Event Handling and Swings

Event Handling

1) Two Event Handling Mechanisms
2) The Delegation Event Model
    a. Events
    b. Event Sources
    c. Event Listeners
3) Event Classes
4) Sources of Events
5) Event Listener Interfaces
6) Adapter Classes

Swings

1) The Origin of Swing
2) Swing is built on the AWT
3) Two Key Features of Swing
4) Swing Components and Containers
5) A Simple Swing Application
6) Event Handling
7) Create a Swing Applet
8) Exploring Swing

Two Event Handling Mechanisms

- **Event** is the change in the state of the object or source. **Events** are generated as result of user interaction with the graphical user interface components.
- **For example**, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen.
- Event Handling is the mechanism that controls the event and decides what should happen if an event occurs. This mechanism have the code which is known as event handler that is executed when an event occurs.
- The Way in which events are handled changed significantly between the original version of java i.e 1.0 and modern versions of java. The Modern approach is called the Delegation Event Model.
- This model defines the standard mechanism to generate and handle the events.

Two Event Handling Mechanisms

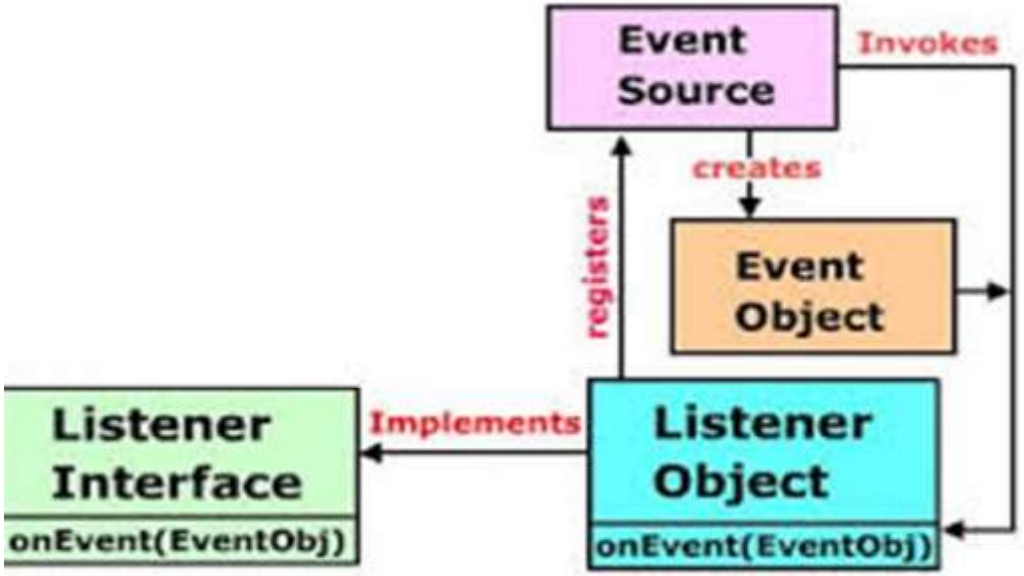
The Delegation Event Model has the following key participants namely

Source- The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. Java provide us with classes for source object.

Listener - 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 then returns.

Two Event Handling Mechanisms

- The Modern approach to handling events is based on the **delegation event model**, which defines standard and consistent mechanisms to generate and process events.
- A source generates an event and sends it to one or more listeners.
- In this schema, the listener simply waits until it receives an event. Once an event is received, the listener processes the event and then returns.
- The advantage of this design is that the application logic that processes events is cleanly separated from the user interface logic that generates those events.
- A User interface element is able to "delegate" the processing of an event to a separate piece of code.
- In the delegation Event model, listeners must register with a source in order to receive an event notification.
- This provides an important benefit notifications are sent only to listeners that want or receive them.
- This is a more efficient want to handle events than the design used by the old java 1.0 approach.



- In this model, there is a source, which generates events.
- There is a Listener, which can listen to the happenings of an event and initiate an action.
- A Listener has to register with a source.
- When an event takes place, it is notified to the listeners, which are registered with the source.
- The Listener then initiates an action.

Event Classes

The Classes that represent events are called Event Classes.

EVENT CLASS	DESCRIPTION
ActionEvent	Generates when a Button is pressed, a List item is double_clicked, or a Menu item is selected.
ItemEvent	Generated when a check box or list item is clicked Also occurs when a choice selection is made or a checkable MenuItem is selected or deselected.
TextEvent	Generated when the value of a TextArea or TextField is changed.
AdujustementEvent	Generated when a ScrollBar is manipulated
ContainerEvent	Generated when a component is added to or removed form a container.
KeyEvent	Generated when inpur is received form the keyboard.
FocusEvent	Generated when a component gains or loses keyboard focus.

