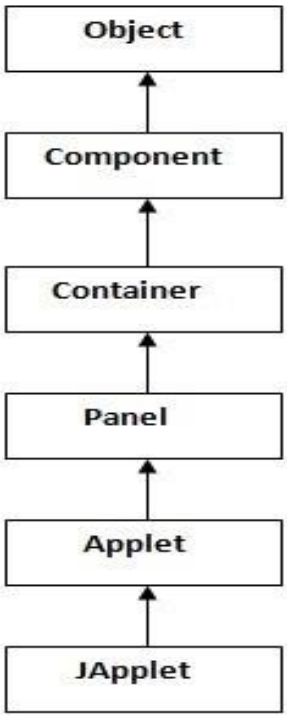
OOPS Through Java
Chapter – 5

What is an Applet? Write its advantages and drawbacks?

An applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at the client side. An applet can be a fully functional Java application because it has the entire Java Application Interface at its disposal. Note that Applet class extends Panel. Panel class extends Container which is the subclass of Component.

Hierarchy of Applet: -



Advantage of Applet: -

1. It works at client side so less response time.
2. It is more Secured
3. It can be executed by browsers running under many platforms, including Linux, Windows, Mac OS etc.

Drawback of Applet: -

Plugin is required at client browser to execute applet.

Main Points: -

1) An applet is a Java class that extends the java.applet.Applet.
2) The main() method is not invoked on an applet, and an applet class will not define main().
3) Applets are designed to be embedded within an HTML page.
4) A separate JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
5) The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
6) An applet works at client side so it requires less response time and more secured.
7) An applet can be executed by browsers that were running under many platforms including Unix, Linux, Ubuntu, Windows, Mac Os etc.

What is AppletViewer?

AppletViewer is a standalone command-line program from Sun Microsystems to run Java applets. AppletViewer is generally used by developers for testing their applets before deploying them to a website. AppletViewer is a preferred option for running Java applets that do not involve the use of a web browser.

The APPLET Class: -

The java.applet package is the smallest package in java. it contains the Applet class only. An applet is automatically loaded and executed when you open a web page that contain the applet. The Applet class has over 20 methods that are used to display text, images, shapes, play audio files, etc. Each method responds when you interact with it.

Syntax to embed an applet class in a java program: -

<APPLET code="name of the class file that extends java.applet.Applet" Height= number Width = number>

How to run an Applet?

There are two ways to run an applet

1. By HTML file.
2. By appletviewer tool (for testing purpose).

Simple example of Applet by HTML file: -

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

Note: -

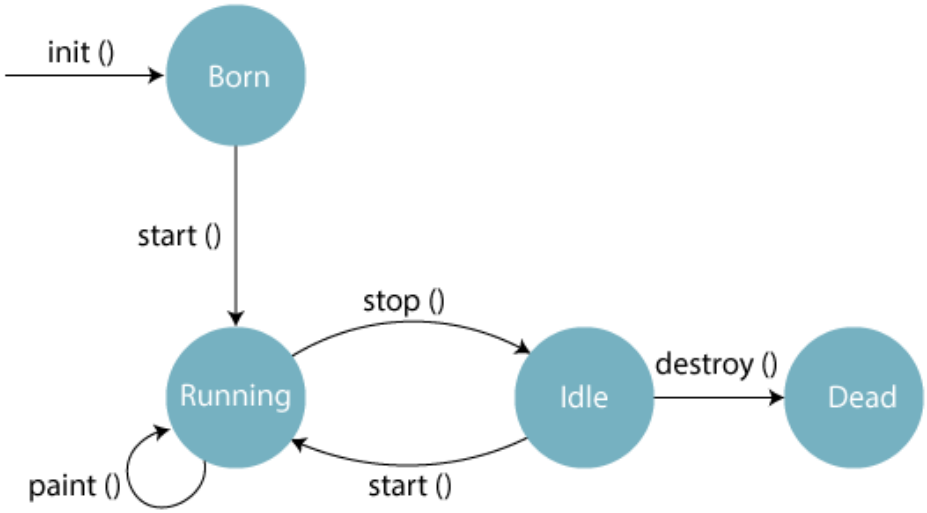
To compile applet program, the syntax is

javac Filename.java

To execute/run applet program the syntax is

AppletViewer Filename.java (or) AppletViewer Filename.html

Explain Life Cycle of an APPLET with a neat diagram?



We can describe the life cycle of an applet through four methods. They are ...

1. init(): - It is invoked at the first time, the applet is loaded into the memory. We can initialize variables and add components like buttons, check boxes, etc. to the applet in the init() method.

2. start(): - It is called immediately after the init() method.
it is also invoked every time the applet receives the focus.

3. stop(): - The stop( ) method is called every time the applet loses the focus. We can use this method to reset the variables and also to stop the threads that are currently running.

4. destroy(): - It is invoked by the browser when the user moves to another page. We can use this method to close the applet file.

Write differences between java application and java applet?

Parameters	Java Application	Java Applet
Definition	Applications are just like a Java program that can be executed independently without using the web browser.	Applets are small Java programs that are designed to be included with the HTML web document. They require a Java-enabled web browser for execution.
main () method	The application program requires a main() method for its execution.	The applet does not require the main() method for its execution instead init() method is required.
Compilation	The "javac" command is used to compile application programs, which are then executed using the "java" command.	Applet programs are compiled with the "javac" command and run using either the "appletviewer" command or the web browser.
File access	Java application programs have full access to the local file system and network.	Applets don't have local disk and network access.
Access level	Applications can access all kinds of resources available on the system.	Applets can only access browser-specific services. They don't have access to the local system.
Installation	First and foremost, the installation of a Java application on the local computer is required.	The Java applet does not need to be installed beforehand.
Program	An application program is needed to perform some tasks directly for the user.	An applet program is needed to perform small tasks or part of them.
Run	It cannot run on its own; it needs JRE to execute.	It cannot start on its own, but it can be executed using a Java-enabled web browser.
Connection with servers	Connectivity with other servers is possible.	It is unable to connect to other servers.
Read and Write Operation	It supports the reading and writing of files on the local computer.	It does not support the reading and writing of files on the local computer.
Security	Application can access the system's data and resources without any security limitations.	Executed in a more restricted environment with tighter security. They can only use services that are exclusive to their browser.
Restrictions	Java applications are self-contained and hence require no additional security because they are trusted.	Applet programs cannot run on their own, necessitating the maximum level of security.

What is Graphics class? Write various methods available in Graphics class?

The Graphics class is an abstract class that represents the display area of the applet. It is a part of the java.awt package. The Graphics class provides methods to draw a number of graphical figures including text, lines, circles and ellipses, rectangles and polygons, images, etc.

A few of the methods are given below

1. public abstract void drawString(String,column,row)
2. public abstract void drawLine(column, row, width, height)

3. public abstract void drawRect(column, row, width, height)
4. public abstract void fillRect(column, row, width, height)
5. public abstract void draw3DRect(column, row, width, height,boolean raised)
6.public abstract void rawRoundRect (column,row,width,height,arcWidth,arcHeight)
7. public abstract void fillRoundRect (column,row,width,height,arcWidth,arcHeight)
8. public abstract void drawOval(column,row,width,height)
9. public abstract void fillOval(column,row,width,height)
We can invoke the above methods using an object of Graphics class.

