UNIT-5: JAVA GUI COMPONENTS

1.Window Fundamentals

The most common windows are those that are derived from Panel and Frame. The window derived from Panel is used by an Applet and from Frame is used to create standard window. The class hierarchy for Panel and Frame is given below.

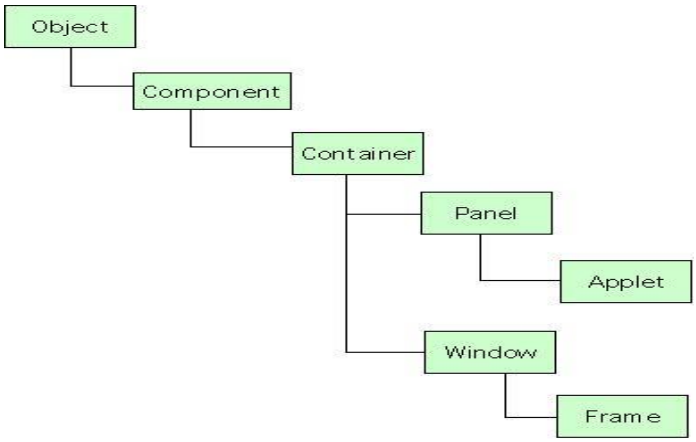

Fig: the class hierarchy for Panel and Frame

1.1 Component

Component is an abstract class that encapsulates all of the attributes of all visual components. All user interface elements that are displayed on the screen and that interact with the user are subclasses of component. It defines hundreds of public methods that are responsible for managing events, such as mouse and keyboard input, positioning and sizing the window, and repainting. The component object is also responsible for remembering the current foreground and background colors and the currently selected text font.

1.2 Container

This is a subclass of Component. It has additional methods that allow other Component object to be nested within it. A container is responsible for laying out i.e. positioning any components that it contains. It does this through the use of various managers.

1.3 Panel

Panel is a concrete subclass of Container. It does not add any new methods, it simply implements Container. Panel is a superclass of Applet. When screen output is directed to an Applet, it is drawn on the surface of a Panel object. A Panel is a window that does not contain a title bar, menu bar or border.

Other components can be added to a Panel object by its add() method which is inherited from Container. Once these components are added, we can position and resize them manually using the setLocation(), setSize() or setBound() methods defined by Component.

1.4 Window

This class creates top-level class window. A top level window is not contained within any other object. It sits directly on the desktop. We cannot create window object directly but can use from the subclass of the Window called Frame.

1.5 Frame

It is the subclass of the window and has a title bar, menu bar, borders and resizing corners. The most common method of creating a frame is by using single argument constructor of the Frame class that contains the single string argument which is the title of the window or frame. Then we can add user interface by constructing and adding different components to the container one by one.

The following program is used to create frame with header and label inside the frame

```
import java.awt.*;
public class AwtFrame extends Frame
{
public static void main(String[] args)
{
Frame frm = new Frame("Java AWT Frame");
Label lbl = new Label("Welcome Dhangadhi Engineering College.", Label.CENTER);
frm.add(lbl);
frm.setSize(400,400);
frm.setVisible(true);
}
}
```

In this program we are constructing a label to display "Welcome to Dhangadhi Engineering College." message on the frame. The center alignment of the label has been defined by the **Label.**CENTER and it is auxillary. The program will work fine with just one argument on the Label() constructor. The frame is initially invisible, so after creating the frame it need to visualize the frame by setVisible(true) method.

Few methods used in above programs

add(lbl): This method has been used to add the label to the frame. Method add() adds a component to its container.

setSize (width, height): This is the method of the Frame class that sets the size of the frame or window. This method takes two arguments width (int), height (int).

setVisible(boolean): This is also a method of the Frame class sets the visibility of the frame. The frame will be invisible if we pass the boolean value false otherwise frame will be visible.

