

# GUI

*Dr. Hadeer A. Hosny*

**Second term  
2022-2021**

# GUI (Graphical User Interface)

## What are learned so far to interact with users?

- In a text-based UI the commands are entered from the keyboard.
- In a console program, the system usually controls user actions.

- > Enter number of classes: 3
- > Enter number of students: 15
- > You have 45 students

## Most modern programs use a GUI :

- **Graphical:** Not just text or characters but windows, menus, buttons, ..
- **User:** Person using the program
- **Interface:** Way to interact with the program

# GUI (New programming approach)

3

## What is the traditional way of programming?

- List of instructions performed in order.
- Next thing to happen is next thing in list.
- Program performed by one agent (computer).

## GUI is a Event-Driven Style of Programming.

- Objects that can fire events and objects that react to events.
- Next thing to happen depends on next event.
- Program is interaction between user and computer.

# GUI (New programming approach)

4

## Programs with GUIs often use Event-Driven Programming

A user interacts with the application by:

- Clicking on a button to choose a program option.
- Making a choice from a menu.
- Entering text in a text field.
- Dragging a scroll bar.
- Clicking on a window's close button.

Program waits for events to occur and then responds

- **Firing an event:** When an object generates an event
- **Listener:** Object that waits for events to occur
- **Event handler:** Method that responds to an event

# Java GUI on Screen

- **How do you put a GUI on the screen?**
  - Create a window (JFrame) object
  - Install components
    - Labels, buttons, etc
  - System manages the window and components by sending notification for user events
    - Drawing clicking typing
  - Components draw themselves

# Java AWT

## ➤ Abstract Windowing Toolkit

- Included in **first release of Java**
- Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.
- Java **AWT components are platform-dependent i.e. components are displayed according to the view of operating system.**
- For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix.
  - In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.
- **AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).**
- – import java.awt.\*
- The java.awt package provides classes for AWT API such as **TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.**

# Components

**An application** is a set of windows

**A window** : is a component itself and contains a set of components.

**A container**: is a component that contains other components, also a window is considered a container .

## Components:

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



# Components

## Container:

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

It is basically a screen where the components are placed at their specific locations. Thus it contains and controls the layout of components.

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog



# Components

## Container:

1. **Window** : The window is the container that have no borders and menu bars. **You must use frame, dialog** or another window for creating a window. We need to create an instance of Window class to create this container.
2. **Panel** : The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.
3. **Frame** : The Frame is **the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc.** Frame is most widely used container while developing an AWT application.

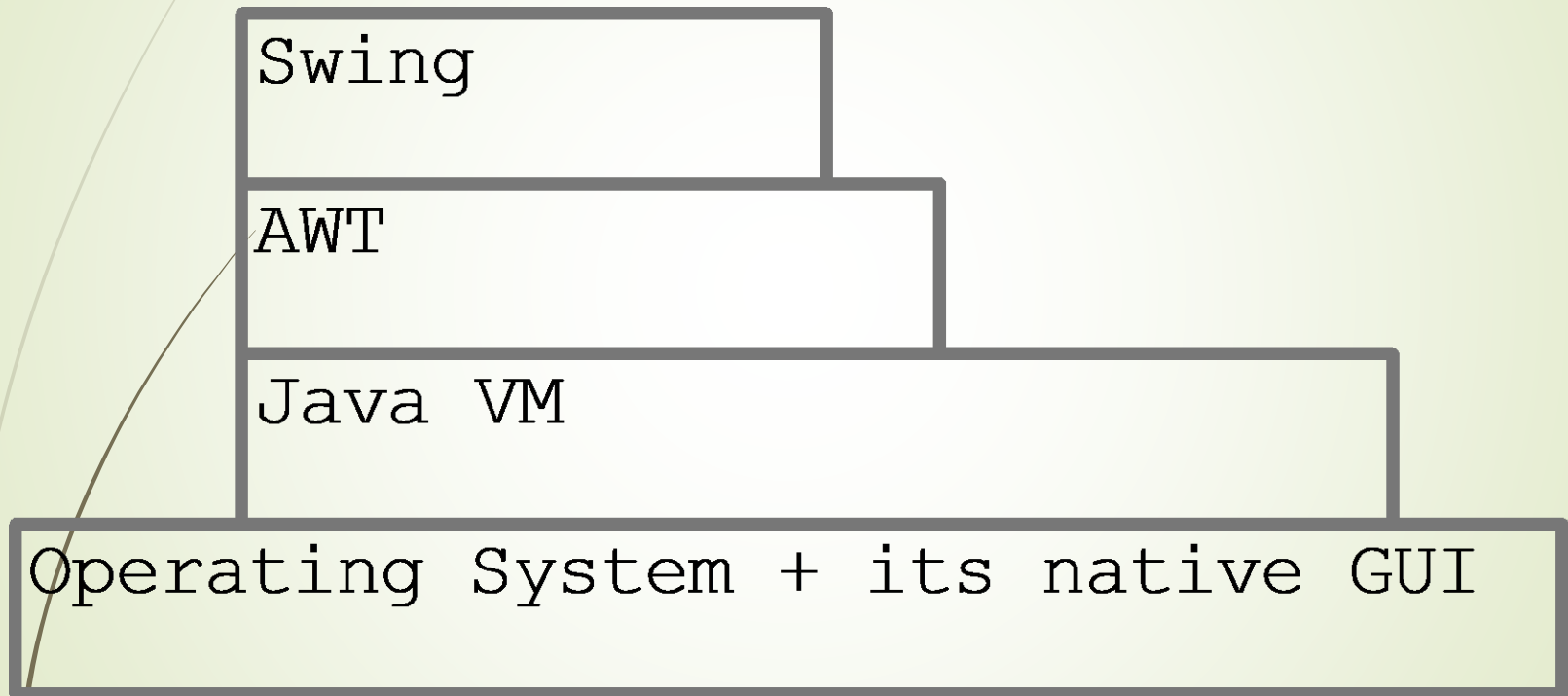
# Java Swing / JFC

- **Replacement/Enhancement for AWT**
- Java Swing tutorial 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, JColorChooser etc.

# Difference between AWT and Swing

Java AWT	Java Swing
AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.

# Java GUI Block Diagram



# Introduction to Swing classes

## ➤ **JComponent**

- Swing analog of the Object class
- Everything inherits from JComponent

## ➤ **JLabel**

- Built in JComponents that displays text
- Example: `new JLabel("Hello World!");`

## ➤ **JFrame**

- A single window
- Has a “content pane” JComponent that can hold other components
  - `frame.getContentPane()`
- Closing a frame simply hides it

# Content Pane / Layout Manager

- **Content pane** is a place holder
  - An empty board where you can place components
  - Use `add()` to put components on the content pane
- Content pane uses a “**Layout Manager**”
  - Programmer provides guidelines for how the interface should look by choosing the correct layout manager
  - Layout Manager determines the size and positioning of components on the content pane.

# Panels and the content pane

## Window:

- Visible
- Has borders, title, and maximize/minimize buttons

## Content Pane:

- Invisible
- Part of window

## Panel:

- Invisible container
- Contains components

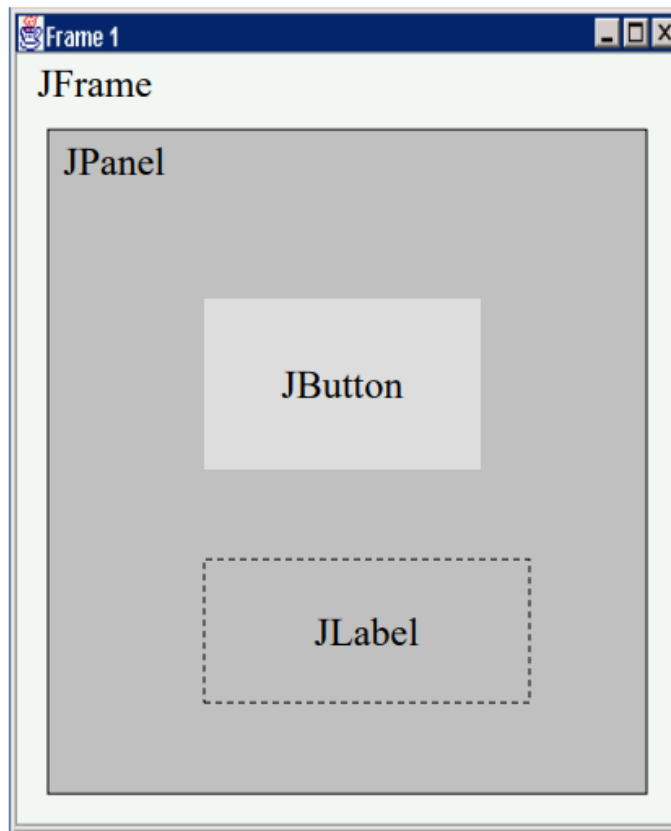
## Components:

- Visible
- Text field, button, radio

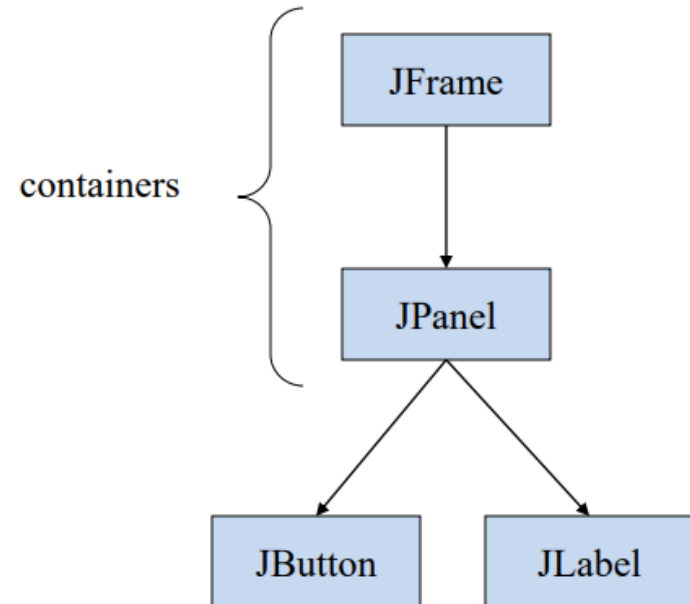


# Anatomy of an Application GUI

## GUI



## Internal structure



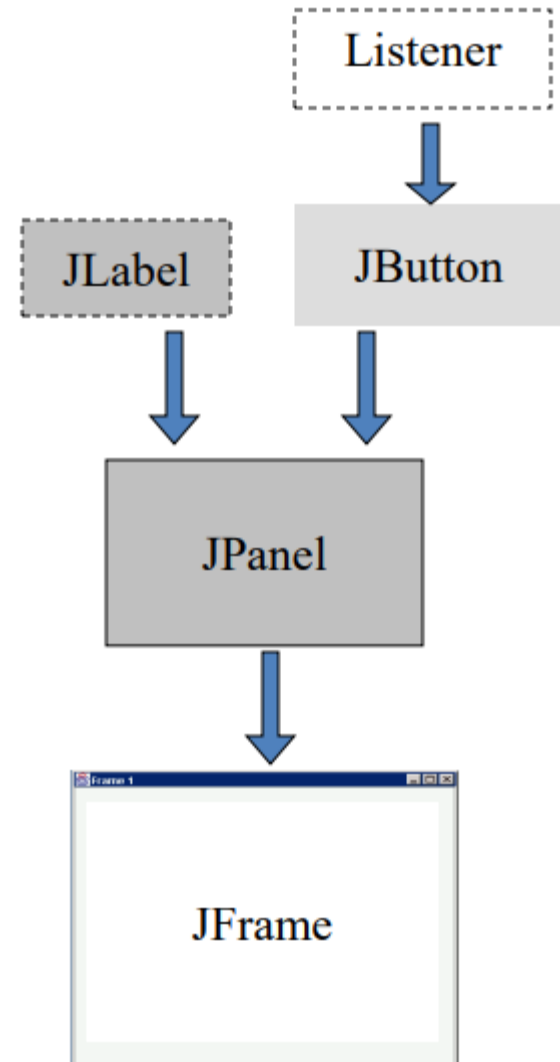
# Build from bottom up

## ➤ Create:

- Frame
- Panel
- Components
- Listeners

## ➤ Add: (bottom up)

- listeners into components
- components into panel
- panel into frame



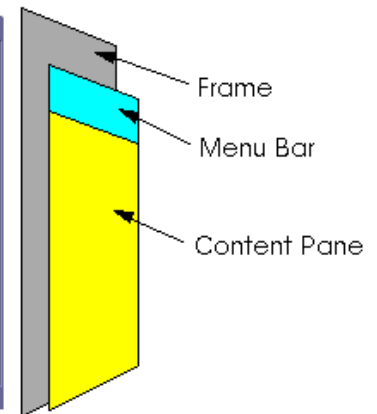
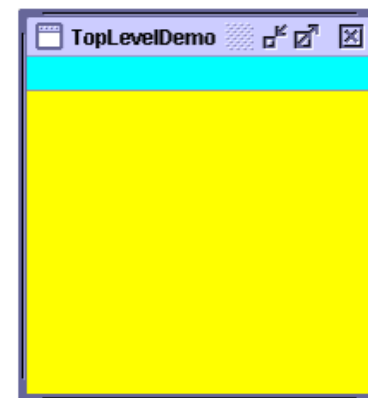
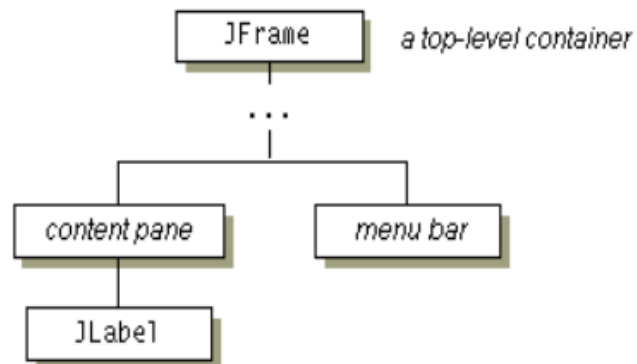
# Containment hierarchy

**Top level containers:** JFrame, JDialog, JApplet

**Content pane:** the main container in JApplet, JDialog, and JFrame Objects

**Basic controls:** JButton, JComboBox, List, Menu, Slider, JTextField, JLabel, progress bar, tool tip

**General purpose containers:** Panel, scroll pane, split pane, tabbed pane, tool bar



# GUI Building Process

## 1. First build the window (frame):

There are two ways to create a frame:

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

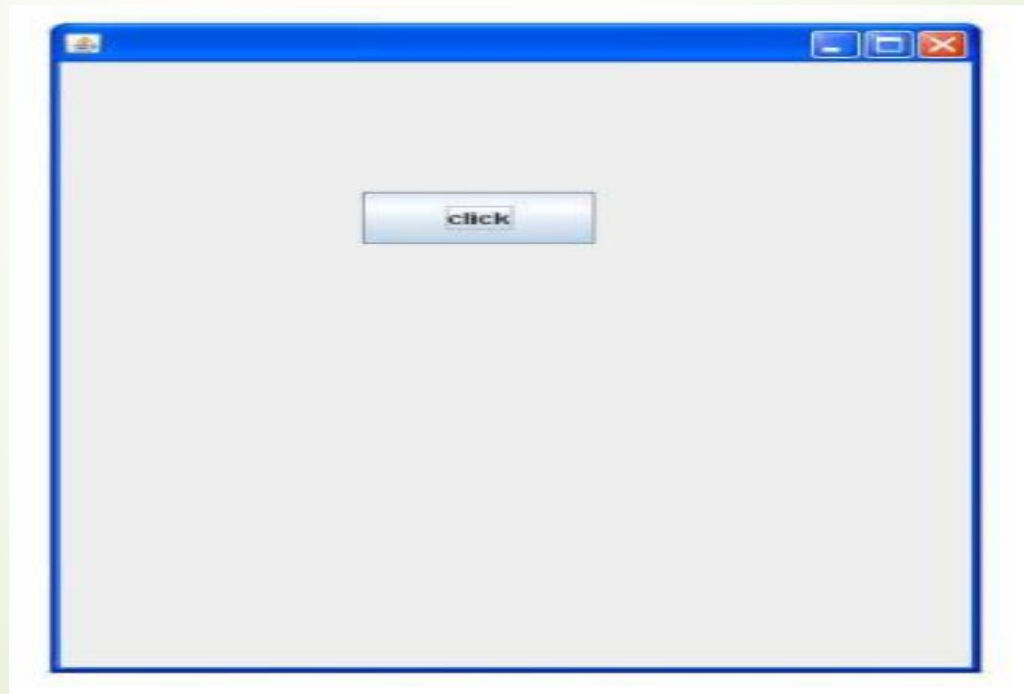
## 2. Create components (Button, Label, TextField)

## 3. Add components to a panel

## 4. Add the panel to the content pane of a window

# build the window (frame)

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




# build the window (frame)

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

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

# build the window (frame)

- By creating the object of Frame class (association)
- We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.



```
import javax.swing.*;
public class Simple {
    JFrame f;
    Simple(){
        f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);

        f.add(b);//adding button in JFrame

        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }

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



# build the window (frame)

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

```
import javax.swing.*;

public class Simple2 extends JFrame{//inheriting JFrame
    JFrame f;
    Simple2(){
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }

    public static void main(String[] args) {
        new Simple2();
    }
}
```

# Events

- **Event** :User action
- **Event Object** : Contains information about the action and is generated by Java
- **Event Source**: The component that has the event
- **Event Listener** : Code that is executed if a certain event takes place

# ActionListener interface

- The Java ActionListener is notified whenever you click on the button or menu item. It is notified against `ActionEvent`. The ActionListener interface is found in `java.awt.event` package. It has only one method: `actionPerformed()`.
- The `actionPerformed()` method is invoked automatically whenever you click on the registered component.
- `public abstract void actionPerformed(ActionEvent e);`

# ActionListener interface

## ► How to write ActionListener

### 1. Implement the ActionListener interface in the class:

```
public class ActionListenerExample implements ActionListener
```

### 2. Register the component with the Listener:

```
component.addActionListener(instanceOfListenerclass);
```

### 3. Override the actionPerformed() method:

```
public void actionPerformed(ActionEvent e){  
    //Write the code here  
}
```

# ActionEvent e

**Contains information about the event**

- **getActionCommand()**

**Event source, as a string**

- **getSource()**

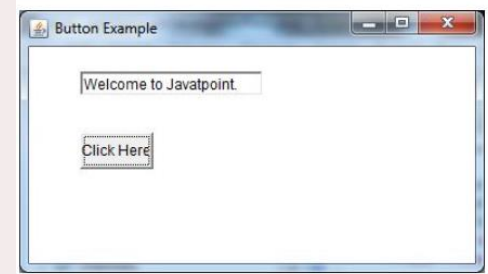
**Event source, as an object**

# Listeners

1. **import java.awt.event.\*;**
2. **Class that implements ActionListener interface**
3. **Code actionPerformed method**
4. **Register with event source**

```
import java.awt.*;
import java.awt.event.*;
//1st step
public class ActionListenerExample implements ActionListener{
    public static void main(String[] args) {
        Frame f=new Frame("ActionListener Example");
        final TextField tf=new TextField();
        tf.setBounds(50,50, 150,20);
        Button b=new Button("Click Here");
        b.setBounds(50,100,60,30);
        //2nd step
        b.addActionListener(this);
        f.add(b);f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    //3rd step
    public void actionPerformed(ActionEvent e){
        tf.setText("Welcome to Javatpoint.");
    }
}
```

new ActionListenerExample



29

1

2

4

3

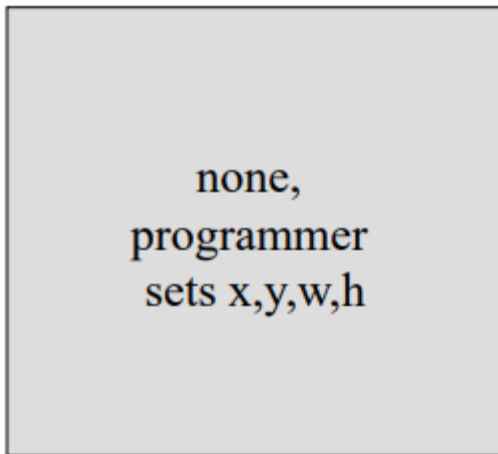


# Layout manager

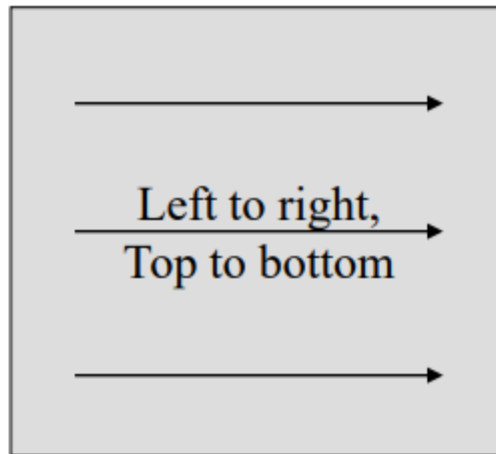
- The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers.
- **There are examples of classes that represent the layout managers:**
  - **java.awt.BorderLayout**
  - **java.awt.FlowLayout**
  - **java.awt.GridLayout**

# Layout manager examples

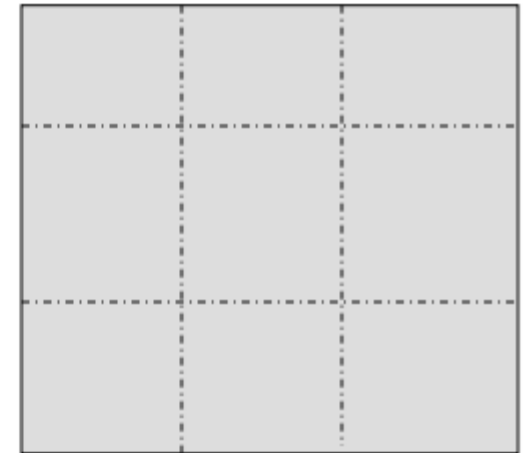
null



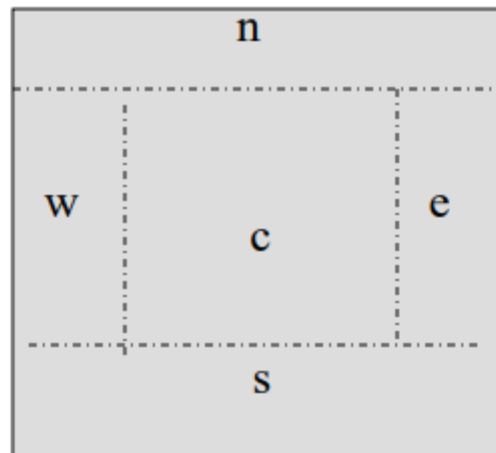
FlowLayout



GridLayout

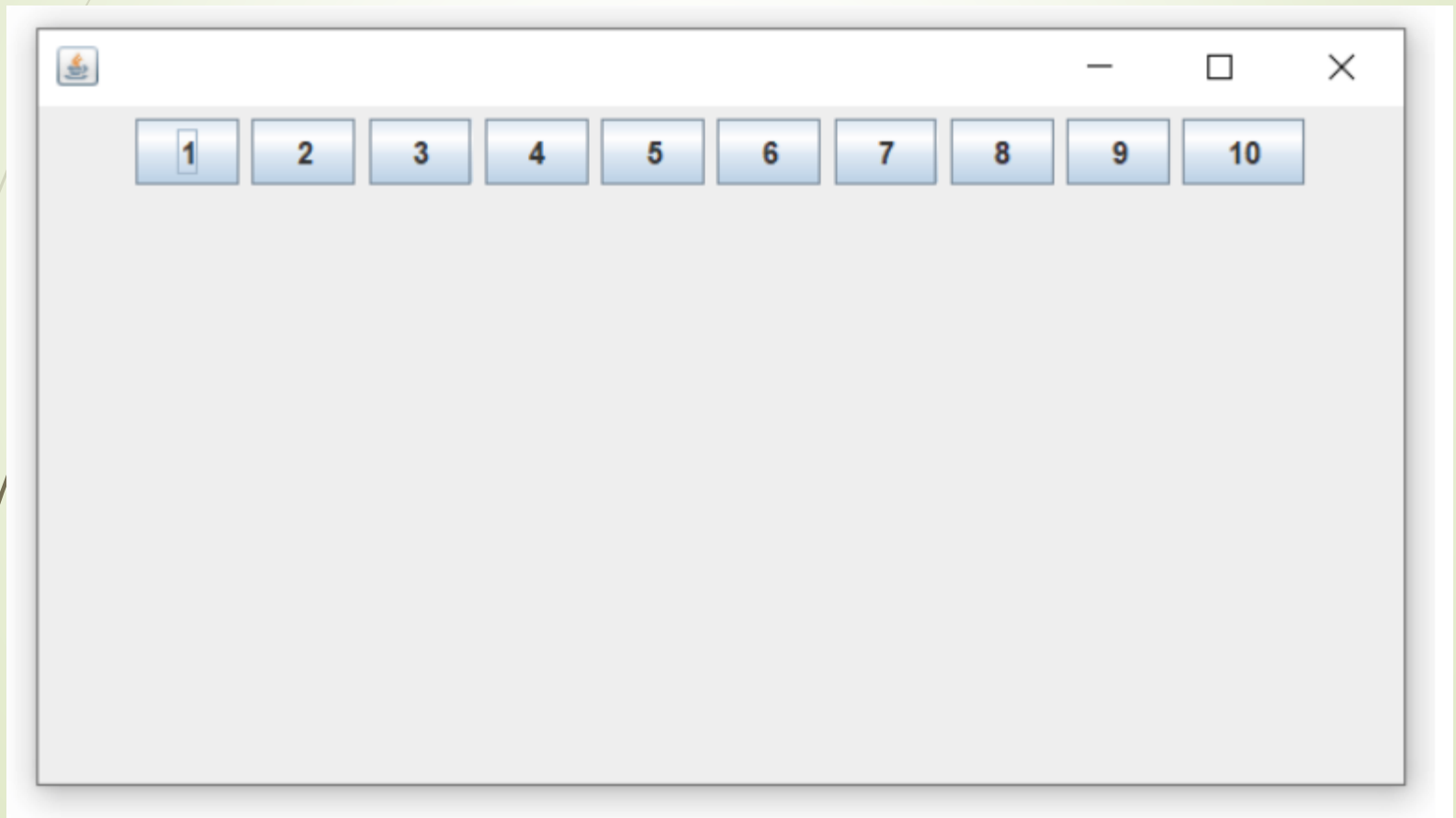


BorderLayout



# Flow layout

The Java `FlowLayout` class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.



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

public class FlowLayoutExample
{

    JFrame frameObj;

    // constructor
    FlowLayoutExample()
    {
        // creating a frame object
        frameObj = new JFrame();

        // creating the buttons
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        JButton b3 = new JButton("3");
        JButton b4 = new JButton("4");
        JButton b5 = new JButton("5");
        JButton b6 = new JButton("6");
        JButton b7 = new JButton("7");
```

```
JButton b8 = new JButton("8");
JButton b9 = new JButton("9");
JButton b10 = new JButton("10");
```

```
// adding the buttons to frame
```

```
frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);
frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);
frameObj.add(b9); frameObj.add(b10);
```

```
// parameter less constructor is used
```

```
// therefore, alignment is center
```

```
// horizontal as well as the vertical gap is 5 units.
```

```
frameObj.setLayout(new FlowLayout());
```

```
frameObj.setSize(300, 300);
```

```
frameObj.setVisible(true);
```

```
}
```

```
// main method
```

```
public static void main(String args[])
```

```
{
```

```
    new FlowLayoutExample();
```

```
}
```

