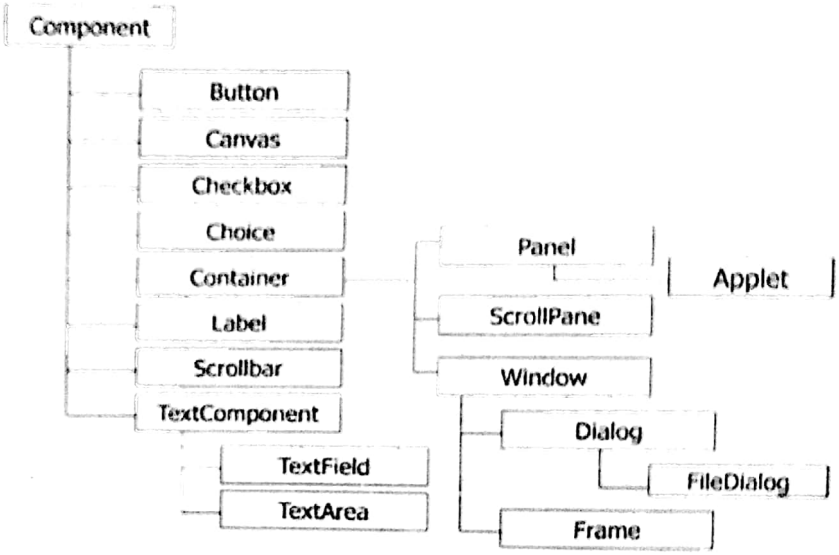
AWT, Swing and Event Handling

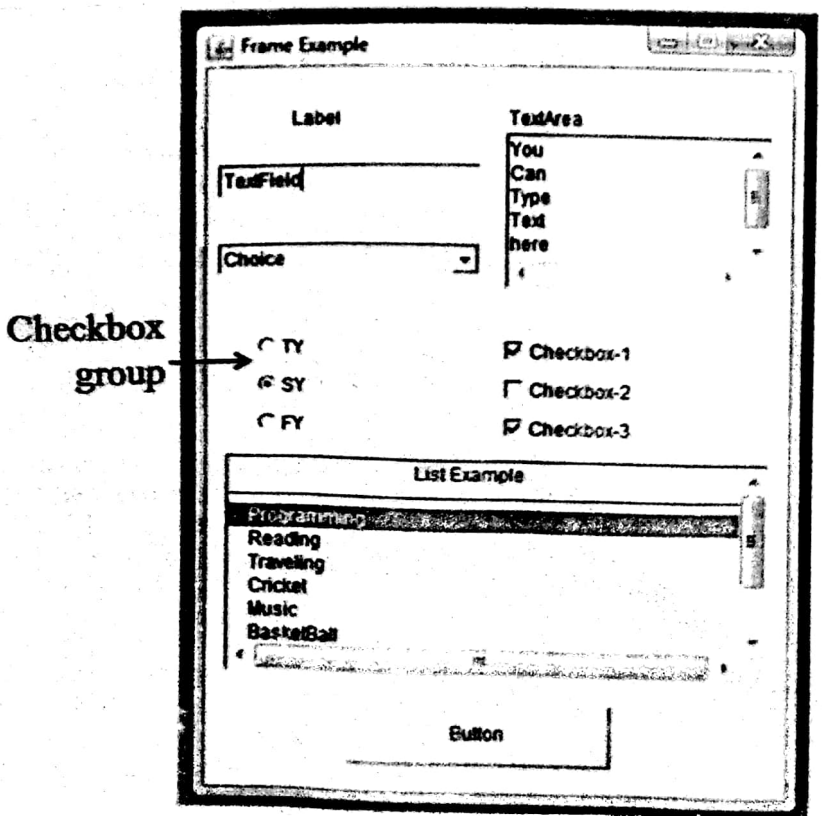
Java provides predefined classes and interfaces using which we can easily create GUI components like window, buttons, menu and it also take the shape and form of underlying OS. As different OS can have different styles of windows and their components, AWT(Abstract Window Toolkit) is used to achieve platform independence. AWT also supports event handling mechanism, so programmer can write event handling code using AWT. AWT components can be used to accept input from user which is better than traditional way of accepting input using I-O streams. AWT has a basic class named ToolKit.

