Chapter 5:

Graphics User Interface(GUI) and Event-Driven Programing

Basics In Java Gui

GUI: Concept

GUI stands for "**Graphical User Interface**" and is pronounced **"gooey.“.** It is a user interface that is mainly made of graphical components like buttons, labels, windows, etc. through which the user can interact with an application

What is GUI is made of?

A GUI program consists of four types of software:

1. Graphical Components that make up the Graphical User Interface. A component **is an object having a graphical representation** that can be displayed on the screen and that can interact with the user. Examples of components are the buttons, checkboxes, and scrollbars of a typical graphical user interface.
2. Layout managers that enable us to control the way in which visual components are arranged in the GUI
3. Listener methods that receive the events and respond to them.
4. Application methods that do useful work for the user.

AWT and Swing are used to develop **window-based applications** in Java.

5. AWT
6. Swing

AWT

AWT

Awt is an abstract window toolkit that provides **various component classes** like Label, Button, TextField, etc., to show window components on the screen. All these classes are part of the **Java.awt package**

AWT Peers and Platform Dependency

Each GUI component that AWT provides has a peer.

A peer is the **implementation** of that **component in the native environment.**

With one exception(The LightWeightPeer), there is a one-to-one correspondence between Component classes and peer interfaces: a Window has a WindowPeer, a Checkbox has a CheckboxPeer, and so on.

AWT translates its various visual components into their corresponding, platform-specific equivalents or peers. This means that t**he look and feel of a component is defined by the**

platform, not by java. Hence they are **platform dependent**.

Heavyweight vs Lightweight Components

A heavyweight component is one that is **associated with its own native screen resource** (commonly known as a peer).

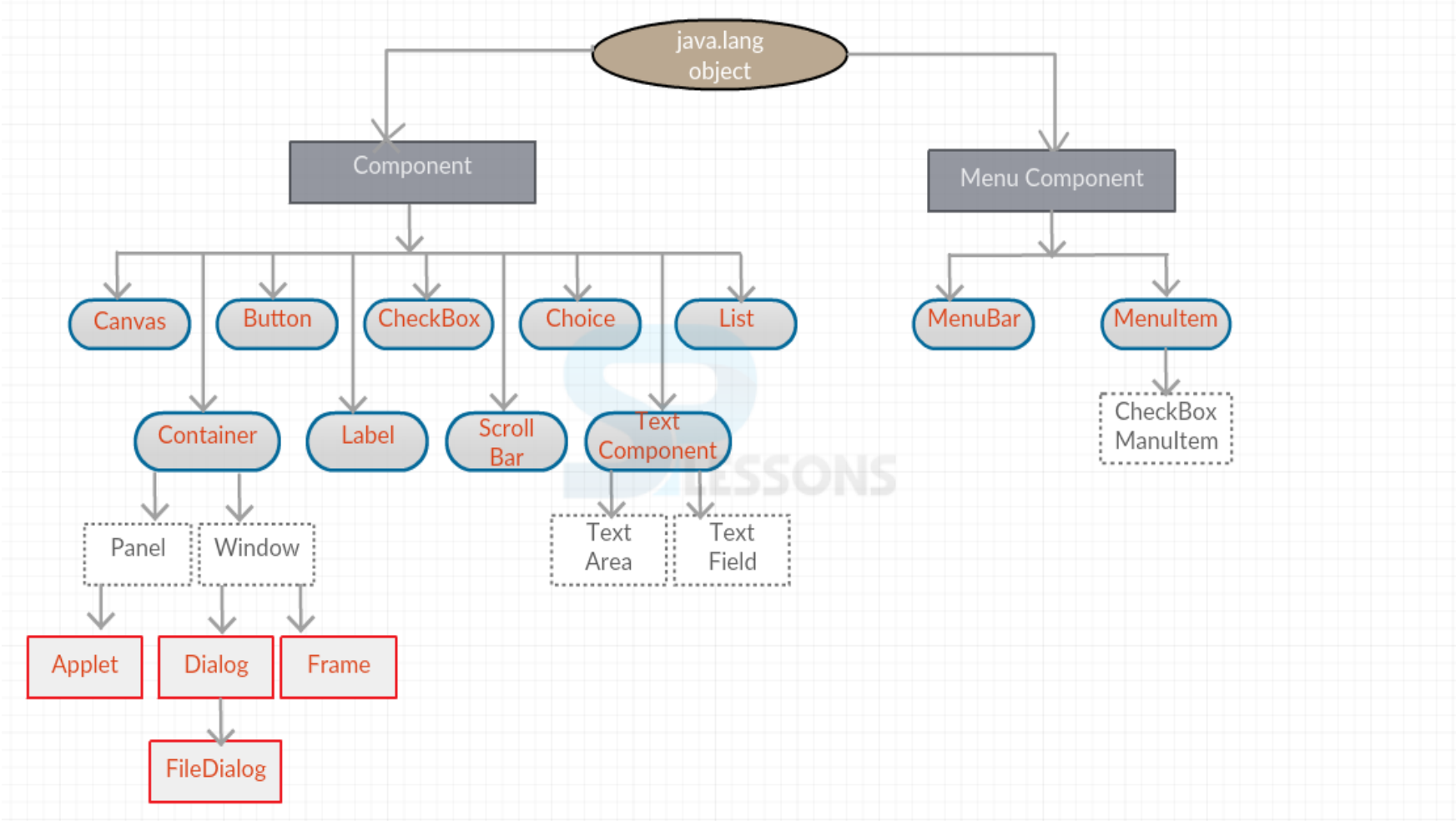
A lightweight component is one that "borrows" the screen resource of an ancestor (which means it has no native resource of its own -- so it's "lighter").

The main differences between heavyweight and lightweight are:

- A lightweight component can have **transparent pixels**; a heavyweight is always opaque.
- A lightweight component can appear to **be non-rectangular** because of its ability to set transparent areas; a heavyweight can only be rectangular.
- **Mouse events on a lightweight component fall through to its parent;** mouse events on a heavyweight component do not fall through to its parent.
- When a lightweight component overlaps a heavyweight component, the heavyweight component is always on top, regardless of the relative z-order of the two components

AWT Class Hierarchy

The AWT set is structured in a class hierarchy



SWING

Swing in Java is **a Graphical User Interface (GUI) toolki**t that includes the GUI components.

Swing provides **a rich set of widgets** and packages to make **sophisticated GUI components** for Java applications such as **tables, trees, sliders, progress bars, internal frames, and text components.**

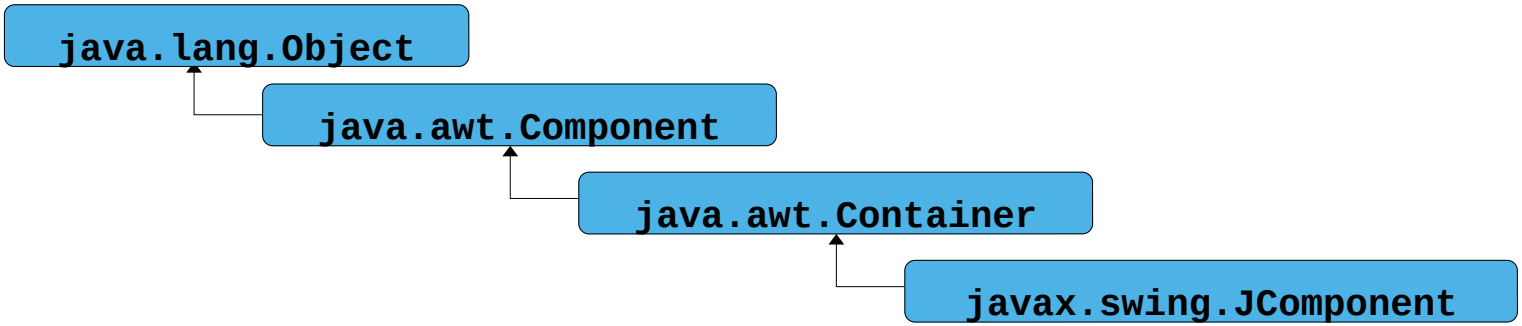
Swing is **a part of Java Foundation Classes(JFC),** which is an API for Java programs that provide GUI.
It is **built on the top of AWT** (Abstract Windowing Toolkit) API and **entirely written in java.**

Swing **supports a pluggable look and feel**.

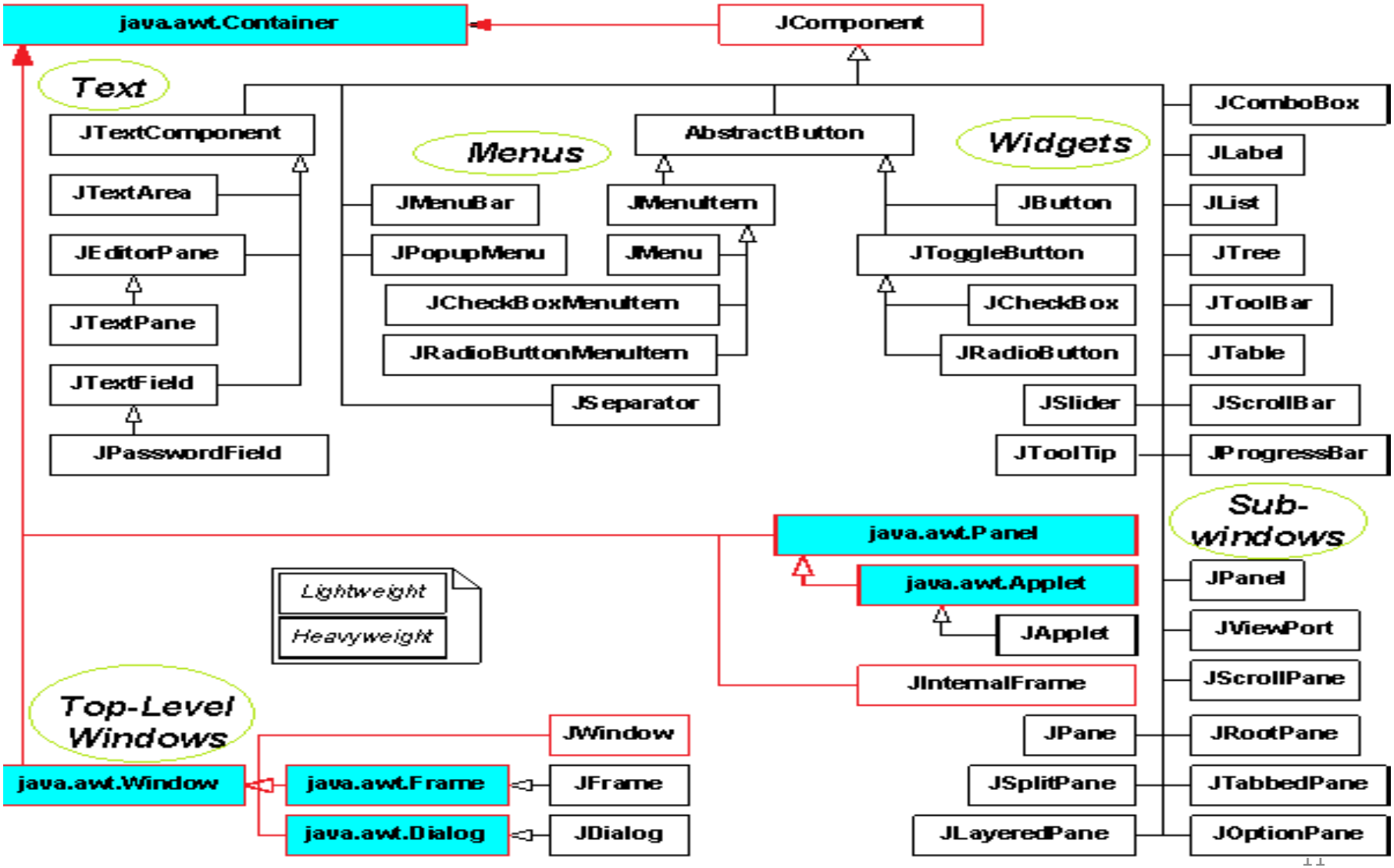
you can create **your own look-and-feel** for each one of the Swing components!

Swing **supports drag and drop(dnd)** and **clipboard(via cut/copy/paste)** data transfer

AWT and Swing Relationship



SWING Class Hierarchy



SWING Components

Java Swing Framework contains **a large set of components** which provide rich functionalities and allow **high level of customization**.

They all are **derived from JComponent** class.

You MUST import the following packages in order to use swing:

import java.awt.*;

import javax.swing.*;

All these components are lightweight components.

