UNIT-V: EVENT DRIVEN PROGRAMMING

PART-A [2 MARKS]

1. What is meant by window adapter classes? (Nov/Dec 2018)

Java adapter classes provide the default implementation of listener interfaces. If you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listener interfaces. So it saves code. The adapter classes are found in **java.awt.event, java.awt.dnd and javax.swing.event** packages.

2. Enumerate the features of AWT in Java. (Nov/Dec 2018)

- **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.

3. What is the relationship between an event-listener interface and an event-adapter class? (May/Jun 2013)

When you declare to implement an event listener, then you should write the definitions of all the methods that have been declared in the interface. This causes inconvenience to the programmers. They need to implement the methods that would never be used. One way to do away with this inconvenience is to use adapter classes. An event adapter class has empty implementations by default.

4. Why swing components are called lightweight components? (May/Jun 2013)

Swing components depend less on the target platform and use less of the native GUI resource.

Hence the Swing components that don't rely on native GUI are referred to as lightweight components.

AWT components on the other hand are referred to as heavyweight components.

5. What is AWT? (May/Jun 2012, Nov/Dev 2012)

Abstract Window Toolkit (AWT) is a set of application program interfaces (API s) used by Java programmers to create graphical user interface (GUI) objects, such as buttons, scroll bars, and windows.

AWT is part of the Java Foundation Classes (JFC) from Sun Microsystems, the company that originated Java. The JFC are a comprehensive set of GUI class libraries that make it easier to develop the user interface part of an application program.

6. **Which class is the top of the AWT event hierarchy? (Nov/Dec 2011)**
   Top class of AWT event hierarchy is **java.awt.AWT Event class**.

7. **Which java.util classes and interfaces support event handling? (Nov/Dec 2011)**
   - java.util classes – EventObject class
   - interfaces support – EventListener interface
     so that the both EventObject and EventListener helps to perform event handling in java.

8. **How are frames created in Java? (Nov/Dec 2011)**
   We can create a Frame in two ways:
   1) By extending Frame class
   2) By creating the instance of Frame class

9. **Mention the subclasses of the AWT Event class. (May/Jun 2014)**

ActionEvent	This is an event that indicates a component-defined action occurred, such as clicking a button.
AdjustmentEvent	emitted by adjustable objects
AncestorEvent	It tells a child component that an event occurred with one of the parent components.
ComponentEvent	indicates a component moved, changed size, became visible or invisible, etc. This event is for notification purposes only.
HierarchyEvent	This event indicates some change to the component hierarchy to which a component belongs.
InputMethodEvent	This event fires when text is entered into an appropriate component. Anytime the text changes, for any reason, the input method sends an event.
InternalFrameEvent	It is an AWTEvent that adds support for JInternalFrame objects as the event source. It has the same event types as WindowEvent.
InvocationEvent	This event will execute the run() method on a Runnable when dispatched by the AWT event dispatcher thread.
ItemEvent	This event indicates that an item was selected or deselected. You will see this associated frequently with list boxes.
TextEvent	This event indicates that an object's text changed.

10. **How can you prevent the overwriting of a displayed text in a TextField of a Java program? (Nov/Dec-2013)**

    - Use the JTextArea append() method instead of setText(). Or
    - Can invoke getText(), and pass that String to the setText() method first, then concatenate it with the new text.

11. **How does a radio button in Java differ from a check box? (Nov/Dec-2013)**
    Checkbox:

    A checkbox is an input option that represents a setting or value with an on, off, or mixed-choice. A check mark within the checkbox indicates that the setting is selected or checked.

    Checkboxes in a group are non-exclusive options; more than one checkbox in a group can be checked at any given time.

Radio Buttons:

A radio button is one of a group of controls representing mutually-exclusive choices. A radio button is typically represented as a round control to distinguish from the square checkbox control.

12. Name the Listener methods that must be implemented for the KeyListener interface. (Nov/Dec-2013)

- void keyTyped(KeyEvent e)
- void keyPressed(KeyEvent e)
- void keyReleased(KeyEvent e)

13. Code a Graphics method in Java to draw the String "Hello World" from the coordinates (100,200). (Nov/Dec-2013)

```
import java.applet.Applet;
import java.awt.Graphics;
public class HelloWorldApplet extends Applet
{
 public void paint(Graphics g)
   {
     g.drawString("Hello World", 100,200);
  }
}
```

14. Write note on AWT in JAVA. (Nov/Dec-2013)

AWT is used to generate Graphical User Interface (GUI). GUI offers user interaction via some graphical components. For example, our underlying Operating System also offers GUI via window, frame, Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc. These all are known as components. Using these components, we can create an interactive user interface for an application.

