CHAPTER 13

JAVA SWING

Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu etc.

Swing features

- **Light Weight** - Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich controls** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, table controls
- **Highly Customizable** - Swing controls can be customized in very easy way as visual apperance is independent of internal representation.
- **Pluggable look-and-feel**- SWING based GUI Application look and feel can be changed at run time based on available values.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser,

		tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Swing API architecture follows loosely based MVC architecture in the following manner.

- A Model represents component's data.
- View represents visual representation of the component's data.
- Controller takes the input from the user on the view and reflects the changes in Component's data.
- Swing component have Model as a seperate element and View and Controller part are clubbed in User Interface elements. Using this way, Swing has pluggable look-and-feel architecture.

What is JFC

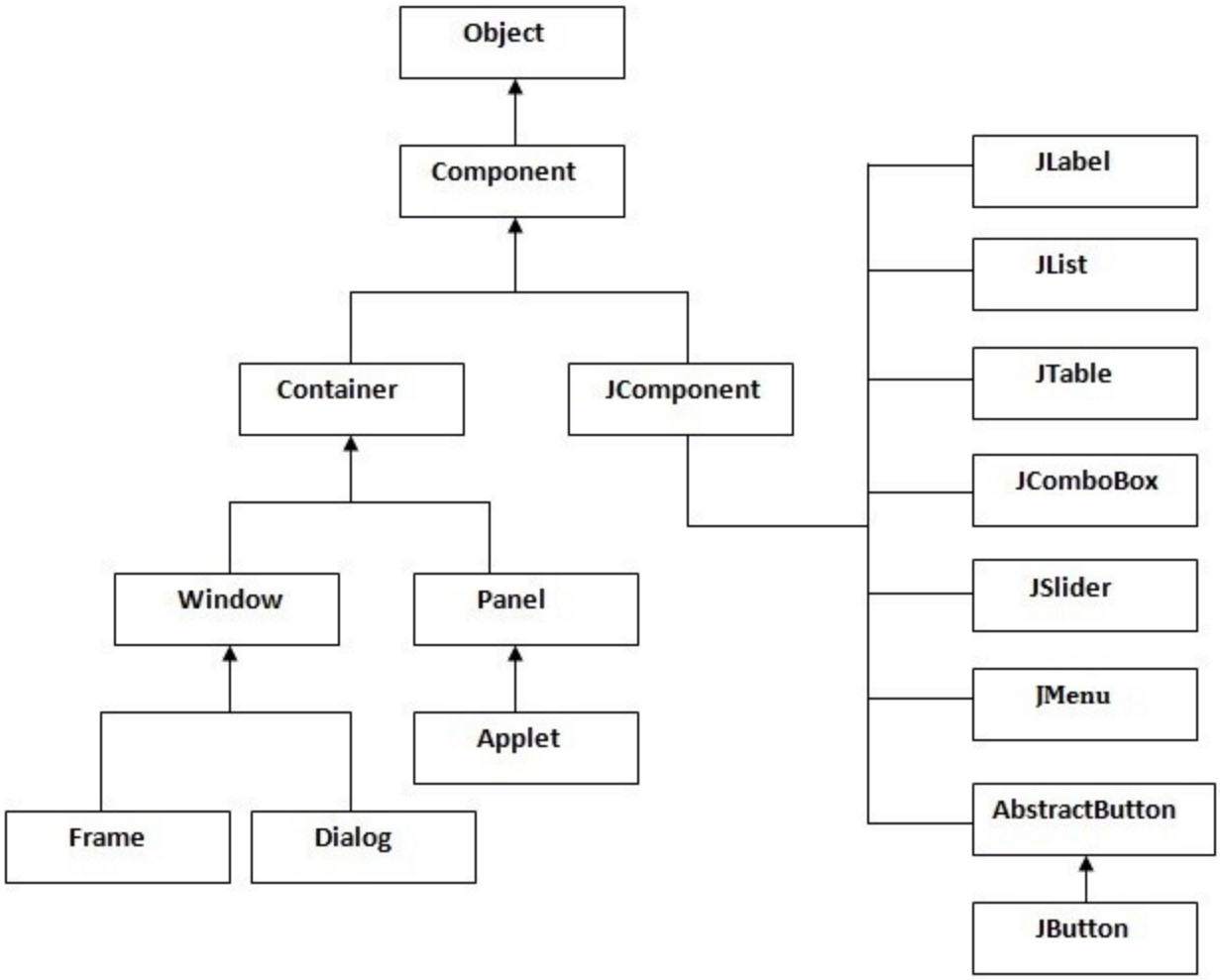
The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Do You Know ?

- How to display image on a button in swing?
- How to change the component color by choosing a color from ColorChooser ?
- How to display the digital watch in swing tutorial ?
- How to create a notepad in swing?
- How to create puzzle game and pic puzzle game in swing ?
- How to create tic tac toe game in swing ?

Hierarchy of javax.swing

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

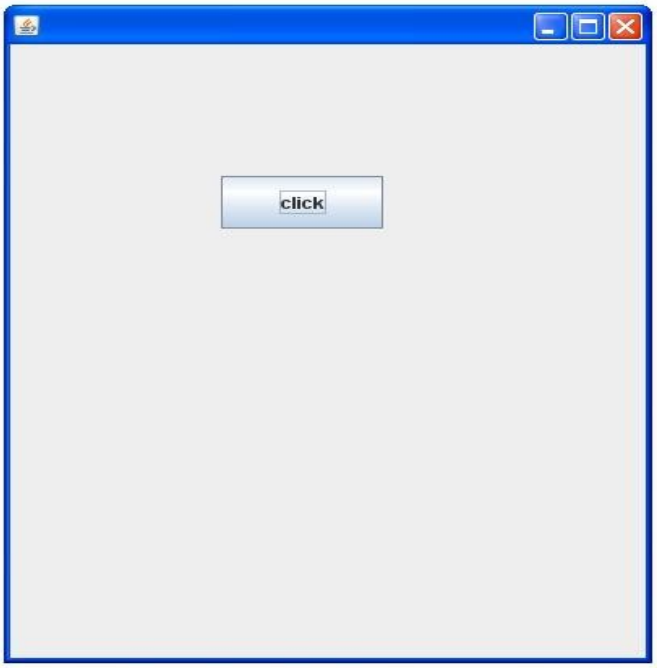
We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

File: FirstSwingExample.java

```
1. import javax.swing.*;
2. public class FirstSwingExample {
3. public static void main(String[] args) {
4. JFrame f=new JFrame();//creating instance of JFrame
5.
6. JButton b=new JButton("click");//creating instance of JButton
7. b.setBounds(130,100,100, 40);//x axis, y axis, width, height
8.
9. f.add(b);//adding button in JFrame
10.
11.f.setSize(400,500);//400 width and 500 height
12.f.setLayout(null);//using no layout managers
13.f.setVisible(true);//making the frame visible
14.}
15.}
```



Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
1. import javax.swing.*;
2. public class Simple {
3. JFrame f;
4. Simple(){
5. f=new JFrame();//creating instance of JFrame
6.
7. JButton b=new JButton("click");//creating instance of JButton
8. b.setBounds(130,100,100, 40);
9.
10.f.add(b);//adding button in JFrame
11.f.setSize(400,500);//400 width and 500 height
12.f.setLayout(null);//using no layout managers
13.f.setVisible(true);//making the frame visible
14.}
15.
16.public static void main(String[] args) {
```

```
17.new Simple();
18.}
19.}
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

File: Simple2.java

```
1. import javax.swing.*;
2. public class Simple2 extends JFrame{//inheriting JFrame
3. JFrame f;
4. Simple2(){
5. JButton b=new JButton("click");//create button
6. b.setBounds(130,100,100, 40);
7.
8. add(b);//adding button on frame
9. setSize(400,500);
10.setLayout(null);
11.setVisible(true);
12.}
13.public static void main(String[] args) {
14.new Simple2();
15.}}
```

JButton class:

The JButton class is used to create a button that have plateform-independent implementation.

Commonly used Constructors:

- **JButton():** creates a button with no text and icon.
- **JButton(String s):** creates a button with the specified text.

- **JButton(Icon i):** creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

1) public void setText(String s): is used to set specified text on button.
2) public String getText(): is used to return the text of the button.
3) public void setEnabled(boolean b): is used to enable or disable the button.
4) public void setIcon(Icon b): is used to set the specified Icon on the button.
5) public Icon getIcon(): is used to get the Icon of the button.
6) public void setMnemonic(int a): is used to set the mnemonic on the button.
7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

Note: The JButton class extends AbstractButton class.

Example of displaying image on the button:

```
1. import java.awt.event.*;
2. import javax.swing.*;
3.
4. public class ImageButton{
5. ImageButton(){
6. JFrame f=new JFrame();
7.
8.  ImageIcon I = new ImageIcon("b.jpg");
9. JButton b=new JButton(i);
10. b.setBounds(130,100,100, 40);
11.
12. f.add(b);
13.
14. f.setSize(300,400);
15. f.setLayout(null);
16. f.setVisible(true);
```

```
17.
18. f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19.
20.    }
21.
22. public static void main(String[] args) {
23.     new ImageButton();
24. }
25. }
```

JRadioButton class

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

Commonly used Constructors of JRadioButton class:

- **JRadioButton():** creates an unselected radio button with no text.
- **JRadioButton(String s):** creates an unselected radio button with specified text.
- **JRadioButton(String s, boolean selected):** creates a radio button with the specified text and selected status.

Commonly used Methods of AbstractButton class:

1) public void setText(String s): is used to set specified text on button.

2) public String getText(): is used to return the text of the button.

3) public void setEnabled(boolean b): is used to enable or disable the button.

4) public void setIcon(Icon b): is used to set the specified Icon on the button.

5) public Icon getIcon(): is used to get the Icon of the button.

6) public void setMnemonic(int a): is used to set the mnemonic on the button.

7) public void addActionListener(ActionListener a): is used to add the action listener to this object.

Note: The JRadioButton class extends the JToggleButton class that extends AbstractButton class.

Example of JRadioButton class:

```
import javax.swing.*;
public class Radio {
JFrame f;

Radio(){
f=new JFrame();

JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) FeMale");
r1.setBounds(50,100,70,30);
r2.setBounds(50,150,70,30);

ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);

f.add(r1);f.add(r2);

f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
   new Radio();
}
}
```

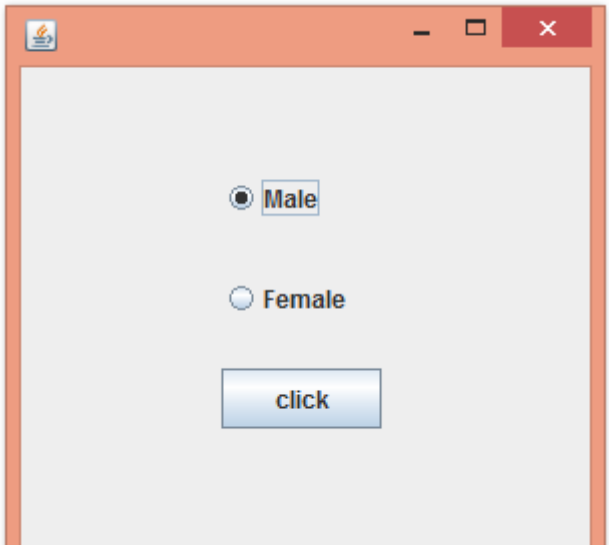
ButtonGroup class:

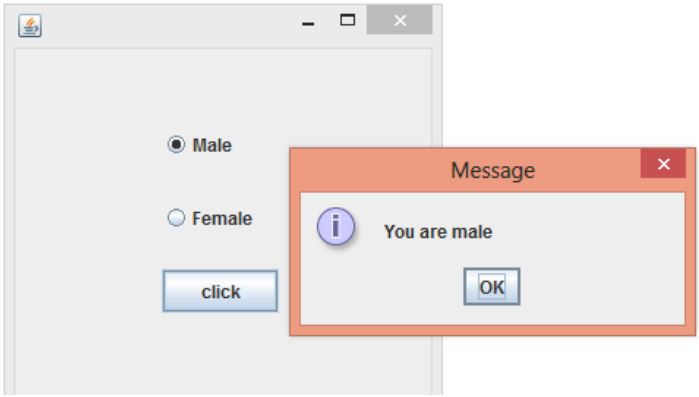
The ButtonGroup class can be used to group multiple radio buttons so that at a time only one button can be selected.

JRadioButton example with event handling

```
import javax.swing.*;
import java.awt.event.*;
class RadioExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
JButton b;
RadioExample(){

rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);

rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);

ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);

b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);

add(rb1);add(rb2);add(b);

setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are male");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are female");
```

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





JTextArea class :

The JTextArea class is used to create a text area. It is a multiline area that displays the plain text only.

Commonly used Constructors:

- **JTextArea():** creates a text area that displays no text initially.
- **JTextArea(String s):** creates a text area that displays specified text initially.

- **JTextArea(int row, int column):** creates a text area with the specified number of rows and columns that displays no text initially..
- **JTextArea(String s, int row, int column):** creates a text area with the specified number of rows and columns that displays specified text.

Commonly used methods of JTextArea class:

1) public void setRows(int rows): is used to set specified number of rows.

2) public void setColumns(int cols):: is used to set specified number of columns.

3) public void setFont(Font f): is used to set the specified font.

4) public void insert(String s, int position): is used to insert the specified text on the specified position.

5) public void append(String s): is used to append the given text to the end of the document.

Example of JTextField class:

```
1. import java.awt.Color;
2. import javax.swing.*;
3. 
4. public class TArea {
5.     JTextArea area;
6.     JFrame f;
7.     TArea(){
8.     f=new JFrame();
9. 
10.    area=new JTextArea(300,300);
11.    area.setBounds(10,30,300,300);
12. 
13.    area.setBackground(Color.black);
14.    area.setForeground(Color.white);
15. 
16.    f.add(area);
17. 
18.    f.setSize(400,400);
```

```
19.    f.setLayout(null);
20.    f.setVisible(true);
21. }
22.    public static void main(String[] args) {
23.        new TArea();
24.    }
25. }
```

JList Class

The class JList is a component which displays a list of objects and allows the user to select one or more items. A separate model, ListModel, maintains the contents of the list.

Field

Following are the fields for **javax.swing.JList** class −

- **static int HORIZONTAL_WRAP** − Indicates a "newspaper style" layout with cells flowing horizontally then vertically.
- **static int VERTICAL** − Indicates a vertical layout of cells, in a single column; the default layout.
- **static int VERTICAL_WRAP** − Indicates a "newspaper style" layout with cells flowing vertically then horizontally.

