UNIT V EVENT DRIVEN PROGRAMMING

Graphics programming - Frame – Components - working with 2D shapes - Using color, fonts,and images - Basics of event handling - event handlers - adapter classes - actions – mouse events - AWT event hierarchy - Introduction to Swing – layout management - Swing Components – Text Fields , Text Areas – Buttons- Check Boxes – Radio Buttons – Lists- choices- Scrollbars – Windows –Menus – Dialog Boxes.

5.1 GRAPHICS PROGRAMMING

- A ***graphics context*** is encapsulated by the Graphics class
- **java.awt.geom** allows you to work with geometric shapes.
- An application or software is created in two ways
  1. Using CUI(Character User Interface)
  2. Using GUI(Graphical User Interface)

CUI(Character User Interface)

- The users has to use characters or commands to interact with an application.
- The main disadvantage is that the user has to remember several commands and their use with correct syntax.

GUI(Graphical User Interface)

- User can interact with any application by clicking on some images or graphics.

Advantages:

- User friendly.
- Adds attraction to any application by adding pictures, colors, menus, etc.
- Possible to simulate real time applications like calculator etc.
- Helps to create graphical components like button, check boxes, etc.

Graphics Class

The Graphics class is the abstract super class for all graphics contexts which allow an application to draw onto components that can be realized on various devices, or onto off-screen images as well.

Methods in the Graphics Class

- *void drawLine(int x0 int y0, int x1 int y1)* – To draw a line between the points (x_0, y_0) and (x1, y1).
- *void drawRect(int x, int y, int w, int h)* – To draw a rectangle with upper-left corner at the coordinates x and y, width w and height h.
- *void drawstring(String str, int x, int y)* – To drae the string str at the location (x,y).
- *void drawOval(int x, int y,int w, int h)* – To draw an Oval in such aa way that the center of the oval will be the centre of a rectangle with upper-left corner at coordinates x and y, width w and height h.
- *void drawPolygon(int x[], int y[]. Int n)* – To draw a polygon with n vertices. The coordinates are given by the element of the array x and y. The first and last points of the polygon are automatically connected.
- *void drawArc(int x, int y, int w, int h, int degree0, int degree1)* - To draw an arc starting from degrees0 and of the length corresponding to degrees1. The centre of the arc will be

same as the centre of a rectangle with upper-left corner at coordinates x and y, width w, and height h.

- *void fillRect(int x, int y, int w, int h)* – To fill a rectangle with upper-left corner at the coordinates x and y, width w and height h.
- *void fillOval(int x, int y,int w, int h)* – To fill an Oval. The center of the oval will be the same as the centre of a rectangle with upper-left corner at coordinates x and y, width w and height h.
- *void fillArc(int x, int y, int w, int h, int degree0, int degree1)* - To fill an arc starting from degrees0 and of the length corresponding to degrees1. The centre of the arc will be same as the centre of a rectangle with upper-left corner at coordinates x and y, width w, and height h.
- *void setColor(Color c)* - Sets this graphics context's current color to the specified color.

5.2 FRAME

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

The frame in java works like the main window where your components (controls) are added to develop an application. In the Java AWT, top-level windows are represented by the **Frame** class. Java supports the look and feel and decoration for the frame. For creating java standalone application you must provide GUI to the user.

The most common method of creating a frame is by using single argument constructor of the **Frame** class that contains the single string argument which is the title of the window or frame. Then you can add user interface by constructing and adding different components to the container one by one.

In this program we are constructing a label to display "Welcome to Java Tutorial." message on the frame. The center alignment of the label has been defined by the **Label.**CENTER. The frame initially invisible, so after creating the frame it need to visualize the frame by setVisible(true) method.

add(lbl):
This method has been used to add the label to the frame. Method add() adds a component to it's container.

setSize (width, height):
This is the method of the Frame class that sets the size of the frame or window. This method takes two arguments width (int), height (int).

setVisible(boolean):
This is also a method of the Frame class sets the visibility of the frame. The frame will be invisible if you pass the boolean value *false* otherwise frame will be visible.

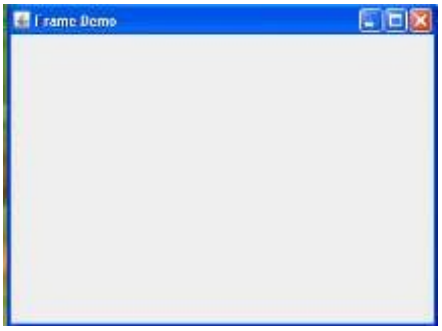
Example :

```
import java.awt.*;
public class AwtFrame{
public static void main(String[] args){
Frame frm = new Frame("Java AWT Frame");
Label lbl = new Label("Welcome to Java Tutorial. ",Label.CENTER);
frm.add(lbl);
frm.setSize(400,400);
frm.setVisible(true);
  }}
```

