UNIT - IV

Event Handling: Events, Event sources, Event classes, Event Listeners, Delegation event model,handling mouse and keyboard events, Adapter classes. The AWT class hierarchy, user interface components- labels, button, canvas, scrollbars, text components, check box, checkbox groups, choices,lists panels –scrollpane, dialogs, menubar, graphics, layout manager – layout manager types – border, grid, flow, card and grid bag.

EVENT HANDLING

- In general we can not perform any operation on dummy GUI screen even any button click or select any item.
- To perform some operation on these dummy GUI screen you need some predefined classes and interfaces.
- All these type of classes and interfaces are available in java.awt.event package.
- Changing the state of an object is known as an event.
- The process of handling the request in GUI screen is known as event handling (event represent an action). It will be changes component to component.

Note: In event handling mechanism event represent an action class and Listener represent an interface. Listener interface always contains abstract methods so here you need to write your own logic.

EVENTS

- The Events are the objects that define state change in a source.
- An event can be generated as a reaction of a user while interacting with GUI elements.
- Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on.
- We can also consider many other user operations as events.

EVENT SOURCES

- A source is an object that causes and generates an event.
- It generates an event when the internal state of the object is changed.
- The sources are allowed to generate several different types of events.
- A source must register a listener to receive notifications for a specific event.
- Each event contains its registration method.

Syntax

```
public void addTypeListener (TypeListener e1)
```

From the above syntax, the Type is the name of the event, and e1 is a reference to the event listener.

- For example, for a keyboard event listener, the method will be called as addKeyListener().
- For the mouse event listener, the method will be called as addMouseMotionListener(). When an event is triggered using the respected source, all the events will be notified to registered listeners and receive the event object.
- This process is known as event multicasting.

EVENT LISTENERS

- It is also known as event handler.
- Listener is responsible for generating response to an event.
- From java implementation point of view the listener is also an object.
- Listener waits until it receives an event.
- Once the event is received, the listener process the event and then returns.

EVENTS	SOURCE	LISTENERS
Action Event	Button, List, MenuItem, Text field	ActionListener
Component Event	Component	Component Listener
Focus Event	Component	FocusListener
Item Event	Checkbox, CheckboxMenuItem, Choice, List	ItemListener
Key Event	when input is received from keyboard	KeyListener
Text Event	Text Component	TextListener
Window Event	Window	WindowListener
Mouse Event	Mouse related event	MouseListener

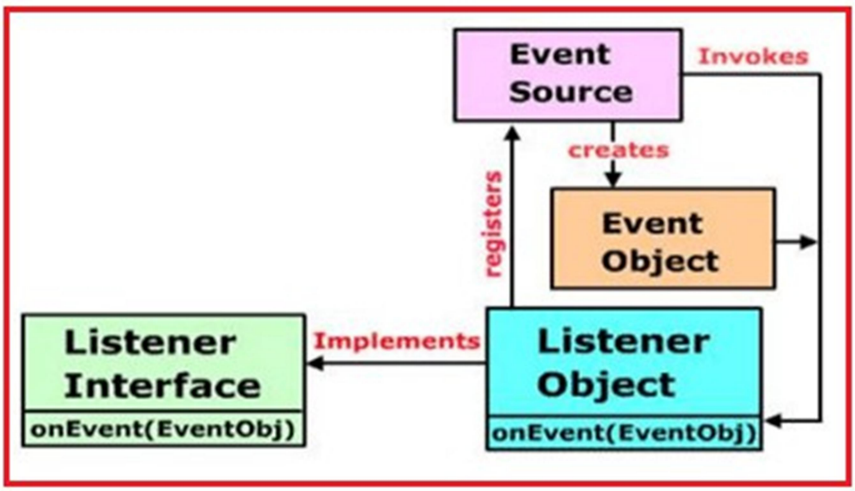
EVENT CLASSES AND LISTENER INTERFACES

Event Classes	Description	Listener Interface
ActionEvent	generated when button is pressed, menu-item is selected, list-item is double clicked	**ActionListener**
MouseEvent	generated when mouse is dragged, moved, clicked, pressed or released and also when it enters or exit a component	**MouseListener**
KeyEvent	generated when input is received from keyboard	**KeyListener**
ItemEvent	generated when check-box or list item is clicked	**ItemListener**
TextEvent	generated when value of textarea or textfield is changed	**TextListener**
MouseWheelEvent	generated when mouse wheel is moved	**MouseWheelListener**
WindowEvent	generated when window is activated, deactivated, deiconified, iconified, opened or closed	**WindowListener**
ComponentEvent	generated when component is hidden, moved, resized or set visible	**ComponentEventListener**
ContainerEvent	generated when component is added or removed from container	**ContainerListener**
AdjustmentEvent	generated when scroll bar is manipulated	**AdjustmentListener**

DELEGATION EVENT MODEL IN JAVA

- The Delegation Event model is defined to handle events in GUI programming languages.
- The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

- The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling. The below image demonstrates the event processing.
- In this model, a source generates an event and forwards it to one or more listeners.
- The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it.



REGISTRATION METHODS

For registering the component with the Listener, many classes provide the registration methods.

Button

public void addActionListener(ActionListener a)

{

}

MenuItem

public void addActionListener(ActionListener a)

{

}

TextField

public void addActionListener(ActionListener a)

{

}

public void addTextListener(TextListener a)

{

}

TextArea

public void addTextListener(TextListener a)

{

}

Checkbox

public void addItemListener(ItemListener a)

{

}

Choice

public void addItemListener(ItemListener a)

{

}

List

public void addActionListener(ActionListener a)

{

}

public void addItemListener(ItemListener a)

{

}

STEPS TO PERFORM EVENT HANDLING

- Following steps are required to perform event handling:
- Implement the Listener interface and overrides its methods
- Register the component with the Listener
- The User clicks the button and the event is generated.
- Now the object of concerned event class is created automatically and information about the source and the event get populated with in same object.
- Event object is forwarded to the method of registered listener class.
- The method is now get executed and returns.

Syntax to Handle the Event class className implements XXXListener

{

.....