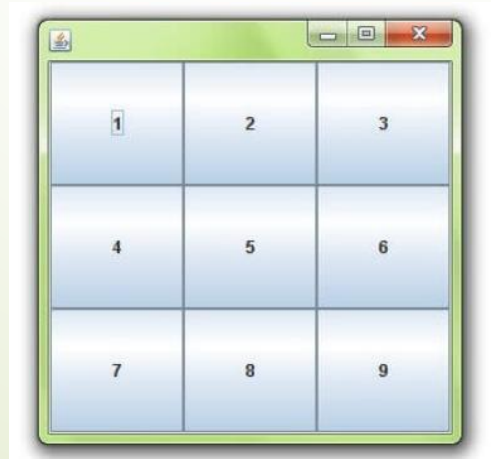
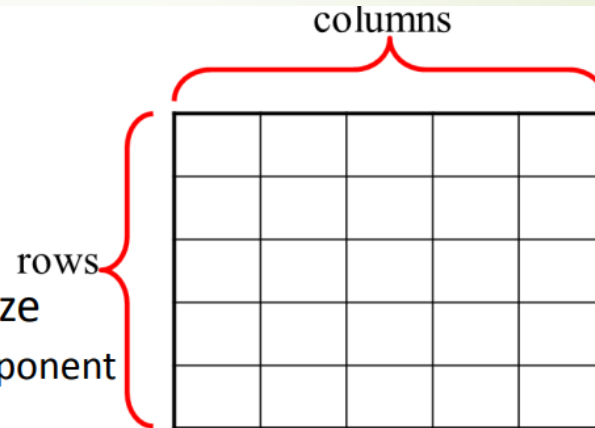
```
}
```

# Grid layout

The Java `GridLayout` class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.

## Grid of cells

- Rows x Columns
- One component per cell
- All cells are of the same size
  - The size of the largest component

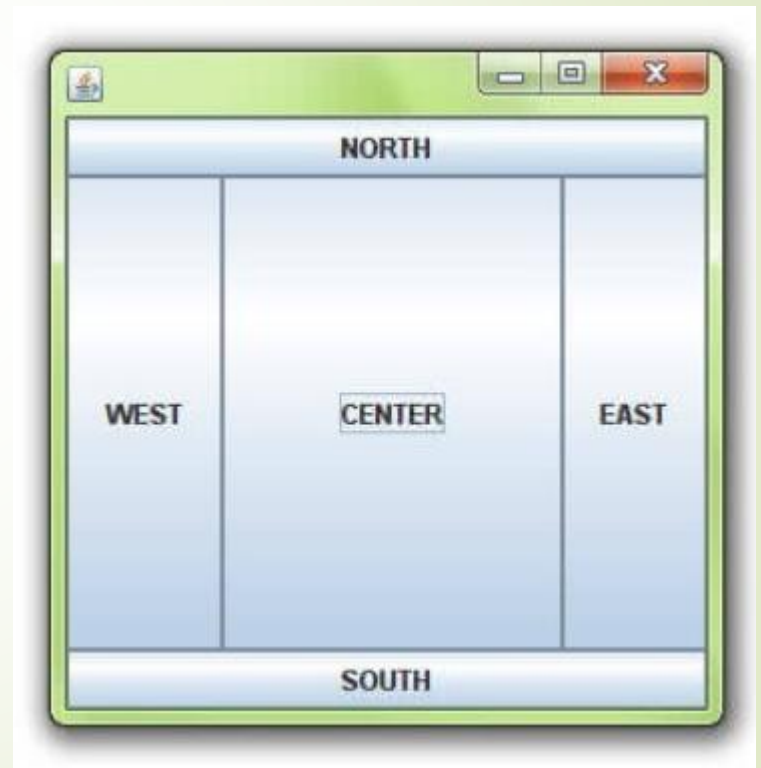
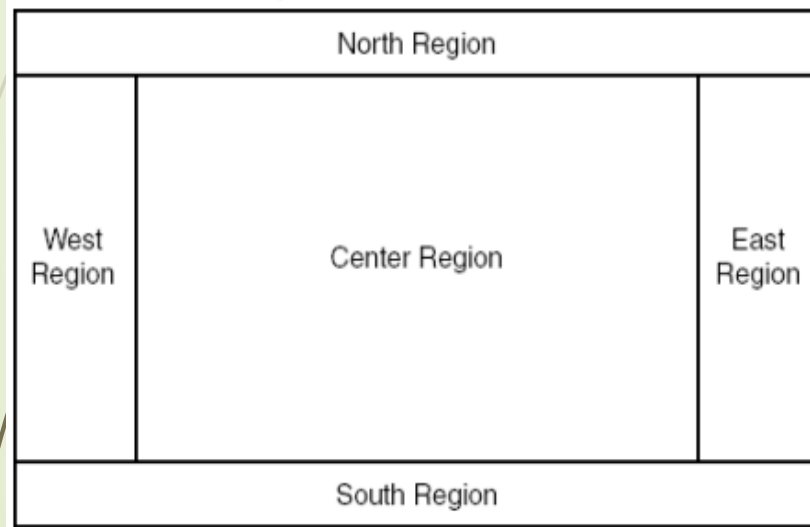


```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout{
    JFrame f;
    MyGridLayout(){
        f=new JFrame();
        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        JButton b6=new JButton("6");
        JButton b7=new JButton("7");
        JButton b8=new JButton("8");
        JButton b9=new JButton("9");
        // adding buttons to the frame
        f.add(b1); f.add(b2); f.add(b3);
        f.add(b4); f.add(b5); f.add(b6);
        f.add(b7); f.add(b8); f.add(b9);
```

```
        // setting grid layout of 3 rows and 3 columns
        f.setLayout(new GridLayout(3,3));
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyGridLayout();
    }
}
```

# Border layout

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





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

```
public class Border
```

```
{
    JFrame f;
    Border()
    {
        f = new JFrame();

        // creating buttons
        JButton b1 = new JButton("NORTH"); // the button will be labeled as NORTH
        JButton b2 = new JButton("SOUTH"); // the button will be labeled as SOUTH
        JButton b3 = new JButton("EAST"); // the button will be labeled as EAST
        JButton b4 = new JButton("WEST"); // the button will be labeled as WEST
        JButton b5 = new JButton("CENTER"); // the button will be labeled as CENTER

        f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction
        f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction
        f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction
        f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction
        f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center
    }
}
```

```
f.setSize(300, 300);
f.setVisible(true);
}

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

# JComponents

- JButton
- JLabel
- JTextField
- JTextArea
- JPasswordField
- JCheckBox
- JRadioButton
- JComboBox
- Jlist
- JOptionPane
- Mnemonics
- ToolTip

# JComponents

- Jslider
- JFileChooser
- JColorChooser
- JMenuBar
  - JMenuItem
  - JMenu
  - JPopupMenu
  - JCheckBoxMenuItem
  - JSeparator

# JButton

40

The JButton class is used to create a labeled button that has platform independent implementation.

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

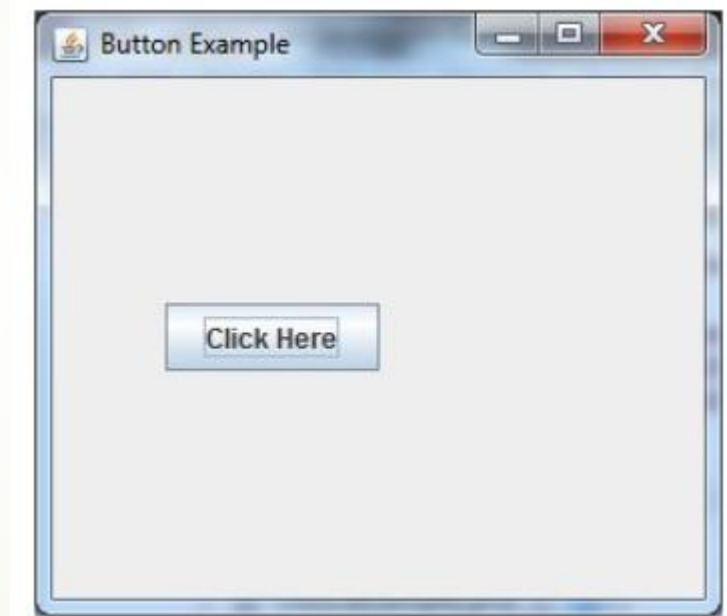
Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the <b>action listener</b> to this object.

# Jbutton example

41

```
import javax.swing.*;

public class ButtonExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



# JLabel

42

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text.

Constructor	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String s)	Creates a JLabel instance with the specified text.
JLabel(Icon i)	Creates a JLabel instance with the specified image.