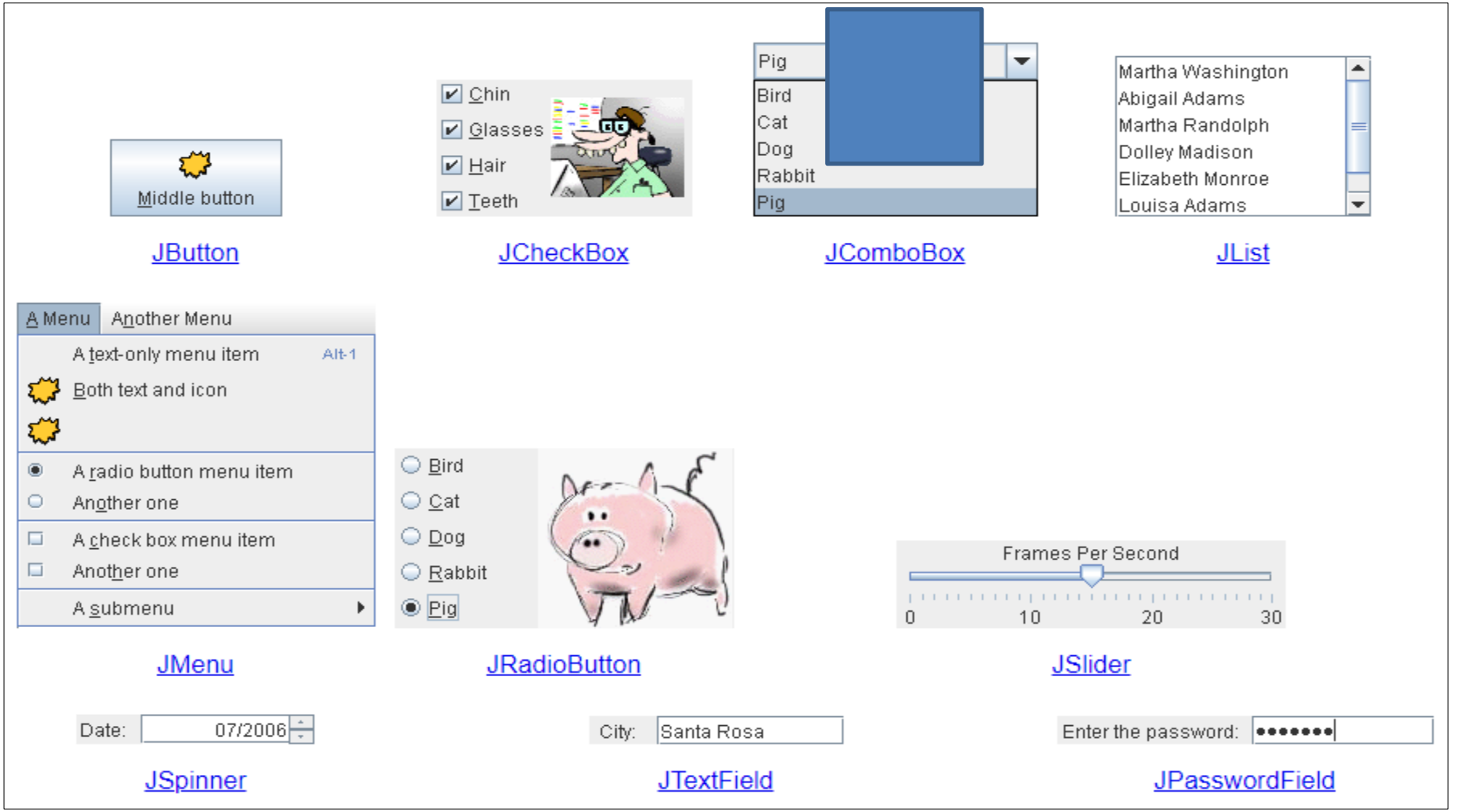
1. **JLabel**: The class is used for placing text in a box. Only Single line text is allowed and the text can not be changed directly.
2. **Jbutton**: The class is used to create a labeled button. GUI component that triggers a certain programmed action upon clicking it.
3. **JTextField**: The class is used for taking input of single line of text.
4. **JTextArea**: The class creates a multi-line text box for users to enter or edit texts.
5. **JPasswordField** : It is used to create a text box specifically used for password and it can be edited..

SWING Components

6. **JCheckbox**: This class is used to create a checkbox. It has two state options; true and false.
7. **JRadioButtonRadio**: This class is used to display a group of related button in which only one can be selected. JRadioButton class is used to create a radio button in Frames.
8. **JComboBox**: This class is a combination of text fields and drop-down list. JComboBox component is used to create a combo box
9. **JList**: This class is used to represent a list of items together. One or more than one item can be selected from the list.
10. **Jslider**: This class is used to create the slider. By using JSlider, a user can select a value from a specific range.
11. **JScrollBar**: This class is used to create a horizontal and a vertical scrollbar.
12. **Jtable**: It used to draw a table to display data.
13. **JTree**: This class is used to display the tree structured data or hierarchical data

Visual Guide For Basic Controls

Simple components that are used primarily to get input from the user; they may also show simple state.



SWING Containers

Swing Container object can **contain other components**. Swing containers can be classified into three main categories:

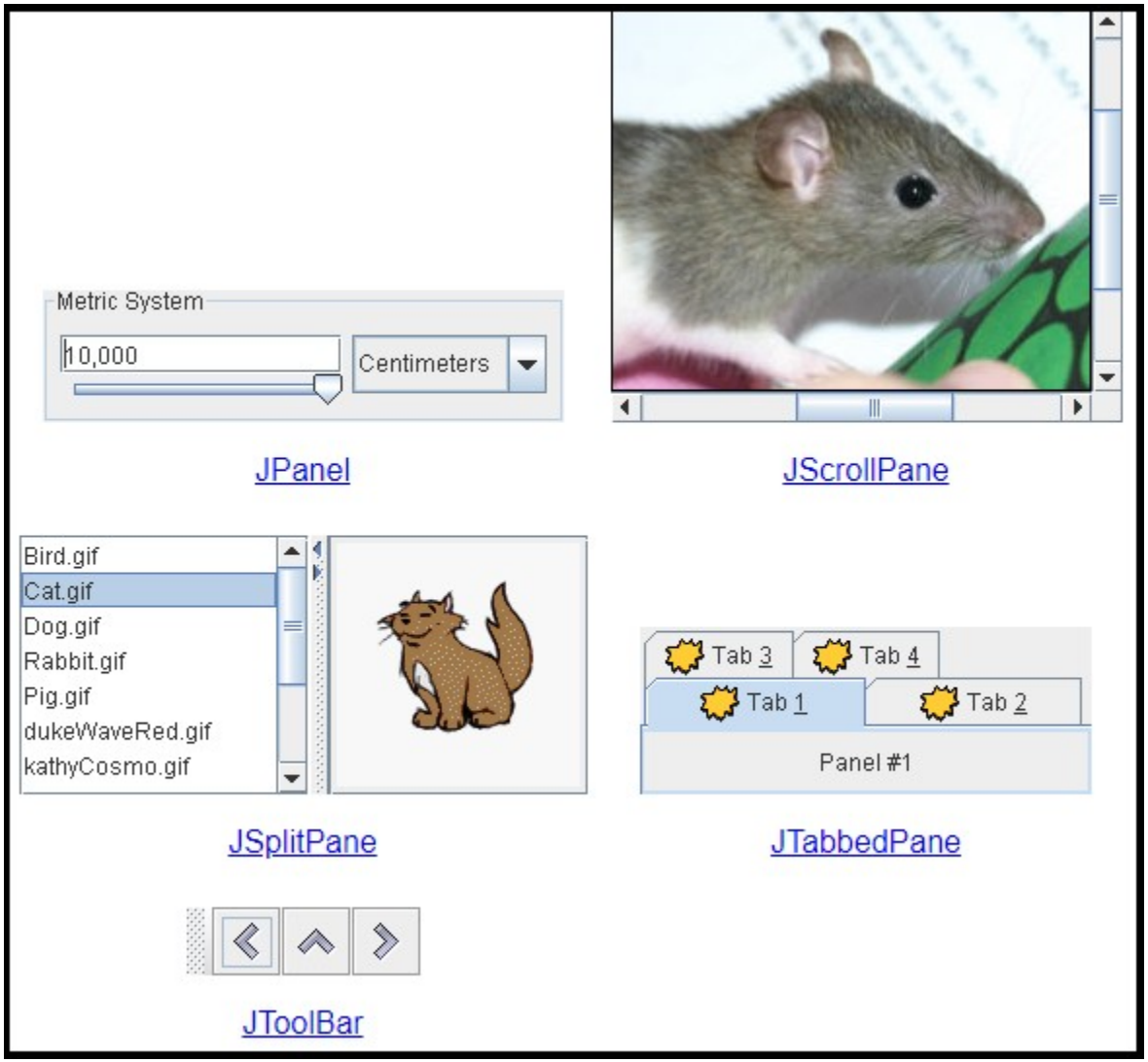
1. **Top-level containers:**
   A top-level container is a swing component with one only purpose, and that is **to hold other swing components (with lower status).**
   JFrame: A top-level window with a title and a border.
   Jwindow: As a rule, not very useful. Provides a window with no controls or title.
   JDialog: This class is the main class for creating a dialog window.
2. **General-purpose containers:**
   An intermediate container is a component with the purpose of making the placement of the **atomic components** more easy.
   JPanel: The JPanel class provides general-purpose containers for lightweight components.

SWING Containers

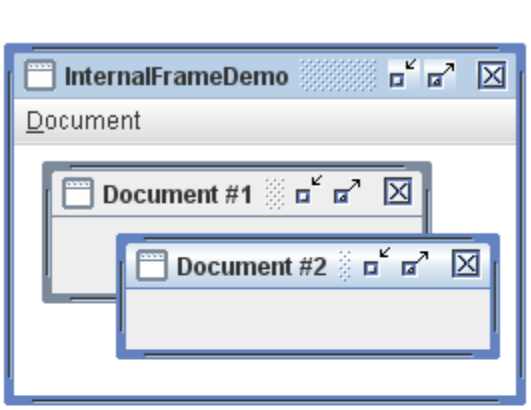
JScrollPane: This class provides a scrollable view of a component
JToolBar: This class is used to create a container that groups several components — usually buttons with icons — into a row or column.
JSplitPane: This class is used to displays two components, either side by side or one on top of the other.
JTabbedPane: This class is used to displays component that lets the user switch between a group of components by clicking on a tab with a given title and/or icon.

3. Special-purpose containers**: play specific roles in the UI**.
JInternalFrame: is a container that provides many features of a frame which includes **displaying title, opening, closing, resizing, support for menu bar, etc.**
JLayeredPane: to show component in different **overlapping layers.** Provides a third dimension for positioning components: depth, also known as Z order
JRootPane: This container used behind the scenes by JFrame, JDialog, JWindow, JApplet, and JInternalFrame.

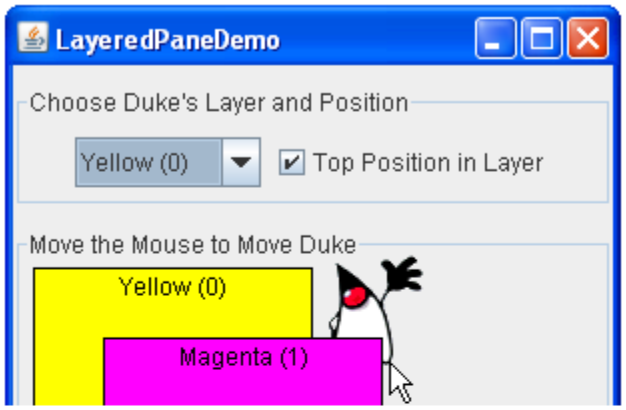
General Purpose Containers



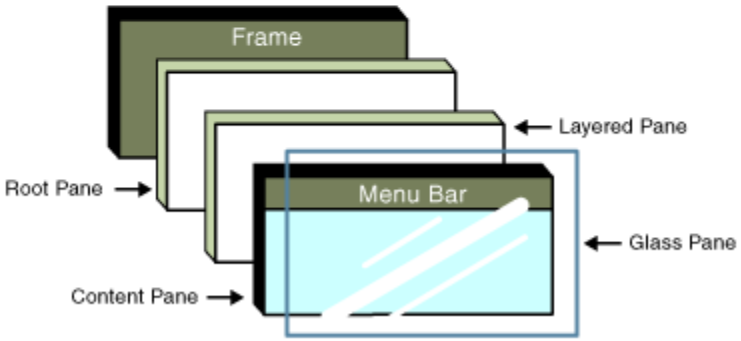
Special Purpose Containers



JInternalFrame



JLayeredPane



Root pane

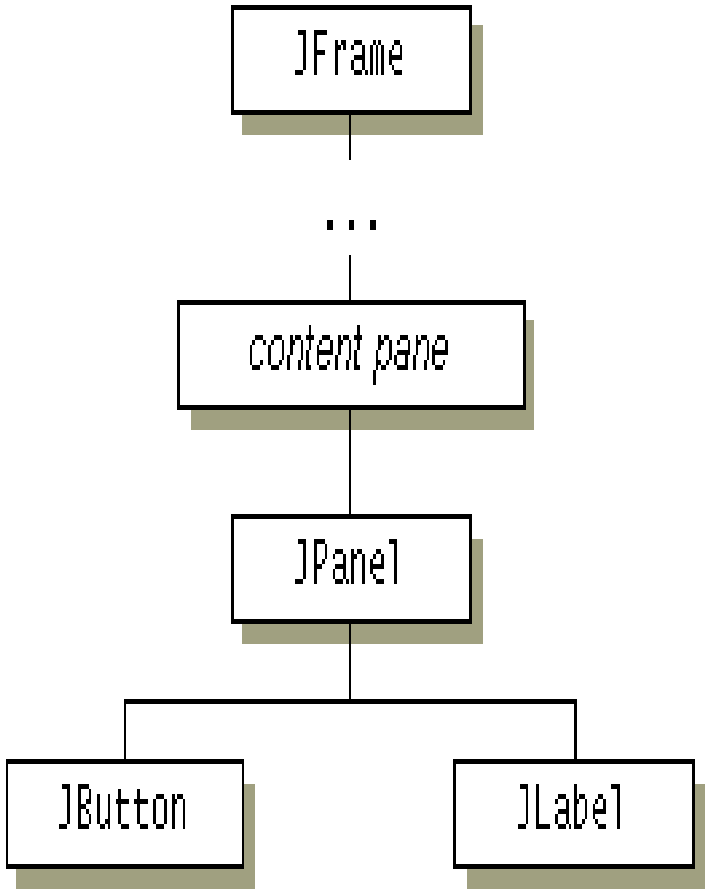
AWT versus Swing

AWT	Swing
The components of are heavy weighted.	The components are light weighted.
less functionality as compared to Swing.	more functionality
The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
The components of are platform dependent.	The components are platform independent.
OS determined look and feel	Customizable and pluggable look and feel
MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
AWT provides comparatively less powerful components.	Swing provides more powerful components.