Sources of Events

- An Event sources is a GUI Object which generates **Events.**
- Buttons, ListBoxes and Menus etc., are common Event sources in GUI based applications. **(or)**
- The Graphical User Interface Components that generates the Events are called **Event Sources.**
- Some of the User Interface Components that can generate Events are

EVENT SOURCE	DESCRIPTION
Button	Generates Action Events when the Button is Pressed
CheckBox	Generates Item Events when the checkbox is Selected or Deselected.
Choice	Generates Item Events when the choice is changed.
List	Generates Action Events when an item is DoubleClicked. Generates Item Events when a Item is Selected or Deselected.
MenuItem	Generates Action Events when an Menu item is Selected. Generates Item Events when a Checkable Menu Item is Selected and Deselected.
ScrollBar	Generates Adjustment Events when the scroll bar is manipulated.
TextComponent	Generates text events when the user enters a character.
Window	Generates window Events when a window is activated, closed, deactivated, deiconified, iconified, opened or quit.

EVENT LISTENER

- When an Event occurs, the event source invokes the appropriate method defined by the listener and provides an event object as its argument.
- The below table lists commonly used listener interfaces and provides a brief description of methods that they define.

INTERFACE	DESCRIPTION
ActionListener	Defines one method to receive action events
ItemListener	Defines one method to recognize when the state of an item changes.
TextListener	Defines one method to recognize when a text value changes.
AdjustementListener	Defines one method to receive adjustment event.
ContainerListener	Defines two method to recognize when a component is added to or removed from a container.
KeyListener	Defines three methods to recognize when a key is pressed, released, or typed.
FocusListener	Defines two methods to recognize when a component gains or losses keyboard focus.

EVENT SOURCE	EVENT CLASS	EVENT LISTENER	METHODS IN LISTENER INTERFACE
Button Clicked MenuItem Selected	ActionEvent	ActionListener	void actionPerformed(ActionEvent ae)
Combo box item selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
List Item Selected	ListSelectionE vent	ListSelectionListene r	void valueChanged(ListSelectionEvent le)
RadioButton Selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
Check Box Selected	ActionEvent ItemEvent	Action Listener ItemListener	void actionPerformed(ActionEvent ae) void itemStateChanged(ItemEvent ie)
Scroll Bar Repositioned	AdjustmentEve nt	AdustmentListener	void adjustmentValueeChanged(AdjustmetEvent ae)
Window Changed	WindowEvent	WindowListener	void windowAcitivated(WindowEvent we) void windowClosed(WindowEvent we) void windowClosing(WindowEvent we) void windowdeactivated(WindowEvent we) void windowDeiconified(WindowEvent we) void windowIconified(WindowEvent we) void windowOpened(WindowEvent we)
Focus Changed	FocusEvent	FocusListener	void focusLost(FocusEvent fe) void focusGain(FocusEvent fe)
Key Pressed	KeyEvent	KeyListener	void keyPressed(KeyEvent ke) void keyReleased(KeyEven tke) void keyTyped(KeyEvent ke)
Mouse clicked	MouseEvent	MouseListener	void mouseClicked(MouseEvent me) void

Adapter Classes

- An Adapter class provides an empty implementation of all methods in an event listener interface.
- Adapter classes are useful when you want to receive and process only some of the events that are handled by a particular event listener interface.
- Commonly used Listener interfaces implemented by Adapter Classes are

Adapter Class	Listener Interface
ComponentAdapter	ComponentListener
CaontainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MousrListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

SWING

1) The Origin of Swing
2) Swing is built on the AWT
3) Two Key Features of Swing
4) Swing Components and Containers
5) A Simple Swing Application
6) Event Handling
7) Create a Swing Applet
8) Exploring Swing

1) The origin of swing

- Swing is a set of classes that provide more powerful and flexible GUI Components than AWT(Abstract Window Toolkit).
- Swing is a package which contains classes related to graphical components like text box, check box, radio button etc...
- awt (abstract window toolkit) is also such package.
- So, awt and swing both are packages which contain classes for creating Graphical User Interface .
- The appearance of graphical components , that are created using awt package will not look consistent because the look and feel of the components depends on the os on which the application is executed.
- To overcome this limitation, java soft people introduced swing package , through which the components look and feel does not vary from os to os.

