

CHAPTER 8

GUI and Event handler programming

References:

Java-Tutorials/tutorial-2015/uiswing/index.html

Objectives

- **AWT vs Swing**
- GUI Basics design
- Top-level container
- Layout Manager
- Common Control
- Event Listener
- Dialogbox
- Advanced Control
- Text component
- Choice component
- Menu
- Tabbed pane
- Scroll pane
- Dialog box
- Jlist
- Jtable
-
- Case study

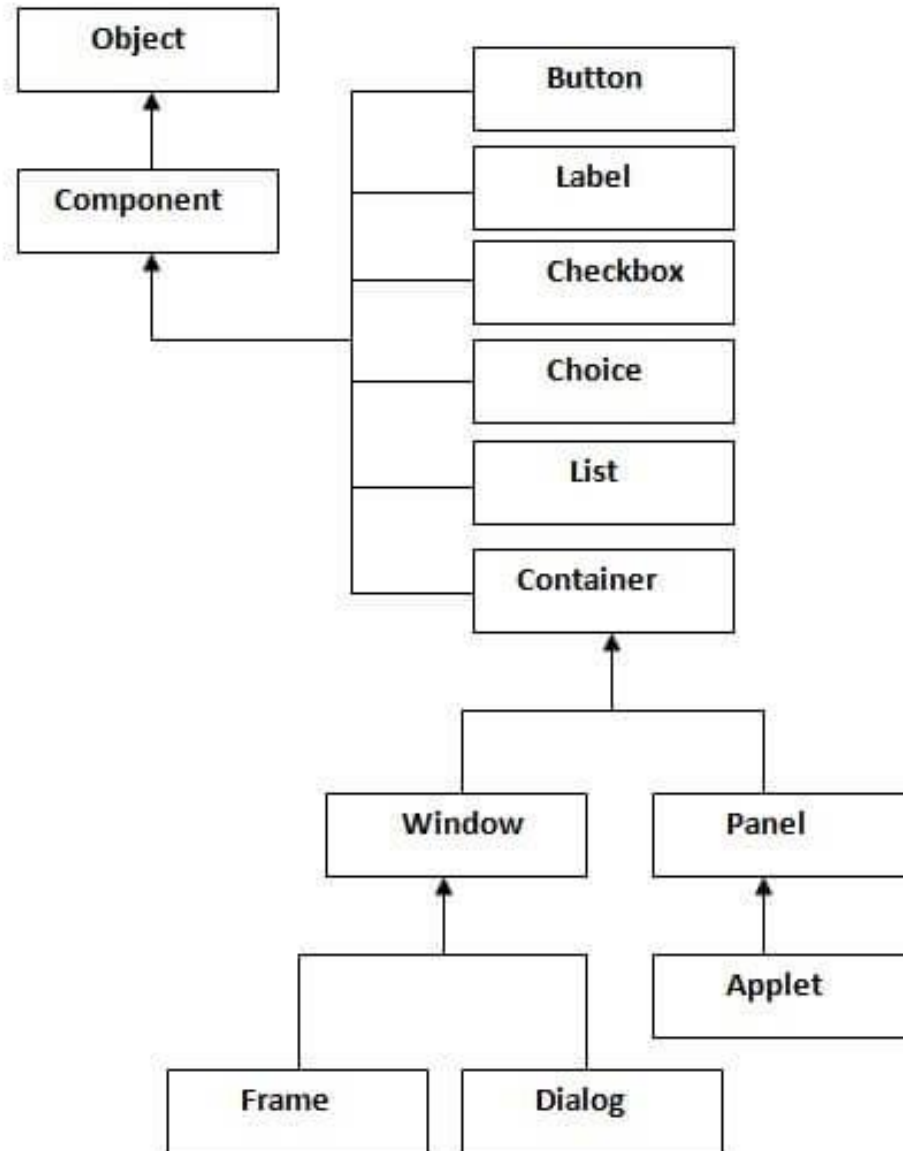
JFC (Java Foundation Classes)

- The Java Foundation Classes, or JFC, is a loose collection of standard Java APIs for client-side graphics, graphical user interfaces (GUIs), and related programming tasks.
- **AWT** (Abstract Windows Toolkit)
- **swing**
- **Accessibility API**: Java Accessibility API is part of Java Accessibility Utilities, which is a set of utility classes that help assistive technologies provide access to GUI toolkits that implement the Java Accessibility API.
- **2D API**: Java 2D is the name for the state-of-the-art two-dimensional graphics API introduced in Java 1.2. Java 2D is built upon the AWT
- **Data transfer** (Drag and Drop): support for data transfer using the drag-and-drop metaphor.

AWT

- **Java AWT** (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is **heavyweight** i.e. its components are using the resources of OS.
- The java.awt package provides classes for AWT api such as TextField, Label, Text Area, RadioButton, CheckBox, Choice, List etc.

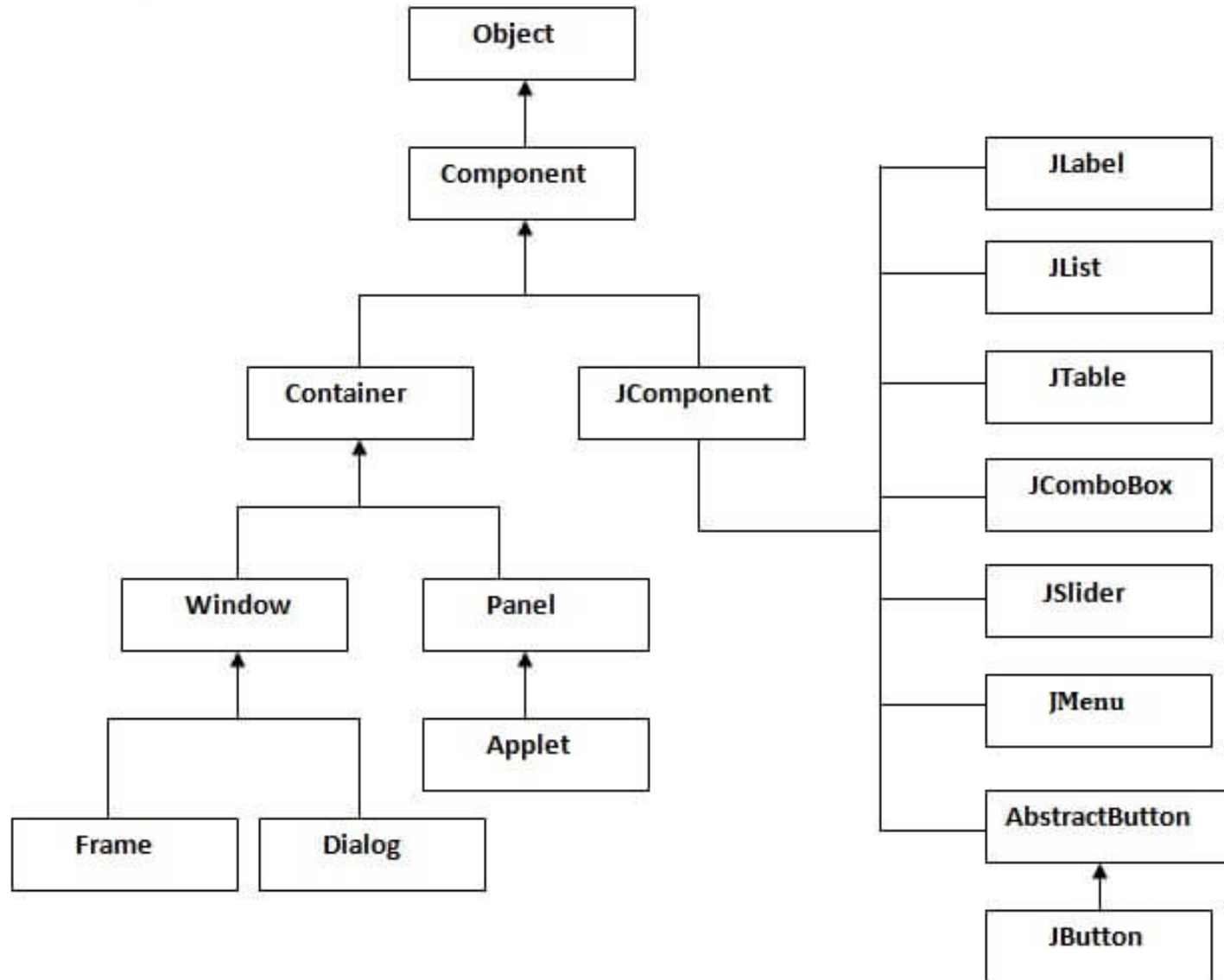
Java AWT Hierarchy



SWING

- **Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Hierarchy of Java Swing classes

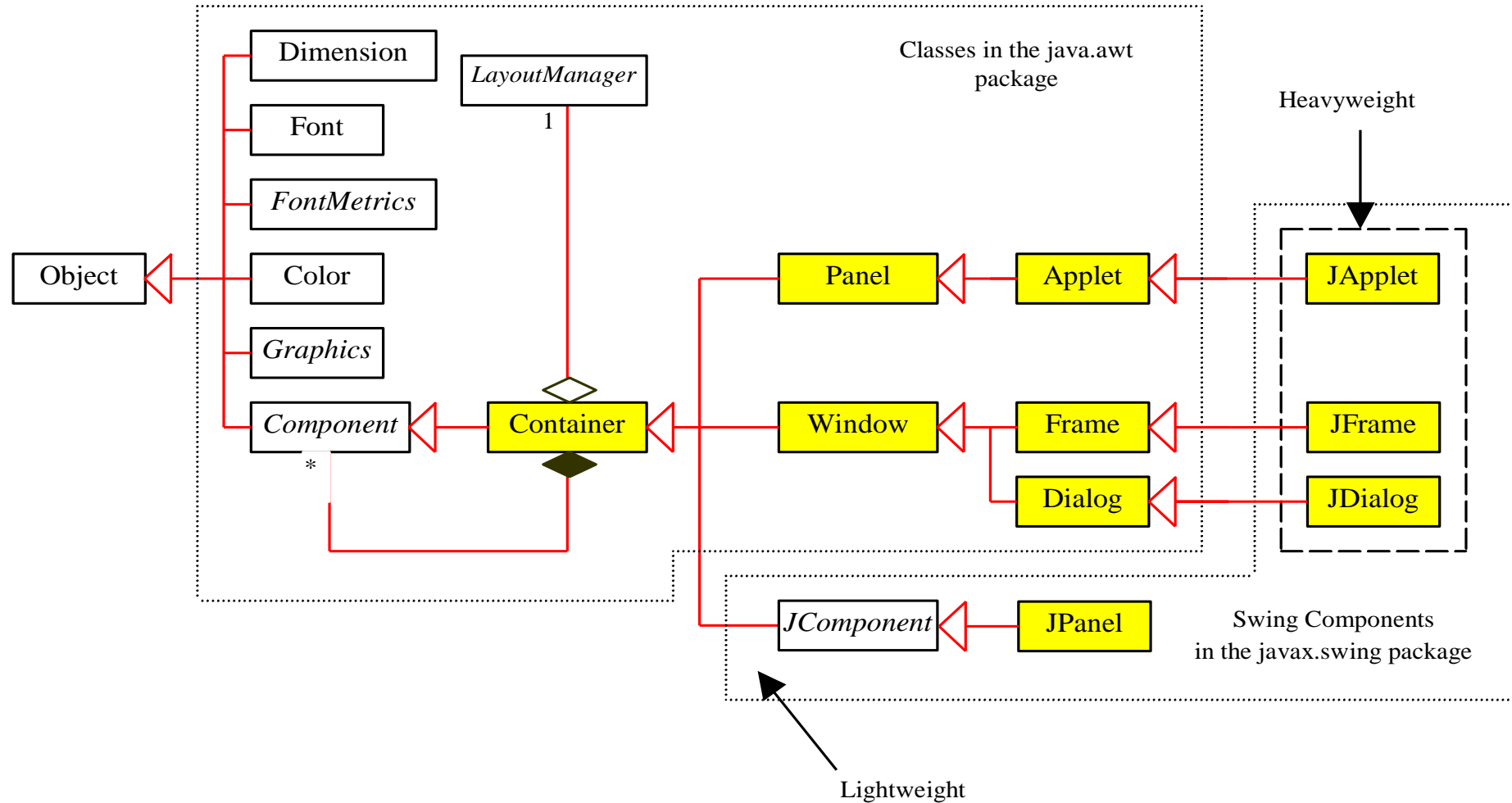


Java GUI

■ **Creating Graphical User Interfaces**

- Container
- Component
- Layout manager
- Graphic và drawing capabilities
- Font
- Event

Container classes

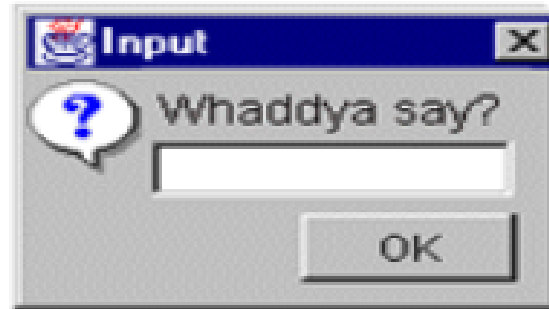


Container

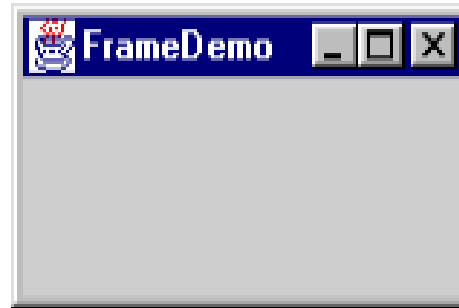
- JFrame, Jframe
- JPanel
- JDialogs
- ScrollPanels
- Applet: Web Applet
- JWindow

Top-level component

- Swing provides three generally useful top-level container classes: [JFrame](#), [JDialog](#), and [JApplet](#).



Dialog



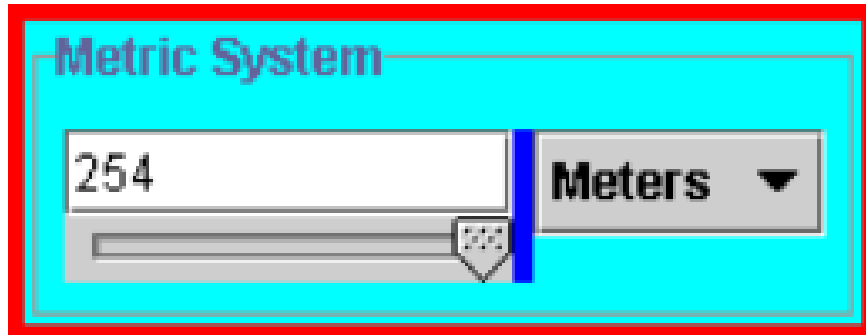
Frame



Applet

Intermediate containers

- Swing provides several general-purpose intermediate containers: scroll pane, split pane, tabbed pane....