The Paint( ) Method: -
It is used to draw graphics in the drawing area of an applet. This method is automatically called at the first time the applet is displayed on the screen and every time the applet receives the focus. The paint method takes an object of the Graphics class as a parameter.

APPLET PROGRAMS

EX1: - Write java program to display a message on an applet

```
/* <Applet Code="JAP1.class" height=200 width=200> </Applet>*/
import java.applet.*;
import java.awt.*;
public class JAP1 extends Applet
{
    public void paint(Graphics  g)
    {
       g.drawString("Hello World",20,180);
    }
}
```

To compile the above program command is
javac JAP1.java
To execute/run the above program command is
AppletViewer JAP1.java

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

What is Font Class?
Font Class: - The Font class is used to state fonts, which are used to render text on an applet in a visible way. The constructor of Font class creates a new font with the specified style and size. The Font class is available in java.awt package.

EX2: -Write java program to print your name and address on an applet with ARIAL font 36 size

```
/* <Applet Code="JAP2.class" height=200 width=200> </Applet>*/
import  java.applet.*;
import  java.awt.*;
public class  JAP2  extends  Applet
{
   public  void  paint(Graphics   g)
   {
     Font  k =new Font("Arial", Font.BOLD, 36);
     g.setFont(k);
     g.drawString("Name : K. Babuji" , 100, 100);
     g.drawString("Job: Lecturer" , 100, 150);
```

```
    g.drawString("Address : Aditya, Kakinada" , 100, 200);
  }
}
```

EX3: -A program to display a message with different colors on an applet

```
import java.applet.Applet;
import java.awt.*;
/* <Applet Code="JAP3.class" height=200 width=200> </Applet>*/
public class JAP3 extends Applet
{
 public void paint(Graphics g)
 {
    Color c[] = {Color.blue,Color.cyan, Color.darkGray, Color.gray, Color.green, Color.lightGray,
                 Color.magenta, Color.orange, Color.pink, Color.red, Color.yellow, Color.black};
    for(int  i = 0; i<c.length; i++)
    {
      Font f = new Font("Ravie",Font.BOLD,26);
      g.setFont(f);
      g.setColor(c[i]);
      g.drawString("Hello World...", 50, 50+(i*30));
    }
 }
}
```

[imp Question] How to pass parameters to an applet?

Java applet has the feature of retrieving parameters passed from the HTML page. So, you can pass the parameters from HTML page to the applet. The **param** tag is used to pass the parameters to an applet. Parameters specify extra information that can be passed to an applet from the HTML page. The following is syntax of **param** tag.

Syntax: <parma name=" " value=" "> </param>

The <param> tag is a sub tag of the <applet> tag. The <param> tag contains two attributes **name** and **value** which are used to specify the name of the parameter and the value of the parameter respectively. For example, to pass name and age of a person to an applet, the following the <param> tags are used.

```
<param name="name" value="Ramesh" />
<param name="age" value="25" />
```

/* Write java program to create ParameterizedApplet to retrieve parameters from HTML page */

File1: - ParameterizedApplet.java

```
import java.applet.Applet;
import java.awt.Graphics;
public class ParameterizedApplet extends Applet
{
  public void paint(Graphics g)
  {
    String sn=getParameter("name");
    String a=getParameter("address");
    g.drawString("Student Name : " + sn, 50, 50);
    g.drawString("Address :  "  + a, 50, 100);
  }
}
```

Note: - To compile the program the command is **Javac ParameterizedApplet.java**

File2: - StudentData.HTML

```
<html>
<body>
<applet code=" ParameterizedApplet.class" width="300" height="300">
<param name="name" value="Srinivas Babu">
<param name="address" value="Bangalore ">
</applet>
</body>
</html>
```

Note: - To execute the program the command is **AppletViewer StudentData.HTML**

EX4: - Write java program to draw line, rectangle and a circle on an applet

```
/* <Applet Code="JAP4.class" height=200 width=200> </Applet>*/
import  java.applet.*;
import  java.awt.*;
public class  JAP4 extends Applet
{
   public void paint(Graphics g)
   {
      g.drawLine(50,50,300,50);
      g.drawRect(50,100,250,100);
      g.fillRect(350,100,250,100);
      g.drawOval(50,250,250,250);
      g.fillOval(350,250,250,250);
   }
}
```

EX5: - Write java program to draw a triangle on an applet

```
/* <applet code="JAP5.class" width=500 height=500></applet> */
import java.applet.*;
import java.awt.*;
public class JAP5 extends Applet
{
  public void paint(Graphics g)
  {
    g.drawLine(177,141,177,361);
    g.drawLine(177,141,438,361);
    g.drawLine(177,361,438,361);
  }
}
```

EX6: - Write java program to print natural numbers from 1 to 20 on an applet

```
/* <applet code=JAP6.class height=500 width=250> </applet> */
import java.awt.*;
import java.applet.*;
public class  JAP6 extends Applet
{
   public void paint(Graphics g)
   {
```

```
      try
     {
       for(int i = 1; i <= 20; i ++)
       {
         g.drawString(String.valueOf(i), 100, 100 + (i * 15));
         Thread.sleep(100);
       }
     }
     catch(InterruptedException e)
     {
       System.out.println(e);
     }
   }
}
```

TextField Class: - To accept textual data from a user, AWT provides TextField class. The TextField class handles a single line of text. To create a textbox, use the following syntax.
TextField object = new TextField(width);

TextArea Class: - To accept textual data from a user, we can use an object of TextArea class. The TextArea class allows to take multiple lines of text. To create a textbox to take multiple lines of text, use the following syntax. TextArea object = new TextArea(rows,columns);

Label Class: - The Label class is used to create titles. Labels do not generate an action event. To create a label, the syntax is as follows.
Label object = new Label("Title", alignment);
The default alignment is Left. We can specify the alignment as Label.RIGHT or Label.CENTER
Ex1:- Label L1= new Label("Student Name",Label.CENTER);
Ex2:- Label L2= new Label("Course",Label.RIGHT);
getText() method: - It is used to read a line of text from the specified TextField.
Syntax:-
StringVariable = TextFieldObject.getText();
setText() method: - It is used to insert the specified text in a TextField.
Syntax:-
TextFieldObject.setText('Text to be inserted');
EX7: - Write java program to read and print name and address on an applet

```
/* <Applet Code="JAP7.class" height=200 width=200> </Applet>*/
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class JAP7 extends Applet
{
 TextField  t1,t2;
 Label L1,L2;
 String x,y;
 Font f=new Font("Arial", Font.BOLD, 16);
 public void init( )
 {
  t1 = new TextField(10);
```

```
  t2 = new TextField(10);
  L1 = new Label("Enter Your Name");
  L2 = new Label("Enter Your Address");
  add(L1); add(t1); add(L2); add(t2);
 }
 public void paint(Graphics  g)
 {
  g.setFont(f);
  x=t1.getText();
  y=t2.getText();
  g.drawString("INPUT",100,20);
  g.drawString("OUTPUT",100,80);
  g.drawString("Name : " + x ,100,110);
  g.drawString("Address : " + y ,100,140);
 }
 public boolean action (Event e, Object obj)
 {
  repaint();
  return(true);
 }
}
```

EX8: - Write java program to perform List box event on an applet

```
/*  <applet code="JAP8.class" width=400 height=200>  </applet>  */
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;
public class JAP8 extends Applet implements ItemListener
{
      List x = new List(5, false);
      Label L = new Label("Select One Item");
      String msg = "";
      public void init()
      {
          add(L);
          x.add("TV");
          x.add("CAR");
          x.add("MOBILE");
          x.add("AC");
          x.add("CAMERA");
          x.add("LAPTOP");
          x.add("Android Tablet");
          add(x);
          x.addItemListener(this);
      }
      public void paint(Graphics g)
      {
         Font f = new Font("Times New Roman", Font.BOLD, 16);
         g.setFont(f);
         g.drawString("OUTPUT :: Selected item is "+ msg, 20, 120);
```

```
    }
    public void itemStateChanged(ItemEvent ie)
    {
       msg = x.getSelectedItem();
       repaint();
    }
}
```

Checkbox Class: -

The Checkbox class is used to create a labeled check box. A check box has two parts. They are label and state. The label is a string that represents the caption of the check box. The state is a boolean value that represents the status of the check box. By default, the status is false which represents the check box is unchecked. Check boxes are used to select one or two or all items in the list.