Working with Swing Containers

The Swing Containment Hierarchy

- **The Swing Containment Hierarchy: -** typically has at least:
  - **Top-Level Container:** the root container that holds everything together. e.g. **JApplet**, **JFrame**, **Window**
  - **Intermediate Container:** a container to simplify the positioning of atomic components. e.g. **JPanel**, **JScrollPane**
  - **Atomic Component:** a self-sufficient component not holding other components. eg. **JButton**, **JLabel**



Java AWT Component Class

The class Component is the abstract base class for the non menu user-interface controls of AWT. It is a super class of AWT container class

Constructor:

protected Component(): creates a component

Commonly used methods of the component class

public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.
void setEnabled(boolean b)	Enables or disables this component,
String getName()	Gets the name of the component

These methods are available for all classes derived from Component

Color getBackground()	Gets the background color of this component.
Rectangle getBounds()	Gets the bounds of this component in the form of a Rectangle object.
Color getForeground()	Gets the foreground color of this component.
Font getFont()	Gets the font of this component..
Point getLocation()	Gets the location of this component in the form of a point specifying the **component's top-left corner.**
Dimension getMinimumSize()	Gets the mininimum size of this component.
void setBackground(Color c)	Sets the background color of this component.
void setBounds(int x,int y,int width,int height)	Moves and resizes this component.
void setForeground(Color c)	Sets the foreground color of this component.

Java AWT Container Class

Container object can contain other components. The class Container is the super class for the containers of AWT.

Constructor

protected Container() : This creates a new Container.

Methods

public Component add (Component component, int position): adds component to the container at position. If position is -1, add() inserts component as the last object within the container

public Component add (Component component):adds component to the container as the last object within the container

public void remove (int index):deletes the component at position index from the container

public void remove (Component component) : The remove() method deletes component from the container

public void removeAll (): method removes all components from the container.

Java AWT Container Class

public LayoutManager getLayout () : method returns the container's current LayoutManager

public void setLayout (LayoutManager manager) : changes the container's LayoutManager to manager

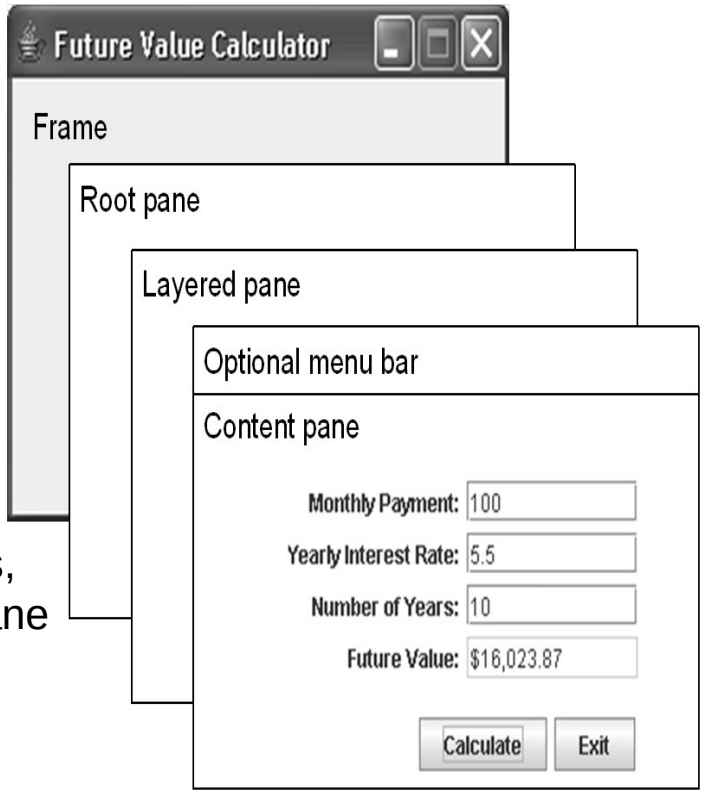
public Component getComponentAt (int x, int y) or public Component getComponentAt (Point p): returns the container or component that contains the coordinate or point

All these and other methods are available in all classes derived from Container

A **frame** is a **base window (with a title, minimize, restore and close buttons and a border)** on which other components rely, such as menu bar, panels, labels, text fields, buttons, etc.

Almost every Swing application starts with a window. It represents the main application window.

When one of the top-level Swing containers, are created such as JApplet, JDialog,and JFrame, the root pane is created which contains four parts



1. Layered Pane
   Serves to position its contents, **which consist of the content pane and the optional menu bar**
2. Content Pane
   The container of the root pane's visible components, excluding the menu bar. We draw on the content pane
3. The optional menu bar
   The home for the root pane's container's menus
4. The glass pane
   The **glass pane** is useful when you want to be able to intercept events or paint over an area that already contains one or more components.

JFrame

JFrame is **a heavyweight container used as the top-level window.** It is the direct descendent of the window class which is also descends directly from container

Constructors

JFrame()

Constructs a new frame that is initially invisible.

JFrame(String title)

Creates a new, initially invisible Frame with the specified title.

The default layout for Jframe is BorderLayout

Methods

void setIconImage(Image image): It sets the image to be displayed as the icon for this window.

setContentPane(Container contentPane): It sets the contentPane property

JFrame…

void setJMenuBar(JMenuBar menubar): It sets the menubar for this frame.
void Set LayeredPane(JLayeredPane layeredPane): It sets the layeredPane property.

JRootPane: getRootPane(): It returns the rootPane object for this frame.

Container getContentPane(): It gets the contentPane object for this frame so that you can add an **object** to it.

void setDefaultCloseOperation(int op): Makes the frame perform the given action when it closes.Common value passed: JFrame.EXIT_ON_CLOSE. If not set, the program will never exit even if the frame is closed.

Creating Jframe Window

There are three ways to create a JFrame window

1. By extending the JFrame

```
public class MyFirstWindow extends JFrame{
public MyFirstWindow() {
                super("My first JFrame window");
                setSize(300,300);//frame size 300 width and 300 height
                setLayout(null);//no layout manager
                setVisible(true);//frame will be visible, by default not visible
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE):
    }
    public static void main(String args[]){
                MyFirstWindow w=new MyFirstWindow();

    }
}
```

Creating JFrame …

2. By using Composition

```
public class MyFirstWindow {
JFrame appWindow;
public MyFirstWindow() {
        appWindow=new JFrame("My first Jframe window");
        appWindow.setSize(300,300);//300 width and 300 height
        appWindow.setLayout(null);//no layout manager
        appWindow.setVisible(true);//now frame will be visible
        appWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]){
            MyFirstWindow w=new MyFirstWindow();
    }
}
```

Creating JFrame …

3. By creating the object of Frame inside the main class

```
public class MyFirstWindow {
public static void main(String args[]){
        Jframe appWindow=new JFrame();// no argument constructor
        appWidow.setTitle((“My first Jframe window”);
        appWindow.setSize(300,300);//300 width and 300 height
        appWindow.setLayout(null);//no layout manager
        appWindow.setVisible(true);//now frame will be visible
        appWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
        ;

    }
}
```

JPanel

JPanel is a lightweight container generally used to organize Graphic user interface components. JPanels are added on top of JFrame, whereas graphical user interface components are added on one or more JPanels.

This class inherits methods from the following classes –

- javax.swing.JComponent
- java.awt.Container
- java.awt.Component
- java.lang.Object

Commonly used constructors

- **JPanel()** : Creates a new JPanel with a double buffer and **a flow layout.**
- **JPanel(LayoutManager layout):** Creates a new buffered JPanel with the specified layout manager.

Creating JPanel

```
import java.awt.Color;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class PanelDemo {
   JFame frame;
   PanelDemo() {
      JFrame frame = new JFrame("JPanel Example");
      frame.setSize(300,300);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setBounds(40, 80, 200, 200);
      panel.setBackground(Color.gray);
      frame.add(panel);
      frame.pack();
      frame.setVisible(true);
  }
   public static void main(String [] args){
       PanelDemo demo= new PanelDemo(); }
}
```

Working with Swing Atomic Components

Atomic Components

Atomic components are those swing components that exist solely to present and perhaps **accept information**, rather than to contain other components. Components that **directly interact with the user**

All the atomic components **descend from the JComponent** class. Thus, they all support standard features such **as tool tips and borders**

All them **are lightweight components**

JLabel

JLabel is used to display a short **string or an image icon**. JLabel can display text, image or both . JLabel **cannot get focus** . JLabel is **inactive to input** events such a mouse focus or keyboard focus. By default labels are vertically centered but the user can change the alignment of label.

Constructors

1. JLabel() : creates a blank label with no text or image in it.
2. JLabel(String s) : creates a new label with the string specified.
3. JLabel(Icon i) : creates a new label with a image on it.
4. JLabel(String s, Icon i, int halign) : creates a new label with a string, an image and a specified horizontal alignment

Where haling: LEFT, CENTER, RIGHT, LEADING or TRAILING.

JLabel methods

Methods	Description
String getText()	t 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.
public void **setIcon**(Icon icon).	It sets a label icon
int getHorizontalAlignment()	It returns the alignment of the label's contents along the X axis.

```
import java.awt.Dimension;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.Border;
import javax.swing.JLabel;
public class AddressBook extends JFrame{
JFrame frm;
JLabel nameLbl;
JLabel addressLbl;
JPanel pnl;
public AddressBook() {
Frm=new JFrame("Address Book");
frm.setSize(300,300);
frm.setLayout(null);
frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
pnl=new JPanel();
pnl.setLocation(10, 10);
```

```
pnl.setSize(262, 240);
pnl.setLayout(null);
Border border= BorderFactory.createLineBorder(Color.red);
pnl.setBorder(border);
nameLbl = new JLabel("Full Name :", JLabel.LEFT);
addressLbl = new JLabel();
addressLbl.setText("Address :");
nameLbl.setBounds(10, 10, 116, 15);
addressLbl.setBounds(10, 35, 116, 15);
pnl.add(nameLbl);
pnl.add(addressLbl);
frm.setVisible(true);
frm.getContentPane().add(pnl);//or add(pnl)
}
public static void main(String [] args){
AddressBook w=new AddressBook();
}
}
```

Button

The JButton which is implementation of push button is used to create labelled button. On pressing it generate an event.

Constructors

1. Button(): Creates a button with no set text or icon.
2. JButton(Icon icon): Creates a button with an icon.
3. JButton(String text): Creates a button with text.
4. JButton(String text, Icon icon): Creates a button with initial text and an icon.

Methods

void setText(String s)	set specified text on button
String getText()	return the text of the button.
void setEnabled(boolean b)	enable or disable the button.
void setIcon(Icon b)	set the specified Icon on the button.
Icon getIcon()	get the Icon of the button.

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

Constructors

1. JTextField() : constructor that creates a new TextField
2. JTextField(int columns) : creates empty TextField with specified number of columns.
3. JTextField(String text) : creates a new empty text field initialized with the given string.
4. JTextField(String text, int columns) : creates a new empty textField with the given string and a specified number of columns .

Methods

void setText(String s)	set specified text on button
String getText()	return the text of the button.
Void setColumns(int n)	set the number of columns within a text field.
Void setFont(Font f)	set the font of the text field displayed text
Int getColumns():	get the column numbers in the text field.

JTextArea

A JTextArea is a multi-line area that displays plain text.

Constructors

1. JTextArea(): Creates a text area that displays no text initially.
2. JTextArea(String s): Creates a text area that displays specified text initially.
3. JTextArea(int row, int column): Creates a text area with the specified number of rows and columns that displays no text initially.
4. JTextArea(String s, int row, int column): Creates a text area with the specified number of rows and columns that displays specified text.Methods

JTextField

Methods

void setRows(int rows)	It is used to set specified number of rows.
void setColumns(int cols)	It is used to set specified number of columns.
void setText(String s)	set specified text on button
String getText()	return the text of the button.
void setFont(Font f)	It is used to set the specified font.
void insert(String s, int position)	It is used to insert the specified text on the specified position.
void append(String s)	It is used to append the given text to the end of the document.

JPasswordField

The class JPasswordField is a component which is specialized to handle password functionality and allows the editing of a single line of text.

Constructors

1. JPasswordField(): constructor that creates a new PasswordField
2. JPasswordField(int columns) : constructor that creates a new empty PasswordField with specified number of columns.
3. JPasswordField(String Password) : constructor that creates a new empty Password field initialized with the given string.
4. JPasswordField(String Password, int columns) : constructor that creates a new empty PasswordField with the given string and a specified number of columns .

JPasswordField

Methods

boolean echoCharIsSet()	Returns true if this JPasswordField has a character set for echoing.
char getEchoChar()	Returns the character to be used for echoing.
char[] getPassword()	Returns the text contained in this TextComponent
protected String paramString()	Returns a string representation of this JPasswordField.
void setEchoChar(char c)	Sets the echo character for this JPasswordField
void insert(String s, int position)	It is used to insert the specified text on the specified position.

JComboxBox

JComboxBox is a Swing component that renders a drop-down list of choices and lets the user selects one item from the list.

The combo box can be read only or editable. An editable combo box allows the user to type a different value other than the fixed ones.