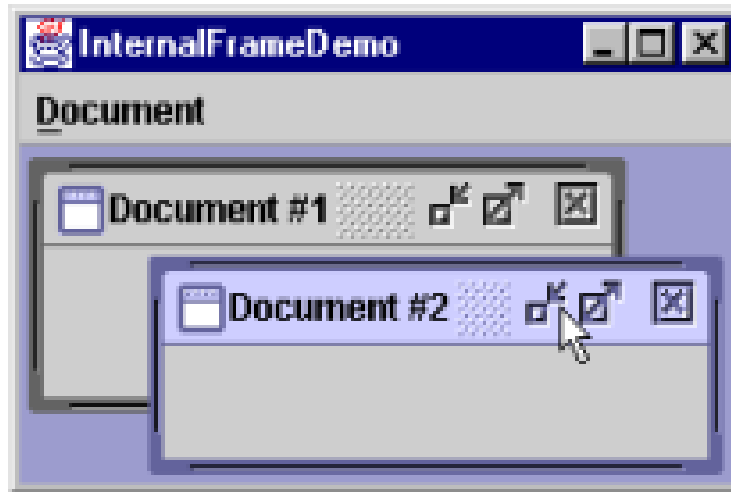
Panel



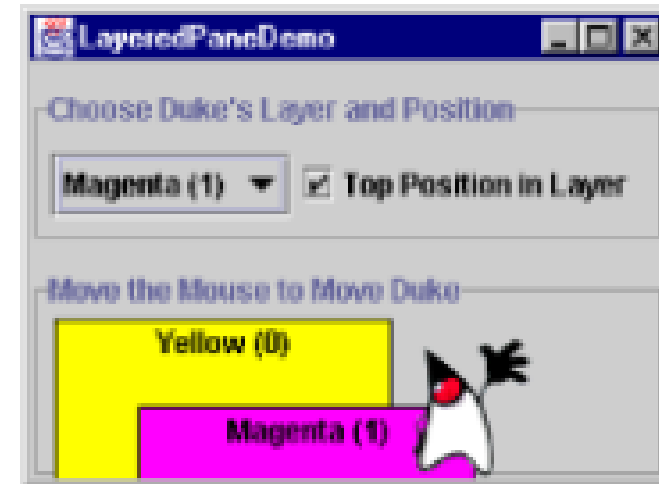
Scroll pane

Special-Purpose Containers

- The rest of the Swing intermediate containers are more specialized:



Internal Frame



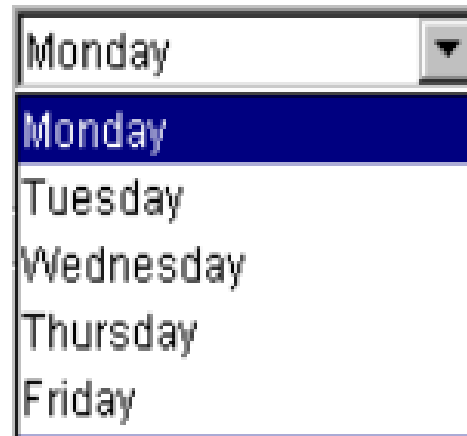
Layered pane

Basic component controls

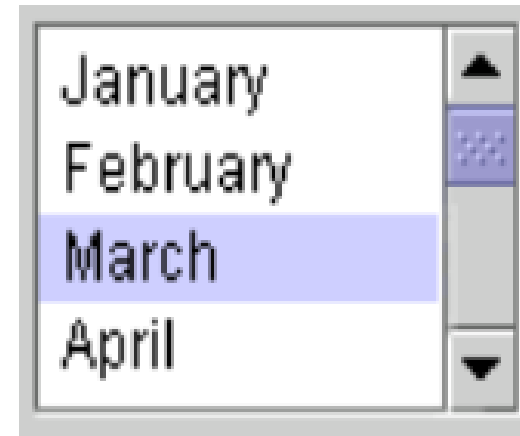
- Receive data from the users



• **Buttons**



Combo Box



List

Basic component controls



Menu



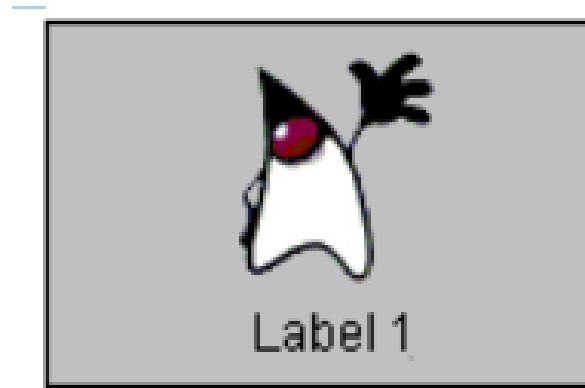
Text fields



Slide

The components for display information

- Use for display information
- not allow to edit information



Label



Tool tip



Progress Bar

JFileChooser and JColorChooser



Color Chooser



File Chooser

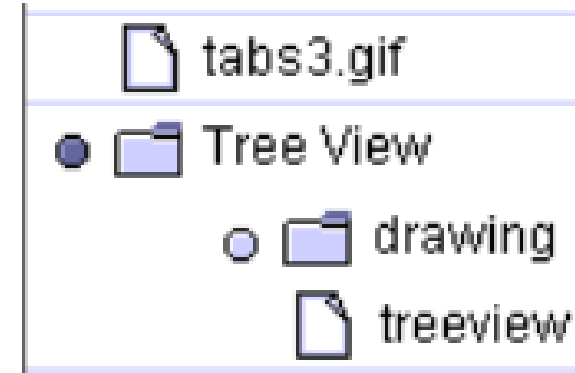
Display information with format

First Na...	Last Name
Mark	Andrews
Tom	Ball
Alan	Chung
Jeff	Dinkins

Table

Verify that the RJ45 cable is connected to the WAN plug on the back of the Pipeline unit.

Text



Tree

Jframe

```
javax.swing.JFrame
```

```
+JFrame()
```

```
+JFrame(title: String)
```

```
+void setSize(width: int, height: int)
```

```
+void setLocation(x: int, y: int)
```

```
+void setVisible(visible: boolean)
```

```
+void setDefaultCloseOperation(mode: int)
```

```
+void setLocationRelativeTo (c: Component)
```

```
1. public class HelloSwing{
2.     public static void main(String[] args) {
3.         JFrame win = new JFrame("Demo");
4.         win.setDefaultCloseOperation(
5.             JFrame.EXIT_ON_CLOSE);
6.         win.setSize(300,200);
7.         win.setLocationRelativeTo(null);
8.         win.setResizable(false);
9.         win.add(new JLabel("Hello world!"));
10.        win.setVisible(true);
11.    }
```

```
1. public class DemoJFrame extends JFrame {  
2.     public      DemoJFrame ()      {  
3.         setTitle("Demo JFrame");  
4.         setSize(300,200);  
5.         setDefaultCloseOperation(EXIT_ON_CLOSE);  
6.         setLocationRelativeTo(null);  
7.         setResizable(false);  
8.     }  
9.     public static void main(String[] args) {  
10.         new DemoJFrame ().setVisible(true);  
11.     }  
12. }
```

JFrame with NetBeans

- click package -> click right mouse -> select **New** -> and **JFrame Form** -> enter name in **Class Name** -> select **Finish**

LoginForm.java X

Source Design History

To change layout manager of a container use Set Layout submenu from its context menu

Khu vực thiết kế giao diện đồ họa

Thành phần đồ họa dùng để thêm vào màn hình

Thuộc tính của một thành phần trong giao diện

Vi dụ thiết lập tiêu đề cho màn hình là "Login Form"

Palette

- Swing Containers**
 - Panel
 - Tabbed Pane
 - Split Pane
 - ScrollPane
 - ToolBar
 - Desktop Pane
 - Internal Frame
 - Layered Pane
- Swing Controls**
 - Label
 - Button
 - ToggleButton
 - CheckBox
 - RadioButton
 - Button Group
 - ComboBox
 - List
 - TextField
 - TextArea
 - ScrollBar
 - Slider
 - ProgressBar
 - Formatted Field
 - Password Field
 - Spinner
 - Separator
 - Text Pane
 - Editor Pane
 - Tree
 - Table
- Swing Menus**
- Swing Windows**

[JFrame] - Properties

Properties Binding Events Code

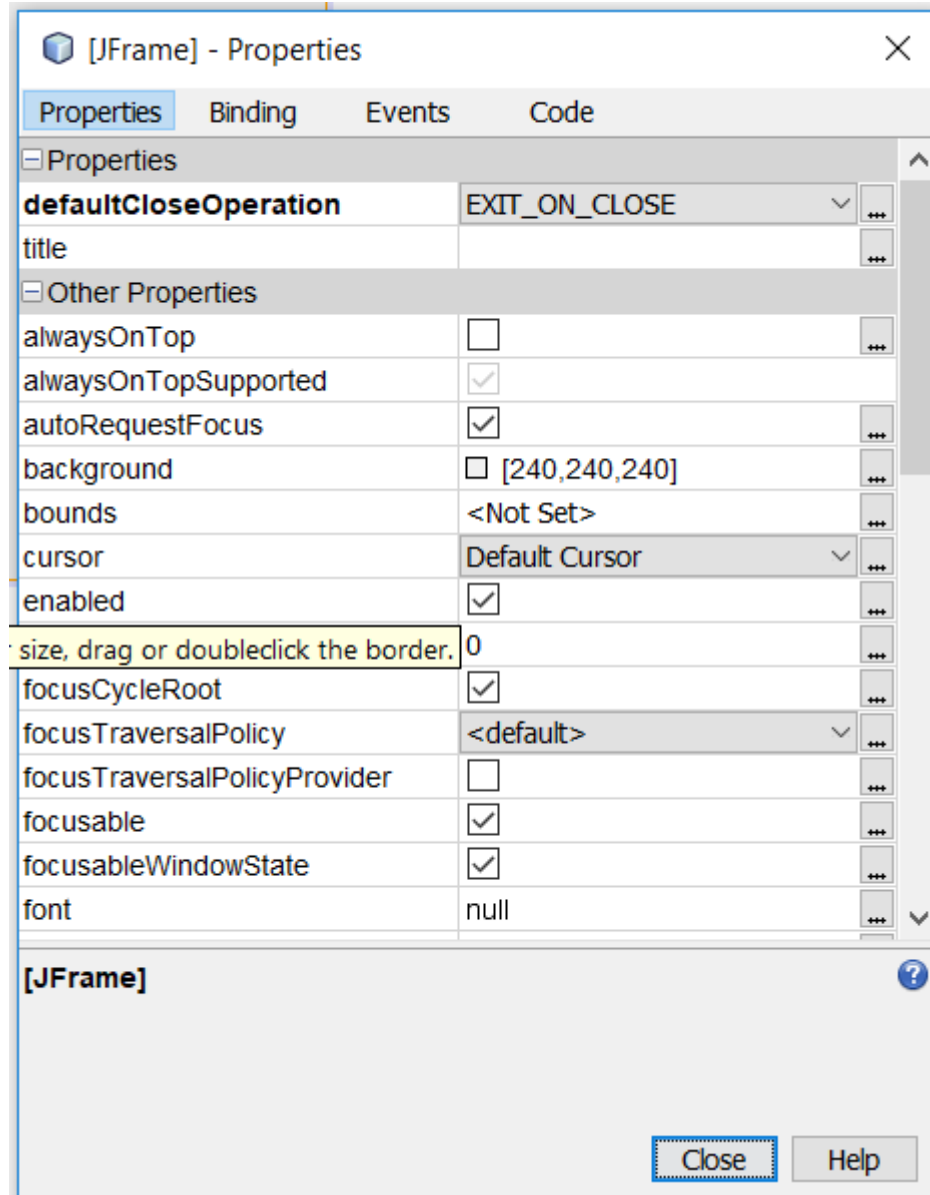
Properties

defaultCloseOperation	EXIT_ON_CLOSE
title	Login Form

Other Properties

background	[240,240,240]
bounds	<Not Set>
cursor	Default Cursor
enabled	<input checked="" type="checkbox"/>

Jframe's properties



JDialog - demo

```
1. public class DemoJDialog extends JDialog{
2.     public DemoJDialog() {
3.         setTitle("Demo JDialog");
4.         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
5.         setSize(300,200);
6.         setResizable(false);
7.     }
8.     public static void main(String[] args) {
9.         new DemoJDialog().setVisible(true);
10.    }
11. }
```

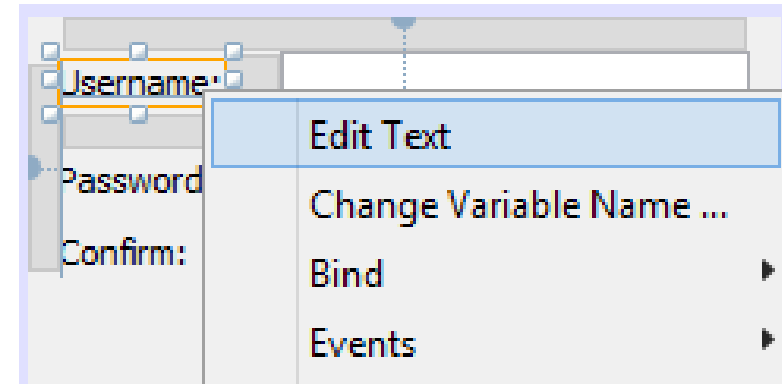
JPanel

- The JPanel is a simplest container class. It provides space in which an application can attach any other component. It inherits the JComponents class.
- It doesn't have title bar.
- Constructors:
 - JPanel()
 - JPanel(LayoutManager lm)

```
1. public class DemoJPanel extends JFrame{
2.     public DemoJPanel() {
3.         setTitle("Demo JPanel");
4.         setSize(300,400);
5.         setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
6.         JPanel p=new JPanel();
7.         p.setBorder(BorderFactory.
            createTitledBorder("Book information"));
8.         p.add(new JLabel("Example!"));
9.         p.add(new JButton("for component"));
10.    this.add(p);
11. }
12. public static void main(String[] args) {
13.     new DemoJPanel().setVisible(true);
14. }
15. }
```

JLabel (1)

- The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.
- **setText(String label), getText()**
- Font getFont()
- void setFont(Font font)
- Gets or sets font



JLabel (2)

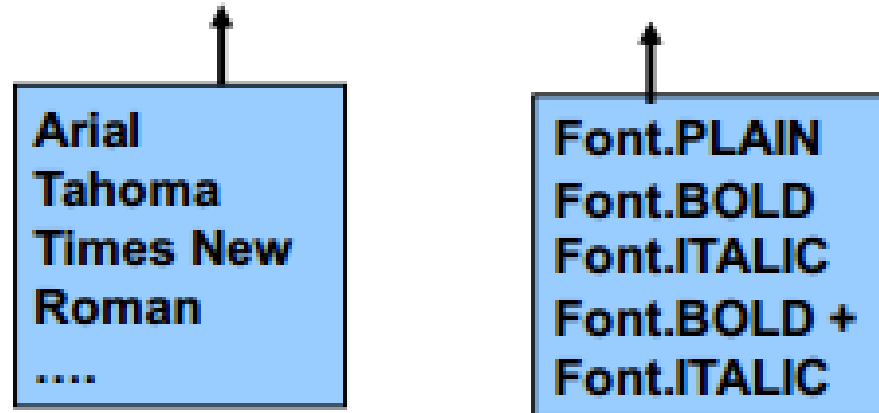
- Constructors:
- **JLabel()**: Creates an empty label
- **JLabel (String labeltext)**: Creates a label with a given text
- **JLabel (String labeltext, int alignment)**: Creates a label with given alignment where alignment can be LEFT, RIGHT, CENTER, LEADING or TRAILING.
- **JLabel (Icon img)**: Only Icon will be used for label
- **JLabel (String str, Icon img, int align)**

```
1. public class DemoJLabel extends JFrame{
2.     public DemoJLabel() {
3.         setLayout(new GridLayout(1,3,5,5));
4.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
5.         Icon icon = new ImageIcon("pic_8.jpg");
6.         JLabel lb1 = new JLabel("Label 1a text");
7.         JLabel lb2 = new JLabel(icon);
8.         JLabel lb3 = new JLabel("icon and
           text",icon, JLabel.CENTER);
9.         lb3.setVerticalTextPosition(JLabel.BOTTOM);
```

```
1.    lb3.setHorizontalTextPosition(  
    JLabel.CENTER) ;  
2.    add(lb1) ;  
3.    add(lb2) ;  
4.    add(lb3) ;  
5.    pack() ;  
6.    setLocationRelativeTo(null) ;  
7. }  
8. public static void main(String[] args) {  
9.    new DemoJLabel().setVisible(true) ;  
10. }  
11. }
```

fonts for text

- To draw characters in a font, you must first create an object of the class Font
- Constructor:
- `Font(String font_name, int font_style, int font_size)`

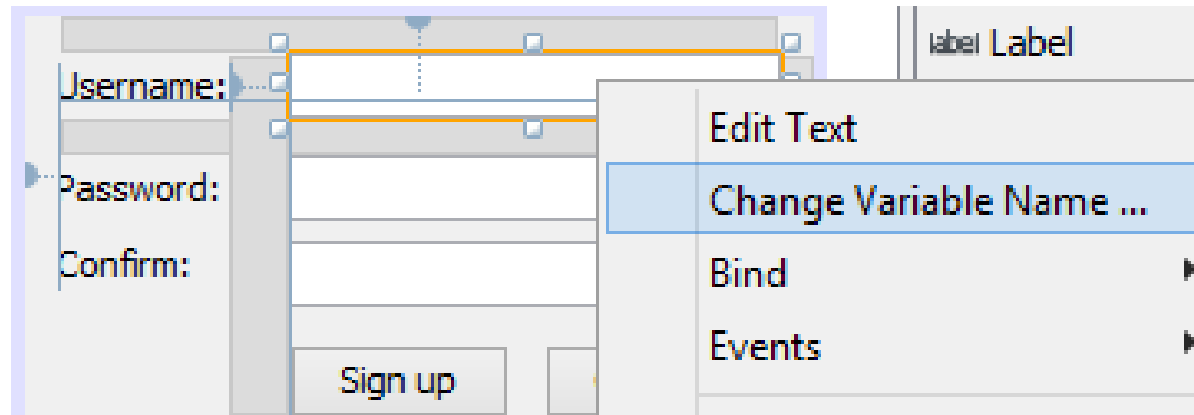
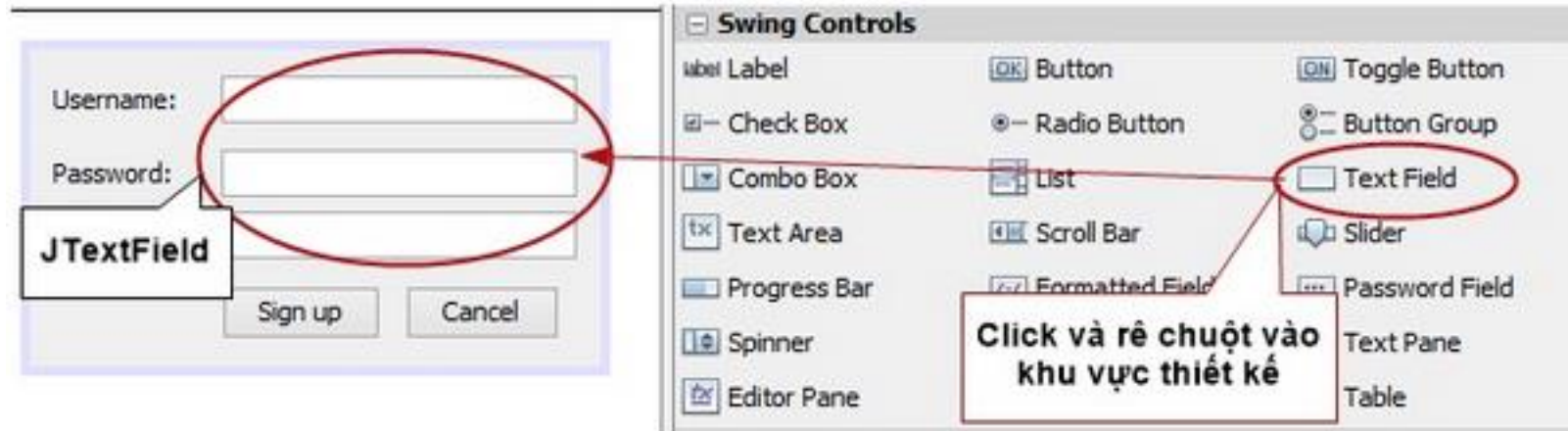



```
1. public class DemoJLabelwithColor extends JFrame{
2.     public DemoJLabelwithColor() {
3.         setLayout(new GridLayout(1, 2, 5, 5));
4.         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
5.         setSize(400, 200);
6.         JLabel lb;
7.         lb = createJLabel("Example 1", Color.red,
           Color.green);
8.         Font font=new Font("Arial",Font.BOLD,40);
9.         lb.setFont(font);
10.        add(lb);
11.        lb = createJLabel("Example 1", Color.blue,
           Color.yellow);
12.        lb.setFont(font);
13.        add(lb);
14.        setLocationRelativeTo(null);}
```

```
1. private JLabel createJLabel(String text, Color  
   textColor,Color backgroundColor) {  
2.     JLabel lb = new JLabel(text);  
3.     lb.setHorizontalAlignment(JLabel.CENTER);  
4.     lb.setForeground(textColor);  
5.     lb.setOpaque(true);  
6.     lb.setBackground(backgroundColor);  
7.     return lb;  
8. }  
9. public static void main(String[] args) {  
10.    new DemoJLabelwithColor().setVisible(true);  
11. }  
12. }
```

JTextField

- JTextField allows the editing of a single line text.