5.2.1 Creating A Frame

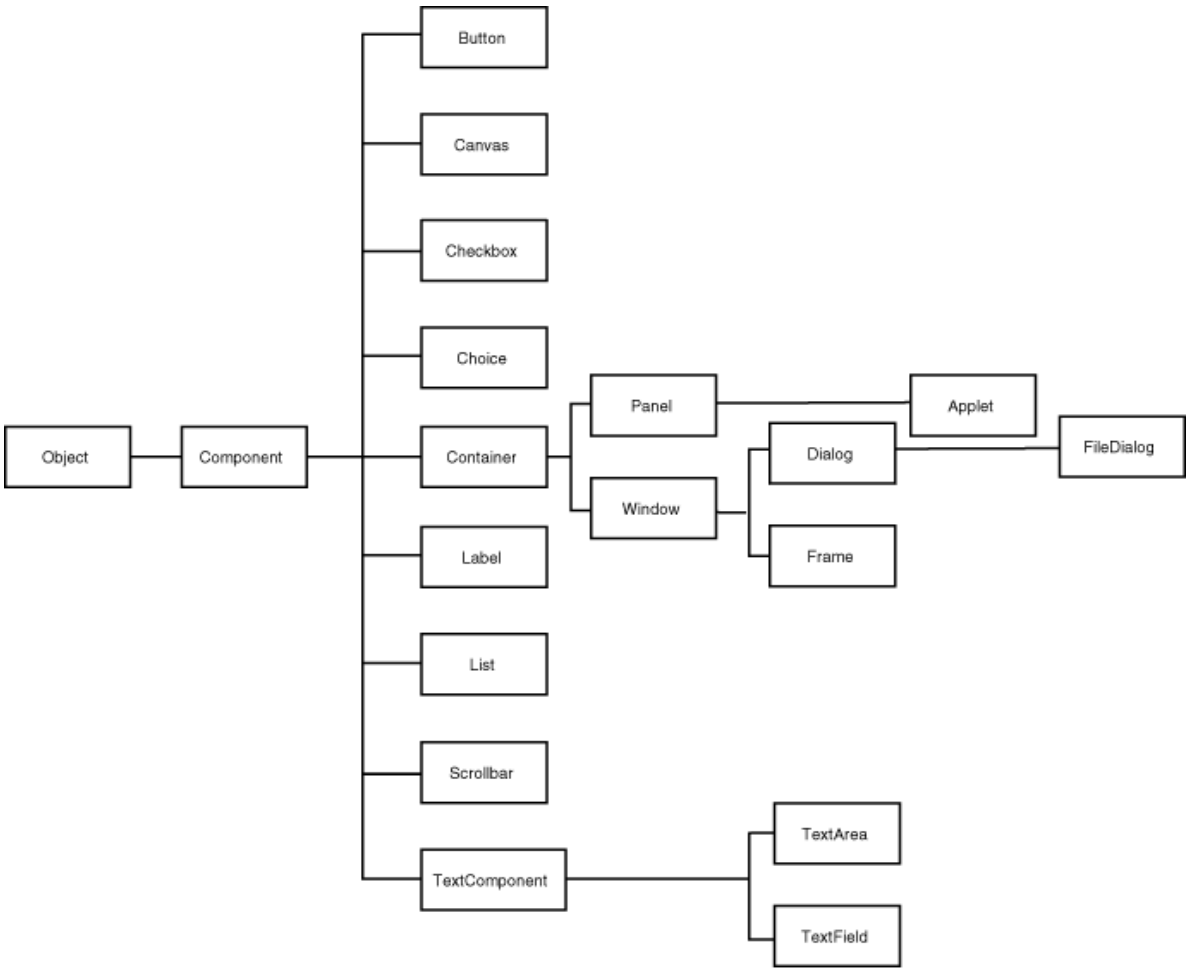
- A frame forms the basic components in AWT.
- It has to be created before any other components so that all other component can be displayed in a frame.