Constructors

1. JComboBox() : creates a new empty JComboBox .
2. JComboBox(E [ ] items) : creates a new JComboBox with items from specified array.
3. JComboBox(Vector items) : creates a new JComboBox with items from the specified vector

JComboxBox

Methods

addItem(E item)	adds the item to the JComboBox.
getItemAt(int i)	returns the item at index i
getItemCount():	returns the number of items from the list
getSelectedItem() :	returns the item which is selected
removeItemAt(int i) :	removes the element at index i
setEditable(boolean b)	Set editable or read only
setSelectedIndex(int i):	selects the element of JComboBox at index i

Exercise

1. **Write a GUI based program that calculate the parking fare for customer who park their cars in a parking lot when the following information is given.**
   - A character showing the type of the vehicle: ‘Car’ for car, ‘Bus’ for bus, ‘Truck’ for truck.
   - The time the vehicle entered the lot, entered as hour:minute.
   - The time the vehicle left the lot, entered as hour:minute.
   - To encourage people to park for a short period of time, the management uses two different rates for each type of vehicle, as shown in the following table.

Type	**First rate**	**Second rate**
Car	br5.00/hr, first 3 hr	br7.50/hr, after the first 3 hr
Truck	br8.00/hr, first 2 hr	br11.30/hr, after the first 2 hr
Bus	br13.00/hr, for first hr	br18.70/hr, after the first hr

Solution

```
import javax.swing.*;
public class ParkingProgram {
   public static void main(String[] args){
    String [] vehicleType= {"Car", "Truck", "Bus"};
    JFrame parkApp=new JFrame("Parking Fare");
         ImageIcon imgicon = new ImageIcon("dahlia.png");
         parkApp.setSize(400, 400);
        parkApp.setLayout(null);
    parkApp.setIconImage(imgicon.getImage());
         JPanel panel=new JPanel();
         panel.setBounds(50, 10, 290,350);
         panel.setLayout(null);
         //panel.add(heading);
         JLabel vehicleLbl= new JLabel("Vehicle Type:");
         vehicleLbl.setBounds(10, 10, 100, 30);
         JComboBox vehicleCbx=new JComboBox(vehicleType);
         vehicleCbx.setBounds(165, 10, 120, 30);
         panel.add(vehicleLbl);
```

Solution

```
panel.add(vehicleCbx);
    JLabel timeInLbl= new JLabel("Time In(h:m):");
    timeInLbl.setBounds(10, 60, 100, 30);
    JTextField timeInTxt= new JTextField(120);
    timeInTxt.setBounds(165, 60, 120, 30);
    panel.add(timeInLbl);
    panel.add(timeInTxt);
    JLabel timeOutLbl= new JLabel("Time Out(h:m):");
    timeOutLbl.setBounds(10, 110, 100, 30);
    JTextField timeOutTxt= new JTextField(120);
    timeOutTxt.setBounds(165, 110, 120, 30);
    panel.add(timeOutLbl);
    panel.add(timeOutTxt);
    JLabel timeParkLbl= new JLabel("Parking Time:");
    timeParkLbl.setBounds(10, 160, 100, 30);
    JTextField timeParkTxt= new JTextField(120);
    timeParkTxt.setEditable(false);
```

Solution

```
timeParkTxt.setBounds(165, 160, 120, 30);
   panel.add(timeParkLbl);
   panel.add(timeParkTxt);
   JLabel hourParkLbl= new JLabel("Parking Hour:");
   hourParkLbl.setBounds(10, 210, 100, 30);
   JTextField hourParkTxt= new JTextField(120);
   hourParkTxt.setEditable(false);
   hourParkTxt.setBounds(165, 210, 120, 30);
   panel.add(hourParkLbl);
   panel.add(hourParkTxt);
   JLabel fareParkLbl= new JLabel("Parking Fare:");
   fareParkLbl.setBounds(10, 260, 100, 30);
   JTextField fareParkTxt= new JTextField(120);
   fareParkTxt.setEditable(false);
   fareParkTxt.setBounds(165, 260, 120, 30);
   panel.add(fareParkLbl);
   panel.add(fareParkTxt);
```

Solution

```
    JButton calcBtn=new JButton("Calculte Fare");
        calcBtn.setBounds(10, 310, 120, 30);
        panel.add(calcBtn);
        JButton clrBtn=new JButton("Clear");
        clrBtn.setBounds(150, 310, 70, 30);
        panel.add(clrBtn);
        parkApp.add(panel);
    parkApp.setVisible(true);
  }
}
```

JCheckBox

The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.

Constructors

1. JCheckBox(): Creates an initially unselected check box button with no text and no icon.
2. JCheckBox(Icon icon): Creates an initially unselected checkbox with an icon.
3. JCheckBox(String text): Creates an initially unselected checkbox with text.
4. JCheckBox(String text, Icon icon): Creates an initially unselected checkbox with the specified text and icon.

Methods

void setSelected(boolean)	To select, or un-select, a box
boolean isSelected()	To check if a box is selected

JRadioButton

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

Constructors

JRadioButton(): Creates an initially unselected radio button with no set text.

JRadioButton(Icon icon): Creates an initially unselected radio button with the specified image but no text

JRadioButton(Icon icon, boolean selected): Creates a radio button with the specified image and selection state, but no text.

JRadioButton(String text, boolean selected):Creates a radio button with the specified text and selection state.

JRadioButton(String text, Icon icon):Creates a radio button that has the specified text and image, and which is initially unselected.

JRadioButton(String text, Icon icon, boolean selected):Creates a radio button that has the specified text, image, and selection state.

JRadioButton

Methods

void setSelected(boolean)	To select, or un-select, a box
boolean isSelected()	To check if a box is selected

Example

```
String names[] = {"LEADING", "CENTER", "TRAILING"};
leadingButton = new JRadioButton(names[0]);
leadingButton.setSelected(true);
centerButton = new JRadioButton(names[1]);
TrailingButton = new JRadioButton(names[2]);
 // Create the button group to keep only one selected.
 ButtonGroup btnGroup = new ButtonGroup();
 btnGroup.add(leadingButton);
 btnGroup.add(centerButton);
 btnGroup.add(trailingButton);
```

JList

The class JCheckBox is an implementation of a check box - an item that can be selected or deselected, and which displays its state to the user.

Constructors

1. JList () : This constructor construct a new JList with an empty, read-only model.
2. JList (E[] listData): This constructor is used to construct a JList in which displays the elements in the specified array.

Methods

void clearSelection():	used to clear the selection.
boolean isSelected()	to get the index of the selected item in the JList
Int getSelectedIndex()	Returns the smallest selected cell index; *the selection* when only a single item is selected in the list.
Int [] getSelectedIndces()	Returns an array of all of the selected indices, in increasing order.

JList

E getSelectedValue()	Returns the value for the smallest selected cell index; *the selected value* when only a single item is selected in the list.
Void setListData(E [])	Sets the list data with given array
setSelectedIndex(int index)	Selects a single cell.
setSelectedIndices(int[] indices)	Changes the selection to be the set of indices specified by the given array.
void setSelectionMode(int selectionMode)	ListSelectionModel.SINGLE_SELECTION - Only one list index can be selected at a time ListSelectionModel.MULTIPLE_INTERVAL_SELECTION - In this mode, there's no restriction on what can be selected. This mode is the default.

To add scrollbar

```
JScrollPane scrollPane = new JScrollPane();
JList list = new JList();
scrollPane.setViewportView(list);
```

Working with Layout

Layout Manager

A layout manager controls how GUI components are organized within a GUI container. Each Swing container (e.g. JFrame, JDialog, JApplet and JPanel) is a subclass of java.awt.Container and so has a layout manager that controls it.

A LayoutManager is an interface (java.awt.LayoutManager)

There are 8 useful layout classes that implement LayoutManager:

1. FlowLayout
2. BorderLayout
3. CardLayout
4. GridLayout
5. GridBagLayout
6. BoxLayout
7. ***OverlayLayout***
8. ***SpringLayout***

Layout Manager

What Layout managers control:

- Where components appear
- What sizes they are
- How they react when they are resized

Why Layout Manager?

1. We would not have to compute locations and sizes for our components
2. Our components will resize automatically when the window is resized
3. Our interface will appear "nicely" on all platforms

Null Layout

We can actually set the layout for our window panel to be null so that we have full control on the exact locations of all objects. This means that we are NOT using a layout manager. So, we must arrange the components on our own.

Advantage

- You can specify exactly where you want to place the components.
- Much simpler than some of the layout managers such as GridBagLayout

Disadvantage

- You do not get to specify any resizing behaviour, so your window does not resize properly !!
- Usually, your component locations rely on other component locations, so moving a single component may cause you to want to move other components around again.
- You MUST pre-plan your entire window (this is actually be a good thing to do anyway) by specifying the exact locations of components.

Null Layout

To choose to use no layout manager, we just send the following message to a container:

setLayout(null);

We must then specifically place all of our components by using the following:. Locations are always with respect to the top left corner of the container which is location (0,0).

//set the location of the component within its container

void setLocation(int x, int y);

comp. setLocation(10, 20); //measurements are in pixels

//set the dimensions of the component

void setSize(int width, int height);

comp. setSize(70, 20); //measurements are in pixels

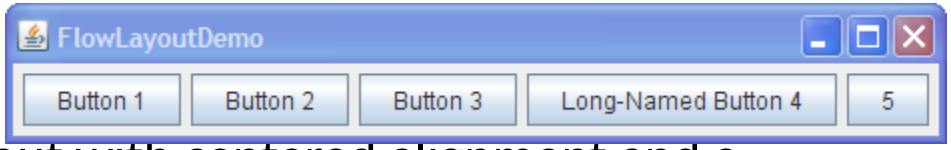
Or use the setBounds(int x, int y, int width, int height)

comp. setBounds(10,20,70, 20); //measurements are in pixels

FlowLayout

The simplest layout manager is java.awt.FlowLayout,

- adds components to the container from left-to-right, top-to-bottom.
- if no space left on current "line", components flow to next line
- components are centered horizontally on each line by default
- It is the default layout for GUI container objects



Constructors

1. FlowLayout(): creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. FlowLayout(int align): creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. FlowLayout(int align, int hgap, int vgap): creates a flow layout with the given alignment and the given horizontal and vertical gap.

- align may be any one of five FlowLayout class constants: LEFT, RIGHT, CENTER, LEADING, TRAILING

Eg FlowLayout.CENTER

FlowLayout

Method

int getAlignment(): Returns the alignment for this layout.

Void setAlignment(int align): used to set the alignment for this layout.

Example

```
import java.awt.*; import javax.swing.*;
public class FlowLayoutExample extends JFrame {
    FlowLayoutExample (String title) {
        super(title);
        setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
        add(new JButton("one"));
        add(new JButton("two"));
        add(new JButton("three"));
        add(new JButton("four", new ImageIcon(“flower.gif”));
        add(new JButton("new ImageIcon(“flower.gif”));
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setSize(500, 200);
    }
    public static void main(String args[]) {
        FlowLayoutExample frame=new FlowLayoutExample(“Flow Layout
    Example");
            frame.setVisible(true);
    }
}
```

BorderLayout

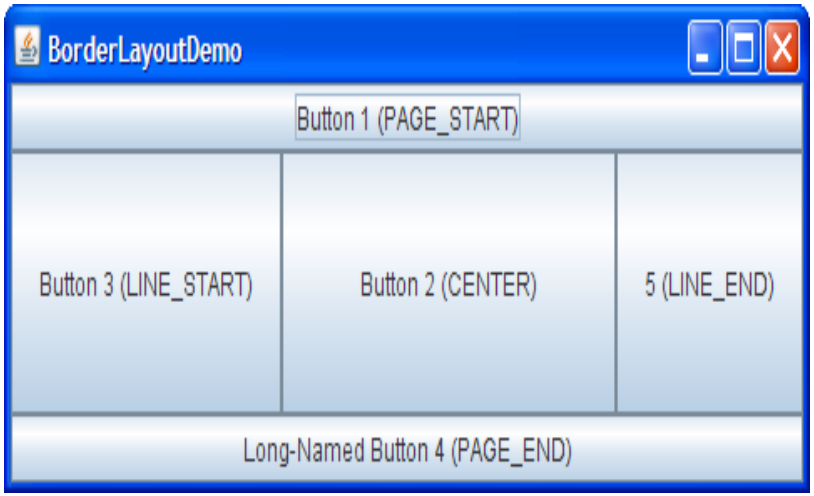
BorderLayout is the default layout for the window objects such as JFrame, JWindow, JDialog, JInternalFrame etc.

BorderLayout arranges the components in the five regions. Four sides are referred to as north, south, east, and west. The middle part is called the center.

Each region can contain only one component and is identified by a corresponding BorderLayout constant as NORTH, SOUTH, EAST, WEST, and CENTER.

Constructors:

1. BorderLayout(): It will construct a new borderlayout with no gaps between the components.
2. BorderLayout(int, int): It will constructs a border layout with the specified gaps between the components.