GUI provides result to end-users in response to its raised events. It is entirely based on events. For example, clicking on a button, closing a window, opening a window, typing something in a text area etc. These activities are known as events. GUI makes it easier for the end user to use an application. It also makes them interesting.

15. State the features of Swing. (Nov/Dec-2013)

- Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

16. What is meant by event-driven programming? (Nov/Dec-2013)

An event-driven program is one that largely responds to user events or other similar input. The concept of event-driven programming is an important one in application development and other kinds of programming, and has generate the emergence of event handlers and other resources.

PART-B [15 MARKS]

1. **Describe in detail about the different layout in Java GUI. Which layout is the default one? (Nov/Dec 2018) or**
   What is layout management? State the various types of layout supported by JAVA. Which layout is default one? (Nov/Dec 2013)

Java Layout Managers:

What is a Layout Manager?

A layout manager is an object that controls the size and position (layout) of components inside a Container object. For example, a window is a container that contains components such as buttons and labels. The layout manager in effect for the window determines how the components are sized and positioned inside the window.

A container can contain another container. For example, a window can contain a panel, which is itself a container. As you can see in the figure, the layout manager in effect for the window determines how the two panels are sized and positioned inside the window, and the layout manager in effect for each panel determines how components are sized and positioned inside the panels.

List of Layout Managers

The java.awt package provides the following predefined layout managers that implement the java.awt.LayoutManager interface. Every Abstract Window Toolkit (AWT) and Swing container has a predefined layout manager as its default. It is easy to use the container. setLayout method to change the layout manager, and you can define your own layout manager by implementing the java.awt.LayoutManager interface. This article describes the predefined AWT layout managers in this list.

- **java.awt.BorderLayout**
- **java.awt.FlowLayout**
- **java.awt.CardLayout**
- **java.awt.GridLayout**
- **java.awt.GridBagLayout**

The Java Development Kit (JDK) 1.2 introduces BoxLayout for placing components from left to right or top to bottom in an AWT or Swing container.

BorderLayout

The border layout is the default layout manager for all Window objects. It consists of five fixed areas: North, South, East, West, and Center. You do not have to put a component in every area of the border layout. The demonstration puts the label "Border Layout" in the North area, the OK button in the South area, and the List of fruit trees in the Center area. The East (right) and West (left) areas are empty.

FlowLayout

The flow layout is the default layout manager for all Panel objects and applets. It places the panel components in rows according to the width of the panel and the number and size of the components. The best way to understand flow layout is to resize the demonstration window and notice how the components flow from one row to the other as you make the window wide and narrow.

CardLayout

The card layout lets you use one container (usually a panel) to display one out of many possible component children (like flipping cards on a table). A program can use this layout to show a different child component to different users. For example, the interface shown to an administrator might have additional functionality from the interface shown to a regular user. With card layout, your program can show the appropriate interface depending on the type of user using the program.

GridLayout

The Grid layout arranges components into a grid of rows and columns. You specify the number of rows and columns, the number of rows only and let the layout manager determine the numer of columns, or the number of columns only and let the layout manager determine the number of rows.

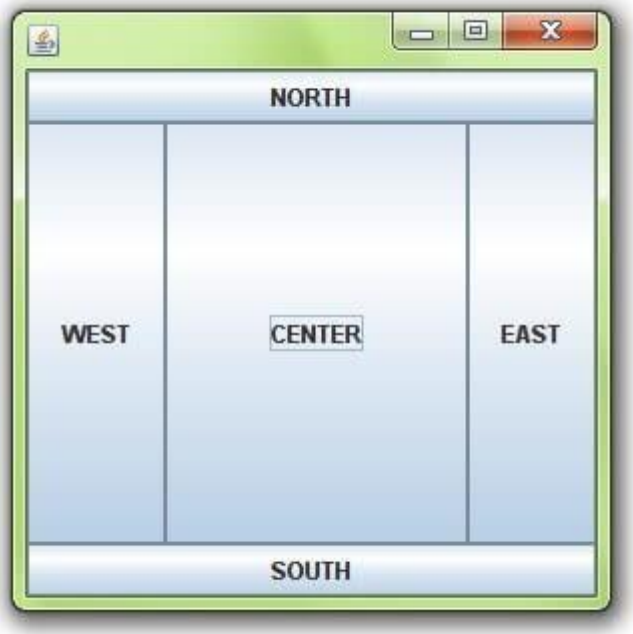
The cells in the grid are equal size based on the largest component in the grid. Resize the window to see how the components are resized to fit cells as they get larger and smaller.

GridBagLayout

The Grid bag layout (like grid layout) arranges components into a grid of rows and columns, but lets you specify a number of settings to fine-tune how the components are sized and positioned within the cells. Unlike the grid layout, the rows and columns are not constrained to be a uniform size. For example, a component can be set to span multiple rows or columns, or you can change its position on the grid.