.....

}

addcomponentobject.addXXXListener(this);

...../// override abstract method of given interface and write proper logic

public void methodName(XXXEvent e)

{

.....

.....

}

.....

}

EVENT HANDLING FOR MOUSE

For handling event for mouse you need Mouse Event class and Mouse Listener interface.

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

Methods of MouseListener interface

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

1. public abstract void mouseClicked(MouseEvent e);

2. public abstract void mouseEntered(MouseEvent e);

3. public abstract void mouseExited(MouseEvent e);

4. public abstract void mousePressed(MouseEvent e);

5. public abstract void mouseReleased(MouseEvent e);

Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
 Label l;
 MouseListenerExample()
 {
 addMouseListener(this);
l=new Label();
l.setBounds(20,50,100,20);
add(l);
 setSize(300,300);
setLayout(null);
setVisible(true);
```

```
}
public void mouseClicked(MouseEvent e)
{
l.setText("Mouse Clicked");
}
public void mouseEntered(MouseEvent e)
{
l.setText("Mouse Entered");
}
public void mouseExited(MouseEvent e)
{
l.setText("Mouse Exited");
}
public void mousePressed(MouseEvent e)
{
l.setText("Mouse Pressed");
}
public void mouseReleased(MouseEvent e)
{
l.setText("Mouse Released");
}
public static void main(String[] args)
{
new MouseListenerExample();
```

}

}

Output:

EVENT HANDLING FOR KEYBOARD

The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent.

The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface

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

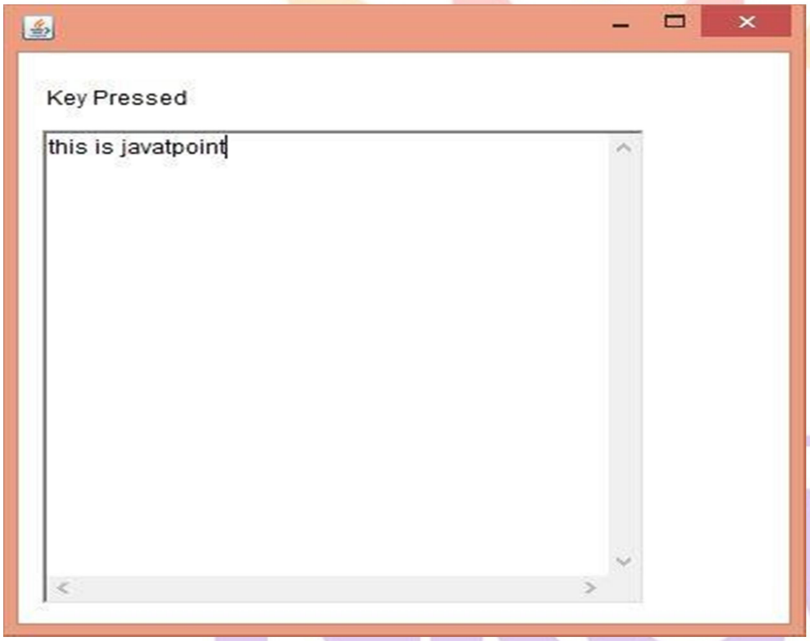
EXAMPLE

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerExample extends Frame implements KeyListener
{
 Label l;
```

```
TextArea area;
KeyListenerExample()
{
l=new Label();
l.setBounds(20,50,100,20);
area=new TextArea();
area.setBounds(20,80,300, 300);
area.addKeyListener(this);
add(l);
add(area);
setSize(400,400);
setLayout(null);
setVisible(true);
}
public void keyPressed(KeyEvent e)
{
l.setText("Key Pressed");
}
public void keyReleased(KeyEvent e)
{
l.setText("Key Released");
}
public void keyTyped(KeyEvent e)
{
```

```
l.setText("Key Typed");
}
public static void main(String[] args)
{
new KeyListenerExample();
}
}
```

Output:



ADAPTER CLASSES

- In a program, when a listener has many abstract methods to override, it becomes complex for the programmer to override all of them.
- For example, for closing a frame, we must override seven abstract methods of WindowListener, but we need only one method of them.
- For reducing complexity, Java provides a class known as "adapters" or adapter class.
- Adapters are abstract classes, that are already being overriden.

Adapter Class	Listener Interface
ComponentAdapter	ComponentListner
ContainerAdapter	ContainerListner
FocusAdapter	FocusListner
KeyAdapter	KeyListner
MouseAdapter	MouseListner
MouseMotionAdapter	MouseMotionListner
WindowAdapter	WindowListner

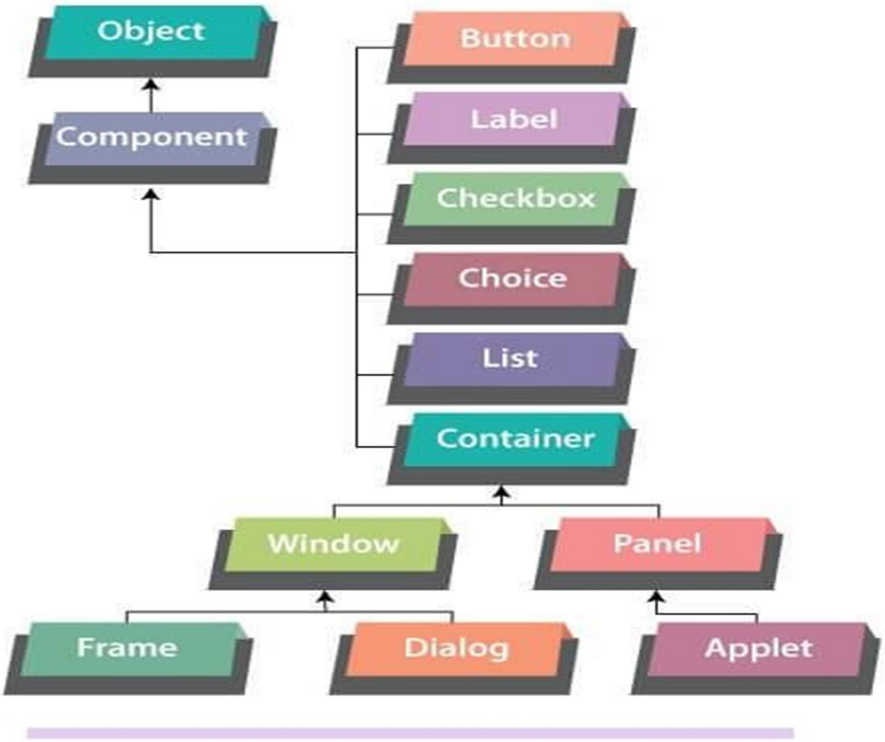
Java WindowAdapter Example

```
import java.awt.*; import java.awt.event.*; public class AdapterExample
{
 Frame f;
 AdapterExample()
 {
 f=new Frame("Window Adapter");
 f.addWindowListener(new WindowAdapter()
 {
 public void windowClosing(WindowEvent e)
 {
 f.dispose();
 }
 });
```

```
f.setSize(400,400);
f.setVisible(true);
} public static void main(String[] args)
{
new AdapterExample();
} }
```