Note that you do NOT have to put something in all of the border

BorderLayout

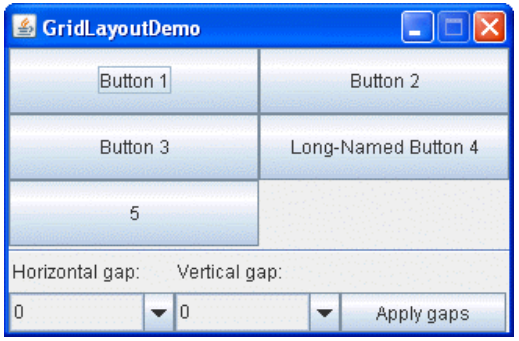
```
import java.awt.*; import javax.swing.*;
Public class BorderLayouExample extends JFrame {
      public BorderLayouExample (String title) {
          super(title);
          setLayout(new BorderLayout());
          JPanel buttonPanel = new JPanel();
          buttonPanel.add(new JButton("Add"));
          buttonPanel.add(new JButton("Remove"));
          buttonPanel.add(new JButton("Insert"));
          buttonPanel.add(new JButton("Edit"));
          buttonPanel.add(new JButton("Details"));
          add(BorderLayout.SOUTH, buttonPanel);
          add(BorderLayout.CENTER, new JTextArea());
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setSize(500, 200);
      }
      public static void main(String args[]) {
            BorderLayouExample frame=new BorderLayouExample(“Border
      Layout Example");
           frame.setVisible(true);
      } }
```

GridLayout

The GridLayout class is used to design layouts that have a specified number of rows and columns in a rectangular grid. Each component is placed in the equal-sized rectangle. The changes are visible for each small-rectangle whenever its size is adjusted..

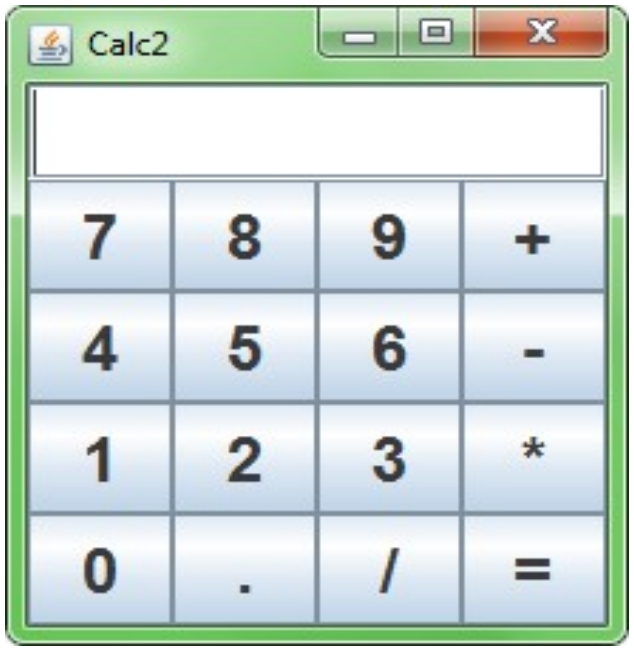
Constructors:

1. GridLayout(): It Creates a grid layout with a default of one column per component, in a single row.
2. GridLayout(int rw, int cl): It creates a grid layout with the specified number of rows and columns.
3. GridLayout(int rw, int cl, int hgap, int vgap): It creates a grid layout with the specified number of rows and columns with horizontal and vertical gap.



Example

Create the following calculator window



GridLayout

```
import java.awt.BorderLayout;
import java.awt.Font;
import java.awt.GridLayout;
import javax.swing.*;
public class Calc2 {
    public static final String[][] BUTTON_TEXTS = {{"7", "8", "9", "+"}, {"4",
"5", "6", "-"},
     {"1", "2", "3", "*"}, {"0", ".", "/", "=" };
   public static final Font BTN_FONT = new Font(Font.SANS_SERIF, Font.BOLD,
24);
   private static void createAndShowGui() {
      JTextField field = new JTextField(10);
      field.setFont(BTN_FONT.deriveFont(Font.PLAIN));
      JPanel btnPanel = new JPanel(new GridLayout(BUTTON_TEXTS.length,
      BUTTON_TEXTS[0].length));
     for (int i = 0; i < BUTTON_TEXTS.length; i++) {
         for (int j = 0; j < BUTTON_TEXTS[i].length; j++) {
            JButton btn = new JButton(BUTTON_TEXTS[i][j]);
            btn.setFont(BTN_FONT);
            btnPanel.add(btn);
         }
```

```
    JPanel mainPanel = new JPanel(new BorderLayout());
     mainPanel.add(field, BorderLayout.PAGE_START);
     mainPanel.add(btnPanel, BorderLayout.CENTER);
     JFrame frame = new JFrame("Calc2");
     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     frame.getContentPane().add(mainPanel);
     frame.pack();
     frame.setLocationRelativeTo(null);
     frame.setVisible(true);
   }
   public static void main(String[] args) {
     SwingUtilities.invokeLater(new Runnable() {
        public void run() {
          createAndShowGui();
        }
     });
   }
}
```

The Parking Problem With GridLayout

```
package parking;
import java.awt.*;
import javax.swing.*;
public class ParkingProgram {
private final String [] vehicleType= {"Car", "Truck",
"Bus"};
private JFrame parkApp;
private JLabel vehicleLbl;
private JComboBox vehicleCbx;
JPanel panel;
JLabel timeInLbl;
JTextField timeInTxt;
JLabel timeOutLbl;
JTextField timeOutTxt;
JLabel timeParkLbl;
JTextField timeParkTxt;
JLabel hourParkLbl;
JTextField hourParkTxt;
```

The Parking Problem With GridLayout

```
JLabel fareParkLbl;
JTextField fareParkTxt;
JButton clrBtn;
JButton calcBtn;
public ParkingProgram() {
    parkApp=new JFrame("Parking Fare");
        ImageIcon imgicon = new ImageIcon("dahlia.png");
         parkApp.setLayout(null);
        parkApp.setSize(400, 400);
    parkApp.setIconImage(imgicon.getImage());
         JPanel panel=new JPanel();
    panel.setBounds(50, 10, 300,350);
         panel.setLayout(new GridLayout(0,2,10,10));
    vehicleLbl= new JLabel("Vehicle Type:");
         vehicleLbl.setPreferredSize(new
Dimension(140,10));
         vehicleCbx=new JComboBox(vehicleType);
         vehicleCbx.setPreferredSize(new
Dimension(140,10));
```

The Parking Problem With GridLayout

```
    vehicleCbx.addItemListener(this);
        panel.add(vehicleLbl);
        panel.add(vehicleCbx);
        timeInLbl= new JLabel("Time In(h:m):");
        timeInLbl.setPreferredSize(new
Dimension(140,10));
        timeInTxt= new JTextField(110);
        timeInTxt.setPreferredSize(new
Dimension(140,10));
timeInTxt.setBorder(BorderFactory.createCompoundBorder(
timeInTxt.getBorder(),BorderFactory.createEmptyBorder(5
, 5, 5, 5)));
  panel.add(timeInLbl);
        panel.add(timeInTxt);
        timeOutLbl= new JLabel("Time Out(h:m):");
        timeOutLbl.setPreferredSize(new
Dimension(140,10));
       timeOutTxt= new JTextField(110);
        timeOutTxt.setPreferredSize(new
```

The Parking Problem With GridLayout

```
timeOutTxt.setBorder(BorderFactory.createCompoundBorder
(timeOutTxt.getBorder(),BorderFactory.createEmptyBorder
(5, 5, 5, 5)));
        panel.add(timeOutLbl);
        panel.add(timeOutTxt);
        timeParkLbl= new JLabel("Parking Time:");
        timeParkLbl.setPreferredSize(new
Dimension(140,10));
        timeParkTxt= new JTextField(110);
        timeParkTxt.setEditable(false);
        timeParkTxt.setPreferredSize(new
Dimension(140,10));
timeParkTxt.setBorder(BorderFactory.createCompoundBorde
r(timeParkTxt.getBorder(),BorderFactory.createEmptyBord
er(5, 5, 5, 5)));
    panel.add(timeParkLbl);
         panel.add(timeParkTxt);
         hourParkLbl= new JLabel("Parking Hour:");
```

The Parking Problem With GridLayout

```
    hourParkLbl.setPreferredSize(new Dimension(140,10));
        hourParkTxt= new JTextField(110);
        hourParkTxt.setEditable(false);
        hourParkTxt.setPreferredSize(new
Dimension(140,10));
hourParkTxt.setBorder(BorderFactory.createCompoundBorder
(hourParkTxt.getBorder(),BorderFactory.createEmptyBorder
(5, 5, 5, 5)));
        panel.add(hourParkLbl);
        panel.add(hourParkTxt);
        fareParkLbl= new JLabel("Parking Fare:");
        fareParkLbl.setPreferredSize(new
Dimension(140,10));
    fareParkTxt= new JTextField(110);
        fareParkTxt.setEditable(false);
        fareParkTxt.setPreferredSize(new
Dimension(140,10));

fareParkTxt.setBorder(BorderFactory.createCompoundBorder
```

The Parking Problem With GridLayout

```
        panel.add(fareParkLbl);
            panel.add(fareParkTxt);
            calcBtn=new JButton("Calculte Fare");
            calcBtn.setPreferredSize(new Dimension(140,10));
            calcBtn.addActionListener(this);
            panel.add(calcBtn);
            clrBtn=new JButton("Clear");
            clrBtn.setPreferredSize(new Dimension(140,10));
            clrBtn.addActionListener(this);
            panel.add(clrBtn);
            parkApp.add(panel);
        parkApp.setVisible(true);
}
public static void main(String[] args){
    new  ParkingProgram();
}
}
```

GridBagLayout

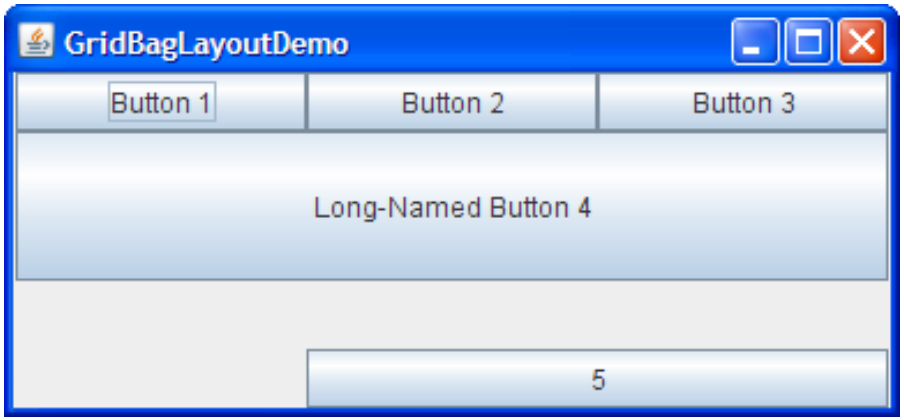
A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.

Not all rows necessarily have the same height.

Similarly, not all columns necessarily have the same width.

Essentially, GridBagLayout places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be.

The way the program specifies the size and position characteristics of its components is by specifying constraints for each component.



Specifying Constraints-GridBagLayout

A GridBagLayout places components in a grid of rows and columns, allowing specified components to span multiple rows or columns.

Not all rows necessarily have the same height.

Similarly, not all columns necessarily have the same width.

Essentially, GridBagLayout places components in rectangles (cells) in a grid, and then uses the components' preferred sizes to determine how big the cells should be.

The way the program specifies the size and position characteristics of its components is by specifying constraints for each component.

- To create a constraint you use the constructor of the GridBagConstraint class GridBagConstraint()

Constraints

- **gridx, gridy**

Specify the row and column at the upper left of the component. The leftmost column has address gridx=0 and the top row has address gridy=0.

Specifying Constraints-GridBagLayout

- **gridwidth, gridheight**

Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area. These constraints specify the number of cells the component uses, not the number of pixels it uses. The default value is 1.

- **weightx, weighty**

Weights are used to determine how to distribute space among columns (weightx) and among rows (weighty); If you set the **weight to 0**, then the area never grows or shrinks beyond its initial size in that direction.

- **ipadx, ipady**

Specifies the internal padding: how much to add to the size of the component. The default value is zero.

- **insets**

Specifies the external padding of the component -- the minimum amount of space between the component and the edges of its display area.