Methods	Description
String getText()	It returns the text string that a label displays.
void setText(String text)	It defines the single line of text this component will display.
void setHorizontalAlignment(int alignment)	It sets the alignment of the label's contents along the X axis.
Icon getIcon()	It returns the graphic image that the label displays.
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

# JLabel example

43

```
import javax.swing.*;

class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



# JTextField

44

The object of a JTextField class is a text component that allows the editing of a single line text.

Constructor	Description
JTextField()	Creates a new TextField
JTextField(String text)	Creates a new TextField initialized with the specified text.

Methods	Description
String getText()	t returns the text string
void setText(String text)	It defines the single line of text this component will display.



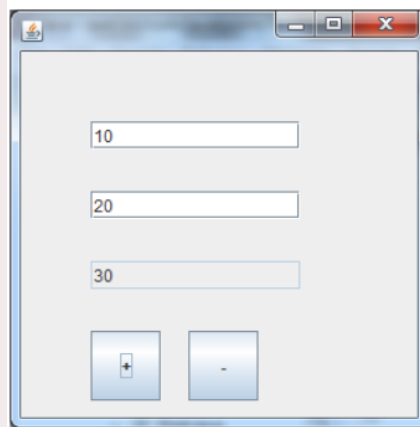
# JTextField example

45

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

public class TextFieldExample implements ActionListener{
    JTextField tf1,tf2,tf3;
    JButton b1,b2;

    TextFieldExample(){
        JFrame f= new JFrame();
        tf1=new JTextField();
        tf1.setBounds(50,50,150,20);
        tf2=new JTextField();
        tf2.setBounds(50,100,150,20);
        tf3=new JTextField();
        tf3.setBounds(50,150,150,20);
        tf3.setEditable(false);
        b1=new JButton("+");
        b1.setBounds(50,200,50,50);
        b2=new JButton("-");
        b2.setBounds(120,200,50,50);
        b1.addActionListener(this);
        b2.addActionListener(this);
        f.add(tf1);f.add(tf2);f.add(tf3);f.add(b1);f.add(b2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



```
public void actionPerformed(ActionEvent e) {
    String s1=tf1.getText();
    String s2=tf2.getText();
    int a=Integer.parseInt(s1);
    int b=Integer.parseInt(s2);
    int c=0;
    if(e.getSource()==b1){
        c=a+b;
    }else if(e.getSource()==b2){
        c=a-b;
    }
    String result=String.valueOf(c);
    tf3.setText(result);
}

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

# JTextArea

46

The object of a JTextArea class is a multi line region that displays text.

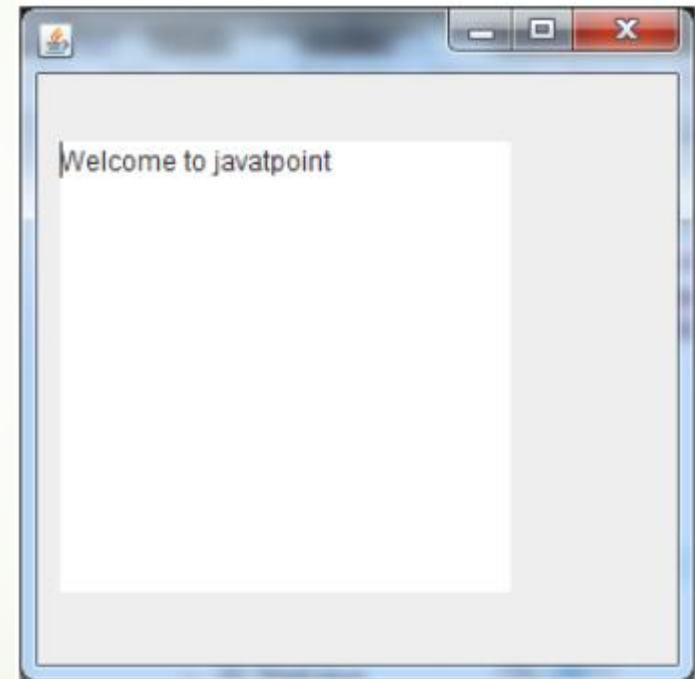
Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.

Methods	Description
String getText()	t returns the text string
void setText(String text)	It defines the single line of text this component will display.
void append(String s)	It is used to append the given text to the end of the document.

# JTextArea example

47

```
import javax.swing.*;
public class TextAreaExample
{
    TextAreaExample(){
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new TextAreaExample();
    }
}
```



# JPasswordField

48

The object of a JPasswordField class is a text component specialized for password entry.

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.

Methods	Description
String getText()	t returns the text string
void setText(String text)	It defines the single line of text this component will display.

# JPasswordField example

49

```
import javax.swing.*;

public class PasswordFieldExample {

    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        JPasswordField value = new JPasswordField();
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



# JCheckBox

50

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on".

Constructor	Description
JCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.

## Handling click events:

- `itemStateChanged()`

## Determine if a checkbox is selected

- `Check1.isSelected()`

## Selecting a checkbox programmatically

- `Check1.doClick()`

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

```
public class CheckBoxExample extends JFrame implements ActionListener{
```

```
    JLabel l;
```

```
    JCheckBox cb1,cb2,cb3;
```

```
    JButton b;
```

```
    CheckBoxExample(){
```

```
        l=new JLabel("Food Ordering System");
```

```
        l.setBounds(50,50,300,20);
```

```
        cb1=new JCheckBox("Pizza @ 100");
```

```
        cb1.setBounds(100,100,150,20);
```

```
        cb2=new JCheckBox("Burger @ 30");
```

```
        cb2.setBounds(100,150,150,20);
```

```
        cb3=new JCheckBox("Tea @ 10");
```

```
        cb3.setBounds(100,200,150,20);
```

```
        b=new JButton("Order");
```

```
        b.setBounds(100,250,80,30);
```

```
        b.addActionListener(this);
```

```
        add(l);add(cb1);add(cb2);add(cb3);add(b);
```

```
        setSize(400,400);
```

```
        setLayout(null);
```

```
        setVisible(true);
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);
```

```
    }
```

```
    public void actionPerformed(ActionEvent e){
```

```
        float amount=0;
```

```
        String msg="";
```

```
        if(cb1.isSelected()){
```

```
            amount+=100;
```

```
            msg="Pizza: 100\n";
```

```
        }
```

```
        if(cb2.isSelected()){
```

```
            amount+=30;
```

```
            msg+="Burger: 30\n";
```

```
        }
```

```
        if(cb3.isSelected()){
```

```
            amount+=10;
```

```
            msg+="Tea: 10\n";
```

```
        }
```

```
        msg+="-----\n";
```

```
        JOptionPane.showMessageDialog(this,msg+"Total: "+amount);
```

```
    }
```

```
    public static void main(String[] args) {
```

```
        new CheckBoxExample();
```

```
    }
```

```
}
```

The image displays a Java Swing application with two windows. The main window, titled "Food Ordering System", contains three checked items: "Pizza @ 100", "Burger @ 30", and "Tea @ 10". Below these items is an "Order" button. A secondary "Message" dialog box is open, displaying the selected items and their prices: "Pizza: 100", "Burger: 30", and "Tea: 10". It also shows the total price as "Total: 140.0" and includes an "OK" button.

Food Ordering System


☒ Pizza @ 100

☒ Burger @ 30

☒ Tea @ 10

Order

Message

 Pizza: 100  
Burger: 30  
Tea: 10

-----

Total: 140.0

OK



# JRadioButton

53

The **JRadioButton** class is used to create a radio button. It is used to choose one option from multiple options.

Constructor	Description
<code>JRadioButton()</code>	Creates an unselected radio button with no text.
<code>JRadioButton(String s)</code>	Creates an unselected radio button with specified text.
<code>JRadioButton(String s, boolean selected)</code>	Creates a radio button with the specified text and selected status.

Methods	Description
<code>void setText(String s)</code>	It is used to set specified text on button.
<code>String getText()</code>	It is used to return the text of the button.
<code>void setEnabled(boolean b)</code>	It is used to enable or disable the button.

# ButtonGroup

54

Groups a set of radio buttons, so that only one radio button is selected at any time. We will add radio button to button group to choose one choice only.

```
import javax.swing.*;
public class RadioButtonExample {
    JFrame f;
    RadioButtonExample(){
        f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");
        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);
        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 RadioButtonExample();
    }
}
```



# JRadioButton

55

## Responding to a click

- `actionPerformed(ActionEvent e)`

## Determining whether a radio button is selected

- `Radio.isSelected()`

## Selecting a radio button programmatically

- `Radio.doClick()`

## Allowing a radio button to programmatically change the selection status.

- `Radio.setSelected(boolean)`

## Specifying a alt-Key alternative for selecting the RadioButton.

- `Radio.setMnemonic(char)`

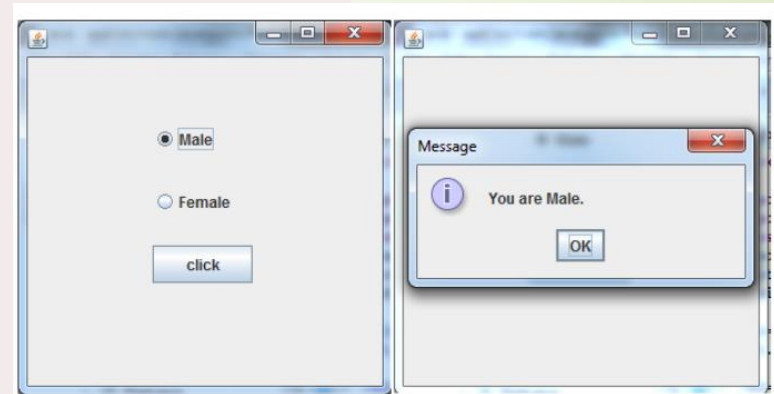
```

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

class RadioButtonExample extends JFrame implements ActionListener{
    JRadioButton rb1,rb2;
    JButton b;

    RadioButtonExample(){
        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 RadioButtonExample();
}

```

# JComboBox

57

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

Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(Object[] items)	Creates a JComboBox that contains the elements in the specified array.

Methods	Description
void addItem(Object anObject)	It is used to add an item to the item list.
void removeItem(Object anObject)	It is used to delete an item to the item list.
void removeAllItems()	It is used to remove all the items from the list.
void setEditable(boolean b)	It is used to determine whether the JComboBox is editable.

# JComboBox

58

Pass an array of objects that are to be displayed as the items in the drop-down list to the constructor.