Constructors

S.N.	Constructor & Description
1	**JList()** Constructs a JList with an empty, read-only, model.
2	**JList(ListModel dataModel)** Constructs a JList that displays elements from the specified, non-null, model.
3	**JList(Object[] listData)**

	Constructs a JList that displays the elements in the specified array.
4	**JList(Vector<?> listData)** Constructs a JList that displays the elements in the specified Vector.

Class methods

S.N.	Method & Description
1	**void addListSelectionListener(ListSelectionListener listener)** Adds a listener to the list, to be notified each time a change to the selection occurs; the preferred way of listening for selection state changes.
2	**void clearSelection()** Clears the selection; after calling this method, isSelectionEmpty will return true.
3	**int getMaxSelectionIndex()** Returns the largest selected cell index, or -1 if the selection is empty.
4	**int getMinSelectionIndex()** Returns the smallest selected cell index, or -1 if the selection is empty.
5	**int getSelectedIndex()** Returns the smallest selected cell index; the selection when only a single item is selected in the list.
6	**int[] getSelectedIndices()** Returns an array of all of the selected indices, in increasing order.
7	**Object[] getSelectedValues()** Returns an array of all the selected values, in increasing order based on their indices in the list.
8	**int getSelectionMode()** Returns the current selection mode for the list.

9	**boolean isSelectedIndex(int index)** Returns true if the specified index is selected, else false.
10	**boolean isSelectionEmpty()** Returns true if nothing is selected, else false.
11	**void setSelectedIndex(int index)** Selects a single cell.

JComboBox class:

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

Commonly used Constructors of JComboBox class:

JComboBox()

JComboBox(Object[] items)

JComboBox(Vector<?> items)

Commonly used methods of JComboBox class:

1) public void addItem(Object anObject): is used to add an item to the item list.

2) public void removeItem(Object anObject): is used to delete an item to the item list.

3) public void removeAllItems(): is used to remove all the items from the list.

4) public void setEditable(boolean b): is used to determine whether the JComboBox is editable.

5) public void addActionListener(ActionListener a): is used to add the ActionListener.

6) public void addItemListener(ItemListener i): is used to add the ItemListener.

Example of JComboBox class:

```
1. import javax.swing.*;
2. public class Combo {
3. JFrame f;
4. Combo(){
5.     f=new JFrame("Combo ex");
6.
7.     String country[]={"India","Aus","U.S.A","England","Newzeland"};
8.
9.     JComboBox cb=new JComboBox(country);
10.    cb.setBounds(50, 50,90,20);
11.    f.add(cb);
12.
13.    f.setLayout(null);
14.    f.setSize(400,500);
15.    f.setVisible(true);
16.
17. }
18. public static void main(String[] args) {
19.    new Combo();
20.
21. }
22. }
```

JList Example

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;
```

```
   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new
SwingControlDemo();
      swingControlDemo.showListDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showListDemo(){

      headerLabel.setText("Control in action: JList");

      final DefaultListModel fruitsName = new
DefaultListModel();

      fruitsName.addElement("Apple");
      fruitsName.addElement("Grapes");
      fruitsName.addElement("Mango");
      fruitsName.addElement("Peer");

      final JList fruitList = new JList(fruitsName);
```

```
fruitList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      fruitList.setSelectedIndex(0);
      fruitList.setVisibleRowCount(3);

      JScrollPane fruitListScrollPane = new
JScrollPane(fruitList);

      final DefaultListModel vegName = new DefaultListModel();

      vegName.addElement("Lady Finger");
      vegName.addElement("Onion");
      vegName.addElement("Potato");
      vegName.addElement("Tomato");

      final JList vegList = new JList(vegName);
      vegList.setSelectionMode(
         ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
      vegList.setSelectedIndex(0);
      vegList.setVisibleRowCount(3);

      JScrollPane vegListScrollPane = new JScrollPane(vegList);

      JButton showButton = new JButton("Show");

      showButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            String data = "";
            if (fruitList.getSelectedIndex() != -1) {
               data = "Fruits Selected: " +
fruitList.getSelectedValue();
               statusLabel.setText(data);
            }
            if(vegList.getSelectedIndex() != -1){
               data += " Vegetables selected: ";
               for(Object
vegetable:vegList.getSelectedValues()){
                  data += vegetable + " ";
               }
            }
            statusLabel.setText(data);
         }
      });

      controlPanel.add(fruitListScrollPane);
      controlPanel.add(vegListScrollPane);
      controlPanel.add(showButton);
```

```
      mainFrame.setVisible(true);
   }
}
```

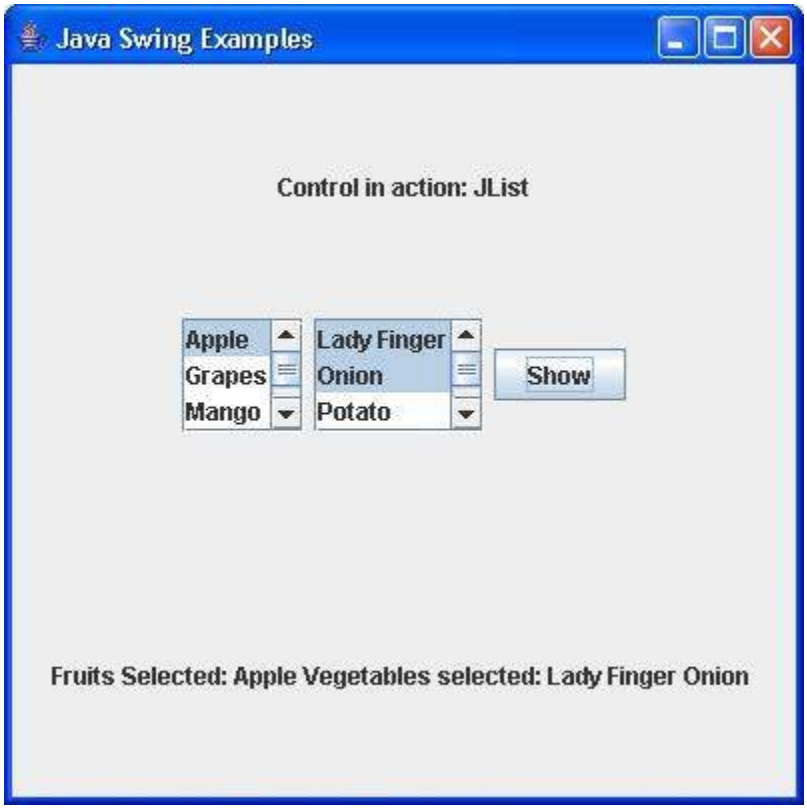
Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

If no error comes that means compilation is successful. Run the program using following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output



JTable class (Swing Tutorial):

The JTable class is used to display the data on two dimensional tables of cells.

Commonly used Constructors of JTable class:

- **JTable():** creates a table with empty cells.
- **JTable(Object[][] rows, Object[] columns):** creates a table with the specified data.

Example of JTable class:

```
1. import javax.swing.*;
2. public class MyTable
3. {
4.     JFrame f;
5.     MyTable(){
6.     f=new JFrame();
7.
8.     String data[][]={ {"101","Amit","670000"},
9.                       {"102","Jai","780000"},
10.                       {"101","Sachin","700000"}};
11.          String column[]={"ID","NAME","SALARY"};
12.
13.    JTable jt=new JTable(data,column);
14. jt.setBounds(30,40,200,300);
15.
16.          JScrollPane sp=new JScrollPane(jt);
17.    f.add(sp);
18.
19.    f.setSize(300,400);
20. //  f.setLayout(null);
21.    f.setVisible(true);
22. }
```

```
23. public static void main(String[] args) {
24.     new MyTable();
25. }
26. }
```

JColorChooser class:

The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

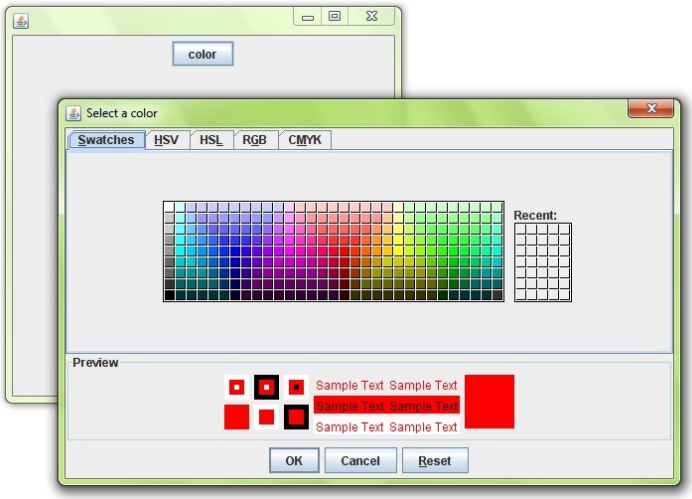
Commonly used Constructors of JColorChooser class:

- **JColorChooser():** is used to create a color chooser pane with white color initially.
- **JColorChooser(Color initialColor):** is used to create a color chooser pane with the specified color initially.

Commonly used methods of JColorChooser class:

public static Color showDialog(Component c, String title, Color initialColor): is used to show the color-chooser dialog box.

Example of JColorChooser class:



```
1. import java.awt.event.*;
2. import java.awt.*;
3. import javax.swing.*;
4.
5. public class JColorChooserExample extends JFrame implements ActionListener{
6. JButton b;
7. Container c;
8.
9. JColorChooserExample(){
10.    c=getContentPane();
11.    c.setLayout(new FlowLayout());
12.
13.    b=new JButton("color");
14.    b.addActionListener(this);
15.
16.    c.add(b);
17. }
18.
19. public void actionPerformed(ActionEvent e) {
20. Color initialcolor=Color.RED;
21. Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
22. c.setBackground(color);
23. }
24.
25. public static void main(String[] args) {
26.    JColorChooserExample ch=new JColorChooserExample();
27.    ch.setSize(400,400);
28.    ch.setVisible(true);
29.    ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
30. }
31. }
```

JFileChooser Class

The class JFileChooser **is a component which provides a simple mechanism for the user to choose a file.**

Field

Following are the fields for **javax.swing.JFileChooser** class:

- **static String ACCEPT_ALL_FILE_FILTER_USED_CHANGED_PROPERTY** -- Identifies whether a the AcceptAllFileFilter is used or not.
- **protected AccessibleContext accessibleContext**
- **static String ACCESSORY_CHANGED_PROPERTY** --Says that a different accessory component is in use (for example, to preview files).
- **static String APPROVE_BUTTON_MNEMONIC_CHANGED_PROPERTY** -- Identifies change in the mnemonic for the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TEXT_CHANGED_PROPERTY** -- Identifies change in the text on the approve (yes, ok) button.
- **static String APPROVE_BUTTON_TOOL_TIP_TEXT_CHANGED_PROPERTY** --Identifies change in the tooltip text for the approve (yes, ok) button.
- **static int APPROVE_OPTION** --Return value if approve (yes, ok) is chosen.
- **static String APPROVE_SELECTION** --Instruction to approve the current selection (same as pressing yes or ok).
- **static int CANCEL_OPTION** --Return value if cancel is chosen.
- **static String CANCEL_SELECTION** --Instruction to cancel the current selection.
- **static String CHOOSABLE_FILE_FILTER_CHANGED_PROPERTY** --Identifies a change in the list of predefined file filters the user can choose from.
- **static String CONTROL_BUTTONS_ARE_SHOWN_CHANGED_PROPERTY** -- Instruction to display the control buttons.
- **static int CUSTOM_DIALOG** --Type value indicating that the JFileChooser supports a developer-specified file operation.
- **static String DIALOG_TITLE_CHANGED_PROPERTY** --Identifies a change in the dialog title.
- **static String DIALOG_TYPE_CHANGED_PROPERTY** --Identifies a change in the type of files displayed (files only, directories only, or both files and directories).
- **static int DIRECTORIES_ONLY** --Instruction to display only directories.
- **static String DIRECTORY_CHANGED_PROPERTY** --Identifies user's directory change.
- **static int ERROR_OPTION** --Return value if an error occured.

- **static String FILE_FILTER_CHANGED_PROPERTY** --User changed the kind of files to display.
- **static String FILE_HIDING_CHANGED_PROPERTY** --Identifies a change in the display-hidden-files property.
- **static String FILE_SELECTION_MODE_CHANGED_PROPERTY** --Identifies a change in the kind of selection (single, multiple, etc.).
- **static String FILE_SYSTEM_VIEW_CHANGED_PROPERTY** --Says that a different object is being used to find available drives on the system.
- **static String FILE_VIEW_CHANGED_PROPERTY** --Says that a different object is being used to retrieve file information.
- **static int FILES_AND_DIRECTORIES** --Instruction to display both files and directories.
- **static int FILES_ONLY** --Instruction to display only files.
- **static String MULTI_SELECTION_ENABLED_CHANGED_PROPERTY** --Enables multiple-file selections.
- **static int OPEN_DIALOG** --Type value indicating that the JFileChooser supports an "Open" file operation.
- **static int SAVE_DIALOG** --Type value indicating that the JFileChooser supports a "Save" file operation.
- **static String SELECTED_FILE_CHANGED_PROPERTY** --Identifies change in user's single-file selection.
- **static String SELECTED_FILES_CHANGED_PROPERTY** --Identifies change in user's multiple-file selection.

Class constructors

S.N.	Constructor & Description
1	**JFileChooser()** Constructs a JFileChooser pointing to the user's default directory.
2	**JFileChooser(File currentDirectory)** Constructs a JFileChooser using the given File as the path.
3	**JFileChooser(File currentDirectory, FileSystemView fsv)** Constructs a JFileChooser using the given current directory and FileSystemView.
4	**JFileChooser(FileSystemView fsv)**

	Constructs a JFileChooser using the given FileSystemView.
5	**JFileChooser(String currentDirectoryPath)** Constructs a JFileChooser using the given path.
6	**JFileChooser(String currentDirectoryPath, FileSystemView fsv)** Constructs a JFileChooser using the given current directory path and FileSystemView.

Class methods

S.N.	Method & Description
1	**boolean accept(File f)** Returns true if the file should be displayed.
2	**void addActionListener(ActionListener l)** Adds an ActionListener to the file chooser.
3	**void addChoosableFileFilter(FileFilter filter)** Adds a filter to the list of user choosable file filters.
4	**void approveSelection()** Called by the UI when the user hits the Approve button (labeled "Open" or "Save", by default).
5	**void cancelSelection()** Called by the UI when the user chooses the Cancel button.
6	**void changeToParentDirectory()** Changes the directory to be set to the parent of the current directory.
7	**protected JDialog createDialog(Component parent)**

	Creates and returns a new JDialog wrapping this centered on the parent in the parent's frame.
8	**void ensureFileIsVisible(File f)** Makes sure that the specified file is viewable, and not hidden.
9	**protected void fireActionPerformed(String command)** Notifies all listeners that have registered interest for notification on this event type.
10	**FileFilter getAcceptAllFileFilter()** Returns the AcceptAll file filter.
11	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext associated with this JFileChooser.
12	**JComponent getAccessory()** Returns the accessory component.
13	**ActionListener[]getActionListeners()** Returns an array of all the action listeners registered on this file chooser.
14	**int getApproveButtonMnemonic()** Returns the approve button's mnemonic.
15	**String getApproveButtonText()** Returns the text used in the ApproveButton in the FileChooserUI.
16	**String getApproveButtonToolTipText()** Returns the tooltip text used in the ApproveButton.
17	**FileFilter[] getChoosableFileFilters()** Gets the list of user choosable file filters.
18	**boolean getControlButtonsAreShown()** Returns the value of the controlButtonsAreShown property.
19	**File getCurrentDirectory()**

	Returns the current directory.
20	**String getDescription(File f)** Returns the file description.
21	**String getDialogTitle()** Gets the string that goes in the JFileChooser's titlebar.
22	**int getDialogType()** Returns the type of this dialog.
23	**boolean getDragEnabled()** Gets the value of the dragEnabled property.
24	**FileFilter getFileFilter()** Returns the currently selected file filter.
25	**int getFileSelectionMode()** Returns the current file-selection mode.
26	**FileSystemView getFileSystemView()** Returns the file system view.
27	**FileView getFileView()** Returns the current file view.
28	**Icon getIcon(File f)** Returns the icon for this file or type of file, depending on the system.
29	**String getName(File f)** Returns the filename.
30	**File getSelectedFile()** Returns the selected file.
31	**File[] getSelectedFiles()** Returns a list of selected files if the file chooser is set to allow multiple

	selection.
32	**String getTypeDescription(File f)** Returns the file type.
33	**FileChooserUI getUI()** Gets the UI object which implements the L&F for this component.
34	**String getUIClassID()** Returns a string that specifies the name of the L&F class that renders this component.
35	**boolean isAcceptAllFileFilterUsed()** Returns whether the AcceptAll FileFilter is used.
36	**boolean isDirectorySelectionEnabled()** Convenience call that determines if directories are selectable based on the current file selection mode.
37	**boolean isFileHidingEnabled()** Returns true if hidden files are not shown in the file chooser; otherwise, returns false.
38	**boolean isFileSelectionEnabled()** Convenience call that determines if files are selectable based on the current file selection mode.
39	**boolean isMultiSelectionEnabled()** Returns true if multiple files can be selected.
40	**boolean isTraversable(File f)** Returns true if the file (directory) can be visited.
41	**protected String paramString()** Returns a string representation of this JFileChooser.
42	**void removeActionListener(ActionListener l)** Removes an ActionListener from the file chooser.

43	**boolean removeChoosableFileFilter(FileFilter f)** Removes a filter from the list of user choosable file filters.
44	**void rescanCurrentDirectory()** Tells the UI to rescan its files list from the current directory.
45	**void resetChoosableFileFilters()** Resets the choosable file filter list to its starting state.
46	**void setAcceptAllFileFilterUsed(boolean b)** Determines whether the AcceptAll FileFilter is used as an available choice in the choosable filter list.
47	**void setAccessory(JComponent newAccessory)** Sets the accessory component.
48	**void setApproveButtonMnemonic(char mnemonic)** Sets the approve button's mnemonic using a character.
49	**void setApproveButtonMnemonic(int mnemonic)** Sets the approve button's mnemonic using a numeric keycode.
50	**void setApproveButtonText(String approveButtonText)** Sets the text used in the ApproveButton in the FileChooserUI.
51	**void setApproveButtonToolTipText(String toolTipText)** Sets the tooltip text used in the ApproveButton.
52	**void setControlButtonsAreShown(boolean b)** Sets the property that indicates whether the approve and cancel buttons are shown in the file chooser.
53	**void setCurrentDirectory(File dir)** Sets the current directory.
54	**void setDialogTitle(String dialogTitle)** Sets the string that goes in the JFileChooser window's title bar.

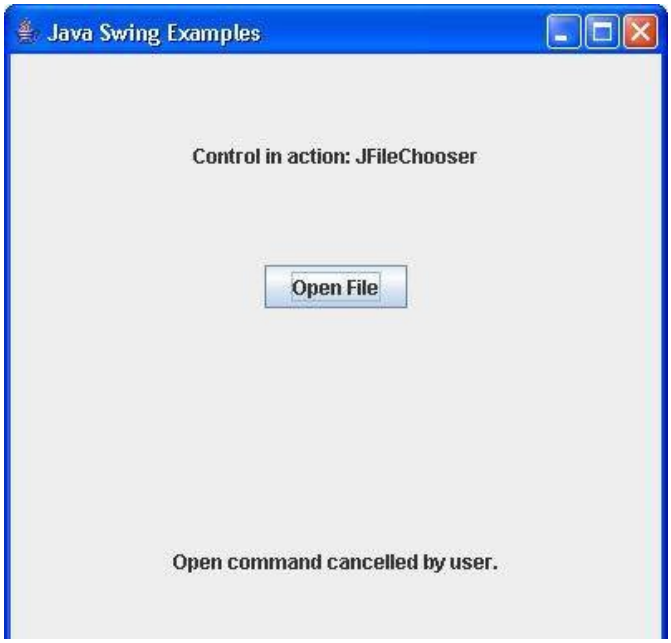
55	**void setDialogType(int dialogType)** Sets the type of this dialog.
56	**void setDragEnabled(boolean b)** Sets the dragEnabled property, which must be true to enable automatic drag handling (the first part of drag and drop) on this component.
57	**void setFileFilter(FileFilter filter)** Sets the current file filter.
58	**void setFileHidingEnabled(boolean b)** Sets file hiding on or off.
59	**void setFileSelectionMode(int mode)** Sets the JFileChooser to allow the user to just select files, just select directories, or select both files and directories.
60	**void setFileSystemView(FileSystemView fsv)** Sets the file system view that the JFileChooser uses for accessing and creating file system resources, such as finding the floppy drive and getting a list of root drives.
61	**void setFileView(FileView fileView)** Sets the file view to used to retrieve UI information, such as the icon that represents a file or the type description of a file.
62	**void setMultiSelectionEnabled(boolean b)** Sets the file chooser to allow multiple file selections.
63	**void setSelectedFile(File file)** Sets the selected file.
64	**void setSelectedFiles(File[] selectedFiles)** Sets the list of selected files if the file chooser is set to allow multiple selection.
65	**protected void setup(FileSystemView view)**

	Performs common constructor initialization and setup.
66	**int showDialog(Component parent, String approveButtonText)** Pops a custom file chooser dialog with a custom approve button.
67	**int showOpenDialog(Component parent)** Pops up an "Open File" file chooser dialog.
68	**int showSaveDialog(Component parent)** Pops up a "Save File" file chooser dialog.
69	**void updateUI()** Resets the UI property to a value from the current look and feel.

Example

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new
SwingControlDemo();
      swingControlDemo.showFileChooserDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
```

```
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showFileChooserDemo(){
      headerLabel.setText("Control in action: JFileChooser");

      final JFileChooser  fileDialog = new JFileChooser();
      JButton showFileDialogButton = new JButton("Open File");
      showFileDialogButton.addActionListener(new
ActionListener() {
         @Override
         public void actionPerformed(ActionEvent e) {
            int returnVal =
fileDialog.showOpenDialog(mainFrame);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
               java.io.File file = fileDialog.getSelectedFile();
               statusLabel.setText("File Selected :"
               + file.getName());
            }
            else{
               statusLabel.setText("Open command cancelled by
user." );
            }
         }
      });
      controlPanel.add(showFileDialogButton);
      mainFrame.setVisible(true);
   }
}
```



JProgressBar class:

The JProgressBar class is used to display the progress of the task.

Commonly used Constructors of JProgressBar class:

- **JProgressBar():** is used to create a horizontal progress bar but no string text.
- **JProgressBar(int min, int max):** is used to create a horizontal progress bar with the specified minimum and maximum value.
- **JProgressBar(int orient):** is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
- **JProgressBar(int orient, int min, int max):** is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used methods of JProgressBar class:

1) public void setStringPainted(boolean b): is used to determine whether string should be displayed.

2) public void setString(String s): is used to set value to the progress string.

3) public void setOrientation(int orientation): is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants..

4) public void setValue(int value): is used to set the current value on the progress bar.

Example of JProgressBar class:

```
1. import javax.swing.*;
2. public class MyProgress extends JFrame{
3. JProgressBar jb;
4. int i=0,num=0;
5.
6. MyProgress(){
7. jb=new JProgressBar(0,2000);
8. jb.setBounds(40,40,200,30);
9.
10. jb.setValue(0);
11. jb.setStringPainted(true);
12.
13. add(jb);
14. setSize(400,400);
15. setLayout(null);
16. }
17.
18. public void iterate(){
19. while(i<=2000){
20.  jb.setValue(i);
21.  i=i+20;
22.  try{Thread.sleep(150);}catch(Exception e){}
23. }
24. }
25. public static void main(String[] args) {
26.    MyProgress m=new MyProgress();
27.    m.setVisible(true);
28.    m.iterate();
29. }
```

```
30. }
```

JSlider class:

The JSlider is used to create the slider. By using JSlider a user can select a value from a specific range.

Commonly used Constructors of JSlider class:

- **JSlider():** creates a slider with the initial value of 50 and range of 0 to 100.
- **JSlider(int orientation):** creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
- **JSlider(int min, int max):** creates a horizontal slider using the given min and max.
- **JSlider(int min, int max, int value):** creates a horizontal slider using the given min, max and value.
- **JSlider(int orientation, int min, int max, int value):** creates a slider using the given orientation, min, max and value.

Commonly used Methods of JSlider class:

1) public void setMinorTickSpacing(int n): is used to set the minor tick spacing to the slider.

2) public void setMajorTickSpacing(int n): is used to set the major tick spacing to the slider.

3) public void setPaintTicks(boolean b): is used to determine whether tick marks are painted.

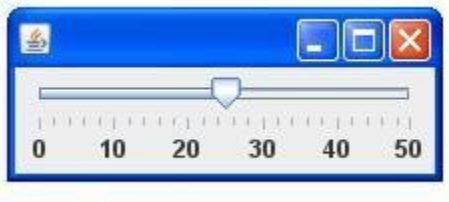
4) public void setPaintLabels(boolean b): is used to determine whether labels are painted.

5) public void setPaintTracks(boolean b): is used to determine whether track is painted.

Simple example of JSlider class:

```
1. import javax.swing.*;
2.
3. public class SliderExample1 extends JFrame{
4.
5. public SliderExample1() {
6. JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
7. JPanel panel=new JPanel();
8. panel.add(slider);
9.
10. add(panel);
11. }
12.
13. public static void main(String s[]) {
14. SliderExample1 frame=new SliderExample1();
15. frame.pack();
16. frame.setVisible(true);
17. }
18. }
```

Example of JSlider class that paints ticks:




```
1. import javax.swing.*;
2.
3. public class SliderExample extends JFrame{
4.
5. public SliderExample() {
6.
7. JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
8. slider.setMinorTickSpacing(2);
```

```
9.  slider.setMajorTickSpacing(10);
10.
11.slider.setPaintTicks(true);
12.slider.setPaintLabels(true);
13.
14.JPanel panel=new JPanel();
15.panel.add(slider);
16.add(panel);
17.}
18.
19.public static void main(String s[]) {
20.SliderExample frame=new SliderExample();
21.frame.pack();
22.frame.setVisible(true);
23.
24.}
25.}
```

JSpinner Class

The class **JSpinner** is a component which lets the user select a number or an object value from an ordered sequence using an input field.

Class constructors

S.N.	Constructor & Description
1	**JSpinner()** Constructs a spinner with an Integer SpinnerNumberModel with initial value 0 and no minimum or maximum limits.
2	**JSpinner(SpinnerModel model)** Constructs a complete spinner with pair of next/previous buttons and an editor for the SpinnerModel.

Class methods

S.N.	Method & Description
1	**void addChangeListener(ChangeListener listener)** Adds a listener to the list that is notified each time a change to the model occurs.
2	**void commitEdit()** Commits the currently edited value to the SpinnerModel.
3	**protected JComponent createEditor(SpinnerModel model)** This method is called by the constructors to create the JComponent that displays the current value of the sequence.
4	**protected void fireStateChanged()** Sends a ChangeEvent, whose source is this JSpinner, to each ChangeListener.
5	**AccessibleContext getAccessibleContext()** Gets the AccessibleContext for the JSpinner.
6	**ChangeListener[] getChangeListeners()** Returns an array of all the ChangeListeners added to this JSpinner with addChangeListener().
7	**JComponent getEditor()** Returns the component that displays and potentially changes the model's value.
8	**SpinnerModel getModel()** Returns the SpinnerModel that defines this spinners sequence of values.
9	**Object getNextValue()** Returns the object in the sequence that comes after the object returned by getValue().
10	**Object getPreviousValue()** Returns the object in the sequence that comes before the object returned by getValue().
11	**SpinnerUI getUI()**

	Returns the look and feel (L&F) object that renders this component.
12	**String getUIClassID()** Returns the suffix used to construct the name of the look and feel (L&F) class used to render this component.
13	**Object getValue()** Returns the current value of the model, typically this value is displayed by the editor.
14	**void removeChangeListener(ChangeListener listener)** Removes a ChangeListener from this spinner.
15	**void setEditor(JComponent editor)** Changes the JComponent that displays the current value of the SpinnerModel.
16	**void setModel(SpinnerModel model)** Changes the model that represents the value of this spinner.
17	**void setUI(SpinnerUI ui)** Sets the look and feel (L&F) object that renders this component.
18	**void setValue(Object value)** Changes current value of the model, typically this value is displayed by the editor.
19	**void updateUI()** Resets the UI property with the value from the current look and feel.