Toolkit class: encapsulates details of the underlying OS & hardware that a JVM is running on. Its object is created when the JVM starts, before any of the application classes are loaded. Toolkit class defines the methods that create OS peers for AWT components and these methods are called automatically as the components are created & thus shouldn't be called by the programmer. You can get a Toolkit object with the **getDefaultToolkit()** method of Toolkit class.

Containers and Components :

A container displays the components and components must be displayed in a container. Component is an abstract class, which is parent class of various components in AWT such as Button, Checkbox, List.
Similarly Container is also an abstract class whose child classes are Window, Panel. As container is also child class of component, container is also a component that allows another Container objects to be placed inside. For example, a Panel is a Container, and a Frame is a Container. Because Panel is also a Component, a Panel can be placed inside a Frame. Such kind of nesting of containers gives better look to UI applications.





Containers:
Container helps in organizing components into manageable group and provides the basic window and dialog services. For creating a GUI, we need at least one Container object. Following are different types of Containers :

Panel	It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window or logically grouping Components together. Panel being a Container, it provides space for other Components. We can add components to Panel using method panelObject.**add**(component) Generally we do not use Panel as the main Container since it is not a Window. We normally create more than one Panels and add them to a Frame, here each Panel can have a different Layout and different Components can be added to these Panels. Default layout for Panel is FlowLayout. It allows sub panels to have their own layout manager.
ScrollPane	ScrollPane is a Container that can hold a Component that is even bigger than the ScrollPane itself. The ScrollPane can have scrollbars with which it can be scrolled to view its contents.
Window	Top level Window, not contained within any other object.(Generally not used)
Frame	It is a **fully functioning window** with its own title and icons.
Dialog	It can be thought of as a **pop-up window** but it is not a fully functioning window like Frame. Dialog is a Window that is used to display some message and get a response from the user. For example, a small window that asks the user "Are you sure?" can be implemented using Dialog The Dialog in the above example can have two Buttons that says "Yes" and "No" Dialog Boxes are primarily used to obtain inputs or display short messages or error messages. It does not have menu bar and it is always a child window of a top level window. Like frame window, controls can be added. They Can be modal or modeless. For example, **FileDialog** is a window that appears whenever we select File – Open in any standard windows application. The user can browse through the different drives and folders and choose a required file.

Components: Following Table explains different components used in AWT

Label	Label is a simple control used to display some message or text
	`Label l = new Label("Hello")`
	The method **setText** can be used to set message on the label
	`l.setText("Hello World");`
	The method **getText** can be used to get the message from the label
	`System.out.println( l.getText( ) );`

Button	It creates push button, which user can click.
	`Button okButton = new Button("Ok");  // Button caption is "Ok"`
	Normally we create more than one buttons from which user selects required button. To create more buttons array of button can be used.

Checkbox	Checkboxes are used when we want to select zero or more items from a set of items
	`Checkbox c1 = new Checkbox("English");`
	`Checkbox c2 = new Checkbox("German");`
	`Checkbox c3 = new Checkbox("French");`
	The method **getState( )** will return true if the Checkbox is selected
	`if ( c1.getState( ) )  System.out.println("The user knows English");`

CheckboxGroup	CheckboxGroup can be used to group Checkbox objects together, so that they become Radio Buttons. Radio Buttons are used when we want to select only one item from a set of items.
	`CheckboxGroup cg = new CheckboxGroup( );`
	`Checkbox c1 = new Checkbox("TY", cg, true);`
	`Checkbox c2 = new Checkbox("SY", cg, false);`
	`Checkbox c3 = new Checkbox("FY", cg, false);`

Choice	Choice is a Drop Down List, which is used to select one and only one item from a set of items.
	`Choice c = new Choice( );`
	`c.add("TY");`
	`c.add("SY");`

	The method **getSelectedItem( )** will return the item selected by the user. If there are few options for choice then we use Radio Buttons(CheckboxGroup) and for more options we use Choice(Drop down list). For example : choosing a country or state will need Choice as there is a big list of options, but choosing sex will need Radio button.