PROGRAM




```
import javax.swing.*;
import java.awt.*;
class FrameDemo extends JFrame
{
    public static void main(String args[])
    {
 JFrame f=new JFrame();
 f.setVisible(true);
 f.setTitle("Frame Demo");
 f.setSize(400,300);
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

5.3 COMPONENTS



5.4 WORKING WITH 2DSHAPES

- Graphics2D class is a subclass of the Graphics class available in java.awt.geom package.
- Methods such as paint Component automatically receive an object of the Graphics2D class and simply use a cast, as follows:

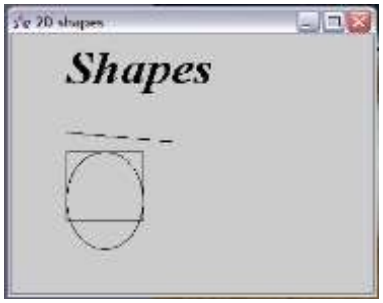
```
public void paintComponent(Graphics g)
{
Graphics2D g2 = (Graphics2D) g;
. . .
}
```

- The Java 2D library organizes geometric shapes in an object-oriented fashion such as classes to represent lines, rectangles, and ellipses.
    - line2D : Line2D.Double(double x, double y, double w, double h)
    - Rectangle2D : Rectangle2D.Double(double x, double y, double w, double h)
    - Ellipse2D : Ellipse2D.Double(double x, double y, double w, double h)
- The RectangularShape class defines over 20 methods that are common to these shapes, among them such useful methods as
    - getWidth
    - getHeight
    - getCenterX
    - getCenterY.
- Rectangle2D and Ellipse2D objects are simple to construct. You need to specify
    - The x- and y-coordinates of the top-left corner; and
    - The width and height.

PROGRAM

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;
class TestFrame
{
public static void main(String args[])throws Exception
{
 FrameDemo f=new FrameDemo();
 f.setTitle("2D shapes");
 f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
class FrameDemo extends JFrame
{
 FrameDemo()
{
 Container c = getContentPane();
 PanelDemo p = new PanelDemo();
 c.add(p);
 setVisible(true);
```

```
 setSize(400,300);
}
}

class PanelDemo extends JPanel
{
    public void paintComponent(Graphics g)
    {
 super.paintComponent(g);
 Font f=new Font("TimesRoman",Font.ITALIC+Font.BOLD,45);
 g.setFont(f);
 g.drawString("Shapes",50,50);
 Graphics2D g2=(Graphics2D)g;
 Line2D line=new Line2D.Double(50,100,150,110);
 g2.draw(line);
 Rectangle2D rect=new Rectangle2D.Double(50,120,70,70);
 g2.draw(rect);
 Ellipse2D ellipse=new Ellipse2D.Double(50,120,70,100);
 g2.draw(ellipse);
    }
}
```



5.5 USING COLORS, FONTS, IMAGES

5.5.1 DISPLAY THE AVAILABLE FONTS

- To set some font to the text,we can use setFont method of Graphics class.
  g.setFont(Font object);
- This method takes Font class objects, which can be created as,
  Font f=new Font("Arial",Font.BOLD,30);
- To know which fonts are available on machine, **getAvailableFontFamilyNames( )** method defined by the GraphicsEnvironment class.

Example

```
//To display the available fonts in the machine
import java.awt.*;
import java.awt.GraphicsEnvironment;
class Fontlist
{
 public static void main(String args[])
 {
GraphicsEnvironment ge =
GraphicsEnvironment.getLocalGraphicsEnvironment();
  Font font[]=ge.getAllFonts();
  for(Font f: font)   //enhanced for loop
   System.out.println(f);
 }
}
```

5.5.2 COLORS

The java.awt.Color class offers predefined constants for the following 13 standard colors: black, blue, cyan, darkgray, gray, green, lightgray, magenta,orange,pink,red,white, yellow
The **setPaint** method of the Graphics2D class lets you select a color that is used for all subsequent
drawing operations on the graphics context.

```
g2.setPaint(Color.RED);
g2.drawString("color!", 100, 100);
g2.fill(rect);    //fill the interiors of closed shapes with a color.
```

5.5.3 IMAGE CLASS

- Images are objects of the Image class, which is part of the java.awt package.
- Images are manipulated using the classes found in the java.awt.image package.
  1. Load the image into Image class object using **getImage()** method of Toolkit class.
     Image img=Toolkit.getDefaultToolkit().getImage("rose.jpg");
  2. Once it is loaded and available in 'img',display the image using drawImage() method of Graphics class.

EXAMPLE

```
import java.awt.*;
import java.awt.geom.*;
import java.awt.image.*;
import javax.imageio.*;
```

```
import java.awt.image.*;
import javax.swing.*;
class ImageTest
{
 public static void main(String args[])throws Exception
 {
  ImageFrame f=new ImageFrame();
 }
}
class ImageFrame extends JFrame
{
 ImageFrame()
 {
  Container c=getContentPane();
  setSize(400,300);
  setVisible(true);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  DPanel p=new DPanel();
  p.setBackground(Color.white);
  c.add(p);
 }
}
class DPanel extends JPanel
{
 public void paintComponent(Graphics g)
 {
  super.paintComponent(g);
  Font f=new Font("TimesRoman",Font.ITALIC+Font.BOLD,20);
  Graphics2D g1=(Graphics2D)g;
  Ellipse2D el=new Ellipse2D.Double(50,120,70,50);
  g1.draw(el);
  g1.setPaint(Color.BLUE);
  g1.fill(el);
  g1.setPaint(Color.RED);
  g1.setFont(f);
  g1.drawString("Using Fonts,Colors,Images",30,30);
  Image img=Toolkit.getDefaultToolkit().getImage("rose.jpg");
  g1.drawImage(img,130,60,null);
 }
}
```

5.6 EVENT HANDLING

- Any program that uses a graphical user interface, such as a Java application written for Windows, is event driven.
- A program has to be broken up into separate pieces to do event-drive programming.
- In Java, all event objects ultimately derive from the class java.util.EventObject.
- Events are supported by a number of packages, including java.util, java.awt, and java.awt.event.