2. Layout Manager

Layout managers arrange GUI components in a container for presentation purposes. We can use the layout managers for basic layout capabilities. All layout managers implement the interface LayoutManager (in package java.awt).

2.1 FlowLayout

This is the default layout manager. FlowLayout implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right. Class FlowLayout allows GUI components to be left aligned, centered (the default) and right aligned. Its constructors are:

FlowLayout(); It creates the default layout, which centers components and leaves five pixels of space between each component.

FlowLayout(int how); It specify how each line is aligned. And its values are:

FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT

FlowLayout(int how, int horz, int vert); It specify the horizontal and vertical space left between components in horz and vert.

//**Simple example of flow layout**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class layOut extends JFrame
{
public static void main( String[] args )
{
JFrame fr1=new JFrame("flowlayout demo");
FlowLayout layout = new FlowLayout();
fr1.setLayout( layout );
JLabel floatLabel = new JLabel( "Being Human" );
fr1.add( floatLabel );
layout.setAlignment( FlowLayout.RIGHT );
layout.layoutContainer( fr1 );
fr1.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
fr1.setSize( 400, 100 );
fr1.setVisible( true );
}
}
```

2.2 BorderLayout

The BorderLayout class implements a common layout style for top-level windows. It has four narrow, fixed-width components as the edges and one large area in the center. The four sides are referred to as north, south, east and west. The middle area is called center.

The components placed in the NORTH and SOUTH regions extend horizontally to the sides of the container and are as tall as the components placed in those regions. The EAST and WEST regions expand vertically between the NORTH and SOUTH regions and are as wide as the components placed in those regions. The component placed in the CENTER region expands to fill all remaining space in the layout.

Its constructors are:

BorderLayout(): It creates a default border layout.

BorderLayout(int horz, int vert): It specify the horizontal and vertical space left between components in horz and vert. The BorderLayout defines the following constants that specify the regions:

BorderLayout.CENTER BorderLayout.SOUTH
BorderLayout.EAST BorderLayout.WEST
BorderLayout.NORTH

While adding component to the applet the following form of add() method is used void add(Component compobj, Object region)

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class demoBorderlayout extends JFrame
{
private JButton button1;
```

```
private JButton button2;
private JButton button3;
private JButton button4;
private JButton button5;
private BorderLayout layout; // borderlayout object
// set up GUI and event handling
public demoBorderlayout ()
{
super( "BorderLayout Demo" );
layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
setLayout( layout ); // set frame layout
button1 = new JButton( "North" );
button2 = new JButton( "South" );
button3 = new JButton( "East" );
button4 = new JButton( "West" );
button5 = new JButton( "Center" );
add( button1, BorderLayout.NORTH ); // add button to north
add( button2, BorderLayout.SOUTH ); // add button to SOUTH
add( button3, BorderLayout.EAST ); // add button to east
add( button4, BorderLayout.WEST ); // add button to west
add( button5, BorderLayout.CENTER ); // add button to center
}
public static void main( String[] args )
{
demoBorderlayout      borderLayoutFrame = new demoBorderlayout ();
borderLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
borderLayoutFrame.setSize( 300, 200 ); // set frame size
borderLayoutFrame.setVisible( true ); // display frame
}
}
```

2.3 GridLayout

GridLayout lays out components in a two-dimensional grid. While instantiating a GridLayout, it define number of rows and columns. Every Component in a GridLayout has the same width and height. Components are added to a GridLayout starting at the top-left cell of the grid and proceeding left to right until the row is full. Then the process continues left to right on the next row of the grid, and so on.

The constructors supported by GridLayout are:

GridLayout(): It creates a single-column grid layout.

GridLayout(int numrows, int numcols): It creates a grid layout with the specified number of rows and columns.

GridLayout(int numrows, int numcols, int horz, int vert): It specifies the horizontal and vertical space left between components in horz and vert.