JTextField - Constructors

- `JTextField()`
 - creates an empty textfield with 1 columns
- `TextField(String s)`
 - creates a new textfield with the given string
- `JTextField(int cols)`
 - creates an empty textfield with given number of columns
- `JTextField(String text, int cols)`
 - creates a new textfield with given string and given number of columns
- Example:
 - `JTextField mmText = new JTextField(10);`
 - `JTextField txtName = new JTextField("To Lan", 20);`

JTextField - Methods

- `String getText()`
- `void setText(String t)`
 - gets or sets the text in text field
- `void setFont(Font font)`
 - sets the font for this text field
- `void setEditable(boolean b)`
 - determines whether the user can edit the content

```
1. public class DemoJTextField extends JFrame {
2.     JTextField name;
3.     JPasswordField pass;
4.     public DemoJTextField() {
5.         super("Example for input");
6.         JPanel p =new JPanel();
7.         p.add(new JLabel("username: "));
8.         p.add(new JTextField(15));
9.         p.add(new JLabel("password:"));
10.        p.add(new JPasswordField(15));
11.        add(p);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12.        setSize(250, 100);
13.    }
14.    public static void main(String[] args) {
15.        new DemoJTextField().setVisible(true);
16.    }
17. }
```

JPasswordField

- It is a text component specialized for password entry (*).
- The default character displayed for whatever you type is '*'. If you want to change that, you can use `setEchoChar(char c)` method.
- `JPasswordField.setEchoChar(char c)`

JTextArea

- The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text.
- `String text = getText();`
- `setText(String value);`
- `setEditable(boolean editable)`

JButton

- constructor:
- **JButton()**: It creates a button with no text and icon.
- **JButton(Icon icon)**: It creates a button with the specified icon object.
- **JButton(String text)**: It creates a button with the specified text.
- **JButton(String text, Icon icon)**: It creates a button with the specified text and the specified icon object.

Methods of AbstractButton class

- **public void setText(String s):** to set specified text on button
- **public String getText():** to return the text of the button.
- **public void setEnabled(boolean b):** to enable or disable the button.
- **public void setIcon(Icon b):** to set the specified Icon on the button.
- **public Icon getIcon():** to get the Icon of the button.
- **public void setMnemonic(int a):** to set the mnemonic on the button.
- **public void addActionListener(ActionListener a):** to add the action listener to this object.

```
1. public class DemoJButton extends JFrame{
2.     JButton b1,b2;
3.     public DemoJButton() {
4.         super("Example for Button");
5.         b1= new JButton("Stop", new ImageIcon("stop.png"));
6.         b2= new JButton("Go", new ImageIcon("go.png"));
7.         JPanel p = new JPanel();
8.         p.add(b1);
9.         p.add(b2);
10.        add(p);
11.        setSize(300,200);
12.    }
13.    public static void main(String[] args) {
14.        new DemoJButton().setVisible(true);
15.    }
16. }
```

Java Event Handling

- Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The `java.awt.event` package provides many event classes and Listener interfaces for event handling.

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Two Kinds of Apps

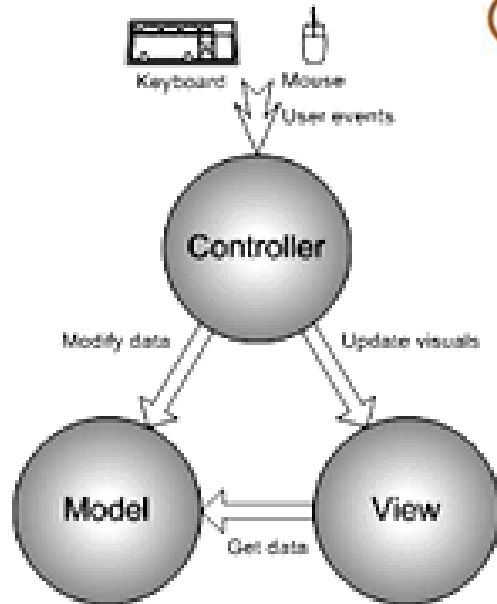
Console App.
Compute-centric
Apps

```
C:\WINDOWS\system32\cmd.exe

G:\GiangDay\FU\CoreJava\Chapter01\build\classes>java myPackage.Ex01_8.3

G:\GiangDay\FU\CoreJava\Chapter01\build\classes>pause
Press any key to continue . . .
```

Event-based App.
User-centric
Apps(GUI)



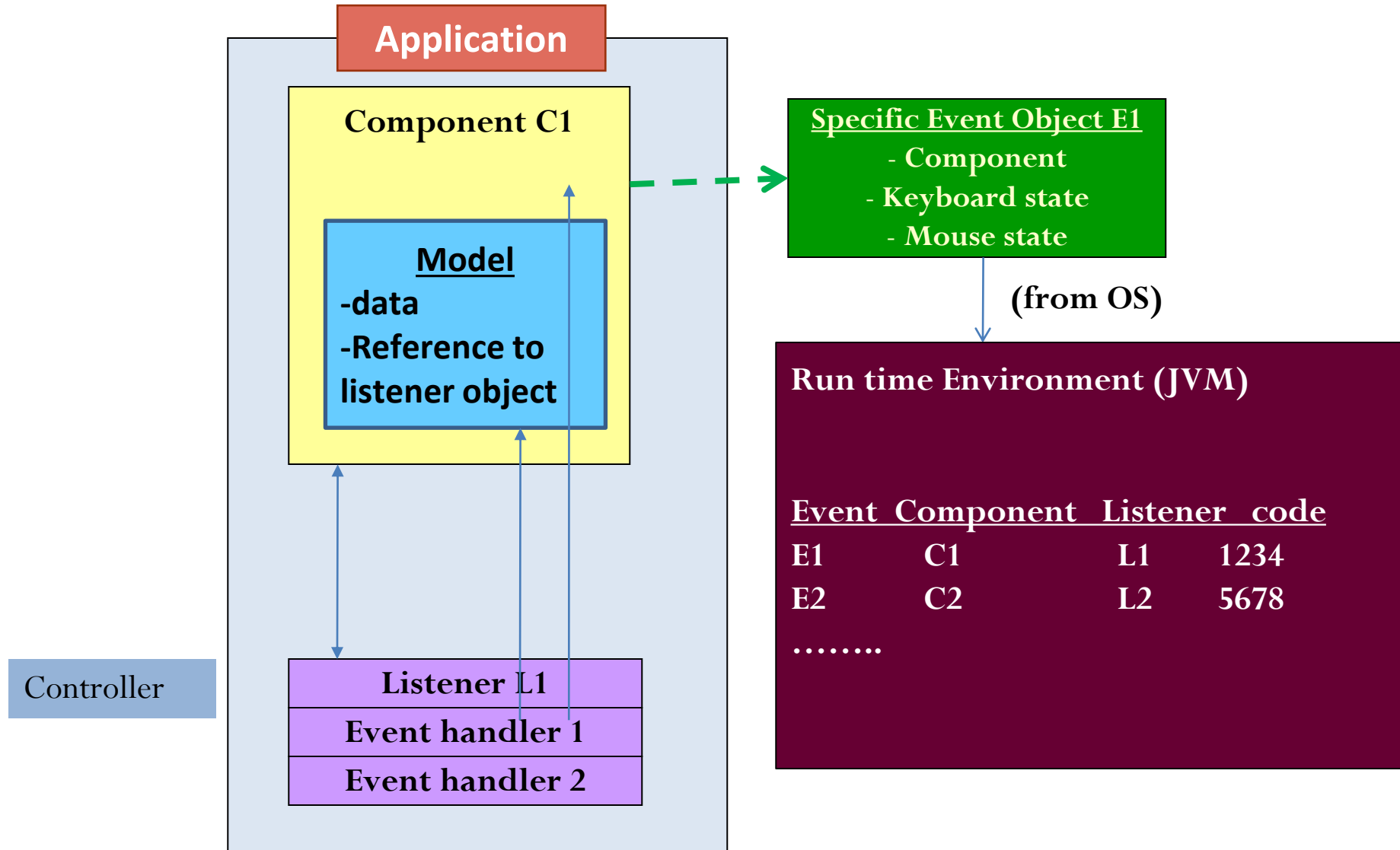
Model-View Controller Architecture for GUI component.

Model: Object contains data.

View: Object users can see it on the screen

Controller: Object manages events

Java model for event management



Steps to perform Event Handling

- Register the component with the Listener
- `public class MyApp extends Frame`
`implements ActionListener`
- For registering the component with the Listener, many classes provide the registration methods. For example:
 - **Button**
 - `public void addActionListener(ActionListener a){}`
 - **MenuItem**
 - `public void addActionListener(ActionListener a){}`

■ **TextField**

- `public void addActionListener(ActionListener a){}`
- `public void addTextListener(TextListener a){}`

■ **TextArea**

- `public void addTextListener(TextListener a){}`

■ **Checkbox**

- `public void addItemListener(ItemListener a){}`

■ **Choice**

- `public void addItemListener(ItemListener a){}`

■ **List**

- `public void addActionListener(ActionListener a){}`
- `public void addItemListener(ItemListener a){}`

Java ActionListener Interface

- **public abstract void** actionPerformed(ActionEvent e);
- **getSource()** method returns the source of the event.

```
1. public class ButtonHandlingDemo implements
   ActionListener {
2. ....
3. btnResult.addActionListener(this);
1. ....
2. public void actionPerformed(ActionEvent ae) {
3. ...
4. if (ae.getSource() == btnResult)
5.     .....
6. }
7. }
```

Example event

```
public class EventExp extends JFrame
    implements ActionListener {
    JLabel lbr=new JLabel("radius: ");
    JTextField tfr=new JTextField(1);
    JLabel lbrs=new JLabel("Area: ");
    JTextField tfrs=new JTextField();
    JButton btnCal=new JButton("Calculate");
    JButton btnExit=new JButton("Exit");
    public EventExp(String title) {
        super(title);
        setLayout(new GridLayout(3, 2));
```

```
        add(lbr);      add(tfr);  
        add(lbrs);     add(tfrs);  
        btnCal.addActionListener(this);  
        btnExit.addActionListener(this);  
        add(btnCal);  add(btnExit);  
    }  
    public static void main(String[] args) {  
        EventExp f=new EventExp("Demo for event");  
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        f.setSize(300, 200);  
        f.setVisible(true);  
    }
```

```
@Override
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == btnCal) {
            double r = Double.parseDouble(tfr.getText());
            double s = r * r * Math.PI;
            DecimalFormat f = new DecimalFormat("#.##");
            tfrs.setText(f.format(s));
        } else if (ae.getSource() == btnExit)
            System.exit(0);
    }
}
```

Some Common Events

<i>Object</i>	<i>Event</i>	<i>Interface</i>	<i>Method</i>
JButton	ActionEvent	ActionListener	actionPerformed()
JCheckBox	ActionEvent ItemEvent	ActionListener ItemListener	actionPerformed() itemStateChanged()
JRadioButton	ActionEvent ItemEvent	ActionListener ItemListener	actionPerformed() itemStateChanged()
JTextField JTextArea	ActionEvent FocusEvent	ActionListener FocusListener	actionPerformed() focusGained(), focusLost()
JPasswordField	ActionEvent	ActionListener	actionPerformed()

Mouse	▶	mouseClicked
MouseMotion	▶	mouseEntered
MouseWheel	▶	mouseExited
		mousePressed
		mouseReleased

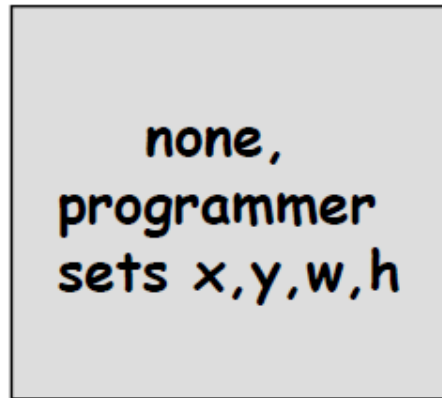
MouseMotion	▶	mouseDragged
MouseWheel	▶	mouseMoved
MouseWheel	▶	mouseWheelMoved

Layout Manager (1)

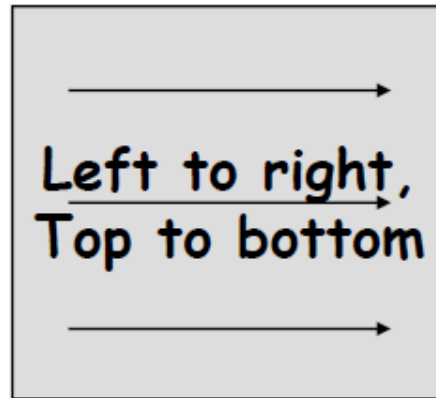
- ✓ Flow Layout
 - ✓ Border Layout
 - ✓ Card Layout
 - ✓ Grid Layout
 - ✓ GridBag Layout
 - ✓ Box Layout
 - ✓ Overlay Layout
-
- Defined in the AWT
- Defined in Swing

Layout Manager (2)