Example of BorderLayout class:

```
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();
}
}
```



2. i) Give an introduction to swings. (8) (Nov/Dec 2011)

Swing is the Java platform's User Interface -- it acts as the software to handle all the interaction between a user and the computer. It essentially serves as the middleman between the user and the guts of the computer. How exactly does Swing do this? It provides mechanisms to handle the UI aspects described in the previous panel:

- Keyboard: Swing provides a way to capture user input.
- Colors: Swing provides a way to change the colors you see on the screen.
- The address bar you type into: Swing provides text components that handle all the mundane tasks.
- The volume of the music: Well ... Swing's not perfect.

MVC:

Swing even goes a step further and puts a common design pattern on top of the basic UI principles. This design pattern is called Model-View-Controller (MVC) and seeks to "separate the roles." MVC keeps the code responsible for how something looks separate from the code to handle the data separate from the code that reacts to interaction and drives changes.

.

JComponent:

The basic building block of the entire visual component library of Swing is the JComponent. It's the super class of every component. It's an abstract class, so you can't actually create a JComponent, but it contains literally hundreds of functions every component in Swing can use as a result of the

class hierarchy. Obviously, some concepts are more important than others, so for this tutorial, the important things to learn are:

- JComponent is the base class not only for the Swing components but also for custom components as well (more information in the "Intermediate Swing" tutorial).
- It provides the painting infrastructure for all components -- something that comes in handy for custom components (again, there's more info on this subject in "Intermediate Swing").
- It knows how to handle all keyboard presses. Subclasses then only need to listen for specific keys.
- It contains the add() method that lets you add other JComponents. Looking at this another way, you can seemingly add any Swing component to any other Swing component to build nested components (for example, a JPanel containing a JButton, or even weirder combinations such as a JMenu containing a JButton).

ii) What is an adapter class? Explain with an example. (7) (Nov/Dec 2011, 2013)

The java.awt.event package provides adapter classes for various event-listener types. The event listener interfaces that contain more than one method have a corresponding event adapter class that implements the interface and defines each method in the interface with an empty method body. For example, the adapter class for the WindowListener interface is WindowAdapter.

The following table lists some of the listener interfaces and their corresponding adapter classes.

No.	Event Listener Interface	Adapter Classes
1	FocusListener	FocusAdapter
2	KeyListener	KeyAdapter
3	MouseListener	MouseAdapter
4	MouseMotionListener	MouseMotionAdapter
5	WindowListener	WindowAdapter
6	ComponentListener	ComponentAdapter

Now instead of implementing an event listener interface, you can extend the corresponding event adapter class and define only those methods in which you are interested. For example, The following code segment shows that a class MyWindowAdapter extends WindowAdapter and implements the windowClosing() method.

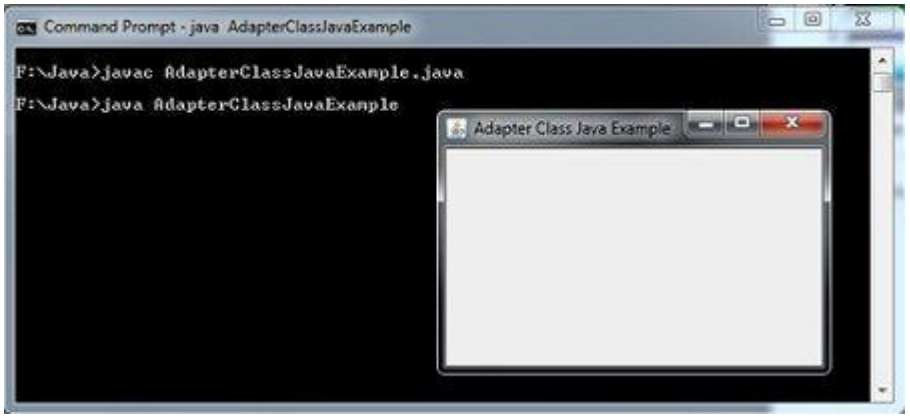
```
class MyWindowAdapter extends WindowAdapter {
  public void windowClosing(WindowEvente){
    System.exit(0);
   }
}
```

The following program demonstrates how the adapter class is created and used.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class AdapterExample extends JFrame
{
```

```
        AdapterExample()
        {
          this.addWindowListener( new WindowAdapter() {
          public void windowClosing(WindowEvent e)
                    {
                      System.exit(0);
                    }
                    }
        }
}
   class AdapterClassJavaExample
{
     public static void main(String [] args)
     {
      AdapterExample frame = new AdapterExample();
      frame.setTitle("Adapter Class Java Example");
      frame.setBounds(100,200,200,200);
      frame.setVisible(true);
     }
}
```

Sample output:



3. **Write a program to simulate the layout and working of a calculator. (May/Jun 2014)**
   Java Program for Calculator Operations Using AWT Controls:

```
import java.awt.*;
import java.awt.event.*;
public class calculator implements ActionListener
{
   int c,n;
   String s1,s2,s3,s4,s5;
   Frame f;
   Button b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15,b16,b17;
   Panel p;
   TextField tf;
   GridLayout g;
   calculator()
```

```
{
   f = new Frame("My calculator");
   p = new Panel();
   f.setLayout(new FlowLayout());
   b1 = new Button("0");
   b1.addActionListener(this);
   b2 = new Button("1");
   b2.addActionListener(this);
   b3 = new Button("2");
   b3.addActionListener(this);
   b4 = new Button("3");
   b4.addActionListener(this);
   b5 = new Button("4");
   b5.addActionListener(this);
   b6 = new Button("5");
   b6.addActionListener(this);
   b7 = new Button("6");
   b7.addActionListener(this);
   b8 = new Button("7");
   b8.addActionListener(this);
   b9 = new Button("8");
   b9.addActionListener(this);
   b10 = new Button("9");
   b10.addActionListener(this);
   b11 = new Button("+");
   b11.addActionListener(this);
   b12 = new Button("-");
   b12.addActionListener(this);
   b13 = new Button("*");
   b13.addActionListener(this);
   b14 = new Button("/");
   b14.addActionListener(this);
   b15 = new Button("%");
   b15.addActionListener(this);
   b16 = new Button("=");
   b16.addActionListener(this);
   b17 = new Button("C");
   b17.addActionListener(this);
   tf = new TextField(20);
   f.add(tf);
   g = new GridLayout(4,4,10,20);
   p.setLayout(g);
   p.add(b1);p.add(b2);p.add(b3);p.add(b4);p.add(b5);p.add(b6);p.add(b7);p.add(b8);
   p.add(b9);p.add(b10);p.add(b11);p.add(b12);p.add(b13);p.add(b14);p.add(b15);
   p.add(b16);p.add(b17);
   f.add(p);
   f.setSize(300,300);
   f.setVisible(true);
}

public void actionPerformed(ActionEvent e)
{
   if(e.getSource()==b1)
   {
```

```
    s3 = tf.getText();
    s4 = "0";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b2)
{
    s3 = tf.getText();
    s4 = "1";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b3)
{
    s3 = tf.getText();
    s4 = "2";
    s5 = s3+s4;
    tf.setText(s5);
}if(e.getSource()==b4)
{
    s3 = tf.getText();
    s4 = "3";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b5)
{
    s3 = tf.getText();
    s4 = "4";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b6)
{
    s3 = tf.getText();
    s4 = "5";
    s5 = s3+s4;
    tf.setText(s5);
}

if(e.getSource()==b7)
{
    s3 = tf.getText();
    s4 = "6";
    s5 = s3+s4;
    tf.setText(s5);
}
if(e.getSource()==b8)
{
    s3 = tf.getText();
    s4 = "7";
    s5 = s3+s4;
    tf.setText(s5);
}
```

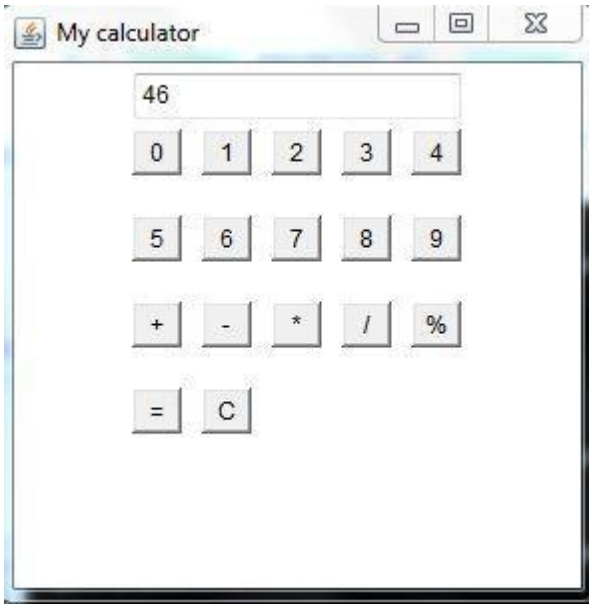
```
if(e.getSource()==b9)
{
   s3 = tf.getText();
   s4 = "8";
   s5 = s3+s4;
   tf.setText(s5);
}
if(e.getSource()==b10)
{
   s3 = tf.getText();
   s4 = "9";
   s5 = s3+s4;
   tf.setText(s5);
}
if(e.getSource()==b11)
{
   s1 = tf.getText();
   tf.setText("");
   c=1;
 }
if(e.getSource()==b12)
{
   s1 = tf.getText();
   tf.setText("");
   c=2;
 }
if(e.getSource()==b13)
{
   s1 = tf.getText();
   tf.setText("");
   c=3;
 }
if(e.getSource()==b14)
{
   s1 = tf.getText();
   tf.setText("");
   c=4;
 }

if(e.getSource()==b15)
{
   s1 = tf.getText();
   tf.setText("");
   c=5;
 }
if(e.getSource()==b16)
{
   s2 = tf.getText();
   if(c==1)
   {
      n = Integer.parseInt(s1)+Integer.parseInt(s2);
      tf.setText(String.valueOf(n));
   }
   else
```