List	List is a scrollable list of items, which can be used to select **zero or more** items from set of items
	List interest = new List( );
	interest.add("Reading");
	interest.add("Traveling");
	The method **getSelectedItem( )** will return items selected by the user.
	Depending on number of options, we can choose between a List and a set of Checkboxes.

Scrollbar	Scrollbar is used to increment or decrement some value Scrollbar s = new Scrollbar(Scrollbar.HORIZONTAL,0, 60, 0, 300); The method **getValue( )** is used to get the current value of the Scrollbar
TextField **TextArea**	TextField and TextArea both permits the user to type in some input, the only difference is TextField is single line whereas TextArea is multi line. TextField tf = new TextField(10); TextArea ta = new TextArea(10, 20); The method **getText( )** gets the text typed in by the user

Canvas	Canvas lets you draw or write on it. We can create our own Canvas class and override the paint method. The paint method is available for all Components and is automatically called whenever a Component is made visible.
	class MyCanvas extends Canvas
	{
	public void paint(Graphics g)
	{ g.drawLine(10, 10, 100, 100);
	g.drawString("Hello", 10, 10);
	}
	}

A Simple GUI Program:

1. **Instantiate :** Create some AWT container's object to hold components, we normally create an object of type Frame.

Frame f = new Frame("Hello");

Frame will be displayed with caption "Hello"

2. **Set size and Visible :**

We can set size of a that Container using method **setSize**(width, height) or **pack( )**

To set position and size of a container, we can use method **setBounds**(x, y , width, height)

3. **Add Components**

Add Components to a Container by using method containerObject.**add**(componentObject);

4. **Set Visiblity :**

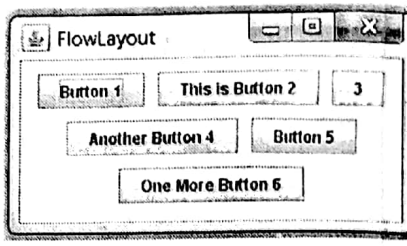
Finally make that container visible by using method containerObject.**setVisible**(true)

```
class SimpleGUI
{
    public static void main(String args[ ])
    {   Frame f = new Frame("Simple GUI Application");
        f.setSize(300, 400);
        Label l1 = new Label("WellCome to my Project");
        f.add( l1 );
        Label l2 = new Label("Click Ok if you agree our terms");
        f.add( l2 );
        Button ob = new Button("Ok");
        f.add( ob );
        f.setVisible(true);
    }
}
```

pack( ) : method resizes window, taking into account the preferred sizes of its components

Layout Manager:

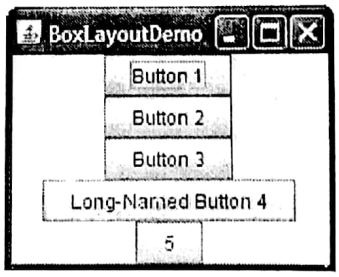
Each Container has a Layout Manager that will decide the size and the position of Components placed on it. The programmer can change the default Layout of the Container using the method **setLayout**. You simply add components to your container, and the layout manager determines where each component will go and what its size will be, Except Null layout as it uses positioning and coordinates given by user. A layout manager lays out components in container that makes GUI more portable. GUI created using No layout with an IDE looks great on Windows, but not so great on a Unix or Macintosh platform. The same problem can occur if you try to lay out components exactly where you want them. Different Layout managers available are Flow Layout, Grid Layout, Border Layout, Card Layout, GridBag Layout and Null Layout .



FlowLayout :

In this type of layout all the components get added to the container in sequential manner from left to right and top to bottom. This type of layout does not change the preferred size of component if space is not enough instead it adds it on next line. Flow layout is default layout for Panel.