```
String[] names = { "Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip",  
"Susan" }; JComboBox nameBox = new JComboBox(names);
```

The button displays the item that is currently selected.

The first item in the list is automatically selected when the combo box is displayed.

When the user clicks on the button, the drop-down



# JComboBox events

59

- When an item in a JComboBox object is selected, it generates an action event.
- Handle action events with an action event listener class, which must have an **actionPerformed** method.
- When the user selects an item in a combo box, the combo box executes its action event listener's **actionPerformed** method, passing an **ActionEvent** object as an argument.
- There are two methods in the JComboBox class that can be used to determine which item in a list is currently selected: –
  - **getSelectedItem**
  - **getSelectedIndex**
- The **getSelectedItem** method returns a reference to the item that is currently selected.
  - **String selectedName;**
  - **selectedName = (String) nameBox.getSelectedItem();**
- The **getSelectedItem** returns an Object reference so we cast the return value to a String.



# JComboBox events

60

- The `getSelectedIndex` method returns the index of the selected item.
  - `String[] names = { "Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip", "Susan" };`
  - `JComboBox nameBox = new JComboBox(names);`
- Get the selected item from the names array:
  - `int index;`
  - `String selectedName;`
  - `index = nameBox.getSelectedIndex();`
  - `selectedName = names[index];`



```

import javax.swing.*;
import java.awt.event.*;
public class ComboBoxExample {
    JFrame f;

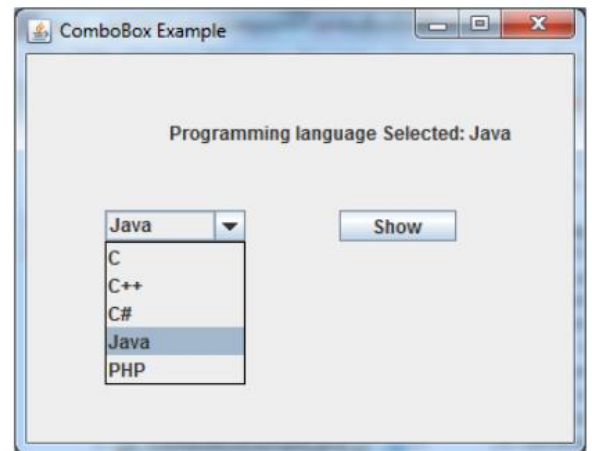
    ComboBoxExample(){
        f=new JFrame("ComboBox Example");
        final JLabel label = new JLabel();
        label.setHorizontalAlignment(JLabel.CENTER);
        label.setSize(400,100);
        JButton b=new JButton("Show");
        b.setBounds(200,100,75,20);
        String languages[]={"C","C++","C#","Java","PHP"};
        final JComboBox cb=new JComboBox(languages);
        cb.setBounds(50, 100,90,20);
        f.add(cb); f.add(label); f.add(b);
        f.setLayout(null);
        f.setSize(350,350);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "Programming language Selected: "
                    + cb.getItemAt(cb.getSelectedIndex());
                label.setText(data);
            }
        });
    }
}

```

```

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

```



# JList

62

The object of JList class represents a list of text items. The list of text items can be set up so that the user can choose either one item or multiple items.

Constructor	Description
JList()	Creates a JList with an empty, read-only, model.
JList(ary[] listData)	Creates a JList that displays the elements in the specified array.

The JList component uses the array to create the list of items.

- **String[] names = { "Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip", "Susan" };**
- **JList nameList = new JList(names);**

Methods	Description
int getSelectedIndex()	It is used to return the smallest selected cell index.
void setListData(Object[] listData)	It is used to create a read-only ListModel from an array of objects.

# JList

63

The **setListData** method allows the adding of items in an existing JList component.

## Example:

```
String[] names = { "Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip",  
"Susan" };  
nameList.setListData(names);
```

You may use: **getSelectedValue** or **getSelectedIndex** methods to determine which item in a list is currently selected.

**getSelectedValue** returns a reference to the item that is currently selected. **String selectedName;**  
**selectedName=(String)nameList.getSelectedValue();**

The return value must be cast to String is required in order to store it in the selectedName variable.

If no item in the list is selected, the method returns null.

# JList

64

The **getSelectedIndex** method returns the index of the selected item, or -1 if no item is selected.

The first item has the index 0.

You can use the index of the selected item to retrieve the item from an array:

```
String[] names = { "Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip",  
"Susan" }; JList nameList = new JList(names);
```

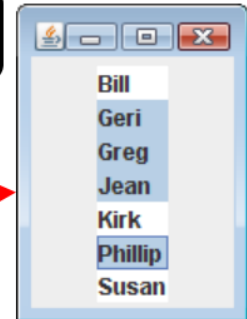
## List Selection Modes:

The JList component can operate in any of three selection modes



Single selection mode allows only one item to be selected at a time.

```
L1.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```



Multiple interval selection mode allows multiple items to be selected with no restrictions.

```
L1.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
```



Single interval selection mode allows a single interval of contiguous items to be selected.

```
L1.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
```

# JList

## List Selection Modes:

The method accepts an int argument that determines the selection mode:

- **ListSelectionMode.SINGLE\_SELECTION**
- **ListSelectionMode.SINGLE\_INTERVAL\_SELECTION**
- **ListSelectionMode.MULTIPLE\_INTERVAL\_SELECTION**

**Example: –**

```
nameList.setSelectionMode(ListSelectionMode.SINGLE_SELECTION);
```

## Single Interval Selection Mode :

- The **getSelectedValue** method returns the first item in the selected interval.
- The **getSelectedIndex** method returns the index of the first item in the selected interval.
- The **getSelectedValues** method. This method returns an array of objects, which are the items in the selected interval.
- The **getSelectedIndices** method returns an array of int values that are the indices of all the selected items in the list



## Multiple Interval Selection Mode:

The user holds down the Ctrl key while clicking on an item – it selects the item without deselecting other items.

- The **getSelectedValue** method returns the first item in the selected interval.
- The **getSelectedIndex** method returns the index of the first item in the selected interval.
- The **getSelectedValues** method. This method returns an array of objects, which are the items in the selected interval.
- The **getSelectedIndices** method returns an array of int values that are the indices of all the selected items in the list

```

import javax.swing.*;
import java.awt.event.*;
import java.util.List;

public class ListExample {

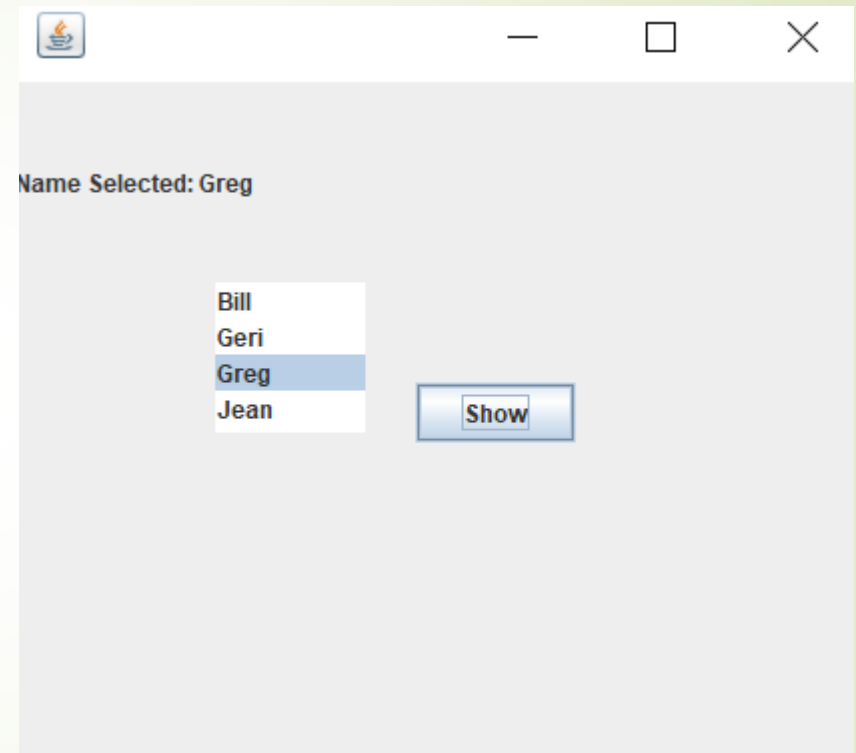
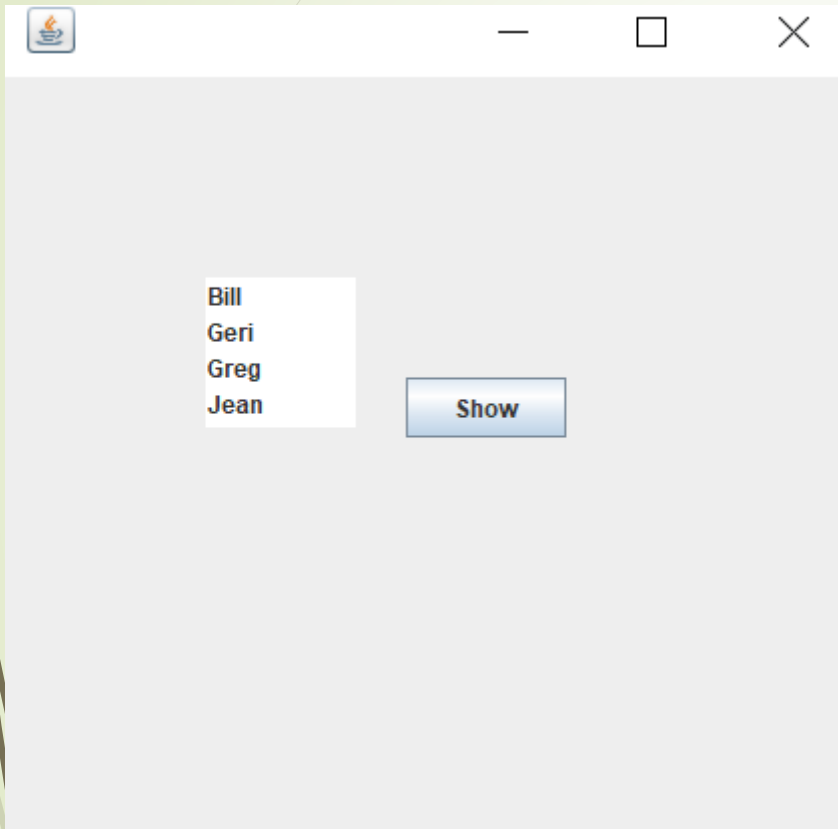
    ListExample () {
        JFrame f = new JFrame();
        JLabel label = new JLabel();
        label.setSize(500, 100);
        JButton b = new JButton("Show");
        b.setBounds(200, 150, 80, 30);
        String[] names = {"Bill", "Geri", "Greg", "Jean", "Kirk", "Phillip", "Susan"};
        JList nameList = new JList(names);
        JList list1 = new JList(names);
        list1.setBounds(100, 100, 75, 75);
        f.add(list1);
        f.add(b);
        f.add(label);
        f.setSize(450, 450);
        f.setLayout(null);
        f.setVisible(true);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                String data = "";
                if (list1.getSelectedIndex() != -1) {
                    data = "Name Selected: " + list1.getSelectedValue();
                    label.setText(data);
                }

                label.setText(data);
            }
        });
    }

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