```
        if(c==2)
        {
           n = Integer.parseInt(s1)-Integer.parseInt(s2);
           tf.setText(String.valueOf(n));
        }
        else
        if(c==3)
        {
           n = Integer.parseInt(s1)*Integer.parseInt(s2);
           tf.setText(String.valueOf(n));
        }
        if(c==4)
        {
           try
           {
              int p=Integer.parseInt(s2);
              if(p!=0)
              {
                              n = Integer.parseInt(s1)/Integer.parseInt(s2);
              tf.setText(String.valueOf(n));
               }
               else
                  tf.setText("infinite");

           }
           catch(Exception i){}
        }
        if(c==5)
        {
           n = Integer.parseInt(s1)%Integer.parseInt(s2);
           tf.setText(String.valueOf(n));
        }
     }
     if(e.getSource()==b17)
     {
        tf.setText("");
     }
  }

  public static void main(String[] abc)
  {
     calculator v = new calculator();
  }
}
```

OUTPUT:

1. Compile: javac calculator.java

2. Run: java calculator



4. **Write a simple program to demonstrate the mouse events. (Nov/Dec 2013)**

The events of Java AWT can be divided into two broad categories – **Semantic events** and **Low-level events**. Semantic events are generated by software (GUI) components like frame, button etc and low-level events are generated by hardware components like mouse, keyboard etc.

The low-level events has category three programs are given on MouseListener, MouseMotionListenerand KeyListener. Now let us go with MouseListener.

Mouse events can be handled with MouseListener and MouseMotionListener. MouseListener handles the events when the **mouse is stable** and MouseMotionListener handles the events while the **mouse is in motion**.

When the mouse is stable, the mouse generates five types of actions represented by five abstract methods of MouseListener. The five actions are:

Action name MouseListener abstract method

MOUSE KEY IS PRESSED	MOUSEPRESSED()
Mouse key is released	mouseReleased()
Mouse key is clicked	mouseClicked()
Mouse is entering a frame	mouseEntered()
Mouse is exiting a frame	mouseExited()

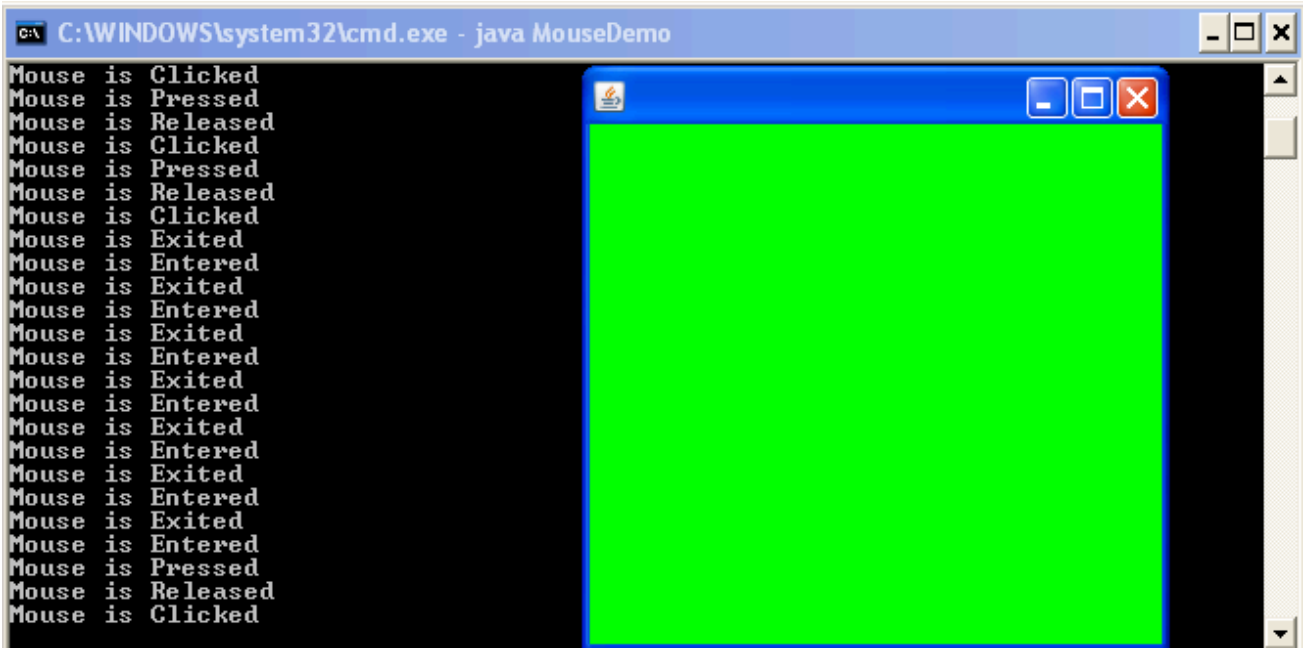
All the above five methods take **MouseEvent** as parameter. When the mouse is entering or exiting a notepad file, a MS Word file or an Excel file, the mouse arrow changes a **double-arrow** symbol. These actions are represented by two methods – **mouseEntered()** and **mouseExited()**.

The event generated by the Mouse is known as **MouseEvent**.

Following program illustrates the way of using the MouseListener with simple event-handling. For a bigger program refer MouseMotionListener. MouseListener Java Example changing the background of the frame as per the action using MouseListener

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

public class MouseDemo extends Frame implements MouseListener
{
public MouseDemo( )
{
addMouseListener(this);                // link the frame with ML
setSize(300,300);
setVisible(true);
}
                        // override the 5 abstract methods of ML
public void mousePressed(MouseEvent e)
{
              // do some action to distinguish from other methods
 setBackground(Color.red);
 System.out.println("Mouse is Pressed");
 }
 public void mouseReleased(MouseEvent e)
 {
 setBackground(Color.blue);
 System.out.println("Mouse is Released");
 }
 public void mouseClicked(MouseEvent e)
 {
  setBackground(Color.green);
  System.out.println("Mouse is Clicked");
```

```
 }
public void mouseEntered(MouseEvent e)
{
 setBackground(Color.cyan);
 System.out.println("Mouse is Entered");
 }
 public void mouseExited(MouseEvent e)
 {
  setBackground(Color.magenta);
  System.out.println("Mouse is Exited");
  }
  public static void main(String args[])
  {
    new MouseDemo();
  }
}
```