Ex.: f.setLayout(new FlowLayout( )); // f is Frame object



BoxLayout :

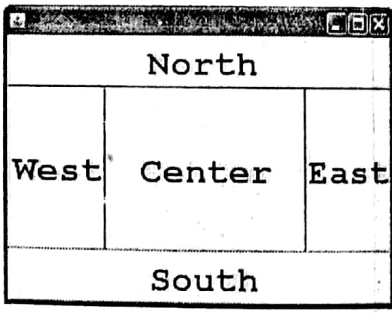
- Takes unlimited number of components.
- Positions each component next to each other only in one direction either horizontall (in one row) or vertically(in one column) .
- Resizes components that are resizable to fill entire container.
- Resizes components that are resizable when the container is resized.
- Syntax : BoxLayout(Container obj, int axis)
  Where axis can be BoxLayout.**X**_AXIS or BoxLayout.**Y**_AXIS

BorderLayout:

This type of layout divides the container into five regions : North, South, East, West, Center. Allowing one component to be added to each region.

BorderLayout is a default layout for Frame.

NORTH, SOUTH, EAST, WEST and CENTER are final static Strings defined in the class BorderLayout,
so instead of "North",
we can even say BorderLayout.NORTH whose value is "North".




```
class TestBorderLayout
{ public static void main(String args[ ])
  {  String direction[ ]={"North","South","East","West","Center");
        Frame f = new Frame("TestBorderLayout");
        f.setSize(300, 400);
        f.setLayout( new BorderLayout( ) );
        Button b[ ]= new Button[5];    // Button array
        for(int i = 0; i <5; i++)
        {       b[i]= new Button("Ok-"+direction[i]);
                f.add(b[i], direction[i]);
        }
        f.setVisible(true);
  }
}
```

CardLayout :

The CardLayout places components on top of each other like a deck of cards and only one card is visible at a time Usually Panel objects are added as cards and each Panel has other components on it. It is little complex to implement as compare to other layouts. Following are steps to use CardLayout :

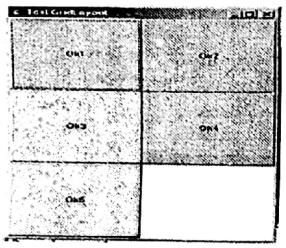
1. Create card panels
2. Add components on card panels
3. Create main panel as card deck
4. Add card panels to card deck (main panel)
5. Add card deck to frame or applet

```
class TestCardLayout
{  public static void main(String args[ ])
   {
        Frame f = new Frame("Test CardLayout");
        f.setSize(300, 400);            // other components can be added on frame

        Panel okPanel = new Panel( );   // 1st card
        okPanel.add(new Label("This is Ok Panel"));

        Panel cancelPanel = new Panel( );       // 2nd card
        cancelPanel.add(new Label("This is Cancel Panel"));

        Panel mainPanel = new Panel( );       // card deck
        CardLayout carlo = new CardLayout();
        mainPanel.setLayout(carlo);

        mainPanel.add(okPanel,"OK");          // 1st card added to Card deck
        mainPanel.add(cancelPanel,"CANCEL");  // 2nd card added

        f.add(mainPanel);  // card deck added to frame
        f.setVisible(true);
   }
}
```

To see other card panel we can use carlo.**show**(mainPanel,"CANCEL");

GridLayout:

It divides a container into a grid of rows and columns, with one component added to each region of the grid and each component having the same size.



```
class TestGridLayout
{  public static void main(String args[ ])
   {     Frame f = new Frame("Test GridLayout");
         f.setSize(300, 400);
         f.setLayout( new GridLayout(3, 2) ); // 3 rows 2 col
         for(int i = 1; i <=5; ++i) f.add(new Button("Ok"+i));
         f.setVisible(true);
   }
}
```

GridBag Layout :

It divides a container into regions similar to GridLayout, except that components do not need to be the same size. Components can span more than one row or column.