EX9: - Write java program to perform Check box event on an applet

```
/*  <applet code="JAP9.class" width=200 height=200>  </applet> */
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;
public class JAP9 extends Applet implements ActionListener
{
 Checkbox cb1 = null;
 Checkbox cb2 = null;
 Checkbox cb3 = null;
 Button b1;
 Button b2;
 Graphics g;
 String msg;
 public void init()
 {
   b1 = new Button("OUTPUT");
   b2 = new Button("CLEAR");
   cb1 = new Checkbox("Telugu");
   cb2 = new Checkbox("English");
   cb3 = new Checkbox("Hindi");
   add(cb1); add(cb2);  add(cb3);
   add(b1);  add(b2);
   b1.addActionListener(this);
   b2.addActionListener(this);
 }
 public void paint(Graphics g)
 {
   g.drawString(msg,10,50);
 }
 public void actionPerformed(ActionEvent  ae)
 {
   String a = ae.getActionCommand();
   if(a.equals("CLEAR"))
   {
     msg="";
     cb1.setState(false);
```

```
      cb2.setState(false);
      cb3.setState(false);
    }
    if(a.equals("OUTPUT"))
    {
         if(cb1.getState()==true &&  cb2.getState()==false && cb3.getState()==false)
              msg="You know Telugu Only";
         else if(cb1.getState()==false &&  cb2.getState()==true && cb3.getState()==false)
              msg="You know English Only";
         else if(cb1.getState()==false &&  cb2.getState()==false && cb3.getState()==true)
              msg="You know Hindi Only";
         else if(cb1.getState()==true && cb2.getState()==true && cb3.getState()==false)
              msg="You know Telugu and English";
         else if(cb1.getState()==true && cb2.getState()==false && cb3.getState()==true)
              msg="You know Telugu and Hindi";
         else if(cb1.getState()==false && cb2.getState()==true && cb3.getState()==true)
              msg="You know English and Hindi";
         else if(cb1.getState()==true && cb2.getState()==true && cb3.getState()==true)
             msg="wow ... you know all the three languages";
         else
             msg="Ok ... you don't know all the three languages";
    }
    repaint();
  }
}
```

Button Class: -

Buttons are used to perform trigger events in a GUI environment. The following syntax is used to create a button.

Button object=new Button("button name");

We can use setLabel() method to change button name. Also getLabel() method is used to retrieve the caption of a button.

Ex1:-Button b1=new Button("Save");

Ex2:-b1.setLabel("Delete") ;

Now the caption of button b1 becomes "Delete".

Ex3:- String s = b1.getLabel();

Now the value of variable "s" is "Delete".

Choice Class: -

A choice menu is used to display a list of choices for the user to select any one choice from it. The choice menu control is also knwon as drop down list box. To create a choice menu use the following syntax.

Choice object = new Choice();

We can add items to the choice menu by calling the addItem() method as follows.

choice-object.addItem("First Option");

choice-object.addItem("Second Option");

EX9: - Write java program to perform an event using drop-down list box on an applet

```
/*  <applet code="JAP8.class" width=800 height=400>  </applet> */
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class JAP8  extends Applet implements ItemListener, ActionListener
```

```
{
  String x,y;
  TextField  tf;
  Button  b1,b2;
  Label L1,L2;
  Choice  q;
  public void init( )
  {
    L1=new Label("Employee Name");
    L2=new Label("Qualification");
    tf= new TextField(20);
    b1 = new Button("OK");
    b2 = new Button("CLEAR");
    q = new Choice();
    q.addItem("SSC");
    q.addItem("ITI");
    q.addItem("INTER");
    q.addItem("DIPLOMA");
    q.addItem("DEGREE");
    q.addItem("B.TECH");
    q.addItem("PG");
    add(L1); add(tf);
    add(L2); add(q);
    add(b1); add(b2);
    q.addItemListener(this);
    b1.addActionListener(this);
    b2.addActionListener(this);
  }
  public  void  paint(Graphics  g)
  {
    g.drawString("OUTPUT : -" , 100,200);
    g.drawString("Candidate Name : " + x, 100,250);
    g.drawString("Qualification : " + y,100,300);
  }
  public  void  actionPerformed(ActionEvent  ae)
  {
    if(ae.getSource()==b1)
    {
        x=tf.getText();
        y=q.getSelectedItem();
    }
    if(ae.getSource()==b2)
    {
        tf.setText("");
        x=""; y="";
    }
     repaint();
  }
  public void itemStateChanged(ItemEvent arg)
```

```
  {
  }
}
```

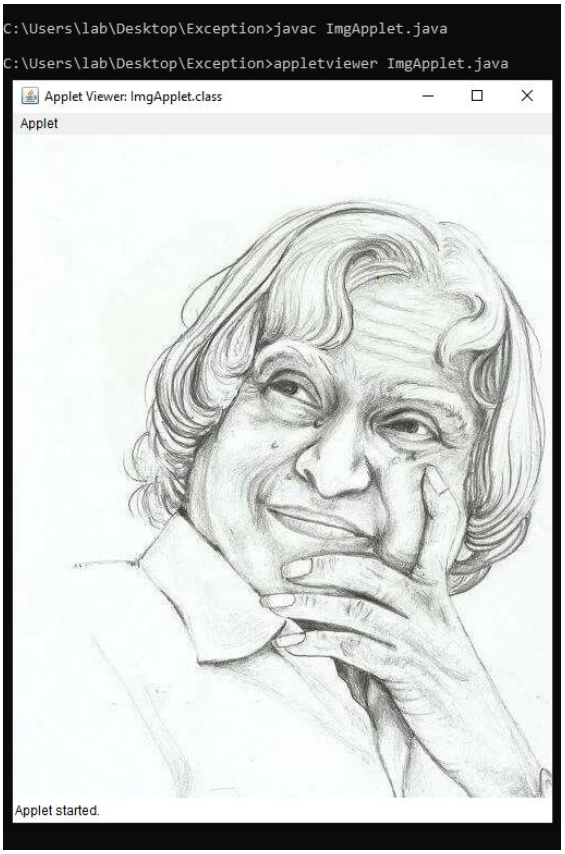
EX10: - Write java program to process employee records on an applet

```
/*  <applet code="JAP11.class" width=400 height=200>  </applet> */
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class JAP11 extends Applet implements ActionListener
{
 TextField  t1,t2,t3;
 Button  b1,b2,b3,b4;
 Label L1,L2,L3;
 String  n[] = new String[10];
 String  j[] = new String[10];
 int s[] = new  int[10];
 int  i,k;
 public void init( )
 {
  setLayout(new GridLayout(5,2));
  i=k=0;
  L1=new Label("Employee Name");
  L2=new Label("Job Title");
  L3=new Label("Monthly Salary");
  t1= new TextField(20);
  t2= new TextField(20);
  t3= new TextField(20);
  b1 = new Button("Add");
  b2 = new Button("First");
  b3 = new Button("Next");
  b4 = new Button("Clear");
  add(L1);add(t1);
  add(L2);add(t2);
  add(L3);add(t3);
  add(b1);add(b2);
  add(b3);add(b4);
  b1.addActionListener(this);
  b2.addActionListener(this);
  b3.addActionListener(this);
  b4.addActionListener(this);
 }
 public void actionPerformed(ActionEvent  ae)
 {
  if (ae.getSource()==b1)
  {
    n[i] = t1.getText();
    j[i] = t2.getText();
    s[i] = Integer.parseInt(t3.getText());
    i++;
```

```
    }
    if (ae.getSource()==b2)
    {
      k=0;
      t1.setText(n[k]);
      t2.setText( j[k] );
      t3.setText(String.valueOf(s[k]));
    }
    if (ae.getSource()==b3)
    {
      if(k<i-1)
        k++;
      t1.setText(n[k]);
      t2.setText( j[k] );
      t3.setText(String.valueOf(s[k]));
    }
    if (ae.getSource()==b4)
    {
      t1.setText(" ");
      t2.setText(" ");
      t3.setText(" ");
    }
  }
}
```