Example of gridlayout:

```
import java.awt.GridLayout;
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JButton;
```

```
public class GridLayoutFrame extends JFrame
{
private JButton button1;
private JButton button2;
private JButton button3;
private JButton button4;
private Container container; // frame container
private GridLayout gLayout; // first gridlayout

// no-argument constructor
public GridLayoutFrame()
{
super( "GridLayout Demo" );
gLayout = new GridLayout( 2, 3, 5, 5 ); // 2 by 3; gaps of 5

container = getContentPane(); // get content pane
setLayout( gLayout ); // set JFrame layout

button1= new JButton("One");
button2= new JButton("Two");
button3= new JButton("Three");
button4= new JButton("Four");
add( button1 );
add( button2 );
add( button3 );
add( button4 );

} // end GridLayoutFrame constructor
public static void main( String[] args )
{
GridLayoutFrame gridLayoutFrame = new GridLayoutFrame();
gridLayoutFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
gridLayoutFrame.setSize( 300, 200 ); // set frame size
gridLayoutFrame.setVisible( true ); // display frame
} // end main
} // end class GridLayoutFrame
```

3. Introduction To Netbeans IDE

It is a free and open source IDE (Integrated Development Environment). It is easy to use and code in netbeans IDE.

Creating GUI components in NetBeans 8.0.1 using drag and drop

Step 1: Create a New Project

Step 2: Choose Java -> Java Application

Step 3: Set a Project Name "demoProject". We should make sure that we deselect the "Create Main Class" checkbox; leaving this option selected generates a new class as the main entry point for the application, but our main GUI window (created in the next step) will serve that purpose, so

checking this box is not necessary. Click the "Finish" button when we are done.

Step 4: Add a JFrame Form-Now right-click the **demoProject** name and choose New -> JFrame Form (JFrame is the Swing class responsible for the main frame for our application.)

Step 5: Name the GUI Class-Next type **demoProjectGUI** as the class name, and **demoPackage** as the package name. The remainder of the fields should automatically be filled in, as shown above. Click the Finish button when we are done.

When the IDE finishes loading, the right pane will display a design-time, graphical view of the **demoProjectGUI**. It is on this screen that we will visually drag, drop, and manipulate the various Swing components like text fields, labels, password fields,checkbox, radio button, etc.

NetBeans IDE Basics

It is not necessary to learn every feature of the NetBeans IDE before exploring its GUI creation capabilities. In fact, the only features that we really need to understand are **the Palette, the Design Area, the Property Editor, and the Inspector.**

The Palette

The Palette contains all of the components offered by the Swing API(JLabel is a text label, JList is a drop-down list, etc.).

The Design Area

The Design Area is where we will visually construct our GUI. It has two views: **source view, and design view.** Design view is the default, as shown below. We can toggle between views at any time by clicking their respective tabs.

The Property Editor

The Property Editor does what its name implies: it allows us to edit the properties of each component. The Property Editor is intuitive to use; in it we will see a series of rows — one row per property — that we can click and edit without entering the source code directly.

The Inspector

The Inspector provides a graphical representation of our application's components. We will use the Inspector only once, to change a few variable names to something other than their defaults.

4. Event Delegation Model, Event Source and Handler, Event Categories, Listeners Interfaces, Adaptor Classes

4.1 Event:

Change in the state of an object is known as event i.e. event describes the change in state of source. Events are generated as result of user interaction with the graphical user interface components. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page are the activities that causes an event to happen. Events may also occur that are not directly caused by interactions with a user interface. For example, an event may be generated when a timer expires, software or hardware failure occurs, or an operation is completed. Most of the events relating to the GUI for a program are represented by classes defined in the package java.awt.event. This package also defines the listener interfaces for the various kinds of events that it defines. The package javax.swing.event defines classes for events that are specific to Swing components.

4.2 Types of Event (Event Categories):

The events can be broadly classified into two categories:

- **Foreground Events** - Those events which require the direct interaction of user are known as foreground events. They are generated as consequences of a person interacting with the graphical components in Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page etc.