Output screen on MouseListener Java Example

---

5. How will you display an image on the frame in a window using Java. (May/Jun 2013)

A Frame is a top-level window with a title and a border. Frames are capable of generating the following types of window events: WindowOpened, WindowClosing, WindowClosed, WindowIconified, WindowDeiconified, WindowActivated, WindowDeactivated.

Frame Constructor

Frame() Constructs a new instance of Frame that is initially invisible.

Frame(String) Constructs a new, initially invisible Frame object with the specified title.

Some of the commonly used methods of Frame class are as follows.

Methods	Description
String getTitle()	Gets the title of the frame.
void setBackground(Color bgColor)	Sets the background color of this window.
void setResizable (boolean resizable)	Sets whether this frame is resizable by the user.
void setShape (Shape shape)	Sets the shape of the window.
void setTitle (String title)	Sets the title for this frame to the specified string.
void setSize (Dimension d)	Resizes this component so that it has width d. width and height d.height.
void setVisible(boolean b)	Shows or hides this Window depending on the value of parameter b.
public void show()	Makes the Window visible
void setMenuBar (MenuBar) mb)	Sets the menu bar for this frame to the specified menu bar

Creating a Frame:

There are two ways to create a Frame. They are,

By Instantiating Frame class
By extending Frame class

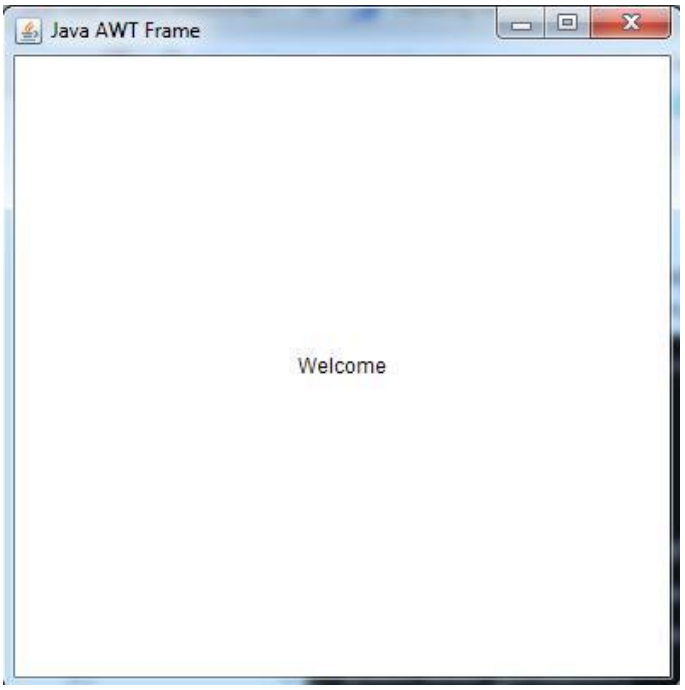
We can generate a window by creating an instance of Frame. The created frame can be made visible by calling setVisible( ). When created, the window is given a default height and width. The size of the window can be changed explicitly by calling the setSize( ) method.A label can be added to the current frame by creating an Label instance and calling the add()method.