2) SWING IS BUILT ON THE AWT

- Although Swing eliminates a number of limitations inherent in the AWT, swing does not replace it.
- Instead, swing is built on the foundation of AWT. This is the reason why the AWT is still a crucial part of java.
- Swing also uses the same event handling mechanism as the AWT.
- Therefore, a basic understanding of the AWT and of event handling is required to use swing.

3) TWO KEY FEATURES OF SWING

- Swing was created to address the limitations present in the AWT.
- It does this through two key features
  - Lightweight components
  - Pluggable look and feel.

1) Swing components are LightWeight

- **Swing components** are lightweight, means that they are written entirely in java and do not map directly to platform-specific peers.
- Because light weight components do not translate into native peers, the look and feel of each component is determined by swing, not by underlying operating system.
- i.e Each component will work in a consistent manner across all platforms.

3) TWO KEY FEATURES OF SWING

2) **Swing supports a pluggable look and feel**

- Swing supports a pluggable look and feel(PLAF).
- Since swing follows MVC architecture, it is possible to separate the look and feel of the component from the logic of the component.
- Separating out the look and feel provides a significant advantage
- It becomes possible to change the way that a component is rendered without affecting any of its other aspects.
- In other words, it is possible to "plug in" a new look and feel for any given component without creating any side effects in the code that uses that component.

3) TWO KEY FEATURES OF SWING

- Instead, swing is built on the foundation of AWT. This is the reason why the AWT is still a crucial part of java.
- Swing also uses the same event handling mechanism as the AWT.
- Therefore, a basic understanding of the AWT and of event handling is required to use swing.

4) Components and containers

- Swing GUI consists of two key items
  1) Components
  2) Containers
- However, this distinction is mostly conceptual because all containers are also Components
- A Component is an independent visual control, such as a push button or radio button. A Container holds a group of components.
- Thus, a container is a special type of component that is designed to hold other components. Furthermore, in order for a component to be displayed, it must be held within a container.
- Thus, all Swing GUIs will have at least one container.
- Because containers are components, a container can also hold other containers.

4) Components and containers

1) Components

- In general, Swing components are derived from JComponent class.
- JComponent provides the functionality that is common to all components.
- All of Swing's components are represented by classes defined within the package javax.swing. Notice that all component classes begin with the letter J.