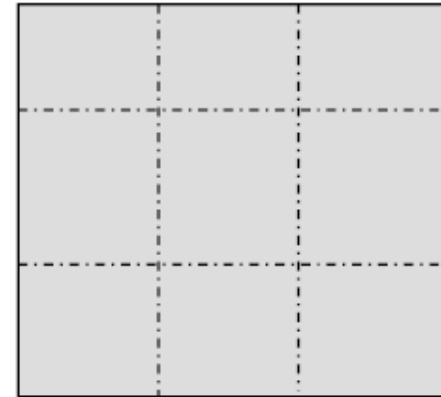
null



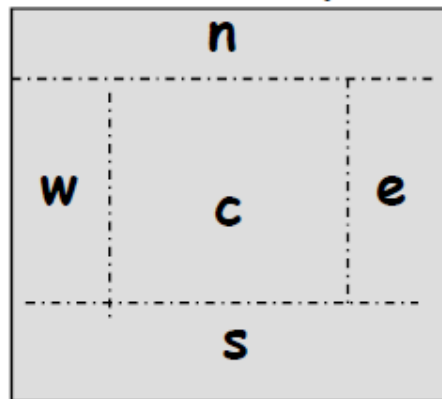
FlowLayout



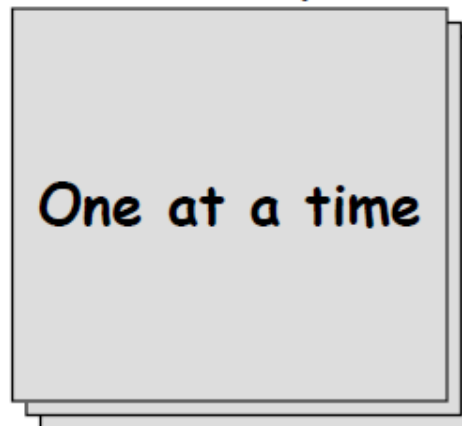
GridLayout



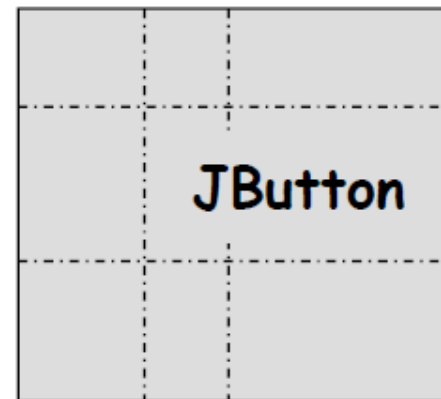
BorderLayout



CardLayout



GridBagLayout

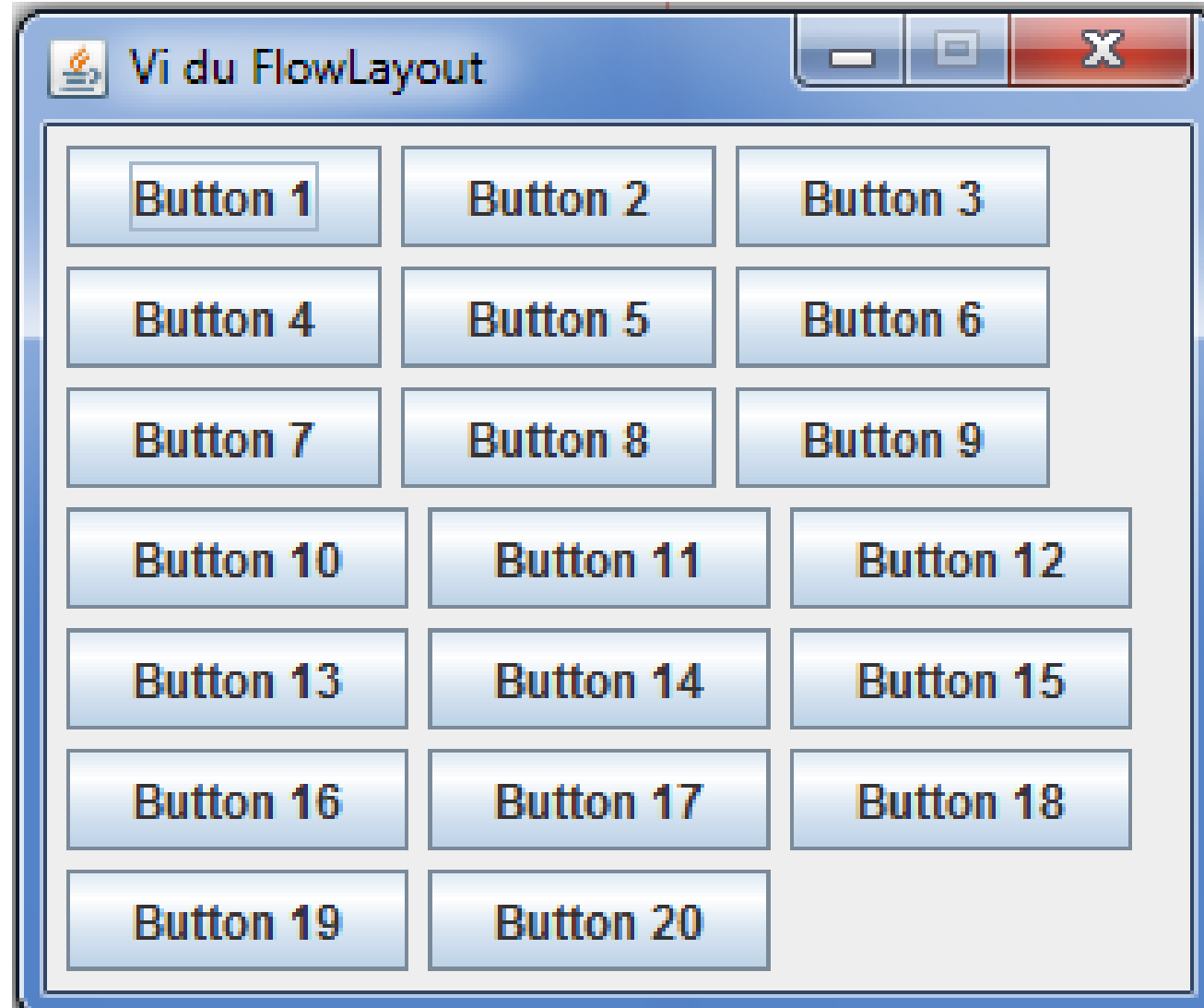


FlowLayout

- The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.
- `public FlowLayout()`
 - Centers each row and keeps 5 pixels between entries in a row and between rows
- `public FlowLayout(int align)`
 - Same 5 pixels spacing, but changes the alignment of the rows to `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`
- `public FlowLayout(int align, int hgap, int vgap)`
 - Specify the alignment as well as the horizontal and vertical spacing between components (in pixel)

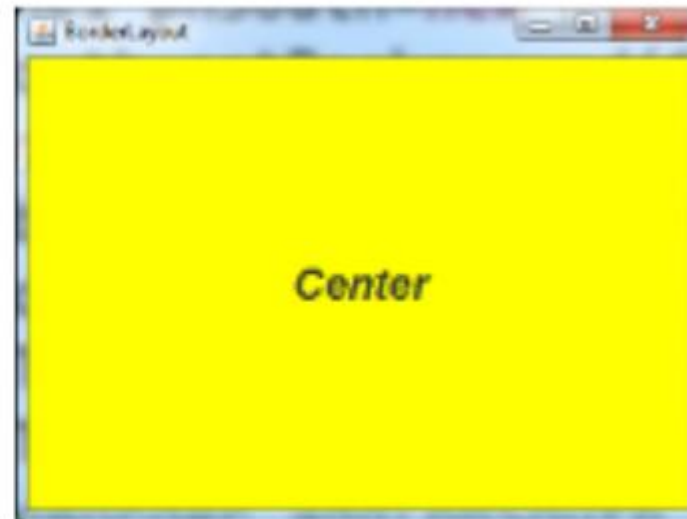
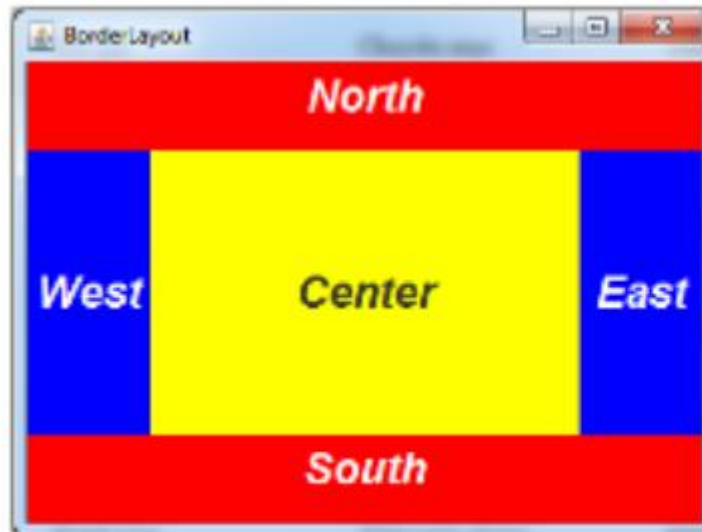
```
1. public class DemoFlowLayout extends JFrame{
2.     public DemoFlowLayout() {
3.         setTitle("Exampe FlowLayout");
4.         setSize(300,250);
5.         setDefaultCloseOperation(EXIT_ON_CLOSE);
6.         setLocationRelativeTo(null);
7.         setResizable(false);
8.         setLayout(new FlowLayout(FlowLayout.LEFT)) ;
9.         for(int i=1;i<=20;i++){
10.            add(new JButton("Button "+i));
11.        }
12.     public static void main(String[] args) {
13.         new DemoFlowLayout().setVisible(true);
14.     }
15. }
```

Result

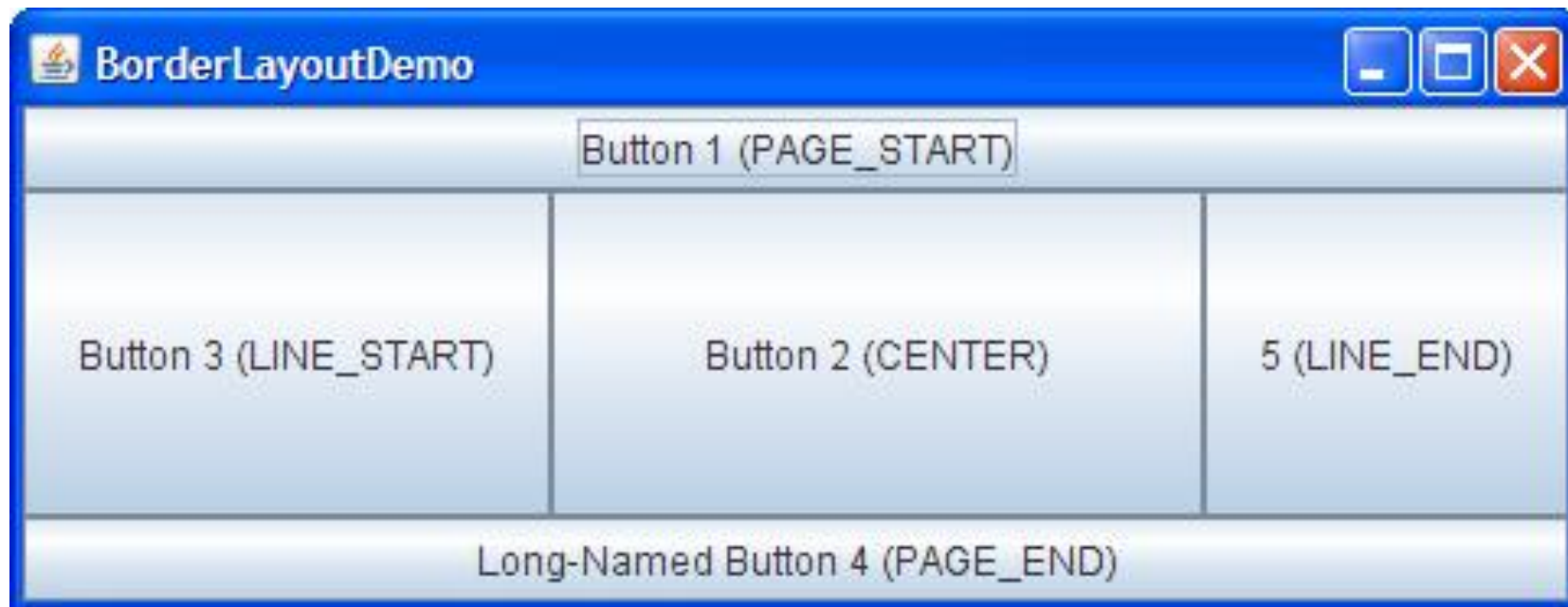


Border Layout

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



- **public static final int NORTH**
- **public static final int SOUTH**
- **public static final int EAST**
- **public static final int WEST**
- **public static final int CENTER**
- Constructors of BorderLayout class:
- **BorderLayout()**: creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap)**: creates a border layout with the given horizontal and vertical gaps between the components.

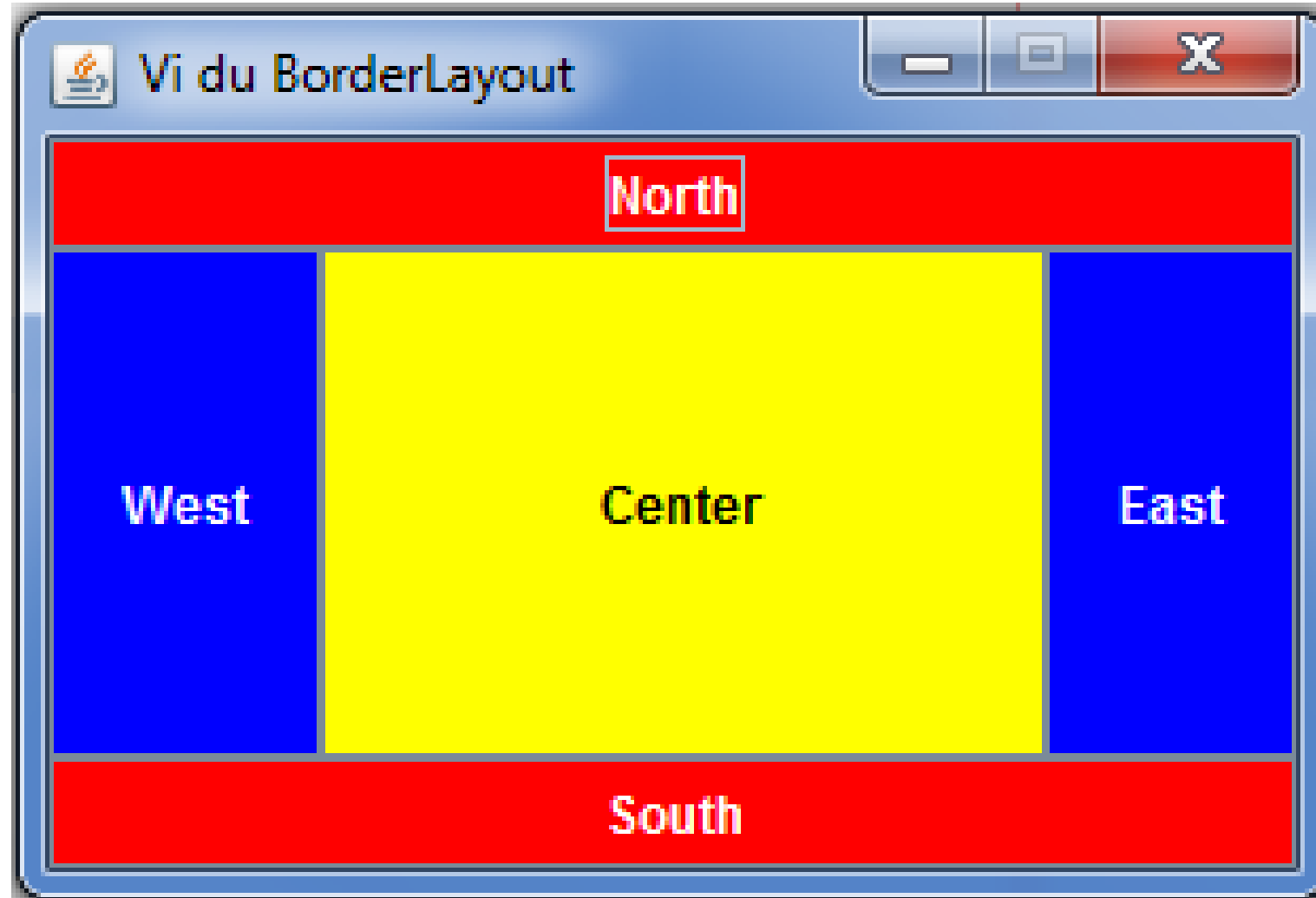


```
1. public class DemoBorderLayout extends JFrame{
2.     private JButton
3.     bn=new JButton("North"),
4.     bs=new JButton("South"),
5.     be=new JButton("East"),
6.     bw=new JButton("West"),
7.     bc=new JButton("Center");
8.     public DemoBorderLayout()    {
9.         setTitle("BorderLayout");
10.        setSize(300,200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
11.        setLocationRelativeTo(null);
12.        setResizable(false);
13.        add(BorderLayout.NORTH, bn);
14.        add(BorderLayout.SOUTH, bs);
15.        add(BorderLayout.EAST, be);
16.        add(BorderLayout.WEST, bw);
17.        add(BorderLayout.CENTER, bc);
```

```
1. bn.setBackground(Color.red);
2.     bs.setBackground(Color.red);
3.     bc.setBackground(Color.YELLOW);
4.     be.setBackground(Color.BLUE);
5.     bw.setBackground(Color.BLUE);
6.     bn.setForeground(Color.WHITE);
7.     bs.setForeground(Color.WHITE);
8.     bc.setForeground(Color.black);
9.     be.setForeground(Color.WHITE);
10.    bw.setForeground(Color.WHITE);
11.    }
12. public static void main(String[] args) {
13.     new DemoBorderLayout().setVisible(true);

14.     }
15. }
```