5.6.1 COMPONENTS OF EVENT HANDLING

- An **event** is an object that describes a state change in a source.
- A **source** is an object that generates an event.
- A **listener** is an object that is notified when an event occurs.

Relationship between event sources and listeners

EVENT CLASS	DESCRIPTION	LISTENER INTERFACE
ActionEvent	Generated when a button is pressed, a list item is double clicked, or a menu item is selected.	ActionListener
AdjustmentEvent	Generated when a scroll bar is manipulated.	AdjustmentListener
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible.	ComponentEventListener
ContainerEvent	Generated when a component is added to or removed from a container.	ContainerListener
FocusEvent	Generated when a component gains or loses keyboard focus.	FocusListener
InputEvent	Abstract superclass for all component input event classes. MouseEvent	InputListener
ItemEvent	Generated when a check box or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected.	ItemListener
KeyEvent	Generated when input is received from the keyboard.	KeyListener
MouseWheelEvent	Generated when the mouse wheel is moved.	MouseWheelListener
TextEvent	Generated when the value of a text area or text field is changed.	TextListener
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.	WindowListener

5.6.2 WINDOW EVENT

WindowEvent is a subclass of ComponentEvent. It defines several constructors.

WindowEvent(Window *src*, int *type*)

WINDOW_ACTIVATED	The window was activated.
WINDOW_CLOSED	The window has been closed.
WINDOW_CLOSING	The user requested that the window be closed.
WINDOW_DEACTIVATED	The window was deactivated.
WINDOW_DEICONIFIED	The window was deiconified.
WINDOW_GAINED_FOCUS	The window gained input focus.
WINDOW_ICONIFIED	The window was iconified.
WINDOW_LOST_FOCUS	The window lost input focus.
WINDOW_OPENED	The window was opened.
WINDOW_STATE_CHANGED	The state of the window changed.

PROGRAM

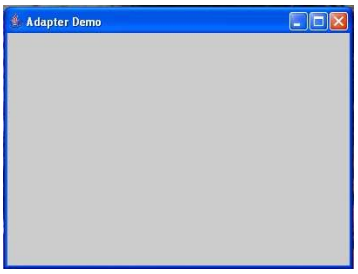
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class Window implements WindowListener
{
    public static void main(String args[])
    {
 JFrame f=new JFrame();
 f.setVisible(true);
 f.setSize(400,300);
 f.setTitle("Window Event");
 f.addWindowListener(new Window());
    }

   public void windowActivated(WindowEvent we){}
   public void windowClosed(WindowEvent we){}
   public void windowClosing(WindowEvent we){System.exit(0);}
   public void windowDeactivated(WindowEvent we) {}
   public void windowDeiconified(WindowEvent we){}
   public void windowIconified(WindowEvent we){}
   public void windowOpened(WindowEvent we){}
}
```

5.7 ADAPTER CLASS

- Java provides a special feature, called an **adapter class**, that can simplify the creation of event handlers in certain situations.
- An adapter class provides an **empty implementation of all methods** in an event listener interface.
- A new class can be defined to act as an event listener by extending one of the adapter classes and implementing only those events in which you are interested.
- The MouseMotionAdapter class has two methods, **mouseDragged()** and **mouseMoved()**, which are the methods defined by the MouseMotionListener interface.
- If only mouse drag events, simply extend MouseMotionAdapter and override **mouseDragged( ).**
- The empty implementation of **mouseMoved( )** would handle the mouse motion events.

PROGRAM

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class AdaptDemo extends WindowAdapter
{
     public static void main(String args[])
     {
 JFrame f=new JFrame();
 f.setVisible(true);
 f.setSize(400,300);
 f.setTitle("Adapter Demo");
 f.addWindowListener(new Window());
     }
     public void windowClosing(WindowEvent we)
     {
System.exit(0);
     }
}
```



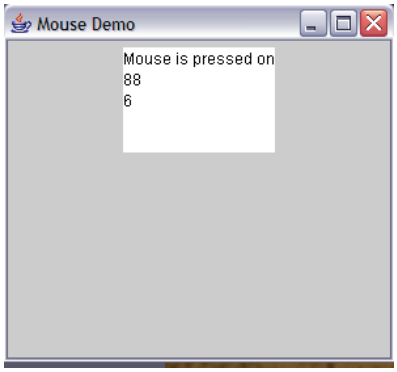
5.7 MOUSE EVENT

- MouseEvent is a subclass of InputEvent.
- When the user clicks a mouse button, three listener methods are called: mousePressed when the mouse is first pressed, mouseReleased when the mouse is released, and, finally, mouse-Clicked.
- To distinguishbetween single, double, and triple (!) clicks, use the getClickCount method.
- There are eight types of mouse events.