JAVA AWT HIERARCHY

The hierarchy of Java AWT classes are given below.



Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

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

AWT FRAME

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

Frame f=new Frame();

Methods

1. setTitle()

It is used to display user defined message on title bar. Frame f=new Frame();

f.setTitle("myframe");

2. setBackground()

It is used to set background or image of frame. Frame f=new Frame();

f.setBackground(Color.red);

3. setForground()

It is used to set the foreground text color. Frame f=new Frame();

f.setForground(Color.red);

4. setSize()

It is used to set the width and height for frame. Frame f=new Frame();

f.setSize(400,300);

5. setVisible()

It is used to make the frame as visible to end user. Frame f=new Frame();

f.setVisible(true);

Note: You can write setVisible(true) or setVisible(false), if it is true then it visible otherwise not visible.

7.add()

It is used to add non-container components (Button, List) to the frame.

```
Frame f=new Frame();

Button b=new Button("Click");

f.add(b);
```

Explanation: In above code we add button on frame using f.add(b), here b is the object of Button class..

Example

```
import java.awt.*;

class FrameDemo {

public static void main(String[] args)

{

Frame fObj;

Button button1, button2, button3;

// instantiating using the constructor

ButtonExample2() {

fObj = new Frame ("Frame to display buttons");

button1 = new Button();

button2 = new Button ("Click here");

button3 = new Button();

button3.setLabel("Button 3");

fObj.add(button1);

fObj.add(button2);

fObj.add(button3);

fObj.setLayout(new FlowLayout());

fObj.setSize(300,400);
```

```
fObj.setVisible(true);

} }

}
```

AWT PANEL

It is a predefined class used to provide a logical container to hold various GUI component. Panel always should exist as a part of frame.

Note: Frame is always visible to end user where as panel is not visible to end user. Panel is a derived class of container class so you can use all the methods which is used in frame. Syntax

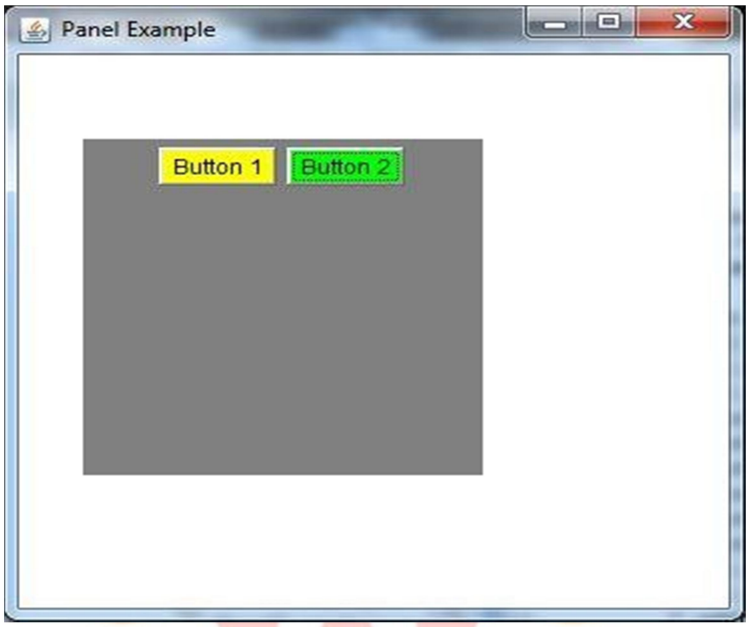
```
Panel p=new Panel();
```

Example

```
import java.awt.*;

public class PanelExample {

   PanelExample()
```

```
{
Frame f= new Frame("Panel Example");
Panel panel=new Panel();
panel.setBounds(40,80,200,200);
panel.setBackground(Color.gray);
Button b1=new Button("Button 1");
b1.setBounds(50,100,80,30);
b1.setBackground(Color.yellow);
Button b2=new Button("Button 2");
b2.setBounds(100,100,80,30);
b2.setBackground(Color.green);
panel.add(b1); panel.add(b2);
f.add(panel);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new PanelExample();
}
}
```

Output:



AWT Label

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

Example

```
import java.awt.*;
public class LabelExample
{
public static void main(String args[])
{
 Frame f = new Frame ("Label example");
 Label l1, l2;
 l1 = new Label ("First Label.");
 l2 = new Label ("Second Label.");
 l1.setBounds(50, 100, 100, 30);
 l2.setBounds(50, 150, 100, 30);
```

```
        f.add(l1);
        f.add(l2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    }
```

Output

Java AWT Button

A button is basically a control component with a label that generates an event when pushed.