Result

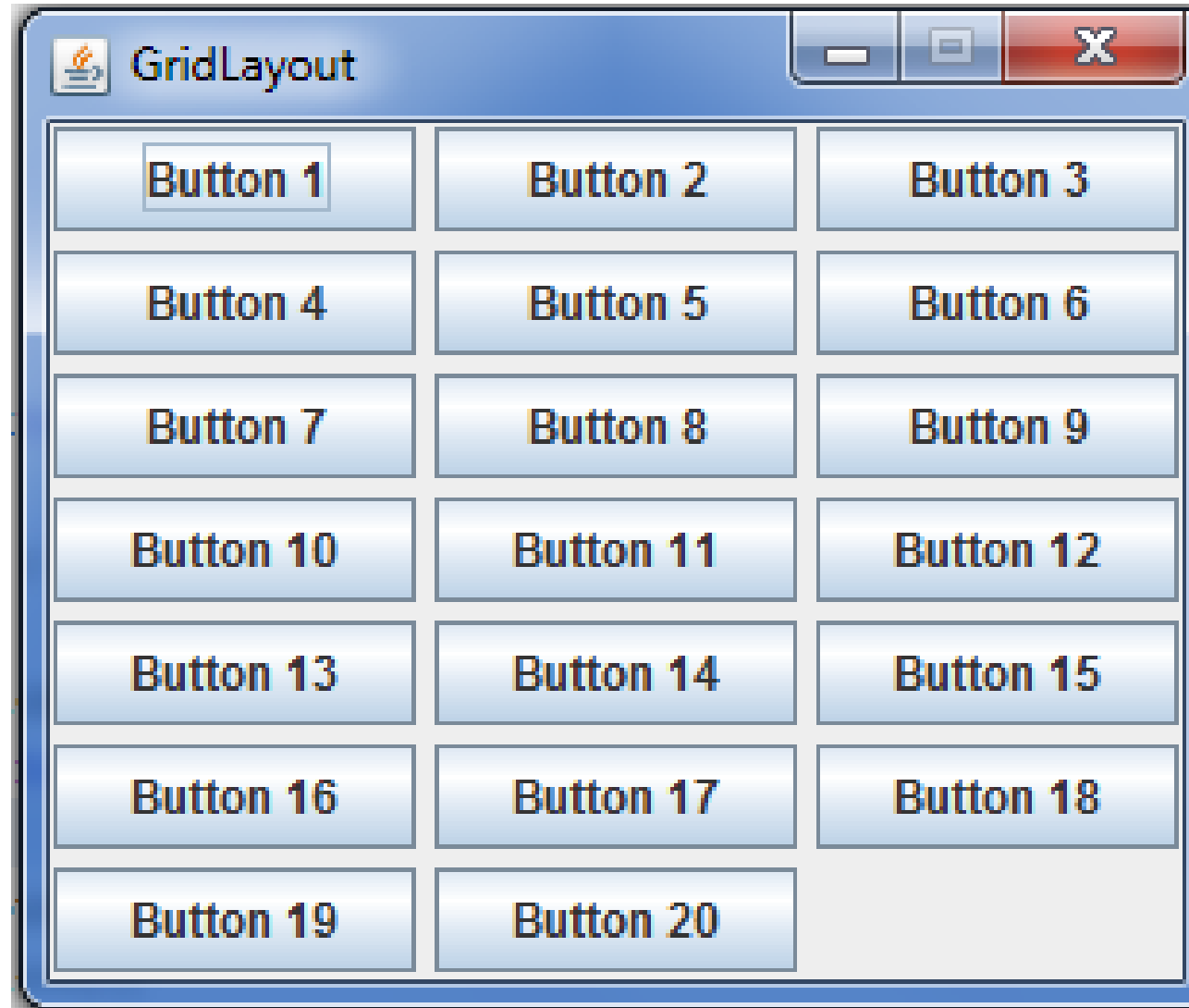


Grid Layout

- The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.
- `public GridLayout()`
 - Creates a single row with one column allocated per component
- `public GridLayout(int rows, int cols)`
 - Divides the window into the specified number of rows and columns
 - Either rows or cols (but not both) can be zero
- `public GridLayout(int rows, int cols, int hgap, int vgap)`
 - Uses the specified gaps between cells

```
1. public class DemoGridLayout extends JFrame{
2.     public DemoGridLayout() {
3.         setTitle("GridLayout");
4.         setSize(300, 250);
5.         setDefaultCloseOperation(EXIT_ON_CLOSE);
6.         setLocationRelativeTo(null);
7.         setResizable(false);
8.         setLayout(new GridLayout(7,3,5,5));
9.         for(int i = 1;i <=20;i++){
10.             add(new JButton("Button "+i));
11.         }
12.     }
13.     public static void main(String[] args) {
14.         new DemoGridLayout().setVisible(true);
15.     }
16. }
```

Result

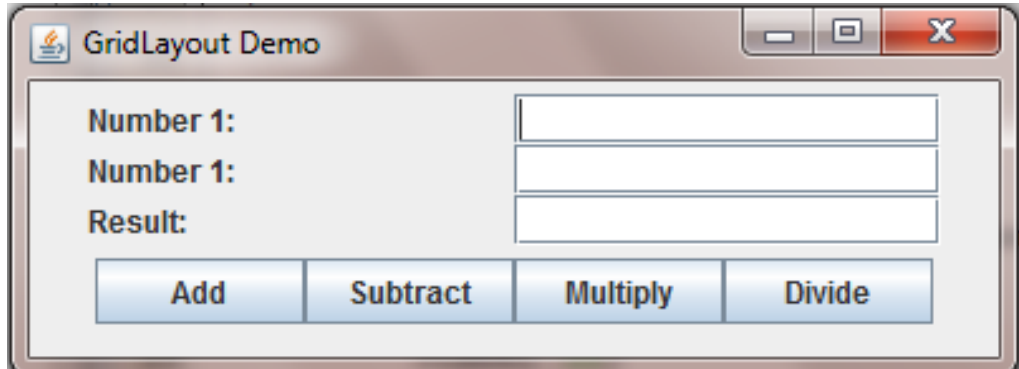


```
1. public class DemoGridLayout1 extends JFrame {
2.     public DemoGridLayout1() {
3.         initUI(); }
4.     public final void initUI() {
5.         JPanel panel = new JPanel();
6.
       panel.setBorder(BorderFactory.createEmptyBorder(5, 5,
       5, 5));
7.         panel.setLayout(new GridLayout(5, 4, 5, 5));
8.         String[] buttons = {
9.             "Cls", "Bck", "", "Close",
10.            "7", "8", "9", "/",
11.            "4", "5", "6", "*",
12.            "1", "2", "3", "-",
13.            "0", ".", "=", "+"
14.        };
```

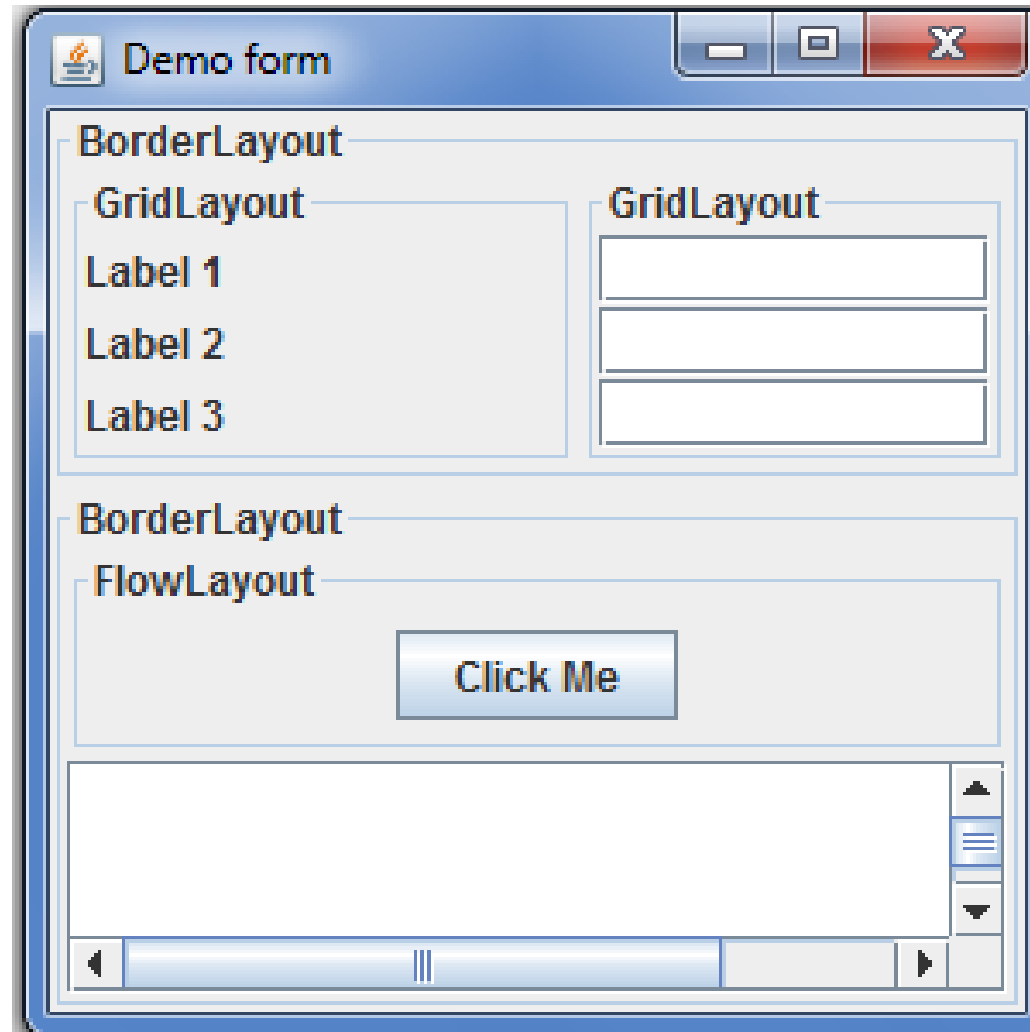
```
1. for (int i = 0; i < buttons.length; i++) {
2.     if (i == 2)
3.         panel.add(new JLabel(buttons[i]));
4.     else
5.         panel.add(new JButton(buttons[i]));
6.     }
7.     add(panel);
8.     setTitle("Example GridLayout");
9.     setSize(350, 300);
10.    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11.    setLocationRelativeTo(null);
12. }
13. public static void main(String[] args) {
14.     DemoGridLayout1 ex = new DemoGridLayout1();
15.     ex.setVisible(true);
16.     }
17. }
```

Combinations (Gridlayout)

- JPanel **p1**=new JPanel();
 p1.setLayout(new GridLayout(3,2));
- JPanel **p2**=new JPanel();
 p2.setLayout(new GridLayout(1,4));
- JPanel **main**=new JPanel();
 main.add(**p1**, BorderLayout.NORTH);
 main.add(**p2**, BorderLayout.SOUTH);
 this.setContentPane(main);
 //add(main);



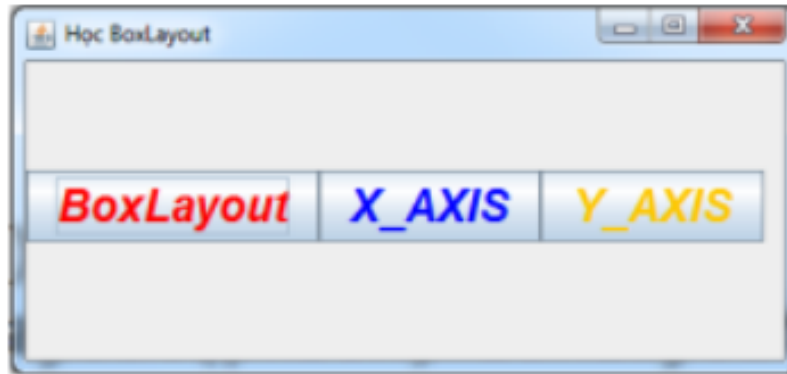
DemoBorder.java



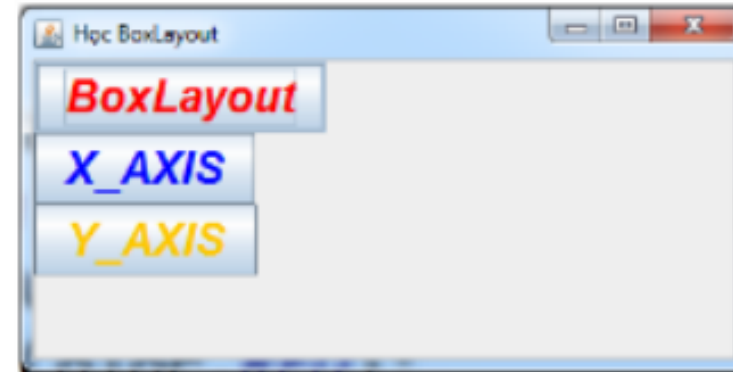
Box Layout

- The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants.
- Fields of BoxLayout class
- **public static final int X_AXIS**
- **public static final int Y_AXIS**
- **public static final int LINE_AXIS**
- **public static final int PAGE_AXIS**
- Constructor of BoxLayout class
- **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

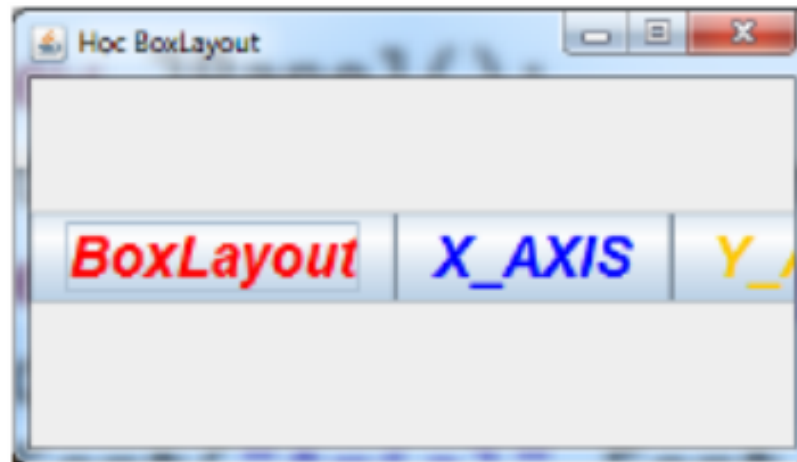
Example: Box Layout



BoxLayout.X_AXIS



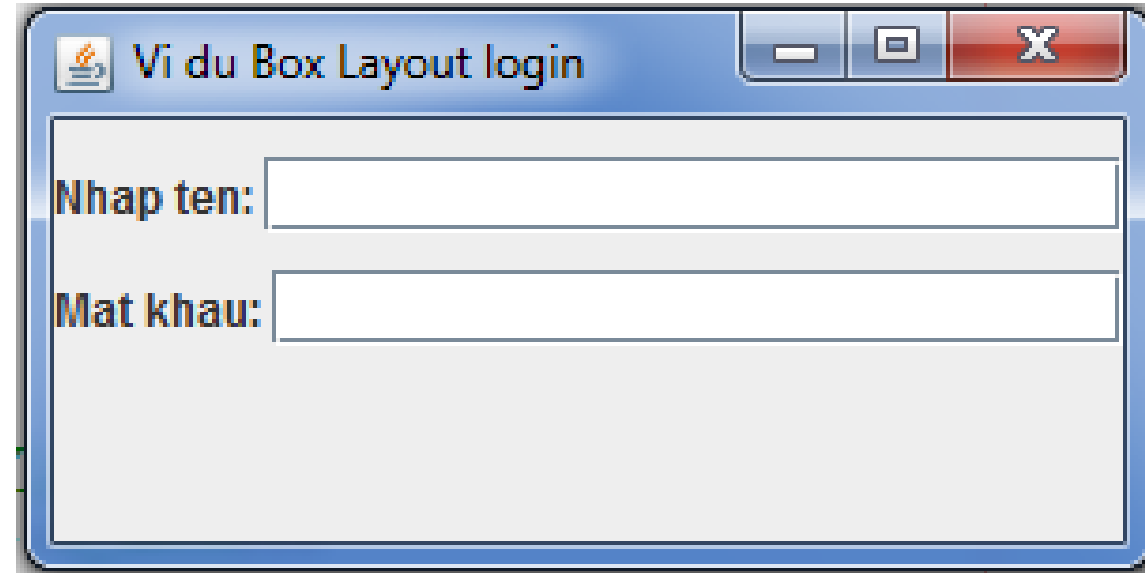
BoxLayout.Y_AXIS



No wrap row when resize dimension

```
1. public class BoxLayout1 extends JFrame{
2.     public BoxLayout1(){
3.         setTitle("Box Layout login example");
4.         setSize(300,150);
5.         setDefaultCloseOperation(EXIT_ON_CLOSE);
6.         JPanel p = new JPanel();
7.         JPanel p1 = new JPanel();
8.         JPanel p2 = new JPanel();
9.         p.setLayout(new BoxLayout(p, BoxLayout.Y_AXIS));
10.        p1.setLayout(new BoxLayout(p1,
        BoxLayout.X_AXIS));
11.        p2.setLayout(new BoxLayout(p2,
        BoxLayout.X_AXIS));
12.        p1.add(new JLabel("enter name: "));
13.        p1.add(new JTextField(15));
14.        p2.add(new JLabel("enter pass: "));
15.        p2.add(new JPasswordField(15));
```

```
1. p.add(Box.createRigidArea(new Dimension(10, 10)));
2. p.add(p1);
3. p.add(Box.createRigidArea(new Dimension(10, 10)));
4. p.add(p2);
5. this.add(p, BorderLayout.NORTH);
6. }
7. public static void main(String[] args) {
8.     new BoxLayout1().setVisible(true);
9. }
10. }
```



```
1. public class BoxLayout2 extends JFrame {  
2.     public BoxLayout2() {  
3.         setTitle("Box Layout login example");  
4.         setSize(300,150);  
5.         setDefaultCloseOperation(EXIT_ON_CLOSE);  
6.         Box b=Box.createVerticalBox();  
7.         Box p1=Box.createHorizontalBox();  
8.         Box p2=Box.createHorizontalBox();  
9.         p1.add(new JLabel("enter name: "));  
10.        p1.add(new JTextField(15));  
11.        p2.add(new JLabel("enter pass: "));  
12.        p2.add(new JPasswordField(15));
```

```
1 .b.add(Box.createRigidArea(new  
    Dimension(10, 10)));  
2 . b.add(p1);  
3 . b.add(Box.createRigidArea(new  
    Dimension(10, 10)));  
4 . b.add(p2);  
5 . this.add(b, BorderLayout.NORTH);  
6 . }  
7 .public static void main(String[] args) {  
8 .    new BoxLayout2().setVisible(true);  
9 .    }  
10 . }
```