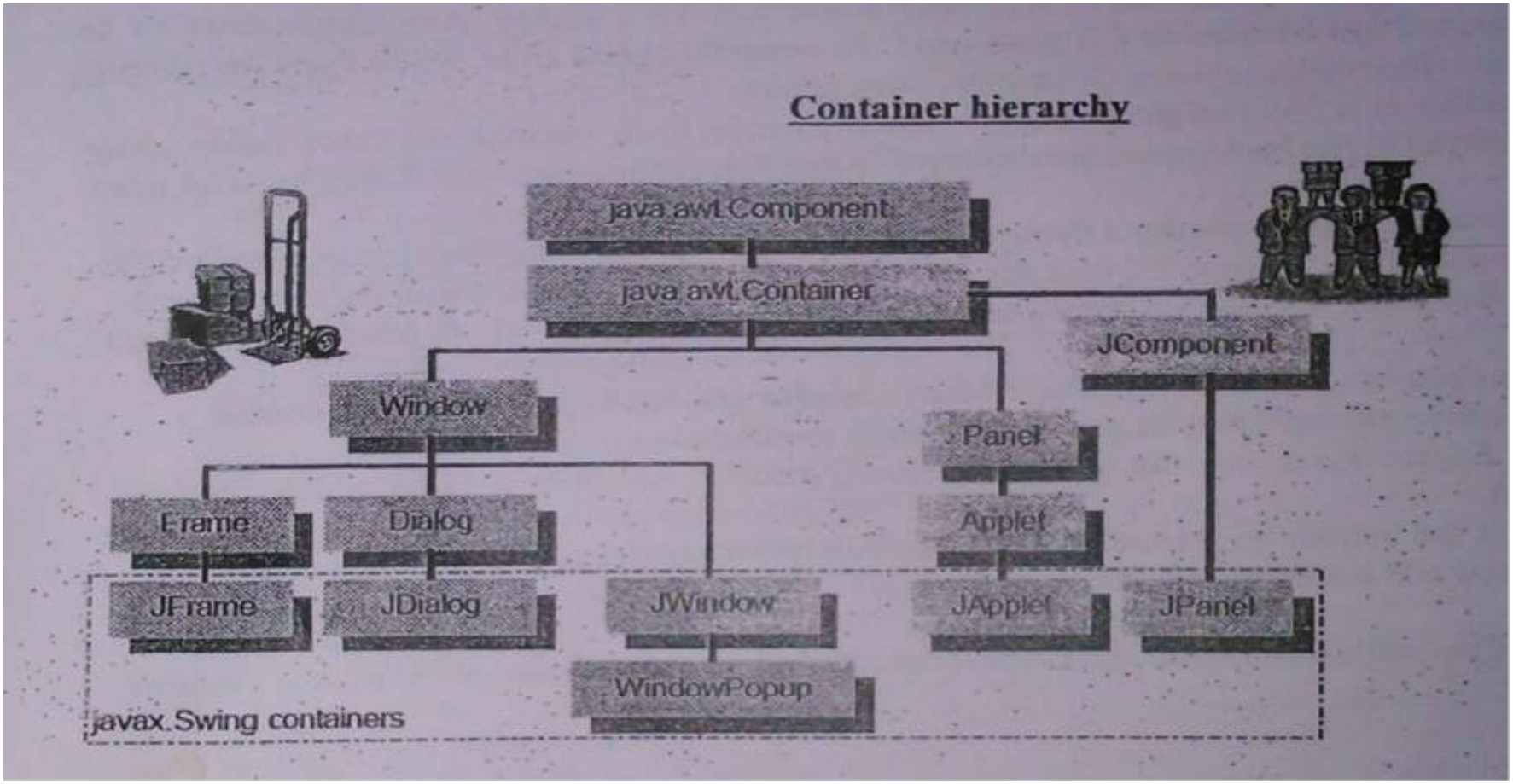
4) Components and containers

2) Containers

Swing defines two types of Containers.

2.1) Top-level containers (or) Heavy Weight Containers

2.2) Non Top-level Containers (or) Light Weight Containers



4) Components and containers

1) Top-level containers (or) Heavy Weight Containers

- The First type of containers supported by Swing are heavy Weight containers (or) Top-level containers. **They are JFrame, JApplet, JDialog, JWindow.**
- These containers do not inherit JComponent. They do, however, inherit the AWT classes Component and Container. So, we call them as Heavy Weight Containers.
- Top-level containers are defined as those which can be displayed directly on the desktop.
- The one most commonly used for applications is JFrame. The one used for Applets is JApplet.

4) Components and containers

2) Non Top-level Containers (or) LightWeight Containers

- The second type of Containers supported by Swing are **Light Weight Containers (or) Non top-level Containers.**
- **JPanel** comes under Light Weight Containers because it inherits from JComponent.
- Light Weight Components **are often used to organize and manage groups of related controls** that are contained within an outer container.

4) Components and containers

1) Top-level Container Panes

- Each top-level container defines a set of window Panes.
- A Window Pane represents a free area of a window where some text or component can be displayed.
- We have 4 types of window panes available in javax.Swing package.
- These panes can be imagined like transparent sheets lying one below the other.
- The Four Panes are
    1) JGlassPane
    2) JRoot Pane
    3) JLayeredPane
    4) JContentPane

4) Components and containers

JContentPane: This is the bottom most pane of all. The pane with which you will add visual components. In other words, when you add a component, such as a button, to a top-level container, you will add it to the content pane.

To reach this ContentPane, we use getContentPane() method of JFrame class which returns container class object

ASIMP

5) A Simple Swing Application

```
import java.awt.*;
import javax.swing.*;
class Demo2 extends JFrame
{
        JLabel i1,i2;
        JTextField t1,t2;
        Container con;
        ButtonGroup rbg;
        Demo2()
        {
                setSize(400,400);
                con=getContentPane();
                con.setLayout(new FlowLayout(FlowLayout.LEFT));
                i1=new JLabel("Name");
                t1=new JTextField(10);
                i2=new JLabel("DOB");
                t2=new JTextField(10);
                rbg=new ButtonGroup();
                JRadioButton rb1=new JRadioButton("male");
                JRadioButton rb2=new JRadioButton("female");
                con.add(i1);
                con.add(t1);
                con.add(i2);
                con.add(t2);
                rbg.add(rb1);
                rbg.add(rb2);
                con.add(rb1);
                con.add(rb2);
                JButton b1=new JButton("Save");
                con.add(b1);}}
```

5) A Simple Swing Application

```
class Demo3
{
        public static void main(String args[])
        {
                Demo2 ob=new Demo2();
                ob.setTitle("trail box");
                ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ob.setVisible(true);
        }
}
```

6) EVENT HANDLING

```
import javax.swing.*;
import java.applet.*;
/*
<applet code=Demo1 width=200 height=200>
</applet> */
public class Demo1  extends  JApplet  implements ActionListener
{
        JButton b1,b2;
        public void init()
        {
                b1 = new JButton("Alpha");
                b2 = new JButton("Beta");
                b1.addActionListener(this);
                b2.addActionListener(this);
                add(b1);
                add(b2);
        }
        public void actionPerformed(ActionEvent ae)
        {
                if (ae.getSource() == b1)
                        showStatus("Alpha is pressed");
                else
                        showStatus("Beta is pressed"); }
        }
}
```