Specifying Constraints-GridBagLayout

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class GridBagLayoutExample extends JFrame{
    public static void main(String[] args) {
            GridBagLayoutExample a = new GridBagLayoutExample();
        }
        public GridBagLayoutExample() {
    GridBagLayoutgrid = new GridBagLayout();
            GridBagConstraints gbc = new GridBagConstraints();
            setLayout(grid);
            setTitle("GridBag Layout Example");
            GridBagLayout layout = new GridBagLayout();
    this.setLayout(layout);
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;
```

Specifying Constraints-GridBagLayout

```
this.add(new Button("Button One"), gbc);
 gbc.gridx = 1;
 gbc.gridy = 0;
 this.add(new Button("Button two"), gbc);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 gbc.ipady = 20;
 gbc.gridx = 0;
 gbc.gridy = 1;
 this.add(new Button("Button Three"), gbc);
 gbc.gridx = 1;
 gbc.gridy = 1;
 this.add(new Button("Button Four"), gbc);
 gbc.gridx = 0;
 gbc.gridy = 2;
 gbc.fill = GridBagConstraints.HORIZONTAL;
 gbc.gridwidth = 2;
```

Specifying Constraints-GridBagLayout

```
    this.add(new Button("Button Five"), gbc);
            setSize(300, 300);
            setPreferredSize(getSize());
            setVisible(true);
            setDefaultCloseOperation(EXIT_ON_CLOSE);

        }

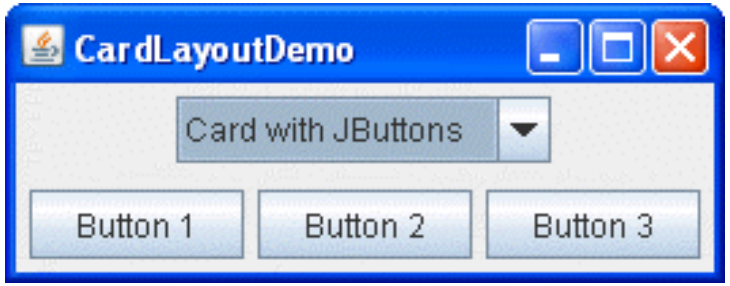
}
```

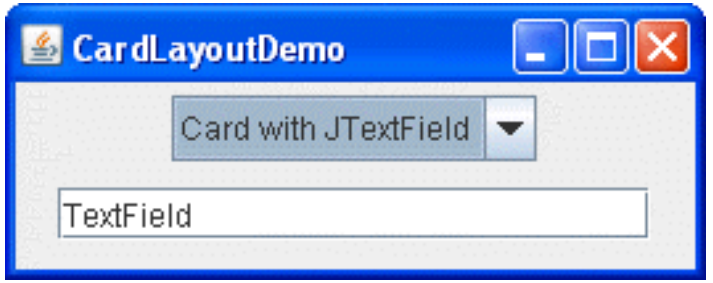
CardLayout

The CardLayout class manages the components in such a way that only one component is visible at a time. It treats each component as a card in the container. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

Constructors:

1. CardLayout(): It is used to create a new card layout with gaps of size is zero.
2. CardLayout(int horizontalgap, int verticalgap): It is used to create a new CardLayout class with the specified horizontal and vertical gaps.



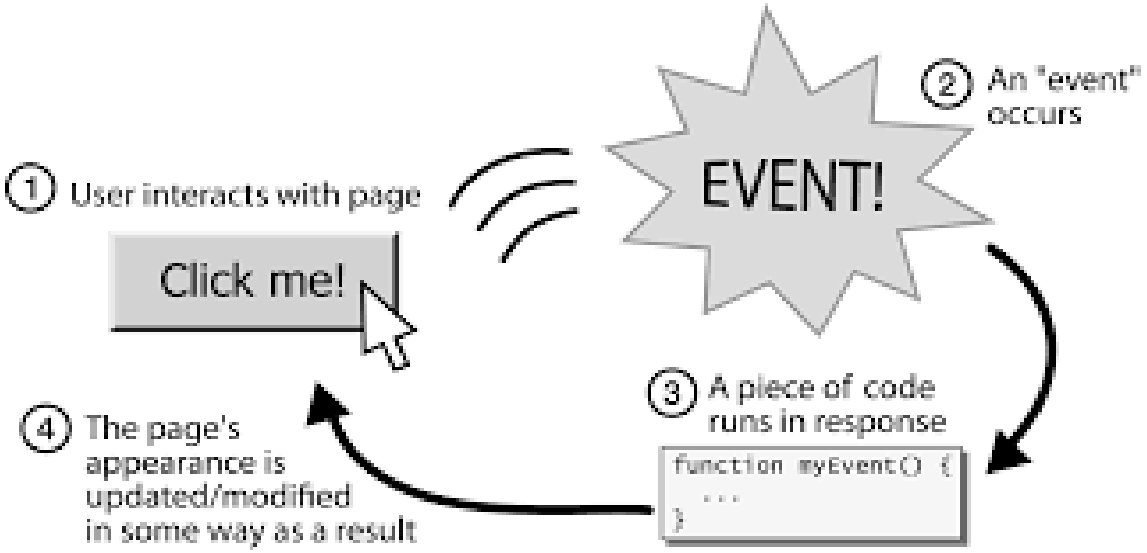


Event Handling

Event Driven Programming

Event Driven Programming is a style of coding where a program's overall flow of execution is dictated by events.

- The program loads, then waits for user input events.
- As each event occurs, the program runs particular code to respond.
- The overall flow of what code is executed is determined by the series of events that occur
- Contrast with application- or algorithm-driven control where program expects input data in a pre-determined order and timing.

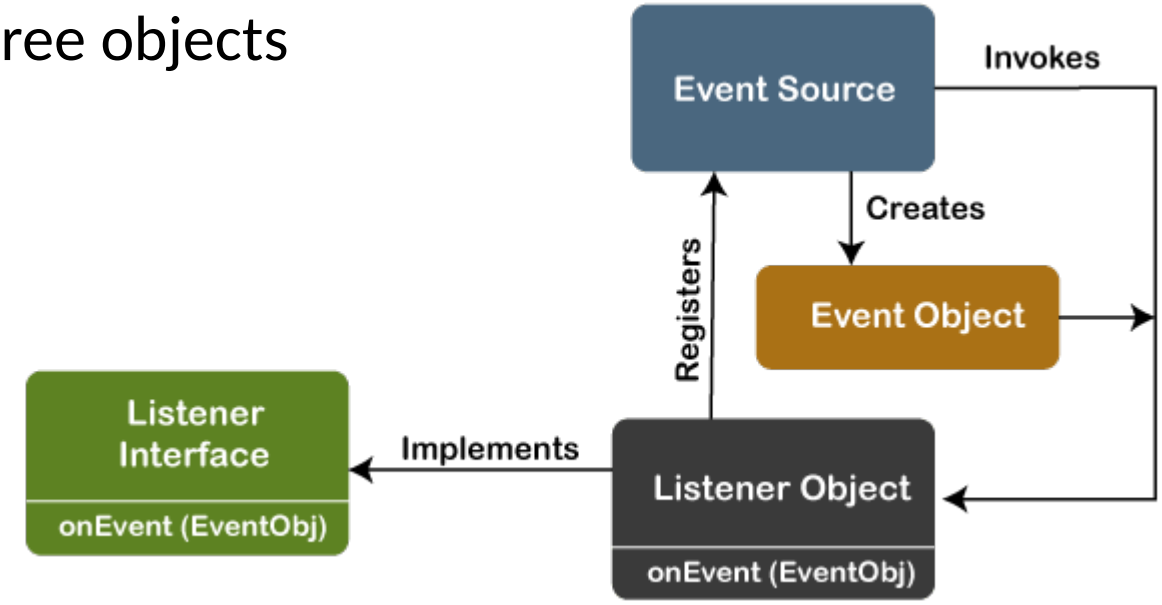


Delegation Event Model

The delegation event model provides a consistent way of generating and processing events. In this model a source generates an event and sends it to one or more listeners. A listener waits until it receives an event and once it receives an event, processes the event and returns

The model involves three objects

1. Event Object
2. Event Source
3. Event Listner

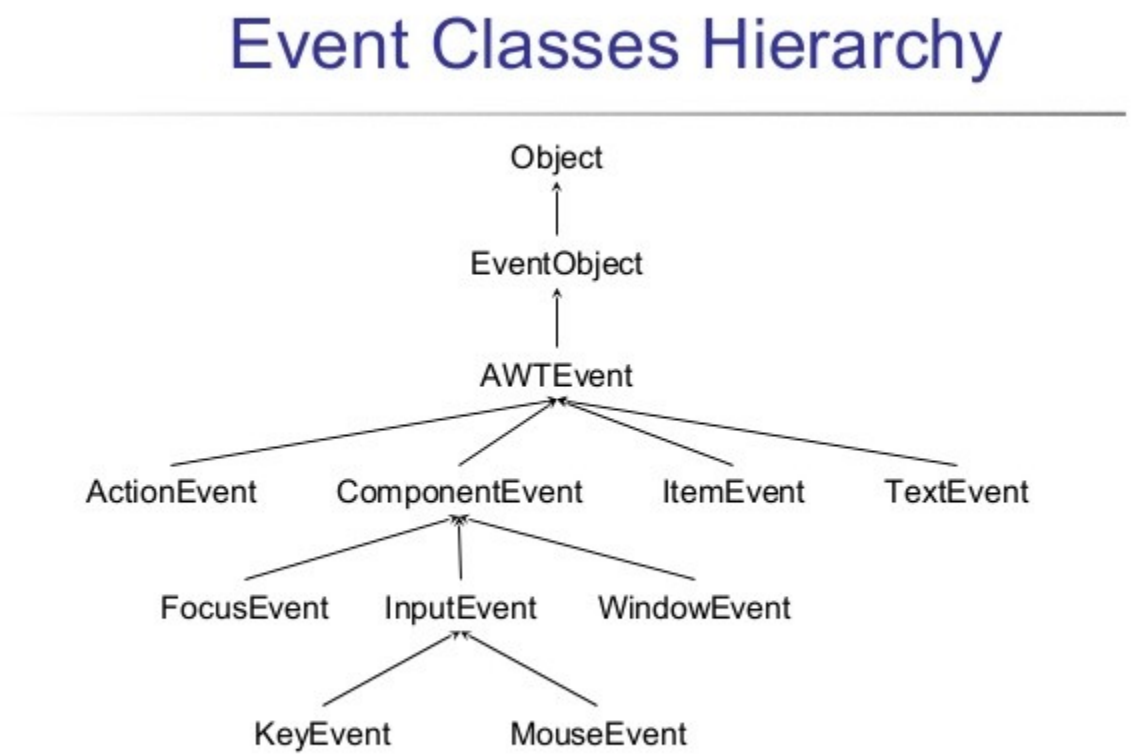


Event Source

A source is an object that generates an event. A source might generate more than one event. Examples of sources are button, check box, radio button, text box etc.

Event…

An event is an object that describes a state change in the source(A GUI component) due to user's interaction.



Event	Description
ActionEvent	Generated when a button is pressed, a list item is double-clicked, or a menu item is selected
AdjustmentEvent	Generated when a scroll bar is manipulated
ComponentEvent	Generated when a component is hidden, moved, resized, or becomes visible
ContainerEvent	Generated when a component is added to or removed from a container
FocusEvent	Generated when a component gains or loses focus
InputEvent	Absract superclass for all component input event classes
ItemEvent	Generated when a checkbox or list item is clicked; also occurs when a choice selection is made or a checkable menu item is selected or deselected
KeyEvent	Generated when input is received from keyboard

Types of Events

Event	Description
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exists a component
MouseMotionEvent	Generated when the mouse wheel is moved
TextEvent	Generated when the value of a text area or text field is changed
WindowEvent	Generated when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit
DocumentEvent	Text component uses a Document to represent its content. Document events occur when the content of a document changes in any way.

Event Listener

A listener is an object that is notified when an event occurs.

A listener has two major requirements.

1. First is, listener must be registered with a source to receive notifications
   The general form of a registration method is as follows:
   public void addTypeListener(TypeListener el)
   Where Type : the type of the event
   Example
   1. if the listener object contains the source, use **this** as a listener
      E,g. sourceObj.addMouseListner(this);
   2. if the listener is another class use an object of that class
      E,g. sourceObj.addMouseListner(new ListenerClass());
   A listener can also be unregistered from the source using the remove method whose syntax is as follows:
   public void removeTypeListener(TypeListener tl);
   E,g. sourceObj.removeMouseListner(ml);
2. it must implement the required **listner interfaces** and all the methods to receive and process these notifications

Event –Event Listener-Listener Method

Event Class	Listener Interface	Listener Method
ActionEvent	ActionListener	actionPerformed(*ActionEvent ae*)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(*AdjustmentEvent ae*)
ComponentEvent	ComponentListener	componentHidden(*ComponentEvent ce*)
		componentMoved(*ComponentEvent ce*)
		componentResized(*ComponentEvent ce*)
		componentShown(*ComponentEvent ce*)
ContainerEvent	ContainerListner	componentAdded(*ContainerEvent ce*)
		componentRemoved(*ContainerEvent ce*)
FocusEvent	FocusListener	focusGained(FocusEvent fe)
		focusLost(FocusEvent fe)

Event Listener

Event Class	Listener Interface	Listener Method
ItemEvent	ItemListener	itemStateChanged(*ItemEvent ie*)
KeyEvent	KeyListener	keyPressed(KeyEvent ke)
		keyReleased(KeyEvent ke)
		keyTyped(KeyEvent ke)
MouseEvent	MouseListner	mouseClicked(MouseEvent me)
		mouseEntered(MouseEvent me)
		mouseExited(MouseEvent me)
		mousePressed(MouseEvent me)
		mouseReleased(MouseEvent me)
MouseMotionEvent	MouseMotionListener	mouseDragged(MouseEvent me)
		mouseMoved(MouseEvent me)
TextEvent	TextListner	textValueChanged(*TextEvent te*)

Event Listener

Event Class	Listener Interface	Listener Method
DocumentEvent	DocumentListener	changedUpdate(DocumentEvent de)
		changedInsert(DocumentEvent de)
		changedInsert(DocumentEvent de)
WindowEvent	WindowListener	windowActivated(*WindowEvent we*)
		windowClosed(*WindowEvent we*)
		windowClosing(*WindowEvent we*)
		windowDeactivated(*WindowEvent we*)
		windowDeiconified(*WindowEvent we*)
		windowIconified(*WindowEvent we*)
		windowOpened(*WindowEvent we*)
		mousePressed(*WindowEvent we*)
		mouseReleased(*WindowEvent we*)

Different Ways of Defining Listener

- By making the class that creates source components a listener class

```
class MyClass extends JFrame implements ActionListener{
        MyClass(){
            .
            .
            .
            <source>.addActionListener(this);
            . . .
        }
        public void actionPerformed(ActionEvent ae){}
}
```

Different Ways of Defining Listener

- **Defining a separate Listener class**

```
class MyClass extends Jframe {
        MyClass(){
                .
                .
                MyHandler mh = new MyHandler();
                <source>.addActionListener(mh);
                . . .
        }
}
class MyHandler implements ActionListener, ItemListener{
        . . .
        public void actionPerformed(ActionEvent ae){. . .}
        public void itemStateChanged(ItemEvent ie){. . .}
}
```

Different Ways of Defining Listener

- **Define Inner Class**

```
class MyClass extends Jframe {
        MyClass(){
            .
            .
            MyHandler mh = new MyHandler();
            <source>.addActionListener(mh);
            . . .
        }
        private class MyHandler implements ActionListener{
            public void actionPerformed(ActionEvent ae){. . .}
        }
}
```

Different Ways of Defining Listener

- **Define Anonymous Inner classes**