Example Login

```
1. public class DemoLogin extends JFrame implements  
   ActionListener{  
2.     private      JButton      bLogon;  
3.     private      JButton      bExit;  
4.     private      JTextField username;  
5.     private      JPasswordField pass;  
6.     public DemoLogin() {  
7.         setTitle("Logon  program");  
8.         setSize(500, 350);  
9.         setDefaultCloseOperation(EXIT_ON_CLOSE);  
10.        setLocationRelativeTo(null);  
11.        buildGUI();  
12.    }
```

```
1. private void buildGUI() {
2.     JPanel p1=new JPanel();
3.     p1.setBorder(BorderFactory.createLineBorder(Color.red));
4.     JLabel tieude;
5.     p1.add(tieude = new JLabel("LOGIN"));
6.     tieude.setFont(new Font("Arial", Font.BOLD, 30));
7.     tieude.setForeground(Color.red);
8.     add(p1,BorderLayout.NORTH);
9.     JPanel p2 = new JPanel();
10.    p2.setBorder(BorderFactory.createLineBorder(Color.red));
11.    p2.add(bLogon=new JButton("Login"));
12.    p2.add(bExit=new JButton("Exit"));
13.    add(p2,BorderLayout.SOUTH);
14.    JPanel p3=new JPanel();
15.    p3.setBorder(BorderFactory.createLineBorder(Color.red));
16.    Box b=Box.createVerticalBox();
17.    Box b1=Box.createHorizontalBox();
18.    Box b2=Box.createHorizontalBox();
```



```
1. JLabel    lblUser, lblPass;
2.          b1.add(lblUser=new JLabel("Name:  "));
3.          lblUser.setFont(new Font("Arial", Font.PLAIN, 15));
4.          b1.add(username= new JTextField(20));
5.          b2.add(lblPass=new JLabel("Password:  "));
6.          lblPass.setFont(new Font("Arial", Font.PLAIN, 15));
7.          b2.add(pass=new JPasswordField(20));
8.          lblUser.setPreferredSize(lblPass.getPreferredSize());
9.          b.add(Box.createVerticalStrut(50));
10.         b.add(b1);
11.         b.add(Box.createVerticalStrut(10));
12.         b.add(b2);
13.         p3.add(b);
14.         add(p3, BorderLayout.CENTER);
15.         username.addActionListener(this);
16.         pass.addActionListener(this);
17.         bLogon.addActionListener(this);
18.         bExit.addActionListener(this);
19.     }
```

```
1. @Override
2.     public void actionPerformed(ActionEvent e) {
3.         if (e.getSource() == bLogon) {
4.
5.             if (username.getText().equalsIgnoreCase("anh") &&
6.                 pass.getText().equalsIgnoreCase("anh")) {
7.                 dispose();
8.                 JOptionPane.showMessageDialog(null, "Lgin success!!!");
9.             }
10.         else {
11.             JOptionPane.showMessageDialog(null, "incorrect!!!");
12.             username.requestFocus();
13.         }
14.         else if (e.getSource() == bExit) {
15.             System.exit(0);
16.         }
17.     }
```

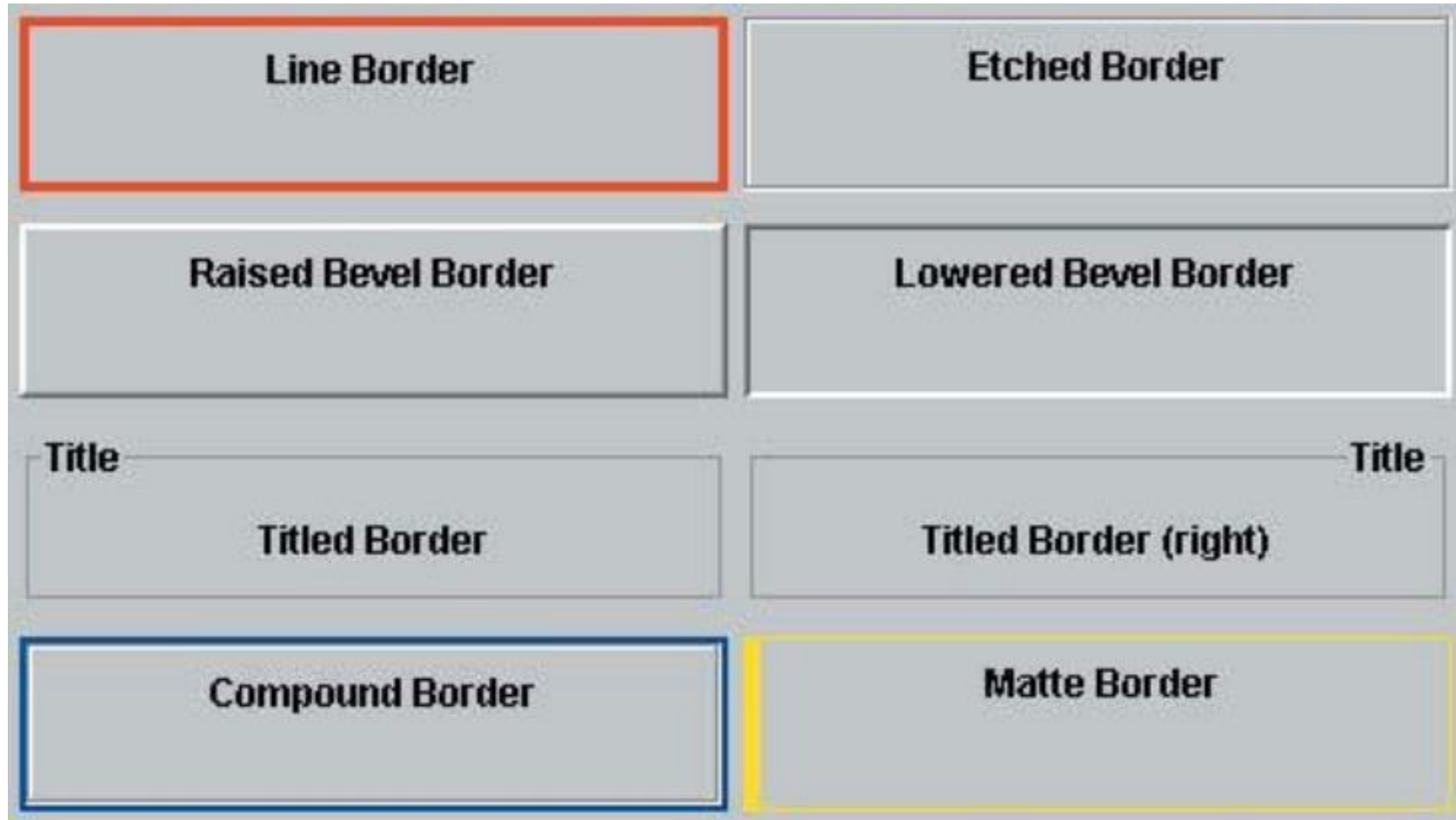
Borders

- Every JComponent can have one or more borders
- To put a border around a JComponent, you use its setBorder method. You can use the [BorderFactory](#)

`BorderFactory.createxxxBorder (...)`

- ```
JPanel pane = new JPanel();
pane.setBorder(BorderFactory.createLineBorder(Color.black));
```

# Example



# JTabbedPane

- The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.
- Constructors:
- JTabbedPane()
- JTabbedPane(int tabPlacement)
- JTabbedPane(int tabPlacement, int tabLayoutPolicy)

# Example

```
public class TabbedPane {
 private static void showGUI() {
 final JFrame m = new JFrame("Tabbed Pane
Example");
 m.setSize(400, 400); m.setVisible(true);
m.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 m.setLayout(new GridLayout(1, 1));
 JTabbedPane tab = new
JTabbedPane(JTabbedPane.TOP);
 tab.addTab("Tab1", addPanel("This is tab1"));
 tab.addTab("Tab2", addPanel("This is tab2"));
 tab.addTab("Tab3", addPanel("This is tab3"));
 tab.addTab("Tab4", addPanel("This is tab4"));
 m.add(tab);
 }
}
```

```
int selectedIndex = tab.getSelectedIndex();
 System.out.println("Default Index:" + selectedIndex);
 tab.setSelectedIndex(tab.getTabCount()-1);
 selectedIndex = tab.getSelectedIndex();
 System.out.println("Index:" + selectedIndex);
}
private static JPanel addPanel(String text) {
 JPanel p = new JPanel();
 p.add(new JLabel(text));
 p.setLayout(new GridLayout(1, 1));
 return p;
}
public static void main(String[] args) {
 showGUI();
}
}
```

# JCheckBox - JRadioButton

- **JCheckBox:** The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on".
- **JRadioButton** The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz. It should be added in ButtonGroup to select one radio button only.



# JCheckBox – Constructor

- JCheckBox()
- JCheckBox(String text)
- JCheckBox(String text, boolean selected)
- JCheckBox(Icon icon)
- JCheckBox(String text, Icon icon)
- JCheckBox(String text, Icon icon, boolean selected)

# JCheckBox – Methods

- `boolean isSelected()`
  - returns the state of the checkbox
- `void setSelected(boolean state)`
  - sets the checkbox to a new state
- `String getText()`
- `void setText(String text)`
  - gets or sets the button's text
- `addItemListener`
  - Add an ItemListener to process ItemEvent in itemStateChanged

# JCheckBox Demo

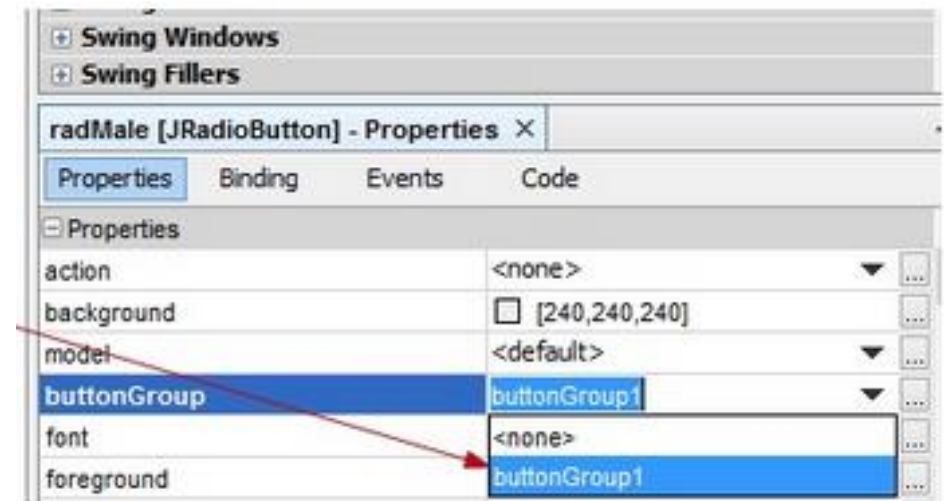
- `JCheckBox ga, bo;`
- `JPanel p2 = new JPanel();`
- `p2.add(chicken=new JCheckBox("fried chicken"));`
- `p2.add(beef = new JCheckBox("beef"));`
- `add(p2);`
- `public void itemStateChanged(ItemEvent e) {`
- `if (e.getItem() == chicken)`
- `JOptionPane.showMessageDialog(null, "You select fried chicken");`
- `if (e.getItem() == beef)`
- `JOptionPane.showMessageDialog(null, "You select beef");`

# JRadioButton – Constructor

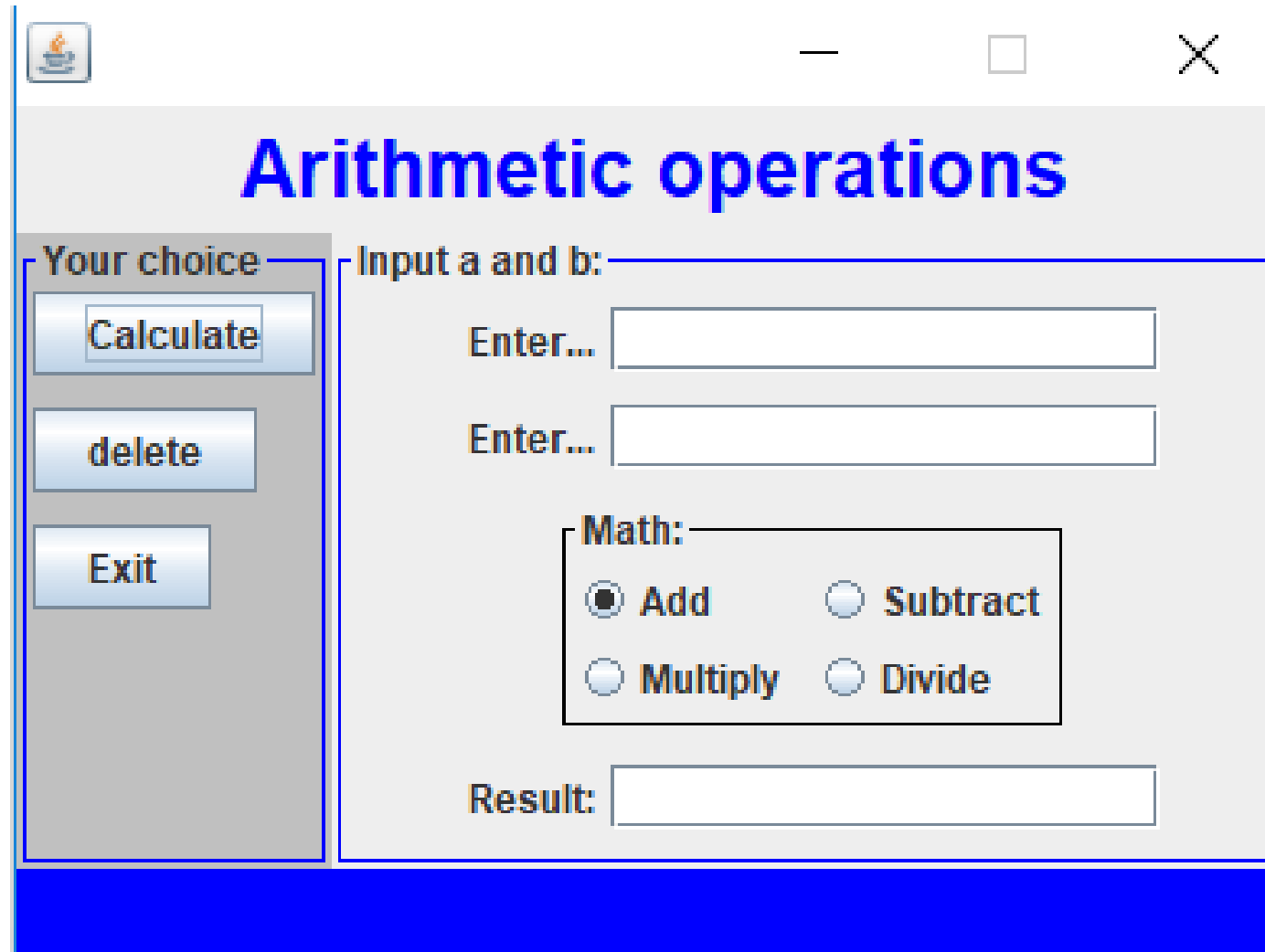
- JRadioButton()
- JRadioButton(String text)
- JRadioButton(String text, boolean selected)
- JRadioButton(Icon icon)
- JRadioButton(String text, Icon icon)
- JRadioButton(String text, Icon icon, boolean selected)

# JRadioButton vs Button Group

- `ButtonGroup pt=new ButtonGroup();`
- `Final JRadioButton add=new JRadioButton("Add");`
- `Final JRadioButton subtract=new  
JRadioButton("Subtract");`
- `Final JRadioButton multiply=new JRadioButton("Multiply");`
- `Final JRadioButton divide=new JRadioButton("Divide");`
- `pt.add(add);pt.add(subtract);`
- `pt.add(multiply);pt.add(divide);`



# Example



The image shows a Java Swing window titled "Arithmetic operations". The window has a standard title bar with a minimize button, a maximize button, and a close button. The main content area is divided into two sections. On the left, there is a vertical panel titled "Your choice" containing three buttons: "Calculate", "delete", and "Exit". On the right, there is a section titled "Input a and b:" followed by two text input fields, each preceded by the label "Enter...". Below these input fields is a section titled "Math:" containing four radio buttons arranged in a 2x2 grid: "Add" (selected), "Subtract", "Multiply", and "Divide". At the bottom of the right section is a text input field preceded by the label "Result:". The entire window has a blue footer bar.

**Arithmetic operations**

Your choice

Calculate

delete

Exit

Input a and b:

Enter...