Example

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

public class SwingControlDemo {

   private JFrame mainFrame;
```

```
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new
SwingControlDemo();
      swingControlDemo.showSpinnerDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());

      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showSpinnerDemo(){
      headerLabel.setText("Control in action: JSpinner");
      SpinnerModel spinnerModel =
         new SpinnerNumberModel(10, //initial value
            0, //min
            100, //max
            1);//step
      JSpinner spinner = new JSpinner(spinnerModel);
      spinner.addChangeListener(new ChangeListener() {
         public void stateChanged(ChangeEvent e) {
```

```
            statusLabel.setText("Value : "
            + ((JSpinner)e.getSource()).getValue());
         }
      });
      controlPanel.add(spinner);
      mainFrame.setVisible(true);
   }
}
```

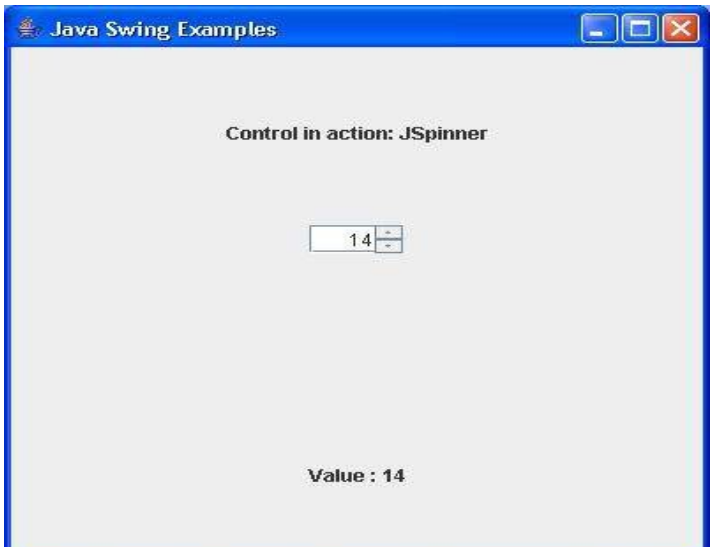
Compile the program using command prompt. Go to **D:/ > SWING** and type the following command.

```
D:\SWING>javac com\tutorialspoint\gui\SwingControlDemo.java
```

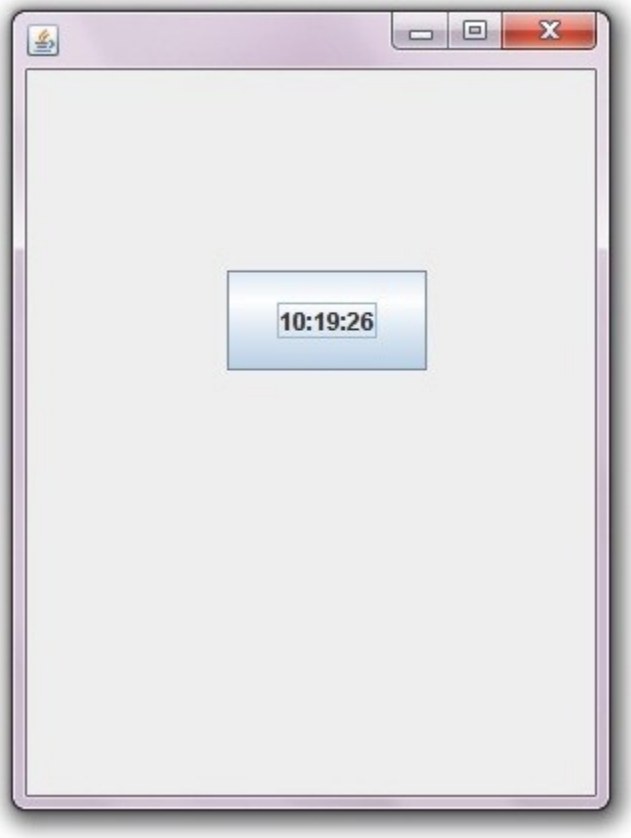
If no error comes that means compilation is successful. Run the program using following command.

```
D:\SWING>java com.tutorialspoint.gui.SwingControlDemo
```

Verify the following output



Example of digital clock in swing:



```
1. import javax.swing.*;
2. import java.awt.*;
3. import java.text.*;
4. import java.util.*;
5. public class DigitalWatch implements Runnable{
6. JFrame f;
7. Thread t=null;
8. int hours=0, minutes=0, seconds=0;
9. String timeString = "";
10. JButton b;
11.
12. DigitalWatch(){
13.    f=new JFrame();
14.
15.    t = new Thread(this);
16.       t.start();
17.
18.    b=new JButton();
19.       b.setBounds(100,100,100,50);
20.
21.    f.add(b);
22.    f.setSize(300,400);
23.    f.setLayout(null);
24.    f.setVisible(true);
25. }
26.
27. public void run() {
28.     try {
29.       while (true) {
30.
31.          Calendar cal = Calendar.getInstance();
32.          hours = cal.get( Calendar.HOUR_OF_DAY );
33.          if ( hours > 12 ) hours -= 12;
34.          minutes = cal.get( Calendar.MINUTE );
35.          seconds = cal.get( Calendar.SECOND );
36.
37.          SimpleDateFormat formatter = new SimpleDateFormat("hh:mm:ss");
38.          Date date = cal.getTime();
39.          timeString = formatter.format( date );
40.
```

```
41.           printTime();
42.
43.           t.sleep( 1000 );  // interval given in milliseconds
44.         }
45.       }
46.       catch (Exception e) { }
47. }
48.
49. public void printTime(){
50. b.setText(timeString);
51. }
52.
53. public static void main(String[] args) {
54.    new DigitalWatch();
55.
56.
57. }
58. }
```

JOptionPane Class

The class **JOptionPane** is a component which provides standard methods to pop up a standard dialog box for a value or informs user of something.

Class declaration

Following is the declaration for **javax.swing.JOptionPane** class:

```
public class JOptionPane
   extends JComponent
      implements Accessible
```

Field

Following are the fields for **javax.swing.JOptionPane** class:

- **static int CANCEL_OPTION** -- Return value from class method if CANCEL is chosen.

- **static int CLOSED_OPTION** -- Return value from class method if user closes window without selecting anything, more than likely this should be treated as either a CANCEL_OPTION or NO_OPTION.
- **static int DEFAULT_OPTION** -- Type meaning Look and Feel should not supply any options -- only use the options from the JOptionPane.
- **static int ERROR_MESSAGE** -- Used for error messages.
- **static int OK_CANCEL_OPTION** -- Type used for showConfirmDialog.
- **static int NO_OPTION** -- Return value from class method if NO is chosen.
- **static int OK_OPTION** -- Return value form class method if OK is chosen.
- **protected int optionType** -- Option type, one of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION or OK_CANCEL_OPTION.
- **static int PLAIN_MESSAGE** -- No icon is used.
- **static int QUESTION_MESSAGE** -- Used for questions.
- **static int WARNING_MESSAGE** -- Used for warning messages.
- **static int YES_NO_CANCEL_OPTION** -- Type used for showConfirmDialog.
- **static int YES_NO_OPTION** -- Type used for showConfirmDialog.
- **static int YES_OPTION** -- Return value from class method if YES is chosen.

Class constructors

S.N.	Constructor & Description
1	**JOptionPane()** Creates a JOptionPane with a test message.
2	**JOptionPane(Object message)** Creates a instance of JOptionPane to display a message using the plain-message message type and the default options delivered by the UI.
3	**JOptionPane(Object message, int messageType)** Creates an instance of JOptionPane to display a message with the specified message type and the default options

4	**JOptionPane(Object message, int messageType, int optionType)** Creates an instance of JOptionPane to display a message with the specified message type and options.
5	**JOptionPane(Object message, int messageType, int optionType, Icon icon)** Creates an instance of JOptionPane to display a message with the specified message type, options, and icon.
6	**JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options)** Creates an instance of JOptionPane to display a message with the specified message type, icon, and options.
7	**JOptionPane(Object message, int messageType, int optionType, Icon icon, Object[] options, Object initialValue)** Creates an instance of JOptionPane to display a message with the specified message type, icon, and options, with the initially-selected option specified.

Class methods

S.N.	Method & Description
1	**JDialog createDialog(Component parentComponent, String title)** Creates and returns a new JDialog wrapping this centered on the parentComponent in the parentComponent's frame.
2	**JDialog createDialog(String title)** Creates and returns a new parentless JDialog with the specified title.
3	**Icon getIcon()** Returns the icon this pane displays.
4	**Object getInitialSelectionValue()** Returns the input value that is displayed as initially selected to the user.
5	**Object getInitialValue()**

	Returns the initial value.
6	**Object getInputValue()** Returns the value the user has input, if wantsInput is true.
7	**Object[] getOptions()** Returns the choices the user can make.
8	**int getOptionType()** Returns the type of options that are displayed.
9	**Object[] getSelectionValues()** Returns the input selection values.
10	**Object getValue()** Returns the value the user has selected.
11	**void setIcon(Icon newIcon)** Sets the icon to display.
12	**void setInitialSelectionValue(Object newValue)** Sets the input value that is initially displayed as selected to the user.
13	**void setInitialValue(Object newInitialValue)** Sets the initial value that is to be enabled -- the Component that has the focus when the pane is initially displayed.
14	**void setInputValue(Object newValue)** Sets the input value that was selected or input by the user.
15	**void setMessage(Object newMessage)** Sets the option pane's message-object.
16	**void setOptions(Object[] newOptions)** Sets the options this pane displays.
17	**void setValue(Object newValue)**

	Sets the value the user has chosen.
18	**static int showConfirmDialog(Component parentComponent, Object message)** Brings up a dialog with the options Yes, No and Cancel; with the title, Select an Option.
19	**static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType)** Brings up a dialog where the number of choices is determined by the optionType parameter.
20	**static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType)** Brings up a dialog where the number of choices is determined by the optionType parameter, where the messageType parameter determines the icon to display.
21	**static int showConfirmDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon)** Brings up a dialog with a specified icon, where the number of choices is determined by the optionType parameter.
22	**static String showInputDialog(Component parentComponent, Object message)** Shows a question-message dialog requesting input from the user parented to parentComponent.
23	**static String showInputDialog(Component parentComponent, Object message, Object initialSelectionValue)** Shows a question-message dialog requesting input from the user and parented to parentComponent.
24	**static String showInputDialog(Component parentComponent, Object message, String title, int essageType)** Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.

25	**static Object showInputDialog(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue)** Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
26	**static String showInputDialog(Object message)** Shows a question-message dialog requesting input from the user.
27	**static String showInputDialog(Object message, Object initialSelectionValue)** Shows a question-message dialog requesting input from the user, with the input value initialized to initialSelectionValue.
28	**static voidshowMessageDialog(Component parentComponent, Object message)** Brings up an information-message dialog titled "Message".
29	**static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType)** Brings up a dialog that displays a message using a default icon determined by the messageType parameter.
30	**static voidshowMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon)** Brings up a dialog displaying a message, specifying all parameters.
31	**static int showOptionDialog(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue)** Brings up a dialog with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.
32	**void updateUI()** Notification from the UIManager that the L&F has changed.
33	**static int showInternalOptionDialog(Component parentComponent,**

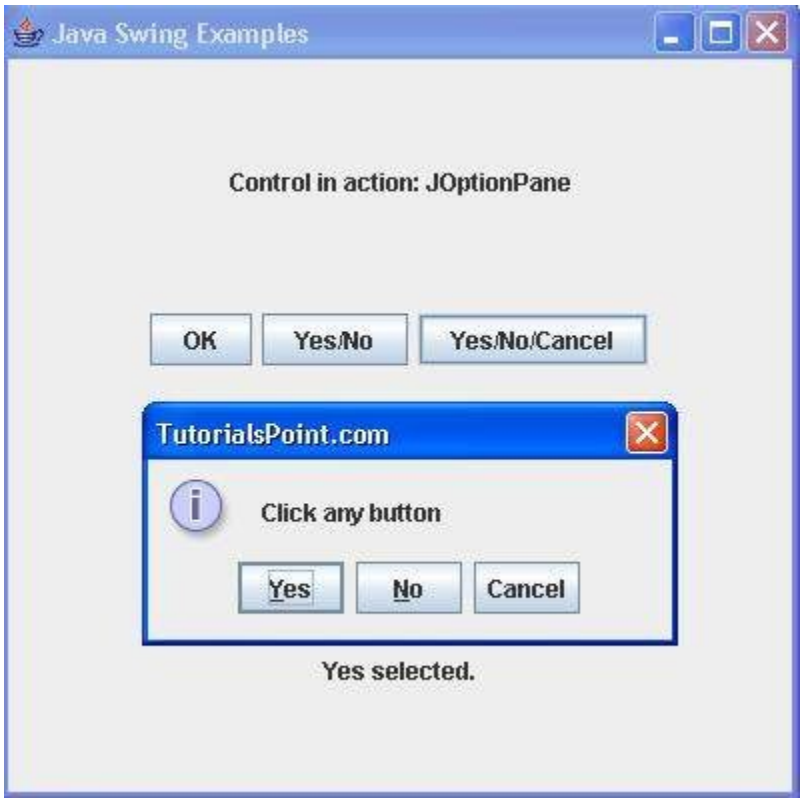
	Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue) Brings up an internal dialog panel with a specified icon, where the initial choice is determined by the initialValue parameter and the number of choices is determined by the optionType parameter.

Example

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

public class SwingControlDemo {

   private JFrame mainFrame;
   private JLabel headerLabel;
   private JLabel statusLabel;
   private JPanel controlPanel;

   public SwingControlDemo(){
      prepareGUI();
   }

   public static void main(String[] args){
      SwingControlDemo  swingControlDemo = new
SwingControlDemo();
      swingControlDemo.showDialogDemo();
   }

   private void prepareGUI(){
      mainFrame = new JFrame("Java Swing Examples");
      mainFrame.setSize(400,400);
      mainFrame.setLayout(new GridLayout(3, 1));
      mainFrame.addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent windowEvent){
            System.exit(0);
         }
      });
      headerLabel = new JLabel("", JLabel.CENTER);
      statusLabel = new JLabel("",JLabel.CENTER);

      statusLabel.setSize(350,100);

      controlPanel = new JPanel();
      controlPanel.setLayout(new FlowLayout());
```

```
      mainFrame.add(headerLabel);
      mainFrame.add(controlPanel);
      mainFrame.add(statusLabel);
      mainFrame.setVisible(true);
   }

   private void showDialogDemo(){
      headerLabel.setText("Control in action: JOptionPane");

      JButton okButton = new JButton("OK");
      JButton javaButton = new JButton("Yes/No");
      JButton cancelButton = new JButton("Yes/No/Cancel");

      okButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(
            mainFrame, "Welcome to TutorialsPoint.com");
         }
      });

      javaButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            int output = JOptionPane.showConfirmDialog(mainFrame
               , "Click any button"
               ,"TutorialsPoint.com"
               ,JOptionPane.YES_NO_OPTION);

            if(output == JOptionPane.YES_OPTION){
               statusLabel.setText("Yes selected.");
            }else if(output == JOptionPane.NO_OPTION){
               statusLabel.setText("No selected.");
            }
         }
      });

      cancelButton.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            int output = JOptionPane.showConfirmDialog(mainFrame
               , "Click any button"
               ,"TutorialsPoint.com"
               ,JOptionPane.YES_NO_CANCEL_OPTION,
               JOptionPane.INFORMATION_MESSAGE);

            if(output == JOptionPane.YES_OPTION){
               statusLabel.setText("Yes selected.");
            }else if(output == JOptionPane.NO_OPTION){
```

```
               statusLabel.setText("No selected.");
            }else if(output == JOptionPane.CANCEL_OPTION){
               statusLabel.setText("Cancel selected.");
            }
         }
      });

      controlPanel.add(okButton);
      controlPanel.add(javaButton);
      controlPanel.add(cancelButton);
      mainFrame.setVisible(true);
   }
}
```