```
class MyClass extends JFrame{
        . . .
        MyClass() {
            . . .
            <source>.addActionListener(new ActionListener(){
                public void actioPerformed(ActionEvent ae){
                  . . .
                }
            } );
            <source>.addWindowListener(new WindowAdapter(){
              public void WindowClosing(WindowEvent ae){
                  dispose();
                  System.exit(0);
              }
            });
}    }
```

MouseEvent Methods

int getButton(): Returns which, if any, of the mouse buttons has changed state. BUTTON1, BUTTON2,or BUTTON3

int getClickCount(): Returns the number of mouse clicks associated with this event.

Point getLocationOnScreen(): Returns the absolute x, y position of the event.

static String getMouseModifiersText(int modifiers):Returns a String describing the modifier keys and mouse buttons that were down during the event, such as "Shift", or "Ctrl+Shift".

Point getPoint(): Returns the x,y position of the event relative to the source component.

int getX() , int getY(): Returns the horizontal x position or the vertical y of the event relative to the source component.

int getXOnScreen(), int getYOnScreen(): Returns the absolute horizontal x or the absolute vertical position of the event.

Example

```
package eventdemo;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import javax.swing.*;
public class EventDemo implements MouseMotionListener,
MouseListener{
   private JFrame appWindow;
   private JTextField text;
   public EventDemo() {
      appWindow=new JFrame("Event Demo");
      text=new JTextField(50);
      JLabel label=new JLabel("Click and drag the mouse");
      appWindow.add(label,BorderLayout.NORTH);
      appWindow.add(text,BorderLayout.SOUTH);
```

```
    Container c = appWindow.getContentPane();
    c.setBackground(Color.red);
    appWindow.setBackground(Color.blue);
    appWindow.addMouseMotionListener(this);
    appWindow.addMouseListener(this);
    appWindow.setSize(300,200) ;
    appWindow.setVisible(true);
}
public void mouseDragged( MouseEvent e) {
    String s = "Mouse dragging: X= " + e.getX() + " Y= "
    + e.getY() ;
    text.setText(s) ;
}
public void mouseEntered( MouseEvent e) {
    String s = "The mouse entered" ;
    text.setText(s) ;
}
```

Example...

```
    public void mouseExited( MouseEvent e) {
        String s = "The mouse has left the building" ;
        text.setText(s) ;
    }
    // All methods of a listener must be present in the
    // class even if they are not used
    public void mouseMoved(MouseEvent e) { }
    public void mousePressed(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public static void main(String[] args) {
        new EventDemo();
    }
}
```

ActionEvent Methods

Important Methods

String getActionCommand(): Returns the command string associated with this action. The caption on the button.

int getModifiers(): Returns the modifier keys held down during this action event.

- **ALT_MASK** -- The alt modifier.
- **CTRL_MASK** -- The control modifier.
- **META_MASK** -- The meta modifier.
- **SHIFT_MASK** -The shift modifier.

long getWhen(): Returns the timestamp of when this event occurred.

ItemEvent Methods and Fields

Field	Description
static int DESELECTED	This value is returned by the get StateChange method when an item in the list is deselected.
static int SELECTED	This value is returned by the get StateChange method when an item in the list is selected.

Method	Description
object getItem()	Returns the item that was selected or deselected
object getSource()	Returns the object on which the event occurred
int getStateChange()	Returns either SELECTED or DESELECTED to indicate whether the item was selected or deselected

DocumentEvent Methods

Document getDocument():Gets the document that sourced the change event.

int getLength(): Returns the length of the change.

int getOffset():Returns the offset within the document of the start of the change.

DocumentEvent.EventType getType(): Gets the type of event(CHANGE, INSERT, or REMOVE)

The Parking Problem: Time Class

```
package parking;
public class Time implements Comparable<Time>{
private int hour;
private int minute;
public Time(String time) {
   String [] t=time.split(":");
   hour=Integer.parseInt(t[0]);
   minute=Integer.parseInt(t[1]);
}
public Time(int hour, int minute) {
   super();
   this.hour = hour;
   this.minute = minute;
}
```

The Parking Problem: Time Class

```
public Time(double hours) {
   this.hour=(int)hours;
   this.minute=(int)((hours%1)*60);
}
public int getHour() {
   return hour;
}
public void setHour(int hour) {
   this.hour = hour;
}
public int getMinute() {
   return minute;
}
public void setMinute(int minute) {
   this.minute = minute;
}
```

The Parking Problem: Time Class

```
@Override
public String toString() {
    return hour + ":" + minute;
}
public double convertToHours() {
    return hour + minute/60.0;
}
public int compareTo(Time t) {
   if(hour >t.getHour())
       return 1;
   else if(hour < t.getHour())
       return -1;
   else {
      if(minute>t.getMinute())
            return 1;
```

The Parking Problem: Time Class

```
    else if(minute < t.getMinute())
        return -1;
    else
        return 0;
}

}

}
```

The Parking Problem: Parking Class

```
package parking;
public class Parking {
public final double CAR_FIRST_lMT=3;
public final double TRUCK_FIRST_lMT=2;
public final double BUS_FIRST_lMT=1;
public final double CAR_FIRST_RATE=5;
public final double CAR_SECOND_RATE=7.50;
public final double TRUCK_FIRST_RATE=8;
public final double TRUCK_SECOND_RATE=11.30;
public final double BUS_FIRST_RATE=13;
public final double BUS_SECOND_RATE=18.70;
public final String CAR="CAR";
public final String TRUCK="TRUCK";
public final String BUS="BUS";
private Time in;
```

The Parking Problem: Parking Class

```
private Time out;
String vehicle;
public Parking() {

}
public Parking(Time in, Time out, String vehicle) throws
IllegalArgumentException{
   String v=vehicle.toUpperCase();
   if(!v.equals(CAR) && !v.equals(TRUCK) && !
   v.equals(BUS)) {
   throw new IllegalArgumentException("Invalid vehicle
   type!");
   }else if(out.compareTo(in)==-1) {
   throw new IllegalArgumentException("Vehicle leaving
   time should be greater than entery time!");
   }
   this.in = in;
```

The Parking Problem: Parking Class

```
    this.out = out;
    this.vehicle=vehicle;
}
public Parking(int hourIn, int minuteIn, int hourOut, int
minuteOut, String vehicle) {
    Time in= new Time(hourIn,minuteIn);
    Time out= new Time(hourOut,minuteOut);
    String v=vehicle.toUpperCase();
    if(!v.equals(CAR) && !v.equals(TRUCK) && !
    v.equals(BUS)) {
        throw new IllegalArgumentException("Invalid vehicle
        type!");
    }else if(out.compareTo(in)==-1) {
        throw new IllegalArgumentException("Vehicle leaving
    time   should be greater than entery time!");
    }
    this.in = new Time(hourIn,minuteIn);
```

The Parking Problem: Parking Class

```
    this.out = new Time(hourOut, minuteOut);
    this.vehicle=vehicle;
}
public Time getIn() {
    return in;
}
public void setIn(Time in) {
    this.in = in;
}
public Time getOut() {
    return out;
}
public void setOut(Time out) {
    this.out = out;
}
```

The Parking Problem: Parking Class

```
public Time calculateParkingTime() {
   int hourIn=in.getHour();
   int hourOut=out.getHour();
   int minuteIn=in.getMinute();
   int minuteOut=out.getMinute();
   if(minuteOut<minuteIn) {
   hourOut--;
   minuteOut += 60;
   }
   return new Time(hourOut-hourIn,minuteOut-minuteIn);
}
public double calculateParkingFare() {
   Time pTime=calculateParkingTime();
   System.out.println(pTime);
```

The Parking Problem: Parking Class

```
double hours=pTime.convertToHours();
String type=vehicle.toUpperCase();
double rate=0;
switch (type) {
case CAR:
   if(hours<=CAR_FIRST_lMT) {
       rate=CAR_FIRST_RATE;
   }else {
       rate=CAR_SECOND_RATE;
   }
   break;
   case TRUCK:
   if(hours<=TRUCK_FIRST_lMT) {
       rate=TRUCK_FIRST_RATE;
```

The Parking Problem: Parking Class

```
        }else {
            rate=TRUCK_SECOND_RATE;
        }
    case BUS:
        if(hours<=BUS_FIRST_lMT) {
            rate=BUS_FIRST_RATE;
        }else {
            rate=BUS_SECOND_RATE;
        }
        break;
    }
        return hours*rate;
    }
```

The Parking Problem: Parking Class

```
    public String toString() {
        return String.format("%-16s%s%n%-16s%s%n%-16s%s%n%-
        16s%s%n%-16s%.2f%n%-20s%.2f Birr%n","Vehicle
        Type",vehicle.toUpperCase(),"Time In", in,"Time
        Out",out,"Parking
        Time",calculateParkingTime(),"Parking
        Hours",calculateParkingTime().convertToHours(),"Park
        ing Fare",calculateParkingFare());
    }
}
```

The Parking Problem: ParkingProgram Class

```
package parking;
import java.awt.*;
import javax.swing.*;
public class ParkingProgram implements ActionListener,
ItemListener {
private final String [] vehicleType= {"Car", "Truck",
"Bus"};
private JFrame parkApp;
private JLabel vehicleLbl;
private JComboBox vehicleCbx;
JPanel panel;
JLabel timeInLbl;
JTextField timeInTxt;
JLabel timeOutLbl;
JTextField timeOutTxt;
JLabel timeParkLbl;
JTextField timeParkTxt;
JLabel hourParkLbl;
JTextField hourParkTxt;
```

The Parking Problem: ParkingProgram Class

```
JLabel fareParkLbl;
JTextField fareParkTxt;
JButton clrBtn;
JButton calcBtn;
public ParkingProgram() {
    parkApp=new JFrame("Parking Fare");
        ImageIcon imgicon = new ImageIcon("dahlia.png");
         parkApp.setLayout(null);
        parkApp.setSize(400, 400);
    parkApp.setIconImage(imgicon.getImage());
         JPanel panel=new JPanel();
    panel.setBounds(50, 10, 300,350);
         panel.setLayout(new GridLayout(0,2,10,10));
    vehicleLbl= new JLabel("Vehicle Type:");
         vehicleLbl.setPreferredSize(new
Dimension(140,10));
         vehicleCbx=new JComboBox(vehicleType);
         vehicleCbx.setPreferredSize(new
Dimension(140,10));
```

The Parking Problem: ParkingProgram Class

```
    vehicleCbx.addItemListener(this);
        panel.add(vehicleLbl);
        panel.add(vehicleCbx);
        timeInLbl= new JLabel("Time In(h:m):");
        timeInLbl.setPreferredSize(new
Dimension(140,10));
        timeInTxt= new JTextField(110);
        timeInTxt.setPreferredSize(new
Dimension(140,10));
timeInTxt.setBorder(BorderFactory.createCompoundBorder(
timeInTxt.getBorder(),BorderFactory.createEmptyBorder(5
, 5, 5, 5)));
  panel.add(timeInLbl);
        panel.add(timeInTxt);
        timeOutLbl= new JLabel("Time Out(h:m):");
        timeOutLbl.setPreferredSize(new
Dimension(140,10));
       timeOutTxt= new JTextField(110);
        timeOutTxt.setPreferredSize(new
```

The Parking Problem: ParkingProgram Class

```
timeOutTxt.setBorder(BorderFactory.createCompoundBorder
(timeOutTxt.getBorder(),BorderFactory.createEmptyBorder
(5, 5, 5, 5)));
        panel.add(timeOutLbl);
        panel.add(timeOutTxt);
        timeParkLbl= new JLabel("Parking Time:");
        timeParkLbl.setPreferredSize(new
Dimension(140,10));
        timeParkTxt= new JTextField(110);
        timeParkTxt.setEditable(false);
        timeParkTxt.setPreferredSize(new
Dimension(140,10));
timeParkTxt.setBorder(BorderFactory.createCompoundBorde
r(timeParkTxt.getBorder(),BorderFactory.createEmptyBord
er(5, 5, 5, 5)));
    panel.add(timeParkLbl);
         panel.add(timeParkTxt);
         hourParkLbl= new JLabel("Parking Hour:");
```

The Parking Problem: ParkingProgram Class

```
    hourParkLbl.setPreferredSize(new Dimension(140,10));
        hourParkTxt= new JTextField(110);
        hourParkTxt.setEditable(false);
        hourParkTxt.setPreferredSize(new
Dimension(140,10));
hourParkTxt.setBorder(BorderFactory.createCompoundBorder
(hourParkTxt.getBorder(),BorderFactory.createEmptyBorder
(5, 5, 5, 5)));
        panel.add(hourParkLbl);
        panel.add(hourParkTxt);
        fareParkLbl= new JLabel("Parking Fare:");
        fareParkLbl.setPreferredSize(new
Dimension(140,10));
    fareParkTxt= new JTextField(110);
        fareParkTxt.setEditable(false);
        fareParkTxt.setPreferredSize(new
Dimension(140,10));

fareParkTxt.setBorder(BorderFactory.createCompoundBorder
```

The Parking Problem: ParkingProgram Class

```
        panel.add(fareParkLbl);
            panel.add(fareParkTxt);
            calcBtn=new JButton("Calculte Fare");
            calcBtn.setPreferredSize(new Dimension(140,10));
            calcBtn.addActionListener(this);
            panel.add(calcBtn);
            clrBtn=new JButton("Clear");
            clrBtn.setPreferredSize(new Dimension(140,10));
            clrBtn.addActionListener(this);
            panel.add(clrBtn);
            parkApp.add(panel);
        parkApp.setVisible(true);
}
public static void main(String[] args){
    new  ParkingProgram();
}
}
```

The Parking Problem: ParkingProgram Class