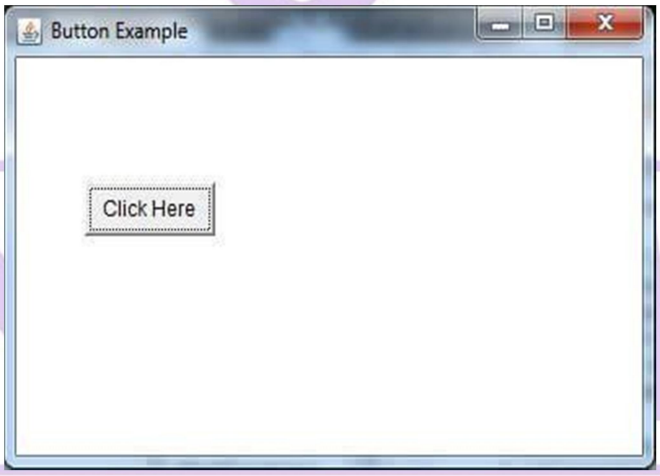
The Button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

Example

```
import java.awt.*;
public class ButtonExample
{
```

```
public static void main (String[] args)
{
 Frame f= new Frame("Button Example");
 Button b = new Button("Click Here");
 b.setBounds(50,100,80,30);
 f.add(b);
 f.setSize(400,400);
 f.setLayout(null);
 f.setVisible(true);
}
}
```

Output:



Java AWT Canvas

The Canvas class controls and represents a blank rectangular area where the application can draw or trap input events from the user. It inherits the Component class.

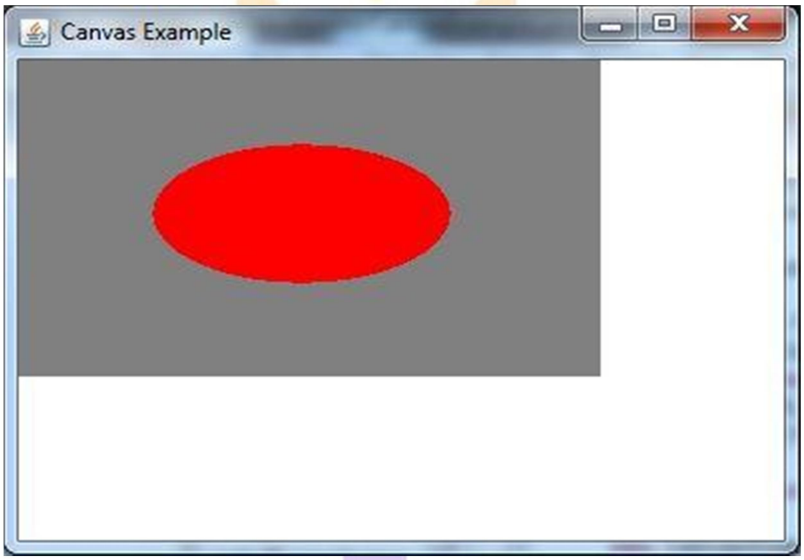
Example

```
import java.awt.*;
public class CanvasExample
```

```
{
 public CanvasExample()
 {
 Frame f= new Frame("Canvas Example");
 f.add(new MyCanvas());
 f.setLayout(null);
 f.setSize(400, 400);
 f.setVisible(true);
 }
 public static void main(String args[])
 {
 new CanvasExample();
 }
}
class MyCanvas extends Canvas
{
 public MyCanvas()
 {
 setBackground (Color.GRAY);
 setSize(300, 200);
 }
 public void paint(Graphics g)
 {
 g.setColor(Color.red);
```

```
g.fillOval(75, 75, 150, 75);
}
}
```

Output:



Java AWT Scrollbar

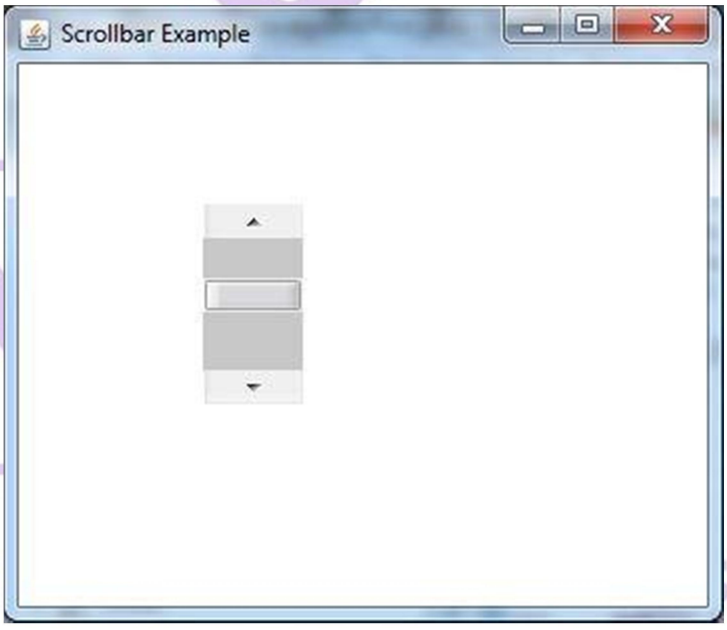
The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

Example

```
import java.awt.*;
public class ScrollbarExample1
{
ScrollbarExample1()
{
Frame f = new Frame("Scrollbar Example");
Scrollbar s = new Scrollbar();
```

```
s.setBounds (100, 100, 50, 100);
f.add(s);
f.setSize(400, 400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ScrollbarExample1();
}
}
```

Output:



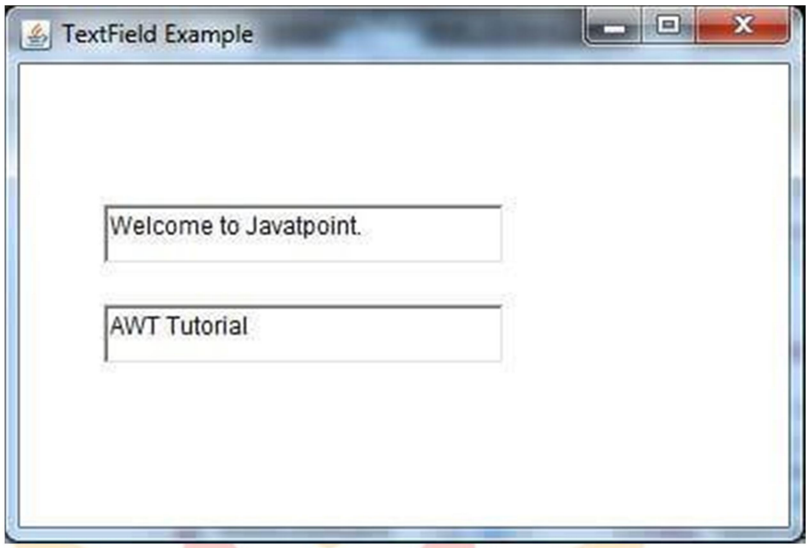
Java AWT TextField

The object of a TextField class is a text component that allows a user to enter a single line text and edit it. It inherits TextComponent class, which further inherits Component class.