6) EVENT HANDLING

7) CREATE A SWING APPLET

```
import java.awt.*;
import javax.swing.*;
import java.applet.*;
/*
<applet code=AppletDemo width=300 height=300>
</applet> */
public class AppletDemo  extends JApplet
{

   public void paint(Graphics g)
   {
         g.drawString("Hello , This is a Simple Applet Program", 20,20);

         }

}
```

8) Exploring Swing

1) JLabel and ImageIcon
2) JTextField
3) The swing Buttons
    a) Jbutton
    b) JRadioButton
    c) JToggleButton
    d) JCheckBox
4) JTabbedPane
5) JScrollPane
6) JList
7) JComboBox
8) JTrees

Exploring Swing

1) JLabel and ImageIcon

- JLabel is Swing's easiest-to-use component.
- It is a passive component In that it does not respond to user input.
- The text can be changed by the application and not by the user. A JLabel can be used to display text and/or an icon.
- The JLabel has the following int type constants that indicate the alignment of the labels content.

JLabel.CENTER,JLabel.LEFT, JLabel.RIGHT,JLabel.TOP,JLabel.BOTTOM

- JLabel defines several constructors. Here are 3 of them

JLabel(Icon i)// Creates a label using the Icon i

JLabel(String str)//Creates a Label with the specified String str.

JLabel(String str, Icon i, int align)// Creates a Label using the icon I, String Str and with the specified Alignment.

Exploring Swing

JLabel class has number of methods. Some of them are

Icon getIcon()//Returns the icon of the Label
String getText()//Returns the text of the Label
void setFont(Font font)//Sets the font for the Label's text
void setText(String str)//Sets the specified String str as the Label's content.

Exploring Swing

```
import java.awt.*; import javax.swing.*;
class lbl extends JFrame
{
        JLabel I1,I2,I3;
        Container con;
        Icon Img1,Img2,Img3;
        lbl()
        {       setSize(400,400); con=getContentPane();
                con.setLayout(new FlowLayout(FlowLayout.LEFT));
                Img1=new ImageIcon("lion.jpg");
                Img2=new ImageIcon("giraffee.jpg");
                Img3=new ImageIcon("PBear.jpg");
                I1=new JLabel("LION",Img1,JLabel.LEFT);
                I2=new JLabel("GIRAFFE",Img2,JLabel.LEFT);
                I3=new JLabel("POLAR BEAR",Img3,JLabel.LEFT);
                con.add(I1); con.add(I2);con.add(I3);
        }
}
class JLabelDemo
{
        public static void main(String args[])
        {
                lbl ob=new lbl();
                ob.setTitle("JLabel in JFrame");
                ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ob.setVisible(true); }
        }
}
```

Exploring Swing

JTextField

- The JTextField class implements a single-line text-entry area, usually called an edit control.
- TextField allow the user to enter Strings.
- JTextField is a subclass of JTextComponent, which is a subclass of JComponent/
- The Alignment of the text is defined by the following int type constants.

  TextField.LEFT, JTextField.CENTER, JTextField.RIGHT
- JTextField defines the following constructors.

  JTextField() // Creates a new Text field; the text is set to null and the number of columns Is set to 0.

  JTextField(int columns) //Creates a new empty textfirld with the specified number of columns.

  JTextField(String str) // Creates a new Text field with the specified string as text.
- JTextField class has number of methods. Some of them are

  String getText() // Returns the text contained in this Text Field.

  String getSelectedText() // Returns the selected text contained in this text field.

  Void setEditable(Boolean edit) // Sets the textfield to editable(true) or not

Exploring Swing

```
Import java.awt.*;
Import javax.swing.*;
Class Txt extends Jframe
{
        JLabel I1,I2,I3;
        JTextField t1,t2,t3;
        Container con;
        Txt()
        {
                setSize(350,200);
                con=getContentPane();
                con.setLayout(new FlowLayout(FlowLayout.LEFT));
                I1=new JLabel("Name"); t1=new JTextField(10);
                I2=new JLabel("RollNo"); t2=new JTextField(10);
                I3=new JLabel("Branch"); t3=new JTextField(10);
                con.add(I1); con.add(t1); con.add(I2); con.add(t2); con.add(I3); con.add(t3);
        }
}
class JTextFieldDemo
{
public static void main(String args[])
{
        Txt ob=new Txt(); ob.setTitle("JTextField in JFrame");
        ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ob.setVisible(true);
}
```

```
}
```

OUTPUT:

Exploring Swing

c) The swing Buttons Jbutton

- The JButton is a concrete subclass of AbstractButton which is a subclass of JComponent. The counterpart of JButton in AWT is Button. Perphaps the most widely used control is the push button.
- A Push button is a component that contains a label and that generates an event when it is pressed.
- **JButton defines four constructors**

  JButton() \\ constructs a button with no label.