Enter...

Math:

☒ Add ☐ Subtract

☐ Multiply ☐ Divide

Result:

# JComboBox (1)

- The object of Choice class is used to show popup menu of choices



## Constructors

- public JComboBox()
- public JComboBox(Object[] items)
- public JComboBox(Vector items)

### Common used methods

- public void addItem(Object item)
- public int getItemCount()
- public int getSelectedIndex()
- public Object getSelectedItem()
- public void setEditable(boolean editable)
- public void setSelectedIndex(int index)
- public void setSelectedItem(Object item)

# JComboBox (2)

The image shows the Swing Controls palette with the JComboBox control selected. Below it, the Properties window for cmbQualification [JComboBox] is open, showing various properties. A callout box points to the 'model' property, indicating where to enter the content for the JComboBox.

**Swing Controls**

- Label Label
- Check Box
- Combo Box
- Text Area
- Progress Bar
- Spinner
- Editor Pane
- Button
- Radio Button
- List
- Scroll Bar
- Formatted Field
- Separator
- Tree
- Toggle Button
- Button Group
- Text Field
- Slider
- Password Field
- Text Pane
- Table

**Swing Menus**

**Swing Windows**

**cmbQualification [JComboBox] - Properties**

Properties Binding Events Code

Properties

|                 |                   |
|-----------------|-------------------|
| background      |                   |
| editable        |                   |
| font            | Tahoma 11         |
| foreground      | [0,0,0]           |
| maximumRowCount | 8                 |
| model           | Graduate, Student |
| selectedIndex   | 0                 |
| selectedItem    | Graduate          |
| toolTipText     |                   |

Other Properties

**model**

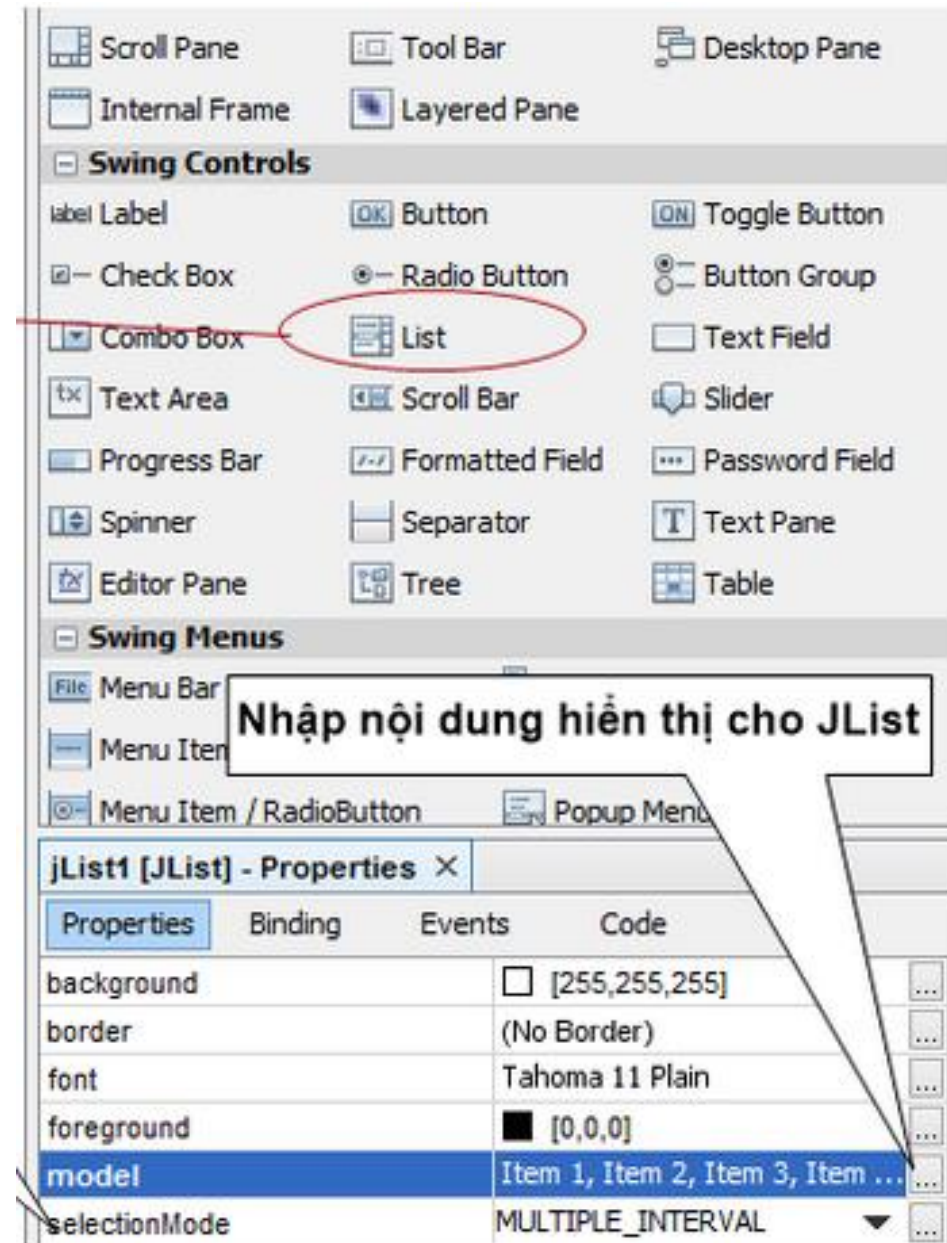
(javax.swing.ComboBoxModel) Model that the combo box uses to get data to display.

Chọn vào đây để nhập nội dung cho JComboBox



# Jlist (1)

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



# Jlist (2)

## ■ Constructors

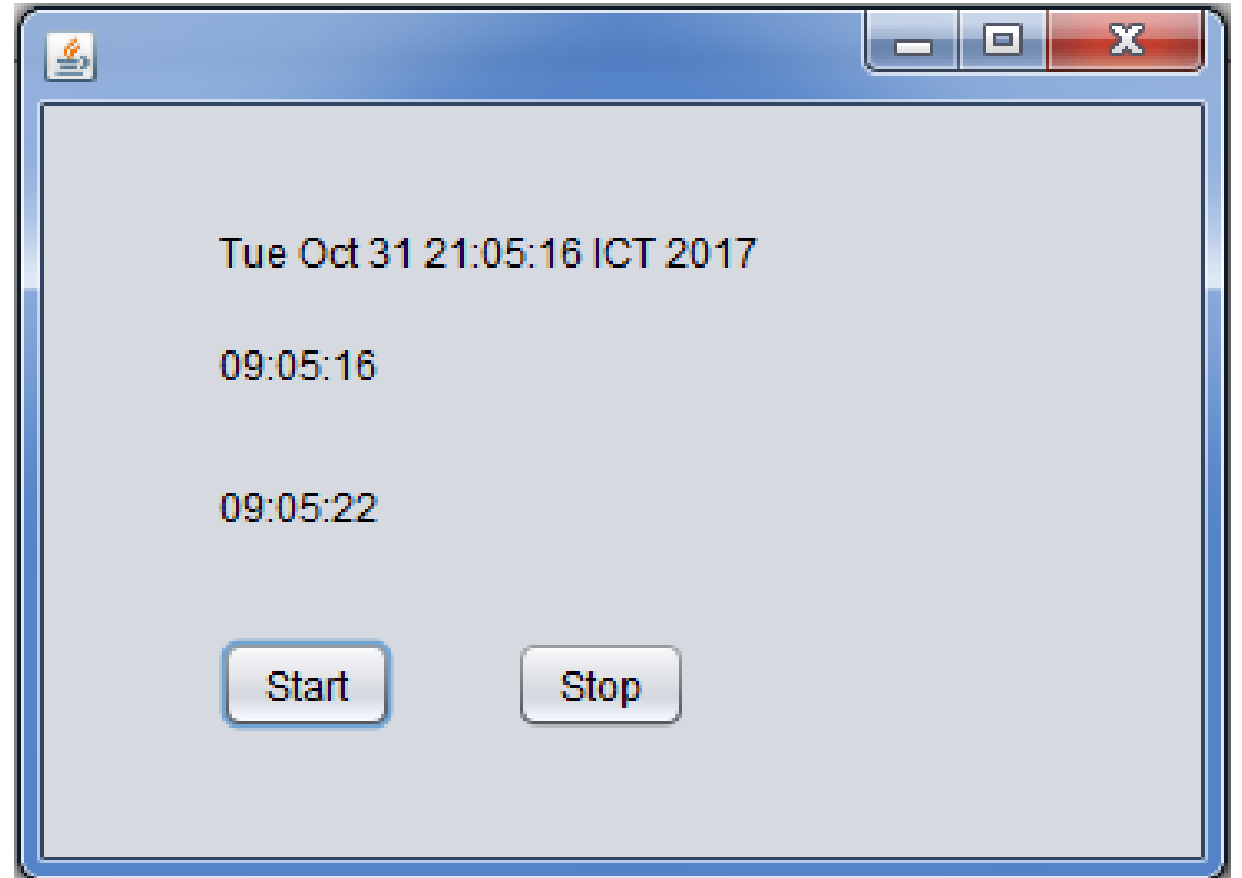
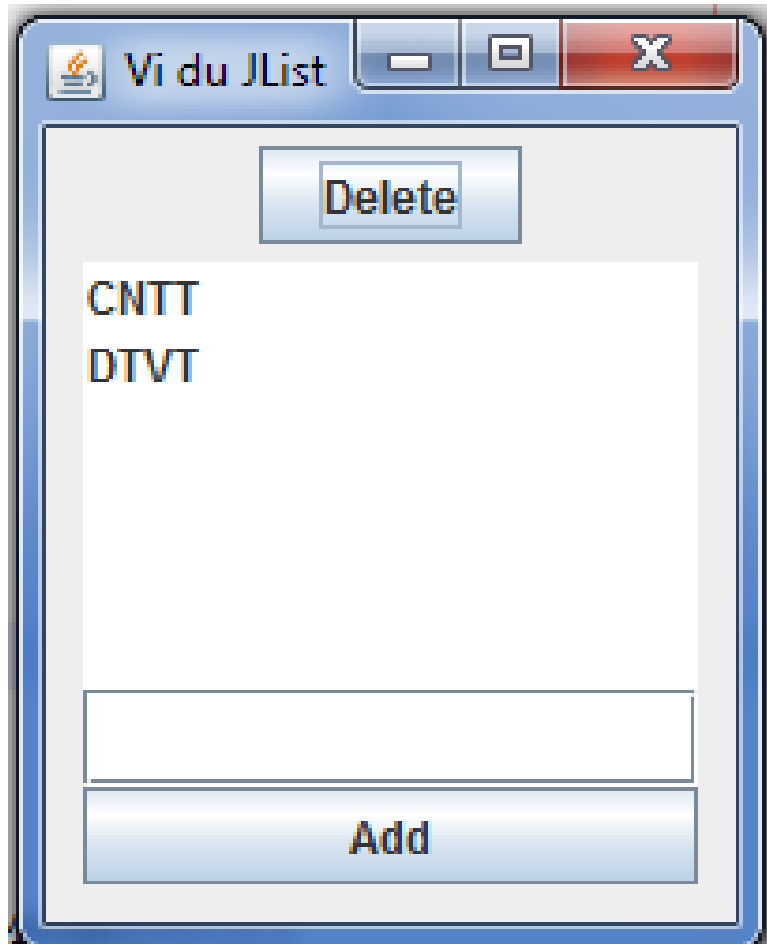
- JList()
- JList ( Object[] dataItems)
- JList ( Vector vectorItems)

## ■ Event handling:

- ListSelectionEvent
- ListSelectionListener
- public void valueChanged(ListSelectionEvent e)

```
public void clearSelection()
public int getSelectedIndex()
public int[] getSelectedIndices()
public boolean isSelectionEmpty()
public void setListData(Object[] items)
public void setSelectedIndex(int index)
public void setSelectedIndices(int[] indices)
```

# Example



# JSlider

- The Java JSlider class is used to create the slider. By using JSlider, a user can select a value from a specific range.
- **JSlider()**: creates a slider with the initial value of 50 and range of 0 to 100.
- **JSlider(int orientation)**: creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.
- **JSlider(int min, int max)**: creates a horizontal slider using the given min and max.
- **JSlider(int min, int max, int value)**: creates a horizontal slider using the given min, max and value.
- **JSlider(int orientation, int min, int max, int value)**: creates a slider using the given orientation, min, max and value.

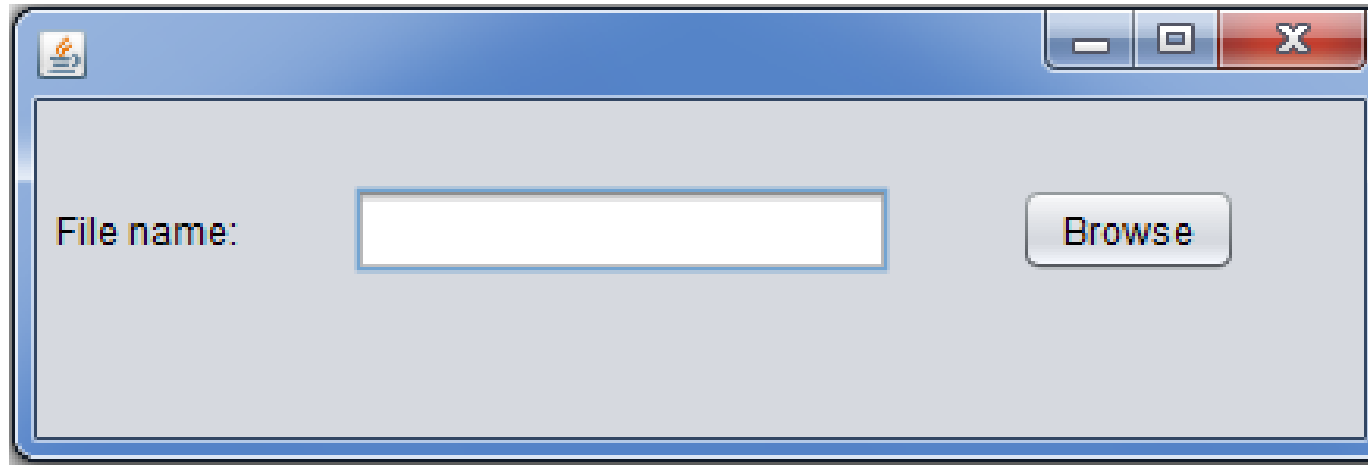
# Example

```
JLabel jLabel1;
public Main() {
 initComponents();
 setLayout(new BorderLayout());
 jLabel1=new JLabel("Java is cool",JLabel.CENTER);
 jLabel1.setFont(new Font("Times New Roman", Font.BOLD,
 32));
 add(jLabel1,BorderLayout.NORTH);
 add(jSlider1,BorderLayout.CENTER);
 jSlider1.setMinimum(200);
 jSlider1.setMaximum(1000);
 jSlider1.setMinorTickSpacing(20);
 jSlider1.setMajorTickSpacing(100);
 jSlider1.setPaintLabels(true);
 jSlider1.setPaintTicks(true);
```

# JFileChooser

- The object of JFileChooser class represents a dialog window from which the user can select file.
- **constructor**
- JFileChooser()
- JFileChooser(File currentDirectory)
- **int showDialog(Component parent, String approveButtonText)** Displays a file chooser with the approve button text specified by the String argument
- **int showSaveDialog(Component parent)** Displays a file chooser with a "Save" approve button
- **int showOpenDialog(Component parent)** Displays an file chooser with an "Open" approve button

# Example



```
private void
 jButton1ActionPerformed(java.awt.
 event.ActionEvent evt) {
 txtfile.setText(""+displayChosenF
 ile());
 }
```

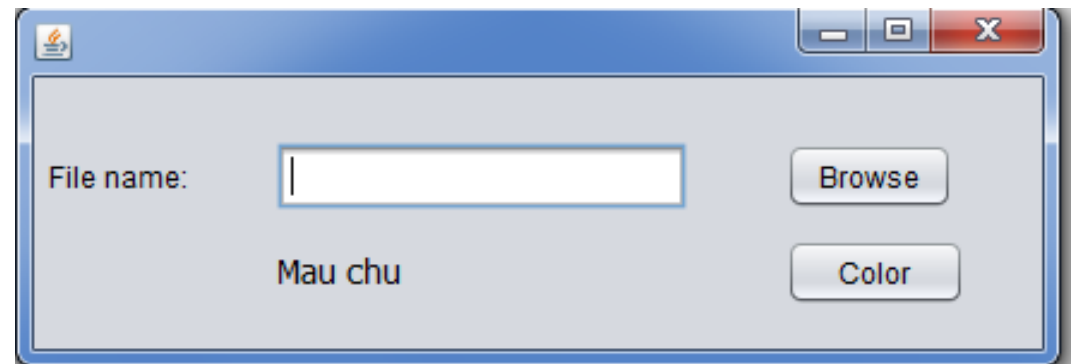
```
private String displayChosenFile() {
 String filestr=null;
 JFileChooser file=new JFileChooser(".");
 int select=file.showOpenDialog(null);
 if(select==JFileChooser.APPROVE_OPTION) {
 File
 selectedFile=file.getSelectedFile();
 System.out.println(selectedFile.getParent());
 System.out.println(selectedFile.getName());
 try{
 filestr=selectedFile.getCanonicalPath();
 }catch(Exception e) {
 e.printStackTrace();
 }
 }
 return filestr;
}
```



