Advanced Programming

Swings Basic

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Introduction

- The design of the API (**A**pplication **P**rogramming **I**nterfaces) for Java GUI programming is an excellent example of how the object-oriented principle is applied.
- Introduce the framework of Java GUI API
- Discuss GUI (**G**raphic **U**ser **I**nterfaces) components and their relationships, containers and layout managers, colors, fonts, borders, image icons, and tool tips.

GUI example

```
import javax.swing.JFrame;
public class TestFrame {
        public static void main(String[] args) {
                JFrame frame = new JFrame();
                frame.setTitle("Jframe Demo");
                frame.setSize(200, 150);
                frame.setLocation(200, 100);
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setVisible(true);
        }
}
```

Swing vs. AWT

- When Java was introduced, the GUI classes were bundled in a library known as the **Abstract Windows Toolkit** (AWT).
  - AWT is fine for developing simple graphical user interfaces, but not for developing comprehensive GUI projects.
  - AWT is prone to platform-specific bugs.
- The AWT user-interface components were replaced by a more robust, versatile, and flexible library known as **Swing** components.

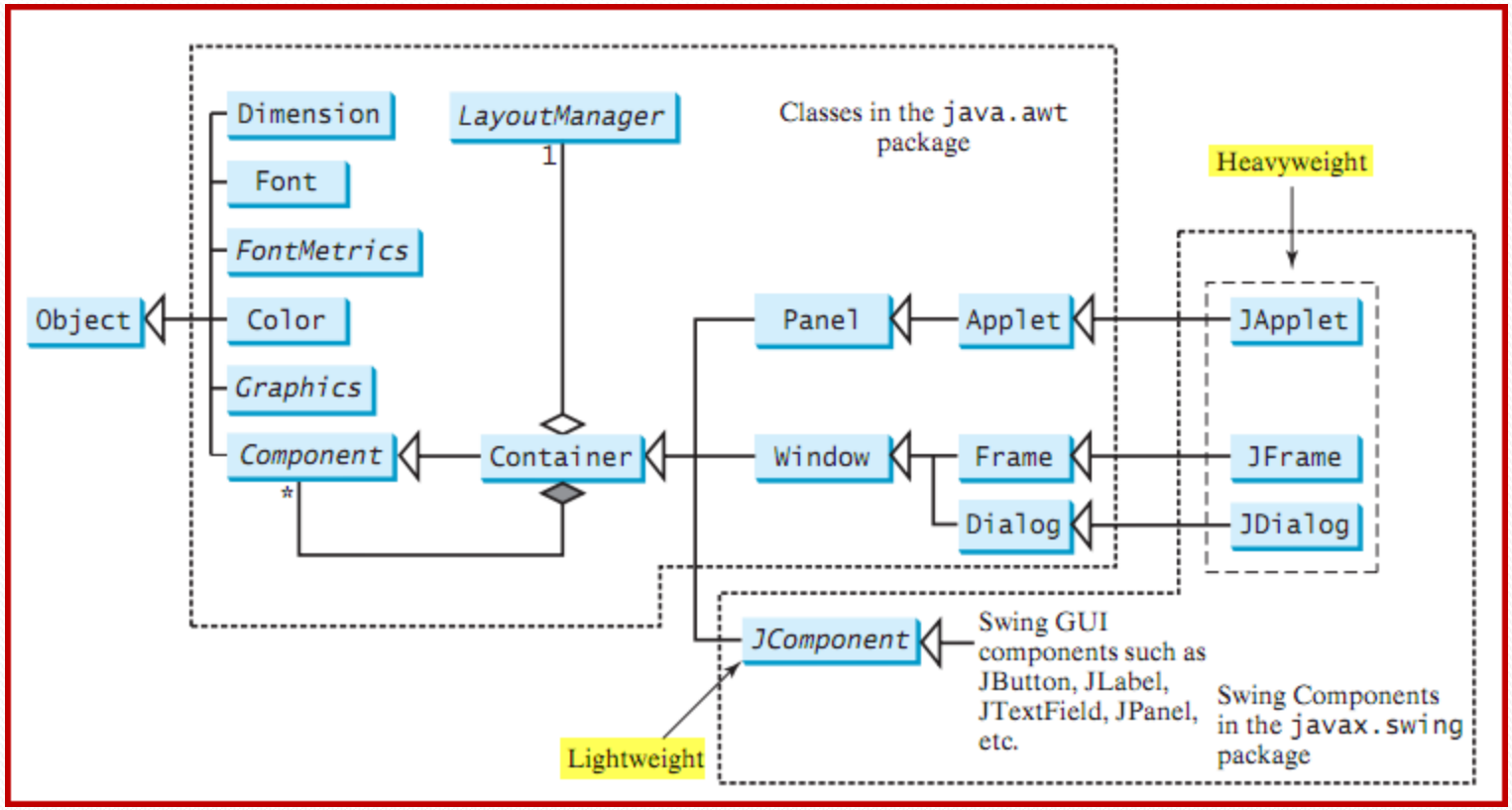
Swing vs. AWT

- **Swing** components depend less on the target platform and use less of the native GUI resource.
  - Swing components that don't rely on native GUI are referred to as **lightweight** components
  - **AWT** components are referred to as **heavyweight** components
- To distinguish new Swing component classes, the **Swing** GUI component classes are named with a prefixed **J**.
- **AWT** components are still supported in Java, it is better to learn to how program using **Swing** components, because the AWT user-interface components will eventually fade away.

The Java GUI API

- The GUI API contains classes that can be classified into three groups:
  - The **component** classes (*JButton, JLabel, and JTextField*), are for **creating** the user interface.
  - The **container** classes (*JFrame, JPanel, and Japplet*), are used to **contain** other components.
  - The **helper** classes (*Graphics, Color, Font, FontMetrics, and Dimension*), are used to **support** GUI components.

The Java GUI API



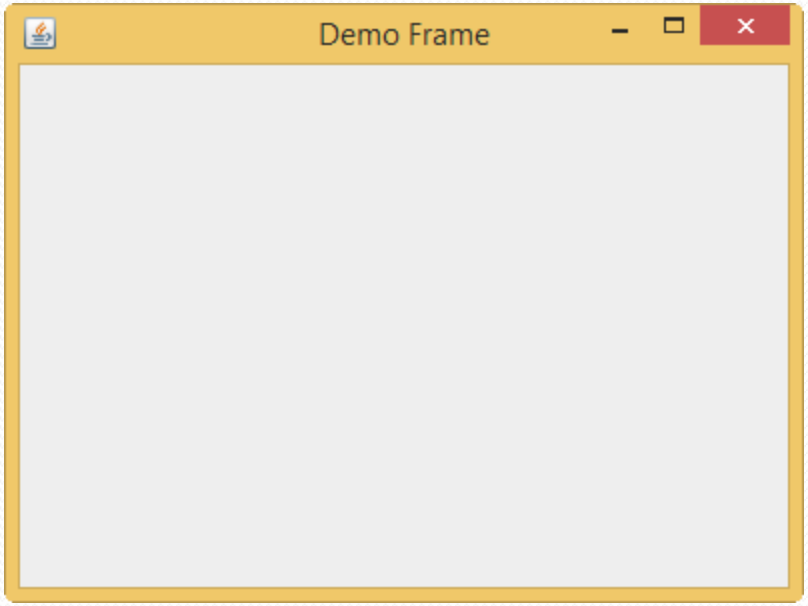
Frames

- To create a user interface, you need to create either a **frame** or an **applet** to hold the user-interface components.

Creating a Frame

```
import javax.swing.JFrame;

public class MyFrame {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Demo Frame");//Create a frame
        frame.setSize(400, 300);//Set the frame size
        frame.setLocationRelativeTo(null);//center the frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

Creating a Frame



Creating a Frame

- To create a frame, use the **JFrame** class:
  - The frame is not displayed until the **frame.setVisible(true)** method is invoked.
  - **frame.setSize(400, 300)** specifies that the frame is 400 pixels wide and 300 pixels high.
  - **setLocationRelativeTo(null)**: centers the frame on the screen.
  - **setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)** tells the program to terminate when the frame is closed.
    - If this statement is not used, the program does not terminate when the frame is closed.
    - You have to stop the program by yourself

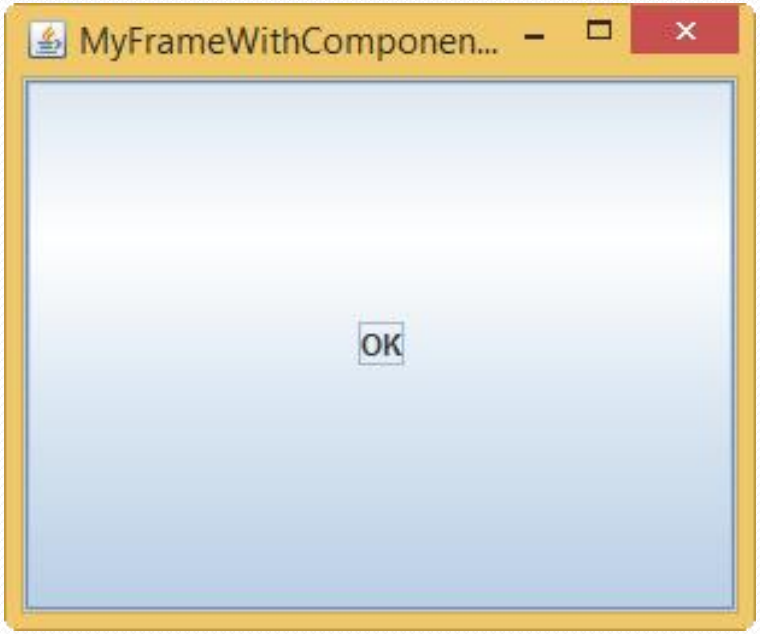
Creating a Frame

javax.swing.JFrame	
+JFrame()	Creates a default frame with no title.
+JFrame(title: String)	Creates a frame with the specified title.
+setSize(width: int, height: int): void	Sets the size of the frame.
+setLocation(x: int, y: int): void	Sets the upper-left-corner location of the frame.
+setVisible(visible: boolean): void	Sets true to display the frame.
+setDefaultCloseOperation(mode: int): void	Specifies the operation when the frame is closed.
+setLocationRelativeTo(c: Component): void	Sets the location of the frame relative to the specified component. If the component is null, the frame is centered on the screen.
+pack(): void	Automatically sets the frame size to hold the components in the frame.

Adding Components to a Frame

```
public class MyFrameWithComponents {
   public static void main(String[] args) {
        JFrame frame = new JFrame("MyFrameWithComponents");
        //Add a button into the frame
        JButton jbtOK = new JButton("OK");
        frame.add(jbtOK);

        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
   }
}
```

Adding Components to a Frame



Adding Components to a Frame

- **JFrame** contains a **content pane**, is an instance of java.awt.Container.
- Components are placed in the *content pane* in a frame.
  - In earlier version of Java:

```
java.awt.Container container = frame.getContentPane();
container.add(jbtOK);
```

Adding Components to a Frame

- Since Java 5 allows you to place components into the content pane by invoking a frame's **add** method:

```
frame.add(jbtOK);
```

- This new feature is called *content-pane delegation*.
  - Strictly speaking, a component is added into the content pane of a frame.
  - For simplicity we say that a component is added to a frame.

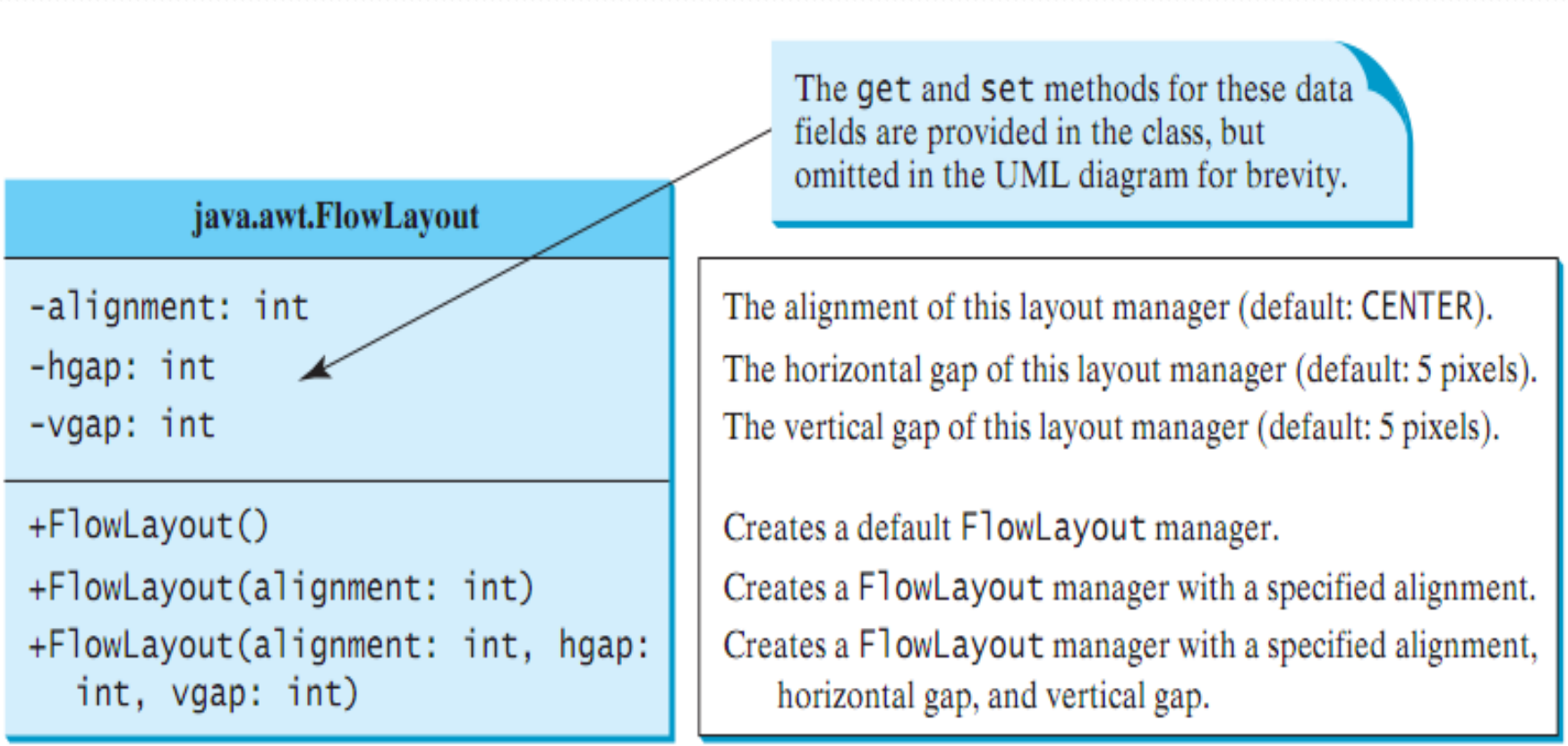
Layout Managers

- FlowLayout
- GridLayout
- BorderLayout
- BoxLayout

FlowLayout

- **FlowLayout** is the simplest layout manager.
  - Components are arranged from *left* to *right* in the order in which they were added.
  - When one row is filled, a new row is started.
- Specify the way the components are *aligned*:
  - FlowLayout.RIGHT
  - FlowLayout.CENTER
  - FlowLayout.LEFT.
- Specify the *gap* between components in pixels.

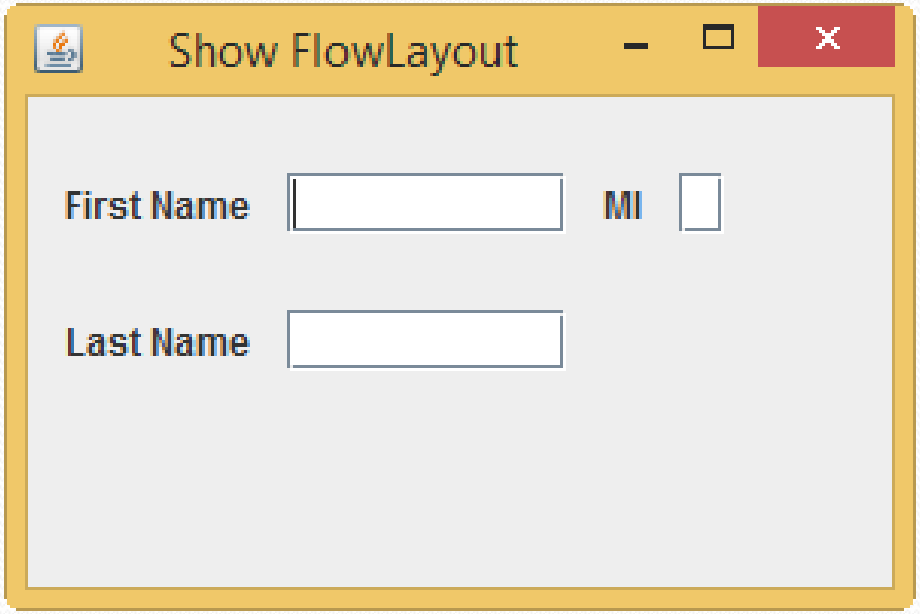
FlowLayout


The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
java.awt.FlowLayout
-alignment: int
-hgap: int
-vgap: int
+FlowLayout()
+FlowLayout(alignment: int)
+FlowLayout(alignment: int, hgap: int, vgap: int)
The alignment of this layout manager (default: CENTER).
The horizontal gap of this layout manager (default: 5 pixels).
The vertical gap of this layout manager (default: 5 pixels).
Creates a default FlowLayout manager.
Creates a FlowLayout manager with a specified alignment.
Creates a FlowLayout manager with a specified alignment, horizontal gap, and vertical gap.

FlowLayout example

```
public class ShowFlowLayout extends JFrame {
    public ShowFlowLayout(){
        setLayout(new FlowLayout(FlowLayout.LEFT, 12, 25));
        add(new JLabel("FirstName")); add(new JTextField(8));
        add(new JLabel("MI")); add(new JTextField(1));
        add(new JLabel("LastName")); add(new JTextField(8));
    }
    public static void main(String[] args) {
        ShowFlowLayout frame = new ShowFlowLayout();
        frame.setTitle("ShowFlowLayout");
        frame.setSize(200, 200);
        frame.setLocationRelativeTo(null);//Center the frame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

FlowLayout Example



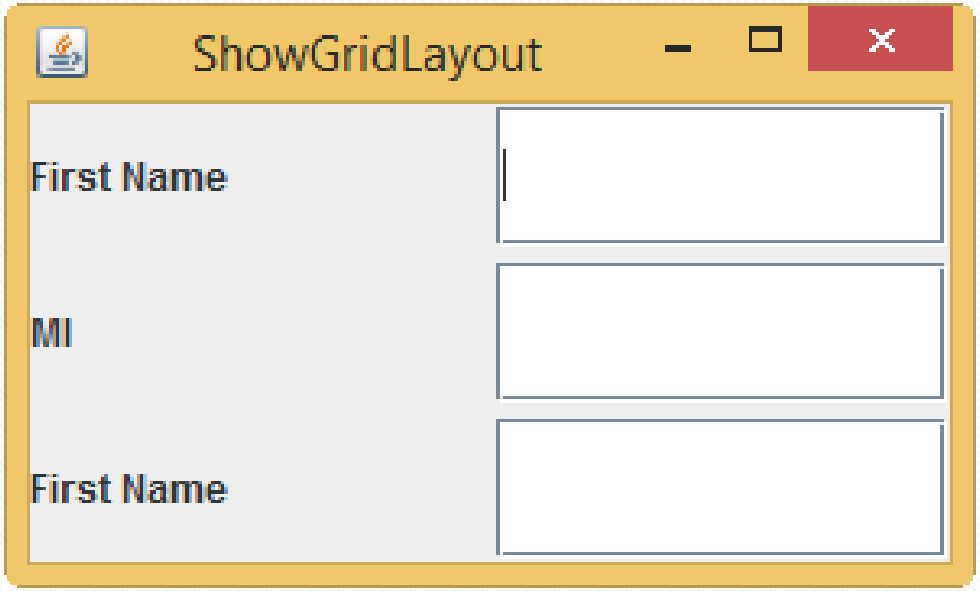
GridLayout

- The **GridLayout** manager arranges components in a grid (matrix) formation.
- The components are placed in the grid from *left to right*, starting with the first row, then the second,... in the order in which they are added.

GridLayout example

```
public class ShowGridLayout extends JFrame {
    public ShowGridLayout(String title) {
       super(title);
       setLayout(new GridLayout(3, 2, 5, 5));
       // Add labels and text fields to the frame
       add(new JLabel("First Name"));
       add(new JTextField(8));
       add(new JLabel("MI"));
       add(new JTextField(1));
       add(new JLabel("First Name"));
       add(new JTextField(8));
       setSize(300, 180);
       setVisible(true);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

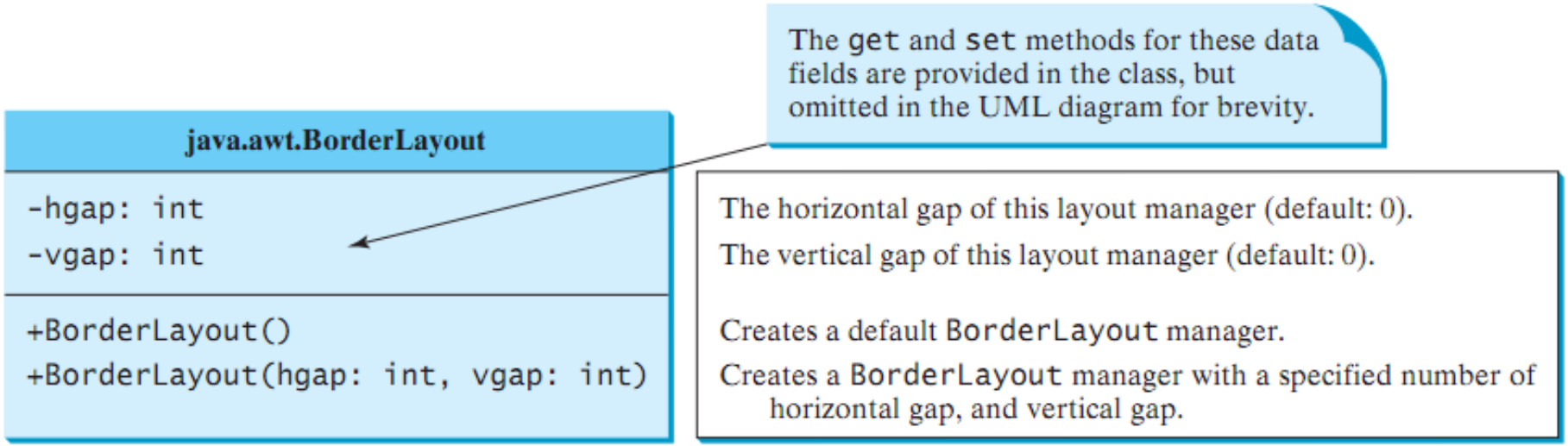
GridLayout



BorderLayout

- The **BorderLayout** manager divides a container into five areas: *East*, *South*, *West*, North, and *Center*.
- Components are added to a BorderLayout by using **add**(Component, *index*), where *index* is a constant:
  - BorderLayout.EAST
  - BorderLayout.SOUTH
  - BorderLayout.WEST
  - BorderLayout.NORTH
  - BorderLayout.CENTER.

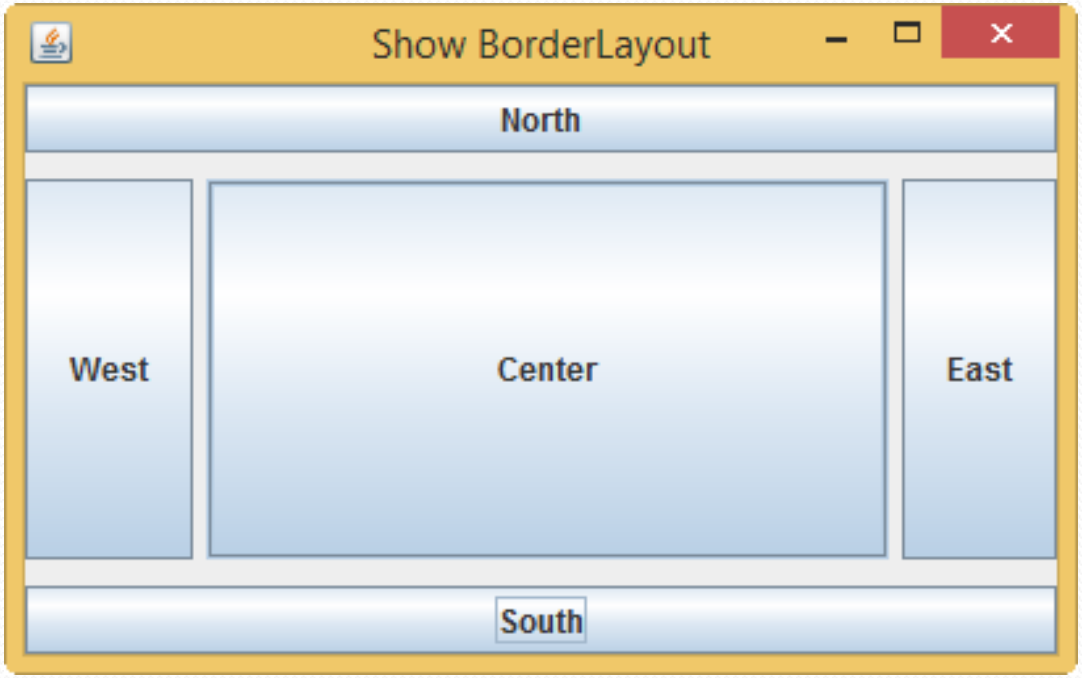
BorderLayout



BorderLayout example

```
public class ShowBorderLayout extends JFrame {
    public ShowBorderLayout(String title) {
        super(title);
        setLayout(new BorderLayout(5, 10));
        add(new JButton("East"), BorderLayout.EAST);
        add(new JButton("South"), BorderLayout.SOUTH);
        add(new JButton("West"), BorderLayout.WEST);
        add(new JButton("North"), BorderLayout.NORTH);
        add(new JButton("Center"), BorderLayout.CENTER);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

BorderLayout



BoxLayout

- **BoxLayout** arranges components in a row or column.
- Constructor:
  public **BoxlayLayout**(**Container** target, **int** axis)
- This constructor is different from other layout constructors.
  - It creates a layout manager that is dedicated to the given target container.
  - The axis parameter is BoxLayout.X_AXIS or BoxLayout.Y_AXIS, which specifies whether the components are laid out horizontally or vertically
- Example:

```
JPanel box1 = new JPanel();
BoxLayout bl1 = new BoxLayout(box1, BoxLayout.X_AXIS);
box1.setLayout(bl1);
```

BoxLayout example

```
public class ShowBoxLayout extends JFrame {
   // Create a label to display flags
   private JLabel jlblFlag = new JLabel();
   // Create image icons for flags
   private ImageIcon imageIconAus = new
        ImageIcon(getClass().getResource("image/aus.gif"));
   private ImageIcon imageIconUS = new
        ImageIcon(getClass().getResource("image/us.gif"));
   private ImageIcon imageIconCanada = new
        ImageIcon(getClass().getResource("image/ca.gif"));
   private ImageIcon imageIconNorway = new
        ImageIcon(getClass().getResource("image/norway.gif"));
   private ImageIcon imageIconGermany = new
        ImageIcon(getClass().getResource("image/germany.gif"));
   private ImageIcon imageIconPrint = new
        ImageIcon(getClass().getResource("image/print.gif"));
   private ImageIcon imageIconSave = new
        ImageIcon(getClass().getResource("image/save.gif"));
```

BoxLayout example

```
// Create buttons to select images
private JButton jbtUS = new JButton("US             ");
private JButton jbtCanada = new JButton("Canada   ");
private JButton jbtAus = new JButton("Australia");
private JButton jbtNorway = new JButton("Norway   ");
private JButton jbtGermany = new JButton("Germany");
```

```
public ShowBoxLayout() {
    JPanel box1 = new JPanel();
    BoxLayout bl1 = new BoxLayout(box1, BoxLayout.X_AXIS);
    box1.setLayout(bl1);

    JPanel box2 = new JPanel();
    BoxLayout bl2 = new BoxLayout(box2, BoxLayout.Y_AXIS);
    box2.setLayout(bl2);

    box1.add(new JButton(imageIconPrint));
    box1.add(new JButton(imageIconSave));
    box2.add(jbtUS);
    box2.add(jbtCanada);
    box2.add(jbtAus);
    box2.add(jbtNorway);
    box2.add(jbtGermany);
```

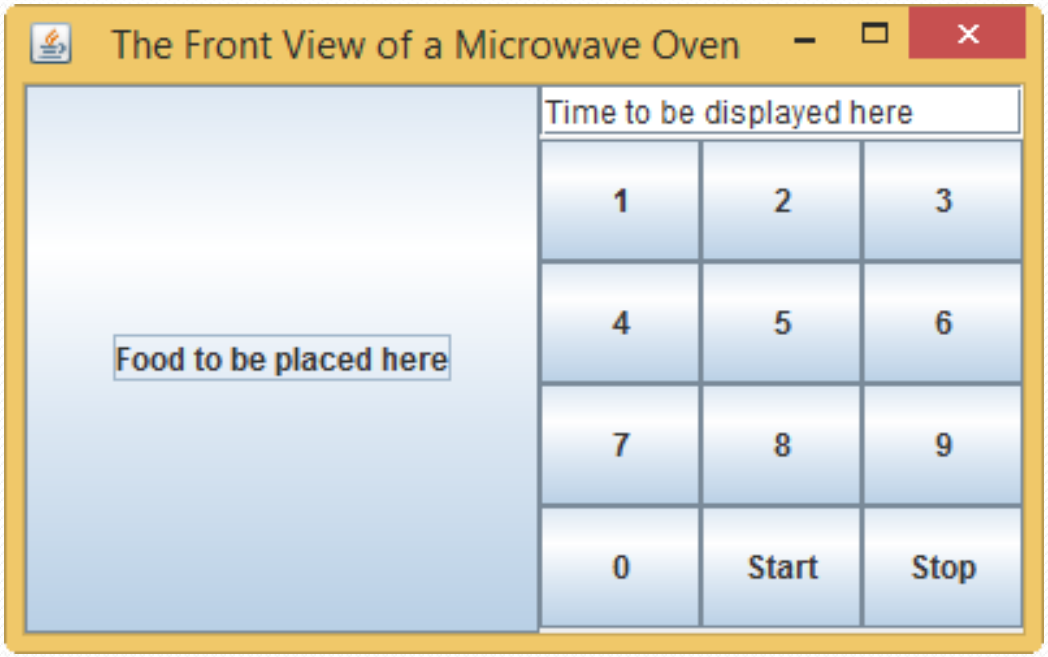
BoxLayout example

```
        box1.setBorder(new
                javax.swing.border.LineBorder(Color.red));
        box2.setBorder(new
                javax.swing.border.LineBorder(Color.black));
        add(box1, BorderLayout.NORTH);
        add(box2, BorderLayout.EAST);
        add(jlblFlag, BorderLayout.CENTER);
    }
}
```

BoxLayout example

Using Panels as Subcontainers

- Suppose that you want to place ten buttons and a text field in a frame.



Using Panels as Subcontainers

- You can divide a window into **panels**.
  - Panels act as subcontainers to group user-interface components.
  - You add the buttons in one panel, then add the panel into the frame.
  - The following code creates a panel and adds a button to it:

```
JPanel p = new JPanel();
p.add(new JButton("OK"));
```

  - Panels can be placed inside a frame or inside another panel.

```
f.add(p); //f is a frame
```

TestPanels.java

```
public class TestPanels extends JFrame{
   public TestPanels() {
       // create panel p1 for the buttons and setGridlayout
       JPanel p1 = new JPanel();
       p1.setLayout(new GridLayout(4, 3));
       // add buttons to p1
       for (int i = 1; i <= 9; i++)
              p1.add(new JButton("" + i));

       p1.add(new JButton("0"));
       p1.add(new JButton("Start"));
       p1.add(new JButton("Stop"));
```

TestPanels.java

```
        //create panel p2 to hold a textfiled and p1
        JPanel p2 = new JPanel(new BorderLayout());
        p2.add(new JTextField("Time to be displayed here"),
                BorderLayout.NORTH);
        p2.add(p1, BorderLayout.CENTER);

        //add contents into the frame
        add(p2, BorderLayout.WEST);
        add(new JButton("Food to be placed here"),
                        BorderLayout.CENTER);
    }
}
```

Image Icons

- An icon is a fixed-size picture; typically it is small and used to decorate components. Images are normally stored in image files.
- Java currently supports three image formats:
  - GIF (Graphics Interchange Format),
  - JPEG (Joint Photographic Experts Group),
  - PNG (Portable Network Graphics).

Image Icons

- To display an image icon, first create an **ImageIcon** object:

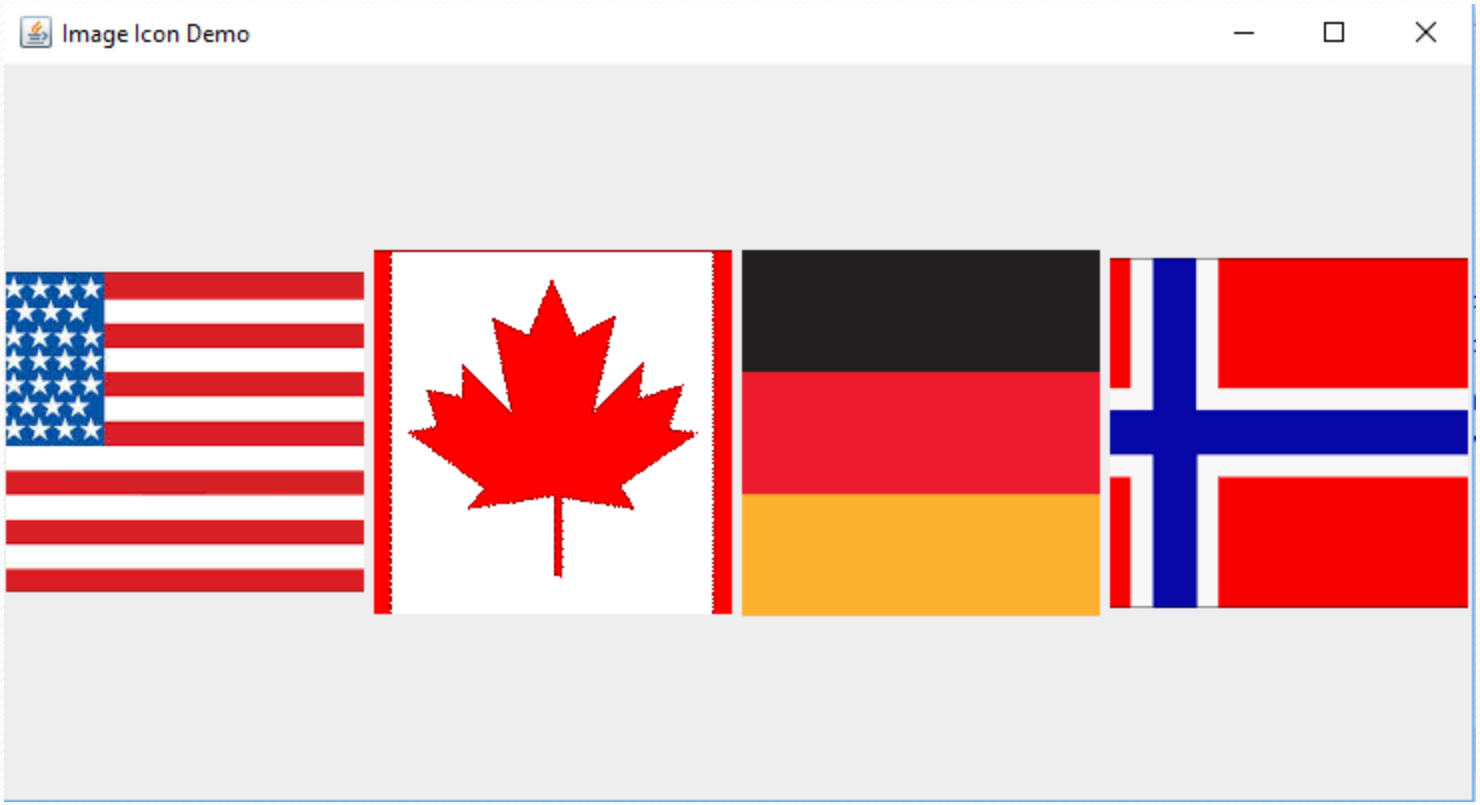
  ImageIcon icon = **new ImageIcon**("**image/us.gif**");

- An image icon can be displayed in a **label** or a **button** using:
  - **new JLabel**(icon)
  - or **new JButton**(icon)

TestImageIcon.java

```
public class TestImageIcon extends JFrame {
   private ImageIcon usIcon = new ImageIcon("images/us.gif");
   private ImageIcon caIcon = new ImageIcon("images/ca.gif");
   private ImageIcon geIcon = new ImageIcon("images/germany.gif");
   private ImageIcon noIcon = new ImageIcon("images/norway.gif");

   public TestImageIcon() {
        setLayout(new GridLayout(1, 4, 5, 5));
        add(new JLabel(usIcon));
        add(new JLabel(caIcon));
        add(new JLabel(geIcon));
        add(new JLabel(noIcon));
   }
}
```

Image Icons



Advanced Programming

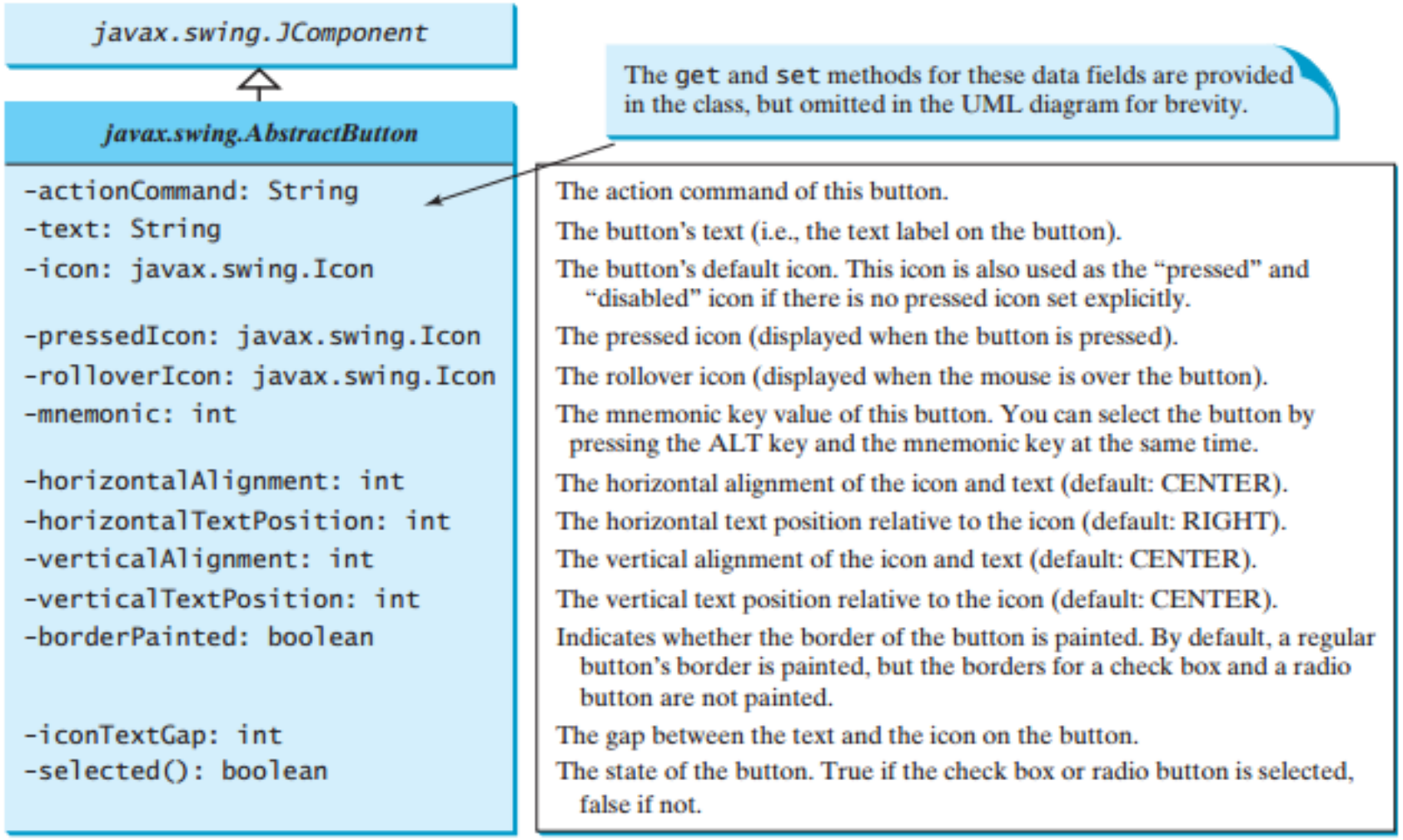
GUI Components

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

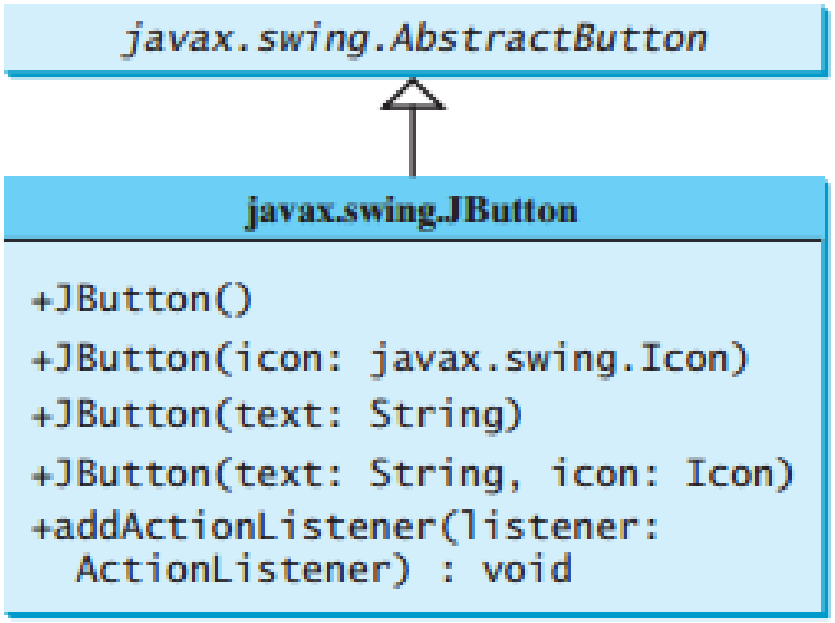
JButton

- A *button* is a component that triggers an action event when clicked.
- Swing provides regular buttons, toggle buttons, check box buttons, and radio buttons.
- The common features of these buttons are defined in **javax.swing. AbstractButton**

JButton


javax.swing.JComponent
javax.swing.AbstractButton
-actionCommand: String
-text: String
-icon: javax.swing.Icon
-pressedIcon: javax.swing.Icon
-rolloverIcon: javax.swing.Icon
-mnemonic: int
-horizontalAlignment: int
-horizontalTextPosition: int
-verticalAlignment: int
-verticalTextPosition: int
-borderPainted: boolean
-iconTextGap: int
-selected(): boolean
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The action command of this button.
The button's text (i.e., the text label on the button).
The button's default icon. This icon is also used as the "pressed" and "disabled" icon if there is no pressed icon set explicitly.
The pressed icon (displayed when the button is pressed).
The rollover icon (displayed when the mouse is over the button).
The mnemonic key value of this button. You can select the button by pressing the ALT key and the mnemonic key at the same time.
The horizontal alignment of the icon and text (default: CENTER).
The horizontal text position relative to the icon (default: RIGHT).
The vertical alignment of the icon and text (default: CENTER).
The vertical text position relative to the icon (default: CENTER).
Indicates whether the border of the button is painted. By default, a regular button's border is painted, but the borders for a check box and a radio button are not painted.
The gap between the text and the icon on the button.
The state of the button. True if the check box or radio button is selected, false if not.

JButton

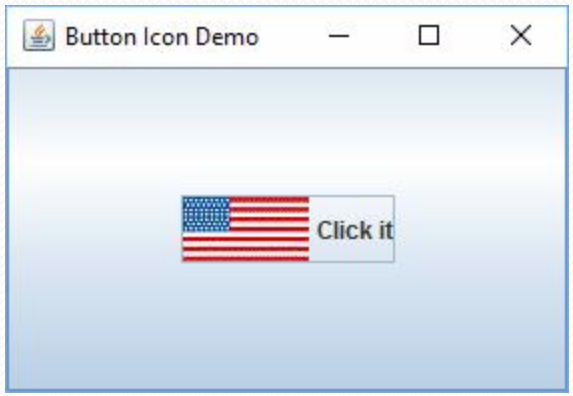


+JButton()	Creates a default button with no text and icon.
+JButton(icon: javax.swing.Icon)	Creates a button with an icon.
+JButton(text: String)	Creates a button with text.
+JButton(text: String, icon: Icon)	Creates a button with text and an icon.
+addActionListener(listener: ActionListener) : void	Adds an `ActionListener` for the button.

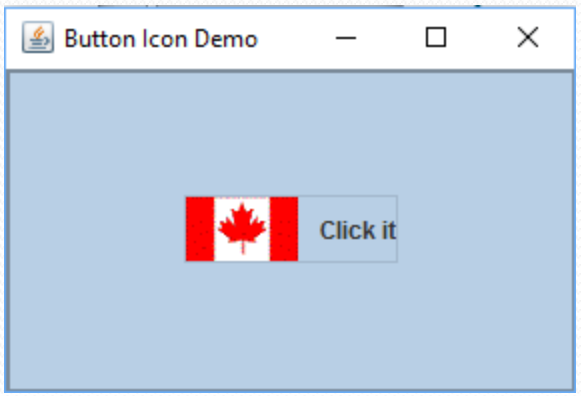
TestButtonIcon.java

```
public class TestButtonIcon extends JFrame {
    public TestButtonIcon() {
        ImageIcon usIcon = new ImageIcon("images/smallUs.gif");
        ImageIcon caIcon = new ImageIcon("images/smallCa.gif");
        ImageIcon geIcon = new ImageIcon("images/smallGermany.gif");

        JButton jbtClick = new JButton("Click it", usIcon);
        jbtClick.setPressedIcon(caIcon);
        jbtClick.setRolloverIcon(geIcon);

        add(jbtClick);
    }
}
```

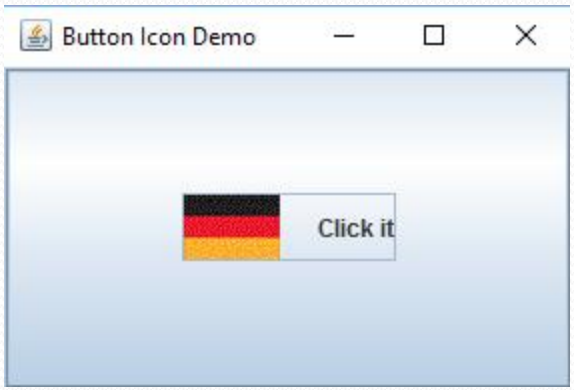
Icons, Pressed Icons, Rollover icons



Default Icon



Pressed Icon

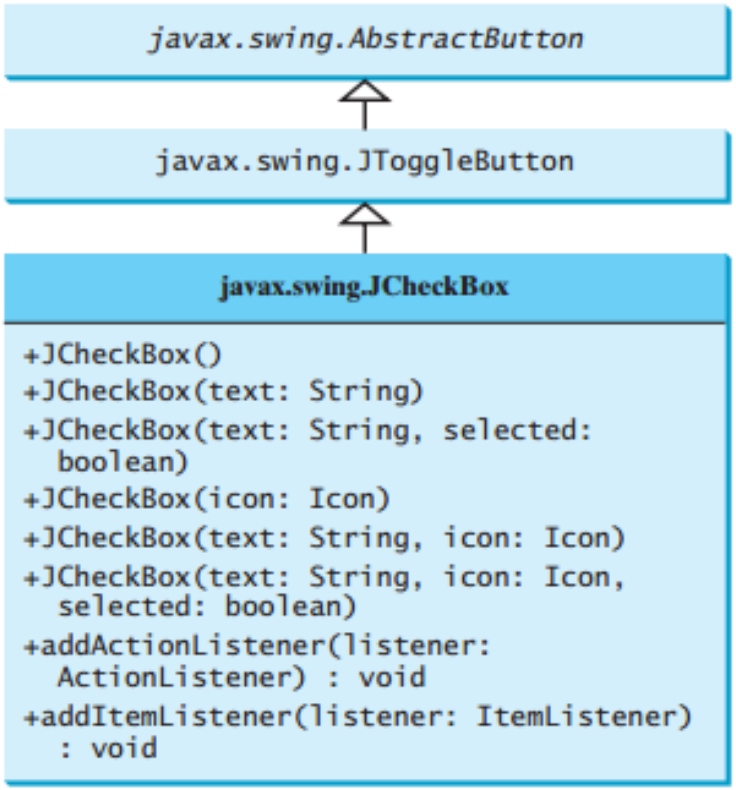


Rollover Icon

JCheckBox

- A *toggle button* is a two-state button like a light switch. **JToggleButton** inherits **AbstractButton** and implements a toggle button.
- Often **JToggleButton**'s subclasses **JCheckBox** and **JRadioButton** are used to enable the user to toggle a choice on or of

JCheckBox

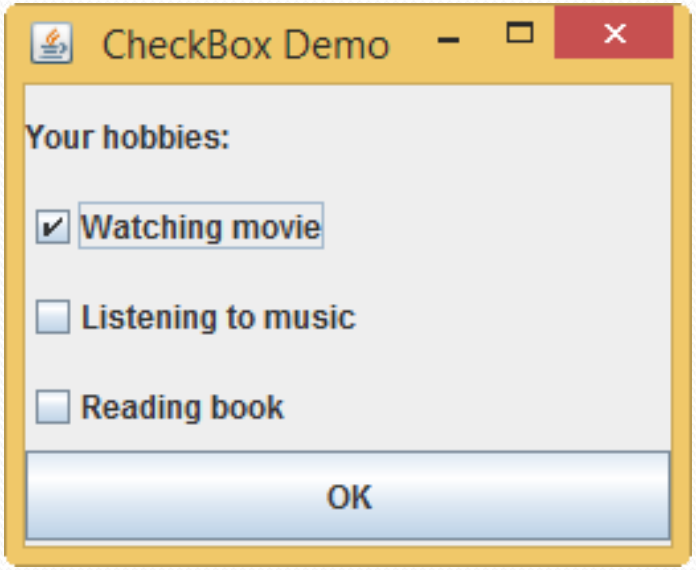


Creates a default check box button with no text and icon.	
Creates a check box with text.	
Creates a check box with text and specifies whether the check box is initially selected.	
Creates a checkbox with an icon.	
Creates a checkbox with text and an icon.	
Creates a check box with text and an icon, and specifies whether the check box is initially selected.	
Adds an `ActionListener` for this object.	
Adds an `ItemListener` for this object.	

JCheckBox example

```
public class CheckBoxDemo extends JFrame {
    public CheckBoxDemo(String title) {
        super(title); setLayout(new GridLayout(5, 2));
        JLabel lblHobbies = new JLabel("Your hobbies:");
        add(lblHobbies);

        JCheckBox cboMovie = new JCheckBox("Watching movie");
        JCheckBox cboMusic = new JCheckBox("Listening to music");
        JCheckBox cboRead = new JCheckBox("Reading book");
        add(cboMovie); add(cboMusic); add(cboRead);

        JButton btnOK = new JButton("OK"); add(btnOK);
        pack(); setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

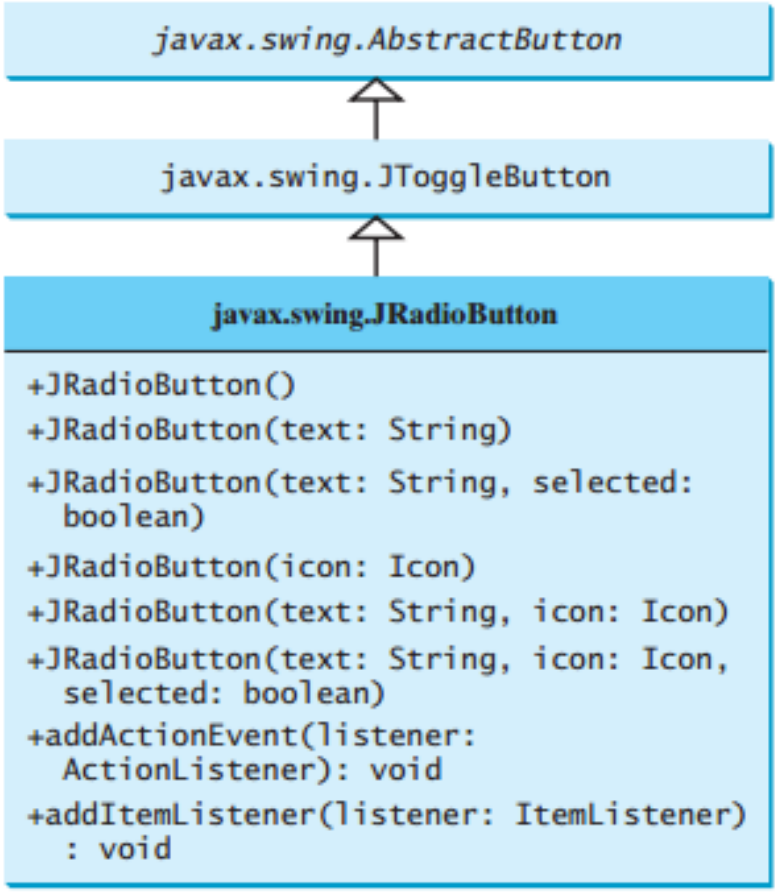
JCheckBox example



JRadioButton

- *Radio buttons*, also known as *option buttons*, enable you to choose a single item from a group of choices

JRadioButton

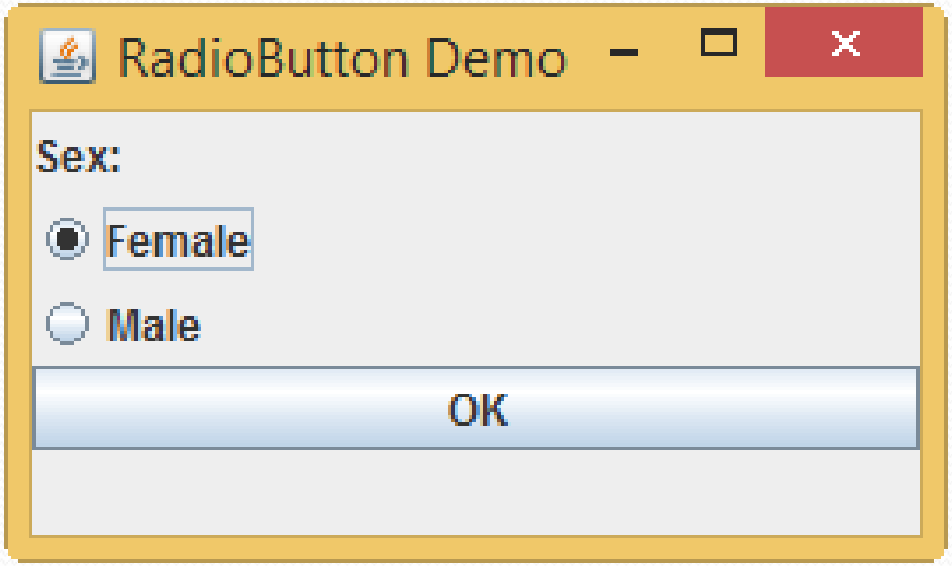


Creates a default radio button with no text and icon.	
Creates a radio button with text.	
Creates a radio button with text and specifies whether the radio button is initially selected.	
Creates a radio button with an icon.	
Creates a radio button with text and an icon.	
Creates a radio button with text and an icon, and specifies whether the radio button is initially selected.	
Adds an `ActionListener` for this object.	
Adds an `ItemListener` for this object.	

JRadioButton example

```
public class RadioButtonDemo extends JFrame {
    public RadioButtonDemo(String title) {
        super(title);
        setLayout(new GridLayout(5, 2));
        JLabel lblSex = new JLabel("Sex:"); add(lblSex);

        JRadioButton rbFem = new JRadioButton("Female");
        JRadioButton rbMal = new JRadioButton("Male");
        ButtonGroup bg = new ButtonGroup();
        bg.add(rbFem); bg.add(rbMal);
        add(rbFem); add(rbMal);

        JButton btnOK = new JButton("OK"); add(btnOK);
        pack(); setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

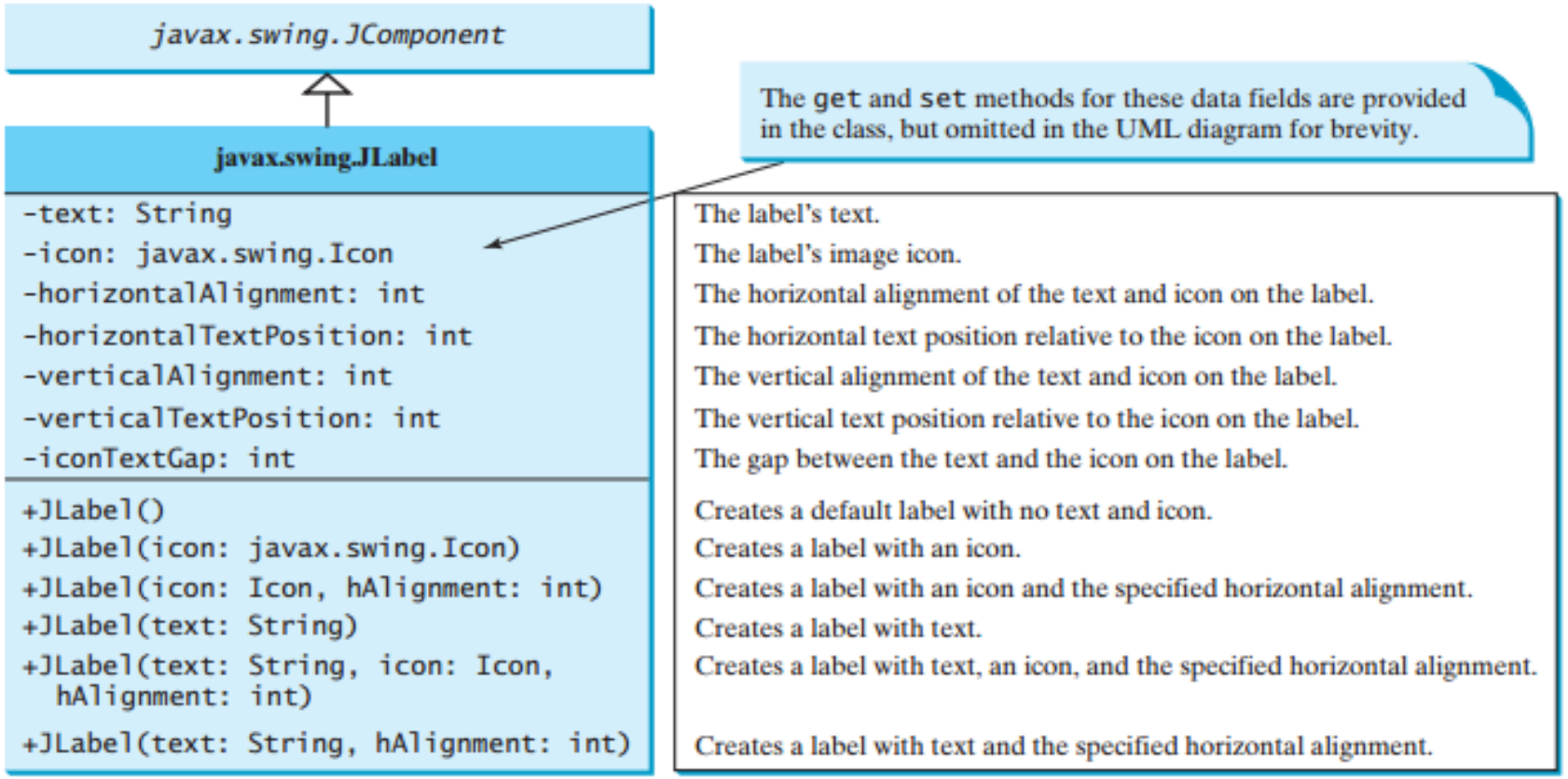
JRadioButton example



JLabel

- A *label* is a display area for a short text, an image, or both.
- It is often used to label other components (usually text fields).

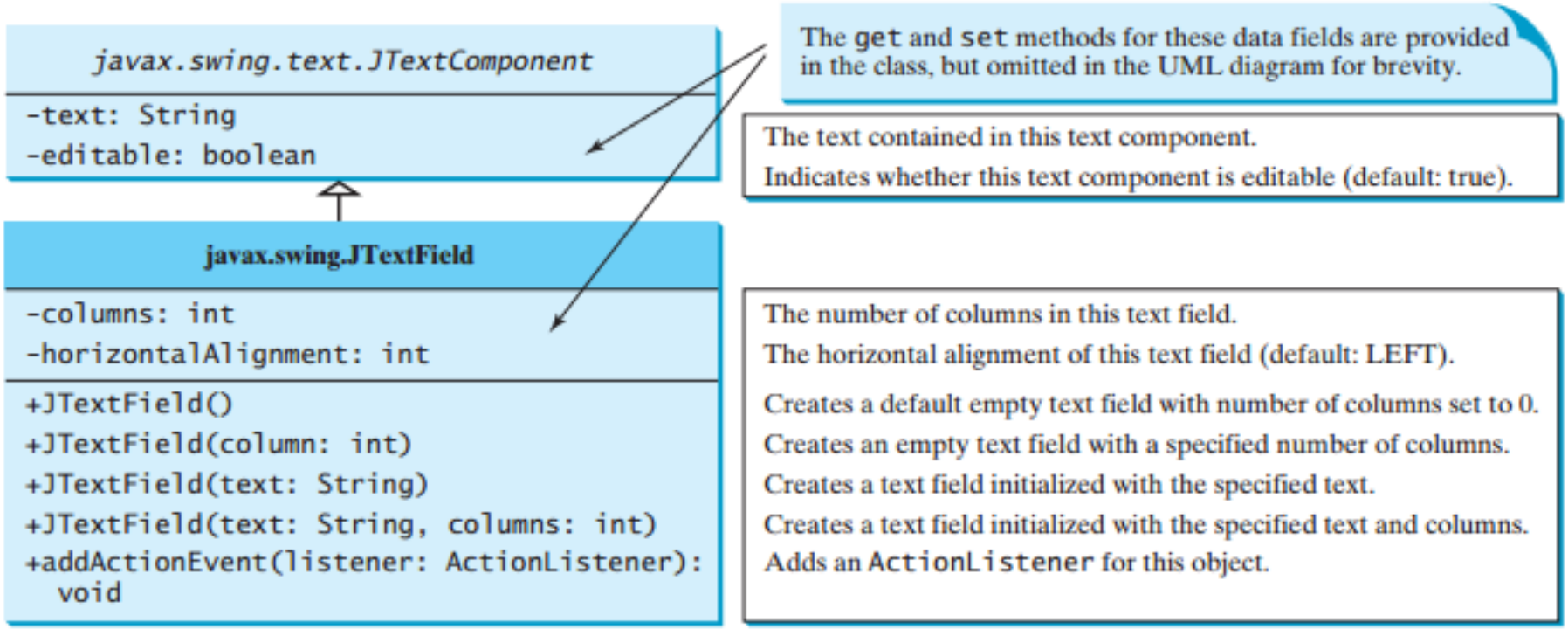
JLabel


javax.swing.JComponent
javax.swing.JLabel
-text: String
-icon: javax.swing.Icon
-horizontalAlignment: int
-horizontalTextPosition: int
-verticalAlignment: int
-verticalTextPosition: int
-iconTextGap: int
+JLabel()
+JLabel(icon: javax.swing.Icon)
+JLabel(icon: Icon, hAlignment: int)
+JLabel(text: String)
+JLabel(text: String, icon: Icon, hAlignment: int)
+JLabel(text: String, hAlignment: int)
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The label's text.
The label's image icon.
The horizontal alignment of the text and icon on the label.
The horizontal text position relative to the icon on the label.
The vertical alignment of the text and icon on the label.
The vertical text position relative to the icon on the label.
The gap between the text and the icon on the label.
Creates a default label with no text and icon.
Creates a label with an icon.
Creates a label with an icon and the specified horizontal alignment.
Creates a label with text.
Creates a label with text, an icon, and the specified horizontal alignment.
Creates a label with text and the specified horizontal alignment.

JTextField

- A *text field* can be used to enter or display a string. **JTextField** is a subclass of **JTextComponent**

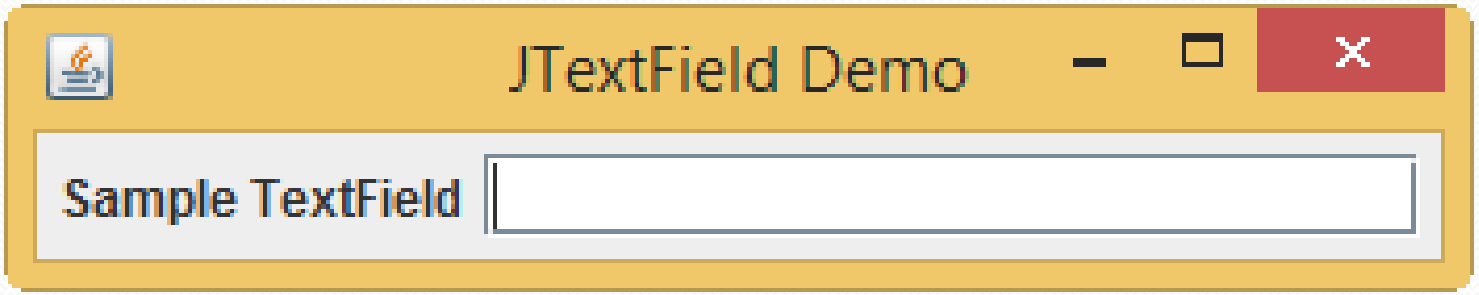
JTextField


javax.swing.text.JTextComponent
-text: String
-editable: boolean
javax.swing.JTextField
-columns: int
-horizontalAlignment: int
+JTextField()
+JTextField(column: int)
+JTextField(text: String)
+JTextField(text: String, columns: int)
+addActionEvent(listener: ActionListener): void
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The text contained in this text component.
Indicates whether this text component is editable (default: true).
The number of columns in this text field.
The horizontal alignment of this text field (default: LEFT).
Creates a default empty text field with number of columns set to 0.
Creates an empty text field with a specified number of columns.
Creates a text field initialized with the specified text.
Creates a text field initialized with the specified text and columns.
Adds an ActionListener for this object.

JTextField example

```
public class TextFieldDemo extends JFrame {
    public TextFieldDemo(String title) {
        super(title);
        setLayout(new FlowLayout());
        JLabel lbl = new JLabel("Sample TextField");
        add(lbl);
        JTextField txt = new JTextField(20);
        add(txt);
        pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    public static void main(String args[]) {
        new TextFieldDemo("Sample JTextField");
    }
}
```

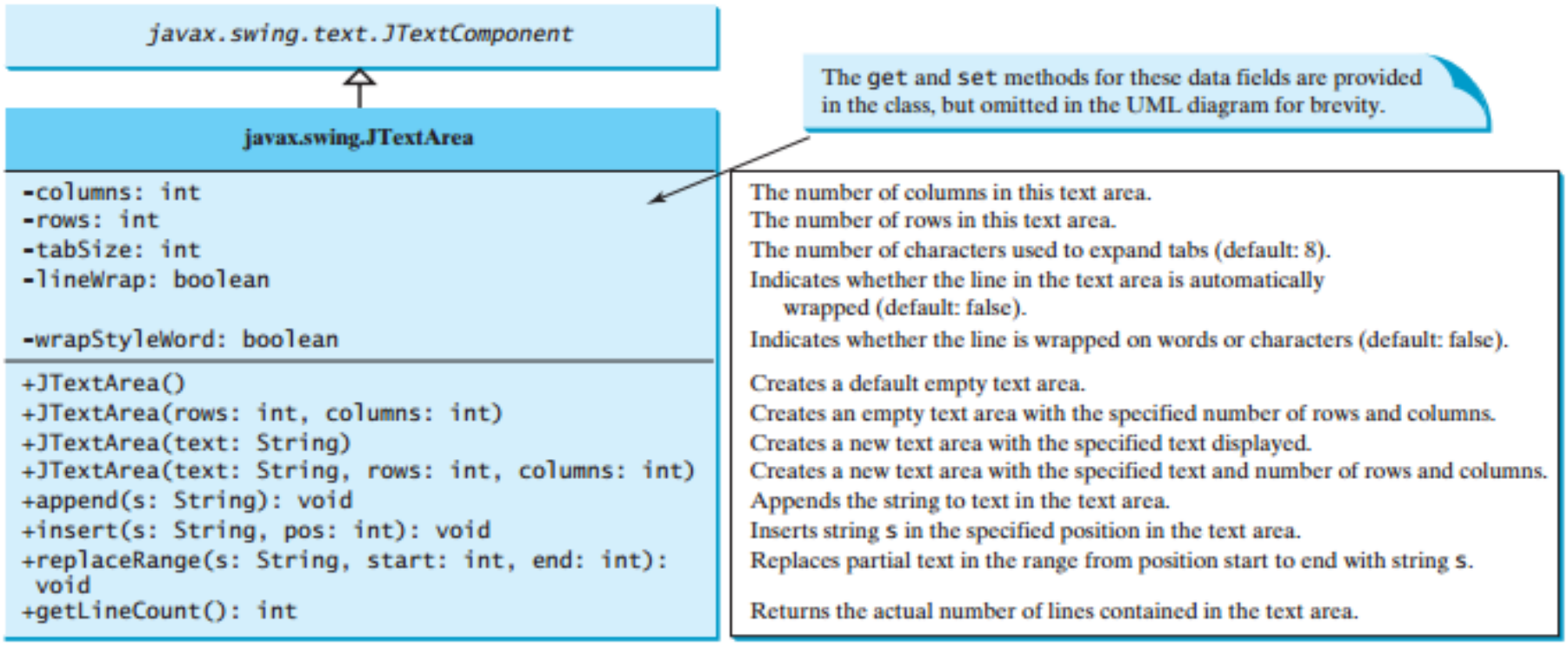
JTextField example



JTextArea

- If you want to let the user enter multiple lines of text ➔ use **JTextArea**, which enables the user to enter multiple lines of text.

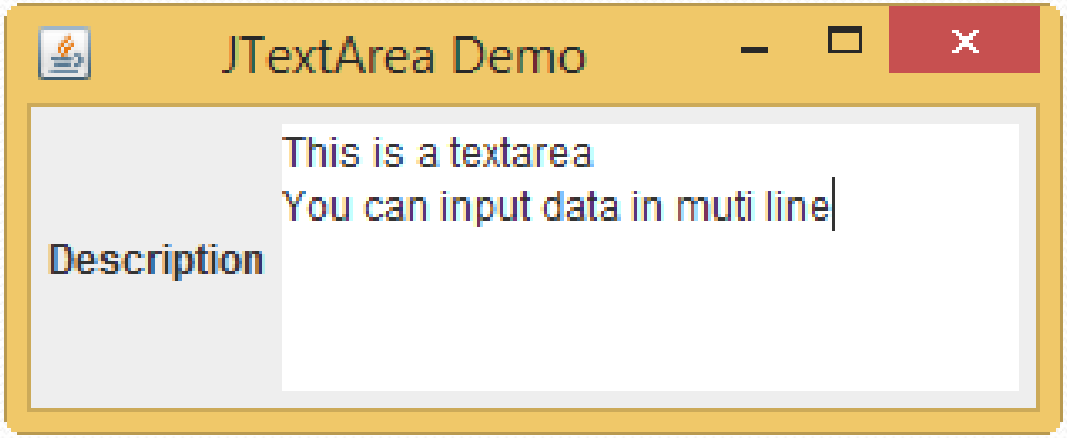
JTextArea


javax.swing.text.JTextComponent
javax.swing.JTextArea
-columns: int
-rows: int
-tabSize: int
-lineWrap: boolean
-wrapStyleWord: boolean
+JTextArea()
+JTextArea(rows: int, columns: int)
+JTextArea(text: String)
+JTextArea(text: String, rows: int, columns: int)
+append(s: String): void
+insert(s: String, pos: int): void
+replaceRange(s: String, start: int, end: int): void
+getLineCount(): int
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The number of columns in this text area.
The number of rows in this text area.
The number of characters used to expand tabs (default: 8).
Indicates whether the line in the text area is automatically wrapped (default: false).
Indicates whether the line is wrapped on words or characters (default: false).
Creates a default empty text area.
Creates an empty text area with the specified number of rows and columns.
Creates a new text area with the specified text displayed.
Creates a new text area with the specified text and number of rows and columns.
Appends the string to text in the text area.
Inserts string s in the specified position in the text area.
Replaces partial text in the range from position start to end with string s.
Returns the actual number of lines contained in the text area.

JTextArea example

```
public class TextAreaDemo extends JFrame {
    public TextAreaDemo(String title) {
        super(title);
        setLayout(new FlowLayout());
        JLabel lbl = new JLabel("Description");
        add(lbl);
        JTextArea txt = new JTextArea(5, 20);
        add(txt);
        pack();
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String args[]) {
        new TextAreaDemo("JTextArea Demo");
    }
}
```

JTextArea example



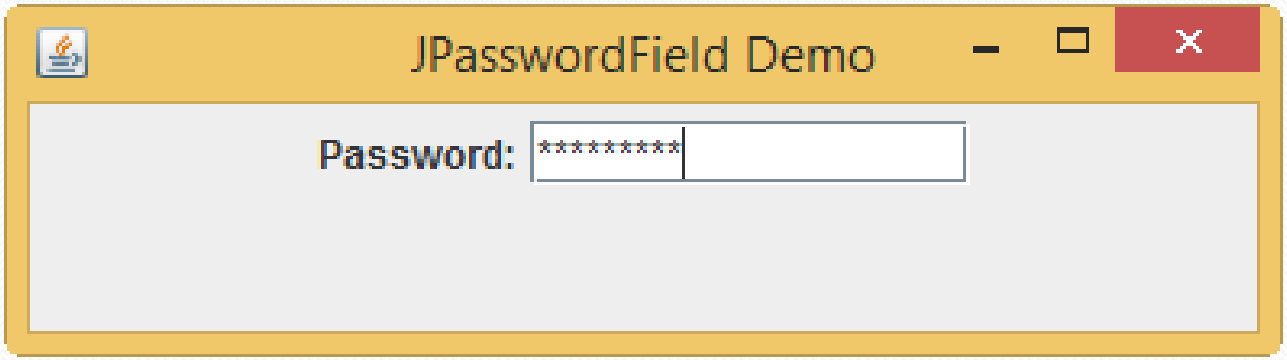
JPasswordField

- It is extend of JTextField
- Hide all the characters that user inputs in the textfield
- Constructor:
  - JPasswordField()
  - JPasswordField(int colums)
  - JPasswordField (String text)
  - JPasswordField(String text, int column)
  - JPasswordField(Document doc, String text, int column)

JPasswordField example

```
public class PasswordFieldDemo extends JFrame {
    public PasswordFieldDemo(String title) {
        super(title);
        setLayout(new FlowLayout());

        JLabel lblPassword = new JLabel("Password:");
        add(lblPassword);
        JPasswordField tfPassword = new JPasswordField(12);
        tfPassword.setEchoChar('*');
        add(tfPassword);

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

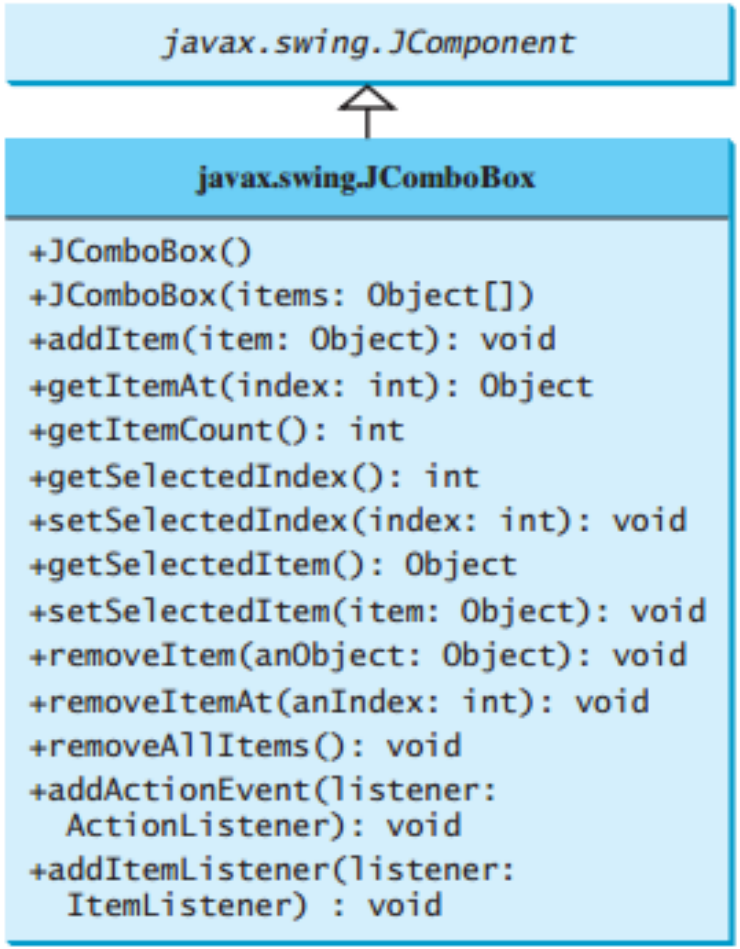
JPasswordField example



JComboBox

- A *combo box*, also known as a *choice list* or *drop-down list*, contains a list of items from which the user can choose.
- It is useful in limiting a user's range of choices and avoids the cumbersome validation of data input

JComboBox

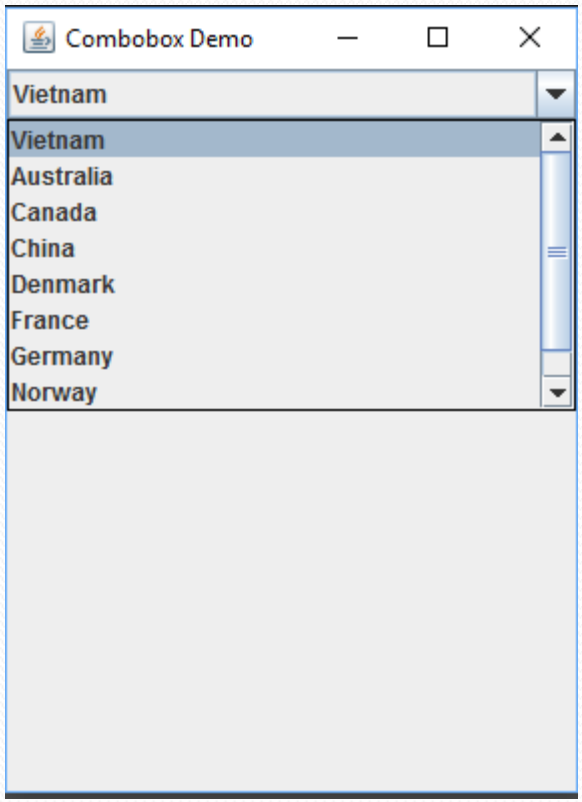


Creates a default empty combo box.	
Creates a combo box that contains the elements in the specified array.	
Adds an item to the combo box.	
Returns the item at the specified index.	
Returns the number of items in the combo box.	
Returns the index of the selected item.	
Sets the selected index in the combo box.	
Returns the selected item.	
Sets the selected item in the combo box.	
Removes an item from the item list.	
Removes the item at the specified index in the combo box.	
Removes all the items in the combo box.	
Adds an `ActionListener` for this object.	
Adds an `ItemListener` for this object.	

JComboBox example

```
public class ComboBoxDemo extends JFrame{
    private String[] flagTitles = { "Vietnam", "Australia",
"Canada", "China", "Denmark", "France", "Germany", "Norway",
"United Kingdom" };
    private JComboBox cbo = new JComboBox(flagTitles);

   public ComboBoxDemo() {
        add(cbo, BorderLayout.NORTH);
   }
}
```

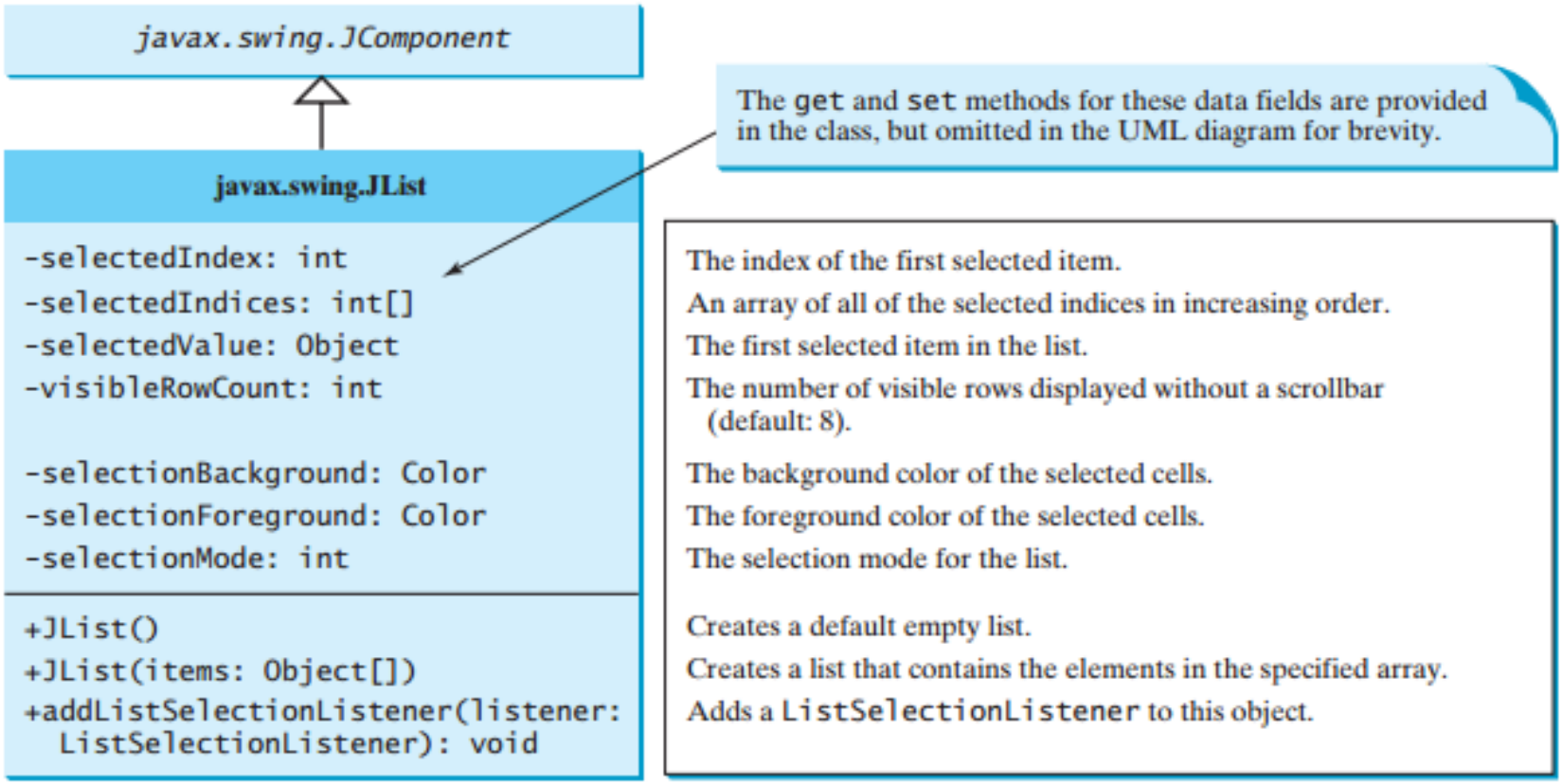
JComboBox


Combobox Demo
Vietnam
Vietnam
Australia
Canada
China
Denmark
France
Germany
Norway

JList

- A list is a component that basically performs the same function as a **combo box** but enables the user to choose a **single** value or **multiple** values

JList


javax.swing.JComponent
javax.swing.JList
-selectedIndex: int
-selectedIndices: int[]
-selectedValue: Object
-visibleRowCount: int
-selectionBackground: Color
-selectionForeground: Color
-selectionMode: int
+JList()
+JList(items: Object[])
+addListSelectionListener(listener: ListSelectionListener): void
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
The index of the first selected item.
An array of all of the selected indices in increasing order.
The first selected item in the list.
The number of visible rows displayed without a scrollbar (default: 8).
The background color of the selected cells.
The foreground color of the selected cells.
The selection mode for the list.
Creates a default empty list.
Creates a list that contains the elements in the specified array.
Adds a ListSelectionListener to this object.

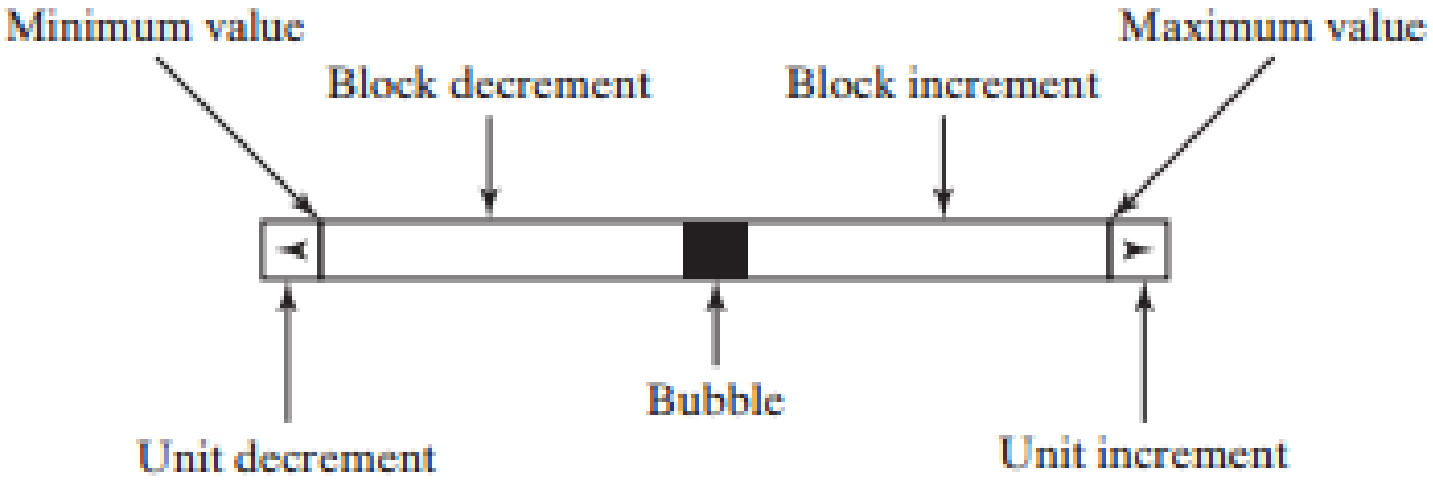
JList example

```
public class ListDemo extends JFrame{
   private String[] flagTitles = { "Vietnam", "Australia", "Canada",
"China", "Denmark", "France", "Germany", "Norway", "United Kingdom" };
   private JList jlst = new JList(flagTitles);

   public ListDemo() {
         add(new JLabel("Please choose a country:"), BorderLayout.NORTH);
         add(jlst, BorderLayout.CENTER);
         add(new JButton("OK"), BorderLayout.SOUTH);
   }
}
```

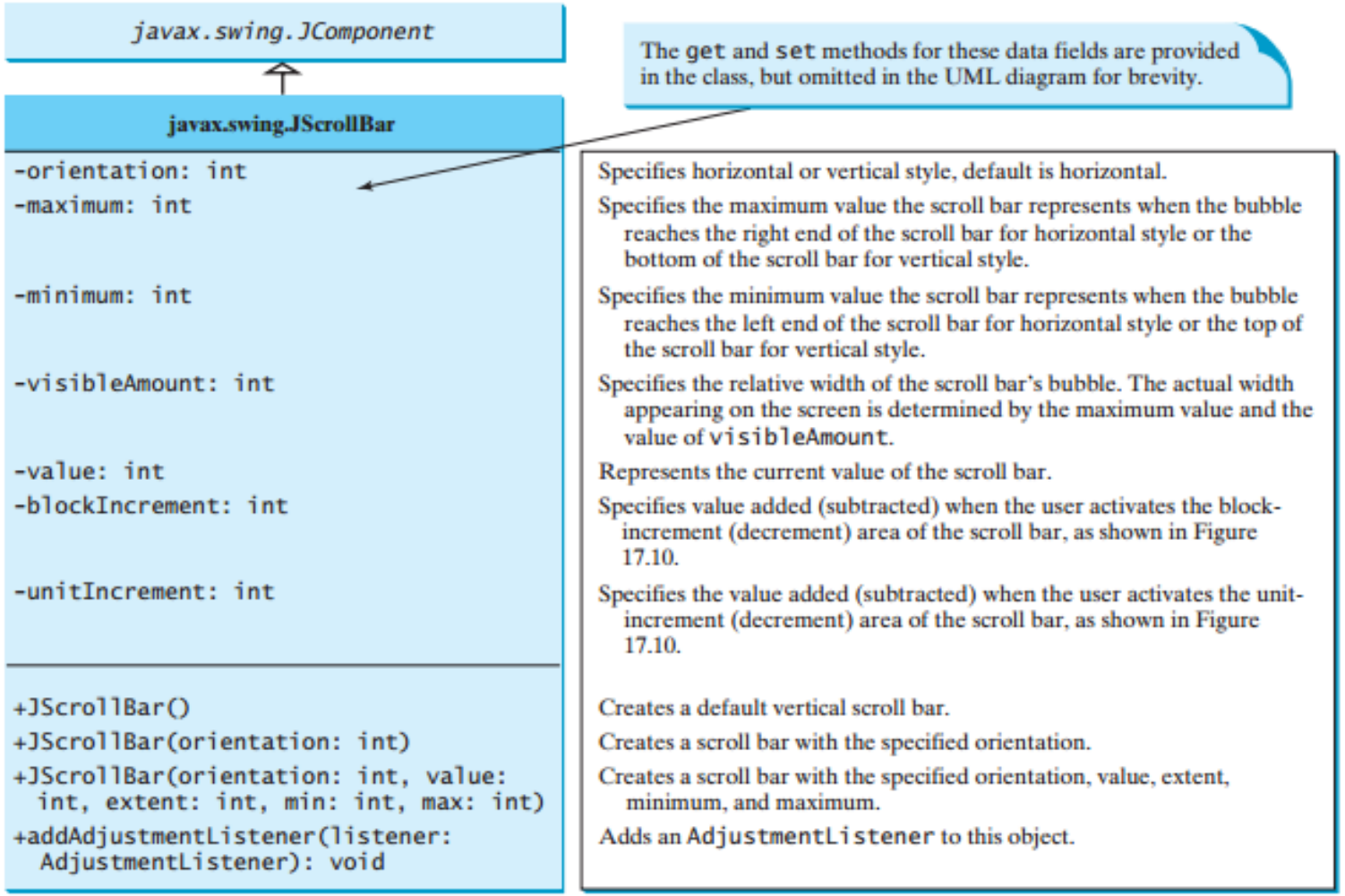
JList example

Scroll bars

- **JScrollBar** *is a component that enables the user to select from a range of values.*



JScrollBar


javax.swing.JComponent
javax.swing.JScrollBar
-orientation: int
-maximum: int
-minimum: int
-visibleAmount: int
-value: int
-blockIncrement: int
-unitIncrement: int
+JScrollBar()
+JScrollBar(orientation: int)
+JScrollBar(orientation: int, value: int, extent: int, min: int, max: int)
+addAdjustmentListener(listener: AdjustmentListener): void
The get and set methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.
Specifies horizontal or vertical style, default is horizontal.
Specifies the maximum value the scroll bar represents when the bubble reaches the right end of the scroll bar for horizontal style or the bottom of the scroll bar for vertical style.
Specifies the minimum value the scroll bar represents when the bubble reaches the left end of the scroll bar for horizontal style or the top of the scroll bar for vertical style.
Specifies the relative width of the scroll bar's bubble. The actual width appearing on the screen is determined by the maximum value and the value of visibleAmount.
Represents the current value of the scroll bar.
Specifies value added (subtracted) when the user activates the block-increment (decrement) area of the scroll bar, as shown in Figure 17.10.
Specifies the value added (subtracted) when the user activates the unit-increment (decrement) area of the scroll bar, as shown in Figure 17.10.
Creates a default vertical scroll bar.
Creates a scroll bar with the specified orientation.
Creates a scroll bar with the specified orientation, value, extent, minimum, and maximum.
Adds an AdjustmentListener to this object.

ScrollBar example

```
public class ScrollBarDemo extends JFrame{
   public ScrollBarDemo() {
         JScrollBar hbar = new JScrollBar(JScrollBar.HORIZONTAL, 30, 20, 0,
200);
         JScrollBar vbar = new JScrollBar(JScrollBar.VERTICAL, 30, 40, 0,
200);

         add(hbar, BorderLayout.SOUTH);
         add(vbar, BorderLayout.EAST);
         add(new JLabel(new ImageIcon("images/flower.jpg")),
                  BorderLayout.CENTER);
   }
}
```

JScrollbar

Advanced Programming

Menu

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Menus

- Menus make selection easier and are widely used in window applications.
- 5 classes that implement menus:
  - JMenuBar
  - JMenu
  - JMenuItem
  - JCheckBoxMenuItem
  - JRadioButtonMenuItem

Menus

- **JMenuBar** is a top-level menu component used to hold the menus.
  - A **menu** consists of **menu items** that the user can select (or toggle on or off).
  - A menu item can be an instance of
    - JMenuItem,
    - JCheckBoxMenuItem
    - JRadioButtonMenuItem.
  - Menu items can be associated with icons, keyboard mnemonics, and keyboard accelerators.
  - Menu items can be separated using separators.

1. Creating Menu Bar

- Create a **menu bar** and associate it with a **frame** or an **applet** by using the **setJMenuBar** method.

```
JFrame frame = new JFrame();
JMenuBar jmb = new JMenuBar();
frame.setJMenuBar(jmb);
frame.setSize(300, 200);
frame.setVisible(true);
```

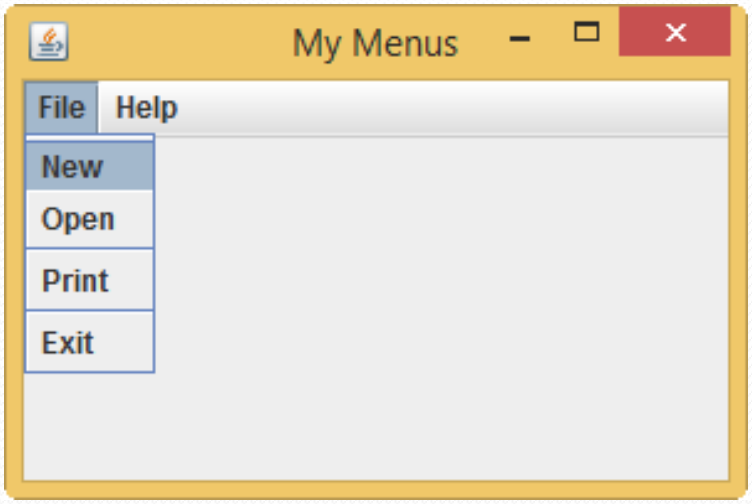
2. Creating Menus

- Create menus and asociate them with the menu bar.

```
JMenu fileMenu = new JMenu("File");
JMenu helpMenu = new JMenu("Help");
jmb.add(fileMenu);
jmb.add(helpMenu);
```

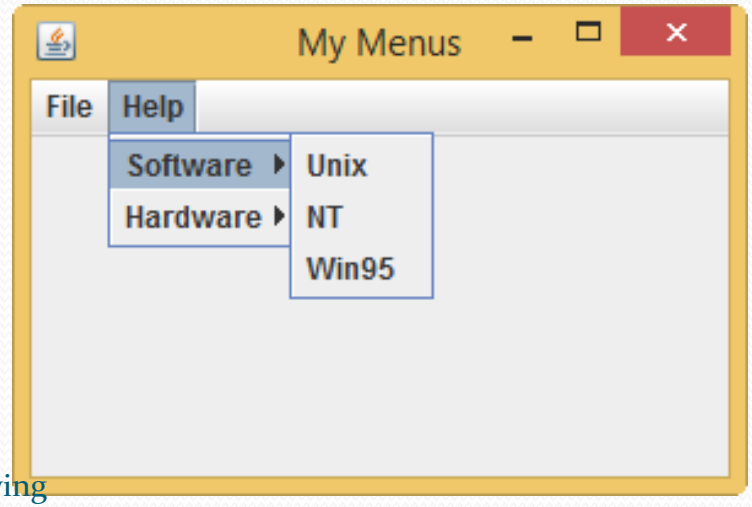
3. Creating menu items

```
fileMenu.add(new JMenuItem("New"));
fileMenu.add(new JMenuItem("Open"));
fileMenu.addSeparator();
fileMenu.add(new JMenuItem("Print"));
fileMenu.addSeparator();
fileMenu.add(new JMenuItem("Exit"));
```



3.1. Creating submenu items

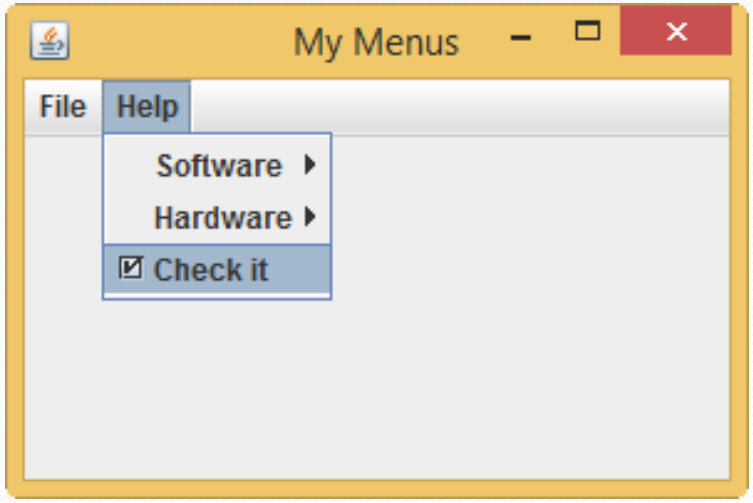
```
JMenu softwareHelpSubMenu = new JMenu("Software");
JMenu hardwareHelpSubMenu = new JMenu("Hardware");
helpMenu.add(softwareHelpSubMenu);
helpMenu.add(hardwareHelpSubMenu);
softwareHelpSubMenu.add(new JMenuItem("Unix"));
softwareHelpSubMenu.add(new JMenuItem("NT"));
softwareHelpSubMenu.add(new JMenuItem("Win95"));
```



3.2. Creating check-box menu items

- You can add a **JCheckBoxMenuItem** to a JMenu.
- **JCheckBoxMenuItem** is a subclass of **JMenuItem** that adds a boolean state to the **JMenuItem**, and displays a check when its state is true.

```
helpMenu.add(new JCheckBoxMenuItem("Check it"));
```



3.3. Creating radio-button menu items

- This is often useful when you have a group of mutually exclusive choices in the menu.

```
JMenu colorHelpSububMenu = new JMenu("Color");
helpMenu.add(colorHelpSububMenu);

JRadioButtonMenuItem jrmiBlue, jrmiYellow, jrmiRed;
colorHelpSububMenu.add(jrmiBlue = new JRadioButtonMenuItem("Blue"));
colorHelpSububMenu.add(jrmiYellow = new JRadioButtonMenuItem("Yellow"));
colorHelpSububMenu.add(jrmiRed = new JRadioButtonMenuItem("Red"));

ButtonGroup btg = new ButtonGroup();
btg.add(jrmiBlue);
btg.add(jrmiYellow);
btg.add(jrmiRed);
```

Radio-button menu example

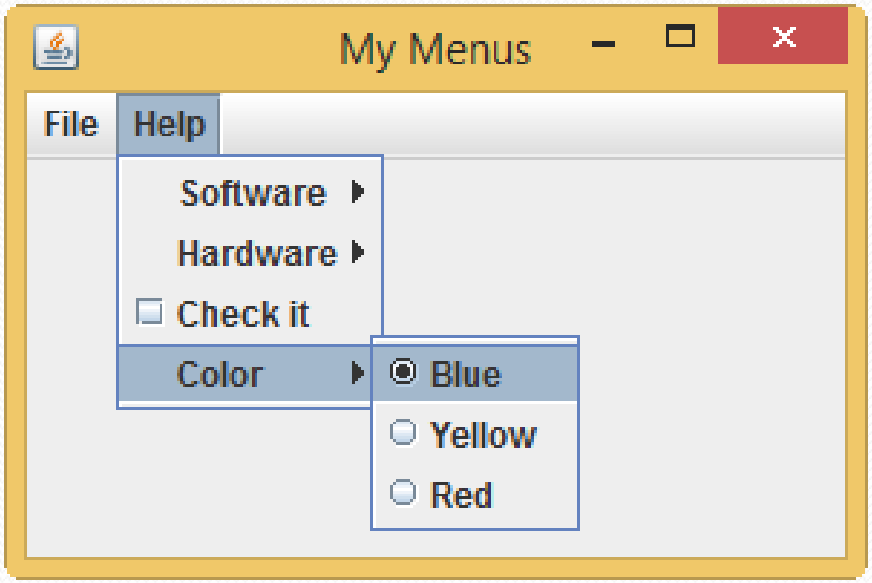


Image Icons, Mnemonics, Accelerators

```
JMenuItem jmiNew, jmiOpen;
fileMenu.add(jmiNew = new JMenuItem("New"));
fileMenu.add(jmiOpen = new JMenuItem("Open"));
jmiNew.setIcon(new ImageIcon("img/button_new.gif"));
jmiOpen.setIcon(new ImageIcon("img/button_open.gif"));
```

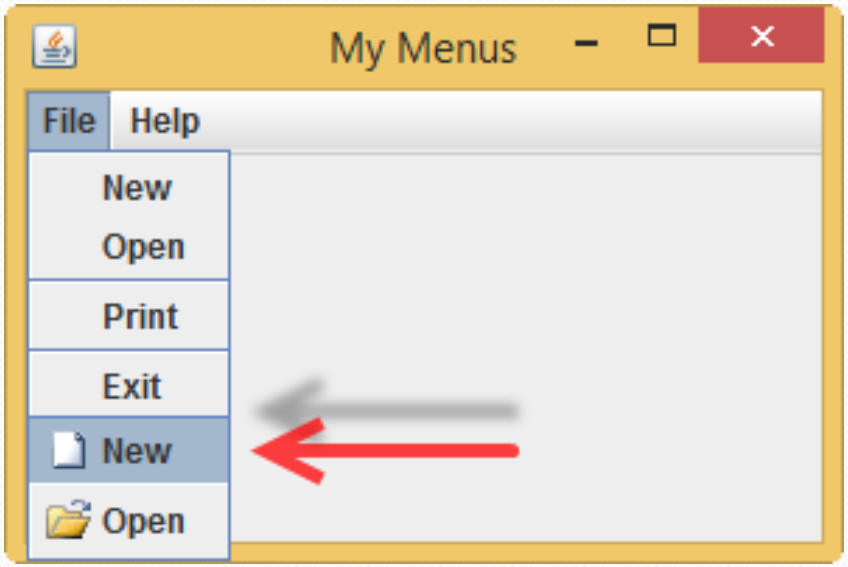


Image Icons, Mnemonics, Accelerators

- To select a menu, press the ALT key and the mnemonic key.
  - Example: press ALT + F to select File menu.

```
helpMenu.setMnemonic('H');
fileMenu.setMnemonic('F');
jmiNew.setMnemonic('N');
jmiOpen.setMnemonic('O');
```

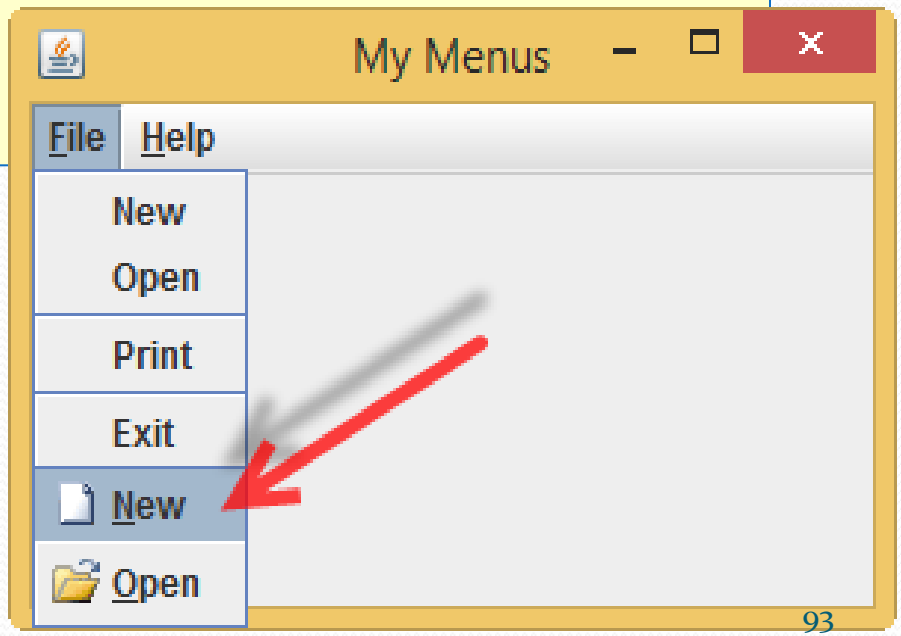


Image Icons, Mnemonics, Accelerators

- Key accelerators: select a menu item directly by pressing CTRL + accelerator keys.
  - Example: you can attach the accelerator CTRL+O to the Open menu item:

```
jmiOpen.setAccelerator(KeyStroke.getKeyStroke(
KeyEvent.VK_O, ActionEvent.CTRL_MASK));
```

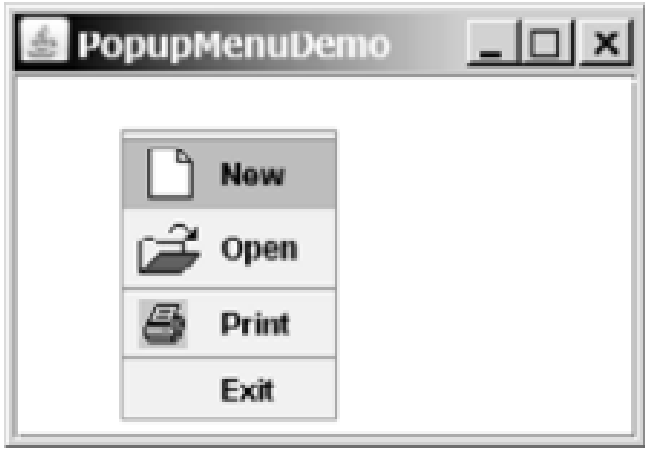
Popup Menus

- A popup menu, also known as a context menu, is like a regular menu, but does not have a menu bar and can float anywhere on the screen.

```
JPopupMenu pm= new JPopupMenu()
jPopupMenu.add(new JMenuItem("New"));
jPopupMenu.add(new JMenuItem("Open"));
```

```
public class PopupMenuDemo extends JFrame {
    JMenuItem mniNew, mniOpen, mniPrint, mniExit;
    JPopupMenu pmn = new JPopupMenu();
    public PopupMenuDemo() {
        pmn.add(mniNew = new JMenuItem("New"));
        pmn.add(mniOpen = new JMenuItem("Open"));
        pmn.add(mniPrint = new JMenuItem("Print"));
        pmn.add(mniExit = new JMenuItem("Exit"));

        // xử lý sự kiện
        this.addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent arg0) {
                showPopup(arg0);
            }
        });

        // for frame
        setVisible(true);
        setSize(200, 200);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }
    private void showPopup(MouseEvent e) {
        pmn.show(e.getComponent(), e.getX(), e.getY());
    }

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

PopupMenu example



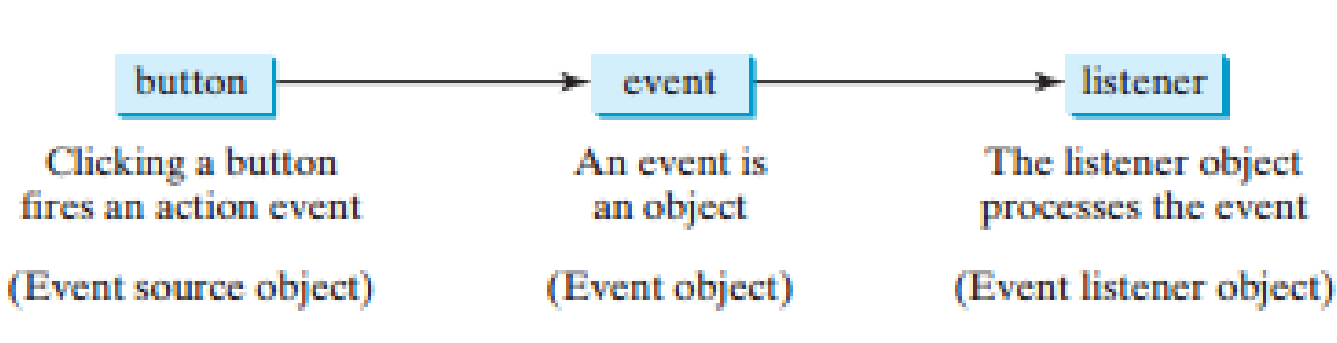
Advanced Programming

Event-Driven Programming

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Introduction

- *You can write code to process events such as a button click or a timer.*
- To respond to a button click, you need to write the code to process the button-clicking action.
- The button is an *event source object*—where the action originates.
- You need to create an object capable of handling the action event on a button. This object is called an *event listener*



Introduction

- To be a listener of an action event, two requirements must be met:
  - The object must be an instance of the **ActionListener** interface. This interface defines the common behavior for all action listeners.
  - The **ActionListener** object **listener** must be registered with the event source object using the method **source.addActionListener(listener)**.

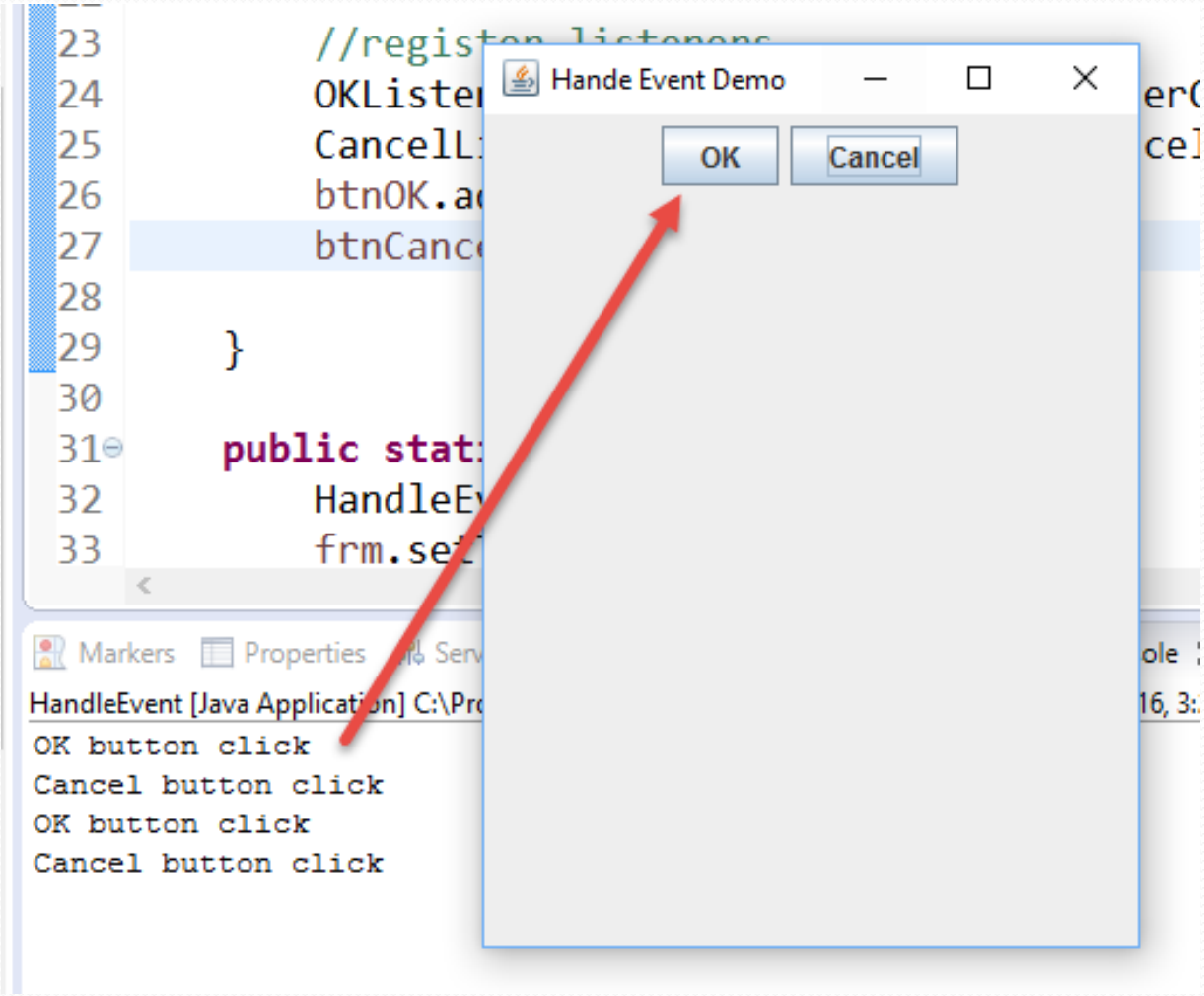
Example

```
public class HandleEvent extends JFrame{
   public HandleEvent() {
         //create 2 buttons
         JButton btnOK = new JButton("OK");
         JButton btnCancel = new JButton("Cancel");

         //create a panel to hold buttons
         JPanel pnlButton = new JPanel();
         pnlButton.add(btnOK);
         pnlButton.add(btnCancel);
         add(pnlButton);
         //register listeners
         OKListenerClass okLis = new OKListenerClass();
         CancelListenerClass canLis = new CancelListenerClass();
         btnOK.addActionListener(okLis);
         btnCancel.addActionListener(canLis);
   }
}
```

Example

```
class OKListenerClass implements ActionListener{
   @Override
   public void actionPerformed(ActionEvent e) {
        System.out.println("OK button click");
   }
}

class CancelListenerClass implements ActionListener{
   @Override
   public void actionPerformed(ActionEvent e) {
        System.out.println("Cancel button click");
   }
}
```

Example


23 //registen listeners
24 OKListe
25 CancelLi
26 btnOK.a
27 btnCanc
28
29 }
30
31 public stat
32 HandleEv
33 frm.set
Markers
Properties
HandleEvent [Java Application] C:\Pr
OK button click
Cancel button click
OK button click
Cancel button click
Hande Event Demo
OK
Cancel

Events and Event Sources

- An **event** is an object created from an **event source**.
  - Firing an **event** means to create an event and delegate the listener to handle the **event**.
- When you run a Java GUI program, the program interacts with the user, and the events drive its execution. This is called *event-driven programming*
- An *event* can be defined as a signal to the program that something has happened

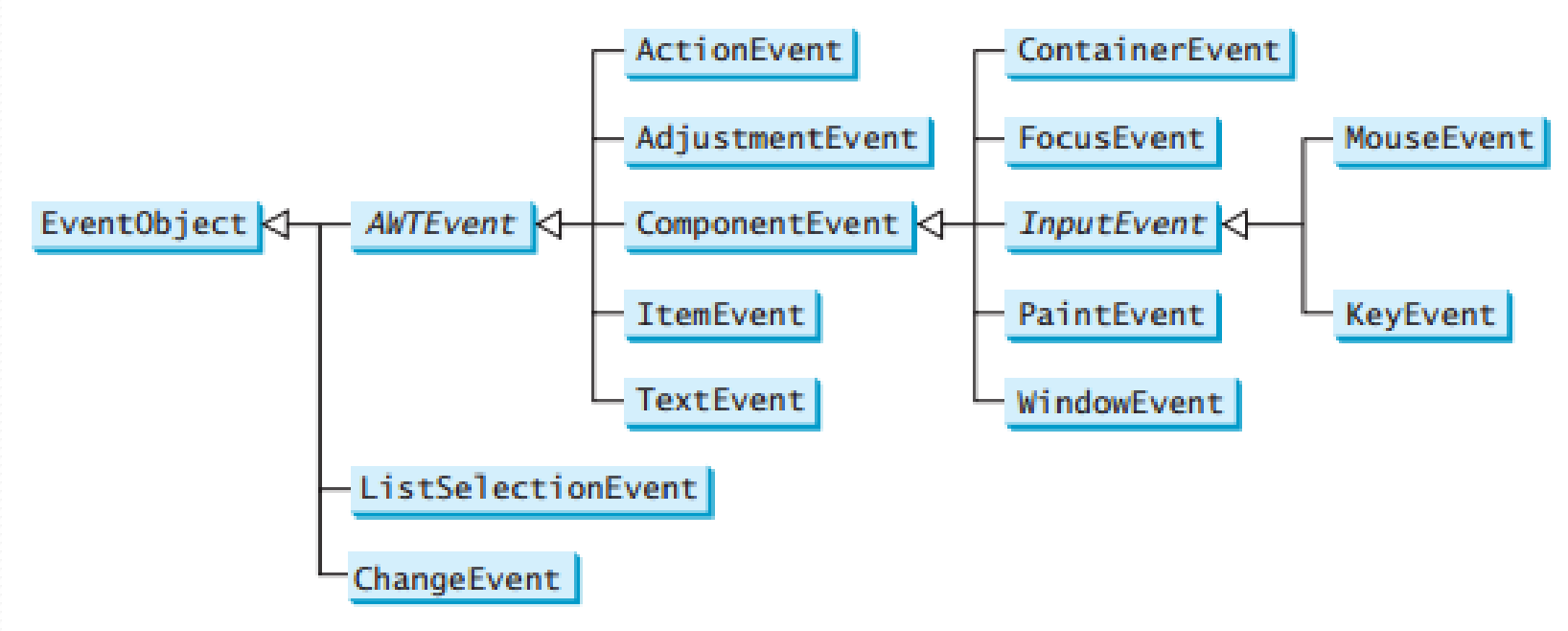
Events and Event Sources

- **Events** are triggered:
  - either by external user actions: mouse movements, button clicks, and keystrokes,
  - or by internal program activities: a timer.
- The program can choose to respond to or ignore an event

Events and Event Sources

- The component that creates an event and fires it is called the *event source object*, or simply *source object* or *source component*.
  - For example, a button is the source object for a buttonclicking action event.
- An event is an instance of an event class.
- The root class of the event classes is **java.util.EventObject**
- You can identify the source object of an event using the **getSource**() instance method in the **EventObject** class.

Events and Event Sources



User Action	*Source Object*	*Event Type Fired*	*Listener Interface*	*Listener Interface Methods*
Click a button	`JButton`	`ActionEvent`	`ActionListener`	`actionPerformed(ActionEvent e)`
Press Enter in a text field	`JTextField`	`ActionEvent`	`ActionListener`	`actionPerformed(ActionEvent e)`
Select a new item	`JComboBox`	`ActionEvent` `ItemEvent`	`ActionListener` `ItemListener`	`actionPerformed(ActionEvent e)` `itemStateChanged(ItemEvent e)`
Check or uncheck	`JRadioButton`	`ActionEvent` `ItemEvent`	`ActionListener` `ItemListener`	`actionPerformed(ActionEvent e)` `itemStateChanged(ItemEvent e)`
Check or uncheck	`JCheckBox`	`ActionEvent` `ItemEvent`	`ActionListener` `ItemListener`	`actionPerformed(ActionEvent e)` `itemStateChanged(ItemEvent e)`
Select a new item	`JComboBox`	`ActionEvent` `ItemEvent`	`ActionListener` `ItemListener`	`actionPerformed(ActionEvent e)` `itemStateChanged(ItemEvent e)`
Mouse pressed	`Component`	`MouseEvent`	`MouseListener`	`mousePressed(MouseEvent e)`
Mouse released				`mouseReleased(MouseEvent e)`
Mouse clicked				`mouseClicked(MouseEvent e)`
Mouse entered				`mouseEntered(MouseEvent e)`
Mouse exited				`mouseExited(MouseEvent e)`
Mouse moved			`MouseMotionListener`	`mouseMoved(MouseEvent e)`
Mouse dragged				`mouseDragged(MouseEvent e)`
Key pressed	`Component`	`KeyEvent`	`KeyListener`	`keyPressed(KeyEvent e)`
Key released				`keyReleased(KeyEvent e)`
Key typed				`keyTyped(KeyEvent e)`

Inner Class

- *An* inner class, *or* nested class, *is a class defined within the scope of another class. Inner classes are useful for defining listener classes.*

```
public class Test {
  ...
}

public class A {
  ...
}
```

(a)

```
public class Test {
  ...

  // Inner class
  public class A {
    ...
  }
}
```

```
// OuterClass.java: inner class demo
public class OuterClass {
  private int data;

  /** A method in the outer class */
  public void m() {
    // Do something
  }

  // An inner class
  class InnerClass {
    /** A method in the inner class */
    public void mi() {
      // Directly reference data and method
      // defined in its outer class
      data++;
      m();
    }
  }
}
```

Inner classes

- An inner class may be used just like a regular class.
- You define an inner class if it is used only by its outer class.
- An inner class has the following features:
  - An inner class is complied into a class named OuterClassName@InnerClassName.class
    - For example: the inner class A in Test is compiled into Test$A.class.
  - An inner class can reference the data and methods defined in the outer class in which it nests➔ you need not pass the reference of an object of the outer class to the constructor of inner class➔ make program simple and concise.

Inner class example

```
public class MyInnerClassListener extends JFrame {
    JButton btnSave, btnCancel;
    JTextField txtNum1, txtNum2, txtResult;

    public MyInnerClassListener() {
        super("My Inner Class Listener Demo");
        setLayout(new FlowLayout());
        // number 1:
        add(new JLabel("Number 1: "));
        add(txtNum1 = new JTextField(10));
        // number 2:
        add(new JLabel("Number 2: "));
        add(txtNum2 = new JTextField(10));
        // result:
        add(new JLabel("Result: "));
        add(txtResult = new JTextField(10));
        txtResult.setEditable(false);
```

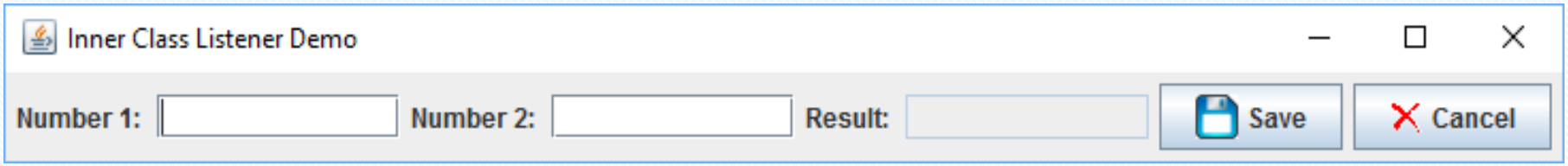
Inner class example

```
        // buttons
        add(btnSave = new JButton("Save"));
        add(btnCancel = new JButton("Cancel"));
        btnSave.setIcon(new ImageIcon("images/save.gif"));
        btnCancel.setIcon(new ImageIcon("images/button_delete.gif"));

        // add listener
        MyListener lst = new MyListener();
        btnSave.addActionListener(lst);
        btnCancel.addActionListener(lst);
    }
```

Inner class example

```
    class MyListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == btnSave) {
                double n1 = Double.parseDouble(txtNum1.getText());
                double n2 = Double.parseDouble(txtNum2.getText());
                txtResult.setText((n1 + n2) + "");
            } else
                System.exit(0);
        }
    }
}
```

Inner class example



Anonymous Class Listeners

- A listener class is designed specifically to create a listener object for a GUI component (e.g., a button). The listener class will not be shared by other applications ➔ define an inner class.
- Inner class listeners can be shortened using anonymous inner class.
- An **anonymous inner class** is an inner class without a name. It combines defining an inner class and creating an instance of the class in one step.

Anonymous Class Listeners

```
public ControlCircle2() {
  // Omitted

  jbtEnlarge.addActionListener(
    new EnlargeListener() );
}

class EnlargeListener
    implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    canvas.enlarge();
  }
}
```

(a) Inner class EnlargeListener

```
public ControlCircle2() {
  // Omitted

  jbtEnlarge.addActionListener(
    new class EnlargeListener
        implements ActionListener() {
      public void
          actionPerformed(ActionEvent e) {
        canvas.enlarge();
      }
    });
}
```

(b) Anonymous inner class

Anonymous class example

```
public class AnonymousListenerDemo extends JFrame {
    public AnonymousListenerDemo() {
        // create 2 buttons
        JButton btnNew = new JButton("New");
        JButton btnOpen = new JButton("Open");
        // create a panel to hold buttons
        JPanel pButtons = new JPanel();
        pButtons.add(btnNew); pButtons.add(btnOpen);

        add(pButtons);
```

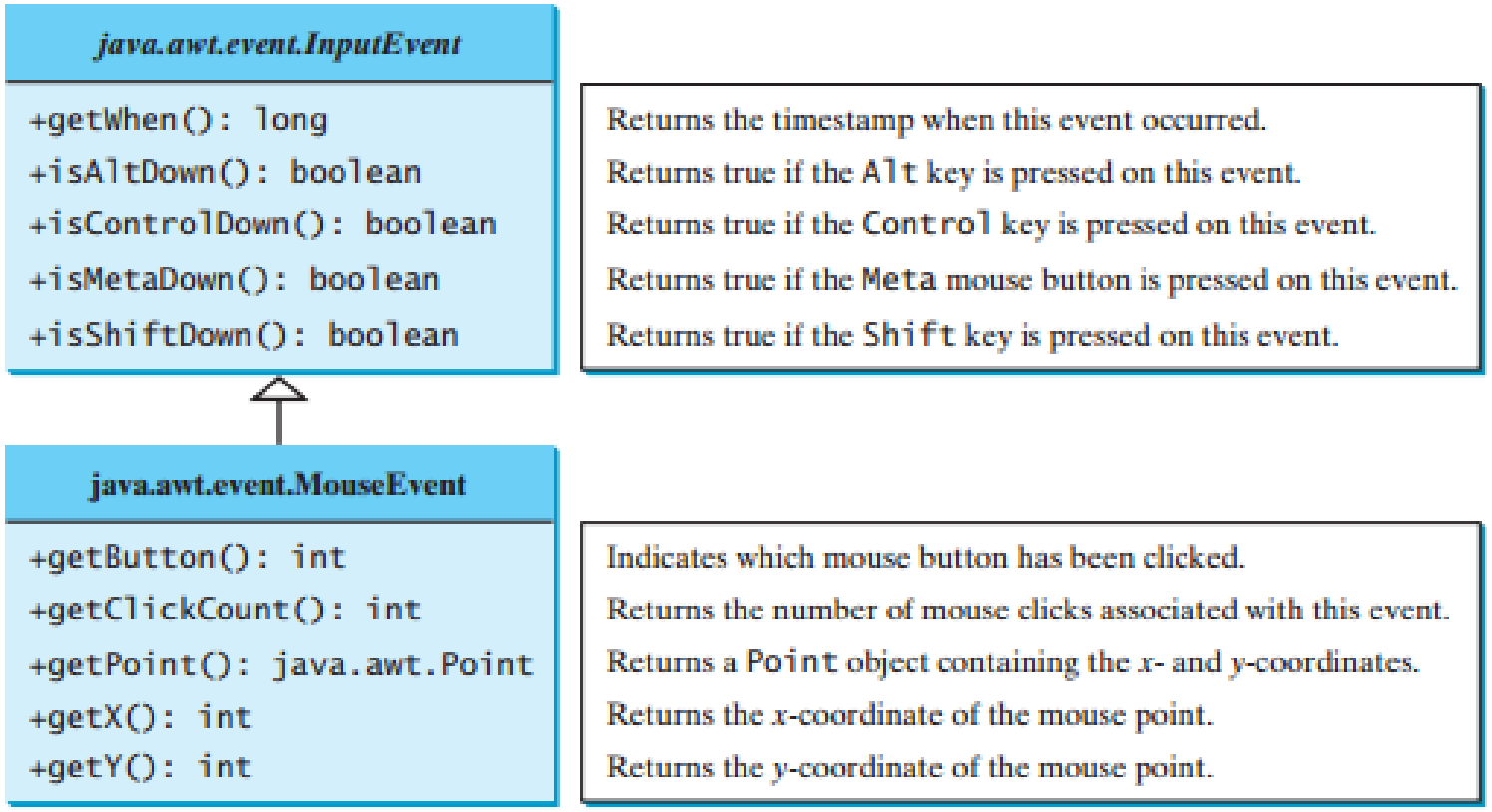
Anonymous class example

```
        // create and register anonymous inner-class listener
        btnNew.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("New button is just clicked");
            }
        });
        btnOpen.addActionListener(new ActionListener() {
        @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Open button is just clicked");
            }
        });
    }
}
```

Mouse Event

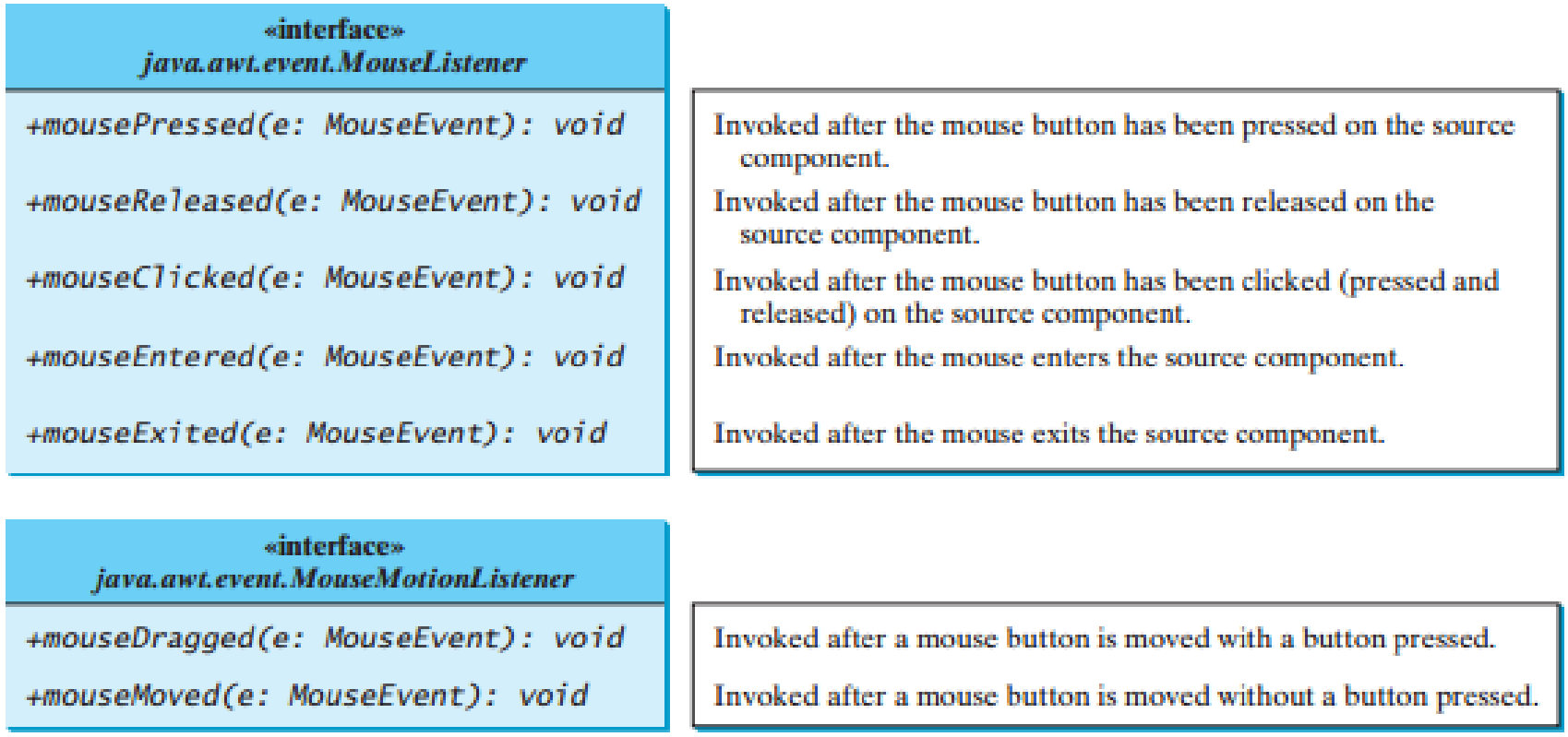
- A **mouse event** is fired whenever a mouse button is pressed, released, or clicked, the mouse is moved, or the mouse is dragged onto a component.
- The **MouseEvent** object captures the event:
  - the number of clicks associated with it
  - the location (the *x*- and *y*-coordinates) of the mouse
  - or which button was pressed

Mouse Event


java.awt.event.InputEvent
+getWhen(): long
+isAltDown(): boolean
+isControlDown(): boolean
+isMetaDown(): boolean
+isShiftDown(): boolean
Returns the timestamp when this event occurred.
Returns true if the Alt key is pressed on this event.
Returns true if the Control key is pressed on this event.
Returns true if the Meta mouse button is pressed on this event.
Returns true if the Shift key is pressed on this event.
java.awt.event.MouseEvent
+getButton(): int
+getClickCount(): int
+getPoint(): java.awt.Point
+getX(): int
+getY(): int
Indicates which mouse button has been clicked.
Returns the number of mouse clicks associated with this event.
Returns a Point object containing the x- and y-coordinates.
Returns the x-coordinate of the mouse point.
Returns the y-coordinate of the mouse point.

Mouse Event

- Java provides two listener interfaces, **MouseListener** and **MouseMotionListener**, to handle mouse events



MouseEvent example

```
public class MouseEventDemo extends JFrame implements MouseListener,
        MouseMotionListener {
        JTextField tf;
   public MouseEventDemo(String title) {
        super(title);
        tf = new JTextField(60);
        addMouseListener(this);
        setLayout(new BorderLayout()); add(tf, BorderLayout.SOUTH);
        setSize(300, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
  }
   public static void main(String[] args) {
        MouseEventDemo me = new MouseEventDemo("Mouse Event Demo");
}
```

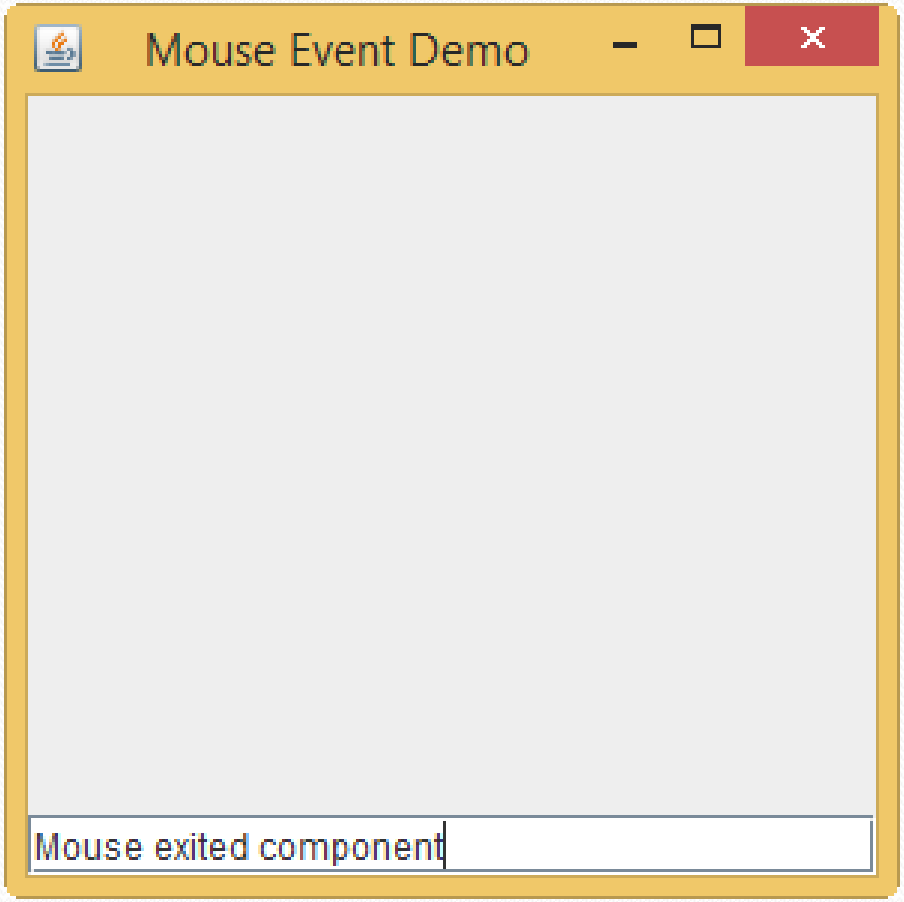
MouseEvent Example

```
public void mouseClicked(MouseEvent e) {
      String msg = "Mouse clicked";
      tf.setText(msg);
}
public void mousePressed(MouseEvent e) {
      String msg = "Mouse pressed";
      tf.setText(msg);
}
public void mouseReleased(MouseEvent e) {
      String msg = "Mouse released";
      tf.setText(msg);
}
public void mouseEntered(MouseEvent e) {
      String msg = "Mouse entered component";
      tf.setText(msg);
}
```

MouseEvent example

```
    public void mouseExited(MouseEvent e) {
          String msg = "Mouse exited component";
          tf.setText(msg);
    }
    public void mouseDragged(MouseEvent e) {
          String msg = "Mouse dragged at" + e.getX() + ", " + e.getY();
          tf.setText(msg);
    }
    public void mouseMoved(MouseEvent e) {
          String msg = "Mouse moved at" + e.getX() + ", " + e.getY();
          tf.setText(msg);
    }
}
```

MouseEvent Example



Listener Interface Adapters

- *A* listener interface adapter *is a class that provides the default implementation for all the methods in the listener interface.*

```
addMouseMotionListener(
    new MouseMotionListener() {
  @Override /** Handle mouse-dragged event */
  public void mouseDragged(MouseEvent e){
    x = e.getX();
    y = e.getY();
    repaint();
  }

  @Override /** Handle mouse-moved event */
  public void mouseMoved(MouseEvent e) {
  }
});
```

(a) Using a listener interface

```
addMouseMotionListener(
    new MouseMotionAdapter() {
  @Override /** Handle mouse-dragged event */
  public void mouseDragged(MouseEvent e){
    x = e.getX();
    y = e.getY();
    repaint();
  }
});
```

(b) Using a listener interface adapter

Listener Interface Adapters

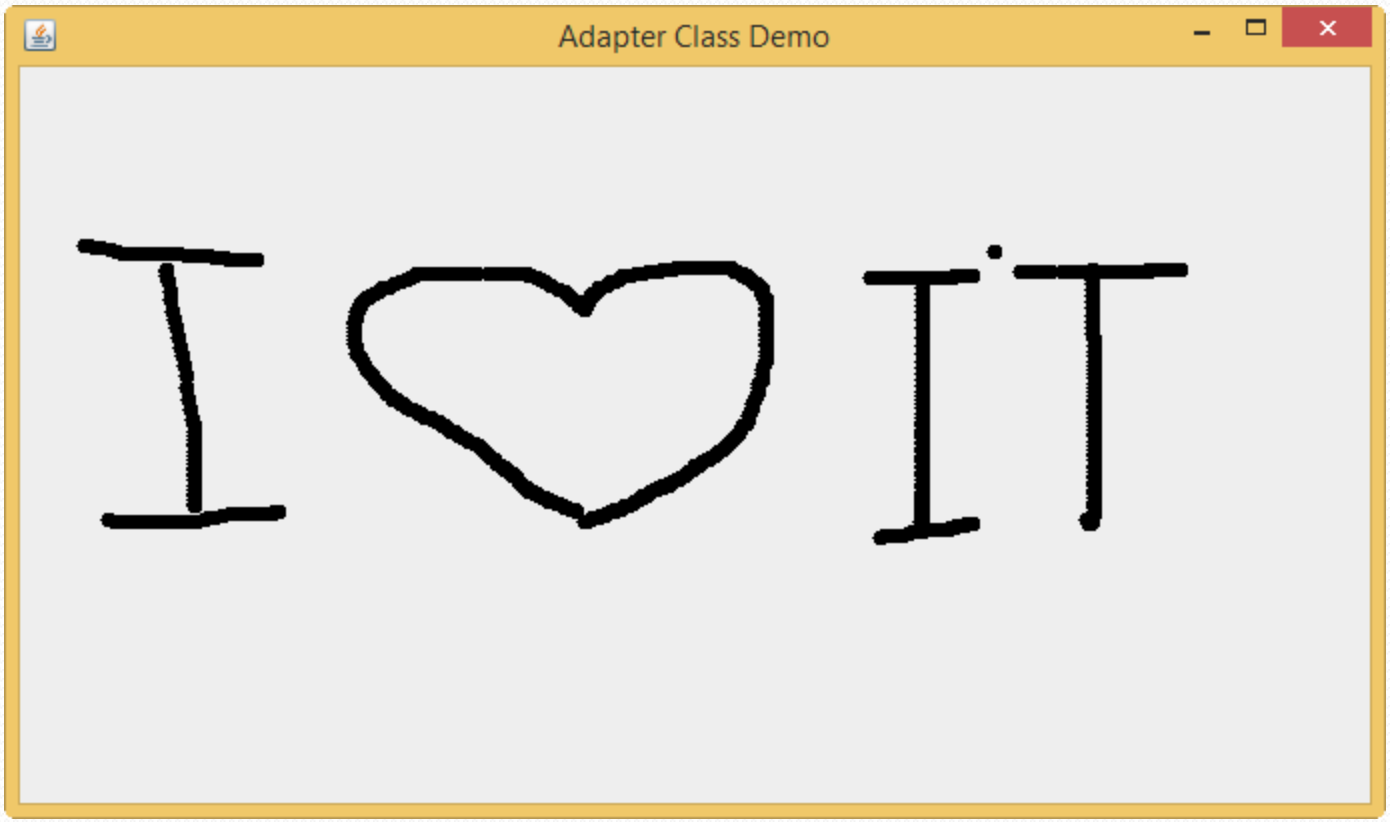
- *A* listener interface adapter *is a class that provides the default implementation for all the methods in the listener interface.*

Listener Interface Adapters

Adapter	*Interface*
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`KeyAdapter`	`KeyListener`
`WindowAdapter`	`WindowListener`

```
public class AdapterClassDemo extends JFrame {
   private Point point = new Point();
   public AdapterClassDemo(String title) {
         super(title);
         // register listener
         addMouseMotionListener(new MouseMotionAdapter() {
                  public void mouseDragged(MouseEvent event) {
                           point = event.getPoint();
                           repaint();
                  }
         });
         setSize(300, 300);
         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         setVisible(true);
   }
   public void paint(Graphics g) {
         g.fillOval(point.x, point.y, 8, 8);
   }
   public static void main(String[] args) {
         AdapterClassDemo frm = new AdapterClassDemo("Adapter Class Demo");
   }
}
```

Ví dụ Adapter



Creating Multiple Windows

- *Multiple windows can be created in one program.*
- You may want to create multiple windows in an application so that the application can open a new window to perform a specified task.
  - The new windows are called *subwindows*
  - The main frame is called the *main window*

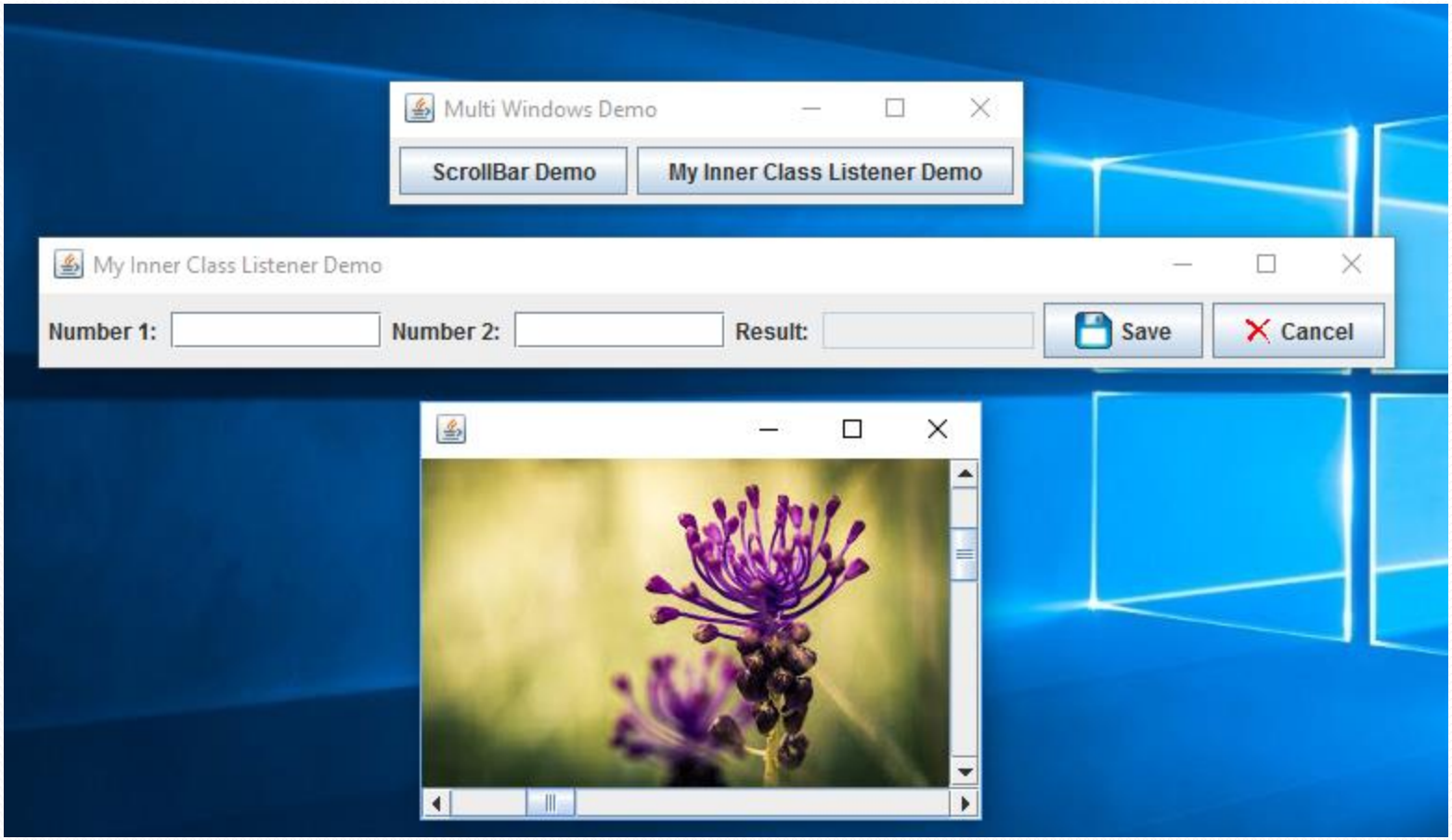
Creating Multiple Windows

```
public class MultiWindowsDemo extends JFrame {
    private JFrame newFrame1 = new MyInnerClassListener();
    private JFrame newFrame2 = new ScrollBarDemo();
    public MultiWindowsDemo() {
        setLayout(new FlowLayout());
        // create 2 buttons
        JButton btnScroll = new JButton("ScrollBar Demo");
        JButton btnInner = new JButton("My Inner Class Listener Demo");

        add(btnScroll);
        add(btnInner);
```

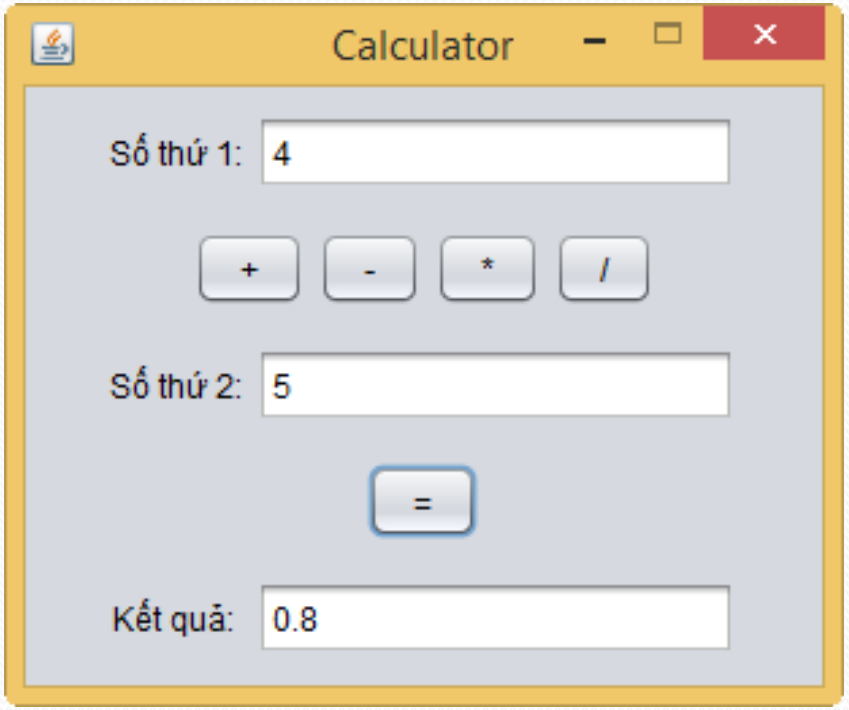
Creating Multiple Windows

```
        // create and register anonymous inner-class listener
        btnScroll.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                newFrame1.setVisible(true);
                newFrame1.pack();
                newFrame1.setLocation(420, 420);
            }
        });

        btnInner.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                newFrame2.setVisible(true);
                newFrame2.pack();
                newFrame2.setLocation(450, 450);
            }
        });
    }
}
```

Creating Multiple Windows

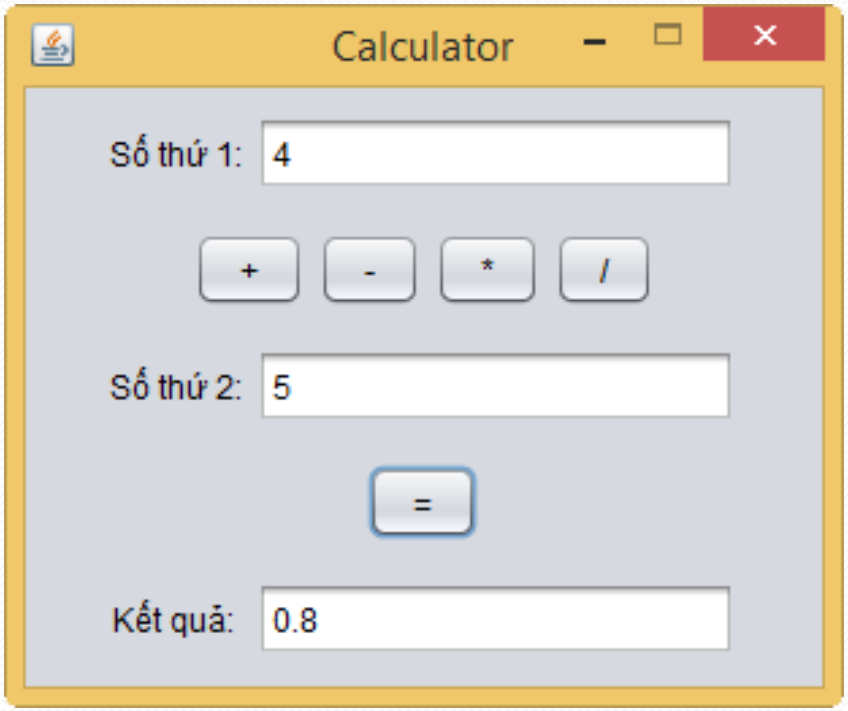


Exercise 1

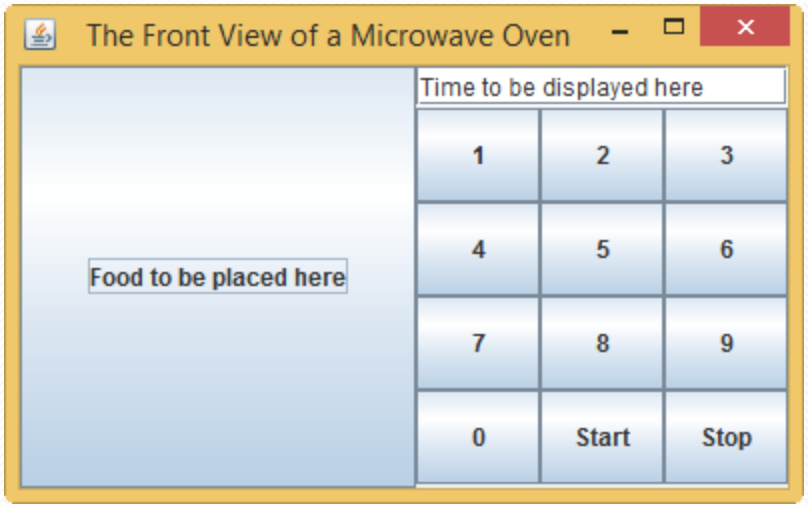


Exercise 1 - Solution

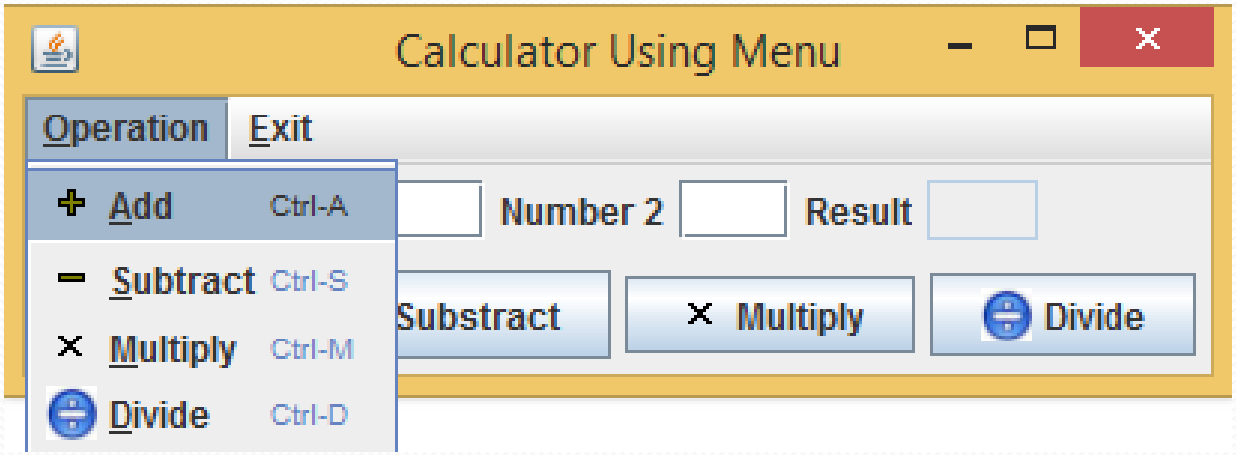
- Với hàng thứ nhất:
  - Tạo 1 JPanel jplP1, set Layout: jplP1.setLayout(**new FlowLayout()**);
  - JLabel lblSo1
  - JTextField jtfSo1
  - Add lblSo1 và jtfSo1 vào jplP1
  - Add jplP1 vào Frame



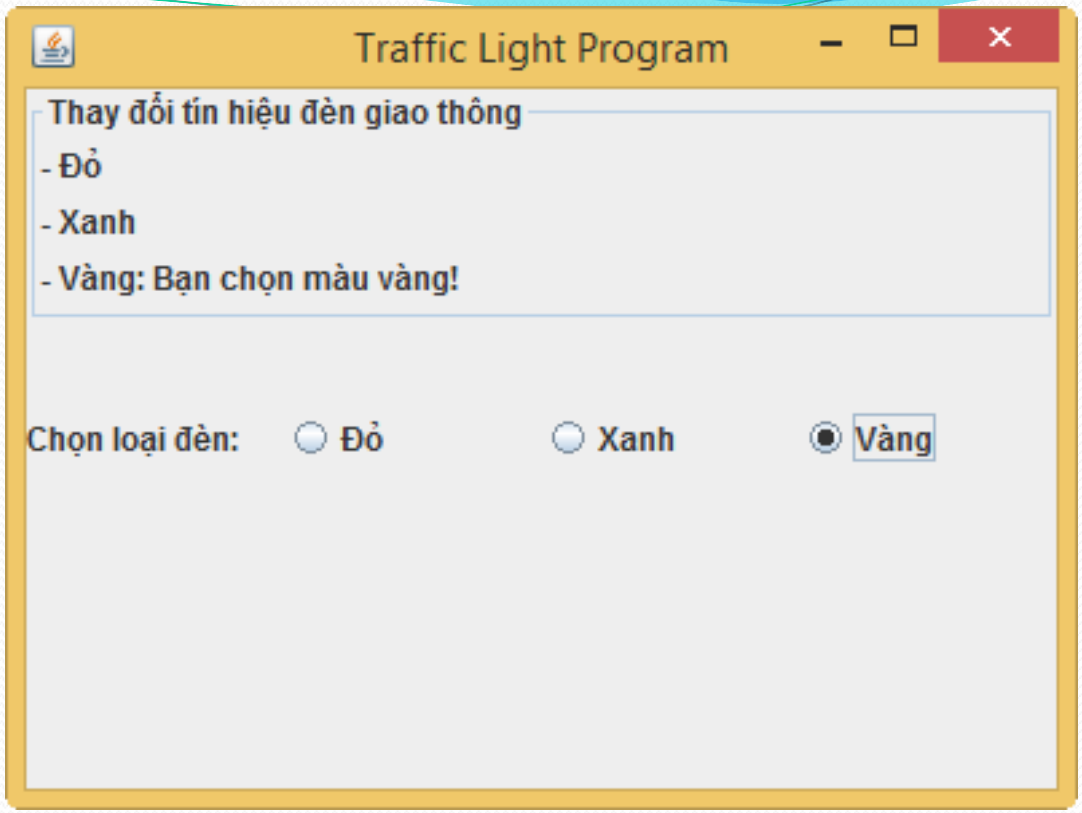
Exercise 2


The Front View of a Microwave Oven
Time to be displayed here
1
2
3
4
5
6
Food to be placed here
7
8
9
0
Start
Stop

Exercise 3

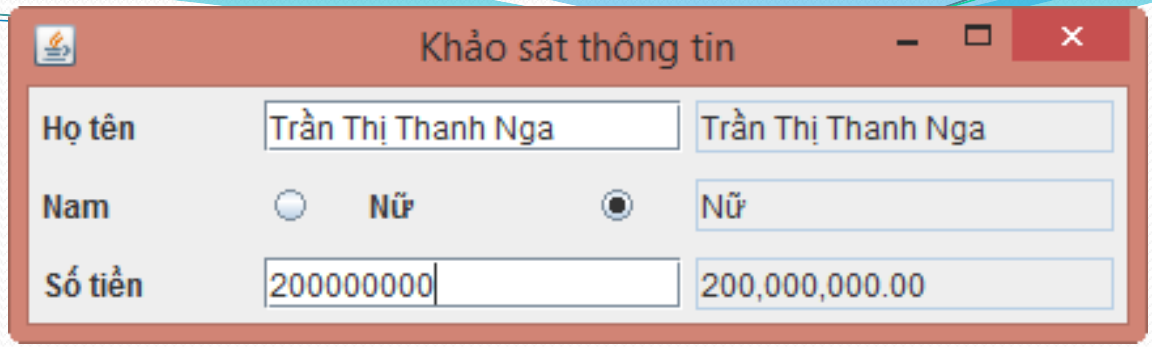

Calculator Using Menu
Operation
Exit
Add Ctrl-A
Subtract Ctrl-S
Multiply Ctrl-M
Divide Ctrl-D
Number 2
Result
Substract
Multiply
Divide

Exercise 4




```
JPanel p1 = new JPanel();
p1.setBorder(new TitledBorder("Thay đổi tín hiệu đèn
                    giao thông"));
add(p1);
```

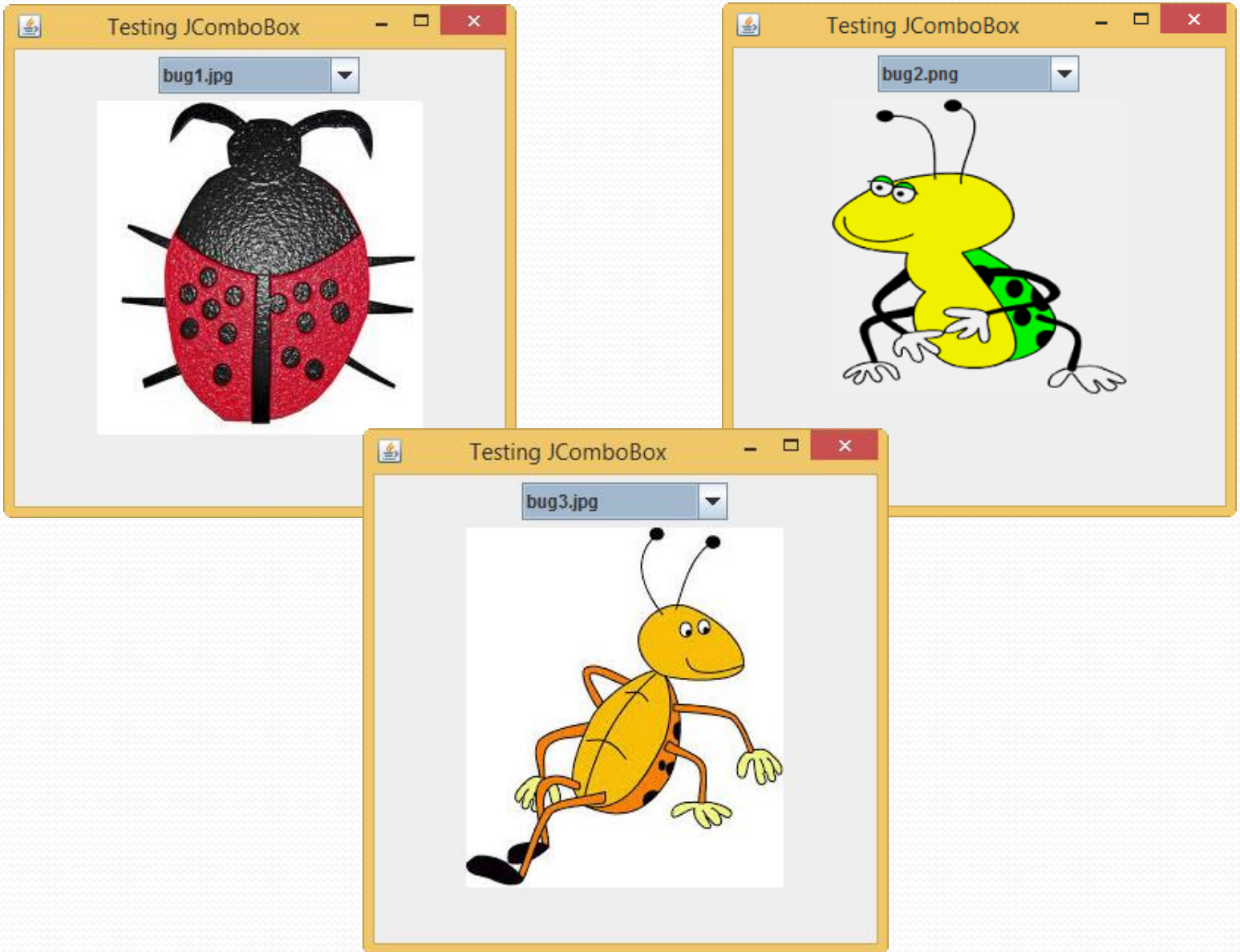
Exercise 5



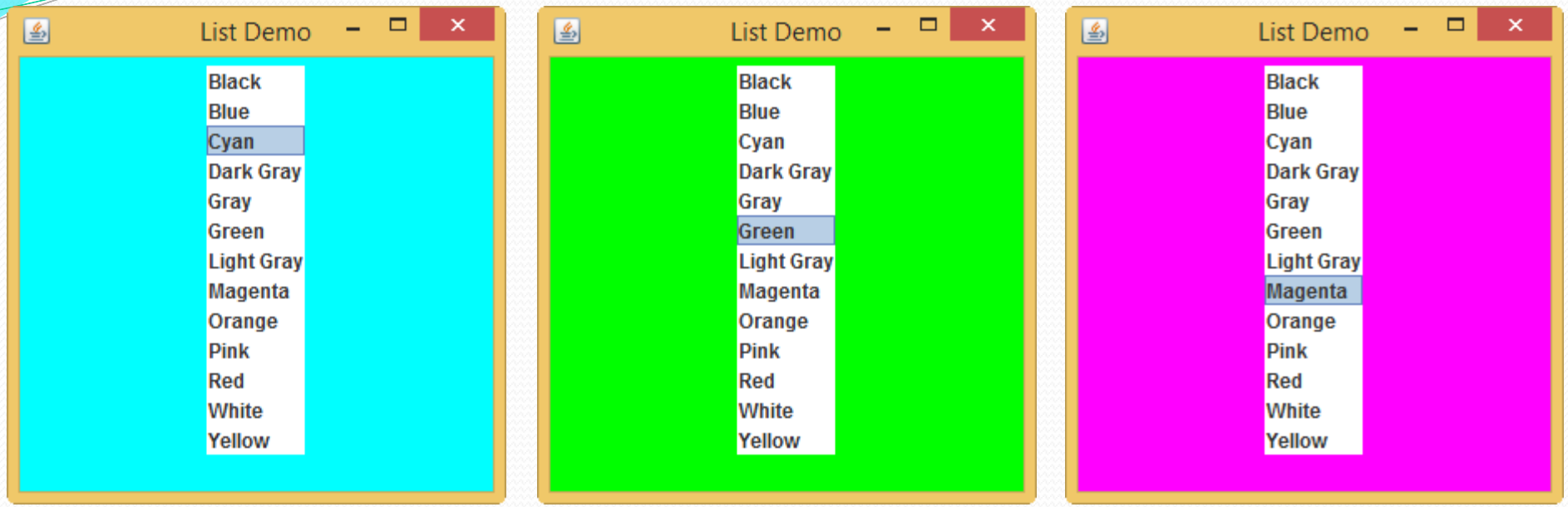

```
txtMoneyShow = new JTextField();
txtMoneyShow.setPreferredSize(dimText);
txtMoneyShow.setEditable(false);
row3.add(txtMoneyShow);
```

```
txtMoney.addMouseListener(new MouseAdapter() {
    public void mouseExited(MouseEvent event) {
        String result = String.format("%,.2f",
        Double.parseDouble(txtMoney.getText()));
        txtMoneyShow.setText(result);
    }
});
```

Exercise 6


Testing JComboBox
bug1.jpg
Testing JComboBox
bug2.png
Testing JComboBox
bug3.jpg

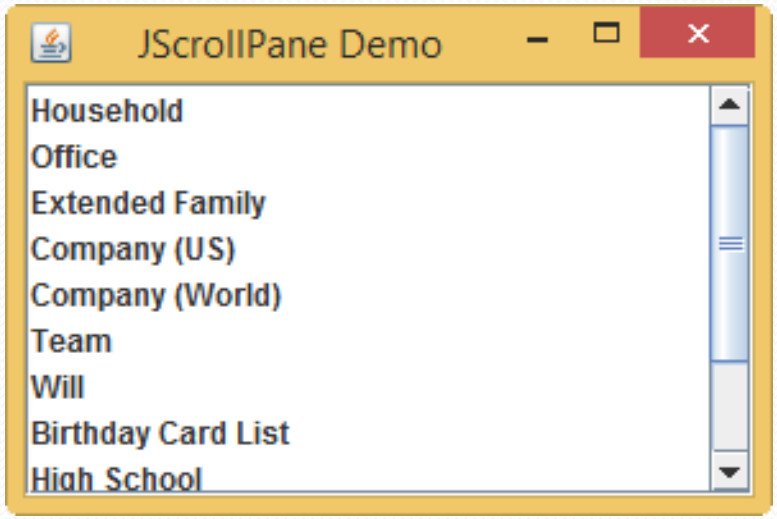
Exercise 7



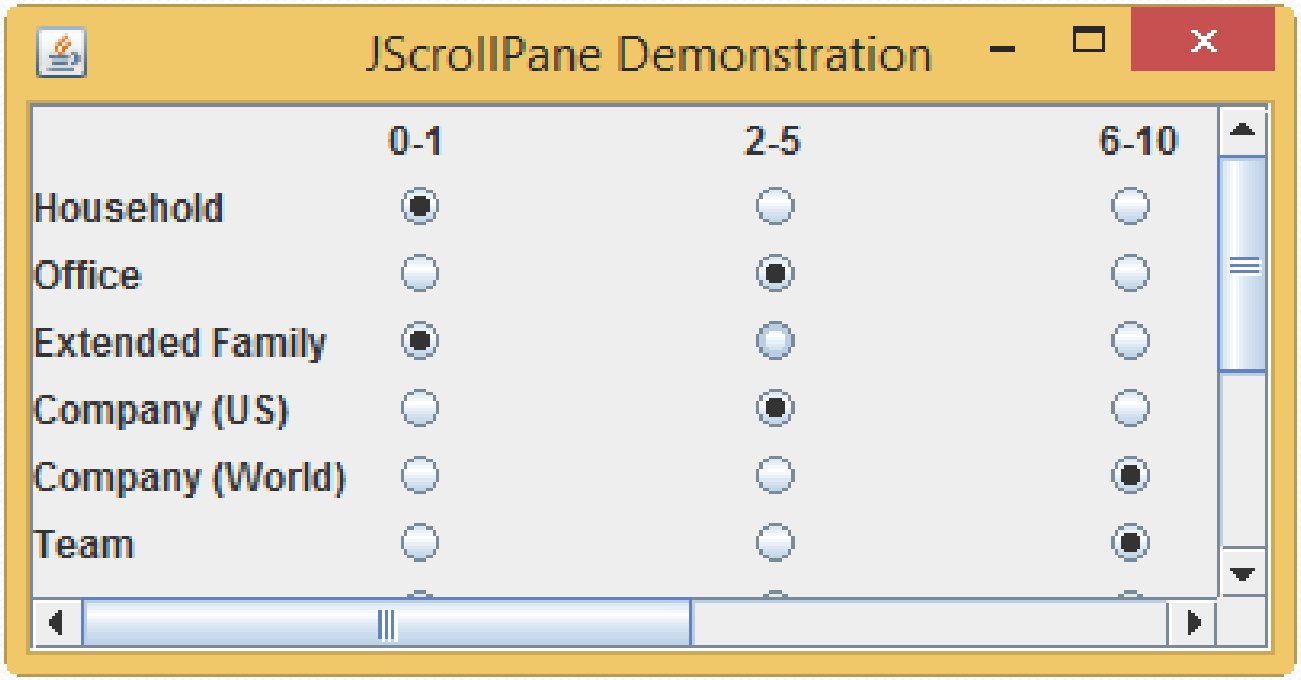

```
lstColor.addListSelectionListener(new ListSelectionListener()
{
    public void valueChanged(ListSelectionEvent e) {
        int index = lstColor.getSelectedIndex();
        contentPane.setBackground(color[index]);
}});
```

Exercise 8: Dùng JScrollPane

```
String categories[] = { "Household", "Office", "Extended
Family", "Company (US)", "Company (World)", "Team", "Will",
"Birthday Card List", "High School", "Country",
"Continent", "Planet" };
JList list = new JList(categories);
scrollpane = new JScrollPane(list);
add(scrollpane, BorderLayout.CENTER);
```

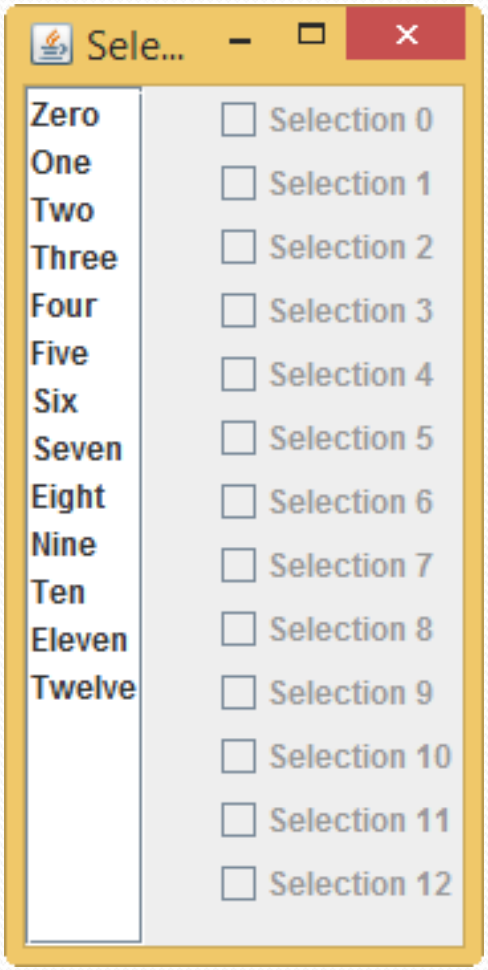


Exercise 9: Dùng JScrollPane

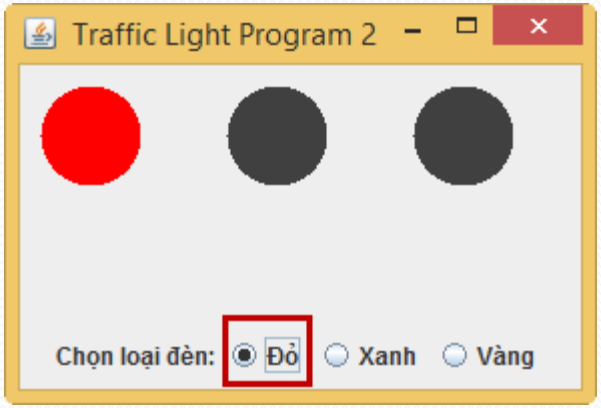


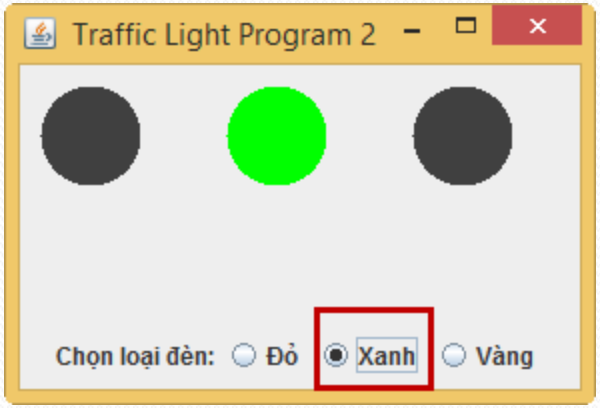
Exercise 10: Dùng ScrollBar

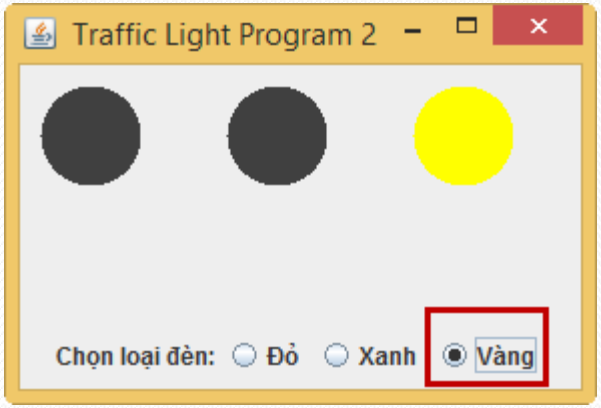
Exercise 11


Sele...
Zero
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
Eleven
Twelve
Selection 0
Selection 1
Selection 2
Selection 3
Selection 4
Selection 5
Selection 6
Selection 7
Selection 8
Selection 9
Selection 10
Selection 11
Selection 12

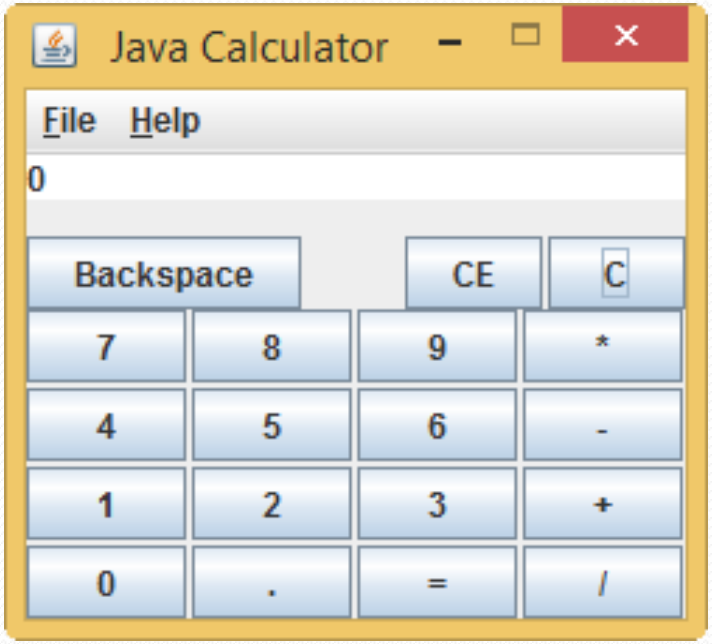
Exercise 12


Traffic Light Program 2
Chọn loại đèn:
Đỏ
Xanh
Vàng

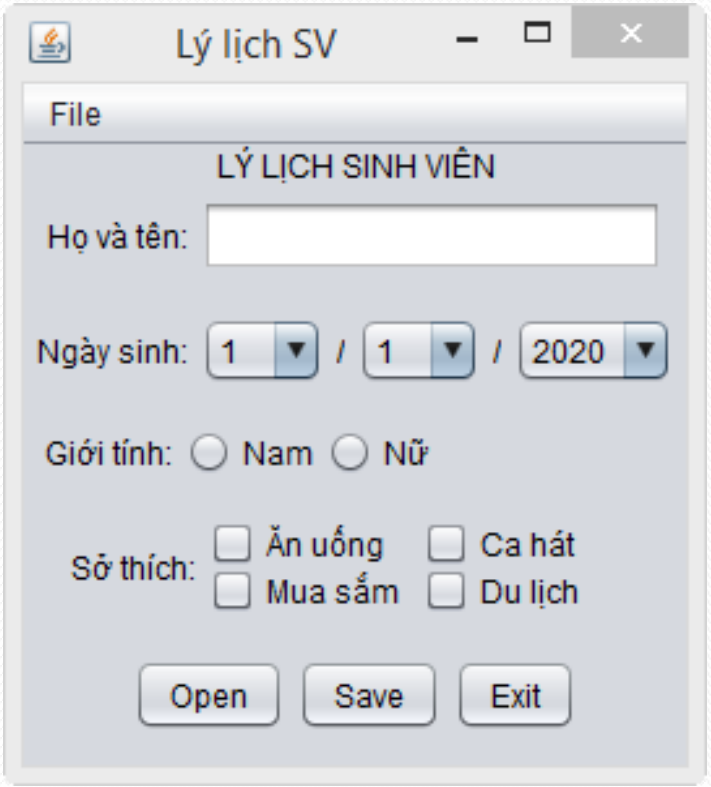

Traffic Light Program 2
Chọn loại đèn:
Đỏ
Xanh
Vàng


Traffic Light Program 2
Chọn loại đèn:
Đỏ
Xanh
Vàng

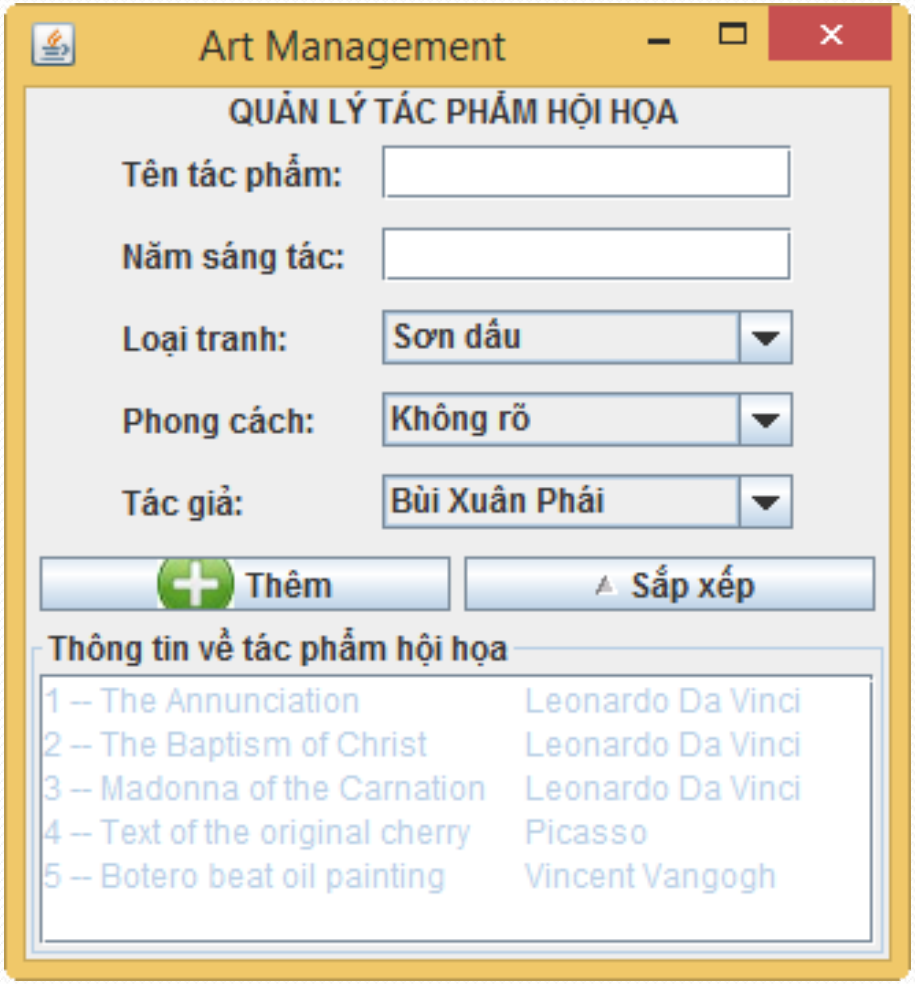
Exercise 13



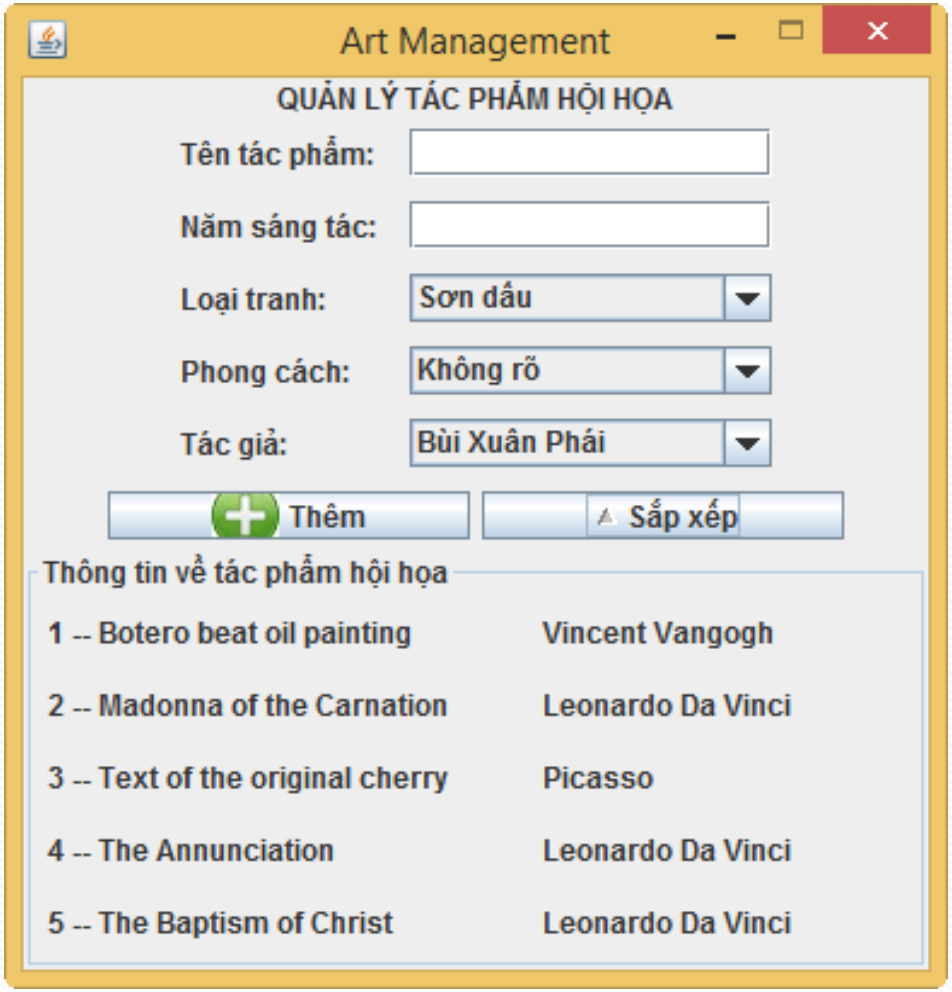
Exercise 14


Lý lịch SV
File
LÝ LỊCH SINH VIÊN
Họ và tên:
Ngày sinh:
1
/
1
/
2020
Giới tính:
Nam
Nữ
Sở thích:
Ăn uống
Ca hát
Mua sắm
Du lịch
Open
Save
Exit

Exercise 15


Art Management
QUẢN LÝ TÁC PHẨM HỘI HỌA
Tên tác phẩm:
Năm sáng tác:
Loại tranh:
Sơn dầu
Phong cách:
Không rõ
Tác giả:
Bùi Xuân Phái
Thêm
Sắp xếp
Thông tin về tác phẩm hội họa
1 -- The Annunciation Leonardo Da Vinci
2 -- The Baptism of Christ Leonardo Da Vinci
3 -- Madonna of the Carnation Leonardo Da Vinci
4 -- Text of the original cherry Picasso
5 -- Botero beat oil painting Vincent Vangogh

Exercise 16


Art Management
QUẢN LÝ TÁC PHẨM HỘI HỌA
Tên tác phẩm:
Năm sáng tác:
Loại tranh:
Sơn dầu
Phong cách:
Không rõ
Tác giả:
Bùi Xuân Phái
Thêm
Sắp xếp
Thông tin về tác phẩm hội họa
1 -- Botero beat oil painting
Vincent Vangogh
2 -- Madonna of the Carnation
Leonardo Da Vinci
3 -- Text of the original cherry
Picasso
4 -- The Annunciation
Leonardo Da Vinci
5 -- The Baptism of Christ
Leonardo Da Vinci

```
setLayout(new BoxLayout(this.getContentPane(), BoxLayout.Y_AXIS));
Dimension dimForLabel = new Dimension(90, 20);
Dimension dimForText = new Dimension(150, 20);
// Tạo tiêu đề
JLabel jlbTitle = new JLabel("QUẢN LÝ TÁC PHẨM HỘI HỌA");
jlbTitle.setAlignmentX(CENTER_ALIGNMENT);
this.add(jlbTitle);
// row1
JPanel jpnRow1 = new JPanel();

JLabel jlbPaintTit = new JLabel("Tên tác phẩm:");
jlbPaintTit.setPreferredSize(dimForLabel);
jpnRow1.add(jlbPaintTit);

txtPaintTit = new JTextField();
txtPaintTit.setPreferredSize(dimForText);
jpnRow1.add(txtPaintTit);
this.add(jpnRow1);
// row2…
```

Dùng ArrayList để tạo danh sách tác phẩm

```
ArrayList<Painting> lstPainting = new ArrayList<Painting>();
private void createPaintingList() {
   lstPainting.add(new Painting("P01", "Leonardo Da Vinci",
       1992, "The Annunciation", "Tác phẩm nổi tiếng",
       "Sơn dầu", "Hiện đại"));
   lstPainting.add(new Painting("P02", "Leonardo Da Vinci",
       1992, "The Baptism of Christ", "Tác phẩm nổi tiếng",
       "Sơn dầu", "Hiện đại"));
       //...
}
```

Hiển thị danh sách tác phẩm

Cách 1: Dùng JTextArea

```
JTextArea txtViewPainting;
 // panel for show data
pnlViewData = new JPanel();
pnlViewData.setLayout(new BorderLayout(0, 0));
txtViewPainting = new JTextArea(6, 0);
txtViewPainting.setEnabled(false);
pnlViewData.add(new JScrollPane(txtViewPainting));
add(pnlViewData);
```

Hiển thị danh sách tác phẩm

```
pnlViewData.setBorder(BorderFactory
.createTitledBorder("Thông tin về sinh viên"));

StringBuffer st = new StringBuffer();
for (int i = 0; i < stdLists.size(); i++) {
        Student p = stdLists.get(i);
        st.append((i + 1) + "- " + p.getStdCode() + "\t\t"
                + p.getStdName() + "\n  ");
}
txtViewStd.setText(st.toString());
```

Sắp xếp danh sách sinh viên

```
Comparator<Student> comp = new Comparator<Student>() {
@Override
public int compare(Student o1, Student o2) {
        return
        (o1.getStdName()).compareToIgnoreCase
                              (o2.getStdName());
        }
};
Collections.sort(this.stdLists, comp);
```

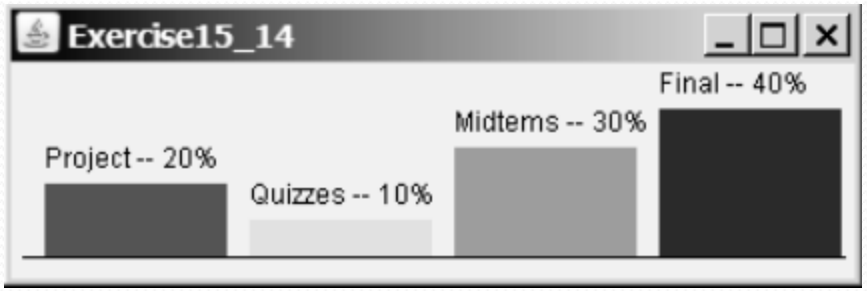
Advanced Programming

Graphic

ThS. Trần Thị Thanh Nga
Khoa CNTT, Trường ĐH Nông Lâm TPHCM
Email: ngattt@hcmuaf.edu.vn

Introduction

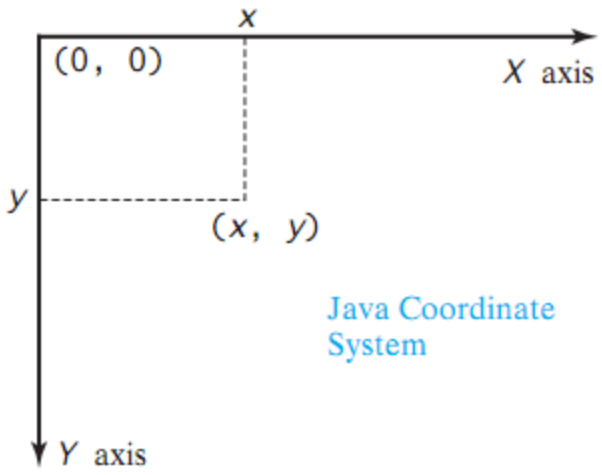
- Suppose you wish to draw shapes such as a bar chart, a clock, or a stop sign. How do you do so?



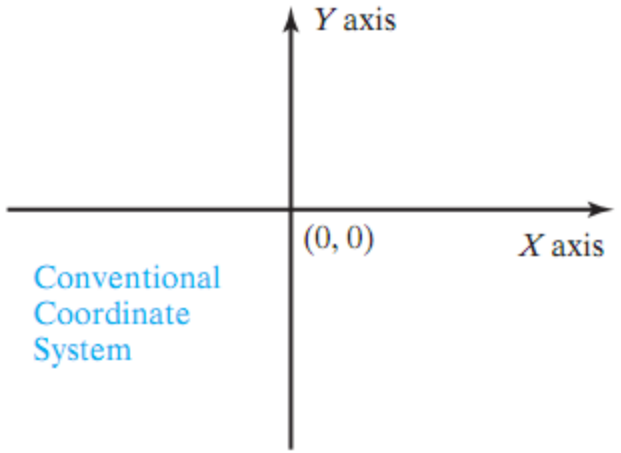
Graphical Coordinate Systems

- To paint, you need to specify where to paint.
- Each component has its own coordinate system with the origin (0, 0) at the upper-left corner.
  - The x-coordinate increases to the right
  - The y-coordinate increases downward.
- Note that the Java coordinate system differs from the conventional coordinate system

Graphical Coordinate Systems



Java Coordinate System



Conventional Coordinate System

The Java coordinate system is measured in pixels, with (0, 0) at its upper-left corner.

java.awt.Graphics	
`+setColor(color: Color): void`	Sets a new color for subsequent drawings.
`+setFont(font: Font): void`	Sets a new font for subsequent drawings.
`+drawString(s: String, x: int, y: int): void`	Draws a string starting at point `(x, y)`.
`+drawLine(x1: int, y1: int, x2: int, y2: int): void`	Draws a line from `(x1, y1)` to `(x2, y2)`.
`+drawRect(x: int, y: int, w: int, h: int): void`	Draws a rectangle with specified upper-left corner point at `(x,y)` and width `w` and height `h`.
`+fillRect(x: int, y: int, w: int, h: int): void`	Draws a filled rectangle with specified upper-left corner point at `(x, y)` and width `w` and height `h`.
`+drawRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void`	Draws a round-cornered rectangle with specified arc width `aw` and arc height `ah`.
`+fillRoundRect(x: int, y: int, w: int, h: int, aw: int, ah: int): void`	Draws a filled round-cornered rectangle with specified arc width `aw` and arc height `ah`.
`+draw3DRect(x: int, y: int, w: int, h: int, raised: boolean): void`	Draws a 3-D rectangle raised above the surface or sunk into the surface.
`+fill3DRect(x: int, y: int, w: int, h: int, raised: boolean): void`	Draws a filled 3-D rectangle raised above the surface or sunk into the surface.
`+drawOval(x: int, y: int, w: int, h: int): void`	Draws an oval bounded by the rectangle specified by the parameters `x`, `y`, `w`, and `h`.
`+fillOval(x: int, y: int, w: int, h: int): void`	Draws a filled oval bounded by the rectangle specified by the parameters `x`, `y`, `w`, and `h`.

java.awt.Graphics	
`+drawArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void`	Draws an arc conceived as part of an oval bounded by the rectangle specified by the parameters `x`, `y`, `w`, and `h`.
`+fillArc(x: int, y: int, w: int, h: int, startAngle: int, arcAngle: int): void`	Draws a filled arc conceived as part of an oval bounded by the rectangle specified by the parameters `x`, `y`, `w`, and `h`.
`+drawPolygon(xPoints: int[], yPoints: int[], nPoints: int): void`	Draws a closed polygon defined by arrays of `x`- and `y`-coordinates. Each pair of `(x[i], y[i])`-coordinates is a point.
`+fillPolygon(xPoints: int[], yPoints: int[], nPoints: int): void`	Draws a filled polygon defined by arrays of `x`- and `y`-coordinates. Each pair of `(x[i], y[i])`-coordinates is a point.
`+drawPolygon(g: Polygon): void`	Draws a closed polygon defined by a `Polygon` object.
`+fillPolygon(g: Polygon): void`	Draws a filled polygon defined by a `Polygon` object.
`+drawPolyline(xPoints: int[], yPoints: int[], nPoints: int): void`	Draws a polyline defined by arrays of `x`- and `y`-coordinates. Each pair of `(x[i], y[i])`-coordinates is a point.

Graphics class

- Provides a device-independent graphics interface for displaying figures and images on the screen on different platforms.
- Whenever a component (e.g., a button, a label, a panel) is displayed, the JVM automatically creates a **Graphics** object for the component and passes this object to invoke the **paintComponent** method.
- **protected void** paintComponent(Graphics g)
  ➔ defined in the **JComponent** class, is invoked whenever a component is first displayed or redisplayed.

Graphics class

- To draw things on a component,
  - define a class that extends **JPanel**
  - overrides its **paintComponent** method to specify what to draw.

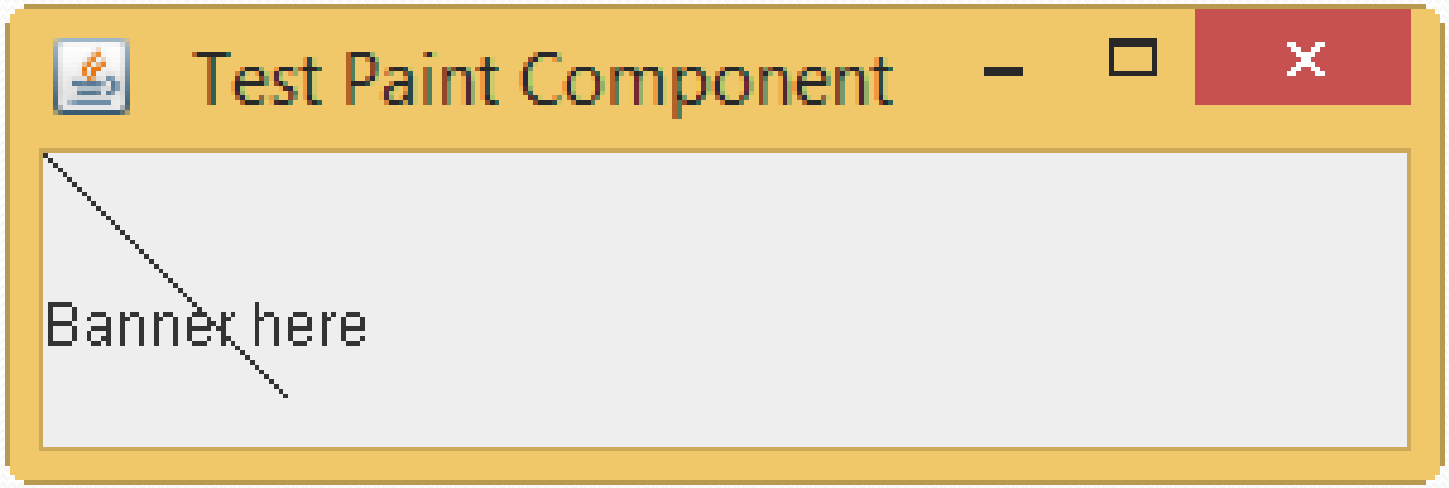
Example

```
public class TestPaintComponent extends JFrame {
      public TestPaintComponent() {
            add(new NewPanel());
            // for frame
            setTitle("Test Paint Component");
            setSize(300, 100);
            setLocationRelativeTo(null);
            setVisible(true);
      }
      public static void main(String[] args) {
            new TestPaintComponent();
      }
}
```

Example

```
class NewPanel extends JPanel {
      protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            g.drawLine(0, 0, 50, 50);
            g.drawString("Banner here", 0, 40);
      }
}
```

Example

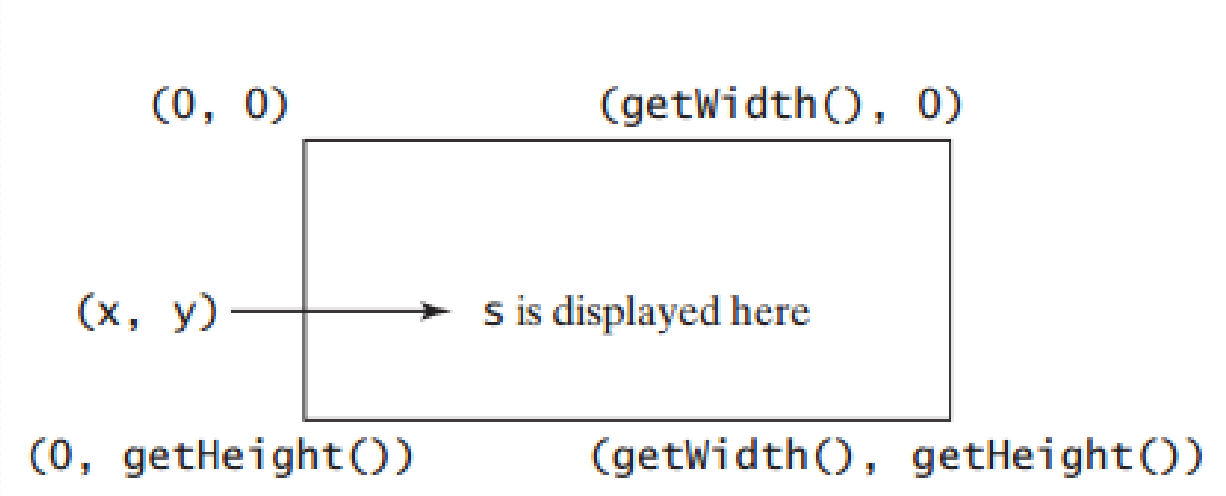


Graphics class

- The **paintComponent** method is automatically invoked to paint graphics when the component is first displayed or whenever the component needs to be redisplayed.
- Invoking **super.paintComponent(g)** ➔ to ensure that the viewing area is cleared before a new drawing is displayed.

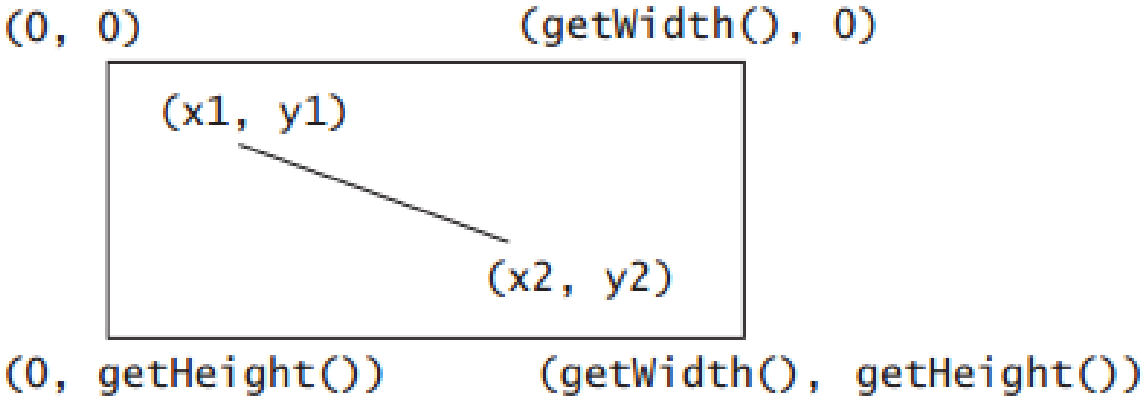
Drawing Strings

- The **drawString(String s, int x, int y)** method draws a string starting at the point **(x, y)**



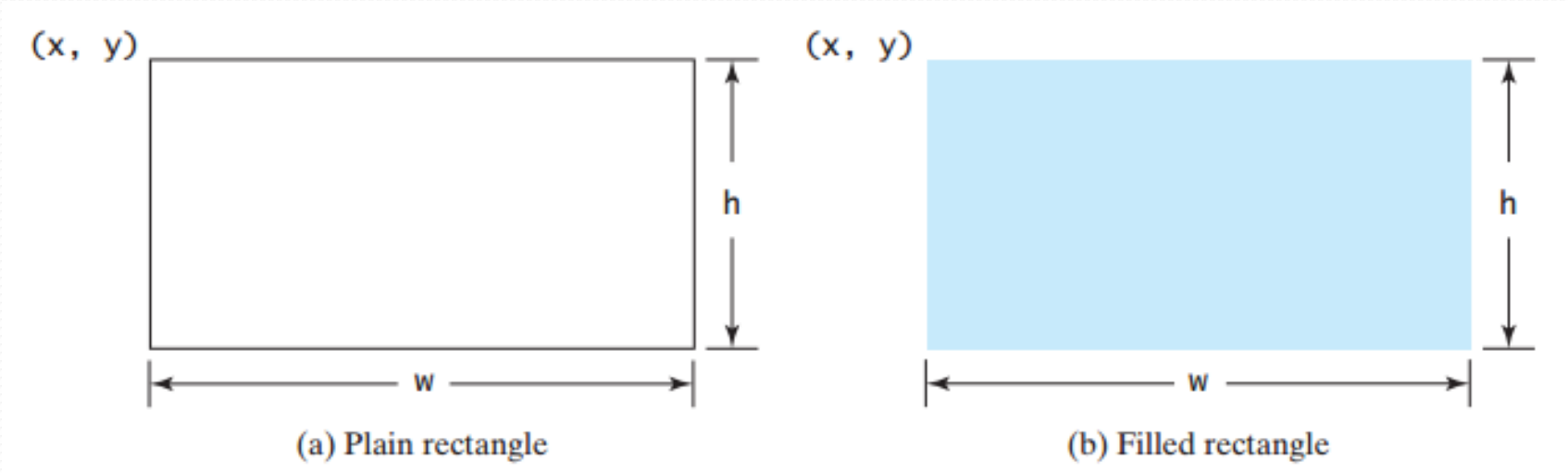
Draw Lines

- The **drawLine(int x1, int y1, int x2, int y2)** method draws a straight line from point **(x1, y1)** to point **(x2, y2)**



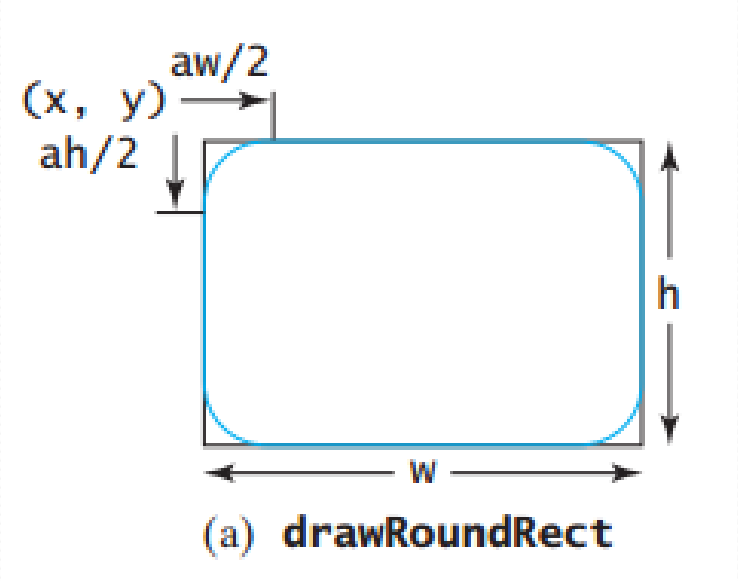
Draw Rectangles

- **drawRect(int x, int y, int w, int h)** method draws a plain rectangle.
- **fillRect(int x, int y, int w, int h)** method draws a filled rectangle



Draw Rectangles

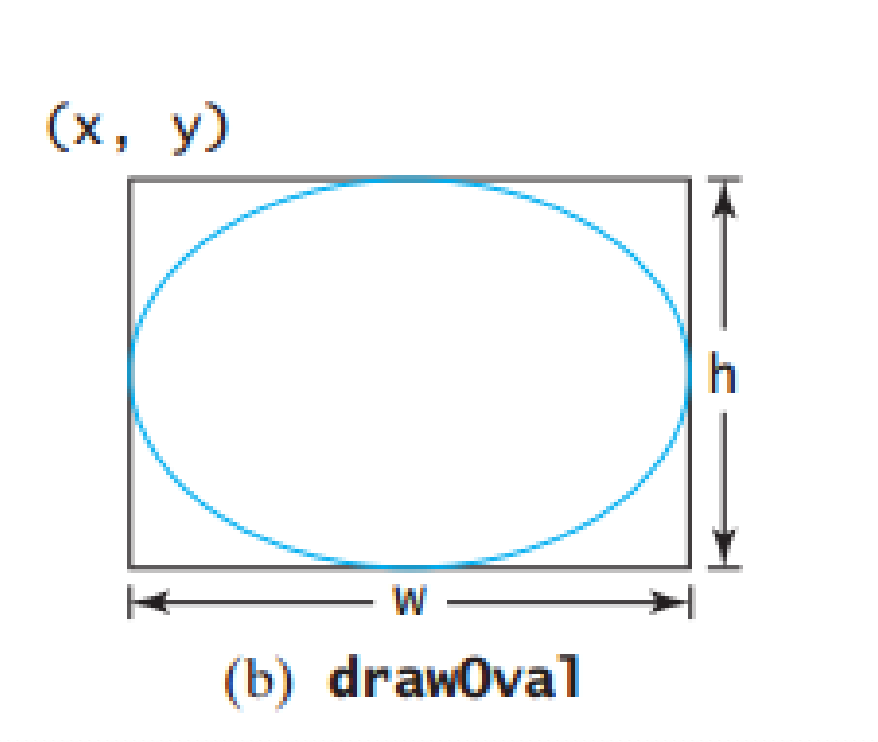
- **drawRoundRect(int x, int y, int w, int h, int aw, int ah)** method draws a round-cornered rectangle.
- **fillRoundRect(int x, int y, int w, int h, int aw, int ah)** method draws a filled round-cornered rectangle



(a) **drawRoundRect**

Draw Rectangles

- **drawOval(int x, int y, int w, int h)** method
- **fillOval(int x, int y, int w, int h)**

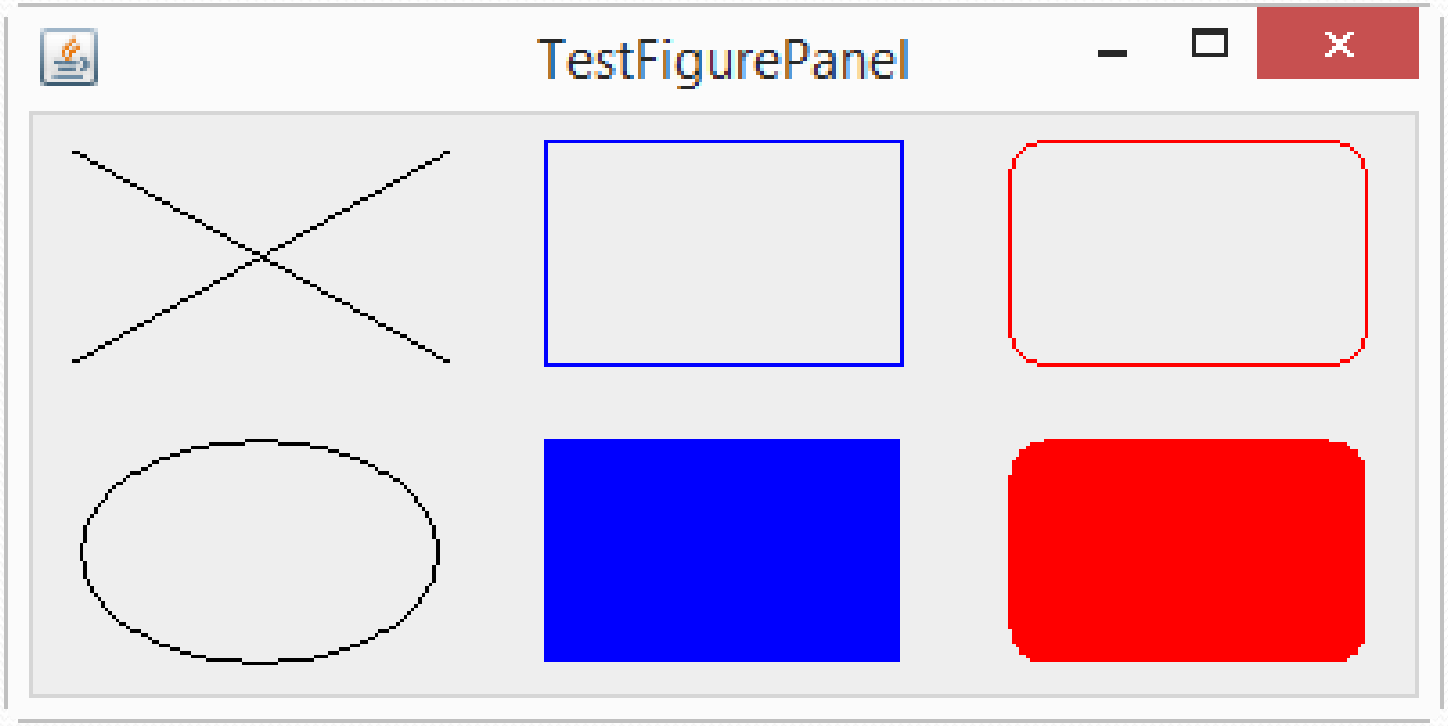


(b) **drawOval**

Draw Rectangles

- **draw3DRect(int x, int y, int w, int h, boolean raised)** method draws a 3D rectangle.
- **fill3DRect(int x, int y, int w, int h, boolean raised)** method draws a filled 3D rectangle

Example



Example

```
public class TestFigurePanel extends JFrame {
        public TestFigurePanel() {
            setLayout(new GridLayout(2, 3, 5, 5));
            add(new FigurePanel(FigurePanel.LINE));
            add(new FigurePanel(FigurePanel.RECTANGLE));
            add(new FigurePanel(FigurePanel.ROUND_RECTANGLE));
            add(new FigurePanel(FigurePanel.OVAL));
            add(new FigurePanel(FigurePanel.RECTANGLE, true));
            add(new FigurePanel(FigurePanel.ROUND_RECTANGLE,
true));
        }
```

```
    public static void main(String[] args) {
        TestFigurePanel frame = new
                TestFigurePanel();
        frame.setSize(400, 200);
        frame.setTitle("TestFigurePanel");
        frame.setLocationRelativeTo(null);

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

```
class FigurePanel extends JPanel {
	public static final int LINE = 1;
	public static final int RECTANGLE = 2;
	public static final int ROUND_RECTANGLE = 3;
	public static final int OVAL = 4;

	private int type = 1;
	private boolean filled = false;

	public FigurePanel() {
	}
	public FigurePanel(int type) {
		this.type = type;
	}
```

```
public FigurePanel(int type, boolean filled) {
      this.type = type;
      this.filled = filled;
}
public int getType() {
      return type;
}

public void setType(int type) {
      this.type = type;
}
```

```
public boolean isFilled() {
      return filled;
}

public void setFilled(boolean filled) {
      this.filled = filled;
}
public Dimension getPreferredSize() {
      return new Dimension(80, 80);
}
```

```
/** Draw a figure on the panel */
protected void paintComponent(Graphics g) {
      super.paintComponent(g);
// Get the appropriate size for the figure
      int width = getWidth();
      int height = getHeight();
      switch (type) {
         case LINE: // Display two cross lines
            g.setColor(Color.BLACK);
            g.drawLine(10, 10, width - 10, height - 10);
            g.drawLine(width - 10, 10, 10, height - 10);
            break;
```

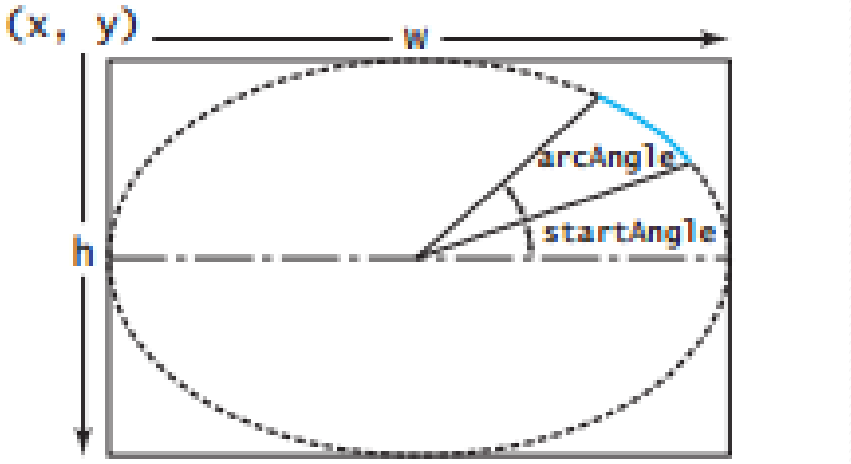
```
case RECTANGLE: // Display a rectangle
   g.setColor(Color.BLUE);
   if (filled)
      g.fillRect(10, 10, 70, 70);
   else
      g.drawRect(10, 10, 70, 70);
      break;
```

```
        case ROUND_RECTANGLE: // Display a round-cornered
rectangle
            g.setColor(Color.RED);
            if (filled)
               g.fillRoundRect(10, 10, 70, 70, 20, 20);
            else
               g.drawRoundRect(10, 10, 70, 70, 20, 20);
            break;
```

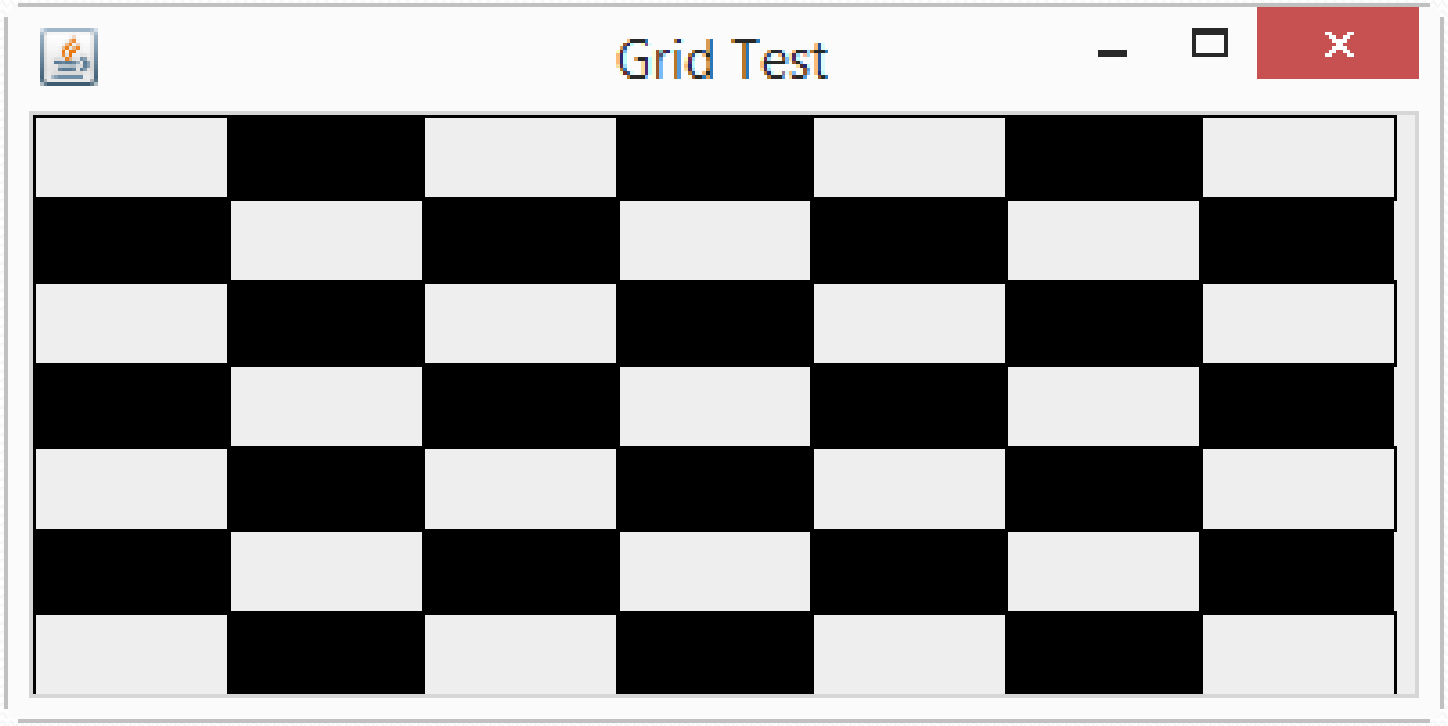
```
        case OVAL:
                g.setColor(Color.BLACK);
                if (filled)
                   g.fillOval(10, 10, 70, 70);
                else
                      g.drawOval(10, 10, 70, 70);
        }
    }
}
```

Drawing Arcs

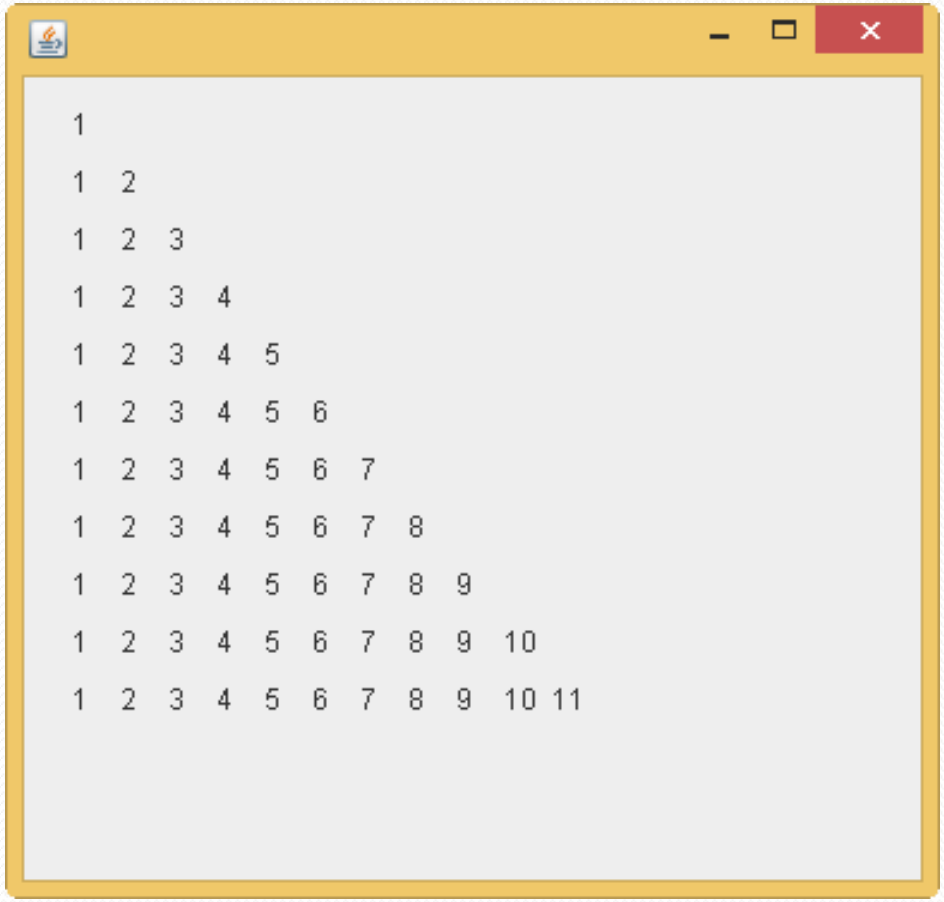
- **drawArc**(**int** x, **int** y, **int** w, **int** h, **int** startAngle, **int** arcAngle);
  fillArc(**int** x, **int** y, **int** w, **int** h, **int** startAngle, **int** arcAngle);



Exercise 1

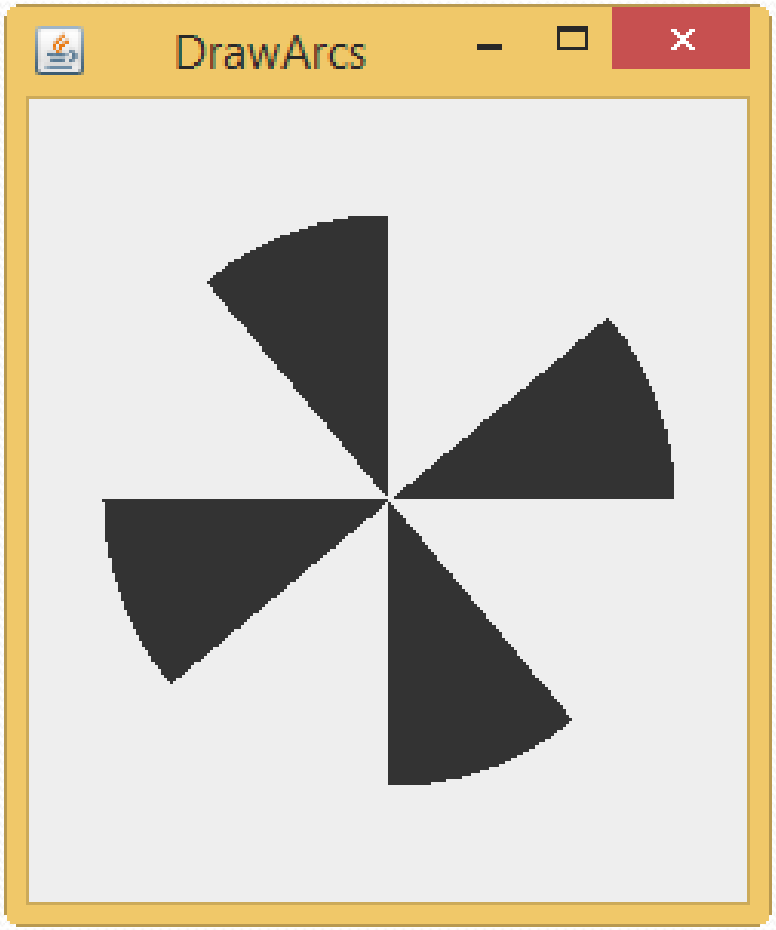


Exercise 2

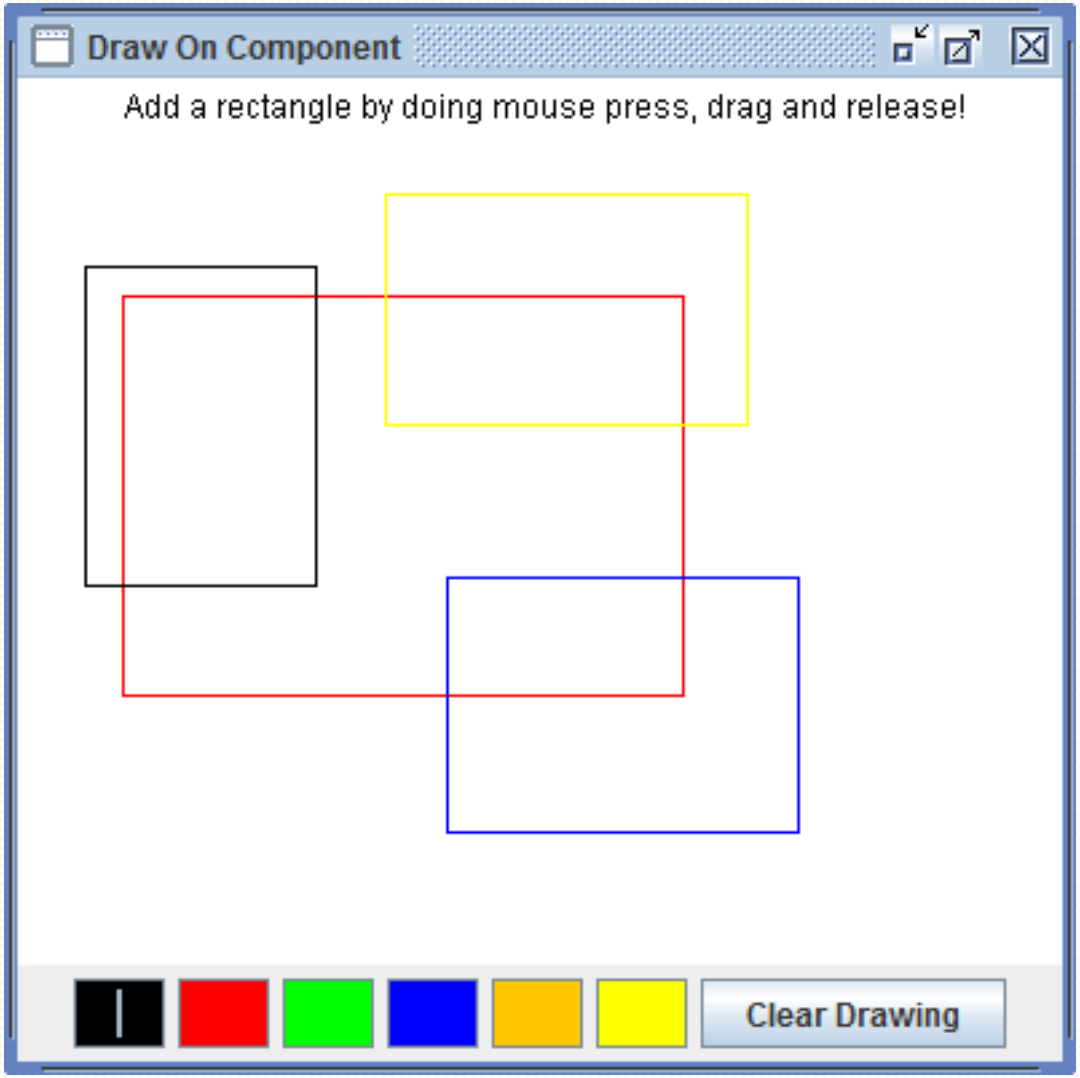

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4 5 6
1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11

Exercise 3

drawArc(**int** x, **int** y, **int** w, **int** h, **int** startAngle, **int** arcAngle);



Exercise 4: Vẽ trên Component



Exercise 5

Reference

- **Introduction to Java Programming** 8th , Y. Daniel Liang.