  JButton(String Str**)** \\ constructs a button with a specified label.

  JButton(Icon i**)** \\ constructs a button with the icon I as button

  JButton(String str, icon i**)** \\ constructs a button with the icon I as button and the string as Label
- **JButton has several methods. Some of them are**

  void setText(String str) \\ sets the buttons label to the specified string

  void getText() \\ Returns the label of the button

  Icon getIcon() \\ Returns the icon of the button

  void setToolTipText(String str) \\ Sets the Tool Tip text to the

  void setMinemonic(char c) \\ sets Shortcut key to the utton

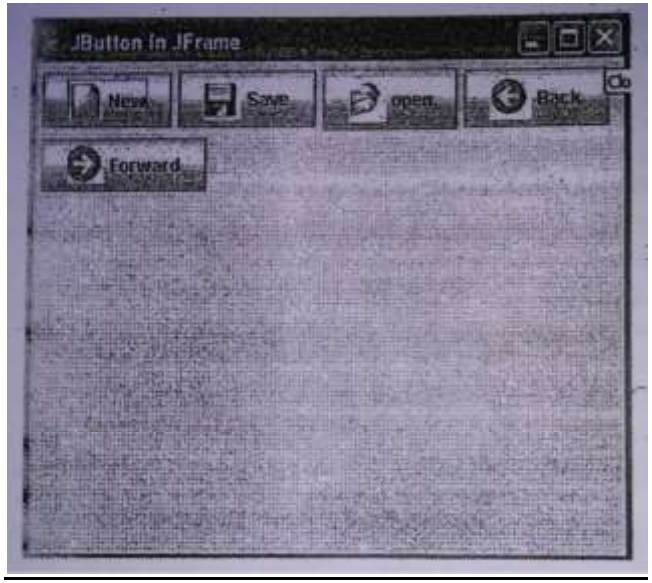
Exploring Swing

```
import java.awt.*;
import javax.swing.*;
class Btn extends JFrame
{
        JButton b1,b2,b3,b4,b5;
        Btn()
        {
                setSize(400,350);
                Container con=getContentPane();
                con.setLayout(new FlowLayout(FlowLayout.LEFT));
                Icon Img1=new ImageIcon("new.jpg");
                Icon Img2=new ImageIcon("save.jpg");
                Icon img3=new ImageIcon("open.jpg");
                Icon img4=new ImageIcon("back.jpg");
                Icon Img5=new ImageIcon("forward.jpg");
                b1=new JButton("New",Img1);
                b2=new JButton("Save",Img2);
                b3=new JButton("open",Img3);
                b4=new JButton("Back",Img4);
                b5=new JButton("Forward",Img5);
                b1.setToolTipText("New");
                b2.setForeground(Color.red);
                b3.setMnemonic('c'); // Shortcut key for save Button(ALT+C)
                con.add(b1);
                con.add(b2);
                con.add(b3);
                con.add(b4);
                con.add(b5);
                }
```

Exploring Swing

```
Class JButtonDemo
{
        public static void main(String args[])
       {
                        Btn ob=new Btn();
                        ob.setTitle("JButton in JFrame");
                        ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                ob.setVisible(true);
        }
}
```

OUTPUT:



JRadioButton

- Radio buttons are like check boxes.
- In Radio button, the selection is displayed in a round graphics.
- Radio buttons are generally used to represent a collection of mutually exclusive options i.e, out of several options,only one will be selected state and all the remaining are in deselected state.
- The radio buttons are created using JRadioButton class, which is subclass of JToggleButton.
- The JRadioButton must be placed in a Button Group.
- The Button group is created using the ButtonGroup class which has no argument constructor.
- After creating JRadioButton, the radio buttons are to be placed to the ButtonGroup using add() method.
- If the JRadio buttons are not grouped using Buttongroup, then each radio button will behave exactly like JCheckBox.
- JRadioButton generates ActionEvent, ItemEvent and ChangeEvent