# JColorChooser

- The JColorChooser class is used to create a color chooser dialog box so that user can select any color.
- **JColorChooser()**: It is used to create a color chooser panel with white color initially.
- **JColorChooser(Color initialColor)**: It is used to create a color chooser panel with the specified color initially.

```
Color c = JColorChooser.showDialog(this,
 "Choose foreground color", Color.BLACK);
 if(c != null){
jTextArea1.setForeground(c);
jButton1.setForeground(c);
 }
```



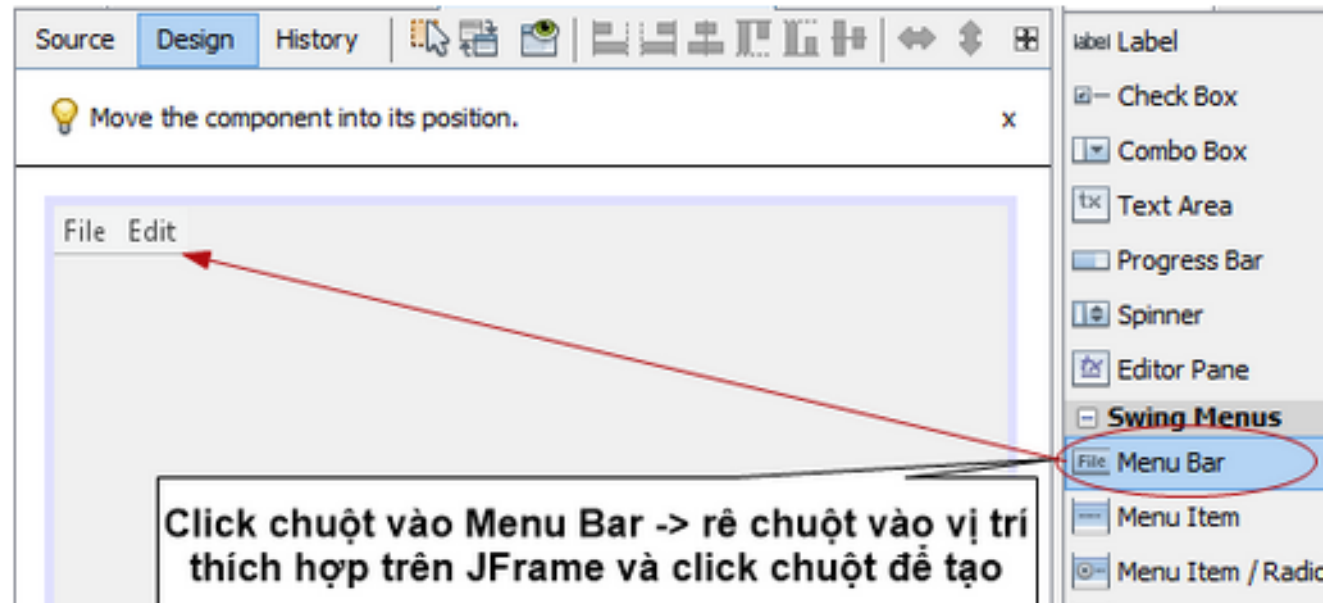
# Swing Menu Components

## Objectives



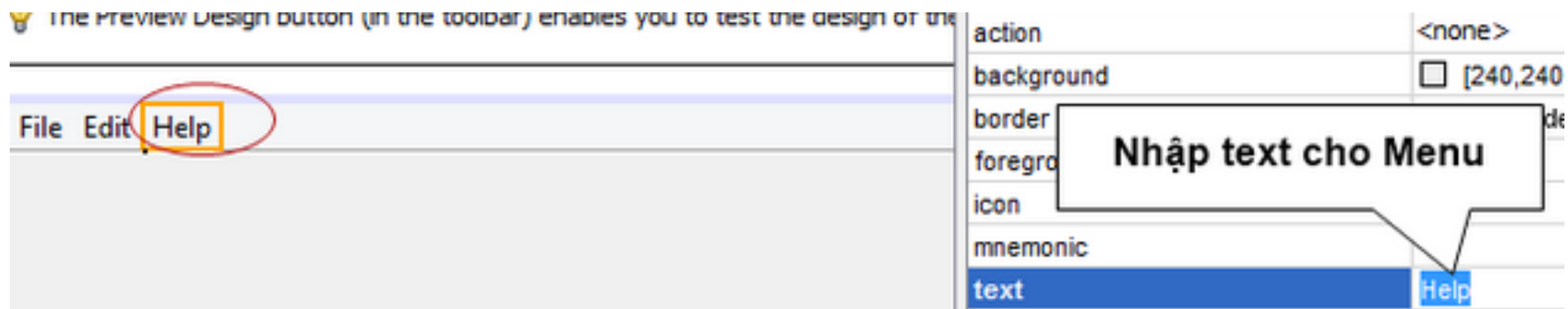
# JMenuBar class

- The JMenuBar class is used to display menubar on the window or frame
- `JFrame f= new JFrame("Menu demo");`
- `JMenuBar bar = new JMenuBar();`
- `f.setJMenuBar(bar);`



# JMenu class

- The object of JMenu class is a pull down menu component which is displayed from the menu bar.
- Constructors:
  - `JMenu()`
  - `JMenu(String label)`
  - `JMenu mfile= new JMenu("File");`
  - `bar.add(mfile);`
  - `mfile.addSeparator();`
  - `mfile.setMnemonic(KeyEvent.VK_F);`



# JMenuItem

## ■ Constructors:

- `JMenuItem()`
- `JMenuItem(Action a)`
- `JMenuItem(Icon icon)`
- `JMenuItem(String text)`
- `JMenuItem(String text, Icon icon)`
- `JMenuItem(String text, int mnemonic)`

## ■ Important methods

- `setEnabled(boolean enable)`
- `setMnemonic(int mnemonic)`
- `setAccelerator(KeyStroke keyStroke)`

## ■ Add menu item to menu

- `menuObject.add(menuItemObject)`

# menu item

File Setting Help

Confirm exit

About Ctrl+Alt+A

Help

About Ctrl+Alt+A

Phím tắt

Ký tự gọi nhớ kết hợp với phím Alt

Text cho menu item

Properties Binding Events

Properties

|                 |               |
|-----------------|---------------|
| action          | <none>        |
| accelerator     | Ctrl+Alt+A    |
| background      | [240,240,240] |
| font            | Segoe UI 12   |
| foreground      | [0,0,0]       |
| icon            | <none>        |
| <b>mnemonic</b> | <b>A</b>      |
| text            | About         |
| toolTipText     |               |

# JCheckBoxMenuItem

## ■ Constructors:

- `JCheckBoxMenuItem()`
- `JCheckBoxMenuItem(Action a)`
- `JCheckBoxMenuItem(String text)`
- `JCheckBoxMenuItem(Icon icon)`
- `JCheckBoxMenuItem(String text, Icon icon)`
- `JCheckBoxMenuItem(String text, boolean b)`
- `JCheckBoxMenuItem(String text, Icon icon, boolean b)`

## ■ Important methods

- `boolean isSelected()`
- `get/ setSelected (boolean)`
- `get/setState (boolean)`

## ■ Add menu item to menu

`menuObject.add (checkBoxMenuItemObject)`

# JCheckBoxMenuItem

## ■ Constructors:

- `JCheckBoxMenuItem()`
- `JCheckBoxMenuItem(Action a)`
- `JCheckBoxMenuItem(String text)`
- `JCheckBoxMenuItem(Icon icon)`
- `JCheckBoxMenuItem(String text, Icon icon)`
- `JCheckBoxMenuItem(String text, boolean b)`
- `JCheckBoxMenuItem(String text, Icon icon, boolean b)`

## ■ Important methods

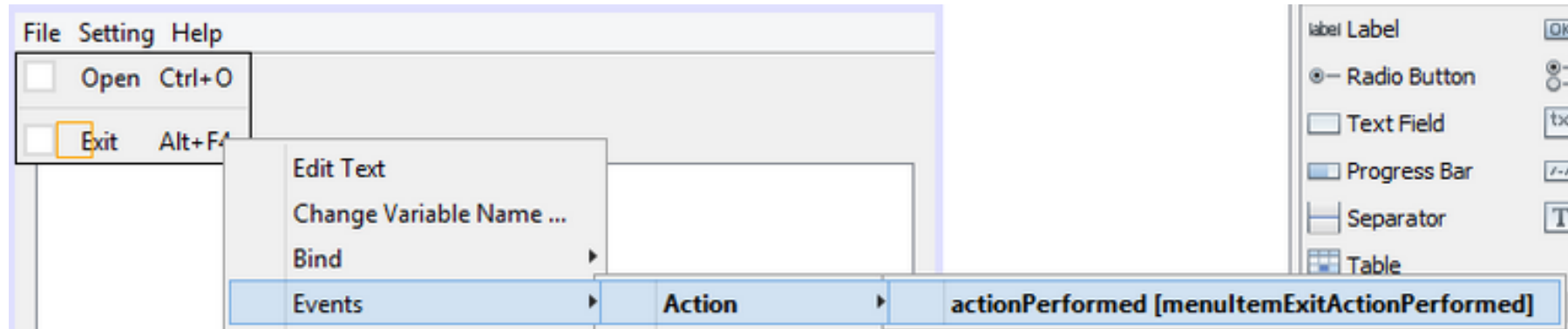
- `boolean isSelected()`
- `get/ setSelected (boolean)`
- `get/setState(boolean)`

## ■ Add menu item to menu

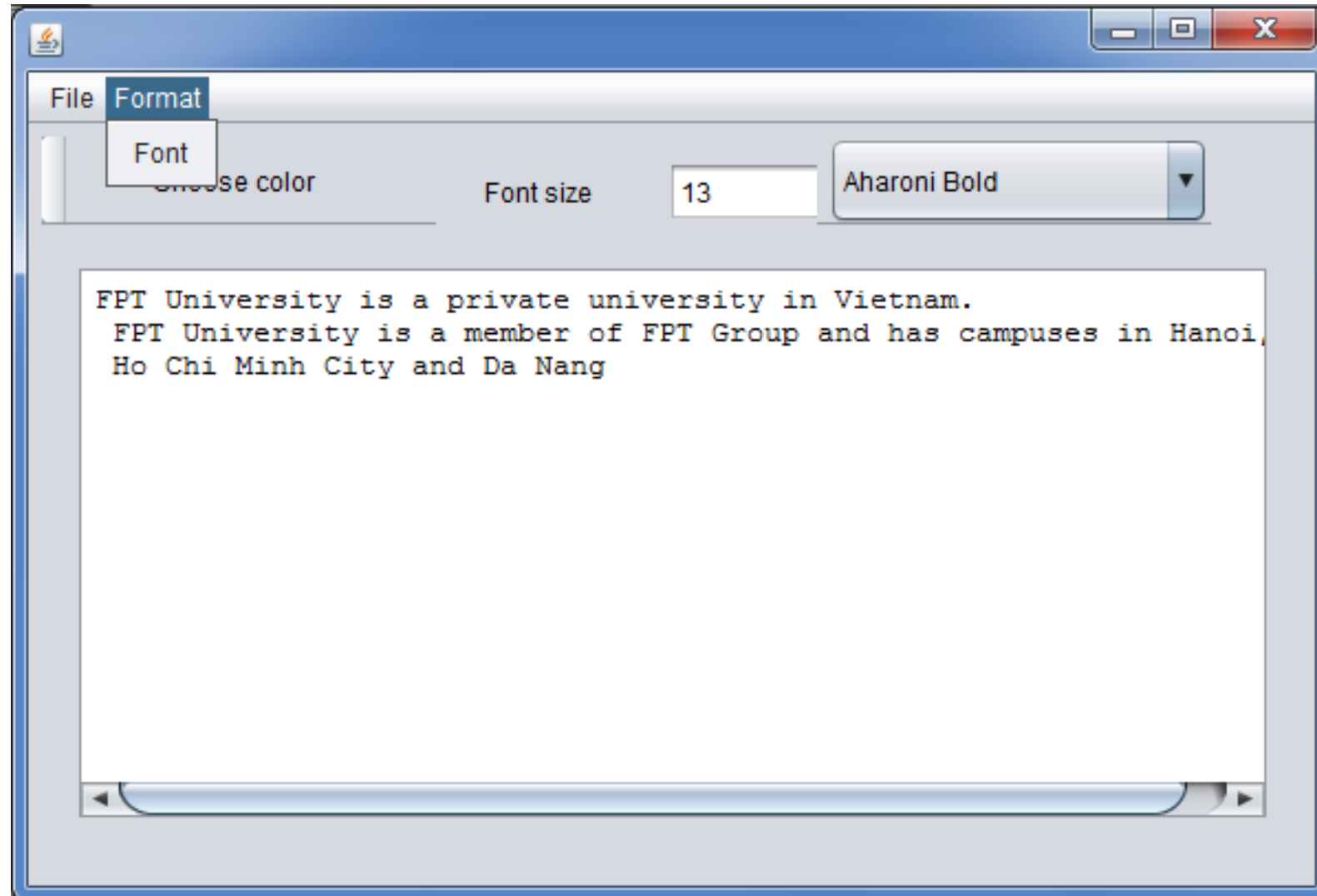
`menuObject.add(checkBoxMenuItemObject)`



# Event for Menu item



# Example



# JTable

- The JTable class is used to display data in tabular form. It is composed of rows and column.
- Constructors - Methods of Jtable
  - JTable( Object[][] entries, Object[] columnNames )
    - constructs a table with a default table model
  - JTable( TableModel model )
    - displays the elements in the specified, non-null table model
  - int **getSelectedRow()**
    - returns the index of the first selected row, -1 if no row is selected
  - Object **getValueAt**( int row, int column )
  - void **setValueAt**( Object value, int row, int column )
    - gets or sets the value at the given row and column
  - int **getRowCount()**
    - returns the number of row in the table

# JTable with changeable choices

- JTable:
- DefaultTableModel
- String[] cols= {"Code", "Name", "number of credits "};
- DefaultTableModel **model**=new DefaultTableModel(cols,0);
- JTable table = new JTable(model);
- JScrollPane pane = new JScrollPane(table);
- Add/remove elements
- Use the **model**, not the JTable directly
- void **addRow**( Object[] rowData )
- void **insertRow**( int row, Object[] rowData
- void **removeRow**( int row )
- void **setValueAt**( Object value, int row, int column )
- Void **fireTableDataChanged**()

# Summary

- **AWT vs Swing**
- GUI Basics design and Top-level container
- Layout Manager, Common Control, Event Listener, Dialogbox
- Advanced Control
- Text component, Choice component, Menu..
- Tabbed pane, Scroll pane
- Dialog box
- Jlist
- Jtable and components in Table

# Case study: TRAIN TICKET MANAGEMENT

- Class Customer includes (code,full name, birth of date,type) – the customer type is Retail, Team, Online, A customer code is a number with 5 digits and auto increasing.
- Class Ticket includes (code, seat type, price), A ticket code is a number with 3 digits and auto increasing.
- Class Bill: each customer can buy one or more tickets. Each seat type, a customer can only buy less than 5 tickets by a date (dd/mm/yyyy) (each customer code and each ticket code is appeared one line in a list of Bills)
- Design the form, named the java file is Main.java that has the menu might look like figure:

|                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div><h3>Data Sort Statistic</h3><div><div><input type="checkbox"/> Customer</div><div>shortcut</div></div><div><div><input type="checkbox"/> Ticket</div><div>shortcut</div></div><div><div><input type="checkbox"/> Bill</div><div>shortcut</div></div></div> <div><p><b>Bill form</b> has only 2 buttons: <b>Add</b> and <b>Save</b> (BILL.DAT)</p></div> | <p>If users select Customer, show <b>Customer form</b>: a table (columns: code, first name, last name, birth of date, type), 3 buttons <b>Add</b>, <b>Save</b> (CUSTOMER.DAT), <b>Edit</b> and textfield for code, full name, birth of date, ComboBox for type.</p> <p>Using the same way above, to select Ticket, <b>Ticket form</b> has 3 buttons <b>Add</b>, <b>Save</b> (TICKET.DAT), <b>Remove</b></p> |
| <div><h3>Data Sort Statistic</h3><div><div><input type="checkbox"/> By Price in Ticket</div><div><input type="checkbox"/> By date in Bill</div><div><input type="checkbox"/> By total money in Bill</div></div></div>                                                                                                                                        | <div><div>1 Sort a list of Tickets by price (descending)</div><div>2 Sort a list of Bills by date (to combine year, month and day)</div><div>3 Sort a list of Bills by total money (total money=number of tickets x price)</div></div> <div><p><b>3 results</b> added to the textarea in Main form</p></div>                                                                                                |
| <div><h3>Data Sort Statistic</h3><div><div><input type="checkbox"/> Sum of total money by year</div></div></div>                                                                                                                                                                                                                                             | <div><div>Output in the textarea in Main form</div><div>Where year extracted from date in Bill</div></div>                                                                                                                                                                                                                                                                                                  |