Example:

```
import java.awt.*;
public class TextFieldExample1
{
 public static void main(String args[])
{
 Frame f= new Frame("TextField Example");
 TextField t1, t2;
 t1 = new TextField("Welcome to Javatpoint.");
 t1.setBounds(50, 100, 200, 30);
 t2 = new TextField("AWT Tutorial");
 t2.setBounds(50, 150, 200, 30);
 f.add(t1);
 f.add(t2);
 f.setSize(400,400);
 f.setLayout(null);
 f.setVisible(true);
}
}
```

Output:



Java AWT TextArea

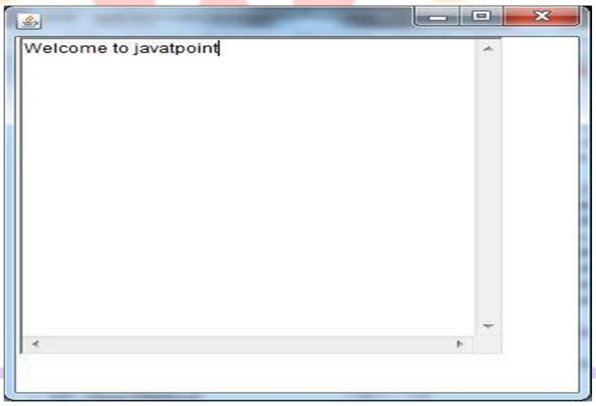
The object of a TextArea class is a multiline region that displays text. It allows the editing of multiple line text. It inherits TextComponent class.

Example

```
import java.awt.*;

public class TextAreaExample

{

 TextAreaExample()

{

 Frame f = new Frame();

 TextArea area = new TextArea("Welcome to javatpoint");

 area.setBounds(10, 30, 300, 300);

 f.add(area);

 f.setSize(400, 400);

 f.setLayout(null);

 f.setVisible(true);

}
```

```
public static void main(String args[])
{
 new TextAreaExample();
}
}
```

Output :



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false).

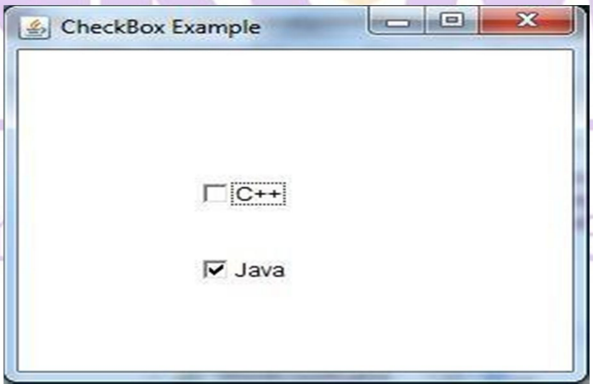
Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

Example

```
import java.awt.*;
public class CheckboxExample1
{
 CheckboxExample1()
{
```

```
Frame f = new Frame("Checkbox Example");
Checkbox checkbox1 = new Checkbox("C++");
checkbox1.setBounds(100, 100, 50, 50);
Checkbox checkbox2 = new Checkbox("Java", true);
checkbox2.setBounds(100, 150, 50, 50);
f.add(checkbox1);
f.add(checkbox2);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main (String args[])
{
new CheckboxExample1();
}
}
```

Output :



Java AWT CheckboxGroup

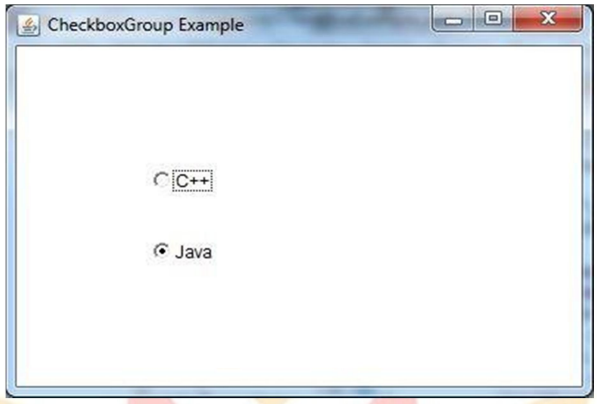
The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

Example

```
import java.awt.*;
public class CheckboxGroupExample
{
 CheckboxGroupExample()
 {
 Frame f= new Frame("CheckboxGroup Example");
 CheckboxGroup cbg = new CheckboxGroup();
 Checkbox checkBox1 = new Checkbox("C++", cbg, false);
 checkBox1.setBounds(100,100, 50,50);
 Checkbox checkBox2 = new Checkbox("Java", cbg, true);
 checkBox2.setBounds(100,150, 50,50);
 f.add(checkBox1);
 f.add(checkBox2);
 f.setSize(400,400);
 f.setLayout(null);
 f.setVisible(true);
 }
public static void main(String args[])
{
 new CheckboxGroupExample();
}
```

}

Output :