EX11: - Write java program to display an image on an applet

```
/* <applet code="JAP11.class" height=200 width=200>
 </applet> */
import  java.awt.*;
import java.applet.*;
public class JAP11 extends Applet
{
  Image  p1;
  public void init()
  {
    p1 = getImage(getDocumentBase(),"abdul-kalam.jpg");
  }
  public void paint(Graphics  g)
  {
    g.drawImage(p1,0,0,this);
  }
}
```



EX12: - Write java program to display animated motion image on an applet using a thread

```
/* <Applet Code="JAP12.class"
height=800 width=1600> </Applet>*/
import java.awt.*;
import java.applet.*;
public class JAP12 extends Applet
{
```

```
 Image pic1;
 public void init()
 {
  pic1 =getImage(getDocumentBase(),"bike_1.gif");
 }
 public void paint(Graphics g)
 {
  for(int i=0;i<1000;i++)
  {
   g.drawImage(pic1, i,30, this);
   try
   {
    Thread.sleep(5);
   }
   catch(Exception e){}
  }
 }
}
```

Note: - Before executing this program make sure that both the gif file and java program should be placed in the same folder.

Explain KeyEvent class and KeyListener interface? Write java program to implement key events?

KeyEvent Class: -

A key event is generated when we use keyboard. It is handled by KeyEvent class. There are three types of key events, which are identified by the integer constants namely KEY_PRESSED, KEY_RELEASED, and KEY_TYPED. The first two events are generated when any key is pressed or released. The last event occurs only when a character keystroke is given. Remember, not all keystrokes generate a character. For example, pressing shift key does not generate a character. There are many other integer constants that are defined by KeyEvent class. They are VK_SHIFT, VK_ALT, VK_CONTROL, VK_ENTER, VK_ESCAPE, VK_PAGE_UP, VK_PAGE_DOWN, VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN, etc. The VK constants specify virtual key codes.

KeyListener Interface: -

The KeyListener is notified whenever the state of a key is changed. It is notified against KeyEvent. The KeyListener interface is found in java.awt.event package, and it has the following three methods.

1. public abstract void keyPressed (KeyEvent e) : It is invoked when a key has been pressed.
2. public abstract void keyReleased (KeyEvent e) : It is invoked when a key has been released.
3. public abstract void keyTyped (KeyEvent e) : It is invoked when a key has been typed.

EX13: - Write java program to implement keyboard events on an applet?

```
/* <Applet code="KeyBoard.class" height=100 width=700></Applet> */
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class KeyBoard extends Applet implements KeyListener
{
   String msg1, msg2;
   public void init()
   {
      msg1=msg2="";
      addKeyListener(this);
```

```
    requestFocus();
  }
  public void keyTyped(KeyEvent ke)
  {
    msg1=""+ke.getKeyChar();
    msg2=" is pressed";
    repaint();
  }
  public void paint(Graphics g)
  {
    g.drawString(msg1+msg2,100,50);
  }
  public void keyReleased(KeyEvent ke)
  {
    msg1="Key Released";
    msg2=" ";
    repaint();
  }
  public void keyPressed(KeyEvent ke)
  {
    msg2=" is pressed";
    int k=ke.getKeyCode();
    switch(k)
    {
      case KeyEvent.VK_F1:msg1="F1";
        break;
      case KeyEvent.VK_F2:msg1="F2";
        break;
      case KeyEvent.VK_F3:msg1="F3";
        break;
      case KeyEvent.VK_ALT:msg1="Alt";
        break;
      case KeyEvent.VK_CONTROL:msg1="Ctrl";
        break;
      case KeyEvent.VK_ESCAPE:msg1="Esc";
        break;
      case KeyEvent.VK_PAGE_UP:msg1="Pageup";
        break;
      case KeyEvent.VK_PAGE_DOWN:msg1="Pagedown";
        break;
      case KeyEvent.VK_LEFT:msg1="Left arrow";
        break;
      case KeyEvent.VK_RIGHT:msg1="Right arrow";
        break;
    }
    repaint();
  }
}
```

Explain about MouseEvent class, MouseListener and MouseMotionListener interfaces in java? Write java program to implement mouse events?

MouseEvent Class: -

There are eight types of mouse events. The MouseEvent class defines the following integer constants that can be used to identify them the mouse events.

MOUSE_CLICKED	: The user clicked the mouse.
MOUSE_DRAGGED	: The user dragged the mouse.
MOUSE_ENTERED	: The mouse entered a component.
MOUSE_EXITED	: The mouse exited from a component.
MOUSE_MOVED	: The mouse moved.
MOUSE_PRESSED	: The mouse was pressed.
MOUSE_RELEASED	: The mouse was released.
MOUSE_WHEEL	: The mouse wheel was moved.

MouseEvent is a subclass of InputEvent.

Two commonly used methods in MouseEvent class are getX() and getY(). These methods return the X and Y coordinates of the mouse within the component when the event occurred.

MouseListener Interface: -

The MouseListener is notified whenever the state of a mouse is changed. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods as follows.

1. public abstract void mouseClicked(MouseEvent e) : It is invoked when the mouse button is clicked
2. public abstract void mouseEntered(MouseEvent e) : It is invoked when the mouse entered into the component.
3. public abstract void mouseExited(MouseEvent e)) : It is invoked when the mouse exited from the component.
4. public abstract void mousePressed(MouseEvent e) : It is invoked when the mouse button is hold-downed.
5. public abstract void mouseReleased(MouseEvent e) : It is invoked when the mouse button is released.

MouseMotionListener Interface: -

The MouseMotionListener is notified whenever the mouse is in moving state. It is notified against MouseEvent. The MouseMotionListener is also present in awt package. It has two methods as follows.

1. public abstract void mouseMoved(MouseEvent e) : It is invoked when the mouse is moved.
2. public abstract void mouseDragged(MouseEvent e) : It is invoked when the mouse is dragged.