```





# JOptionPane

70





- The **JOptionPane** class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.
- These dialog boxes are used to display information or get input from the user.

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType)	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

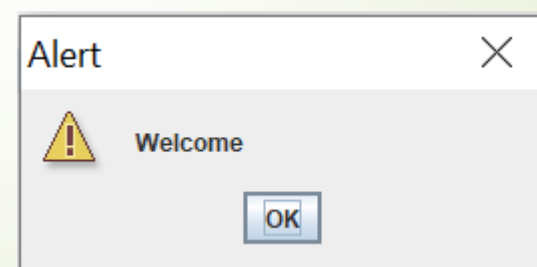
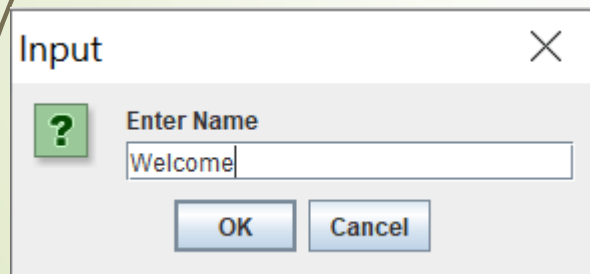
# JOptionPane

71

Methods	Description
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.

Message dialog type	Icon	Description
<b>ERROR_MESSAGE</b>		A dialog that indicates an error to the user.
<b>INFORMATION_MESSAGE</b>		A dialog with an informational message to the user.
<b>WARNING_MESSAGE</b>		A dialog warning the user of a potential problem.
<b>QUESTION_MESSAGE</b>		A dialog that poses a question to the user. This dialog normally requires a response, such as clicking a <b>Yes</b> or a <b>No</b> button.
<b>PLAIN_MESSAGE</b>	no icon	A dialog that contains a message, but no icon.

```
import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample() {
        f=new JFrame();
        String name=JOptionPane.showInputDialog(f,"Enter Name");
        JOptionPane.showMessageDialog(f,name,"Alert",JOptionPane.WARNING_MESSAGE);
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}
```



# Mnemonics

73

- A **mnemonic** is a key that you press in combination with the Alt key to quickly access a component.
- These are sometimes referred to as **hot keys**.
- If the letter is in the component's text, the first occurrence of that letter will appear underlined.
- If the letter does not appear in the component's text, then no letter will appear underlined.
- A hot key is assigned to a component through the component's **setMnemonic** method.

# Mnemonics

74

- The key codes are predefined constants in the **KeyEvent class** (**java.awt.event package**).
- These constants take the form: – **KeyEvent.VK\_x**, where **x** is a **key** on the keyboard.
- The letters **VK** in the constants stand for “**virtual key**”.
- To assign **the A key as a mnemonic**, use **KeyEvent.VK\_A**.

**Example:**

```
JButton exitButton = new JButton("Exit");  
exitButton.setMnemonic(KeyEvent.VK_X);
```

**Note :** You can also assign mnemonics to radio buttons and check boxes

# ToolTip

75

- You can create a tool tip for any **JComponent** with **setToolTipText()** method. This method is used to set up a tool tip for the component.

```
import javax.swing.*;

public class ToolTipExample {

    public static void main(String[] args) {
        JFrame f=new JFrame("Password Field Example");
        //Creating PasswordField and label
        JPasswordField value = new JPasswordField();
        value.setBounds(100,100,100,30);
        value.setToolTipText("Enter your Password");
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        //Adding components to frame
        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```





# Jslider

76

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

Constructor	Description
JSlider()	creates a slider with the initial value of 50 and range of 0 to 100.

Method	Description
public void setMinorTickSpacing(int n)	is used to set the minor tick spacing to the slider.
public void setMajorTickSpacing(int n)	is used to set the major tick spacing to the slider.
public void setPaintTicks(boolean b)	is used to determine whether tick marks are painted.
public void setPaintLabels(boolean b)	is used to determine whether labels are painted.
public void setPaintTracks(boolean b)	is used to determine whether track is painted.



```
import javax.swing.*;

public class SliderExample extends JFrame{

    public SliderExample() {
        JSlider slider = new JSlider(JSlider.HORIZONTAL, 0, 50, 25);
        slider.setMinorTickSpacing(2);
        slider.setMajorTickSpacing(10);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);