Java AWT Choice

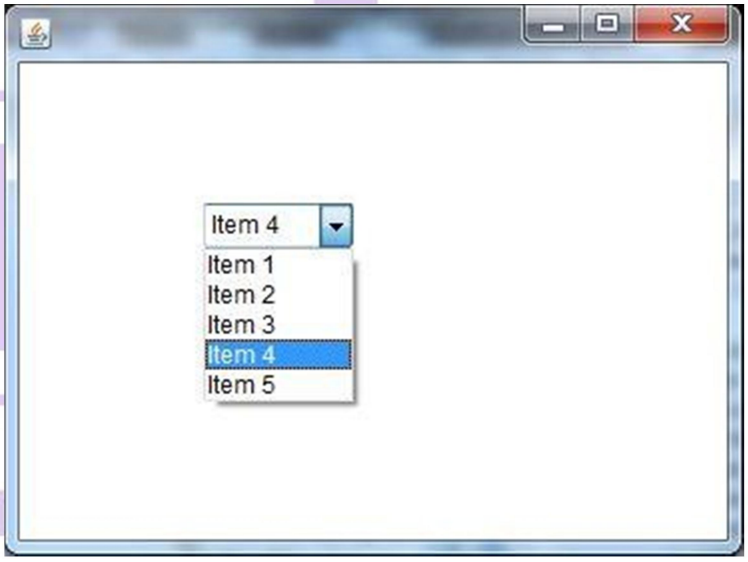
The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

Example

```
import java.awt.*;
public class ChoiceExample1
{
 ChoiceExample1()
{
 Frame f = new Frame();
 Choice c = new Choice();
 c.setBounds(100, 100, 75, 75);
 c.add("Item 1");
 c.add("Item 2");
 c.add("Item 3");
```

```
c.add("Item 4");
c.add("Item 5");
f.add(c);
f.setSize(400, 400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new ChoiceExample1();
}
}
```

Output:



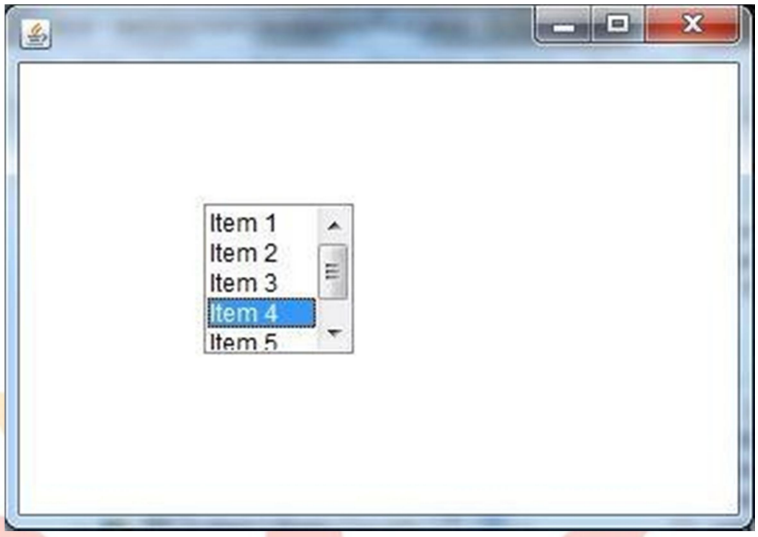
Java AWT List

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

Example

```
import java.awt.*;
public class ListExample1
{
 ListExample1()
{
 Frame f = new Frame();
 List l1 = new List(5);
 l1.setBounds(100, 100, 75, 75);
 l1.add("Item 1");
 l1.add("Item 2");
 l1.add("Item 3");
 l1.add("Item 4");
 l1.add("Item 5");
 f.add(l1);
 f.setSize(400, 400);
 f.setLayout(null);
 f.setVisible(true);
 }
 public static void main(String args[])
{
 new ListExample1();
}
}
```

Output :



Java AWT Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class.

Example

```
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
 private static Dialog d;
 DialogExample()
{
 Frame f= new Frame();
 d = new Dialog(f , "Dialog Example", true);
 d.setLayout( new FlowLayout() );
 Button b = new Button ("OK");
```

```
b.addActionListener ( new ActionListener()
{
public void actionPerformed( ActionEvent e )
{
DialogExample.d.setVisible(false);
}
});
d.add( new Label ("Click button to continue."));
d.add(b);
d.setSize(300,300);
d.setVisible(true);
}
public static void main(String args[])
{
new DialogExample();
}
}
```

Output :

Java AWT Menubar:

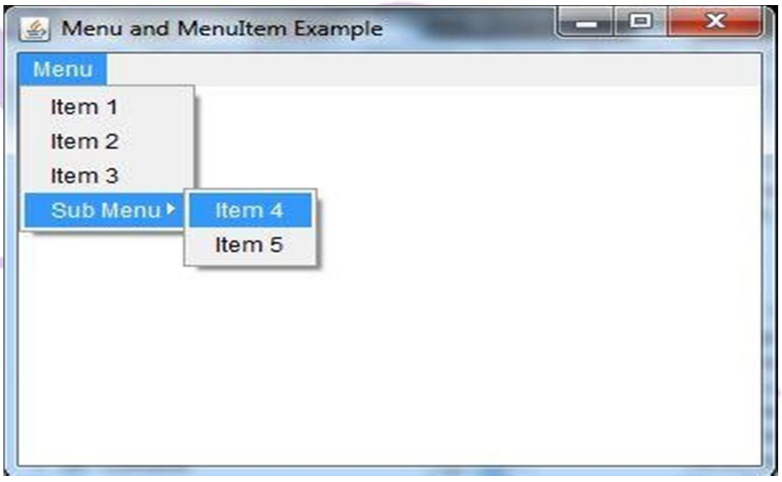
The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

```
import java.awt.*;
class MenuExample
{
 MenuExample()
 {
 Frame f= new Frame("Menu and MenuItem Example");
 MenuBar mb=new MenuBar();
 Menu menu=new Menu("Menu");
 Menu submenu=new Menu("Sub Menu");
 MenuItem i1=new MenuItem("Item 1");
 MenuItem i2=new MenuItem("Item 2");
 MenuItem i3=new MenuItem("Item 3");
 MenuItem i4=new MenuItem("Item 4");
 MenuItem i5=new MenuItem("Item 5");
 menu.add(i1);
 menu.add(i2);
 menu.add(i3);
 submenu.add(i4);
```

```
submenu.add(i5);
menu.add(submenu);
mb.add(menu);
f.setMenuBar(mb);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



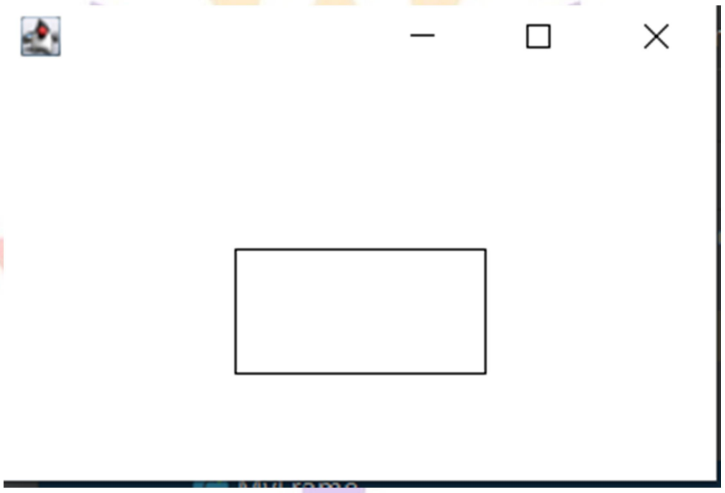
Java AWT Graphics

Graphics is an abstract class provided by Java AWT which is used to draw or paint on the components. It consists of various fields which hold information like components to be painted, font, color, XORmode, etc., and methods that allow drawing various shapes on the GUI components. Example

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
public class MyFrame extends Frame
{
 public MyFrame()
 {
 setVisible(true);
 setSize(300, 200);
 addWindowListener(new WindowAdapter()
{
 @Override
 public void windowClosing(WindowEvent e)
 {
 System.exit(0);
 }
 });
 }
 public void paint(Graphics g)
 {
 g.drawRect(100, 100, 100, 50);
 }
```

```
    public static void main(String[] args)
    {
    new MyFrame();
    }
   }
```

Output:



JAVA LAYOUT MANAGERS

- The Layout Managers are used to arrange components in a particular manner.
- Layout Manager is an interface that is implemented by all the classes of layout managers.

There are following classes that represents the layout managers: 1. BorderLayout

2. FlowLayout

3. GridLayout

4. CardLayout

5. GridBagLayout

BORDERLAYOUT

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

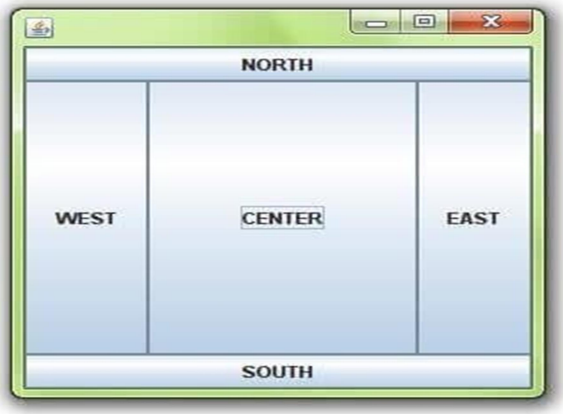
- public static final int NORTH
- public static final int SOUTH
- public static final int EAST
- public static final int WEST
- public static final int CENTER

EXAMPLE

```
import java.awt.*;
import javax.swing.*; public class Border { Border()
{
 JFrame 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();
 }
```

}

Output:



FLOWLAYOUT

- This layout is used to arrange the GUI components in a sequential flow (that means one after another in horizontal way)
- You can also set flow layout of components like flow from left, flow from right.

FlowLayout Left

```
Frame f=new Frame();

f.setLayout(new FlowLayout(FlowLayout.LEFT));
```

FlowLayout Right

```
Frame f=new Frame();

f.setLayout(new FlowLayout(FlowLayout.RIGHT))
```

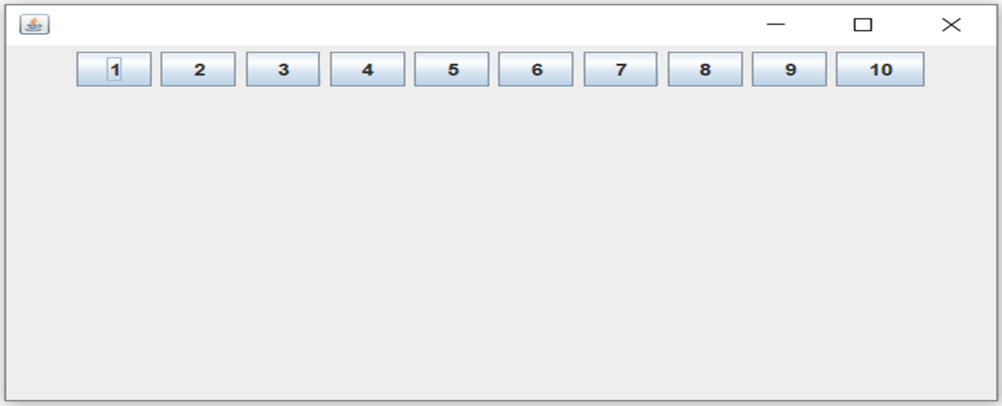
EXAMPLE

```
import java.awt.*;

import javax.swing.*;

 public class FlowLayoutExample

{
```

```
 JFrame frameObj;
  // constructor
FlowLayoutExample()
{
    // creating a frame object
    frameObj = new JFrame();
    // creating the buttons
    JButton b1 = new JButton("1");
    JButton b2 = new JButton("2");
    JButton b3 = new JButton("3");
    JButton b4 = new JButton("4");
    JButton b5 = new JButton("5");
    JButton b6 = new JButton("6");
    JButton b7 = new JButton("7");
    JButton b8 = new JButton("8");
    JButton b9 = new JButton("9");
    JButton b10 = new JButton("10");
            // adding the buttons to frame
    frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
    frameObj.add(b5); frameObj.add(b6);  frameObj.add(b7);  frameObj.add(b8);
    frameObj.add(b9);  frameObj.add(b10);
    frameObj.setLayout(new FlowLayout());
    frameObj.setSize(300, 300);
    frameObj.setVisible(true);
```

```
}
 // main method
public static void main(String argvs[])
{
    new FlowLayoutExample();
}
}
```