- **Background Events** - Those events that do not require the interaction of end user are known as background events. For example: Operating system interrupts, hardware or software failure, if timer expires, an operation completion, etc.

The table below shows most of the important event classes and their description:

Event Class	**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 losses keyboard focus.
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 the keyboard.
MouseEvent	Generated when the mouse is dragged, moved, clicked, pressed, or released; also generated when the mouse enters or exits a component.
MouseWheelEvent	Generated when the mouse wheel is moved.

4.3 The Event Delegation Model

The modern approach that is used to handle an event is delegation event model. It is a standard and consistent mechanism to generate and process event. It implements the following concept:

Source - The source is an object on which event occurs. Source is responsible for providing information of the occurred event to its handler. A source generates an event and sends it to one or more listeners.

Listener (Event Handler) - It is also known as event handler. Listener is an object that is responsible for generating response to an event. Listener waits until it receives an event. Once the event is received, the listeners process the event and then returns.

The advantage of this design is that the application logic that processes events is clearly separated from the user interface logic that generates those events. This is an efficient way of handling the event because the event notifications are sent only to those listeners that want to receive them.

4.4 Event source

A source is an object that generates an event. This occurs when the internal state of that object changes. Sources may generate more than one type of event. A source must register listeners in order for the listeners to receive notifications about a specific type of event. Each type of event has its own registration method.

The general form of registration method is:

public void addTypeListener(TypeListener el)

Here type is the name of event and el is the reference to the event listener. For example, the method that registers a keyboard event listener is called addKeyListener(KyeListener el). The method that register a mouse motion listener is called addMouseMotionListener(MouseMotionListener el). When an event occurs,

all registered listeners are notified and receive a copy of the event object. This is known as multicasting the event.

The table below shows most of the important event sources and their description:

Event Sources	Description
Button	Generates action events when the button is pressed.
Checkbox	Generates item events when the checkbox is selected or deselected.
Choice	Generates item events when the choice is changed.
Menu Item	Generates action events when the menu is selected; generates item events when a checkable menu item is selected or deselected.
List	Generates action events when an item is double-clicked; generates item events when an item is selected or deselected.
Scrollbar	Generates adjustment events when the scroll bar is manipulated.
Text Components	Generates text events when the user enters the character.
Window	Generates window events when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit.

4.5 Event Listener Interfaces

Listeners are created by implementing one or more of the interfaces defined by the java.awt.event package. To react to an event, we implement appropriate event listener interfaces. A listener is an object that is notified when an event occurs. It has two requirements. First, it must have been registered with one or more sources to receive notifications about specific types of events. Second, it must implement methods to receive and process these notifications. We use addTypeListener() method to register the event listener with the source. Similarly, we can also unregister a listener by using removeTypeListener() method. The table below shows most of the important event listener and their description:

Interface	Description
ActionListener	Declares one method to receive action events. void actionPerformed(ActionEvent e)
AdjustmentListener	Declares one method when a scrollbar is manipulated. void adjustmentValueChanged (AdjustmentEvent e)
ComponentListener	Declares four methods to recognize when a component is hidden, moved, resized, or shown. void componentResized(ComponentEvent e) void componentMoved(ComponentEvent e) void componentShown(ComponentEvent e) void componentHidden(ComponentEvent e)
ContainerListener	Declares two methods to recognize when a component is added to or removed from a container. void componentAdded(ContainerEvent e) void