Example:

```
import java.awt.*;

 public class AwtFrame

{

 public static void main(String[] args){

Frame frm = new Frame("Java AWT Frame");

Label lbl = new Label("Welcome",Label.CENTER);
```

```
frm.add(lbl);
 frm.setSize(400,400);
 frm.setVisible(true);
}}
```

Sample Output:



Creating an Frame Window in an Applet

The steps to be followed to create a child frame within an applet are as follows.

- Create a subclass of Frame
- Override any of the standard window methods, such as init(),start(),stop(),and paint().
- Implement the windowClosing() method of the windowListener interface,calling setVisible(false) when the window is closed
- Once you have defined a Frame subclass, you can create an object of that class. But it will note be initially visible
- When created, the window is given a default height and width
- You can set the size of the window explicitly by calling the setSize() method

Example:AppletFrame.java

```
// Create a child frame window from within an applet.
```

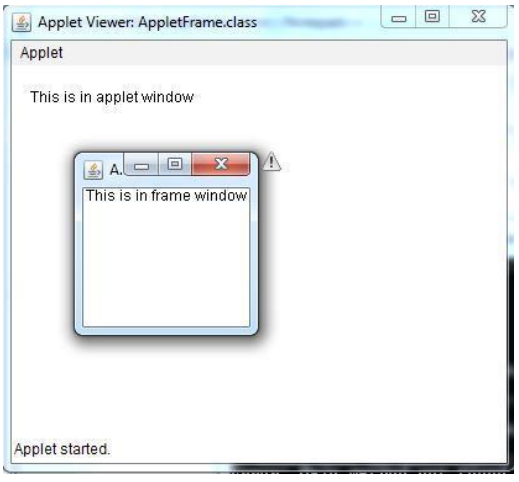
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
// Create a subclass of Frame.
class SampleFrame extends Frame {
SampleFrame(String title) {
super(title);
//create an object to handle window events
MyWindowAdapter adapter = new MyWindowAdapter(this);
//register it to receive those events
addWindowListener(adapter);
}
public void paint(Graphics g) {
g.drawString("This is in frame window", 10, 40);}
}
class MyWindowAdapter extends WindowAdapter {
SampleFrame sampleFrame;
public MyWindowAdapter(SampleFrame  sampleFrame) {
this.sampleFrame = sampleFrame;}
public void windowClosing(WindowEvent we) {
 sampleFrame.setVisible(false);}
}
//Create frame window.
public class AppletFrame extends Applet {
Frame f;
//init(), start(), paint(), and stop() methods are called automatically in the specified sequence.
public void init() {
```

```
f = new SampleFrame("A Frame Window");

f.setSize(150,150);

f.setVisible(true);

}

public void start() {

f.setVisible(true);                    // make the window visible

}

public void stop() {

f.setVisible(false); // hide the window

}

public void paint(Graphics g) {

g.drawString("This is in applet window", 15, 30); // Display the given text in the
win-dow

