resource (critical section) which is being held by some other process. In this case, the processes just keep waiting and execute no further;
- starvation, which occurs when a process is waiting to enter the critical section, but other processes monopolize the critical section, and the first process is forced to wait indefinitely;
- priority inversion, which occurs when a high-priority process is in the critical section, and it is interrupted by a medium-priority process. This violation of priority rules can happen under certain circumstances and may lead to serious consequences in real-time systems;
- busy waiting, which occurs when a process frequently polls to determine if it has access to a critical section. This frequent polling robs processing time from other processes.

Web Architecture:

Web architecture is the conceptual structure of the World Wide Web. The WWW or internet is a constantly changing medium that enables communication between different users and the technical interaction (interoperability) between different systems and subsystems. The basis for this is different components and data formats, which are usually arranged in tiers and build on each other. Overall, they form the infrastructure of the internet, which is made possible by the three core components of data transmission protocols (TCP/IP, HTTP, HTTPS), representation formats (HTML, CSS, XML), and addressing standards (URI, URL).

Types of web architectures

Client-server model

Initially, the web consisted of a two-tiered architecture: clients and servers. Clients and servers shared the tasks and services that the system was supposed to perform. For example, the client may request a service from the server; the server answers the request by providing the service. Retrieving a website using a URL address that directs to a server to load the site in the client's browser is an example of the two-layer model, also known as the client-server model.

Three-tier model

Three-tier models include an application logic between the client and the server, which handles the data processing and allows a certain degree of interaction. For example, an application server can process data while a database server is dedicated solely to data storage. In this way, content can be dynamically loaded and saved. The script language JavaScriptis often responsible for the behavior of the client.

Service-oriented architectures (SOA)

Today the web is used for the networking of globally distributed IT structures. Each IT system can, in turn, consist of subsections whose individual components are linked to one another via a fixed structure or architecture. Think intranet and internal enterprise software. Modern IT and web applications are much more complex than the client-server model. Distributed web services, which are set up as service-oriented architectures (SOA), offer many functions and modular functional units, which can be supplemented. With SOAs, business processes can be automated by the involved systems communicating with one another - partly without human intervention - and performing certain tasks. Examples include online banking, e-commerce, e-learning, online marketplaces, and business intelligence applications. These architectures are not only much more complex but can also be modularly extended. They are known as N-tier architectures and have so far been used primarily in the business sector.

RMI Architecture.

RMI stands for **Remote Method Invocation**. It is a mechanism that allows an object residing in one system (JVM) to access/invoke an object running on another JVM.

RMI is used to build distributed applications; it provides remote communication between Java programs. It is provided in the package **java.rmi**.

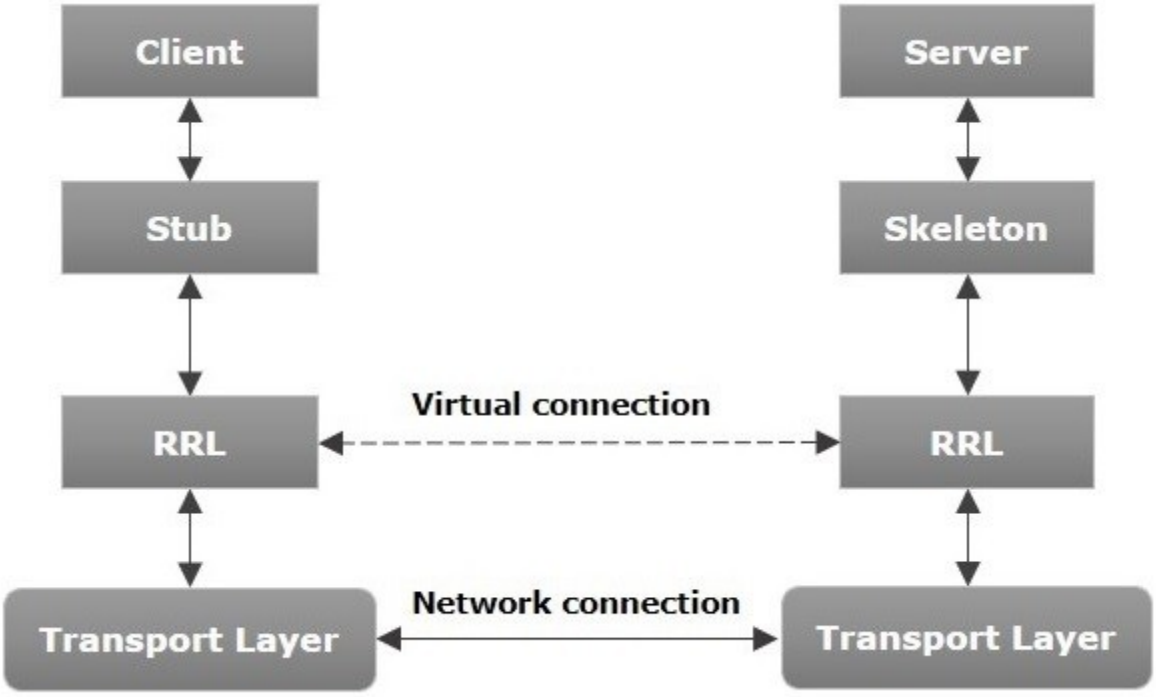
Architecture of an RMI Application

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).

- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.



Let us now discuss the components of this architecture.

- **Transport Layer** − This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **Stub** − A stub is a representation (proxy) of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- **Skeleton** − This is the object which resides on the server side. **stub** communicates with this skeleton to pass request to the remote object.
- **RRL(Remote Reference Layer)** − It is the layer which manages the references made by the client to the remote object.