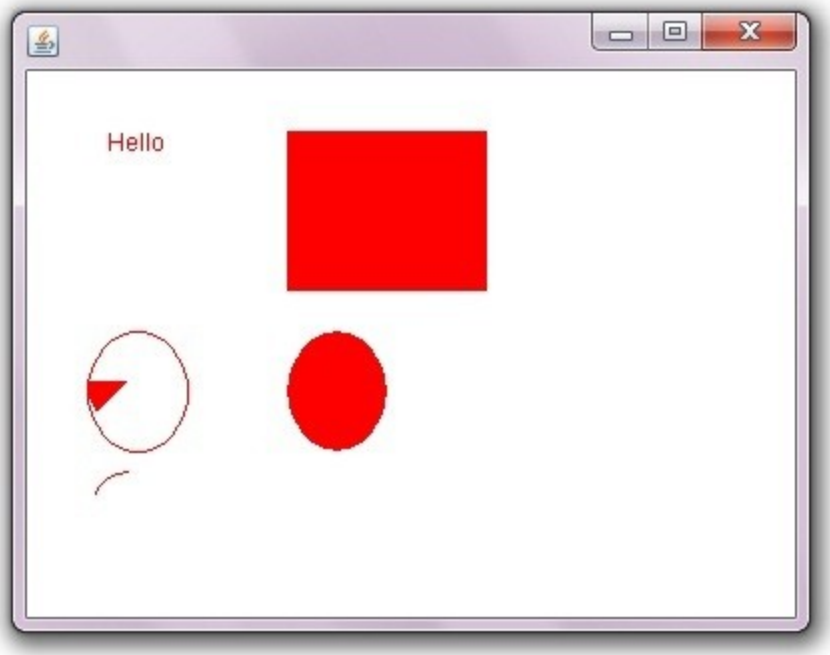
Displaying graphics in swing:

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of displaying graphics in swing:




```
1.  import java.awt.*;
2.  import javax.swing.JFrame;
3.
4.  public class DisplayGraphics extends Canvas{
5.
6.      public void paint(Graphics g) {
7.          g.drawString("Hello",40,40);
8.          setBackground(Color.WHITE);
9.          g.fillRect(130, 30,100, 80);
10.         g.drawOval(30,130,50, 60);
11.         setForeground(Color.RED);
12.         g.fillOval(130,130,50, 60);
13.         g.drawArc(30, 200, 40,50,90,60);
14.         g.fillArc(30, 130, 40,50,180,40);
15.
16.   }
17.         public static void main(String[] args) {
18.         DisplayGraphics m=new DisplayGraphics();
19.         JFrame f=new JFrame();
20.         f.add(m);
21.         f.setSize(400,400);
22.         //f.setLayout(null);
```

```
23.        f.setVisible(true);
24.    }
25.
26.}
```

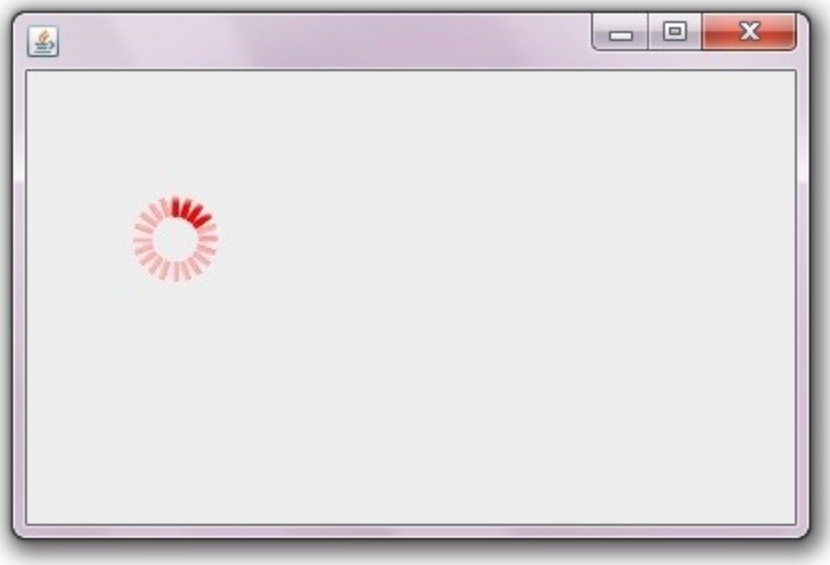
Displaying image in swing:

For displaying image, we can use the method drawImage() of Graphics class.

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

Example of displaying image in swing:



```
1. import java.awt.*;
2. import javax.swing.JFrame;
3.
4. public class MyCanvas extends Canvas{
5.
```

```
6.      public void paint(Graphics g) {
7.
8.          Toolkit t=Toolkit.getDefaultToolkit();
9.          Image i=t.getImage("p3.gif");
10.         g.drawImage(i, 120,100,this);
11.
12.     }
13.         public static void main(String[] args) {
14.         MyCanvas m=new MyCanvas();
15.         JFrame f=new JFrame();
16.         f.add(m);
17.         f.setSize(400,400);
18.         f.setVisible(true);
19.     }
20.
21. }
```

Example of creating Edit menu for Notepad:

```
1.  import javax.swing.*;
2.  import java.awt.event.*;
3.
4.  public class Notepad implements ActionListener{
5.  JFrame f;
6.  JMenuBar mb;
7.  JMenu file,edit,help;
8.  JMenuItem cut,copy,paste,selectAll;
9.  JTextArea ta;
10.
11. Notepad(){
12. f=new JFrame();
13.
14. cut=new JMenuItem("cut");
15. copy=new JMenuItem("copy");
16. paste=new JMenuItem("paste");
17. selectAll=new JMenuItem("selectAll");
18.
19. cut.addActionListener(this);
20. copy.addActionListener(this);
21. paste.addActionListener(this);
```

```
22. selectAll.addActionListener(this);
23.
24. mb=new JMenuBar();
25. mb.setBounds(5,5,400,40);
26.
27. file=new JMenu("File");
28. edit=new JMenu("Edit");
29. help=new JMenu("Help");
30.
31. edit.add(cut);edit.add(copy);edit.add(paste);edit.add(selectAll);
32.
33.
34. mb.add(file);mb.add(edit);mb.add(help);
35.
36. ta=new JTextArea();
37. ta.setBounds(5,30,460,460);
38.
39. f.add(mb);f.add(ta);
40.
41. f.setLayout(null);
42. f.setSize(500,500);
43. f.setVisible(true);
44. }
45.
46. public void actionPerformed(ActionEvent e) {
47. if(e.getSource()==cut)
48. ta.cut();
49. if(e.getSource()==paste)
50. ta.paste();
51. if(e.getSource()==copy)
52. ta.copy();
53. if(e.getSource()==selectAll)
54. ta.selectAll();
55. }
56.
57. public static void main(String[] args) {
58.     new Notepad();
59. }
60. }
```

Example of open dialog box:

```
1. import java.awt.*;
2. import javax.swing.*;
3. import java.awt.event.*;
4. import java.io.*;
5.
6. public class OpenMenu extends JFrame implements ActionListener{
7. JMenuBar mb;
8. JMenu file;
9. JMenuItem open;
10. JTextArea ta;
11. OpenMenu(){
12. open=new JMenuItem("Open File");
13. open.addActionListener(this);
14.
15. file=new JMenu("File");
16. file.add(open);
17.
18. mb=new JMenuBar();
19. mb.setBounds(0,0,800,20);
20. mb.add(file);
21.
22. ta=new JTextArea(800,800);
23. ta.setBounds(0,20,800,800);
24.
25. add(mb);
26. add(ta);
27.
28. }
29. public void actionPerformed(ActionEvent e) {
30. if(e.getSource()==open){
31. openFile();
32. }
33. }
34.
35. void openFile(){
36. JFileChooser fc=new JFileChooser();
37. int i=fc.showOpenDialog(this);
38.
```

```
39. if(i==JFileChooser.APPROVE_OPTION){
40. File f=fc.getSelectedFile();
41. String filepath=f.getPath();
42.
43. displayContent(filepath);
44.
45. }
46.
47. }
48.
49. void displayContent(String fpath){
50. try{
51. BufferedReader br=new BufferedReader(new FileReader(fpath));
52. String s1="",s2="";
53.
54. while((s1=br.readLine())!=null){
55. s2+=s1+"\n";
56. }
57. ta.setText(s2);
58. br.close();
59. }catch (Exception e) {e.printStackTrace();  }
60. }
61.
62. public static void main(String[] args) {
63.     OpenMenu om=new OpenMenu();
64.     om.setSize(800,800);
65.     om.setLayout(null);
66.     om.setVisible(true);
67.     om.setDefaultCloseOperation(EXIT_ON_CLOSE);
68. }
69. }
```