Null Layout (Absolute Positioning):

The programmer can cancel any of the default Layout of a Container by calling the method setLayout with null as the parameter. If Container does not have a Layout then Component's size and position can be set using method setBounds

Component.**setBounds**(x, y, width, height);

Mentioned x and y coordinates are in pixels considering upper-left corner of the screen as the origin (0,0). X coordinate is the number of pixels to the right of the origin, and y is number of pixels down from the origin.

To set the position of the component we can use Component.**setPosition**(x, y);

To set the size of the component we can use Component.**setSize**(width, height);

Swing

AWT library took help of underlying OS to generate peers for each UI component, whereas Swing components are purely written in Java. Therefore they are lightweight and platform independent. Due to platform independence swing components need not take on the "look & feel" of the target platform; thus helps in maintaining a **consistent "look & feel"** over all platforms.

For example, an AWT UI program will show button similar to windows button when it is run on windows, the same program shows Macintosh button when it is run on Macintosh system. Whereas swing makes program look like Windows program no matter on what system it is running on.

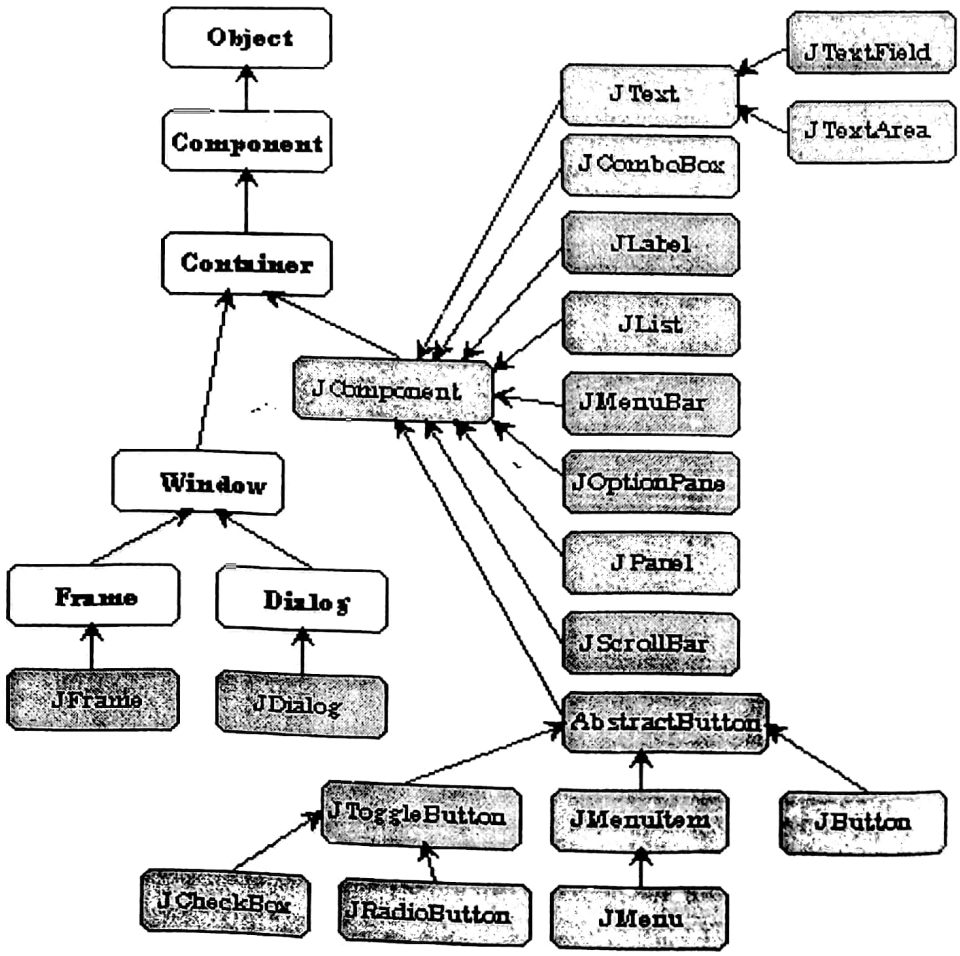
Import javax.swing.*;

Swing is an extended UI(user interface) package in Java, which provides more capabilities for look n feel. For example, swing provides classes for tabbed panes, scroll panes, trees and tables which was not provided by AWT. Swing also provides facility of **tooltip**(whenever mouse cursor moves over any component a hint(text message) can be displayed to help the user).