	componentRemoved(ContainerEvent e)
FocusListener	Defines two methods to recognize when a component gains or loses keyboard focus. void focusGained(FocusEvent e) void focusLost(FocusEvent e)
ItemListener	Defines one method to recognize when a check box or list item is clicked, when a choice selection is made, or when a checkable menu item is selected or deselected. void itemStateChanged(ItemEvent e)
KeyListener	Defines three method to recognize when a key is pressed, released, or typed. void keyPressed(KeyEvent e) void keyReleased(KeyEvent e) void keyTyped(KeyEvent e)
MouseListener	Defines five method to recognize when the mouse is clicked, enters a component, exits a component, is pressed, or is released. void mouseClicked(MouseEvent e) void mouseEntered(MouseEvent e) void mouseExited(MouseEvent e) void mousePressed(MouseEvent e) void mouseReleased(MouseEvent e)
MouseMotionListener	Defines two method to recognize when the mouse is dragged or moved. void mouseDragged(MouseEvent e) void mouseMoved(MouseEvent e)
MouseWheelListener	Defines one method to recognize when the mouse wheel is moved. void mouseWheelMoved(MouseWheelEvent e)
TextListener	Defines one method to recognize when a text value changes. void textChanged(TextEvent e)
WindowFocusListener	Defines two method to recognize when a window gains or loses input focus. void windowGainedFocus(WindowEvent e) void windowLostFocus(WindowEvent e)
WindowListener	Defines seven method to recognize when a window is activated, closed, deactivated, deiconified, iconified, opened, or quit. void windowActivated(WindowEvent e) void windowDeactivated(WindowEvent e) void windowClosed(WindowEvent e) void windowClosing(WindowEvent e) void windowOpened(WindowEvent e) void windowIconified(WindowEvent e) void windowDeiconified(WindowEvent e)

Example of event handling

Example1:

```
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
public class EventExample extends JFrame implements ActionListener
{
public static void main(String args[])
{
EventExample e1=new EventExample();
JFrame f1=new JFrame("Event handling");
JButton b1=new JButton("Click me");
f1.add(b1);
b1.addActionListener(e1);
f1.setVisible(true);
f1.setSize(400,450);
}
public void actionPerformed(ActionEvent e)
{
JOptionPane.showMessageDialog(null, "The button is clicked");
}
}
```

Example 2:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class EventExample extends JFrame
{
public static void main(String args[])
{
JFrame f1=new JFrame("Event handling");
JButton b1=new JButton("Click me");
f1.add(b1);
f1.setVisible(true);
f1.setSize(400,450);
b1.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent e)
{
String firstNumber=JOptionPane.showInputDialog( "Enter first integer" );
String secondNumber=JOptionPane.showInputDialog( "Enter second integer" );
// convert String inputs to int values for use in a calculation
int number1 = Integer.parseInt( firstNumber );
int number2 = Integer.parseInt( secondNumber );
int sum = number1 + number2; // add numbers
// display result in a JOptionPane message dialog
JOptionPane.showMessageDialog( null, "The sum is " + sum, "Sum of Two Integers",
JOptionPane.PLAIN_MESSAGE );
}
}
);
}
}
```

4.6 Adaptor Classes

Java provides a special feature, called an adapter class that can simplify the creation of event handlers in certain situations. An adapter class provides an empty implementation of all methods in an event listener interface i.e. this class itself write definition for methods which are present in particular event listener interface. Adapter classes are useful when we want to receive and process only some of the events that are handled by a particular event listener interface. We can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events which we want.

For example: Suppose we want to use MouseClicked Event or method from MouseListener, if we do not use adapter class then unnecessarily we have to define all other methods from MouseListener such as MouseReleased, MousePressed ec. But if we use adapter class then we can only define MouseClicked method.

The different adapter classes are;

- ComponentAdapter class
- ContainerAdapter class
- FocusAdapter class
- KeyAdapter class
- MouseAdapter class
- MouseMotionAdapter class
- WindowAdapter class

Example:

```
import java.awt.*;
import java.awt.event.*;
public class WindowAdapterEx extends Frame
{
public WindowAdapterEx()
{
WindowAdapterClose clsme=new WindowAdapterClose();
addWindowListener(clsme);
setTitle("WindowAdapter  frame closing");
setSize(400,400);
setVisible(true);
}
public static void main(String args[])
{
new WindowAdapterEx();
}
public class WindowAdapterClose extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
}
}
}
```

Advantages Of Adapter Classes

- Assists unrelated classes to work together.
- Provides a way to use classes in multiple ways.

- Increase the transparency of classes.
- Makes a class highly reusable.

5. Swing Libraries, Model View Controller Design Pattern, Different Layout and All Swing Components

5.1 Swing Libraries

Swing is described as a set of customizable graphical components whose look-and-feel (L&F) can be dictated at runtime. Swing is the next-generation GUI toolkit that Sun Microsystems created to enable enterprise development in Java. By enterprise development, we mean that programmers can use Swing to create large-scale Java applications with a wide array of powerful components. In addition, we can easily extend or modify these components to control their appearance and behavior.

Swing is a package (library) in java API that contains everything we need when we are doing GUI. It is actually part of a larger family of Java products known as the Java Foundation Classes (JFC). The main package of swing is javax.swing.

5.2 The Model/View/Controller (MVC) Design Pattern

Swing uses the model-view-controller architecture (MVC) as the fundamental design behind each of its components. MVC breaks GUI components into three elements. Each of these elements plays an important role in how the component behaves. It is a design pattern for the architecture of web applications whose purpose is to achieve a clean separation between three components of most any web application.

- **Model**

The model represents the state data for each component. Model represents knowledge. A model stores data that is retrieved to the controller and displayed in the view. Whenever there is change in the data it is updated by the controller.

There are different models for different types of components. For example, the model of a menu may contain a list of the menu items the user can select from. This information remains the same no matter how the component is painted on the screen; model data is always independent of the component's visual representation.

- **View**

The view refers to how we see the component on the screen. It is a visual representation of its model. A view is attached to its model and gets the data necessary for the presentation from the model by asking questions. It may also update the model by sending appropriate messages. A view requests information from the model that it uses to generate an output representation to the user.

As an example we can look at an application window on two different GUI platforms. Almost all window frames have a title bar spanning the top of the window. However, the title bar may have a close box on the left side (like the Mac OS platform), or it may have the close box on the right side (as in the Windows platform). These are examples of different types of views for the same window object.

- **Controller**

The controller is the portion of the user interface that dictates how the component interacts with events. The controller decides how each component reacts to the event. It is a link between the user and the system. It provides the user with input by arranging relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. It receives such user output, translates it into the appropriate messages and passes these messages on to one or more of the views.

A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).

5.3 Different Layouts and Swing Components

Layout means the arrangement of components within the container i.e. placing the components at a particular position within the container. The task of layouting the control is done automatically by the layout manager. The different layout managers are FlowLayout, BorderLayout, Grid Layout, etc.

Swing Components

GUIs are built from GUI components. These are also called controls or widgets. A GUI component is an object with which the user interacts via the mouse, the keyboard or another form of input, such as voice recognition. Swing components are the basic building blocks of an application. The Swing GUI components are defined in the javax.swing package. Swing has a wide range of various components, including buttons, check boxes, labels, panels, sliders, etc.

Component	Description
`JLabel`	Displays uneditable text and/or icons.
`JTextField`	Typically receives input from the user.
`JButton`	Triggers an event when clicked with the mouse.
`JCheckBox`	Specifies an option that can be selected or not selected.
`JComboBox`	A drop-down list of items from which the user can make a selection.
`JList`	A list of items from which the user can make a selection by clicking on any one of them. Multiple elements can be selected.
`JPanel`	An area in which components can be placed and organized.

fig. Some basic swing GUI components

```
import javax.swing.*;
public class SwingComponent
{
public static void main(String args[])
{
JFrame  fr=new JFrame("Swingcomponentdemo");
JPanel panel=new JPanel();
fr.add(panel);
JTextField txt=new JTextField(20);
panel.add(txt);
JButton btn=new JButton("Click");
panel.add(btn);
JCheckBox chk=new JCheckBox();
panel.add(chk);
JComboBox  combo= new JComboBox();
panel.add(combo);
fr.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
fr.setSize( 400, 100 );
fr.setVisible( true );
}
}
```