Example of Notepad

```
1. import java.io.*;
2. import java.util.Date;
3. import java.awt.*;
4. import java.awt.event.*;
5. import javax.swing.*;
6. import javax.swing.event.*;
```

```
7.
8.  /**********************************/
9.  class FileOperation
10. {
11. Notepad npd;
12.
13. boolean saved;
14. boolean newFileFlag;
15. String fileName;
16. String applicationTitle="Javapad";
17.
18. File fileRef;
19. JFileChooser chooser;
20. ////////////////////////////
21. boolean isSave(){return saved;}
22. void setSave(boolean saved){this.saved=saved;}
23. String getFileName(){return new String(fileName);}
24. void setFileName(String fileName){this.fileName=new String(fileName);}
25. ////////////////////////
26. FileOperation(Notepad npd)
27. {
28. this.npd=npd;
29.
30. saved=true;
31. newFileFlag=true;
32. fileName=new String("Untitled");
33. fileRef=new File(fileName);
34. this.npd.f.setTitle(fileName+" - "+applicationTitle);
35.
36. chooser=new JFileChooser();
37. chooser.addChoosableFileFilter(new MyFileFilter(".java","Java Source Files(*.java
    )"));
38. chooser.addChoosableFileFilter(new MyFileFilter(".txt","Text Files(*.txt)"));
39. chooser.setCurrentDirectory(new File("."));
40.
41. }
42. ////////////////////////////////////
43.
44. boolean saveFile(File temp)
45. {
```

```
46. FileWriter fout=null;
47. try
48. {
49. fout=new FileWriter(temp);
50. fout.write(npd.ta.getText());
51. }
52. catch(IOException ioe){updateStatus(temp,false);return false;}
53. finally
54. {try{fout.close();}catch(IOException excp){}}
55. updateStatus(temp,true);
56. return true;
57. }
58. ////////////////////////
59. boolean saveThisFile()
60. {
61.
62. if(!newFileFlag)
63.     {return saveFile(fileRef);}
64.
65. return saveAsFile();
66. }
67. ////////////////////////////////
68. boolean saveAsFile()
69. {
70. File temp=null;
71. chooser.setDialogTitle("Save As...");
72. chooser.setApproveButtonText("Save Now");
73. chooser.setApproveButtonMnemonic(KeyEvent.VK_S);
74. chooser.setApproveButtonToolTipText("Click me to save!");
75.
76. do
77. {
78. if(chooser.showSaveDialog(this.npd.f)!=JFileChooser.APPROVE_OPTION)
79.     return false;
80. temp=chooser.getSelectedFile();
81. if(!temp.exists()) break;
82. if(   JOptionPane.showConfirmDialog(
83.     this.npd.f,"<html>"+temp.getPath()+" already exists.<br>Do you want to replac
    e it?<html>",
84.     "Save As",JOptionPane.YES_NO_OPTION
```

```
85.              )==JOptionPane.YES_OPTION)
86.    break;
87. }while(true);
88.
89.
90. return saveFile(temp);
91. }
92.
93. ////////////////////////
94. boolean openFile(File temp)
95. {
96. FileInputStream fin=null;
97. BufferedReader din=null;
98.
99. try
100.        {
101.        fin=new FileInputStream(temp);
102.        din=new BufferedReader(new InputStreamReader(fin));
103.        String str=" ";
104.        while(str!=null)
105.        {
106.        str=din.readLine();
107.        if(str==null)
108.        break;
109.        this.npd.ta.append(str+"\n");
110.        }
111.
112.        }
113.        catch(IOException ioe){updateStatus(temp,false);return false;}
114.        finally
115.        {try{din.close();fin.close();}catch(IOException excp){}}
116.        updateStatus(temp,true);
117.        this.npd.ta.setCaretPosition(0);
118.        return true;
119.        }
120.        ////////////////////////
121.        void openFile()
122.        {
123.        if(!confirmSave()) return;
124.        chooser.setDialogTitle("Open File...");
```

```
125.        chooser.setApproveButtonText("Open this");
126.        chooser.setApproveButtonMnemonic(KeyEvent.VK_O);
127.        chooser.setApproveButtonToolTipText("Click me to open the selected file.
    !");
128.
129.        File temp=null;
130.        do
131.        {
132.        if(chooser.showOpenDialog(this.npd.f)!=JFileChooser.APPROVE_OPTION)
133.           return;
134.        temp=chooser.getSelectedFile();
135.
136.        if(temp.exists())   break;
137.
138.        JOptionPane.showMessageDialog(this.npd.f,
139.           "<html>"+temp.getName()+"<br>file not found.<br>"+
140.           "Please verify the correct file name was given.<html>",
141.           "Open", JOptionPane.INFORMATION_MESSAGE);
142.
143.        } while(true);
144.
145.        this.npd.ta.setText("");
146.
147.        if(!openFile(temp))
148.           {
149.           fileName="Untitled"; saved=true;
150.           this.npd.f.setTitle(fileName+" - "+applicationTitle);
151.           }
152.        if(!temp.canWrite())
153.           newFileFlag=true;
154.
155.        }
156.        //////////////////////
157.        void updateStatus(File temp,boolean saved)
158.        {
159.        if(saved)
160.        {
161.        this.saved=true;
162.        fileName=new String(temp.getName());
163.        if(!temp.canWrite())
```

```
164.            {fileName+="(Read only)"; newFileFlag=true;}
165.        fileRef=temp;
166.        npd.f.setTitle(fileName + " - "+applicationTitle);
167.        npd.statusBar.setText("File : "+temp.getPath()+" saved/opened successfull
    y.");
168.        newFileFlag=false;
169.        }
170.        else
171.        {
172.        npd.statusBar.setText("Failed to save/open : "+temp.getPath());
173.        }
174.        }
175.        //////////////////////
176.        boolean confirmSave()
177.        {
178.        String strMsg="<html>The text in the "+fileName+" file has been changed.
    <br>"+
179.            "Do you want to save the changes?<html>";
180.        if(!saved)
181.        {
182.        int x=JOptionPane.showConfirmDialog(this.npd.f,strMsg,applicationTitle,
183.        JOptionPane.YES_NO_CANCEL_OPTION);
184.
185.        if(x==JOptionPane.CANCEL_OPTION) return false;
186.        if(x==JOptionPane.YES_OPTION && !saveAsFile()) return false;
187.        }
188.        return true;
189.        }
190.        ////////////////////////////////////
191.        void newFile()
192.        {
193.        if(!confirmSave()) return;
194.
195.        this.npd.ta.setText("");
196.        fileName=new String("Untitled");
197.        fileRef=new File(fileName);
198.        saved=true;
199.        newFileFlag=true;
200.        this.npd.f.setTitle(fileName+" - "+applicationTitle);
201.        }
```

```
202.    ////////////////////////////////////
203.    }// end defination of class FileOperation
204.    /**********************************/
205.    public class Notepad  implements ActionListener, MenuConstants
206.    {
207.
208.    JFrame f;
209.    JTextArea ta;
210.    JLabel statusBar;
211.
212.    private String fileName="Untitled";
213.    private boolean saved=true;
214.    String applicationName="Javapad";
215.
216.    String searchString, replaceString;
217.    int lastSearchIndex;
218.
219.    FileOperation fileHandler;
220.    FontChooser fontDialog=null;
221.    FindDialog findReplaceDialog=null;
222.    JColorChooser bcolorChooser=null;
223.    JColorChooser fcolorChooser=null;
224.    JDialog backgroundDialog=null;
225.    JDialog foregroundDialog=null;
226.    JMenuItem cutItem,copyItem, deleteItem, findItem, findNextItem,
227.    replaceItem, gotoItem, selectAllItem;
228.    /***************************/
229.    Notepad()
230.    {
231.    f=new JFrame(fileName+" - "+applicationName);
232.    ta=new JTextArea(30,60);
233.    statusBar=new JLabel("||       Ln 1, Col 1  ",JLabel.RIGHT);
234.    f.add(new JScrollPane(ta),BorderLayout.CENTER);
235.    f.add(statusBar,BorderLayout.SOUTH);
236.    f.add(new JLabel("  "),BorderLayout.EAST);
237.    f.add(new JLabel("  "),BorderLayout.WEST);
238.    createMenuBar(f);
239.    //f.setSize(350,350);
240.    f.pack();
241.    f.setLocation(100,50);
```

```
242.        f.setVisible(true);
243.        f.setLocation(150,50);
244.        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
245.
246.        fileHandler=new FileOperation(this);
247.
248.        ////////////////////
249.
250.        ta.addCaretListener(
251.        new CaretListener()
252.        {
253.        public void caretUpdate(CaretEvent e)
254.        {
255.        int lineNumber=0, column=0, pos=0;
256.
257.        try
258.        {
259.        pos=ta.getCaretPosition();
260.        lineNumber=ta.getLineOfOffset(pos);
261.        column=pos-ta.getLineStartOffset(lineNumber);
262.        }catch(Exception excp){}
263.        if(ta.getText().length()==0){lineNumber=0; column=0;}
264.        statusBar.setText("||      Ln "+(lineNumber+1)+", Col "+(column+1));
265.        }
266.        });
267.        /////////////////
268.        DocumentListener myListener = new DocumentListener()
269.        {
270.        public void changedUpdate(DocumentEvent e){fileHandler.saved=false;}
271.        public void removeUpdate(DocumentEvent e){fileHandler.saved=false;}
272.        public void insertUpdate(DocumentEvent e){fileHandler.saved=false;}
273.        };
274.        ta.getDocument().addDocumentListener(myListener);
275.        /////////
276.        WindowListener frameClose=new WindowAdapter()
277.        {
278.        public void windowClosing(WindowEvent we)
279.        {
280.        if(fileHandler.confirmSave())System.exit(0);
281.        }
```

```
282.        };
283.        f.addWindowListener(frameClose);
284.        /////////////////
285.        /*
286.        ta.append("Hello dear hello hi");
287.        ta.append("\nwho are u dear mister hello");
288.        ta.append("\nhello bye hel");
289.        ta.append("\nHello");
290.        ta.append("\nMiss u mister hello hell");
291.        fileHandler.saved=true;
292.        */
293.        }
294.        ///////////////////////////////////
295.        void goTo()
296.        {
297.        int lineNumber=0;
298.        try
299.        {
300.        lineNumber=ta.getLineOfOffset(ta.getCaretPosition())+1;
301.        String tempStr=JOptionPane.showInputDialog(f,"Enter Line Number:",""+li
    neNumber);
302.        if(tempStr==null)
303.           {return;}
304.        lineNumber=Integer.parseInt(tempStr);
305.        ta.setCaretPosition(ta.getLineStartOffset(lineNumber-1));
306.        }catch(Exception e){}
307.        }
308.        ///////////////////////////////////
309.        public void actionPerformed(ActionEvent ev)
310.        {
311.        String cmdText=ev.getActionCommand();
312.        ///////////////////////////////////
313.        if(cmdText.equals(fileNew))
314.           fileHandler.newFile();
315.        else if(cmdText.equals(fileOpen))
316.           fileHandler.openFile();
317.        ///////////////////////////////////
318.        else if(cmdText.equals(fileSave))
319.           fileHandler.saveThisFile();
320.        ///////////////////////////////////
```

```
321.        else if(cmdText.equals(fileSaveAs))
322.           fileHandler.saveAsFile();
323.        ////////////////////////////////
324.        else if(cmdText.equals(fileExit))
325.           {if(fileHandler.confirmSave())System.exit(0);}
326.        ////////////////////////////////
327.        else if(cmdText.equals(filePrint))
328.        JOptionPane.showMessageDialog(
329.           Notepad.this.f,
330.           "Get ur printer repaired first! It seems u dont have one!",
331.           "Bad Printer",
332.           JOptionPane.INFORMATION_MESSAGE
333.           );
334.        ////////////////////////////////
335.        else if(cmdText.equals(editCut))
336.           ta.cut();
337.        ////////////////////////////////
338.        else if(cmdText.equals(editCopy))
339.           ta.copy();
340.        ////////////////////////////////
341.        else if(cmdText.equals(editPaste))
342.           ta.paste();
343.        ////////////////////////////////
344.        else if(cmdText.equals(editDelete))
345.           ta.replaceSelection("");
346.        ////////////////////////////////
347.        else if(cmdText.equals(editFind))
348.        {
349.        if(Notepad.this.ta.getText().length()==0)
350.           return; // text box have no text
351.        if(findReplaceDialog==null)
352.           findReplaceDialog=new FindDialog(Notepad.this.ta);
353.        findReplaceDialog.showDialog(Notepad.this.f,true);//find
354.        }
355.        ////////////////////////////////
356.        else if(cmdText.equals(editFindNext))
357.        {
358.        if(Notepad.this.ta.getText().length()==0)
359.           return; // text box have no text
360.
```

```
361.        if(findReplaceDialog==null)
362.           statusBar.setText("Use Find option of Edit Menu first !!!!");
363.        else
364.           findReplaceDialog.findNextWithSelection();
365.        }
366.        ////////////////////////////////
367.        else if(cmdText.equals(editReplace))
368.        {
369.        if(Notepad.this.ta.getText().length()==0)
370.           return; // text box have no text
371.
372.        if(findReplaceDialog==null)
373.           findReplaceDialog=new FindDialog(Notepad.this.ta);
374.        findReplaceDialog.showDialog(Notepad.this.f,false);//replace
375.        }
376.        ////////////////////////////////
377.        else if(cmdText.equals(editGoTo))
378.        {
379.        if(Notepad.this.ta.getText().length()==0)
380.           return; // text box have no text
381.        goTo();
382.        }
383.        ////////////////////////////////
384.        else if(cmdText.equals(editSelectAll))
385.           ta.selectAll();
386.        ////////////////////////////////
387.        else if(cmdText.equals(editTimeDate))
388.           ta.insert(new Date().toString(),ta.getSelectionStart());
389.        ////////////////////////////////
390.        else if(cmdText.equals(formatWordWrap))
391.        {
392.        JCheckBoxMenuItem temp=(JCheckBoxMenuItem)ev.getSource();
393.        ta.setLineWrap(temp.isSelected());
394.        }
395.        ////////////////////////////////
396.        else if(cmdText.equals(formatFont))
397.        {
398.        if(fontDialog==null)
399.           fontDialog=new FontChooser(ta.getFont());
400.
```

```
401.        if(fontDialog.showDialog(Notepad.this.f,"Choose a font"))
402.           Notepad.this.ta.setFont(fontDialog.createFont());
403.        }
404.        ////////////////////////////////
405.        else if(cmdText.equals(formatForeground))
406.           showForegroundColorDialog();
407.        ////////////////////////////////
408.        else if(cmdText.equals(formatBackground))
409.           showBackgroundColorDialog();
410.        ////////////////////////////////
411.
412.        else if(cmdText.equals(viewStatusBar))
413.        {
414.        JCheckBoxMenuItem temp=(JCheckBoxMenuItem)ev.getSource();
415.        statusBar.setVisible(temp.isSelected());
416.        }
417.        ////////////////////////////////
418.        else if(cmdText.equals(helpAboutNotepad))
419.        {
420.        JOptionPane.showMessageDialog(Notepad.this.f,aboutText,"Dedicated 2
    u!",
421.        JOptionPane.INFORMATION_MESSAGE);
422.        }
423.        else
424.           statusBar.setText("This "+cmdText+" command is yet to be implemente
    d");
425.        }//action Performed
426.        ////////////////////////////////
427.        void showBackgroundColorDialog()
428.        {
429.        if(bcolorChooser==null)
430.           bcolorChooser=new JColorChooser();
431.        if(backgroundDialog==null)
432.           backgroundDialog=JColorChooser.createDialog
433.              (Notepad.this.f,
434.              formatBackground,
435.              false,
436.              bcolorChooser,
437.              new ActionListener()
438.              {public void actionPerformed(ActionEvent evvv){
```

```
439.                Notepad.this.ta.setBackground(bcolorChooser.getColor());}},
440.             null);
441.
442.       backgroundDialog.setVisible(true);
443.       }
444.       ////////////////////////////////
445.       void showForegroundColorDialog()
446.       {
447.       if(fcolorChooser==null)
448.          fcolorChooser=new JColorChooser();
449.       if(foregroundDialog==null)
450.          foregroundDialog=JColorChooser.createDialog
451.             (Notepad.this.f,
452.             formatForeground,
453.             false,
454.             fcolorChooser,
455.             new ActionListener()
456.             {public void actionPerformed(ActionEvent evvv){
457.                Notepad.this.ta.setForeground(fcolorChooser.getColor());}},
458.             null);
459.
460.       foregroundDialog.setVisible(true);
461.       }
462.
463.       ////////////////////////////////
464.       JMenuItem createMenuItem(String s, int key,JMenu toMenu,ActionListene
    r al)
465.       {
466.       JMenuItem temp=new JMenuItem(s,key);
467.       temp.addActionListener(al);
468.       toMenu.add(temp);
469.
470.       return temp;
471.       }
472.       ////////////////////////////////
473.       JMenuItem createMenuItem(String s, int key,JMenu toMenu,int aclKey,Act
    ionListener al)
474.       {
475.       JMenuItem temp=new JMenuItem(s,key);
476.       temp.addActionListener(al);
```

```
477.        temp.setAccelerator(KeyStroke.getKeyStroke(aclKey,ActionEvent.CTRL_M
    ASK));
478.        toMenu.add(temp);
479.
480.        return temp;
481.        }
482.        ////////////////////////////////
483.        JCheckBoxMenuItem createCheckBoxMenuItem(String s,
484.         int key,JMenu toMenu,ActionListener al)
485.        {
486.        JCheckBoxMenuItem temp=new JCheckBoxMenuItem(s);
487.        temp.setMnemonic(key);
488.        temp.addActionListener(al);
489.        temp.setSelected(false);
490.        toMenu.add(temp);
491.
492.        return temp;
493.        }
494.        ////////////////////////////////
495.        JMenu createMenu(String s,int key,JMenuBar toMenuBar)
496.        {
497.        JMenu temp=new JMenu(s);
498.        temp.setMnemonic(key);
499.        toMenuBar.add(temp);
500.        return temp;
501.        }
502.        /*******************************/
503.        void createMenuBar(JFrame f)
504.        {
505.        JMenuBar mb=new JMenuBar();
506.        JMenuItem temp;
507.
508.        JMenu fileMenu=createMenu(fileText,KeyEvent.VK_F,mb);
509.        JMenu editMenu=createMenu(editText,KeyEvent.VK_E,mb);
510.        JMenu formatMenu=createMenu(formatText,KeyEvent.VK_O,mb);
511.        JMenu viewMenu=createMenu(viewText,KeyEvent.VK_V,mb);
512.        JMenu helpMenu=createMenu(helpText,KeyEvent.VK_H,mb);
513.
514.        createMenuItem(fileNew,KeyEvent.VK_N,fileMenu,KeyEvent.VK_N,this);
515.        createMenuItem(fileOpen,KeyEvent.VK_O,fileMenu,KeyEvent.VK_O,this);
```

```
516.        createMenuItem(fileSave,KeyEvent.VK_S,fileMenu,KeyEvent.VK_S,this);
517.        createMenuItem(fileSaveAs,KeyEvent.VK_A,fileMenu,this);
518.        fileMenu.addSeparator();
519.        temp=createMenuItem(filePageSetup,KeyEvent.VK_U,fileMenu,this);
520.        temp.setEnabled(false);
521.        createMenuItem(filePrint,KeyEvent.VK_P,fileMenu,KeyEvent.VK_P,this);
522.        fileMenu.addSeparator();
523.        createMenuItem(fileExit,KeyEvent.VK_X,fileMenu,this);
524.
525.        temp=createMenuItem(editUndo,KeyEvent.VK_U,editMenu,KeyEvent.VK_
    Z,this);
526.        temp.setEnabled(false);
527.        editMenu.addSeparator();
528.        cutItem=createMenuItem(editCut,KeyEvent.VK_T,editMenu,KeyEvent.VK_
    X,this);
529.        copyItem=createMenuItem(editCopy,KeyEvent.VK_C,editMenu,KeyEvent.
    VK_C,this);
530.        createMenuItem(editPaste,KeyEvent.VK_P,editMenu,KeyEvent.VK_V,this);

531.        deleteItem=createMenuItem(editDelete,KeyEvent.VK_L,editMenu,this);
532.        deleteItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE,0
    ));
533.        editMenu.addSeparator();
534.        findItem=createMenuItem(editFind,KeyEvent.VK_F,editMenu,KeyEvent.VK
    _F,this);
535.        findNextItem=createMenuItem(editFindNext,KeyEvent.VK_N,editMenu,thi
    s);
536.        findNextItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F3,0));

537.        replaceItem=createMenuItem(editReplace,KeyEvent.VK_R,editMenu,KeyE
    vent.VK_H,this);
538.        gotoItem=createMenuItem(editGoTo,KeyEvent.VK_G,editMenu,KeyEvent.
    VK_G,this);
539.        editMenu.addSeparator();
540.        selectAllItem=createMenuItem(editSelectAll,KeyEvent.VK_A,editMenu,Key
    Event.VK_A,this);
541.        createMenuItem(editTimeDate,KeyEvent.VK_D,editMenu,this)
542.        .setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F5,0));
543.
```

```
544.        createCheckBoxMenuItem(formatWordWrap,KeyEvent.VK_W,formatMen
    u,this);
545.
546.        createMenuItem(formatFont,KeyEvent.VK_F,formatMenu,this);
547.        formatMenu.addSeparator();
548.        createMenuItem(formatForeground,KeyEvent.VK_T,formatMenu,this);
549.        createMenuItem(formatBackground,KeyEvent.VK_P,formatMenu,this);
550.
551.        createCheckBoxMenuItem(viewStatusBar,KeyEvent.VK_S,viewMenu,this).s
    etSelected(true);
552.        /************For Look and Feel***/
553.        LookAndFeelMenu.createLookAndFeelMenuItem(viewMenu,this.f);
554.
555.
556.        temp=createMenuItem(helpHelpTopic,KeyEvent.VK_H,helpMenu,this);
557.        temp.setEnabled(false);
558.        helpMenu.addSeparator();
559.        createMenuItem(helpAboutNotepad,KeyEvent.VK_A,helpMenu,this);
560.
561.        MenuListener editMenuListener=new MenuListener()
562.        {
563.          public void menuSelected(MenuEvent evvvv)
564.           {
565.           if(Notepad.this.ta.getText().length()==0)
566.           {
567.           findItem.setEnabled(false);
568.           findNextItem.setEnabled(false);
569.           replaceItem.setEnabled(false);
570.           selectAllItem.setEnabled(false);
571.           gotoItem.setEnabled(false);
572.           }
573.           else
574.           {
575.           findItem.setEnabled(true);
576.           findNextItem.setEnabled(true);
577.           replaceItem.setEnabled(true);
578.           selectAllItem.setEnabled(true);
579.           gotoItem.setEnabled(true);
580.           }
581.           if(Notepad.this.ta.getSelectionStart()==ta.getSelectionEnd())
```

```
582.          {
583.          cutItem.setEnabled(false);
584.          copyItem.setEnabled(false);
585.          deleteItem.setEnabled(false);
586.          }
587.          else
588.          {
589.          cutItem.setEnabled(true);
590.          copyItem.setEnabled(true);
591.          deleteItem.setEnabled(true);
592.          }
593.          }
594.         public void menuDeselected(MenuEvent evvvv){}
595.         public void menuCanceled(MenuEvent evvvv){}
596.       };
597.       editMenu.addMenuListener(editMenuListener);
598.       f.setJMenuBar(mb);
599.       }
600.       /*************Constructor**************/
601.       ////////////////////////////////
602.       public static void main(String[] s)
603.       {
604.       new Notepad();
605.       }
606.       }
607.       /************************************/
608.       //public
609.       interface MenuConstants
610.       {
611.       final String fileText="File";
612.       final String editText="Edit";
613.       final String formatText="Format";
614.       final String viewText="View";
615.       final String helpText="Help";
616.
617.       final String fileNew="New";
618.       final String fileOpen="Open...";
619.       final String fileSave="Save";
620.       final String fileSaveAs="Save As...";
621.       final String filePageSetup="Page Setup...";
```

```
622.        final String filePrint="Print";
623.        final String fileExit="Exit";
624.
625.        final String editUndo="Undo";
626.        final String editCut="Cut";
627.        final String editCopy="Copy";
628.        final String editPaste="Paste";
629.        final String editDelete="Delete";
630.        final String editFind="Find...";
631.        final String editFindNext="Find Next";
632.        final String editReplace="Replace";
633.        final String editGoTo="Go To...";
634.        final String editSelectAll="Select All";
635.        final String editTimeDate="Time/Date";
636.
637.        final String formatWordWrap="Word Wrap";
638.        final String formatFont="Font...";
639.        final String formatForeground="Set Text color...";
640.        final String formatBackground="Set Pad color...";
641.
642.        final String viewStatusBar="Status Bar";
643.
644.        final String helpHelpTopic="Help Topic";
645.        final String helpAboutNotepad="About Javapad";
646.
647.        final String aboutText="Your Javapad";
648.        }
```