Exploring Swing

JRadioButton defines several constructors

1) JRadioButton() // Creates a radio button without any label; the Radio Buton is set to deselected state
2) JRadioButton(String str)//Creates a radio button with the string str as label; the ardio button is set to deselected state
3) JRadioButton(String str, Boolean state)//Creates a radio button with the string str as label; the radio button is set to the specified state
4) JRadioButton(Icon i)// Creates a radio button using the icon I; the radio button is set to deselected state JRadioButton(Icon I, Boolean state)//Creates a radio button using the icon i;the radio button is set to deselected state.
5) JRadioButton(String str,Icon i)// Creates a radio button using the icon I with the string str as label
6) JRadioButton(String str, Icon I, Boolean state)//Creates a radio button using the Icon I with the string str set as label; the radio button is set to the specified state

Exploring Swing

```
import java.awt.*; import javax.swing.*;
class RButton extends JFrame
{
        JRadioButton rb1,rb2,rb3,rb4,mb,fb;
        ButtonGroup rbg;
        JLabel l1,l2;
        RButton()
                { setSize(600,300);
                Container c = getContentPane();
                c.setLayout(new FlowLayout());
                l1 = new JLabel("Buttons to be added");
                 l2 = new JLabel("Buttons Not added to group");
                mb = new JRadioButton("Male");
                fb = new JRadioButton("Female");
                rb1= new JRadioButton("Times New Roman");
                rb2 = new JRadioButton("Courier");
                rb3 = new JRadioButton("Terminal");
                rb4 = new JRadioButton("Arial");
                rbg = new ButtonGroup(); c.add(l1);
                c.add(rb1);
                c.add(rb2);
                c.add(rb3);
                c.add(rb4);
                rbg.add(rb1);
                rbg.add(rb2);
                rbg.add(rb3);
                rbg.add(rb4);
                c.add(l2);
                c.add(mb);
```

Exploring Swing

```
class JRadioButtonDemo
{
public static void main(String args[])
{
RButton ob = new RButton(); ob.setTitle("JRadioButton in Jframe");
ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ob.setVisible(true);
}
}
```

JCheckBox

The JCheckBox class provides the functionality of a check box. Its immediate superclass is JToggleButton, which provides support for two-state buttons.

JCheckBox defines the following constructors

JCheckBox() //Creates an initially unselected check box button with no text, no icon.

JCheckBox(Icon icon)//Creates an initially unselected check box with an icon.

JCheckBox(String text)//Creates an initially unselected check box with text.

JCheckBox(Icon icon, boolean selected)//Creates a check box with an icon and specifies whether or not it is initially selected.

Exploring Swing

```
import java.awt.*; import javax.swing.*;
class Check extends JFrame
{
        JCheckBox r,s,m;
        JLabel l1,l2;
        Check()
                { setSize(600,300);
                Container c = getContentPane();
                c.setLayout(new FlowLayout());
                 l1 = new JLabel("Hobbies");
                r = new JCheckBox("Reading");
                s = new JCheckBox("Singing");
                m = new JCheckBox("Listening Music");
                c.add(l1);
                c.add(r);
                c.add(s);
                c.add(m);
        }
}
class JCheckBoxDemo
{
        public static void main(String args[])
        {
                Check ob = new Check(); ob.setTitle("JCheckBox in Jframe");
                ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); ob.setVisible(true);
        }
}
```

OUTPUT


JCheckBox in Jframe
Hobbies
Reading
Singing
Listening Music

Exploring Swing

JComoBox

- The user can select a single item only. A combo box is a visual Swing graphical component that gives popup list when clicked.
- It is the combination of JList and JTextField.
- In Combo box, only one item is visible at a time. A Popup menu displays the choices a user can select from.
- In JList, the items cannot be edited, but in combo box, the items can be edited by setting the JComboBox editable.
- JCombobox defines the following constructors

JComboBox() \\ Creates empty Combo Box

JComboBox(object[] arr) \\ Creates a combo box taking the items from the specified Object Array

String 1st=("India","America","germany");

Eg:

JComboBoxbox=newJComboBox(1st);

JComboBox(Vector v)\\ Creates a combo box taking the items from the specified vector

Exploring Swing

JComboBox has so many methods. Some of them are

Void addItem(Object obj) \\ adds the specified object to the list

Eg:box.addItem("Japan");

Object getSelectItem() \\Returns the currently selected item

Eg: Object obj=box.getSelectedItem();

Int getSelectIndex() \\ Returns the index of item in the list

Eg: int I=box.getSelectedIndex();

Int getItemCount() \\ Returns the number of items in the list

Eg: int I=box.getItemCount();

Boolean isEditable() \\ Returns a Boolean specifying whether the combobox items are Editable or not