AWT	Swing
Heavy Weight components	Light weight components
Platform dependent components, so look of UI is defined by underlying OS	Platform independent components, so look of UI is defined by the programmer
AWT has class names as Button, Checkbox, etc	Classes names from swing package begins with "J". Ex. JButton, JCheckbox
AWT Frame cannot close on its own	JFrame handles the close event by default as it hides the frame
Not all AWT components are containers	All the components in Swing are containers
Provides basic UI components	Swing provides all AWT components in better style and also provides extra components like tabbed pane, scroll pane, tree and table
Provides basic layout mangers like FlowLayout, BorderLayout,etc.	AWT layout managers + Swing provides extra layout manager : BoxLayout, CenterLayout, SpringLayout

All swing component classes are present in **javax.swing** package.

Swing components hierarchy is :



Swing components :

AbstractButon : Abstract super class for swing buttons, extended from JComponent

ButtonGroup : Encapsulates mutually exclusive set of buttons

ImageIcon : Encapsulates an icon

JApplet : Swing version of an Applet, supports various panes like content pane, glass pane or root pane.

JButton : Swing version of Push button class that can be associated with **icon** or text(string)

JCheckBox : Swing version of CheckBox

JComboBox : Encapsulates a combo box which is a combination of a drop down list and text field

JLabel : Swing version of Lebel that can display text and/or **icon**. JLabel extends from JComponent.

JRadioButton : Swing version of Radio button

ScrollPane : Encapsulates scrollable pane that can have vertical and/or horizontal scrollbar

JTabbedPane : Encapsulates tabbed panes, multiple panes from which user can select one. Contents of selected pane are visible to the user. Tab can be added to tabbed pane using method **addTab(String, Component)**

JTable : Encapsulates table based control

JTextField : Swing version of a text field that allows you to edit one line of text

JTree : Encapsulates tree based control

void **setToolTipText**(String text)
sets the text that should be displayed as a tooltip when the mouse hovers over the component.

void **setMnemonic**(char) - this method can be used to specify mnemonic character associated with button or menu. Example: b. **setMnemonic**('X'); button will be selected(clicked) when Alt+'X' keys are pressed.
We can also provide other keyboard shortcuts through *accelerators*.
Example: openMenuItem.**setAccelerator**(KeyStroke.getKeyStroke("ctrl O"));

Creating Menu for your application :

Menu system can be designed for a window in 3 steps using classes MenuBar, Menu and MenuItem.

1) **MenuBar** is the area in the window where the menu system appears.
   MenuBar mb = new MenuBar( );

   A menu system is added to a window by setting a MenuBar to that window
   f.**setMenuBar**(mb);

2) The different items that appear on the MenuBar are represented by the class **Menu**.
   Menu m1 = new Menu("File");
   Menu m2 = new Menu("Edit");
   Menu m3 = new Menu("Help");

   A Menu can be added to a MenuBar using **add** method.
   mb.add(m1);
   mb.add(m2);

3) Items that drop down when you select a Menu are objects of **MenuItem**
   MenuItem mi1 = new MenuItem("Open");

   A MenuItem can be added to a Menu using **add** method
   m1.add(mi1);

```
class TestMenu
{       public static void main(String args[ ])
        {       MenuBar mainMenuBar = new MenuBar( );
                Menu fileMenu = new Menu("File");
                fileMenu.add(new MenuItem("New"));
                fileMenu.add(new MenuItem("Open"));
                fileMenu.addSeparator( );
                fileMenu.add(new MenuItem("Save"));
                fileMenu.add(new MenuItem("Save As"));
                mainMenuBar.add(fileMenu);

                Menu editMenu = new Menu("Edit");
                editMenu.add(new MenuItem("Cut"));
                editMenu.add(new MenuItem("Copy"));
                editMenu.add(new MenuItem("Paste"));
                mainMenuBar.add(editMenu);

                Frame f = new Frame("Menu Test");
                f.setMenuBar(mainMenuBar);
                f.setSize(300,300);
                f.setVisible(true);
        }
}
```

To create menu using Swing we have similar 3 classes **JMenuBar**, **JMenu**, **JMenuItem** and we can do it in same 3 steps explained above. To set the menu bar to the frame we need to use **setJMenuBar** method.