MOUSE_CLICKED	**MOUSE_CLICKED**
MOUSE_DRAGGED	**MOUSE_DRAGGED**
MOUSE_ENTERED	**MOUSE_ENTERED**
MOUSE_EXITED	**MOUSE_EXITED**
MOUSE_MOVED	**MOUSE_MOVED**
MOUSE_PRESSED	**MOUSE_PRESSED**
MOUSE_RELEASED	**MOUSE_RELEASED**
MOUSE_WHEEL	**MOUSE_WHEEL**

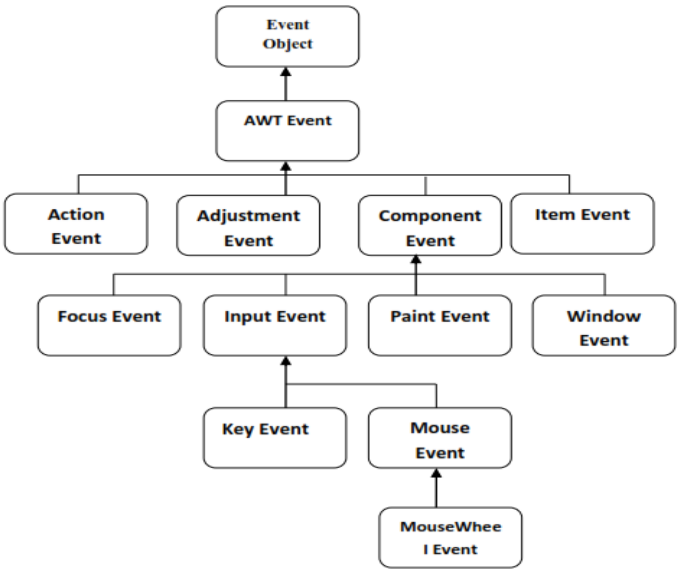
PROGRAM

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class MouseDemo extends JFrame implements MouseListener,MouseMotionListener
{
 JTextArea ta;
 String s;
 int x,y;
MouseDemo()
 {
  Container c=getContentPane();
  c.setLayout(new FlowLayout());
  ta=new JTextArea(5,10);
  c.add(ta);
  ta.addMouseListener(this);
  ta.addMouseMotionListener(this);
  setVisible(true);
  setSize(300,400);
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setTitle("Mouse Demo");
 }
 public static void main(String args[])
 {
  MouseDemo m=new MouseDemo();
 }
 public void mouseEntered(MouseEvent me)

{
  s+="Mouse Entered";
  this.display();
 }
          public void mouseExited(MouseEvent me)
 {
  s+="Mouse exited";
  this.display();
 }
 public void mouseClicked(MouseEvent me)
 {
  if(me.getButton()==1)
   s+="Left Button";
  else if(me.getButton()==2)
   s+="Middle Button";
  else if(me.getButton()==3)
   s+="Right Button";
  this.display();
 }
 public void mousePressed(MouseEvent   me)
 {
  s+="Mouse is pressed on\n"+x+"\n"+y;
  this.display();
```