Eg: Boolean x=box.isEditable();

Void removeItem(Object ob) \\ Removes the specified item from the list

Eg: box.removeItem("germany");

Exploring Swing

```
import java.awt.*;
import javax.swing.*;
class cmbbox extends JFrame
{
                JComboBox box1,box2;
                String str[]={"Andhra Pradesh","Tamil Nadu","Karnataka"};
                 cmbbox()
                 {
                                  Container c=getContentPane();
                                 c.setLayout(new FlowLayout());
                                 box1=new JComboBox();
                                 box2=new JComboBox(str);
                                  JLabel I1=new JLabel("Countries");
                                  JLabel I2=new JLabel("States");
                                 box1.addItem("India");
                                 box1.addItem("germany");
                                 box1.addItem("Japan");
                                 c.add(I1);
                               c.add(box1);
                               c.add(I2);
                               c.add(box2);

                }
}
class JComboDemo
{
public static void main(String args[])
{
     cmbbox ob=new cmbbox();       ob.setTitle("JComboBox in JFrame");   ob.setSize(400,300);
 ob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   ob.setVisible(true);
```

Exploring Swing

String class Methods

No.	Method	Description
1	char charAt(int index)	returns char value for the particular index
2	int length()	returns string length
3	String substring(int beginIndex)	returns substring for given begin index.
4	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.
5	boolean equals(Object another)	checks the equality of string with the given object.
6	String concat(String str)	concatenates the specified string.
7	String replace(char old, char new)	replaces all occurrences of the specified char value.
8	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
9	String[] split(String regex)	returns a split string matching regex.
10	int indexOf(int ch)	returns the specified char value index.
11	int indexOf(int ch, int fromIndex)	returns the specified char value index starting with given index.
12	int indexOf(String substring)	returns the specified substring index.
13	String toLowerCase()	returns a string in lowercase.
14	String toUpperCase()	returns a string in uppercase.
15	String trim()	removes beginning and ending spaces of this string.

String class Methods

The string charAt() method returns a character at specified index.

```
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
```

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

```
String s="Sachin";
System.out.println(s.toUpperCase());//SACHIN
System.out.println(s.toLowerCase());//sachin
System.out.println(s);//Sachin(no change in original)
```

The string trim() method eliminates white spaces before and after string.

```
String s="  Sachin  ";
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
```

Java String startsWith() and endsWith() method

```
String s="Sachin";
 System.out.println(s.startsWith("Sa"));//true
 System.out.println(s.endsWith("n"));//true
```

The string length() method returns length of the string.

```
String s="Sachin";
System.out.println(s.length());//6
```

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

```
String s1="Java is a programming language. Java is a platform. Java is an Island.";
String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
System.out.println(replaceString);
```

StringBuffer class and its Methods

Java StringBuffer class is used to create mutable (modifiable) string.
The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

```
class StringBufferExample
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello ");
                sb.append("Java");//now original string is changed
                System.out.println(sb);//prints Hello Java
        }
}
```

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

```
class StringBufferExample2
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello ");
                sb.insert(1,"Java");//now original string is changed
                System.out.println(sb);//prints HJavaello
        }
}
```

StringBuffer class and its Methods

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
class StringBufferExample3
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello");
                sb.replace(1,3,"Java");
                System.out.println(sb);//prints HJavalo
        }
}
```

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
class StringBufferExample4
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello");
                sb.delete(1,3);
                System.out.println(sb);//prints Hlo
        }
}
```

StringBuffer class and its Methods

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5
{
        public static void main(String args[])
        {
                StringBuffer sb=new StringBuffer("Hello");
                sb.reverse();
                System.out.println(sb);//prints olleH
        }
}
```

To Check Given String is Palindrome or Not

```
import java.io.*;
class Palin
{
        public static void main(String args[]) throws Exception
        {
                String str;
                BufferedReader br;
                br=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter a String:");
                str=br.readLine();
                StringBuffer str1=new StringBuffer(str);
                str1.reverse();
                String rev=str1.toString();
                if(rev.compareTo(str)==0)
                System.out.println("The Given String" + str+ "  is Palindrome");
                else
                System.out.println("The Given String" +str +"  is Not Palindrome");
        }
}
```

To Check Given String is Palindrome or Not

OUTPUT :

```
Z:\JAVA>javac Palin.java
Z:\JAVA>java Palin
Enter a String:
MADAM
The Given String MADAM  is Palindrome

Z:\JAVA>javac Palin.java
Z:\JAVA>java Palin
Enter a String:
SITAMS
The Given String SITAMS  is Not Palindrome
```