Online Exam Project in Java Swing without database

In this project, there are given 10 questions to play. User can bookmark any question for the reconsideration while going to result.

We are using here java array to store the questions, options and answers not database. You can use collection framework or database in place of array.

```
1. /*Online Java Paper Test*/
2.
3. import java.awt.*;
4. import java.awt.event.*;
```

```
5.  import javax.swing.*;
6.
7.  class OnlineTest extends JFrame implements ActionListener
8.  {
9.     JLabel l;
10.    JRadioButton jb[]=new JRadioButton[5];
11.    JButton b1,b2;
12.    ButtonGroup bg;
13.    int count=0,current=0,x=1,y=1,now=0;
14.    int m[]=new int[10];
15.    OnlineTest(String s)
16.    {
17.       super(s);
18.       l=new JLabel();
19.       add(l);
20.       bg=new ButtonGroup();
21.       for(int i=0;i<5;i++)
22.       {
23.          jb[i]=new JRadioButton();
24.          add(jb[i]);
25.          bg.add(jb[i]);
26.       }
27.       b1=new JButton("Next");
28.       b2=new JButton("Bookmark");
29.       b1.addActionListener(this);
30.       b2.addActionListener(this);
31.       add(b1);add(b2);
32.       set();
33.       l.setBounds(30,40,450,20);
34.       jb[0].setBounds(50,80,100,20);
35.       jb[1].setBounds(50,110,100,20);
36.       jb[2].setBounds(50,140,100,20);
37.       jb[3].setBounds(50,170,100,20);
38.       b1.setBounds(100,240,100,30);
39.       b2.setBounds(270,240,100,30);
40.       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41.       setLayout(null);
42.       setLocation(250,100);
43.       setVisible(true);
44.       setSize(600,350);
```

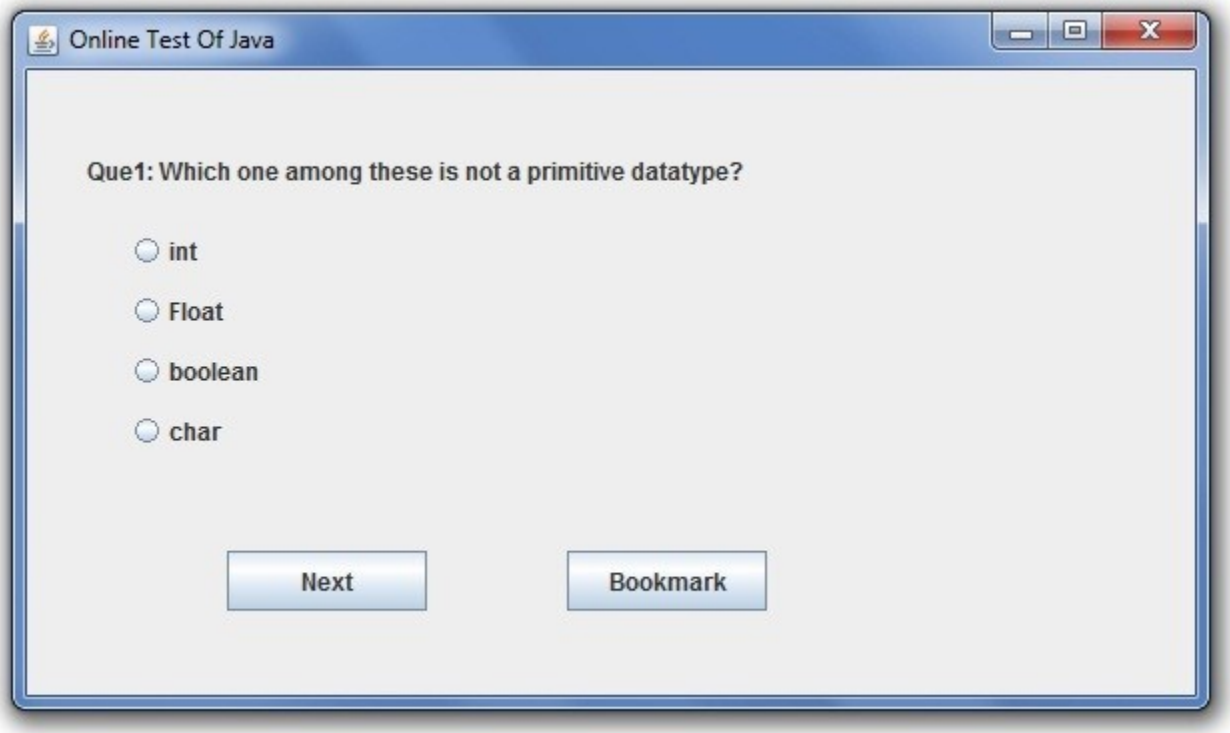
```
45.  }
46.  public void actionPerformed(ActionEvent e)
47.  {
48.     if(e.getSource()==b1)
49.     {
50.        if(check())
51.           count=count+1;
52.        current++;
53.        set();
54.        if(current==9)
55.        {
56.           b1.setEnabled(false);
57.           b2.setText("Result");
58.        }
59.     }
60.     if(e.getActionCommand().equals("Bookmark"))
61.     {
62.        JButton bk=new JButton("Bookmark"+x);
63.        bk.setBounds(480,20+30*x,100,30);
64.        add(bk);
65.        bk.addActionListener(this);
66.        m[x]=current;
67.        x++;
68.        current++;
69.        set();
70.        if(current==9)
71.           b2.setText("Result");
72.        setVisible(false);
73.        setVisible(true);
74.     }
75.     for(int i=0,y=1;i<x;i++,y++)
76.     {
77.     if(e.getActionCommand().equals("Bookmark"+y))
78.     {
79.        if(check())
80.           count=count+1;
81.        now=current;
82.        current=m[y];
83.        set();
84.        ((JButton)e.getSource()).setEnabled(false);
```