EX14: - Write java program to perform mouse events on an applet

Filename: Mymouse.java

```
/*<applet code="Mymouse.class" height=200 width=400> </applet>*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class Mymouse extends Applet implements MouseListener, MouseMotionListener
{
 String msg;
 public void init()
 {
   msg="Applet started";
   addMouseListener(this);
   addMouseMotionListener(this);
 }
```

```
 public void paint(Graphics g)
 {
   Font f=new Font("Arial",Font.BOLD,16);
   g.setFont(f);
   g.drawString(msg,100,100);
 }
 public void mouseEntered(MouseEvent e)
 {
   msg="Mouse entered into the applet";
   repaint();
 }
   public void mouseExited(MouseEvent e)
 {
   msg="Mouse Exited from the applet";
   repaint();
 }
 public void mouseClicked(MouseEvent e)
 {
   msg="Mouse clicked on the applet";
   repaint();
 }
 public void mouseDragged(MouseEvent e)
 {
   msg="Mouse dragged on the applet";
   repaint();
 }
 public void mouseReleased(MouseEvent e)
 {
   msg="Mouse Button Released";
   repaint();
 }
 public void mousePressed(MouseEvent e)
 {
   msg="Mouse Button Pressed";
   repaint();
 }
 public void mouseMoved(MouseEvent  me)
 {
   showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
 }
}
```

JAVA SWING

Java Swing is more familiar and advanced than the applet. Swing provides latest tools like animated buttons, tabbed panes, etc. Swing is a library with Java Foundation Classes [JFC]. Swing is an extension of AWT. Swing offers much-improved functionality than AWT. It provides new components, expanded components features and excellent event handling with drag-and-drop support. Swing is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent lightweight components. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing?

1. AWT components are platform-dependent. Java swing components are platform-independent.
2. AWT components are heavyweight. Swing components are lightweight.
3. AWT doesn't support pluggable look and feel. Swing supports pluggable look and feel.
4. AWT provides less components than Swing. Swing provides more powerful and advanced components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5. AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. Swing follows MVC.

Write different features of Java Swing?

Platform Independent: javax.swing is a platform independent tool which provides components that run under any client machine. It provides GUI look and feel, owned and delivered by a platform-specific OS.
Lightweight Components: Starting with the JDK 1.1, swing supported lightweight component development. For a component to qualify as lightweight, it must not depend on any non-Java (OS based) system classes. Swing components have their own view supported by Java's look and feel classes.
Pluggable Look and Feel: This feature enables the user to switch the look and feel of Swing components without restarting an application. The Swing library supports components' look and feels that remain the same across all platforms wherever the program runs. The Swing library provides an API that gives real flexibility in determining the look and feel of the GUI of an application
Highly Customizable: Swing controls can be customized in a very easy way as visual appearance is independent of internal representation.
Rich Controls: Swing provides a rich set of advanced controls like Tree TabbedPane, slider, color picker, and table controls.

Explain different classes available in javax.swing package?

javax standards for Java extensions. Javax includes extensions such as **javax.swing, javax.servlet**, **javax.jcr**, etc. **javax.swing** deals with classes for java swing API such as JLabel, JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc. **javax.servlet** deals with running servlets, and **javax.jcr** deals with the Java Content Library.

Classes available in javax.swing: -
1. JLabel: - It is used to display a short string or an image icon. JLabel can display text, image or both. JLabel is only a display of text or image and it cannot get focus. The commonly used methods of JLabel class are getIcon(), setIcon(Icon i), getText() and setText(), etc.

2. JTextField: - The object of a **JTextField** class is to create a text box. It allows to enter text in a single line. By using setText() and getText() methods we can process the text within the JTextField object.

3. JPasswordField: - The object of a **JPasswordField** class is a text component specialized for password entry. It allows to enter text in a single line as a password. It inherits JTextField class. To read the password entered by the user is obtained by using **new String (JPasswordField Object.getPassword())** method.

4. JApplet: - Applet is in AWT whereas JApplet is in swing. JApplet is the extended version of Applet, therefore it is more recent. JApplet is a GUI container that has some extra structure to allow it to be used in the "alien" environment of a web browser. It provides inti(), paint(), start(), stop() and destroy() methods.

5. JButtoon: - JButton is an implementation of a push button available in java swing package. This component has a label and generates an event when pressed. It can also have an image. JButton provides buttons with

curved or rounded edges, coloured buttons and animated buttons. The methods used with JButton object are getText(), setText(), getIcon(), setIcon(), addActionListener(), setBounds(), addActionListener(), etc.

6. JComboBox: - It inherits JComponent class. JComboBox shows a popup menu that shows a list and the user can select a option from that specified list. JComboBox can be editable or read- only depending on the choice of the programmer. **addItem(), addItemListener(), getItemCount(),getSelectedItem(), showPopup(), setSelectedIndex(), setActionCommand(String a) etc.** are methods available in JButton class.

7. JFileChooser: - This class is used to create a control with a dialog window from which the user can select a file. getAbsolutePath() method of FileChooser class is used to find the path of a file.

8. JWindow: - It is a container that can be displayed anywhere on the desktop.

9. JViewpoint: - It creates the "viewport" or "porthole" through which we see the underlying information. When we scroll, what moves is the viewport. It is like peering through a camera's viewfinder. Moving the viewfinder upwards brings new things into view at the top of the picture and loses things that were at the bottom.

10. JPanel: - It is a generic lightweight container. add(), setLayout(), setUI(), getUI(),**updateUI()** etc. are methods in JPanel class.

A program to read and print details of an employee on an applet using swings

```
/* <applet code="Swing1.class" height=200 width=200> </applet> */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Swing1 extends JApplet implements ActionListener
{
    JTextField  t1,t2,t3;
    JButton  b1,b2,b3;
    JLabel L1,L2,L3;
    String  n, j, s;
    public void init( )
    {
        L1=new JLabel("Employee Name");
        L2=new JLabel("Job Title");
        L3=new JLabel("Monthly Salary");
        t1= new JTextField(20);
        t2= new JTextField(20);
        t3= new JTextField(20);
        b1 = new JButton("Insert");
        b2 = new JButton("Display");
        b3 = new JButton("Clear");
        Container cp = getContentPane();
        cp.setLayout(new FlowLayout());
        cp.add(L1); cp.add(t1);
        cp.add(L2); cp.add(t2);
        cp.add(L3); cp.add(t3);
        cp.add(b1); cp.add(b2);  cp.add(b3);
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
    }
    public void actionPerformed(ActionEvent  ae)
    {
```

```
        if (ae.getSource()==b1)
        {
            n = t1.getText();
            j = t2.getText();
            s = t3.getText();
        }
        if (ae.getSource()==b2)
        {
            t1.setText(n);
            t2.setText( j );
            t3.setText(s);
        }
        if (ae.getSource()==b3)
        {
            t1.setText(" ");
            t2.setText(" ");
            t3.setText(" ");
        }
    }
}
```

What is JRadioButton? Create an applet to perform an event using JRadioButton?

javax.swing provides JRadioButton class to create a radio button. Radio button is use to select one option from multiple options. It is used in filling forms, online objective papers and quiz.

We add radio buttons in a ButtonGroup so that we can select only one radio button at a time. We use "ButtonGroup" class to create a ButtonGroup and add radio buttons to the group.

Methods used with JRadioButton: -

The method addActionListener(), is used to register the JRadioButton event.