```
 }
 public void mouseDragged(MouseEvent me)
 {
  s+="Mouse Dragged";
  this.display();
 }
 public void mouseMoved(MouseEvent me)
 {
  s+="Mouse Moved";
  this.display();
 }
          public void mouseReleased(MouseEvent me)
 {
   x=me.getX();
   y=me.getY();
  s+="Mouse isreleased on\n"+x+"\n"+y;
  this.display();
 }
 public void display()
 {
  ta.setText(s);
  s="";
 }
}
```

5.8 AWT EVENT HIERARCHY

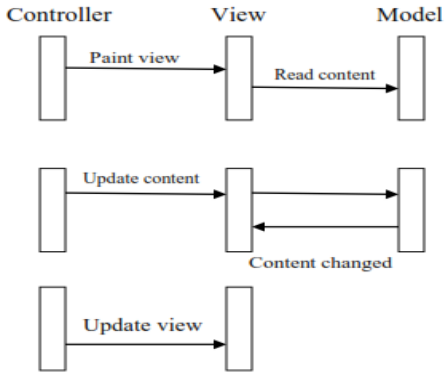


5.9 A MODEL-VIEW-CONTROLLER DESIGN PATTERN

- Every component has three characteristics:
  - Its content, such as the state of a button (pushed in or not), or the text in a text field
  - Its visual appearance (color, size, and so on)
  - Its behavior (reaction to events)
- The model-view-controller (MVC) design pattern teaches how to accomplish this.

- Implement three separate classes:
  - The model, which stores the content
  - The view, which displays the content
  - The controller, which handles user input.
- The model corresponds to the state information associated with the component.
- For example, in the case of a check box,
  1. The model contains a field that indicates if the box is checked or unchecked.
  2. The view determines how the component is displayed on the screen, including any aspects of the view that are affected by the current state of the model.
  3. The controller determines how the component reacts to the user.
- By separating a component into a model, a view, and a controller, the specific implementation of each can be changed without affecting the other two.

Communication between the model, view, controller



SWING BUTTONS

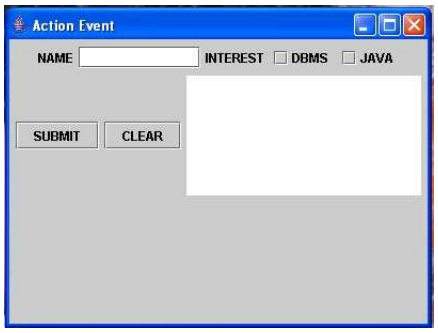
Property Name	Value
actionCommand	The action command string associated with this button
mnemonic	The keyboard mnemonic for this button
armed	true if the button was pressed and the mouse is still over the button
enabled	true if the button is selectable
pressed	true if the button was pressed but the mouse button hasn't yet been released
rollover	true if the mouse is over the button
selected	true if the button has been toggled on (used for checkboxes and radio buttons)

5.10 LAYOUT MANAGEMENT

FLOW LAYOUT

- The buttons are contained in a JPanel object and are managed by the flow layout manager, the default layout manager for a panel.
- Components are placed inside containers, and a layout manager determines the positions and sizes of the components in the container.

- Buttons, text fields, and other user interface elements extend the class Component.
- Components can be placed inside containers such as panels.
- Because containers can themselves be put inside other containers, the class Container extends Component.



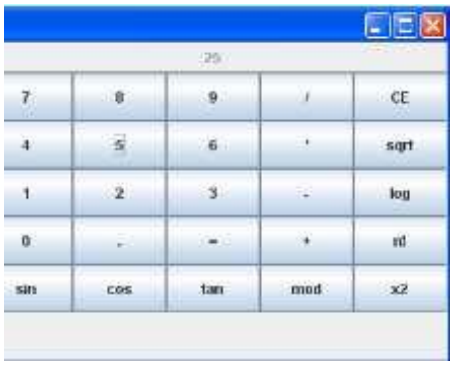
BORDER LAYOUT

- The border layout manager is the default layout manager of the content pane of every JFrame.
- Unlike the flow layout manager, which completely controls the position of each component, the border layout manager lets you choose where you want to place each component.

GRID LAYOUT

The grid layout arranges all components in rows and columns like a spreadsheet.

- All components are given the same size.
- The calculator program uses a grid layout to arrange the calculator buttons.
- When you resize the window, the buttons grow and shrink, but all buttons have identical sizes.



5.11 SWING COMPONENT

USING TOP-LEVEL CONTAINERS

- Discusses how to use the features shared by the JFrame, JDialog, and JApplet classes-content panes, menu bars, and root panes.
- It also discusses the containment hierarchy, which refers to the tree of components contained by a top-level container.

THE JCOMPONENT CLASS

- Tells you about the features JComponent provides to its subclasses which include almost all Swing components and gives tips on how to take advantage of these features.
- This section ends with API tables describing the commonly used API defined by JComponent and its superclasses, Container and Component.

USING TEXT COMPONENTS

- Describes the features and API shared by all components that descend from JtextComponent, not needed to read this section if you are just using text fields.

HOW TO WORK WITH IT

- It is recommend reading the relevant this, sections, once you are ready to start using Swing components in your own programs.
- For example, if your program needs a frame, a label, a button, and a color chooser, you should read How to Make Frames, How to Use Labels, How to Use Buttons, and How to Use Color Choosers.

USING MODELS

- Tells you about the Swing model architecture.
- This variation on Model-View-Controller (MVC) means that you can, if you wish, specify how the data and state of a Swing component are stored and retrieved.
- The benefits are the ability to share data and state between components, and to greatly improve the performance of components such as tables that display large amounts of data.

USING BORDERS

- Borders are very handy for drawing lines, titles, and empty space around the edges of components.
- This section tells you how to add a border to any JComponent.

USING ICONS

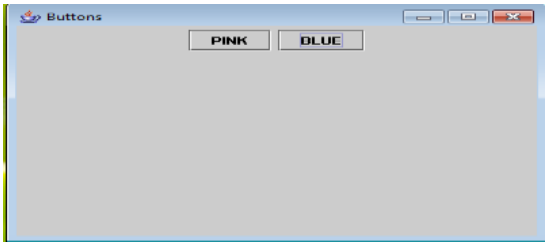
- Many Swing components can display icons. Usually, icons are implemented as instances of the ImageIcon class.
    - Eg: for Button. getActionCommand( ) method to get the action command from an actionevent.

ACTIONS

- An **action** is an object that encapsulates
    - A description of the command (as a text string and an optional icon)
    - Parameters that are necessary to carry out the command
- The Action interface has the following methods:
    1. void actionPerformed(ActionEvent event)
    2. void setEnabled(boolean b)
    3. boolean isEnabled()
    4. void putValue(String key, Object value)
    5. Object getValue(String key)
    6. void addPropertyChangeListener(PropertyChangeListener listener)
    7. void removePropertyChangeListener(PropertyChangeListener listener)
- Button class is useful to create push buttons.
- A push button is useful to perform a particular action.

PROGRAM

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class A extends JFrame implements ActionListener
{
 JButton b1,b2;
 A()
 {
  Container m=getContentPane();
  m.setLayout(new FlowLayout());
  b1=new JButton("BLUE");
  b2=new JButton("PINK");
  m.add(b2);
  m.add(b1);

  b1.addActionListener(this);
  b2.addActionListener(this);
 }
 public void actionPerformed(ActionEvent ae)
 {
  String s=ae.getActionCommand();
   if(s.equals("BLUE")) this.setBackground(Color.blue);
  if(s.equals("PINK")) this.setBackground(Color.pink);
 }
 public static void main(String args[])
 {
  A a=new A();
  a.setSize(400,300);
  a.setVisible(true);
  a.setTitle("Buttons");
  a.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```



SWING COMPONENTS

1. **CHECK BOX**
   - A checkbox is a square shaped box which displays an option to the user.
   - The user can select one or more options from a group of checkboxes.
2. **RADIO BUTTON**
   - A radio button represents a round shaped button,such that only one can be selected at a time from a group of buttons.
   - Radio buttons can be created using **CheckBoxGroup** class and checkbox classes.
3. **TEXT FIELD**
   - Represents a long rectangular box, where the user can type a single line of text.
   - To create a textfield,

     JTextField t=new JTextField();

- To retrieve text from textfield,
  String s=t.getText();
- To set the text to textfield,
  t.setText("HELLO");
- To hide the text being typed into a textfield by a character *,
  t.setEchoCharacter('*');

4. TEXT AREA

- Similar to textfield but it accommodate several lines of text.
- To create a textarea:
  JTextArea ta=new JTextArea();
- To append the given text to the current text area,
  ta.append("java");

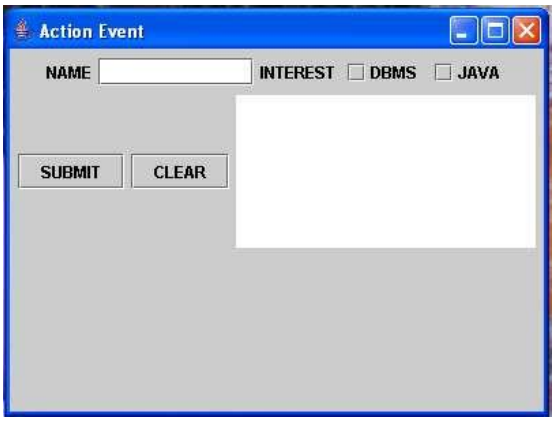
5. LABEL

- A constant text displayed along with TextField or TextArea
- To create a label,
  Label l=new Label();
  Label l=new Label("Name",Label.LEFT);

PROGRAM

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
class ActionDemo extends JFrame implements ActionListener
{
  JTextArea ta;
  String msg;
  JLabel lb1,lb2;
  JTextField ta1;
  JButton b1,b2;
  JCheckBox cb1,cb2;
  ActionDemo()
  {
 Container c=getContentPane();
c.setLayout(new FlowLayout());
lb1=new JLabel("NAME");
lb2=new JLabel("INTEREST");
ta1=new JTextField(10);
b1=new JButton("SUBMIT");
b2=new JButton("CLEAR");
cb1=new JCheckBox("DBMS");
cb2=new JCheckBox("JAVA");
ta=new JTextArea(7,20);
c.add(lb1);
c.add(ta1);
c.add(lb2);
c.add(cb1);
c.add(cb2);
c.add(b1);
c.add(b2);
c.add(ta); setSize
(400,300);
```

```
 setVisible(true);
 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 setTitle("Action Event");
 b1.addActionListener(this);
 b2.addActionListener(this);
   }
   public static void main(String args[])
   {
  ActionDemo a=new ActionDemo();
   }
   public void actionPerformed(ActionEvent ae)
   {
 msg=ta1.getText();
 if(cb1.getModel().isSelected())msg+="\n DBMS";
 if(cb2.getModel().isSelected())msg+="\n JAVA";
 ta.setText(msg);
 if(ae.getActionCommand()=="CLEAR")
 {
  ta1.setText("");
  ta.setText("");
 }
    }
}
```