        JPanel panel=new JPanel();
        panel.add(slider);
        add(panel);
    }

    public static void main(String s[]) {
        SliderExample frame=new SliderExample();
        frame.pack();
        frame.setVisible(true);
    }
}
```



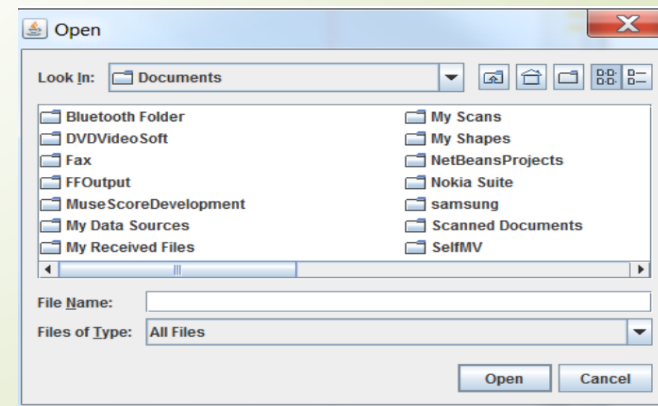
# JFileChooser

78

- The object of JFileChooser class represents a dialog window from which the user can select file.

Constructor	Description
JFileChooser()	Constructs a JFileChooser pointing to the user's default directory.

1. **JFileChooser FC =new JFileChooser();**
2. **Ret= FC.showOpenDialog(null);**
3. **if (Ret== JFileChooser.APPROVE\_OPTION)**  
**FName=FC.getSelectedFile().getPath());**



# JColorChooser

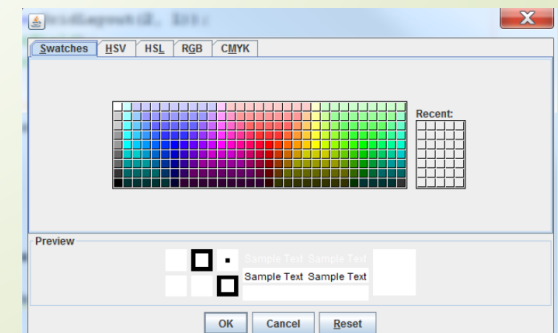
79

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

Constructor	Description
JColorChooser()	It is used to create a color chooser panel with white color initially.

Method	Description
static Color showDialog(Component c, String title, Color initialColor)	It is used to show the color chooser dialog box.

1. JColorChooser colorChooser = new JColorChooser();
2. Color color;
3. color=colorChooser.showDialog(null, null, null);
4. colorTextField.setBackground(color);



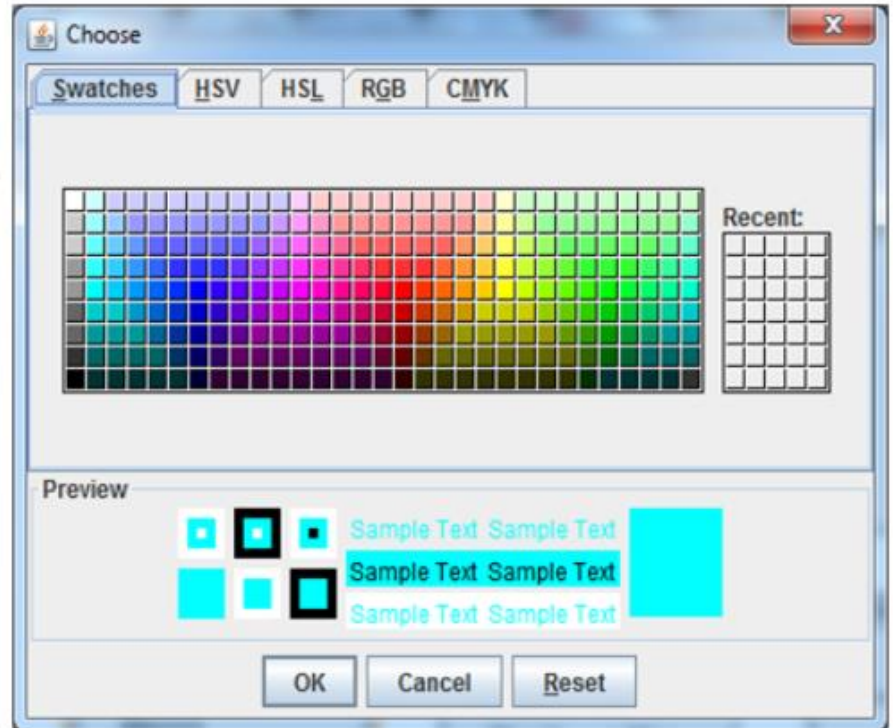
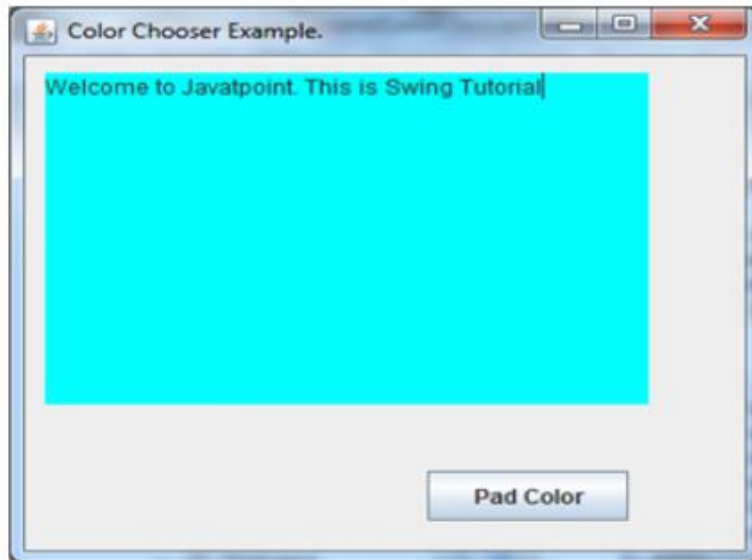
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ColorChooserExample extends JFrame implements ActionListener{
    JFrame f;
    JButton b;
    JTextArea ta;

    ColorChooserExample(){
        f=new JFrame("Color Chooser Example.");
        b=new JButton("Pad Color");
        b.setBounds(200,250,100,30);
        ta=new JTextArea();
        ta.setBounds(10,10,300,200);
        b.addActionListener(this);
        f.add(b);f.add(ta);
        f.setLayout(null);
        f.setSize(400,400);
        f.setVisible(true);
    }

    public void actionPerformed(ActionEvent e){
        Color c=JColorChooser.showDialog(this,"Choose",Color.CYAN);
        ta.setBackground(c);
    }

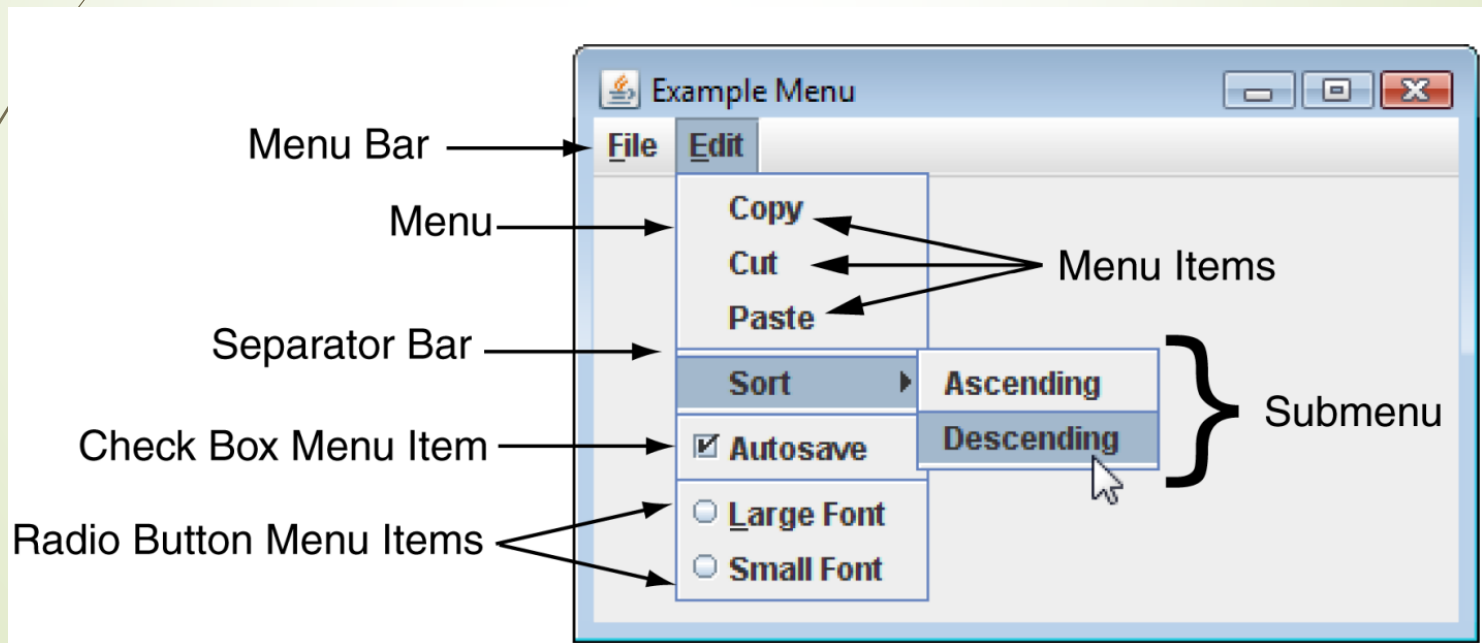
    public static void main(String[] args) {
        new ColorChooserExample();
    }
}
```



# JMenuBar

82

- The **JMenuBar** class is used to display menubar on the window or frame. It may have several menus.
- The object of **JMenu** class is a pull down menu component which is displayed from the menu bar.
- The object of **JMenuItem** class adds a simple labeled menu item.



# JMenuBar

83

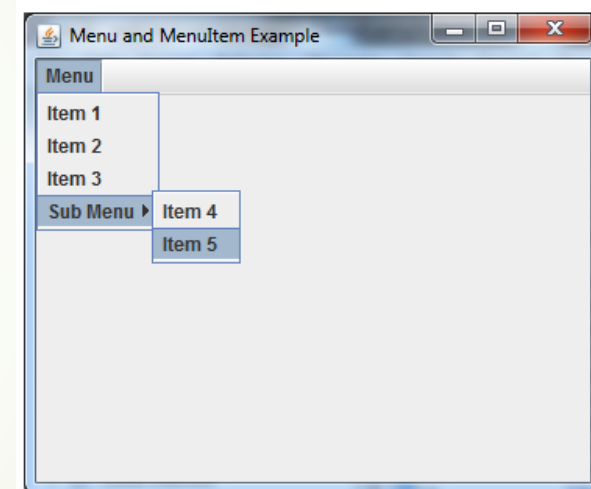
- **JCheckBoxMenuItem** – Used to create a check box menu item.
- **JRadioButtonMenuItem** – Used to create a radio button menu item.
- The object of **JSeparator** class is used to provide a general purpose component for implementing divider lines. It is used to draw a line to separate widgets in a Layout.



```

import javax.swing.*;
class MenuExample
{
    JMenu menu, submenu;
    JMenuItem i1, i2, i3, i4, i5;
    MenuExample(){
        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1); menu.add(i2); menu.add(i3);
        submenu.add(i4); submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
        new MenuExample();
    }
}

```





```
import javax.swing.*;
import java.awt.*;
public class SeparatorExample
{
    public static void main(String args[]) {
        JFrame f = new JFrame("Separator Example");
        f.setLayout(new GridLayout(0, 1));
        JLabel l1 = new JLabel("Above Separator");
        f.add(l1);
        JSeparator sep = new JSeparator();
        f.add(sep);
        JLabel l2 = new JLabel("Below Separator");
        f.add(l2);
        f.setSize(400, 100);
        f.setVisible(true);
    }
}
```