We can also associate image with Menuitem, Syntax: JMenuItem(String label, Icon icon)
Example: JMenuItem cutItem = new JMenuItem("Cut", new ImageIcon("cut.gif"));

Note : Menu Item is actually a button so can associate ActionListener to each menuItem.

Some more methods are :
JMenuItem **insert**(JMenuItem menu, int index) - adds a new menu item (or submenu) to the menu at a specific index.
void **remove**(int index) or void **remove**(JMenuItem item) - removes a specific item from the menu

Creating Checkbox Menu Item
JCheckBoxMenuItem(String label) - constructs the checkbox menu item with the given label.
JCheckBoxMenuItem(String label, boolean state)
- constructs the checkbox menu item with the given label and the given initial state (true is checked).

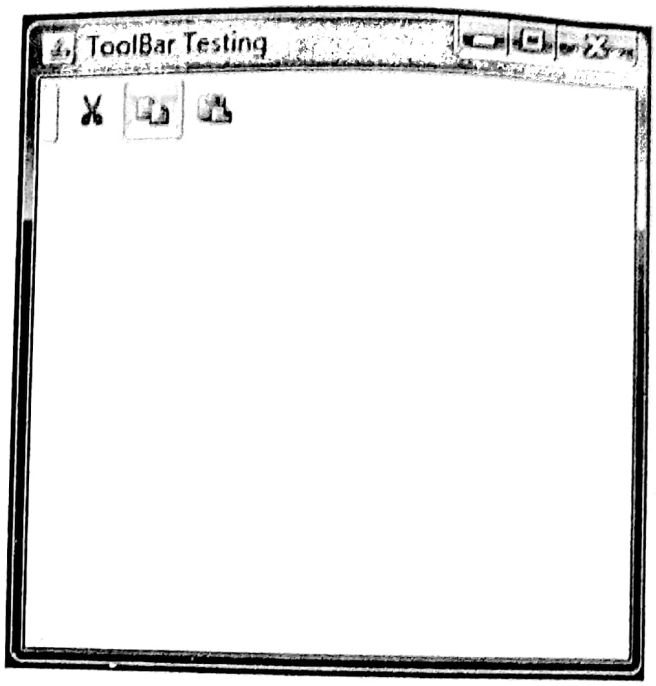
Pop-Up Menus
appears when user clicks right button of mouse. For this we need to handle MouseEvent.
We are showing popup menu at the place of mouse cursor position which can be obtained from mouseEvent object.


```
JFrame f = new JFrame("PopUp Menu Test");

JPopupMenu pm = new JPopupMenu();

pm.add(new JMenuItem("Cut"));
pm.add(new JMenuItem("Copy"));
pm.add(new JMenuItem("Paste"));

f.addMouseListener(new MouseAdapter() {
   public void mousePressed(MouseEvent me)
   {
    if(me.getButton() == MouseEvent.BUTTON3 ) // right click
            pm.show(f, me.getX( ), me.getY( ));
   }    } );

f.setVisible(true);
```

Creating **Toolbar** :

Toolbar is button bar that gives quick access to the most commonly used commands in a program.