PART A

1. **What is the relationship between the Canvas class and the Graphics class?**
   A Canvas object provides access to a Graphics object via its paint() method.
2. **How would you create a button with rounded edges?**
   There's 2 ways. The first thing is to know that a JButton's edges are drawn by a Border. so you can override the Button's paintComponent(Graphics) method and draw a circle or rounded rectangle (whatever), and turn off the border. Or you can create a custom border that draws a circle or rounded rectangle around any component and set the button's border to it.
3. **What is the difference between the 'Font' and 'FontMetrics' class?**
   The Font Class is used to render 'glyphs' - the characters you see on the screen. FontMetrics encapsulates information about a specific font on a specific Graphics object. (width of the characters, ascent, descent)
4. **What is the difference between the paint() and repaint() methods?**
   The paint() method supports painting via a Graphics object. The repaint() method is used to cause paint() to be invoked by the AWT painting thread.
5. **Which containers use a border Layout as their default layout?**
   The window, Frame and Dialog classes use a border layout as their default layout.
6. **What is the difference between applications and applets?**
   - Application must be run on local machine whereas applet needs no explicit installation on local machine.
   - Application must be run explicitly within a java-compatible virtual machine whereas applet loads and runs itself automatically in a java-enabled browser.
   - Application starts execution with its main method whereas applet starts execution with its init method.
   - Application can run with or without graphical user interface whereas applet must run within a graphical user interface.
7. **What is the difference between Swing and AWT?**