The methgod, actionPerformed(ActionEvent) is used to describe the event of JRadioButton.

The methgod, getActionCommand() is used to find out which radio button is being selected.

Ex1: - /* Create an applet to perform an event using JRadioButton*/

```
/* <applet code="Swing2.class" height=200 width=400> </applet> */
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import javax.swing.*;
public class Swing2 extends JApplet implements ActionListener
{
   JLabel   jl;
    JTextField  tf;
    JRadioButton  b1,b2,b3,b4;
    public void init( )
    {
        tf= new JTextField(20);
        jl = new JLabel("Select Your Course");
        Container cp = getContentPane();
        cp.setLayout(new FlowLayout());

        b1 = new JRadioButton("C");
```

```
        b2 = new JRadioButton("C++");
        b3 = new JRadioButton("JAVA");
        b4 = new JRadioButton("Oracle");

        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        b4.addActionListener(this);

        ButtonGroup bg = new ButtonGroup();
        bg.add(b1);
        bg.add(b2);
        bg.add(b3);
        bg.add(b4);

        tf.setForeground(Color.blue);
        tf.setBackground(Color.yellow);
        tf.setHorizontalAlignment(JTextField.CENTER);
        cp.add(jl);
        cp.add(b1);
        cp.add(b2);
        cp.add(b3);
        cp.add(b4);
        cp.add(tf);
    }
    public void actionPerformed(ActionEvent ae)
    {
        tf.setText("Your course is  " + ae.getActionCommand());
    }
}
```

Ex2: - /* Write java program to perform an event using JRadioButton */

```
import javax.swing.*;
import java.awt.event.*;
class RadioExample extends JFrame implements ActionListener
{
 JRadioButton rb1,rb2;
 JButton b;
 RadioExample()
 {
  rb1=new JRadioButton("Male");
  rb1.setBounds(100,50,100,30);
  rb2=new JRadioButton("Female");
  rb2.setBounds(100,100,100,30);
  ButtonGroup bg=new ButtonGroup();
  bg.add(rb1);bg.add(rb2);
  b=new JButton("Submit");
  b.setBounds(100,150,80,30);
  b.addActionListener(this);
```

```
   add(rb1);add(rb2);add(b);
   setSize(300,300);
   setLayout(null);
   setVisible(true);
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
  public void actionPerformed(ActionEvent e)
  {
   if(rb1.isSelected())
   {
    JOptionPane.showMessageDialog(this,"You are male");
   }
   if(rb2.isSelected())
   {
    JOptionPane.showMessageDialog(this,"You are female");
   }
  }
}
class Swing4
{
   public static void main(String args[])
   {
    new RadioExample();
   }
}
```

TABBED PANES

A tabbed pane is a component that appears as a group of folders in a file cabinet. Each folder has a title. When a user selects a folder, its contents become visible. Only one of the folders may be selected at a time. Tabbed Panes are commonly used for setting configuration options. Tabbed panes are encapsulated by the JTabbedPane class, which extends JComponent.

Ex: - A program to design tabbed panes

```
/* <applet code="Swing3.class" height=200 width=200> </applet> */
import  javax.swing.*;
import  java.awt.*;
class CitiesPanel extends JApplet
{
  Container cp;
  public  CitiesPanel()
  {
    cp = getContentPane();
    cp.setLayout(new FlowLayout());
    JButton  b1 = new JButton("New York");
    JButton  b2 = new JButton("London");
    JButton  b3 = new JButton("New Delhi");
    JButton  b4 = new JButton("Hong Kong");
    cp.add(b1);
    cp.add(b2);
    cp.add(b3);
    cp.add(b4);
```

```
  }
}
class ColoursPanel extends JApplet
{
  Container cp;
  public  ColoursPanel()
  {
    cp = getContentPane();
    cp.setLayout(new FlowLayout());
    JCheckBox  cb1 = new JCheckBox("Red");
    JCheckBox  cb2 = new JCheckBox("Green");
    JCheckBox  cb3 = new JCheckBox("Yellow");
    JCheckBox  cb4 = new JCheckBox("Orange");
    cp.add(cb1);
    cp.add(cb2);
    cp.add(cb3);
    cp.add(cb4);
  }
}
class FlavorsPanel extends JApplet
{
  Container cp;
  public  FlavorsPanel()
  {
    cp = getContentPane();
    cp.setLayout(new FlowLayout());
    JComboBox jcb = new JComboBox();
    jcb.addItem("Vanilla");
    jcb.addItem("Chocolate");
    jcb.addItem("Strawberry");
    jcb.addItem("ButterScotch");
    cp.add(jcb);
  }
}
public class Swing3 extends JApplet
{
 Container cp;
 public  void  init()
 {
   cp = getContentPane();
   cp.setLayout(new FlowLayout());
   JTabbedPane jtp = new JTabbedPane();
   jtp.addTab("Cities" , new CitiesPanel());
   jtp.addTab("Colours" , new ColoursPanel());
   jtp.addTab("Flavors" , new FlavorsPanel());
   cp.add(jtp);
 }
}
```

JColorChooser

It provides a pane of controls designed to allow a user to manipulate and select a color.

Creating a Custom Chooser Panel: The default color chooser provides five chooser panels as follows.

1. **Swatches:** For choosing a color from a collection of swatches.
2. **HSV:** For choosing a color using the Hue-Saturation-Value color representation. Prior to JDK 7, It was known as HSB, for Hue-Saturation-Brightness.
3. **HSL:** For choosing a color using the Hue-Saturation-Lightness color representation.
4. **RGB:** For choosing a color using the Red-Green-Blue color model.
5. **CMYK:** For choosing a color using the process color or four color model.

Below programs illustrate the use of JColorChooser class:

Java program to implement JColorChooser class using ChangeListener interface?

In this program, we first create a label at the top of the window where some text is shown in which we will apply color changes. Set the foreground and background color. Set the size and type of the font. Create a Panel and set its layout. Now set up the color chooser for setting text color.
Using *stateChanged()* method, event is generated for change in color of the text by using *getColor()* method. Now create the GUI, create a setup window. Set the default close operation of the window. Create and set up the content Pane and add content to the frame and display the window.

Ex: - /* Write java program to implement a ColorChooser interface */

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.swing.event.*;
public class ColorChooserDemo extends JPanel implements ChangeListener
{
   protected JColorChooser Jcc;
   protected JLabel label;

   public ColorChooserDemo()
   {
      super(new BorderLayout());

      label = new JLabel("Welcome to GeeksforGeeks", JLabel.CENTER);
      label.setForeground(Color.green);
      label.setBackground(Color.WHITE);
      label.setOpaque(true);
      label.setFont(new Font("SansSerif", Font.BOLD, 30));
      label.setPreferredSize(new Dimension(100, 65));

      JPanel bannerPanel = new JPanel(new BorderLayout());
      bannerPanel.add(label, BorderLayout.CENTER);
      bannerPanel.setBorder(BorderFactory.createTitledBorder("Label"));

     Jcc = new JColorChooser(label.getForeground());
     Jcc.getSelectionModel().addChangeListener(this);
     Jcc.setBorder(BorderFactory.createTitledBorder("Choose Text Color"));

      add(bannerPanel, BorderLayout.CENTER);
      add(Jcc, BorderLayout.PAGE_END);
```

```
    }

    public void stateChanged(ChangeEvent e)
    {
        Color newColor = Jcc.getColor();
        label.setForeground(newColor);
    }

    private static void createAndShowGUI()
    {

        JFrame frame = new JFrame("ColorChooserDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JComponent newContentPane = new ColorChooserDemo();

        newContentPane.setOpaque(true);
        frame.setContentPane(newContentPane);
        frame.pack();
        frame.setVisible(true);
    }

   public static void main(String[] args)
   {
        javax.swing.SwingUtilities.invokeLater(new Runnable()
        {
          public void run()
          {
             createAndShowGUI();
          }
        });
    }
}
```

What is a Slider? Create an applet to perform an event using Slider?

Slider: - The Slider class is used to create slider control in java which is used to display a continuous or discrete range of valid numeric choices and allows the user to interact with the control. A slider is rendered as a vertical or horizontal bar with a knob that the user can slide to indicate the desired value.

Methods of Slider class: -

getMax()	To get maximum value of the slider.
getMin()	To get minimum value of the slider.
getValue()	To get current value of the slider.
setMax(double value)	To set maximum value to a slider.
setMin(double value)	To set minimum value to a slider.