}

}
```

Test1.html

```
<html>

<body>

<applet code="AppletFrame.class" width="400" height="300">

</applet>

</body>

</html>
```

Sample output:



6. With an example describe in detail about how to work with 2D shapes in java. (May-12, 15, Dec-12)

Java supports 2-dimensional shapes, text and images using methods available in Graphics2D class. The Graphics2D class extends the Graphics class to provide more sophisticated control over geometry, coordinate transformations, color management, and text layout. Some of the commonly used methods of Graphics2D class are as follows

Method	Description
void draw(Shape s)	Strokes the outline of a Shape using the settings of the current Graphics2D context
void draw3DRect(int x, int y, int width, int height, boolean raised)	Draws a 3-D highlighted outline of the specified rectangle.
void drawImage(BufferedImage img, BufferedImageOp op, int x, int y)	Renders a BufferedImage that is filtered with a BufferedImageOp.
boolean drawImage(Image img, Affine Transform xform, ImageObserver obs)	Renders an image, applying a transform from image space into user space before drawing.
void drawString(String str, float x, float y)	Renders the text specified by the specified String, using the current text attribute state in the Graphics2D context
void fill(Shape s)	Fills the interior of a Shape using the settings of the Graphics2D context.
void rotate(double theta)	oncatenates the current Graphics2D Transform with a rotation transform.
void scale(double sx, double sy)	oncatenates the current Graphics2D Transform with a scaling transformation Subsequent rendering is resized according to the specified scaling factors relative to the

	previous scaling.
void setBackground(Color color)	Sets the background color for the Graphics2D context.
void setPaint(Paint paint)	Sets the Paint attribute for the Graphics2D context.
void setStroke(Stroke s)	Sets the Stroke for the Graphics2D context.
void shear(double shx, double shy)	Concatenates the current Graphics2D Transform with a shearing transform.
void transform(AffineTransform Tx)	Composes an AffineTransform object with the Transform in this Graphics2D according to the rulelast-specified-first-applied.

Graphics2D class Constructor

Graphics2D() //Constructs a new Graphics2D object.

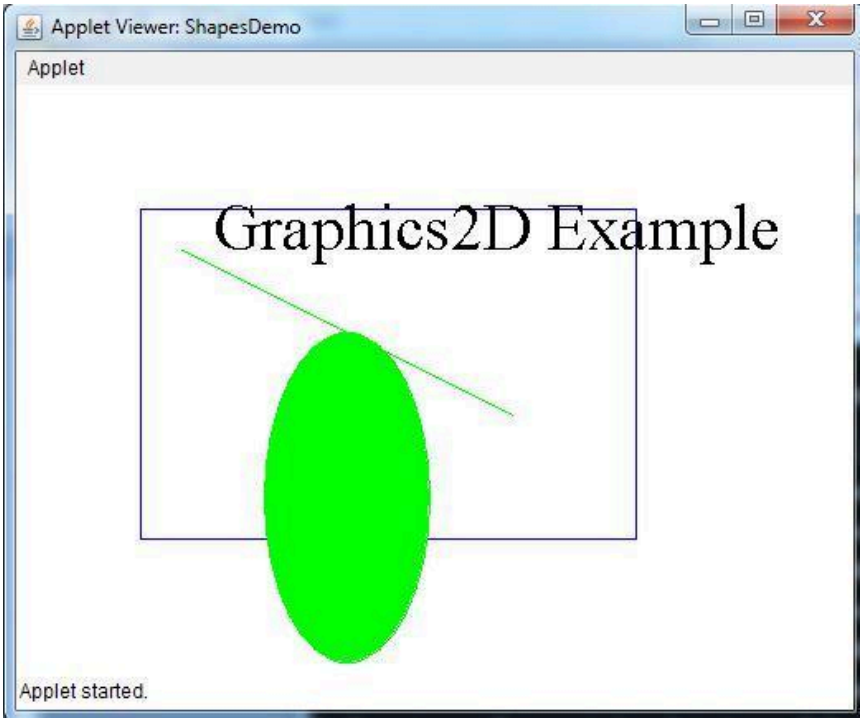
This class inherits the methods from java.lang.Object.

Example:

```
import java.awt.*;
import java.applet.*;
/*
<applet code=”ShapesDemo” width=350 height=300>
</applet>
*/
public class ShapesDemo extends Applet {
public void init() {}
public void paint(Graphics g) {
Graphics2D g2d = (Graphics2D)g;
```

```
g2d.setColor(Color.blue);

g2d.drawRect(75,75,300,200);

Font exFont = new Font("TimesRoman",Font.PLAIN,40);

g2d.setFont(exFont);

g2d.setColor(Color.black);

g2d.drawString("Graphics2D Example",120.0f,100.0f);

g2d.setColor(Color.green);

g2d.drawLine(100,100,300,200);

g2d.drawOval(150,150,100,200);

g2d.fillOval(150,150,100,200);

}}
```

Sample Output:



Colors in Java

To support different colors Java package comes with the Color class. The Color class states colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.

Color class constructor: Color(float r, float g, float b) – create color with specified red, green, and blue values in the range (0.0 - 1.0)

Color(int r, int g, int b)- create color with the specified red, green, and blue values in the range (0 - 255).

Fonts in Java

The Font class states fonts, which are used to render text in a visible way.

Font class constructor:

Font(Font font) //Creates a new Font from the specified font.

Font(String name, int style, int size) //Creates a new Font from the specified name, style and point size.

Images in Java

Image control is superclass for all image classes representing graphical images.

Image class constructor:

Image() // create an Image object

The java.applet.Applet class provides following methods to access image.

- getImage() method that returns the object of Image. Its syntax is as follows. public Image getImage(URL u, String image){}
- getDocumentBase() method returns the URL of the document in which applet is embedded. public URL getDocumentBase(){}
- URL getCodeBase() method returns the base URL. public URL getCodeBase()