Sl.No	AWT (Abstract Window Toolkit)	SWING
1	AWT components are heavy weight, components are platform dependent.	Swing components are light weight, components are platform independent
2	AWT components support Delegate Event Model	Swing components support MVC (Model, View, Control Architecture)
3	AWT components provide static look and feel	Swing component provide dynamic look and feel.
4	It does not provide Tooltip text for components	It provide Tooltip text for component.

8. **What is a layout manager and what are different types of layout managers available in java AWT?**
   A layout manager is an object that is used to organize components in a container. The different layouts are available are FlowLayout, BorderLayout, CardLayout, GridLayout and GridBagLayout.
9. **How are the elements of different layouts organized?**
   FlowLayout: The elements of a FlowLayout are organized in a top to bottom, left to right fashion.
   BorderLayout: The elements of a BorderLayout are organized at the borders (North, South, East and West) and the center of a container.
   CardLayout: The elements of a CardLayout are stacked, on top of the other, like a deck of cards.

GridLayout: The elements of a GridLayout are of equal size and are laid out using the square of a grid.

GridBagLayout: The elements of a GridBagLayout are organized according to a grid. However, the elements are of different size and may occupy more than one row or column of the grid. In addition, the rows and columns may have different sizes.

The *default* Layout Manager of Panel and Panel sub classes is FlowLayout.

10. Why would you use SwingUtilities.invokeAndWait or SwingUtilities.invokeLater?

I want to update a Swing component but I'm not in a callback. If I want the update to happen immediately (perhaps for a progress bar component) then I'd use invokeAndWait. If I don't care when the update occurs, I'd use invokeLater.

11. What is an event and what are the models available for event handling?

An event is an event object that describes a state of change in a source. In other words, event occurs when an action is generated, like pressing button, clicking mouse, selecting a list, etc. There are two types of models for handling events and they are: a) event-inheritance model and b) eventdelegation model

12. What is the difference between scrollbar and scrollpane?

A Scrollbar is a Component, but not a Container whereas Scrollpane is a Conatiner and handles its own events and perform its own scrolling.

13. Why won't the JVM terminate when I close all the application windows?

The AWT event dispatcher thread is not a daemon thread. You must explicitly call System.exit to terminate the JVM.

14. What is meant by controls and what are different types of controls in AWT?

Controls are components that allow a user to interact with your application and the AWT supports the following types of controls: Labels, Push, Buttons, Check Boxes, Choice Lists, Lists, Scrollbars, and Text Components. These controls are subclasses of Component.

15. What is the difference between a Choice and a List?

A Choice is displayed in a compact form that requires you to pull it down to see the list of available choices. Only one item may be selected from a Choice. A List may be displayed in such a way that several List items are visible. A List supports the selection of one or more List items.

16. What is the purpose of the enableEvents() method?

The enableEvents() method is used to enable an event for a particular object. Normally, an event is enabled when a listener is added to an object for a particular event. The enableEvents() method is used by objects that handle events by overriding their eventdispatch methods.

17. What is the difference between the File and RandomAccessFile classes?

The File class encapsulates the files and directories of the local file system. The RandomAccessFile class provides the methods needed to directly access data contained in any part of a file.

18. What is the lifecycle of an applet?

- init() method - Can be called when an applet is first loaded
- start() method - Can be called each time an applet is started.
- paint() method - Can be called when the applet is minimized or maximized.
- stop() method - Can be used when the browser moves off the applet's page.
- destroy() method - Can be called when the browser is finished with the applet.

19. What is the difference between a MenuItem and a CheckboxMenuItem?

The CheckboxMenuItem class extends the MenuItem class to support a menu item that may be checked or unchecked.

20. What class is the top of the AWT event hierarchy?

The java.awt.AWTEvent class is the highest-level class in the AWT event-class hierarchy.

21. What is source and listener?

source : A source is an object that generates an event. This occurs when the internal state of that object changes in some way.

listener : A listener is an object that is notified when an event occurs. It has two major requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications.

22. Explain how to render an HTML page using only Swing.

Use a JEditorPane or JTextPane and set it with an HTMLEditorKit, then load the text into the pane.

23. How would you detect a keypress in a JComboBox?

This is a trick. most people would say 'add a KeyListener to the JComboBox' - but the right answer is 'add a KeyListener to the JComboBox's editor component.'

24. What an I/O filter?

An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.

25. How can I create my own GUI components?

Custom graphical components can be created by producing a class that inherits from java.awt.Canvas. Your component should override the paint method, just like an applet does, to provide the graphical features of the component.