```
/* Create an applet to perform an event using Slider */
import javax.swing.*;
import javax.swing.event.*;
public class SliderExample extends JFrame implements ChangeListener
{
  JSlider slider;
```

```
 JButton jb;
 JPanel panel;
 public SliderExample()
 {
   slider = new JSlider(JSlider.HORIZONTAL, 1, 75, 25);
   jb= new JButton("OUTPUT");
   slider.setMinorTickSpacing(2);
   slider.setMajorTickSpacing(10);
   slider.setPaintTicks(true);
   slider.setPaintLabels(true);
   slider.addChangeListener(this);
   panel=new JPanel();
   panel.add(slider);
   panel.add(jb);
   add(panel);
 }
 public void stateChanged(ChangeEvent  e)
 {
   if(e.getSource()==slider)
     jb.setText("Your Age: "+ slider.getValue() +" Years");
 }
 public static void main(String s[])
 {
   SliderExample frame=new SliderExample();
   frame.pack();
   frame.setVisible(true);
 }
}
```

What is JTree? Write java program to create a tree?

JTree is a Swing component with which we can display hierarchical data. JTree is quite a complex component. A JTree has a '**root node**' which is the top-most parent for all nodes in the tree. A node is an item in a tree. A node can have many children nodes. These children nodes themselves can have further children nodes. If a node doesn't have any children node, it is called a **leaf node**. By using the constructor of DefaultMutableTreeNode class we can define nodes to JTree object.

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample
{
  JFrame f;
  TreeExample()
  {
    f=new JFrame();
    DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
    DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
    DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
    style.add(color);
    style.add(font);
    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
```

```
    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");

    DefaultMutableTreeNode darkblue=new DefaultMutableTreeNode("Dark Blue");
    DefaultMutableTreeNode skyblue=new DefaultMutableTreeNode("Sky Blue");
    DefaultMutableTreeNode navyblue=new DefaultMutableTreeNode("Navy Blue");
    DefaultMutableTreeNode royalblue=new DefaultMutableTreeNode("Royal Blue");

    DefaultMutableTreeNode arial=new DefaultMutableTreeNode("Arial");
    DefaultMutableTreeNode calibri=new DefaultMutableTreeNode("Calibri");
    DefaultMutableTreeNode vivaldi=new DefaultMutableTreeNode("Vivaldi");
    DefaultMutableTreeNode simsun=new DefaultMutableTreeNode("SimSun");

    color.add(red); color.add(blue); color.add(black); color.add(green);
    blue.add(darkblue); blue.add(skyblue); blue.add(navyblue); blue.add(royalblue);
    font.add(arial); font.add(calibri); font.add(vivaldi); font.add(simsun);
    JTree jt=new JTree(style);
    f.add(jt);
    f.setSize(200,200);
    f.setVisible(true);
  }
  public static void main(String[] args)
  {
    new TreeExample();
  }
}
```

What is JTable? Write java program to display a table of on a frame using JTable?

The JTable class is a part of Java Swing Package and is generally used to display or edit two-dimensional data that is having both rows and columns. It is similar to a spreadsheet. It arranges data in a tabular form.

Constructors in JTable Class: -

- JTable(): A table is created with empty cells.
- JTable(int rows, int cols): Creates a table of size rows * cols.

Methods of JTable Class: -

- addColumn(TableColumn []column) : It adds a column at the end of the JTable.
- clearSelection() : It clears the contents in all the selected cells.
- editCellAt(int row, int col) : It edits the contents of the specified cell programmatically, if the given indices are valid and the corresponding cell is editable.
- setValueAt(Object value, int row, int col) : It sets the cell value with "value" in the specified row and column position.

```
/* Write java program to display a table on a frame */
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.SwingUtilities;
public class TableExample extends JFrame
{
```

```
  public TableExample()
  {
     //headers for the table
    String[] columns = new String[] {"Id", "Name", "Hourly Rate", "Department"};
    //actual data for the table in a 2nd array
    Object[][] data = new Object[][] {{1, "Sampath", 40.0, "Civil" },{2, "Sudheer", 70.0, "Mechanical"},
    {3, "Sesidhar", 60.0, "Electronics" }};
    //create table with data
    JTable table = new JTable(data, columns);
    //add the table to the frame
    this.add(new JScrollPane(table));
    this.setTitle("Table Example"); this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.pack();
    this.setVisible(true);
  }
  public static void main(String[] args)
  {
    SwingUtilities.invokeLater(new Runnable()
    {
      public void run()
      {
        new TableExample();
      }
    });
  }
}
```

What is JScrollPane? Write java program using JScrollPane with JFrame?

A scroll pane is a container that represents a small area to view other component. If the component is larger than the visible area, scroll pane provides horizontal and/or vertical scroll bars automatically for scrolling the components through the pane. A scroll pane is an object of the JScrollPane class which extends JComponent.

```
/* Write java program using JScrollPane with JFrame */
import java.awt.BorderLayout;
import java.awt.GridLayout;
import javax.swing.ButtonGroup;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
public class ScrollPanel extends JFrame
{
  JScrollPane scrollpane;
  public ScrollPanel()
  {
    super("JScrollPane Demonstration");
    setSize(300, 200);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    init();
    setVisible(true);
```

```
  }
  public void init()
  {
     JRadioButton form[][] = new JRadioButton[12][5];
     String counts[] = {"", "1 star", "2 star", "3 star", "4 star", "5 star"};
     String categories[] = {"HTML", "C Language", "Java", "Sudo Placement", "Python", "CS Subject",
       "Operating System", "Data Structure", "Algorithm", "PHP language", "JAVASCRIPT", "C Sharp" };
      JPanel p = new JPanel();
      p.setSize(600, 400);
      p.setLayout(new GridLayout(13, 6, 10, 0));
      for (int row = 0; row < 13; row++)
     {
        ButtonGroup bg = new ButtonGroup();
         for (int col = 0; col < 6; col++)
         {
           if (row == 0)
           {
              p.add(new JLabel(counts[col]));
           }
           else
           {
              if (col == 0)
              {
                 p.add(new JLabel(categories[row-1]));
              }
              else
              {
                 form[row-1][col-1] = new JRadioButton();
                 bg.add(form[row-1][col-1]);
                 p.add(form[row-1][col-1]);
              }
           }
        }
     }
     scrollpane = new JScrollPane(p);
     getContentPane().add(scrollpane, BorderLayout.CENTER);
  }
  public static void main(String args[])
  {
     new ScrollPanel();
  }
}
```

Explain about MVC architecture in Java?