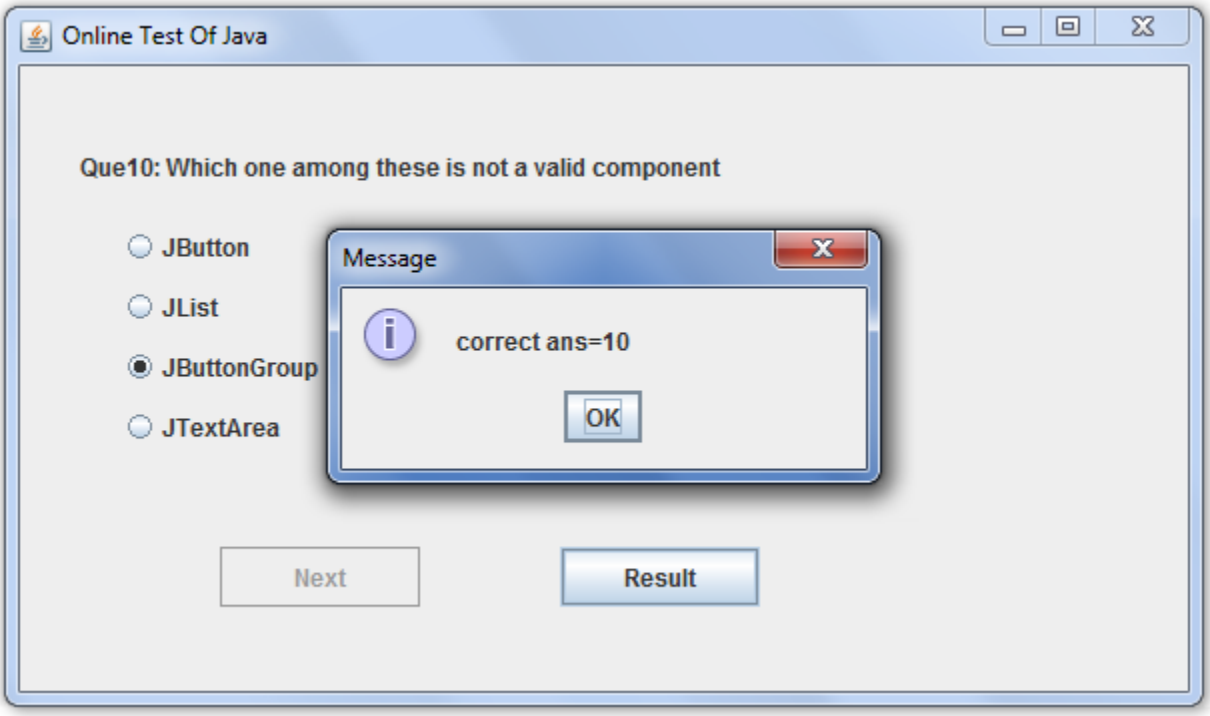
```
85.          current=now;
86.       }
87.       }
88.
89.       if(e.getActionCommand().equals("Result"))
90.       {
91.          if(check())
92.             count=count+1;
93.          current++;
94.          //System.out.println("correct ans="+count);
95.          JOptionPane.showMessageDialog(this,"correct ans="+count);
96.          System.exit(0);
97.       }
98.    }
99.    void set()
100.          {
101.             jb[4].setSelected(true);
102.             if(current==0)
103.             {
104.                l.setText("Que1: Which one among these is not a primitive datatype
    ?");
105.                jb[0].setText("int");jb[1].setText("Float");jb[2].setText("boolean");jb
    [3].setText("char");
106.             }
107.             if(current==1)
108.             {
109.                l.setText("Que2: Which class is available to all the class automaticall
    y?");
110.                jb[0].setText("Swing");jb[1].setText("Applet");jb[2].setText("Object"
    );jb[3].setText("ActionEvent");
111.             }
112.             if(current==2)
113.             {
114.                l.setText("Que3: Which package is directly available to our class wit
    hout importing it?");
115.                jb[0].setText("swing");jb[1].setText("applet");jb[2].setText("net");jb[
    3].setText("lang");
116.             }
117.             if(current==3)
118.             {
```

```
119.                 l.setText("Que4: String class is defined in which package?");
120.                 jb[0].setText("lang");jb[1].setText("Swing");jb[2].setText("Applet");j
    b[3].setText("awt");
121.             }
122.             if(current==4)
123.             {
124.                 l.setText("Que5: Which institute is best for java coaching?");
125.                 jb[0].setText("Utek");jb[1].setText("Aptech");jb[2].setText("SSS IT");
    jb[3].setText("jtek");
126.             }
127.             if(current==5)
128.             {
129.                 l.setText("Que6: Which one among these is not a keyword?");
130.                 jb[0].setText("class");jb[1].setText("int");jb[2].setText("get");jb[3].se
    tText("if");
131.             }
132.             if(current==6)
133.             {
134.                 l.setText("Que7: Which one among these is not a class? ");
135.                 jb[0].setText("Swing");jb[1].setText("Actionperformed");jb[2].setTe
    xt("ActionEvent");
136.                         jb[3].setText("Button");
137.             }
138.             if(current==7)
139.             {
140.                 l.setText("Que8: which one among these is not a function of Object
    class?");
141.                 jb[0].setText("toString");jb[1].setText("finalize");jb[2].setText("equa
    ls");
142.                         jb[3].setText("getDocumentBase");
143.             }
144.             if(current==8)
145.             {
146.                 l.setText("Que9: which function is not present in Applet class?");
147.                 jb[0].setText("init");jb[1].setText("main");jb[2].setText("start");jb[3].
    setText("destroy");
148.             }
149.             if(current==9)
150.             {
```

```
151.                 l.setText("Que10: Which one among these is not a valid component
    ?");
152.                 jb[0].setText("JButton");jb[1].setText("JList");jb[2].setText("JButton
    Group");
153.                         jb[3].setText("JTextArea");
154.             }
155.             l.setBounds(30,40,450,20);
156.             for(int i=0,j=0;i<=90;i+=30,j++)
157.                 jb[j].setBounds(50,80+i,200,20);
158.         }
159.         boolean check()
160.         {
161.             if(current==0)
162.                 return(jb[1].isSelected());
163.             if(current==1)
164.                 return(jb[2].isSelected());
165.             if(current==2)
166.                 return(jb[3].isSelected());
167.             if(current==3)
168.                 return(jb[0].isSelected());
169.             if(current==4)
170.                 return(jb[2].isSelected());
171.             if(current==5)
172.                 return(jb[2].isSelected());
173.             if(current==6)
174.                 return(jb[1].isSelected());
175.             if(current==7)
176.                 return(jb[3].isSelected());
177.             if(current==8)
178.                 return(jb[1].isSelected());
179.             if(current==9)
180.                 return(jb[2].isSelected());
181.             return false;
182.         }
183.         public static void main(String s[])
184.         {
185.             new OnlineTest("Online Test Of Java");
186.         }
187.     }
```

Output


Online Test Of Java
Que1: Which one among these is not a primitive datatype?
int
Float
boolean
char
Next
Bookmark


Online Test Of Java
Que10: Which one among these is not a valid component
JButton
JList
JButtonGroup
JTextArea
Message
correct ans=10
OK
Next
Result

BoxLayout class:

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

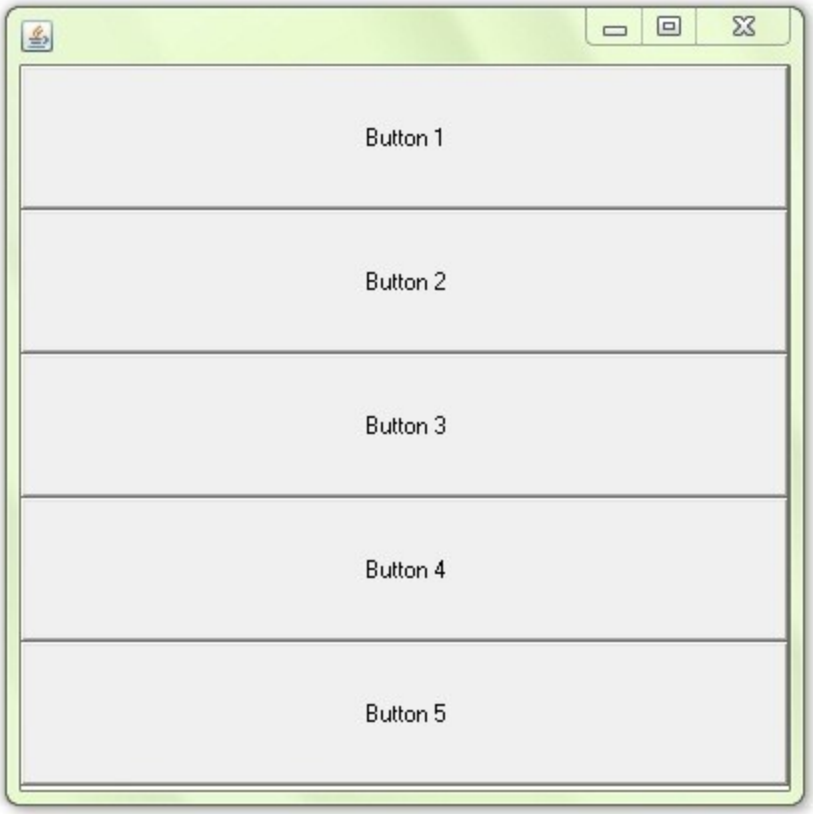
Fields of BoxLayout class:

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class:

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

Example of BoxLayout class with Y-AXIS:

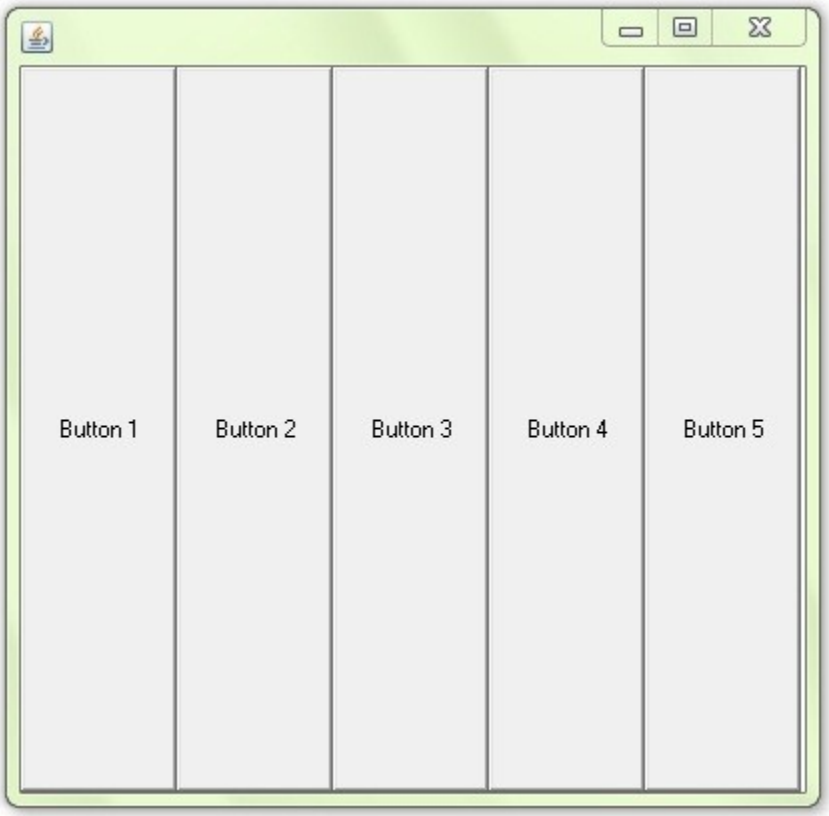



```
1.  import java.awt.*;
2.  import javax.swing.*;
3.
4.  public class BoxLayoutExample1 extends Frame {
5.   Button buttons[];
6.
7.   public BoxLayoutExample1 () {
8.     buttons = new Button [5];
9.
10.   for (int i = 0;i<5;i++) {
11.      buttons[i] = new Button ("Button " + (i + 1));
12.      add (buttons[i]);
13.    }
14.
15. setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
```

```
20. public static void main(String args[]){
21. BoxLayoutExample1 b=new BoxLayoutExample1();
22. }
23. }
```

download this example

Example of BoxLayout class with X-AXIS:




```
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class BoxLayoutExample2 extends Frame {
5.  Button buttons[];
6.
7.  public BoxLayoutExample2() {
8.    buttons = new Button [5];
9.
```

```
10.   for (int i = 0;i<5;i++) {
11.      buttons[i] = new Button ("Button " + (i + 1));
12.      add (buttons[i]);
13.    }
14.
15. setLayout (new BoxLayout(this, BoxLayout.X_AXIS));
16. setSize(400,400);
17. setVisible(true);
18. }
19.
20. public static void main(String args[]){
21. BoxLayoutExample2 b=new BoxLayoutExample2();
22. }
23. }
```

CardLayout class

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.

Constructors of CardLayout class:

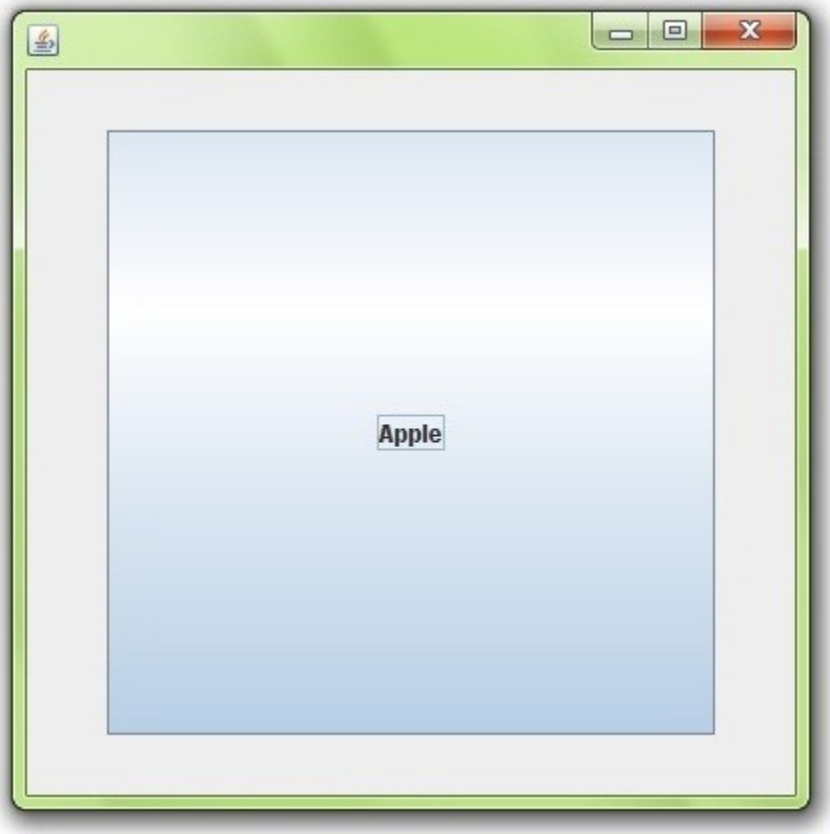
1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class:

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified

card with the given name.

Example of CardLayout class:




```
1. import java.awt.*;
2. import java.awt.event.*;
3.
4. import javax.swing.*;
5.
6. public class CardLayoutExample extends JFrame implements ActionListener{
7. CardLayout card;
8. JButton b1,b2,b3;
9. Container c;
10.    CardLayoutExample(){
11.
12.        c=getContentPane();
```

```
13.       card=new CardLayout(40,30);
14. //create CardLayout object with 40 hor space and 30 ver space
15.       c.setLayout(card);
16.
17.       b1=new JButton("Apple");
18.       b2=new JButton("Boy");
19.       b3=new JButton("Cat");
20.       b1.addActionListener(this);
21.       b2.addActionListener(this);
22.       b3.addActionListener(this);
23.
24.       c.add("a",b1);c.add("b",b2);c.add("c",b3);
25.
26.    }
27.    public void actionPerformed(ActionEvent e) {
28.    card.next(c);
29.    }
30.
31.    public static void main(String[] args) {
32.       CardLayoutExample cl=new CardLayoutExample();
33.       cl.setSize(400,400);
34.       cl.setVisible(true);
35.       cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
36.    }
37. }
```