```
private void clear() {
        timeInTxt.setText("");
    timeOutTxt.setText("");
    timeParkTxt.setText("");
    hourParkTxt.setText("");
    fareParkTxt.setText("");
}
@Override
public void itemStateChanged(ItemEvent e) {
    clear();
}
@Override
public void actionPerformed(ActionEvent e) {
if(e.getSource()==clrBtn) {
    clear();
```

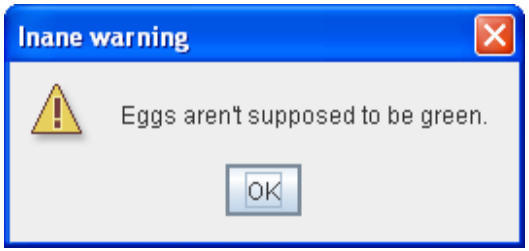
The Parking Problem: ParkingProgram Class

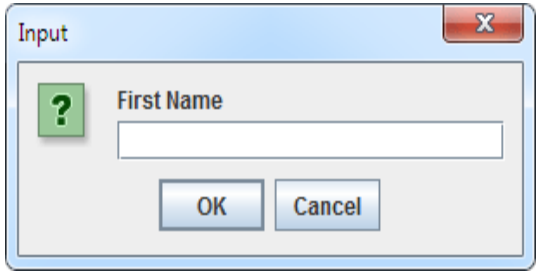
```
}else if(e.getSource()==calcBtn){
    Parking park=new Parking(new
    Time(timeInTxt.getText()),new
    Time(timeOutTxt.getText()),(String)
    vehicleCbx.getSelectedItem());
    Time parkTime=park.calculateParkingTime();
    timeParkTxt.setText(parkTime.toString());
    hourParkTxt.setText(String.format("%.2f",parkTime.conv
    ertToHours()));
    fareParkTxt.setText(String.format("%.2f",park.calculat
    eParkingFare()));

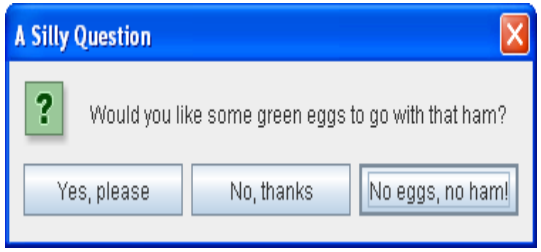
}
}
}
}
```

Working with Standard Dialogs

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box.







Message Dialog

Static Methods To display message dialog

- static void showMessageDialog(Component parentComponent, Object message): Brings up an information-message dialog titled "Message".
- static void showMessageDialog(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.
- static void: showMessageDialog(Component parentComponent, Object message, String title, int messageType, Icon icon): Brings up a dialog displaying a message, specifying all parameters.

Where

messageType: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE

QUESTION_MESSAGE, PLAIN_MESSAGE

E,g.

JFrame fr= new Jframe(“Demo”);

JOptionPane.showMessageDialog(fr, “Error occurred!!!", “Error", JOptionPane.ERROR_MESSAGE);

Confirm Dialog

Static Methods To display message dialog

- static int showConfirmDialog(Component parentComponent, Object message): Brings up a dialog with the options Yes, No and Cancel; with the title, Select an Option.
- 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.
- 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.

E,g.

JFrame fr= new Jframe(“Demo”);

int ans=JOptionPane.showInternalConfirmDialog(fr,"please choose one", "information", JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE);

Static Methods To display message dialog

- static String showInputDialog(Component parentComponent, Object message): Shows a question-message dialog requesting input from the user parented to parentComponent.
- static String showInputDialog(Component parentComponent, Object message, Object initialSelectionValue): Shows a question-message dialog requesting input from the user and parented to parentComponent.
- static String showInputDialog(Component parentComponent, Object message, String title, int messageType): Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.
- 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.

JFrame fr= new Jframe(“Demo”);

String ans=JOptionPane.showInternalConfirmDialog(fr,“Enter your full name", “Enter Name", JOptionPane.QUESTION_MESSAGE);

Ans will be null if the user choose cancel.

Example Message Dialog

```
package hello;
import java.awt.Container;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
public class DialogDemo implements ItemListener{
    private final String [] dialogType= {"","Message Box", "Confirm Box", "Input Box"};
    private JFrame appWindow;
    private JComboBox dialogCbx;
    private JLabel dialogLbl;
    public DialogDemo() {
```

Evample Message Dialog

```
        appWindow=new JFrame("Dialog Demo");
        appWindow.setLayout(null);
        dialogLbl= new JLabel("Select Dialog");
        dialogLbl.setBounds(60, 20, 100, 30);
        dialogCbx=new JComboBox(dialogType);
        dialogCbx.setBounds(165, 20, 150, 30);
        dialogCbx.addItemListener(this);
        Container c=appWindow.getContentPane();
        c.add(dialogLbl);
        c.add(dialogCbx);
        appWindow.setSize(400,400) ;
        appWindow.setVisible(true);
}
public static void main(String[] args) {
        new DialogDemo();
}
```

Evample Message Dialog

```
@Override
public void itemStateChanged(ItemEvent e) {
 if(e.getItem().equals("Message Box")) {
            if(e.getStateChange()==ItemEvent.SELECTED){
            JOptionPane.showMessageDialog(appWindow, "Java is really
            cool.","Message Box",JOptionPane.INFORMATION_MESSAGE);
        dialogCbx.setSelectedIndex(0);
            }
 }else if(e.getItem().equals("Confirm Box")) {
            if(e.getStateChange()==ItemEvent.SELECTED) {
        int op=JOptionPane.showConfirmDialog (appWindow, "Do you like
Java?.","Confirm Box", JOptionPane.YES_NO_OPTION,
JOptionPane.QUESTION_MESSAGE);
            if(op==JOptionPane.YES_OPTION) {
```

Evample Message Dialog

```
                    JOptionPane.showMessageDialog(appWindow, "That is
                great!!!.", "MessageBox",
                JOptionPane.INFORMATION_MESSAGE);
            }else {
                JOptionPane.showMessageDialog(appWindow, "Sorry, we will
miss you!!! .","Message Box",
JOptionPane.INFORMATION_MESSAGE);
            }
            dialogCbx.setSelectedIndex(0);
        }
      }else if(e.getItem().equals("Input Box")) {
       if(e.getStateChange()==ItemEvent.SELECTED) {
            String ans=JOptionPane.showInputDialog(appWindow, "How old are
            you?", "Input Box", JOptionPane.QUESTION_MESSAGE);
```

Evample Message Dialog

```
        if(ans==null) {
                        JOptionPane.showMessageDialog(appWindow, "You are not
                        interested!!!.","Message Box",JOptionPane.INFORMATION_MESSAGE);
        }else{
                    int val=Integer.parseInt(ans);
                    JOptionPane.showMessageDialog(appWindow, "Your parents are atleast "
                    + 15 + "years!!!","Message Box",JOptionPane.INFORMATION_MESSAGE);
              dialogCbx.setSelectedIndex(0);
        }
      }

   }
}
```

Standard Dialog

Common Methods

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
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 int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Working with Menu

Menu

JMenuBar, JMenu, JMenuItem, JCheckBoxMenuItem, and JRadioButtonMenuItem are a part of Java Swing package to implement menus in a frame

1. JMenuBar: is an implementation of menu bar . It is used to display menubar on the window or frame. It may have several menus. The JMenuBar contains one or more JMenu objects
2. JMenu is a pull down menu component which is displayed from the menu bar. It inherits the JMenuItem class. It contains several JMenuItem Object.
3. The JMenuItem class represents the actual item in a menu. All items in a menu should derive from class JMenuItem, or one of its subclasses.
4. JCheckBoxMenuItem: represents checkbox which can be included on a menu
5. JRadioButtonMenuItem: is a menu item that is part of a group of menu items in which only one item in the group can be selected

JMenuBar

Constructors :

JMenuBar() : Creates a new MenuBar.

JMenu() : Creates a new Menu with no text.

JMenu(String name) : Creates a new Menu with a specified name.

JMenuItem(): Creates a JMenuItem with no set text or icon.
JMenuItem(String text):Creates a JMenuItem with the specified text.
JMenuItem(Icon icon): Creates a JMenuItem with the specified icon.
JMenuItem(String text, Icon icon): Creates a JMenuItem with the specified text and icon.
JMenuItem(String text, int mnemonic): Creates a JMenuItem with

JMenuBar

Constructors :

JMenuBar() : Creates a new MenuBar.

In order to add the menu bar to the container(usually top level container). Use the setMenuBar method of the container.

Example

Jframe frame = new JFrame(“My Window”);

…

…

JMenuBar menuBar= new JMenuBar(); //creating menu bar

…//add menu to menu bar

frame.setMenuBar(); //attaching the menubar with the frame

JMenu

Constructors :

JMenu() : Creates a new Menu with no text.

JMenu(String name) : Creates a new Menu with a specified name.

In order to add the menu to the menubar use the add method of the menubar container.

```
Jframe frame = new JFrame("My Window");
...
JMenuBar menuBar= new JMenuBar(); //creating menu bar
JMenu  fileMnu= new Jmenu("File");
fileMenu.setMnemonic(KeyEvent.VK_F);//setting mnemonic
menuBar.add(fileMenu); //adding menu to the menubar
...//create and add more menu
frame.setMenuBar(); //attaching the menubar with the frame
```

JMenuItem

Constructors :

JMenuItem(): Creates a JMenuItem with no set text or icon.
JMenuItem(String text):Creates a JMenuItem with the specified text.
JMenuItem(Icon icon): Creates a JMenuItem with the specified icon.
JMenuItem(String text, Icon icon): Creates a JMenuItem with the specified text and icon.
JMenuItem(String text, int mnemonic): Creates a JMenuItem with the specified text and keyboard mnemonic.

JMenuItem…

```
Jframe frame = new JFrame(“My Window”);
…
JMenuBar menuBar= new JMenuBar(); //creating menu bar
JMenu  fileMnu= new Jmenu(“File”);
fileMenu.setMnemonic(KeyEvent.VK_F);//setting mnemonic
menuBar.add(fileMenu); //adding menu to the menubar
…//create and add more menu
JMenuItem openMnuItm=new MenuItem(“Open”, KeyEvent.VK_O);
fileMenu.add(openMnuItm);
JMenuItem impMnuItm=new MenuItem(“Import”, KeyEvent.VK_I);
fileMenu.add(impMnuItm);
JMenuItem exitMnuItm=new MenuItem(“Exit”, KeyEvent.VK_X);
fileMenu.add(exitMnuItm);
frame.setMenuBar(); //attaching the menubar with the frame
```

Creating sub menu

```
Jframe frame = new JFrame(“My Window”);
…
JMenuBar menuBar= new JMenuBar(); //creating menu bar
JMenu  fileMnu= new Jmenu(“File”);
fileMenu.setMnemonic(KeyEvent.VK_F);//setting mnemonic
menuBar.add(fileMenu); //adding menu to the menubar
…//create and add more menu
JMenuItem openMnuItm=new MenuItem(“Open”, KeyEvent.VK_O);
fileMenu.add(openMnuItm);
JMenuItem impMnuItm=new MenuItem(“Import”, KeyEvent.VK_I);
fileMenu.add(impMnuItm);
JMenuItem fImpMnuItm=new MenuItem(“Import File”, KeyEvent.VK_L);
JMenuItem aImpMnuItm=new MenuItem(“Import Archive”, KeyEvent.VK_A);
impMenu.add(aImpMnuItm);
impMenu.add(fImpMnuItm);
JMenuItem exitMnuItm=new MenuItem(“Exit”, KeyEvent.VK_X);
fileMenu.add(exitMnuItm);
frame.setMenuBar(); //attaching the menubar with the frame
```

Handling Events from Menu Items

To detect when the user chooses a JMenuItem, you can listen for action events (just as you would for a JButton). Your class implements ActionListner and implement ActionPerformed

```
Jframe frame = new JFrame(“My Window”);
…
JMenuBar menuBar= new JMenuBar(); //creating menu bar
JMenu  fileMnu= new Jmenu(“File”);
fileMenu.setMnemonic(KeyEvent.VK_F);//setting mnemonic
menuBar.add(fileMenu); //adding menu to the menubar
…//create and add more menu
JMenuItem openMnuItm=new MenuItem(“Open”, KeyEvent.VK_O);
fileMenu.add(openMnuItm);
```

Handling Events…

```
JMenuItem impMnuItm=new MenuItem(“Import”, KeyEvent.VK_I);
fileMenu.add(impMnuItm);
JMenuItem fImpMnuItm=new MenuItem(“Import File”, KeyEvent.VK_L);
JMenuItem aImpMnuItm=new MenuItem(“Import Archive”, KeyEvent.VK_A);
impMenu.add(aImpMnuItm);
impMenu.add(fImpMnuItm);
JMenuItem exitMnuItm=new MenuItem(“Exit”, KeyEvent.VK_X);
fileMenu.add(exitMnuItm);
openMnuItm.addActionListner(this);
fImpMnuItm.addActionListner(this);
aImpMenuItem.addActionListner(this);
exitMnuItm.addActionListner(this);
frame.setMenuBar(); //attaching the menubar with the frame
```

Handling Events…

```
public void actionPerformed(ActionEvent e) {
      if(e.getSource()== exitMnuItm){
           System.exit(0);
     }else if(……){ //for each menu item
     }
     …
     …
}
```