The Model-View-Controller (MVC) is a well-known design pattern in the web development field. It is way to organize our code. It specifies that a program or application shall consist of data model, presentation information and control information. The MVC pattern needs all these components to be separated as different objects.

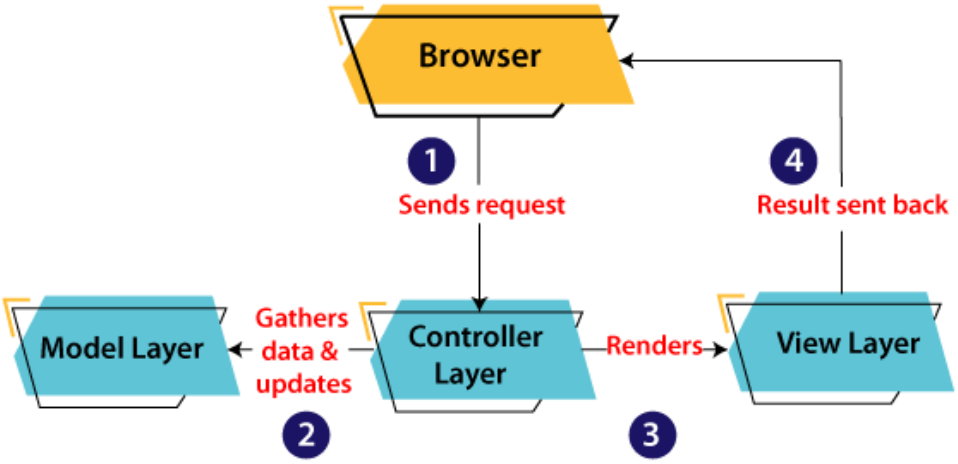
The MVC pattern architecture consists of three layers:

- **Model:** It represents the business layer of application. It is an object to carry the data that

can also contain the logic to update controller if data is changed.

- **View:** It represents the presentation layer of application. It is used to visualize the data that the model contains.
- **Controller:** It works on both the model and view. It is used to manage the flow of application, i.e. data flow in the model object and to update the view whenever data is changed.

In Java Programming, the Model-View-Controller contains the Java classes, the View used to display the data and the Controller contains the servlets. Due to this separation the user requests are processed as follows:



1. A client (browser) sends a request to the controller on the server side, for a page.
2. The controller then calls the model. It gathers the requested data.
3. Then the controller transfers the data retrieved to the view layer.
4. Now the result is sent back to the browser (client) by the view.

Advantages of MVC Architecture: -

The advantages of MVC architecture are as follows:

- MVC has the feature of scalability that in turn helps the growth of application.
- The components are easy to maintain because there is less dependency.
- A model can be reused by multiple views that provides reusability of code.
- The developers can work with the three layers (Model, View, and Controller) simultaneously.
- Using MVC, the application becomes more understandable.
- Using MVC, each layer is maintained separately therefore we do not require to deal with massive code.
- The extending and testing of application is easier.

Creating Menus in Java: - There are two different kinds of menus supported by Java. They are regular menus and pop-up menus. Regular menu is present on the menu bar whereas popup menu is displayed when you chick on the frame. javax.swing provides the following classes to create and manage menus.

1.JMenuBar: Used to create a menu bar.
2.JMenu: Used to create menus like file, edit, etc.
3.JMenuItem: Used to create menu items like new, open, save, print, exit, etc.
4.JCheckBoxMenuItem : Used to create a checkbox in a menu.
5.JRadioButtonMenuItem : Used to create a radio button in a menu.

Write a program to create menu bar with regular menus using swings */

```
import  javax.swing.*;
public class Swing5
{
public static void main(String args[])
{
JFrame  f = new JFrame("Student Details Window");
```

```
JMenuBar  mb =  new  JMenuBar( );
f.setJMenuBar(mb);
JMenu  fm = new JMenu("File");
JMenu  em = new JMenu("Edit");
JMenu  hm = new JMenu("Help");
mb.add(fm);
mb.add(em);
mb.add(hm);
JMenuItem  mnew,open,save,close,exit,cut,copy,paste, del;
JMenuItem about,topics;
JCheckBoxMenuItem  find;
mnew = new JMenuItem("New");
open  = new JMenuItem("Open");
save  = new JMenuItem("Save");
close  = new JMenuItem("Close");
exit  = new JMenuItem("Exit");
cut  = new JMenuItem("Cut");
copy  = new JMenuItem("Copy");
paste  = new JMenuItem("Paste");
del  = new JMenuItem("Delete");
about = new JMenuItem("About Pace");
topics = new JMenuItem("Help Topics");
find = new JCheckBoxMenuItem("Find");
fm.add(mnew);
fm.add(open);
fm.add(save);
fm.add(close);
close.setEnabled(false);
em.add(cut);
em.add(copy);
em.add(paste);
em.add(del);
em.add(find);
hm.add(about);
hm.add(topics);
JMenu  pm = new JMenu("Print");
JMenuItem  setup, cp, ap;
setup = new JMenuItem("Page Setup");
cp = new JMenuItem("Current Page");
ap = new JMenuItem("All Pages");
fm.add(pm);
fm.add(exit);
pm.add(setup);
pm.add(cp);
pm.add(ap);
f.setSize(400,400);
f.setVisible(true);
}
}
```

/* Write java program to provide a popup menu in an applet and perform menu item selected event with showMessageDialog method using swing */

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class PopupMenuExample extends JApplet  implements ActionListener
{
 private JPopupMenu popup = new JPopupMenu();
 private JLabel l1 = new JLabel("Select Your County using POPUP menu");
 public void init()
 {
  Container cp = getContentPane();
  cp.setLayout(new FlowLayout());
  cp.add(l1);
  JMenuItem m = new JMenuItem("America");
  m.addActionListener(this);
  popup.add(m);
  m = new JMenuItem("Japan");
  m.addActionListener(this);
  popup.add(m);
  m = new JMenuItem("Africa");
  m.addActionListener(this);
  popup.add(m);
  popup.addSeparator();
  m = new JMenuItem("India");
  m.addActionListener(this);
  popup.add(m);
  PopupListener pl = new PopupListener();
  addMouseListener(pl);
  addMouseListener(pl);
 }
 public void actionPerformed(ActionEvent e)
 {
  JOptionPane.showMessageDialog(this,"Your Country is " + ((JMenuItem) e.getSource()).getText());
 }
 class PopupListener extends MouseAdapter
 {
  public void mousePressed(MouseEvent e)
  {
   maybeShowPopup(e);
  }
  public void mouseReleased(MouseEvent e)
  {
   maybeShowPopup(e);
  }
  private void maybeShowPopup(MouseEvent e)
  {
   if (e.isPopupTrigger())
```

```
      popup.show(((JApplet) e.getComponent()).getContentPane(), e.getX(), e.getY());
    }
  }
  public static void run(JApplet applet, int width, int height)
  {
    JFrame frame = new JFrame("Country Applet");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().add(applet);
    frame.setSize(width, height);
    applet.init();
    applet.start();
    frame.setVisible(true);
  }
  public static void main(String[] args)
  {
    run(new PopupMenuExample(), 600, 400);
  }
}
```

```
C:\Users\lab\Desktop>javac PopupMenuExample.java

C:\Users\lab\Desktop>java PopupMenuExample
```