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


public class ToolBarDemo
{
   public static void main(String args[])
    {
         JFrame f = new JFrame("ToolBar Testing");
         f.setSize(300,300);
         f.setLayout(new BorderLayout( ));

         JToolBar tb = new JToolBar( );

         JButton cutB =new JButton(new ImageIcon("cut.gif"));
         tb.add(cutB);

         JButton copyB =new JButton(new ImageIcon("copy.gif"));
         tb.add(copyB);

         JButton pasteB=new JButton(new ImageIcon("paste.gif"));
         tb.add(pasteB);

         f.add(tb,BorderLayout.NORTH);


         f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         f.setVisible(true);
   }
}
```



Dialogs

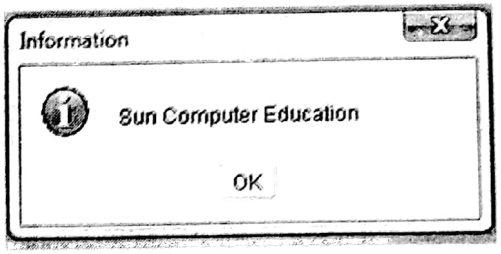
When program is running it might need little interaction with user (need to inform or warn user or accept input from user), in such case dialog box is good option.

- Dialog box is always a child window as it pops up from main window.
- Unlike Frame window, Dialog box cannot have menu bar.
- Dialog box can be **modal or modeless**. Modal dialog box is active and will not allow user to interact with other components of program or main window unless and until dialog box is closed, whereas modeless dialog box allows user to interact with other components of program.
- Swing provides readymade dialog boxes.
  JOptionPane class provides following 4 static methods using which we can easily create dialog box.
  Every dialog box has 3 components: icon, message and One or more option buttons

1) Message dialog
   It is a modal window that shows a message and wait for the user to click OK.

JOptionPane.**showMessageDialog**(Component parent, String message, String title, int messageType, Icon icon)

Parameters :
parent - The parent component (can be null)
message - The message to show on the dialog
(can be a string, icon, component)
title - The string in the title bar of the dialog
messageType - One of ERROR_MESSAGE, PLAIN_MESSAGE,
INFORMATION_MESSAGE,
WARNING_MESSAGE, QUESTION_MESSAGE,
icon - An icon to show instead of one of the standard icon

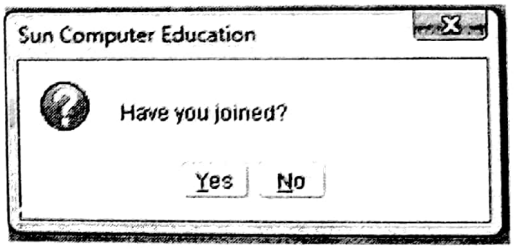


2) Confirmation Dialog - Show a message and get a confirmation (like OK/Cancel)

static int **showConfirmDialog**(Component parent, String message, String title, int optionType, int messageType)

optionType can be One of DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION

Confirmation dialog box is like message box but it returns an integer indicating which option was selected by user.



3) Option Dialog - Show a message and get a user option from a set of options

static int **showOptionDialog**(Component parent, String message, String title, int optionType, int messageType, Icon icon, Object[] options, Object default)

4) Input Dialog - Show a message and get one line of user input

static String **showInputDialog**(Component parent, String message)

It returns string entered by the user in text field provided in dialog box.

File Dialog box

Swing provides **JFileChooser** class to create File dialog box which can be used to open or save file. showOpenDialog method can be used for opening file and **showSaveDialog** method can be used for saving file. This is used to get required file name from user and does not open or save file automatically(u need to write to code for that).
We can use different methods to set up file dialog box initially, for example : **setCurrentDirectory**(File) to specify current folder, **setFileFilter**(FileFilter) to retrieve specific type of files, **setMultiSelctionEnabled**(Boolean) to set multiple selection option.

```
JFileChooser jc = new JFileChooser( );
jc.showOpenDialog(f);       // f is frame
File f1 = jc.getSelectedFile( );
System.out.println(f1.getName( ));
```

Color Chooser dialog box

Swing provides **JColorChooser** class to create color chooser dialog box which will return color selected by user.

Color c= JColorChooser.**showDialog**(f, "Choose Color", Color.BLUE);

- "Choose Color" will be title of color dialog box.
- Initial color in dialog will be Color.BLUE
- Color object will contain 3 components red, green and blue.