Output:



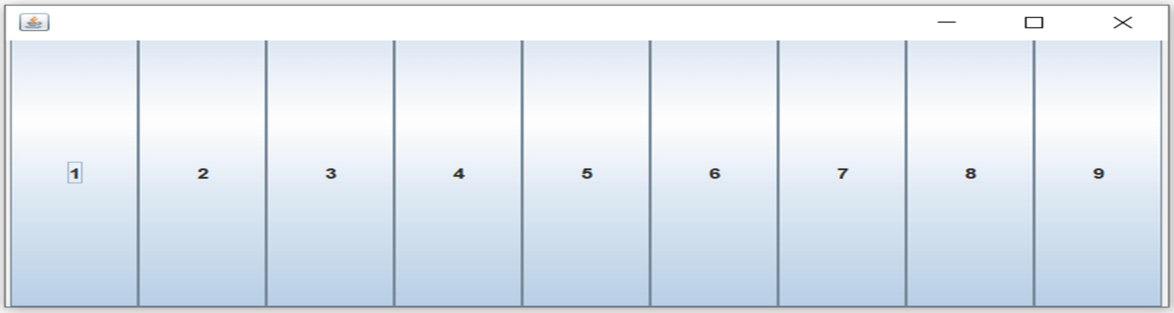
GRIDLAYOUT

This layout is used to arrange the GUI components in the table format. EXAMPLE

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout
{
MyGridLayout()
{
 JFrame f=new JFrame();
 JButton b1=new JButton("1");
```

```
JButton b2=new JButton("2");
JButton b3=new JButton("3");
JButton b4=new JButton("4");
JButton b5=new JButton("5");
JButton b6=new JButton("6");
JButton b7=new JButton("7");
JButton b8=new JButton("8");
JButton b9=new JButton("9");
f.add(b1);
f.add(b2);
f.add(b3);
f.add(b4);
f.add(b5); f.add(b6);
f.add(b7);
f.add(b8);
f.add(b9);
f.setLayout(new GridLayout(3,3));
f.setSize(300,300);
f.setVisible(true); } public static void main(String[] args)
{
new MyGridLayout(); } }
```


Output



CARDLAYOUT

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Commonly used methods of CardLayout class

1. public void next(Container parent): is used to flip to the next card of the given container.

2. public void previous(Container parent): is used to flip to the previous card of the given container.

3. public void first(Container parent): is used to flip to the first card of the given container.

4. public void last(Container parent): is used to flip to the last card of the given container.

5. public void show(Container parent, String name): is used to flip to the specified card with the given name.

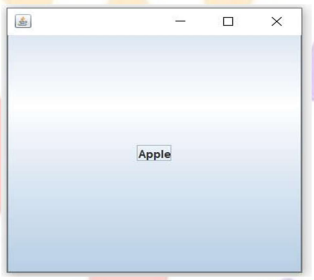
EXAMPLE

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener
{
CardLayout card;
JButton b1,b2,b3;
```

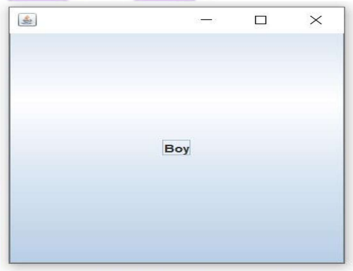
```
Container c;
CardLayoutExample()
{
 c=getContentPane();
 card=new CardLayout(40,30);
 //create CardLayout object with 40 hor space and 30 ver space
c.setLayout(card);
 b1=new JButton("Apple");
 b2=new JButton("Boy");
 b3=new JButton("Cat");
 b1.addActionListener(this);
 b2.addActionListener(this);
 b3.addActionListener(this);
 c.add("a",b1);
c.add("b",b2);
c.add("c",b3);
}
 public void actionPerformed(ActionEvent e)
 {
 card.next(c);
 }
public static void main(String[] args)
{
 CardLayoutExample cl=new CardLayoutExample();
```

```
cl.setSize(400,400);
cl.setVisible(true);
cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

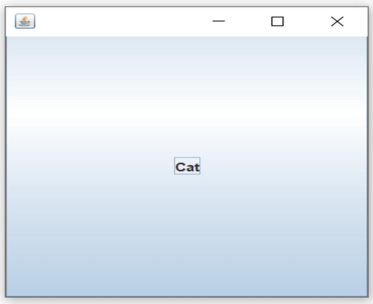
Output:



When the button named apple is clicked, we get



When the boy button is clicked, we get



GridBagLayout

- The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
- The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells.
- Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints.
- With the help of constraints object we arrange component's display area on the grid.
- The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Example

```
import java.awt.Button;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import javax.swing.*; public class GridBagLayoutExample extends JFrame

{

 public static void main(String[] args)

 {

 GridBagLayoutExample a = new GridBagLayoutExample();

 }

 public GridBagLayoutExample()

 {

 GridBagLayoutgrid = new GridBagLayout();

 GridBagConstraints gbc = new GridBagConstraints();

 setLayout(grid);

 setTitle("GridBag Layout Example");

 GridBagLayout layout = new GridBagLayout();

 this.setLayout(layout);
```

```
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridx = 0;
gbc.gridy = 0;
this.add(new Button("Button One"), gbc);
gbc.gridx = 1;
gbc.gridy = 0;
this.add(new Button("Button two"), gbc);
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.ipady = 20;
gbc.gridx = 0;
gbc.gridy = 1;
this.add(new Button("Button Three"), gbc);
gbc.gridx = 1;
gbc.gridy = 1;
this.add(new Button("Button Four"), gbc);
gbc.gridx = 0;
gbc.gridy = 2;
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.gridwidth = 2;
this.add(new Button("Button Five"), gbc);
setSize(300, 300);
setPreferredSize(getSize());
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

}

}

output: