Java Application Programming

Week 8

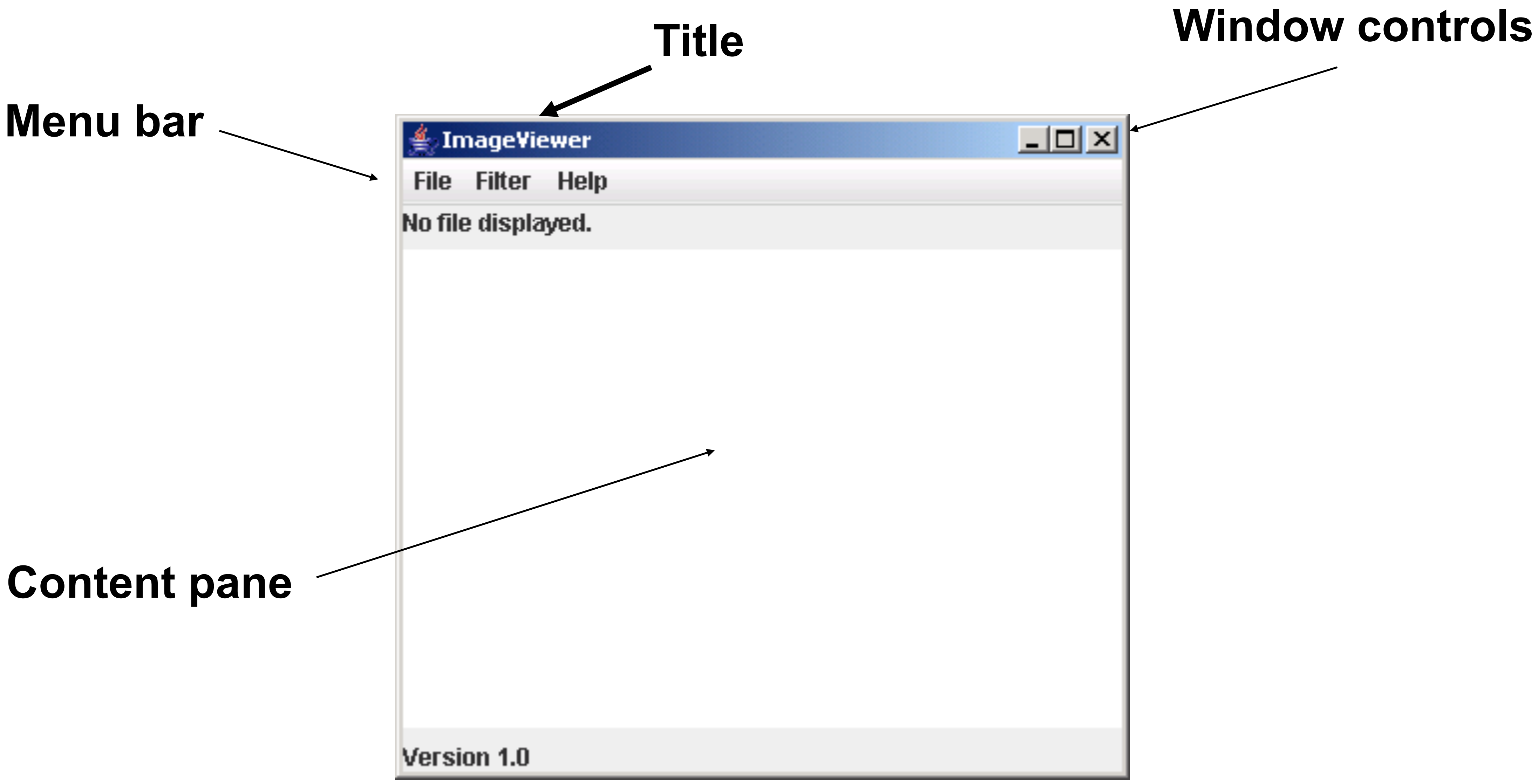
GUI II Event

Weng Kai

LayoutManager

Frame

Elements of a frame



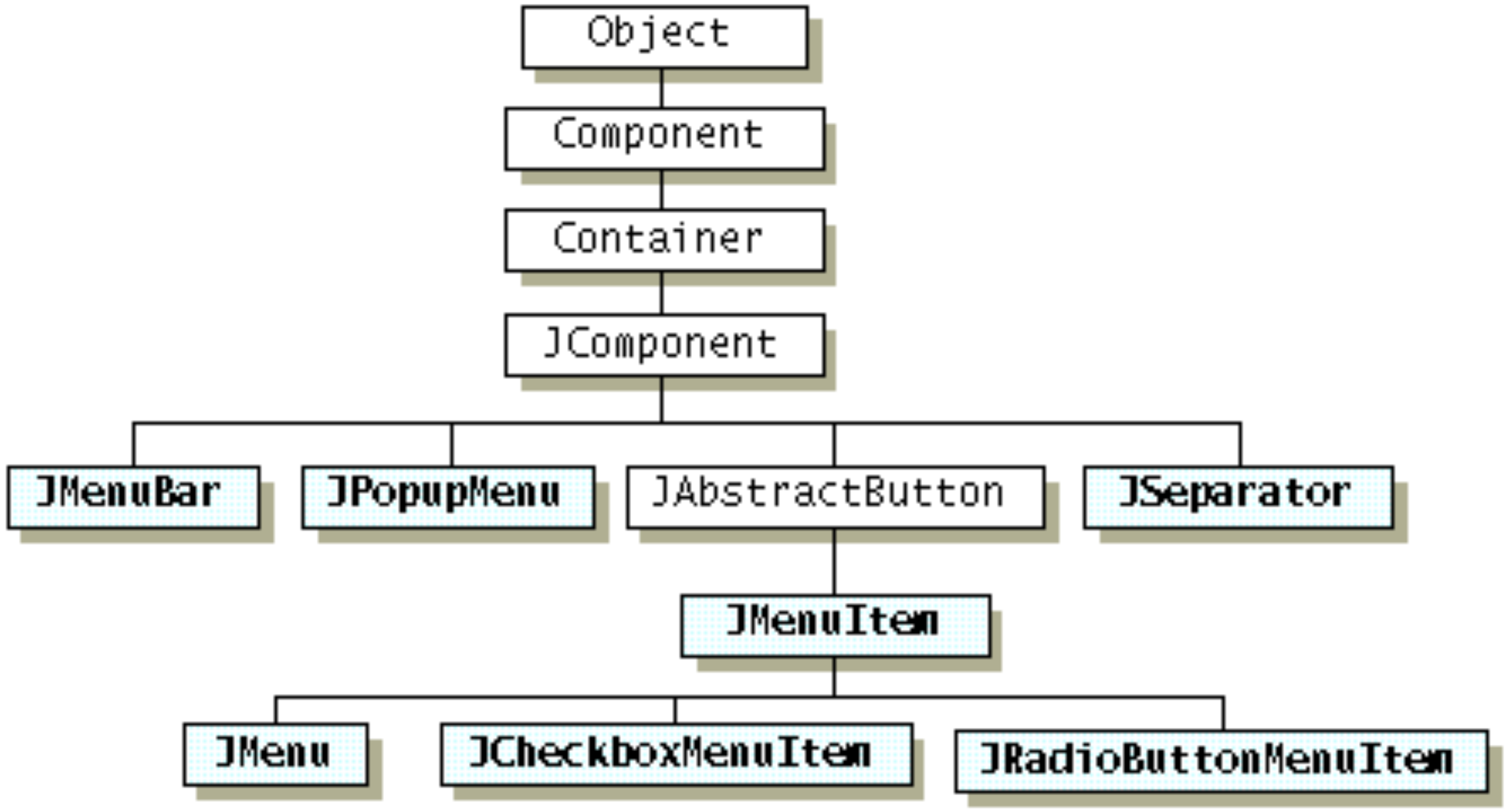
No file displayed.

Version 1.0

Adding menus

- `JMenuBar`
  - Displayed below the title.
  - Contains the menus.
- `JMenu`
  - e.g. *File*. Contains the menu items.
- `JMenuItem`
  - e.g. *Open*. Individual items.

The Menus


Object
Component
Container
JComponent
JMenuBar
JPopupMenu
JAbstractButton
JSeparator
JMenuItem
JMenu
JCheckboxMenuItem
JRadioButtonMenuItem

```
private void makeMenuBar(JFrame frame)
{
    JMenuBar menubar = new JMenuBar();
    frame.setJMenuBar(menubar);

    // create the File menu
    JMenu fileMenu = new JMenu("File");
    menubar.add(fileMenu);

    JMenuItem openItem = new JMenuItem("Open");
    fileMenu.add(openItem);

    JMenuItem quitItem = new JMenuItem("Quit");
    fileMenu.add(quitItem);
}
```

To create menus

- Create a JMenuBar, set it to the JFrame
- Create some JMenus, add them to the JMenuBar
- Create some JMenuItems, add them to the JMenus
- Add ActionListener to every JMenuItems to receive the event

Event handling

- Events correspond to user interactions with components.
- Components are associated with different event types.
  - Frames are associated with **`WindowEvent`**.
  - Menus are associated with **`ActionEvent`**.
- Objects can be notified when an event occurs.
  - Such objects are called *listeners*.

Centralized event receipt

- A single object handles all events.
  - Implements the **`ActionListener`** interface.
  - Defines an **`actionPerformed`** method.
- It registers as a listener with each component.
  - **`item.addActionListener(this)`**
- It has to work out which component has dispatched the event.

```
public class ImageViewer implements ActionListener
{
    …
    public void actionPerformed(ActionEvent e)
    {
        String command = e.getActionCommand();
        if(command.equals("Open")) {
            …
        }
        else if (command.equals("Quit")) {
            …
        }
        …
    }
    …
    private void makeMenuBar(Jframe frame)
    {
        …
        openItem.addActionListener(this);
        …
    }
}
```

Centralized event handling

- The approach works.
- It is used, so you should be aware of it.
- However …
  - It does not scale well.
  - Identifying components by their text is fragile.
- An alternative approach is preferred.

Anonymous action listener

```
JMenuItem openItem = new JMenuItem("Open");

openItem.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
         openFile();
     }
});
```

Inner classes

- Since Java 1.1, it is possible to place a class definition within another class definition.
- Case: Parcel1.java
- Case: Parcel3.java

Where are inner classes?

- A class defined within a method. Destination.java Contents.java Wrapping.java Parcel4.java
- A class defined within a scope inside a method. Parcel5.java
- An anonymouse class implementing an interface. Parcel6.java

- An anonymouse class extending a class that has a non-default constructor. Parcel7.java
- An anonymouse class that perform fields initialization. Parcel8.java
- An anonymouse class that perform construction using instance initialization. Parcel9.java

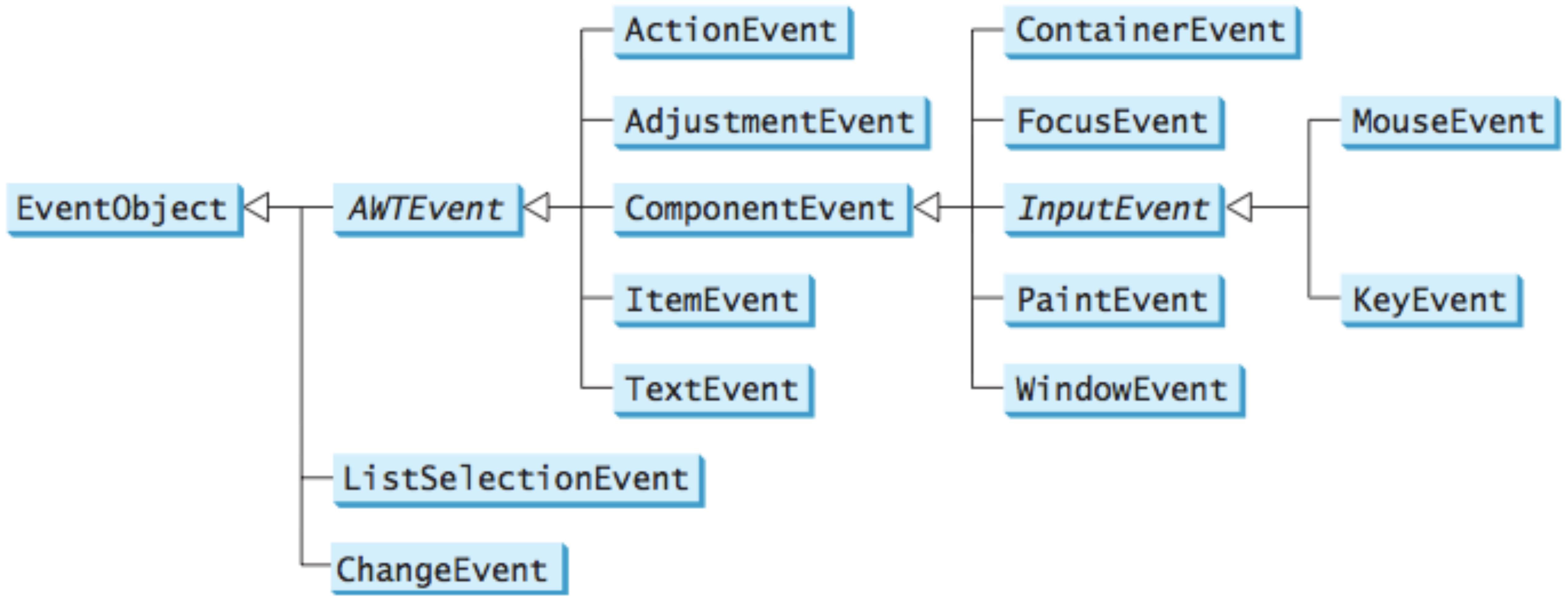
Outter class?

- As a member, the inner class can access everything of its outter class.
- case: Sequence.java

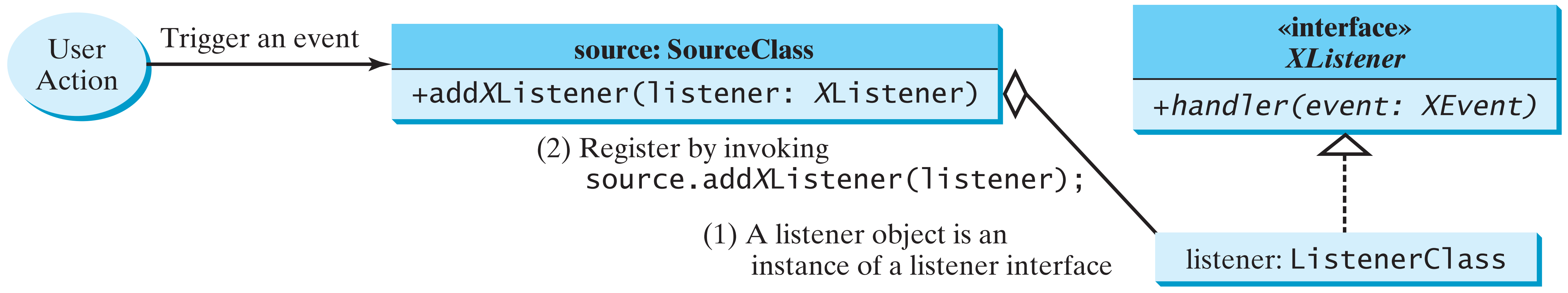
Anonymous action listener

```
JMenuItem openItem = new JMenuItem("Open");

openItem.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
         openFile();
     }
});
```

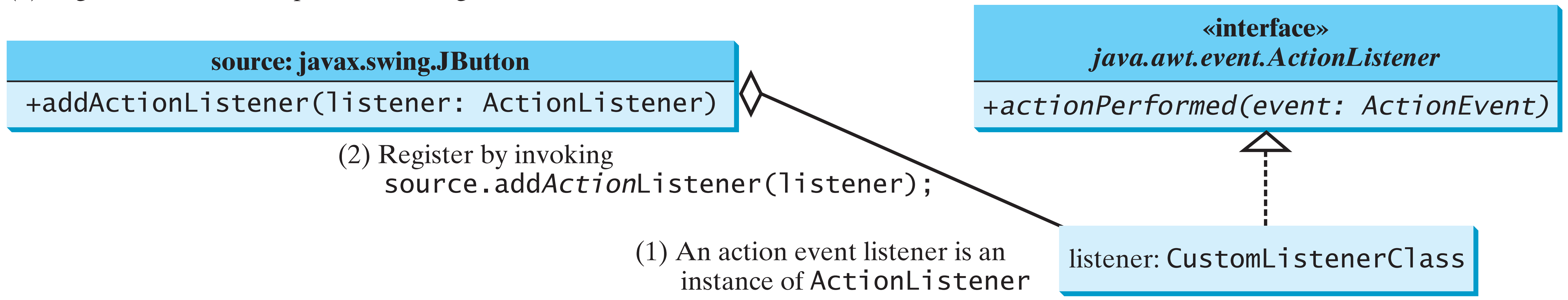
What Events We Have?



User Action	*Source Object*	*Event Type Fired*
Click a button	`JButton`	`ActionEvent`
Press return on a text field	`JTextField`	`ActionEvent`
Select a new item	`JComboBox`	`ItemEvent`, `ActionEvent`
Select item(s)	`JList`	`ListSelectionEvent`
Click a check box	`JCheckBox`	`ItemEvent`, `ActionEvent`
Click a radio button	`JRadioButton`	`ItemEvent`, `ActionEvent`
Select a menu item	`JMenuItem`	`ActionEvent`
Move the scroll bar	`JScrollBar`	`AdjustmentEvent`
Move the scroll bar	`JSlider`	`ChangeEvent`
Window opened, closed, iconified, deiconified, or closing	`Window`	`WindowEvent`
Mouse pressed, released, clicked, entered, or exited	`Component`	`MouseEvent`
Mouse moved or dragged	`Component`	`MouseEvent`
Key released or pressed	`Component`	`KeyEvent`
Component added or removed from the container	`Container`	`ContainerEvent`
Component moved, resized, hidden, or shown	`Component`	`ComponentEvent`
Component gained or lost focus	`Component`	`FocusEvent`


User Action
Trigger an event
source: SourceClass
+addXListener(listener: XListener)
(2) Register by invoking source.addXListener(listener);
«interface» XListener
+handler(event: XEvent)
(1) A listener object is an instance of a listener interface
listener: ListenerClass

(a) A generic source component with a generic listener


source: javax.swing.JButton
+addActionListener(listener: ActionListener)
(2) Register by invoking source.addActionListener(listener);
«interface» java.awt.event.ActionListener
+actionPerformed(event: ActionEvent)
(1) An action event listener is an instance of ActionListener
listener: CustomListenerClass

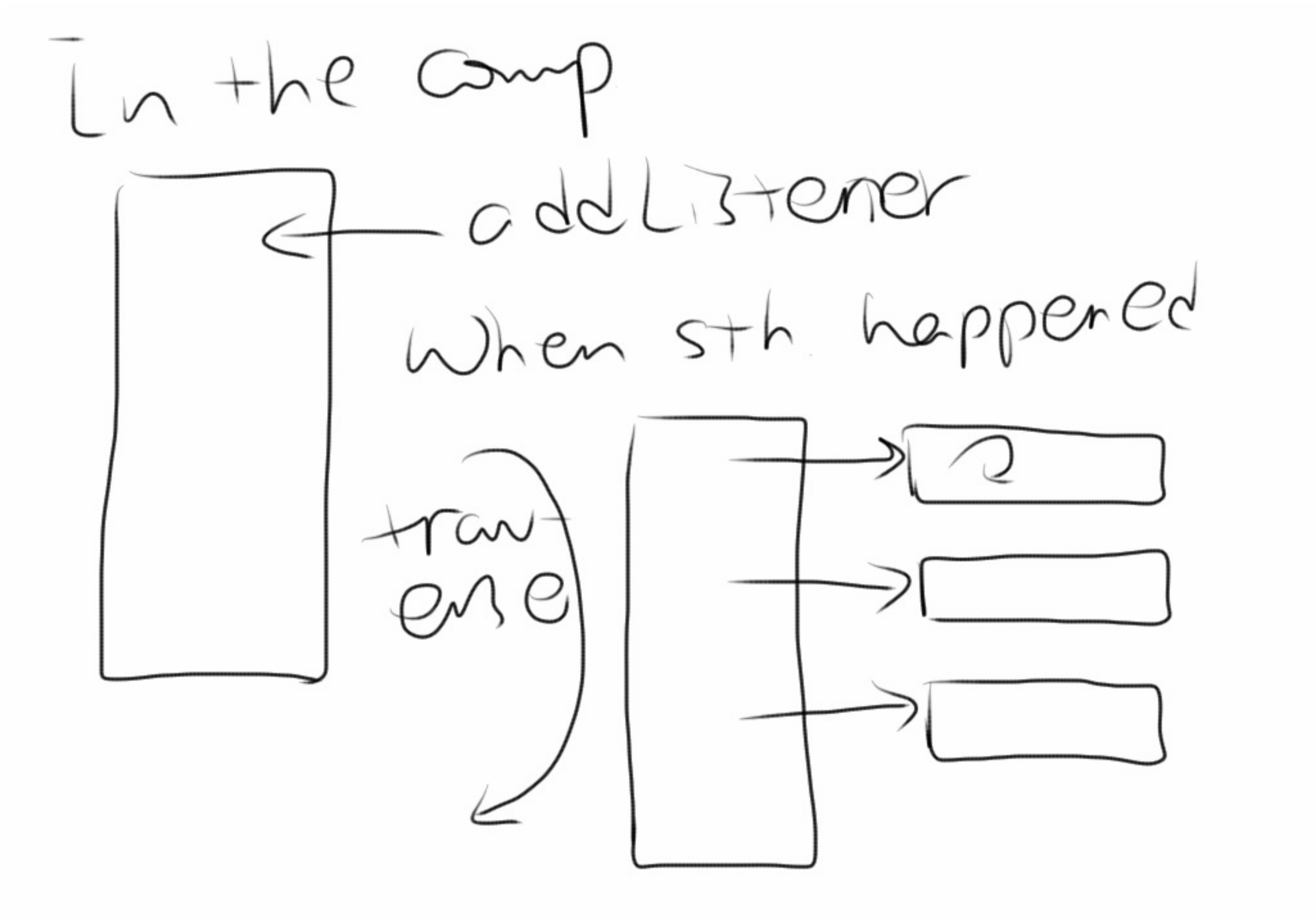
(b) A `JButton` source component with an `ActionListener`

Dynamic Events

- There is more than one listener attached to each components.
- During the execution of the program, listeners are dynamically added and removed.

DynamicEvents.java

How Does It Work?



Java Application Design

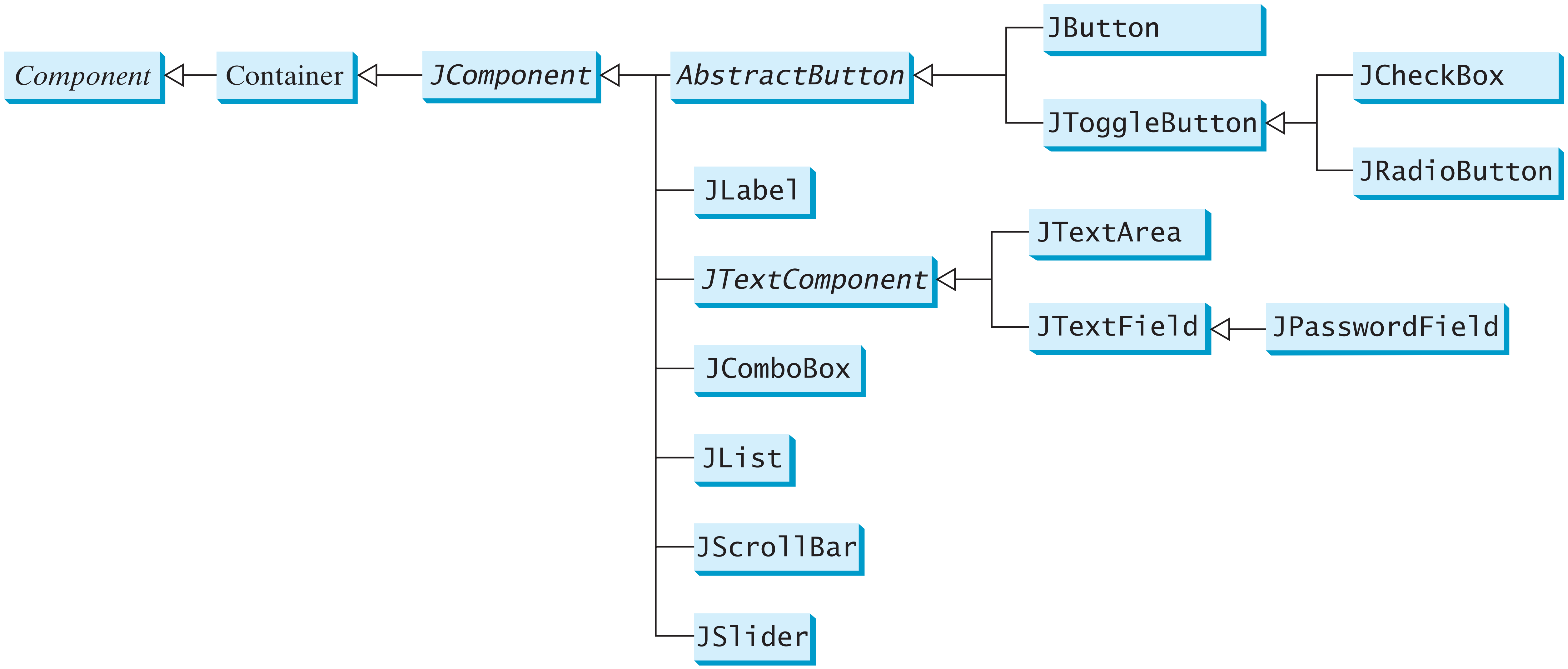
Week 7

GUI III components and dialogs

Weng Kai

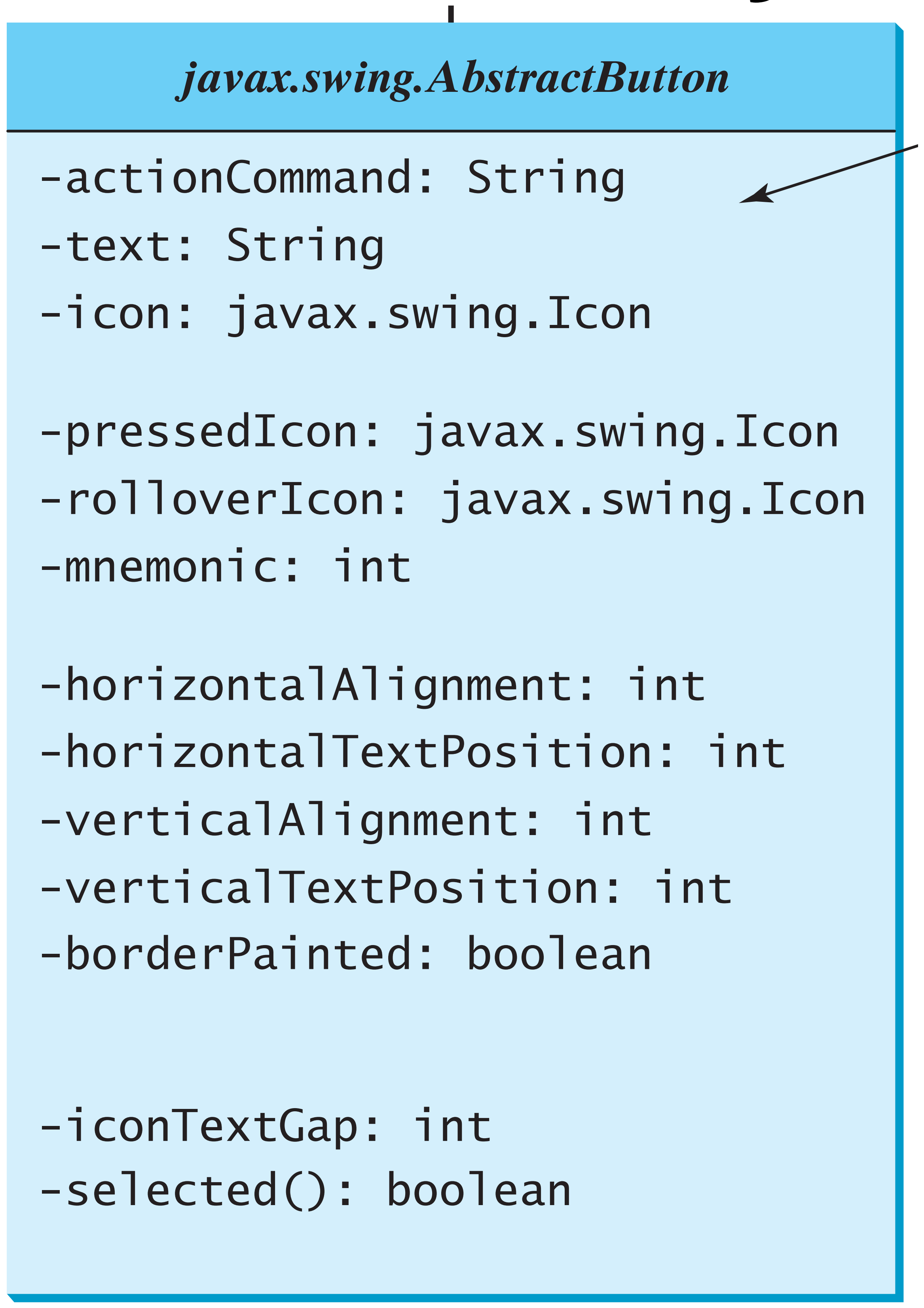
http://fm.zju.edu.cn

Components



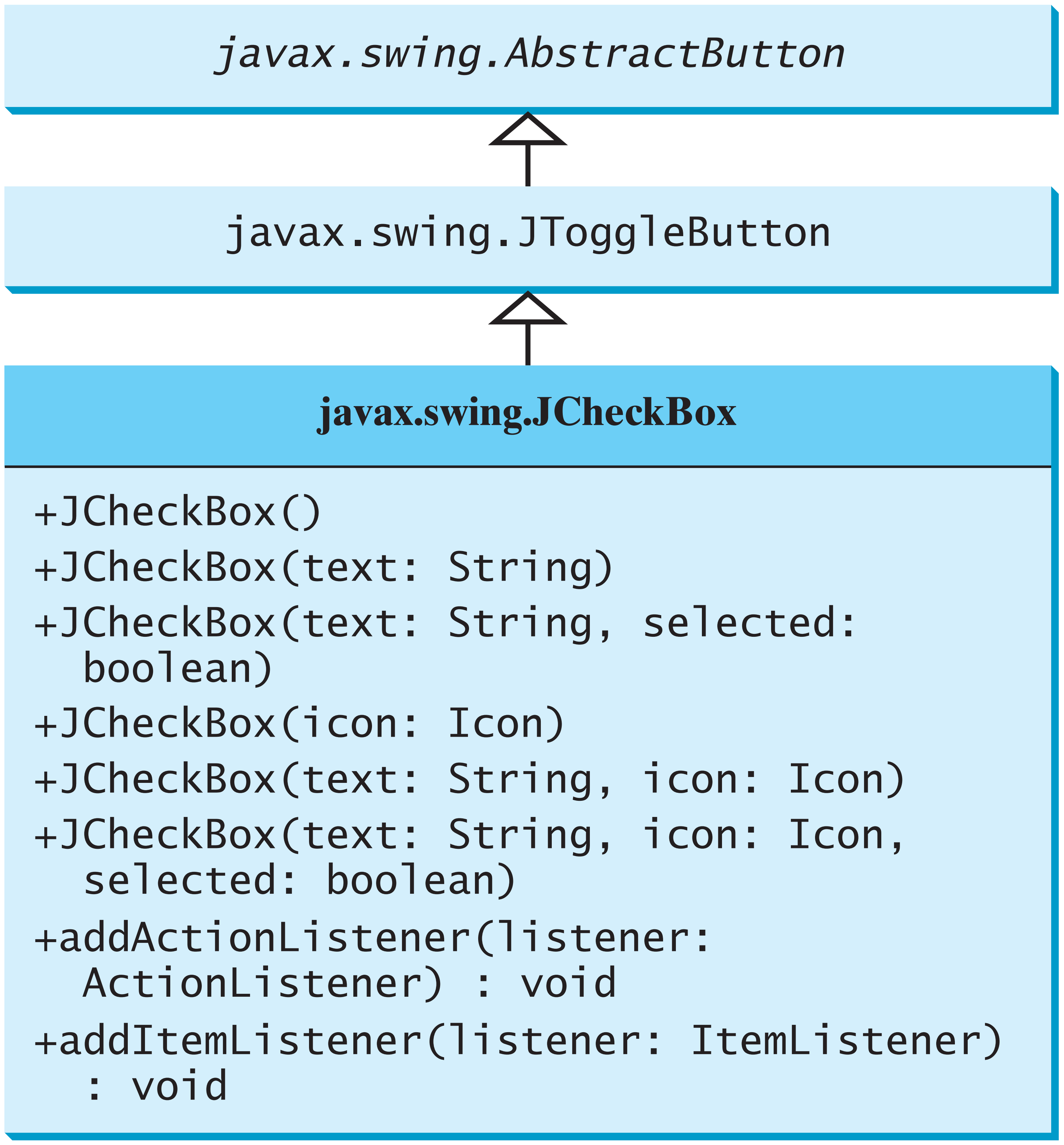
Component
Container
JComponent
AbstractButton
JButton
JToggleButton
JCheckBox
JRadioButton
JLabel
JTextComponent
JTextArea
JTextField
JPasswordField
JComboBox
JList
JScrollBar
JSlider

JButton


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

- TestButtonIcons.java
- ButtonDemo.java

CheckBoxes

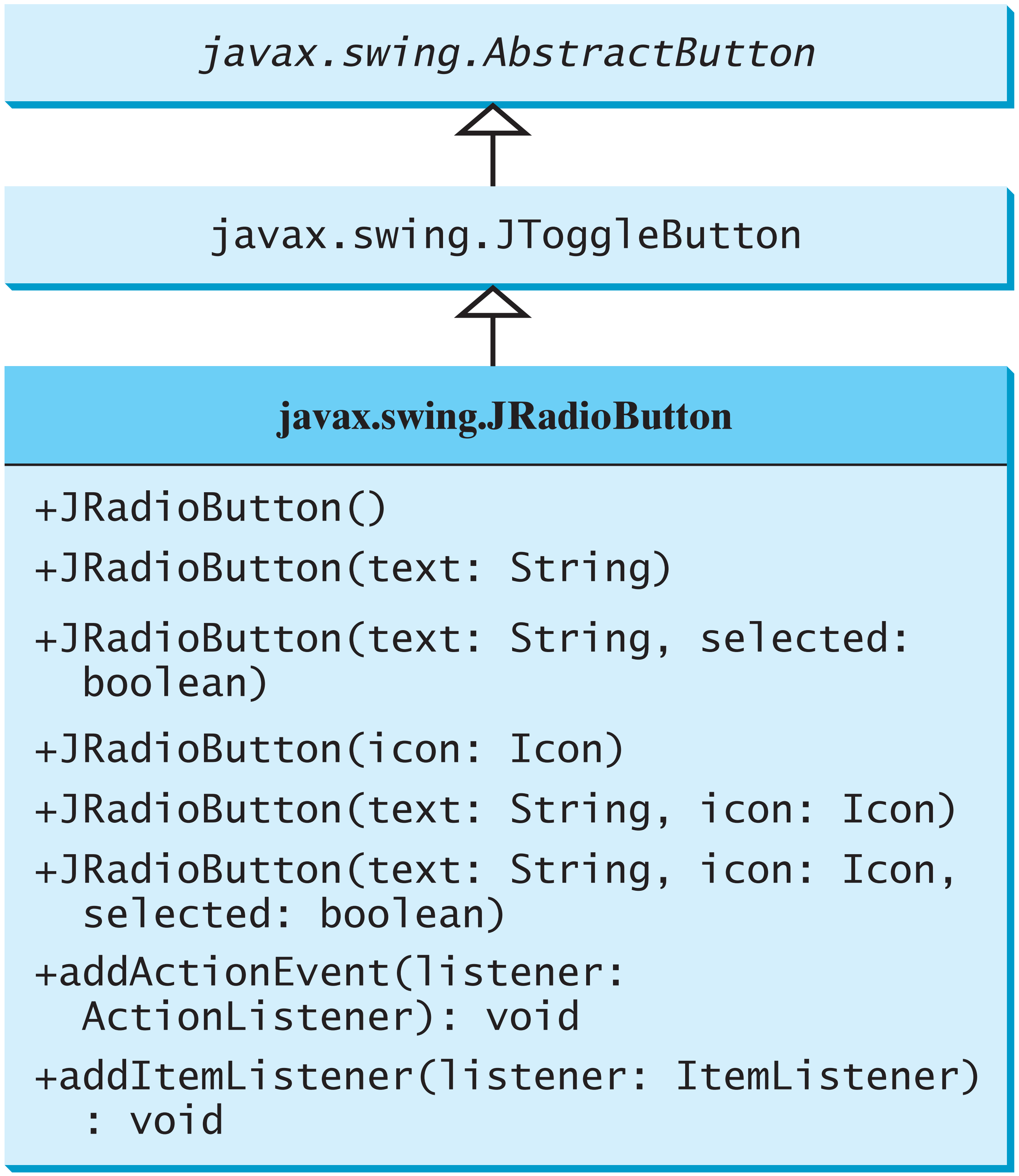


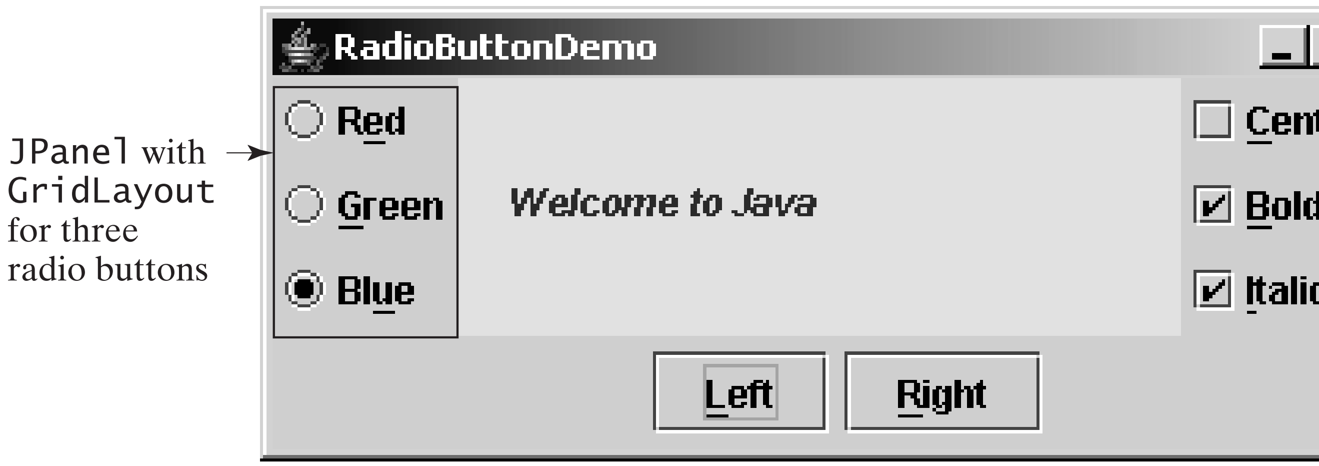
javax.swing.AbstractButton
javax.swing.JToggleButton
javax.swing.JCheckBox

```
+JCheckBox()
+JCheckBox(text: String)
+JCheckBox(text: String, selected:
  boolean)
+JCheckBox(icon: Icon)
+JCheckBox(text: String, icon: Icon)
+JCheckBox(text: String, icon: Icon,
  selected: boolean)
+addActionListener(listener:
  ActionListener) : void
+addItemListener(listener: ItemListener)
  : void
```

- CheckBoxDemo.java

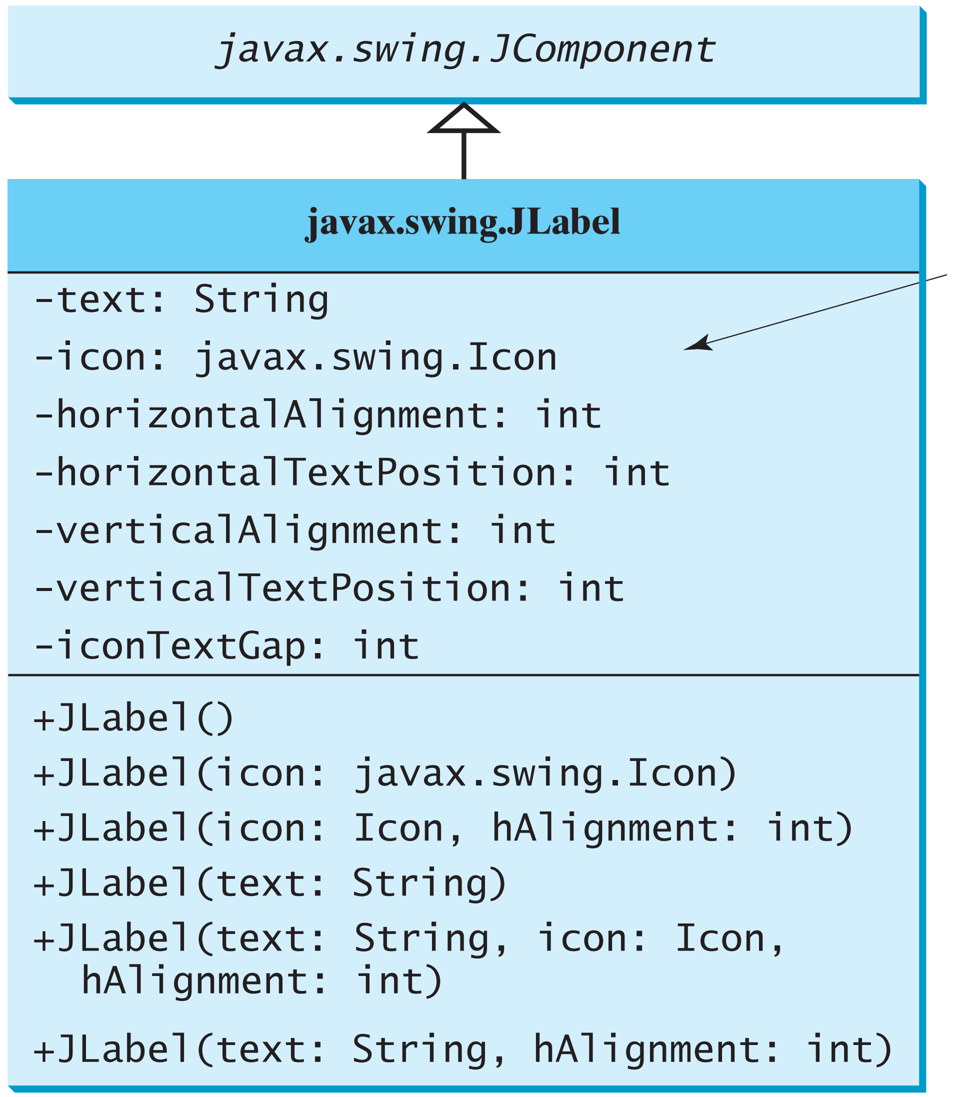
RadioButton


javax.swing.AbstractButton
javax.swing.JToggleButton
javax.swing.JRadioButton
+JRadioButton()
+JRadioButton(text: String)
+JRadioButton(text: String, selected: boolean)
+JRadioButton(icon: Icon)
+JRadioButton(text: String, icon: Icon)
+JRadioButton(text: String, icon: Icon, selected: boolean)
+addActionEvent(listener: ActionListener): void
+addItemListener(listener: ItemListener) : void



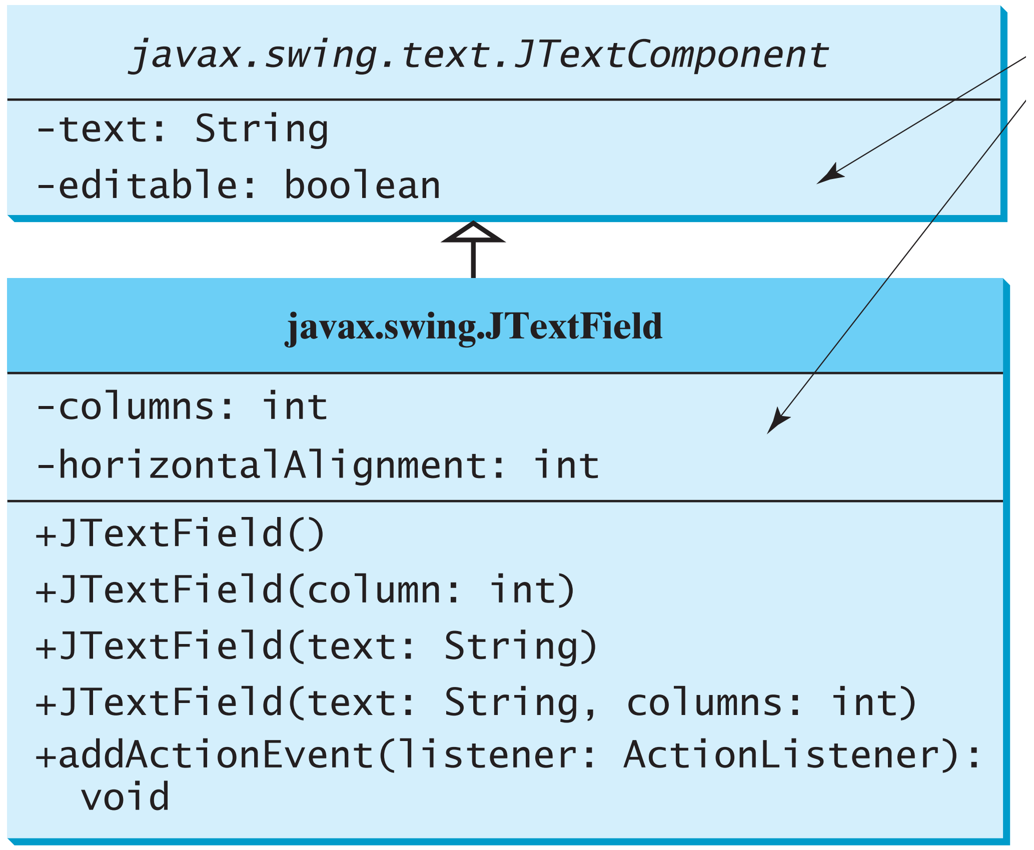
- RadioButtonDemo.java

Label

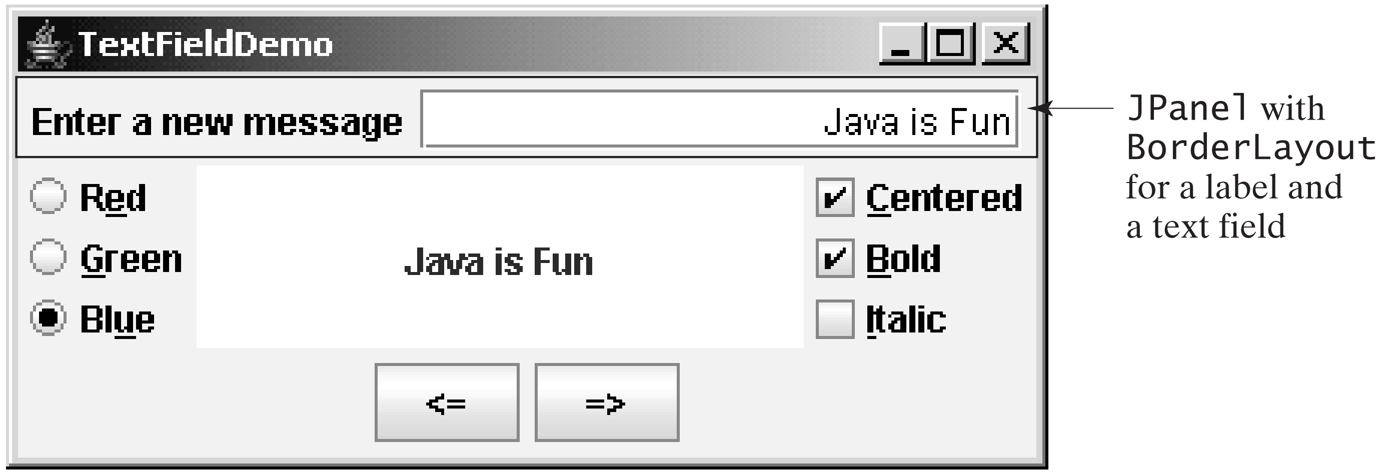

javax.swing.JComponent
javax.swing.JLabel
-text: String
-icon: javax.swing.Icon
-horizontalAlignment: int
-horizontalTextPosition: int
-verticalAlignment: int
-verticalTextPosition: int
-iconTextGap: int
+JLabel()
+JLabel(icon: javax.swing.Icon)
+JLabel(icon: Icon, hAlignment: int)
+JLabel(text: String)
+JLabel(text: String, icon: Icon, hAlignment: int)
+JLabel(text: String, hAlignment: int)

```java
// Create an image icon from an image file
ImageIcon icon = new ImageIcon("image/grapes.gif");
// Create a label with a text, an icon,
// with centered horizontal alignment
JLabel jlbl = new JLabel("Grapes", icon, SwingConstants.CENTER);
//Set label's text alignment and gap between text and icon
jlbl.setHorizontalTextPosition(SwingConstants.CENTER);
jlbl.setVerticalTextPosition(SwingConstants.BOTTOM);
jlbl.setIconTextGap(5);
```

TextField

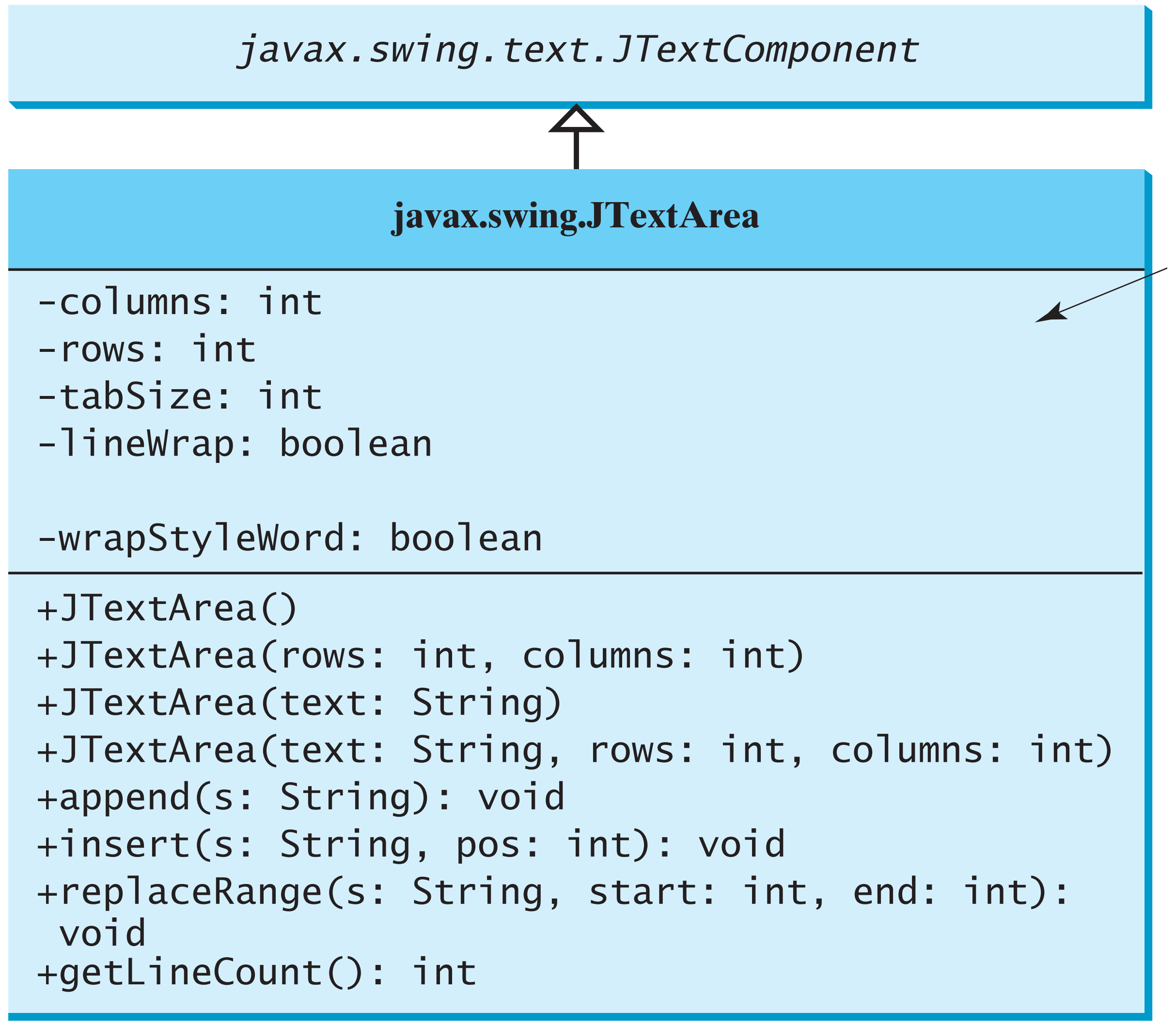


javax.swing.text.JTextComponent
-text: String
-editable: boolean
javax.swing.JTextField
-columns: int
-horizontalAlignment: int
+JTextField()
+JTextField(column: int)
+JTextField(text: String)
+JTextField(text: String, columns: int)
+addActionEvent(listener: ActionListener): void



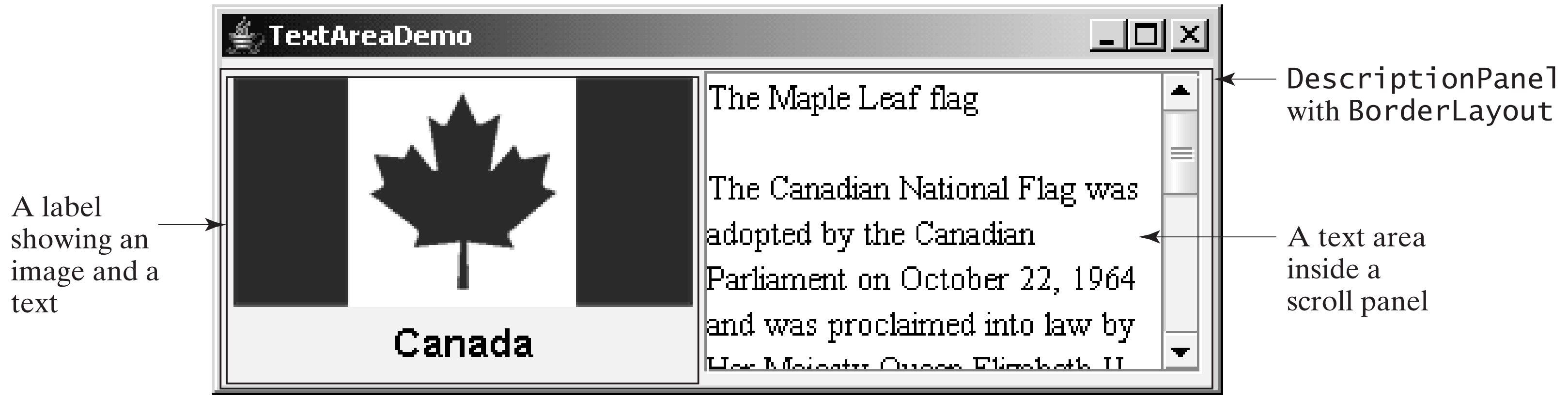
- TextFieldDemo.java

TextArea



```
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
```

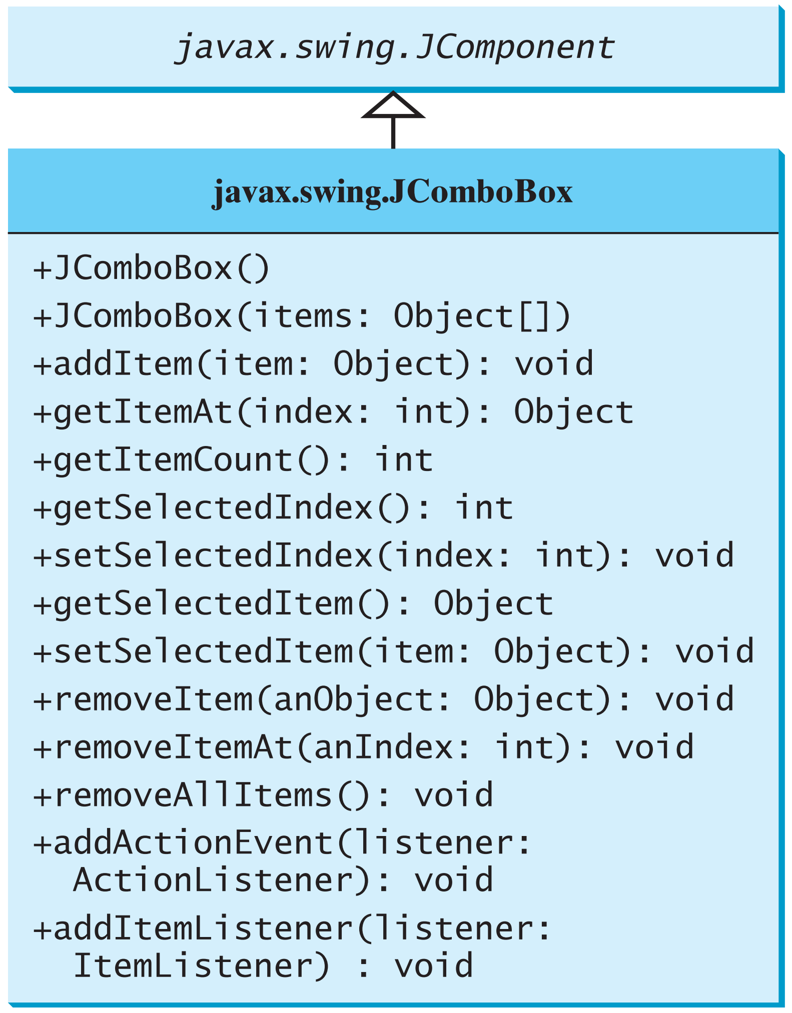
- DescriptionPanel.java
- TextAreaDemo.java


TextAreaDemo
Canada
A label showing an image and a text
DescriptionPanel with BorderLayout
A text area inside a scroll panel

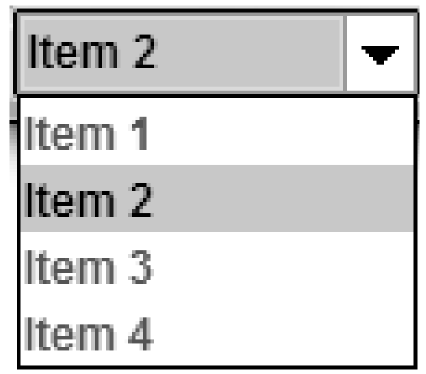
The Maple Leaf flag

The Canadian National Flag was adopted by the Canadian Parliament on October 22, 1964 and was proclaimed into law by

ComboBox

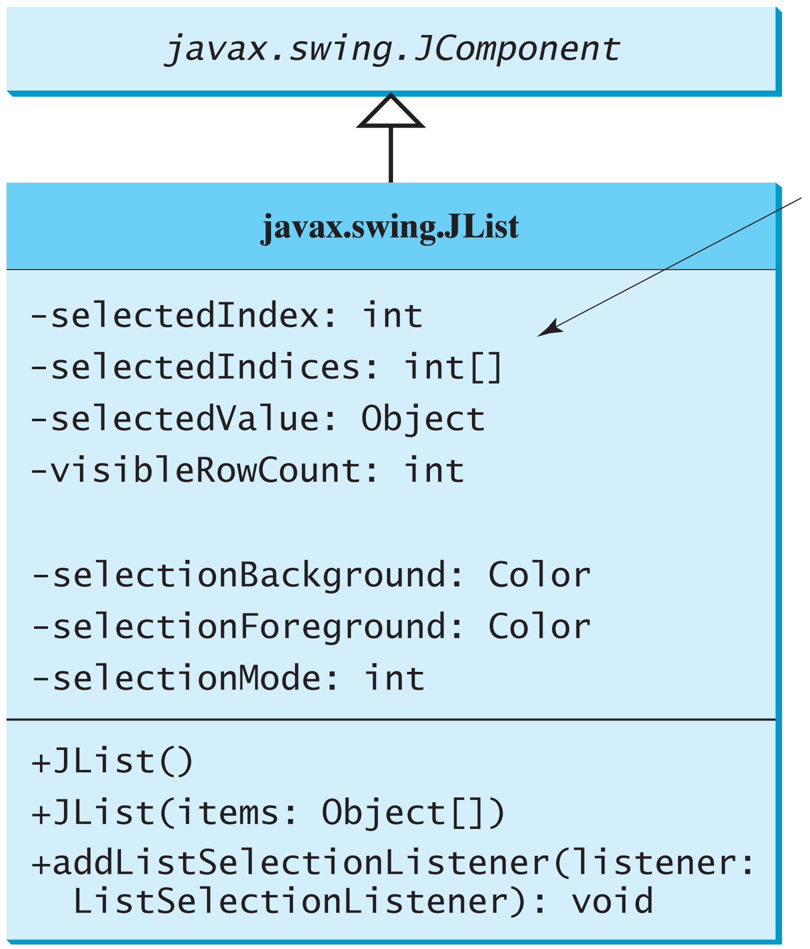


```
javax.swing.JComponent
        △
        |
javax.swing.JComboBox
------------------------------
+JComboBox()
+JComboBox(items: Object[])
+addItem(item: Object): void
+getItemAt(index: int): Object
+getItemCount(): int
+getSelectedIndex(): int
+setSelectedIndex(index: int): void
+getSelectedItem(): Object
+setSelectedItem(item: Object): void
+removeItem(anObject: Object): void
+removeItemAt(anIndex: int): void
+removeAllItems(): void
+addActionEvent(listener:
  ActionListener): void
+addItemListener(listener:
  ItemListener) : void
```

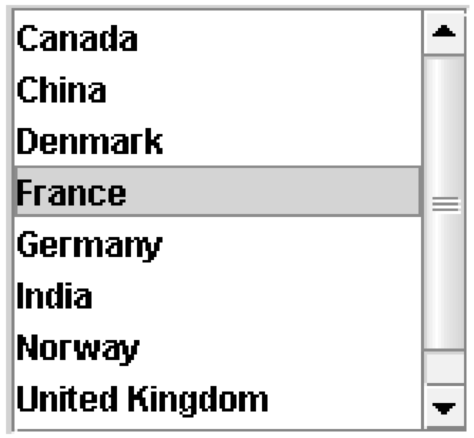


- ComboBoxDemo.java

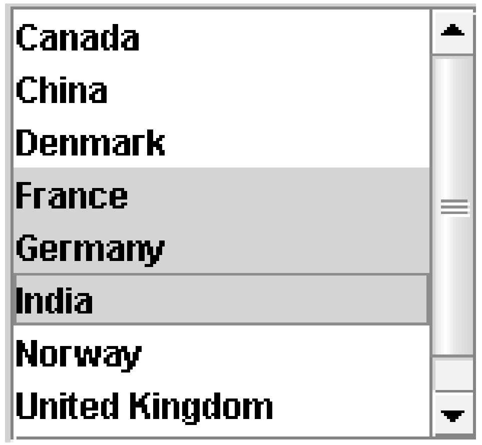
List



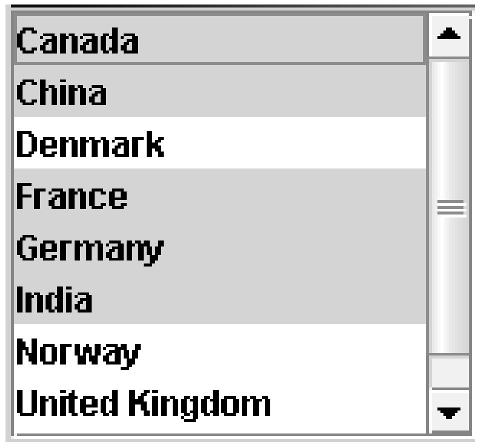
```
javax.swing.JComponent

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
```



(a) Single selection



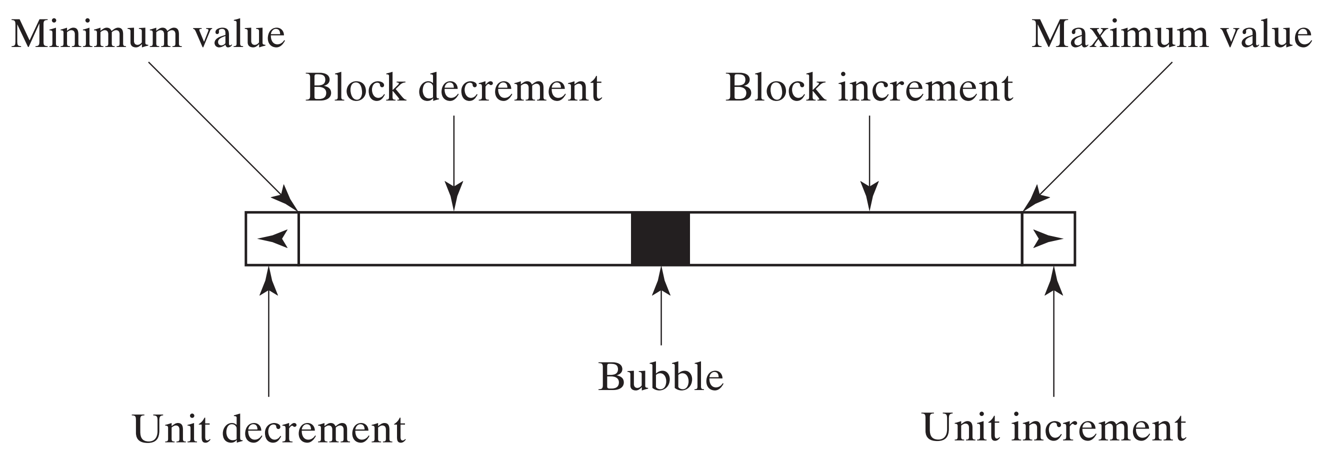
(b) Single-interval selection

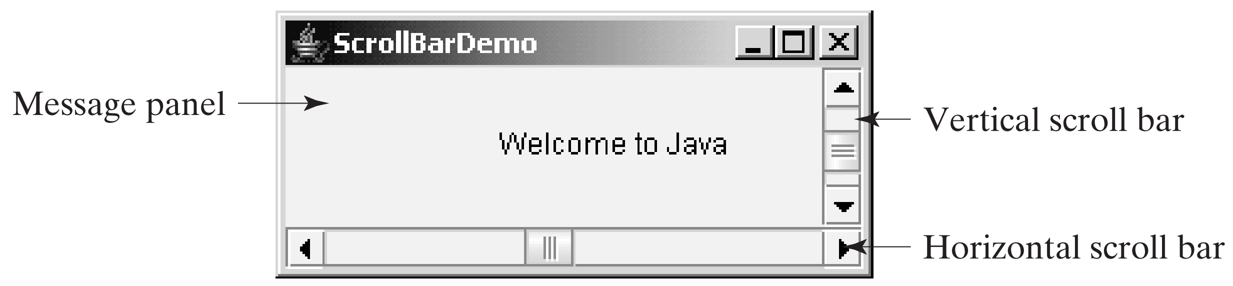


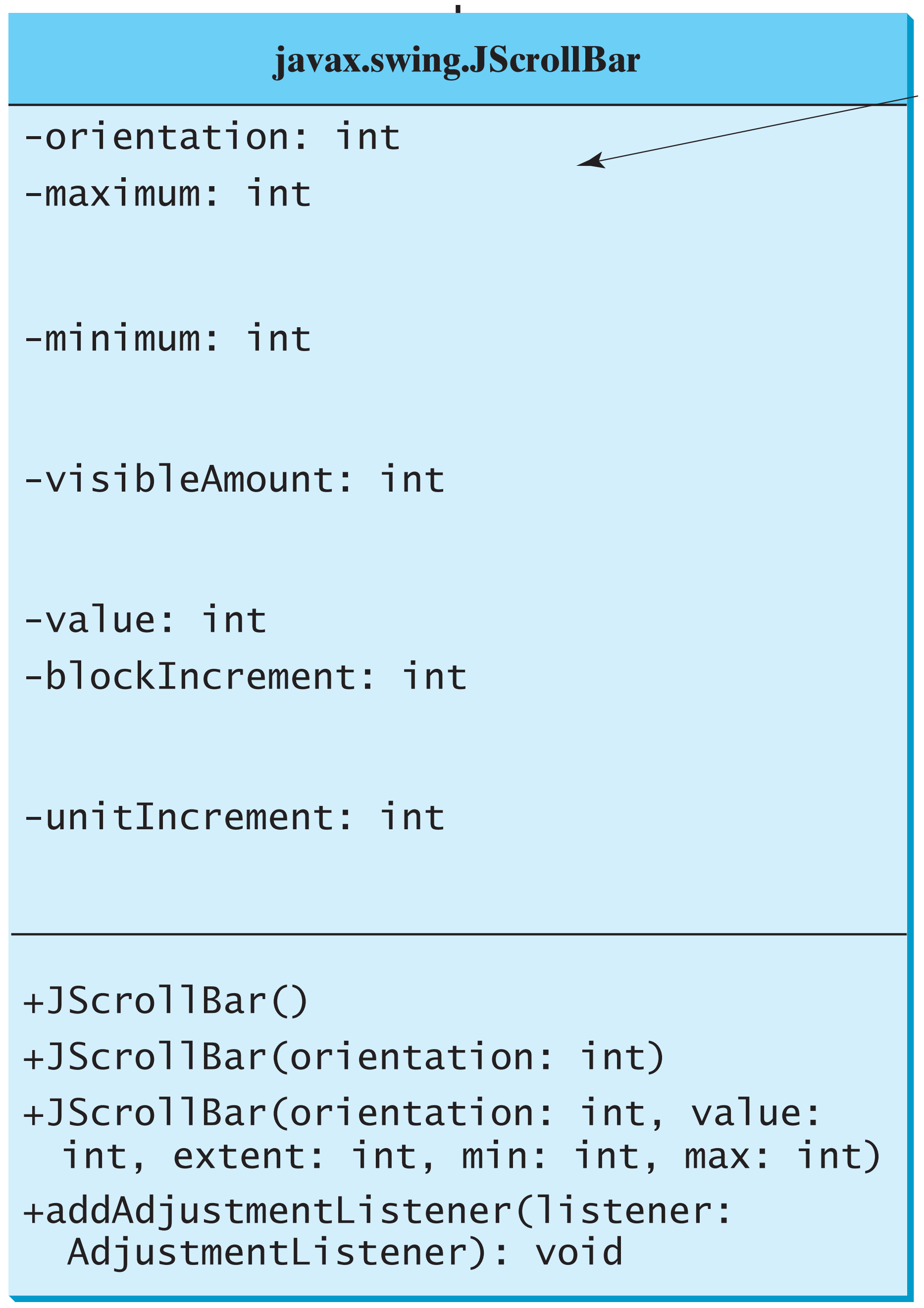
(c) Multiple-interval selection

- ListDemo.java

ScrollBar







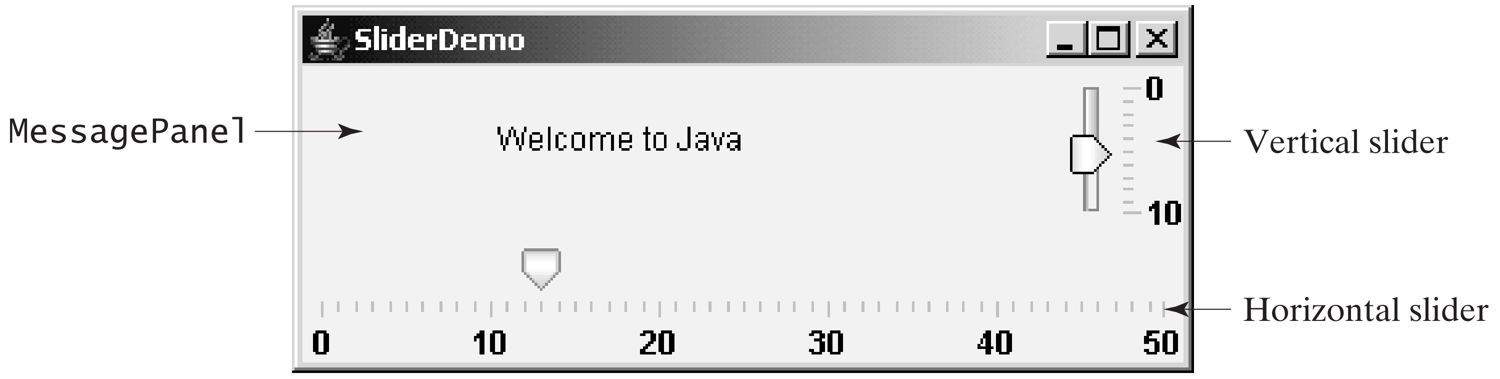
javax.swing.JScrollBar

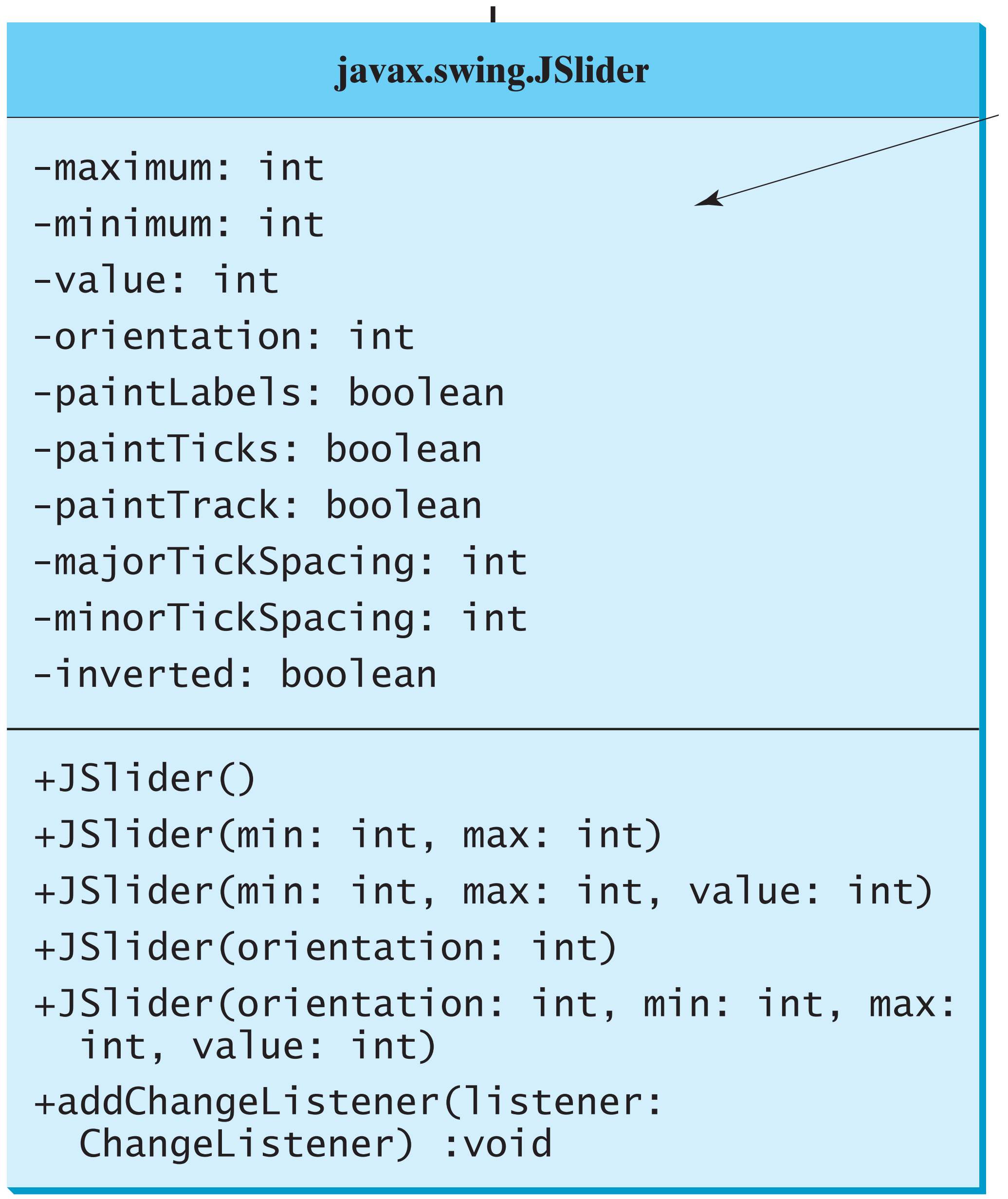
```
-orientation: int
-maximum: int
-minimum: int
-visibleAmount: int
-value: int
-blockIncrement: int
-unitIncrement: int
```

```
+JScrollBar()
+JScrollBar(orientation: int)
+JScrollBar(orientation: int, value:
  int, extent: int, min: int, max: int)
+addAdjustmentListener(listener:
  AdjustmentListener): void
```

- ScrollBarDemo.java

Slider

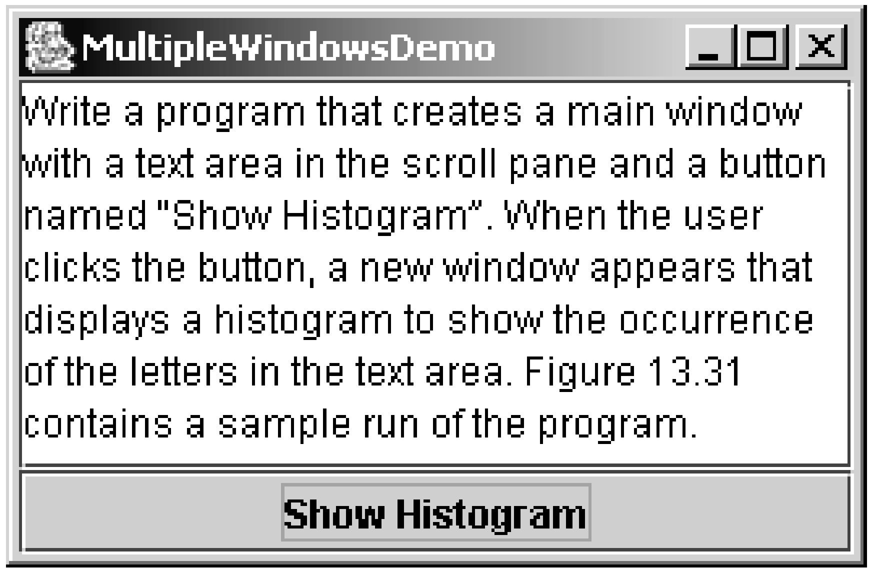




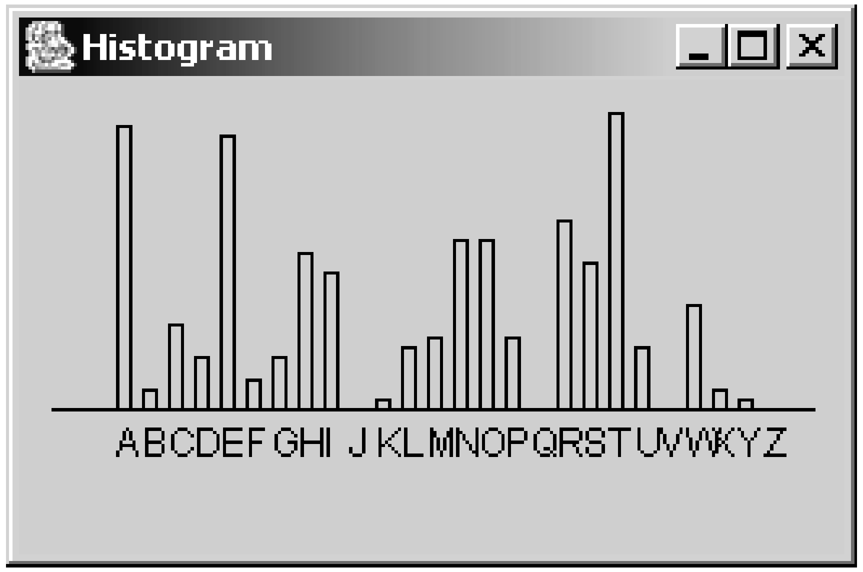
```
javax.swing.JSlider

-maximum: int
-minimum: int
-value: int
-orientation: int
-paintLabels: boolean
-paintTicks: boolean
-paintTrack: boolean
-majorTickSpacing: int
-minorTickSpacing: int
-inverted: boolean

+JSlider()
+JSlider(min: int, max: int)
+JSlider(min: int, max: int, value: int)
+JSlider(orientation: int)
+JSlider(orientation: int, min: int, max:
  int, value: int)
+addChangeListener(listener:
  ChangeListener) :void
```

- SliderDemo.java

Multiple Windows

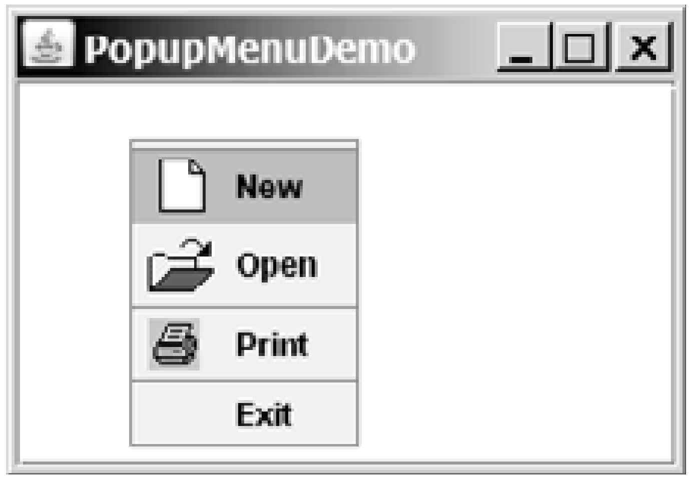


Write a program that creates a main window with a text area in the scroll pane and a button named "Show Histogram". When the user clicks the button, a new window appears that displays a histogram to show the occurrence of the letters in the text area. Figure 13.31 contains a sample run of the program.



- MultipleWindowsDemo.java
- Histogram.java

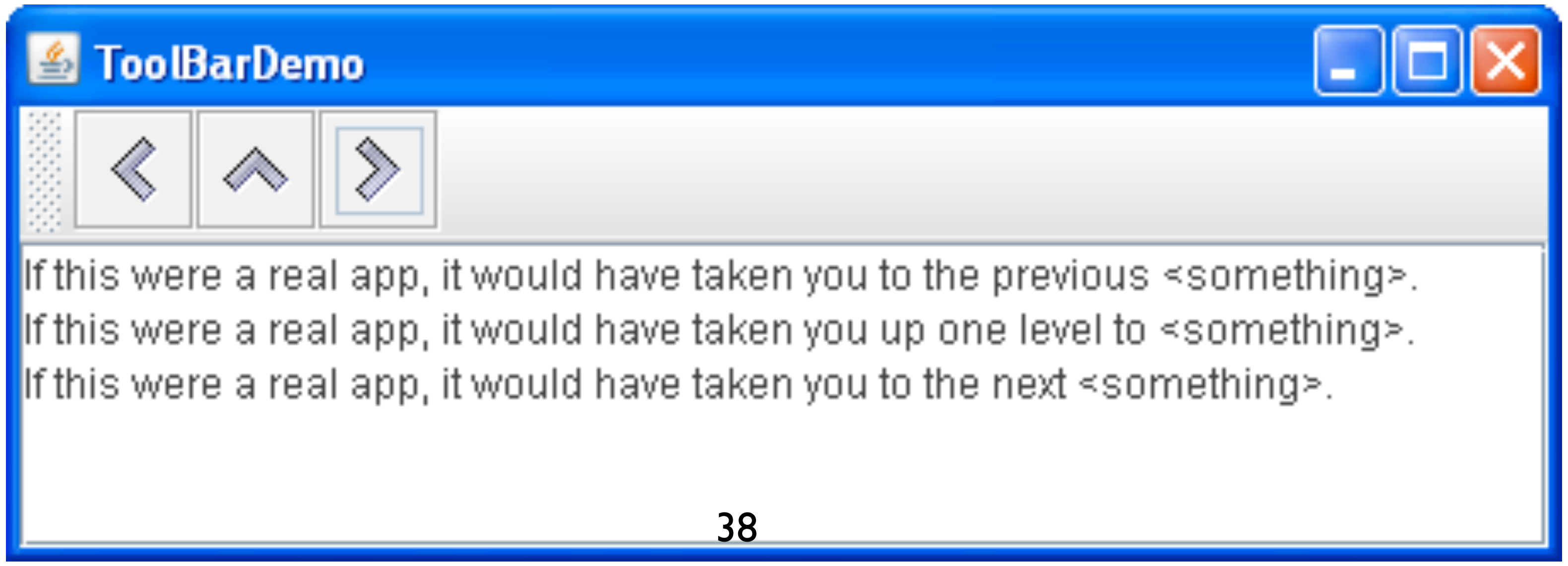
PopupMenu



- PopupMenuDemo.java

ToolBar

- A JToolBar is a container that groups several components — usually buttons with icons — into a row or column. Often, tool bars provide easy access to functionality that is also in menus. How to Use Actions describes how to provide the same functionality in menu items and tool bar buttons.



If this were a real app, it would have taken you to the previous <something>.
If this were a real app, it would have taken you up one level to <something>.
If this were a real app, it would have taken you to the next <something>.

To use JToolBar

- Create a JToolBar, add it to the JFrame
- Create and add JButtons and other components to the JToolBar
- Add ActionListeners to the JButtons

ToolBarDemo.java

Dialog

Dialogs

- Modal dialogs block all other interaction.
  - Forces a response from the user.
- Non-modal dialogs allow other interaction.
  - This is sometimes desirable.
  - May be difficult to avoid inconsistencies.

JDialog

- Every dialog is dependent on a frame. When that frame is destroyed, so are its dependent dialogs. When the frame is iconified, its dependent dialogs disappear from the screen. When the frame is deiconified, its dependent dialogs return to the screen. The Swing automatically provides this behavior.
- A dialog can be modal. When a modal dialog is visible, it blocks user input to all other windows in the program. The JDialogs that JOptionPane creates are modal. To create a non-modal dialog, you must use the JDialog class directly.

Code for JDialog

- To define a dialog:

```
public class DlgReport extends JDialog {
   public DlgReport(JFrame frame... ) {
         super(frame,"The Title",true);
```

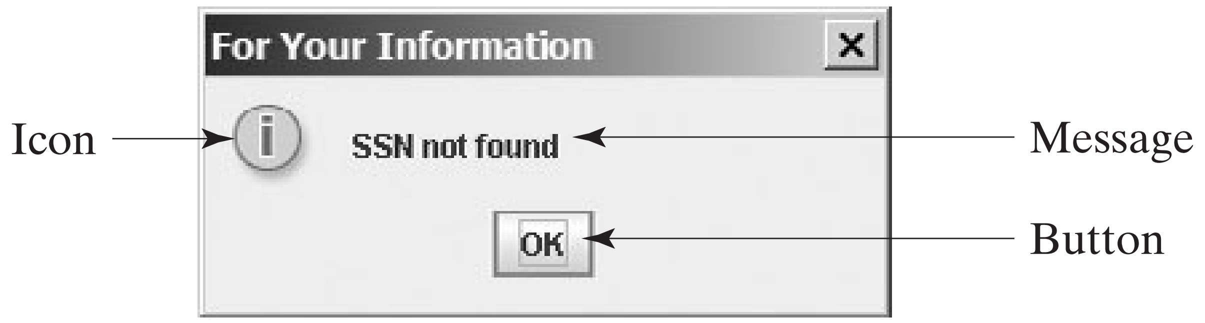
- To use that dialog:

```
DlgReport dlg = new DlgReport(theFrame,...);
dlg.pack();
dlg.setVisible(true);
```

Code for Dialogs

- Write your own code to describe the components in the dialog, and define the event handlers
- Write get/set functions to access the variables

JOptionPane standard dialogs



- Message dialog
  - Message text plus an OK button.
- Confirm dialog
  - Yes, No, Cancel options.
- Input dialog
  - Message text and an input field.
- Variations are possible.

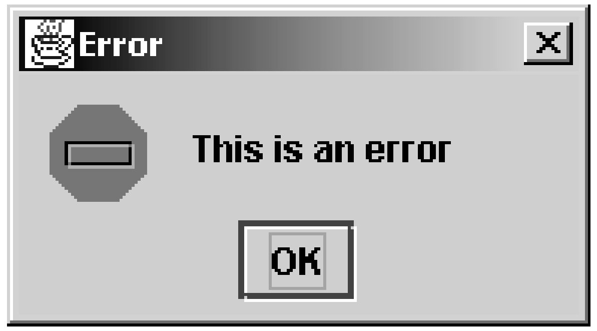
JOptionPane

- JOptionPane makes it easy to pop up a standard dialog box that prompts users for a value or informs them of something.

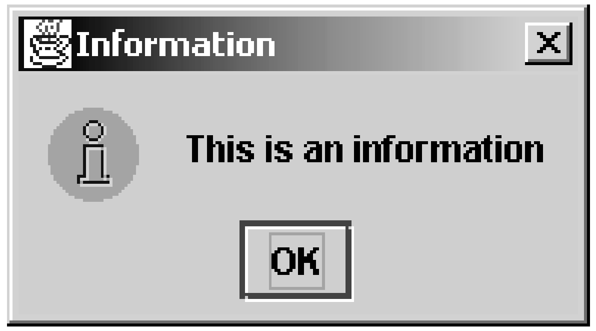
Method Name	Description
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened
showOptionDialog	The Grand Unification of the above three.

Message Dialogs

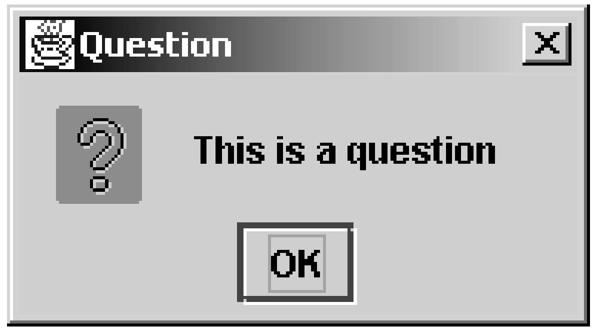
- JOptionPane.ERROR_MESSAGE
- JOptionPane.INFORMATION_MESSAGE
- JOptionPane.PLAIN_MESSAGE
- JOptionPane.WARNING_MESSAGE
- JOptionPane.QUESTION_MESSAGE



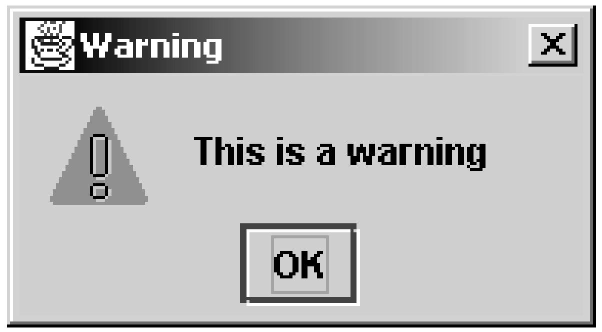
This is an error



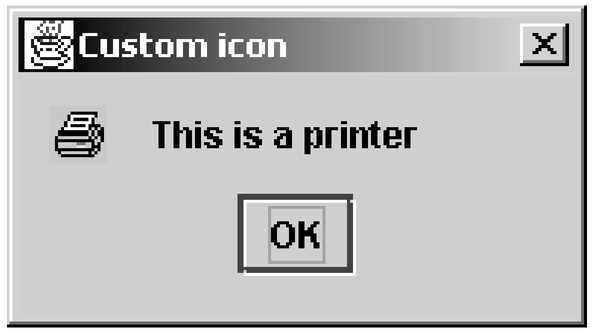
This is an information



This is a question



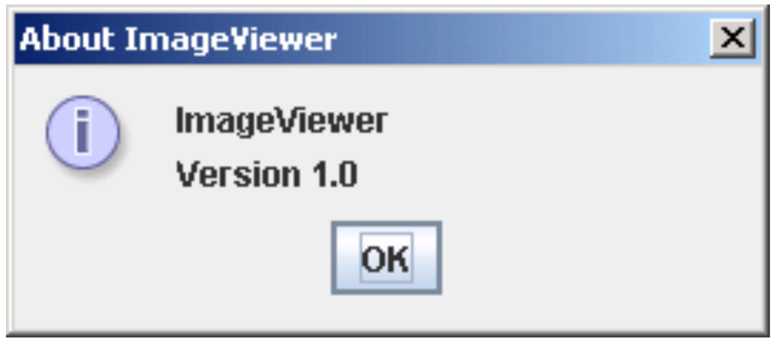
This is a warning



This is a printer

A message dialog

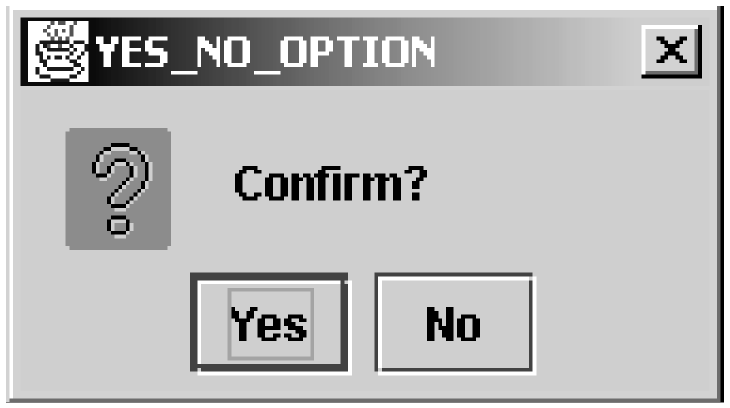
```
private void showAbout()
{
    JOptionPane.showMessageDialog(frame,
                "ImageViewer\n" + VERSION,
                "About ImageViewer",
                JOptionPane.INFORMATION_MESSAGE);
}
```

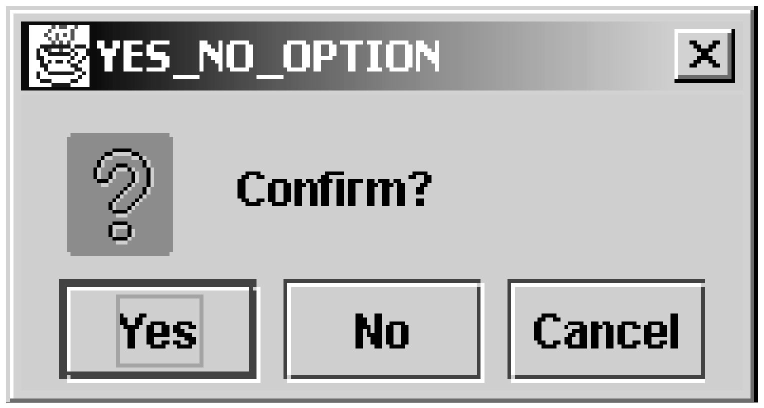

About ImageViewer

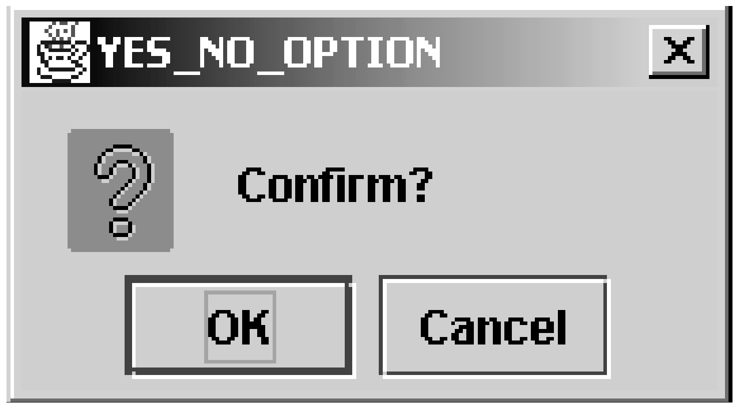
ImageViewer
Version 1.0

ConfirmDialogs

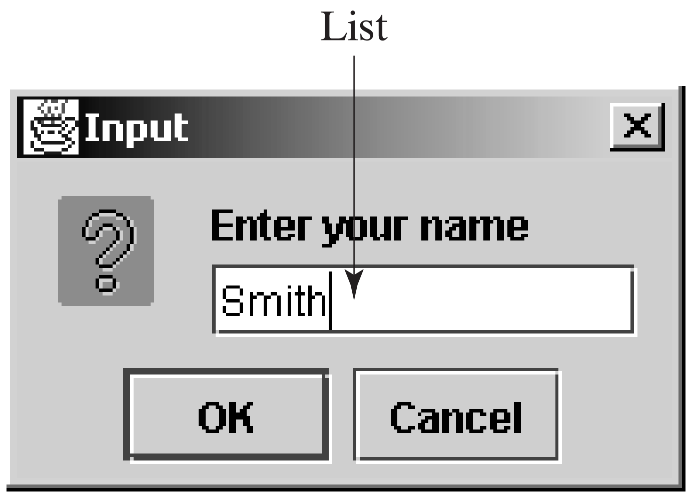
- JOptionPane.YES_NO_OPTION
- JOptionPane.YES_NO_CANCEL_OPTION
- JOptionPane.OK_CANCEL_OPTION







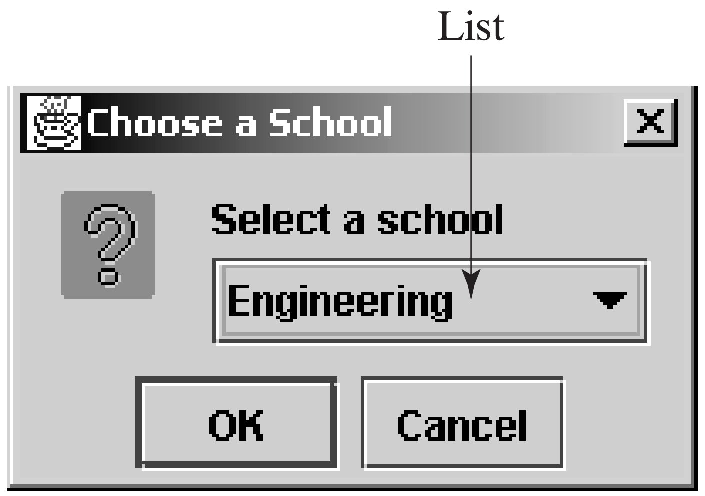
InputDialogs



Enter your name

Smith

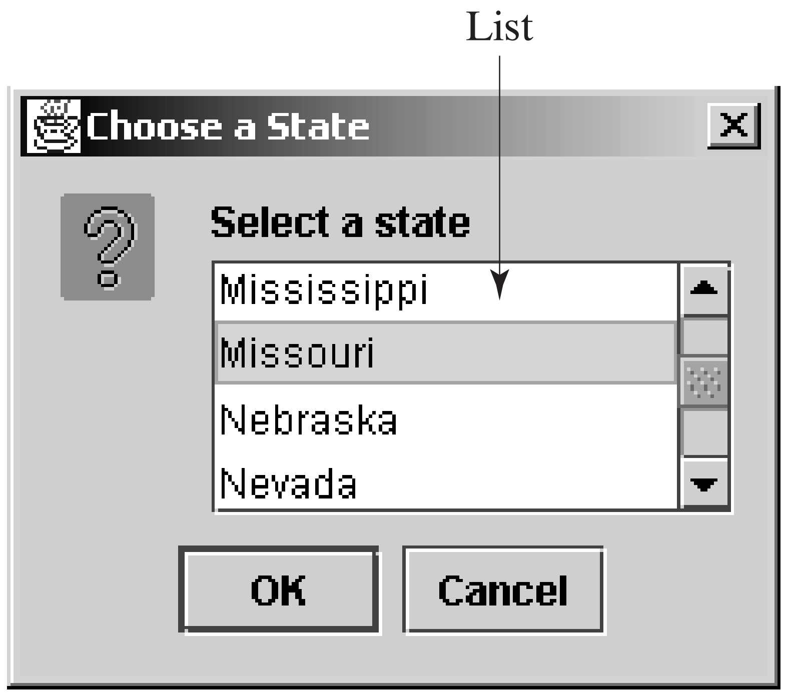
(a) Text field



Select a school

Engineering

(b) Combo box



Select a state

Mississippi
Missouri
Nebraska
Nevada

(c) List

Code for JOptionPane

```
JOptionPane.showMessageDialog(theFrame,"(C)2007 BA5AG");

String name = JOptionPane.showInputDialog(theFrame,"Input the name:","Search
Author",JOptionPane.QUESTION_MESSAGE);

if ( JOptionPane.showConfirmDialog( this, "delete?", "delete?",
JOptionPane.YES_NO_OPTION ) == JOptionPane.NO_OPTION )
```

JOptionPaneDemo.java

Create Your Dialog

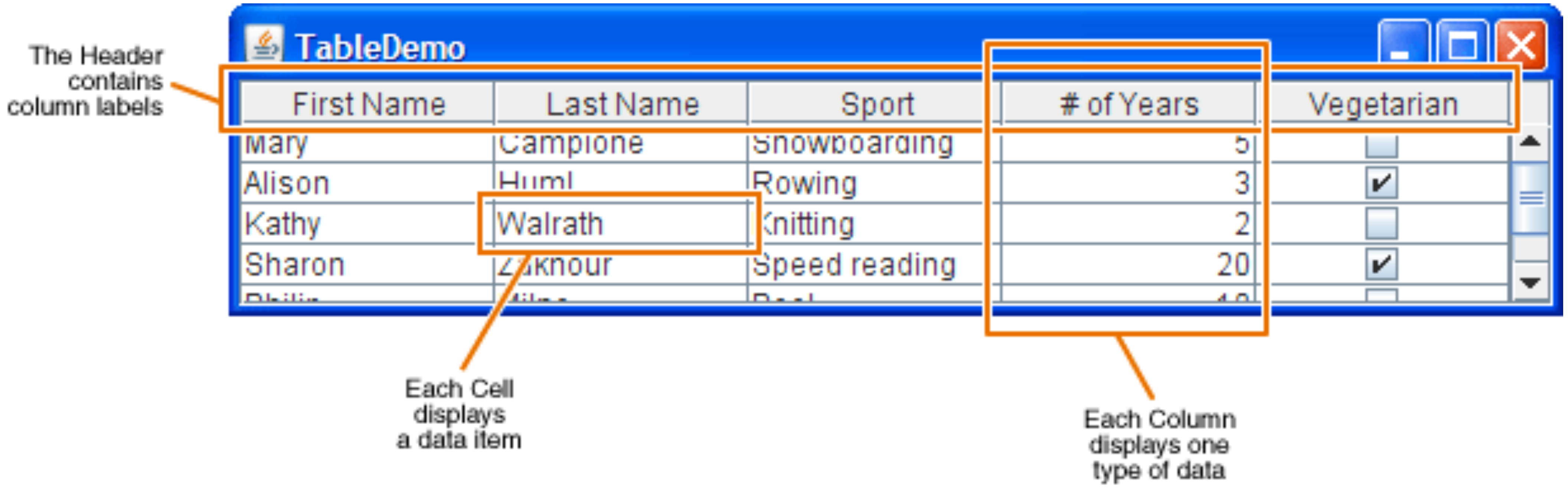
- extend class JDialog
- set modal
- set visible
- dispose

```
public ColorDialog(java.awt.Frame parent, boolean modal) {
  super(parent, modal);
```

ColorDialog.java

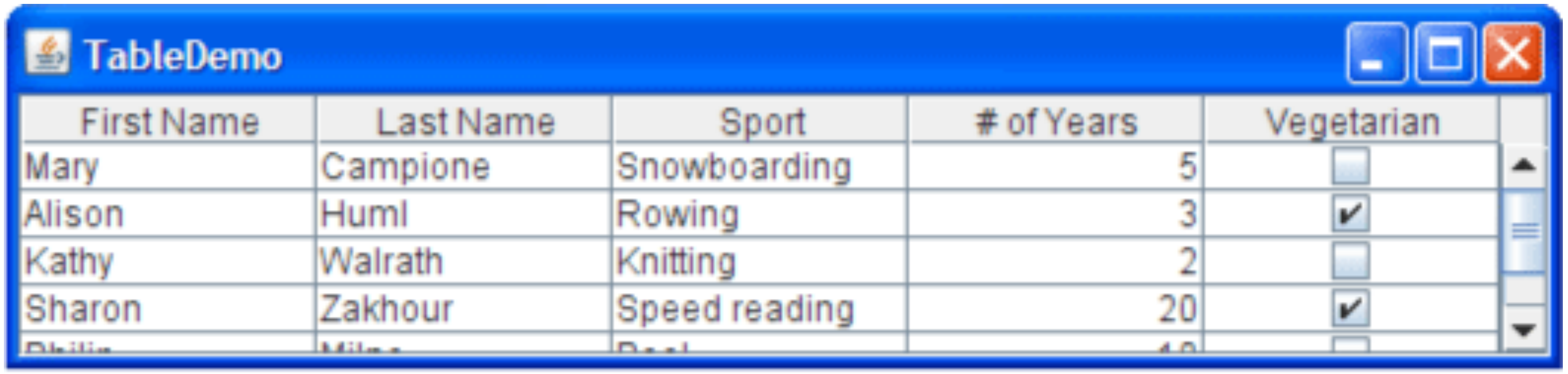
JTable

- With the JTable class you can display tables of data, optionally allowing the user to edit the data. JTable does not contain or cache data; it is simply a view of your data.


TableDemo
The Header contains column labels
Each Cell displays a data item
Each Column displays one type of data

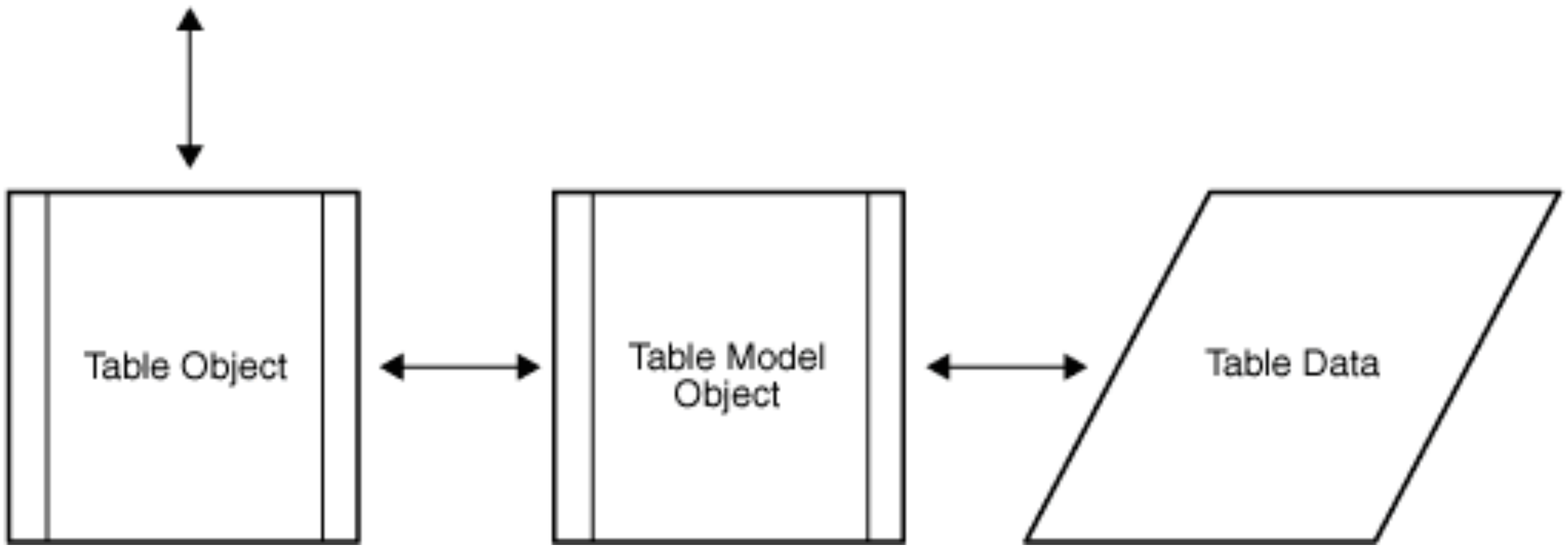
First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	☐
Alison	Huml	Rowing	3	✓
Kathy	Walrath	Knitting	2	☐
Sharon	Zakhnour	Speed reading	20	✓

TableModel



TableDemo

First Name	Last Name	Sport	# of Years	Vegetarian
Mary	Campione	Snowboarding	5	☐
Alison	Huml	Rowing	3	✓
Kathy	Walrath	Knitting	2	☐
Sharon	Zakhour	Speed reading	20	✓



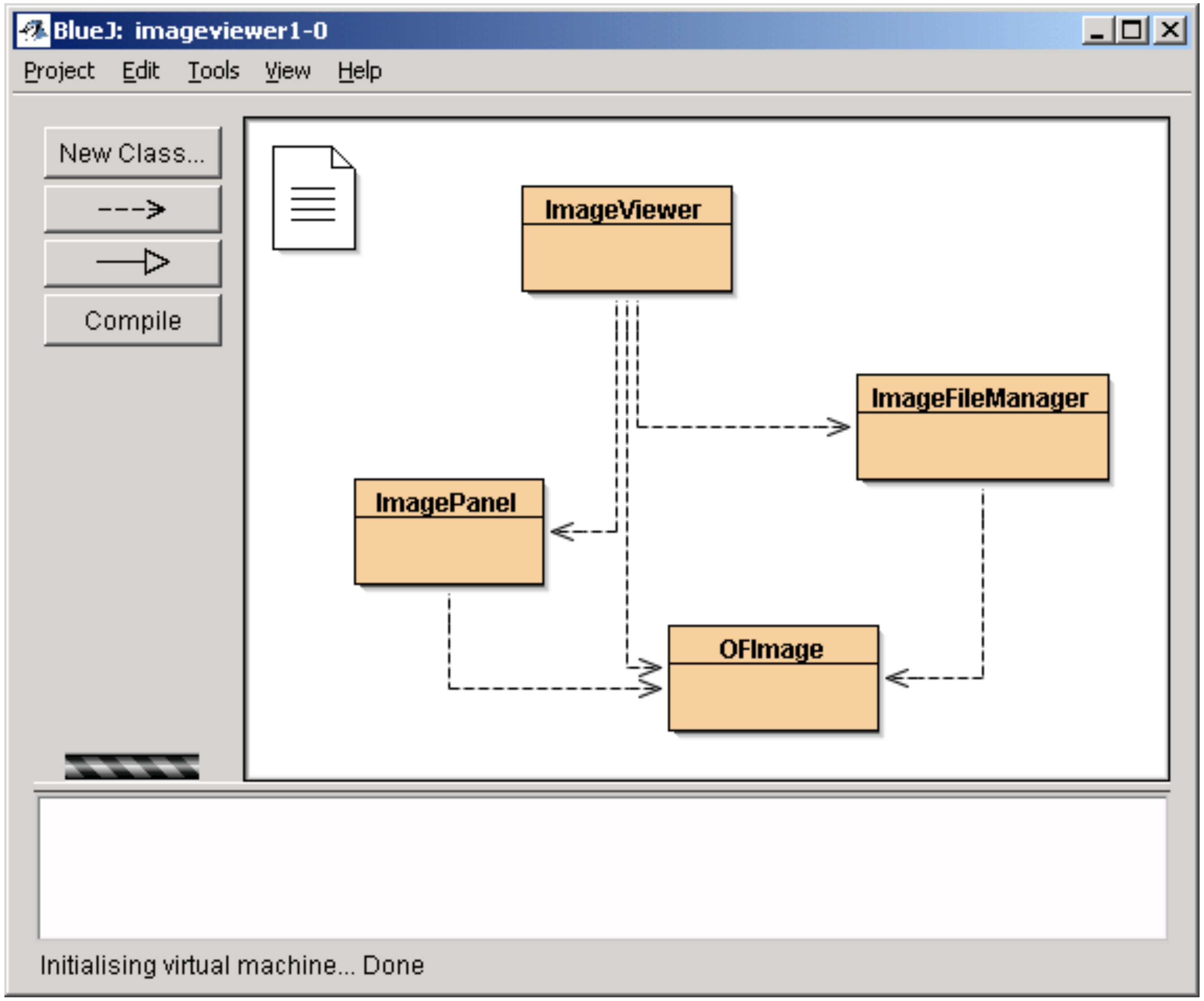
Code for JTable

```
JTable table = new JTable(new MyDataModel());
JScrollPane scrollPane = new JScrollPane(table);
add(scrollPane, BorderLayout.CENTER);
```

AbstractTableModel

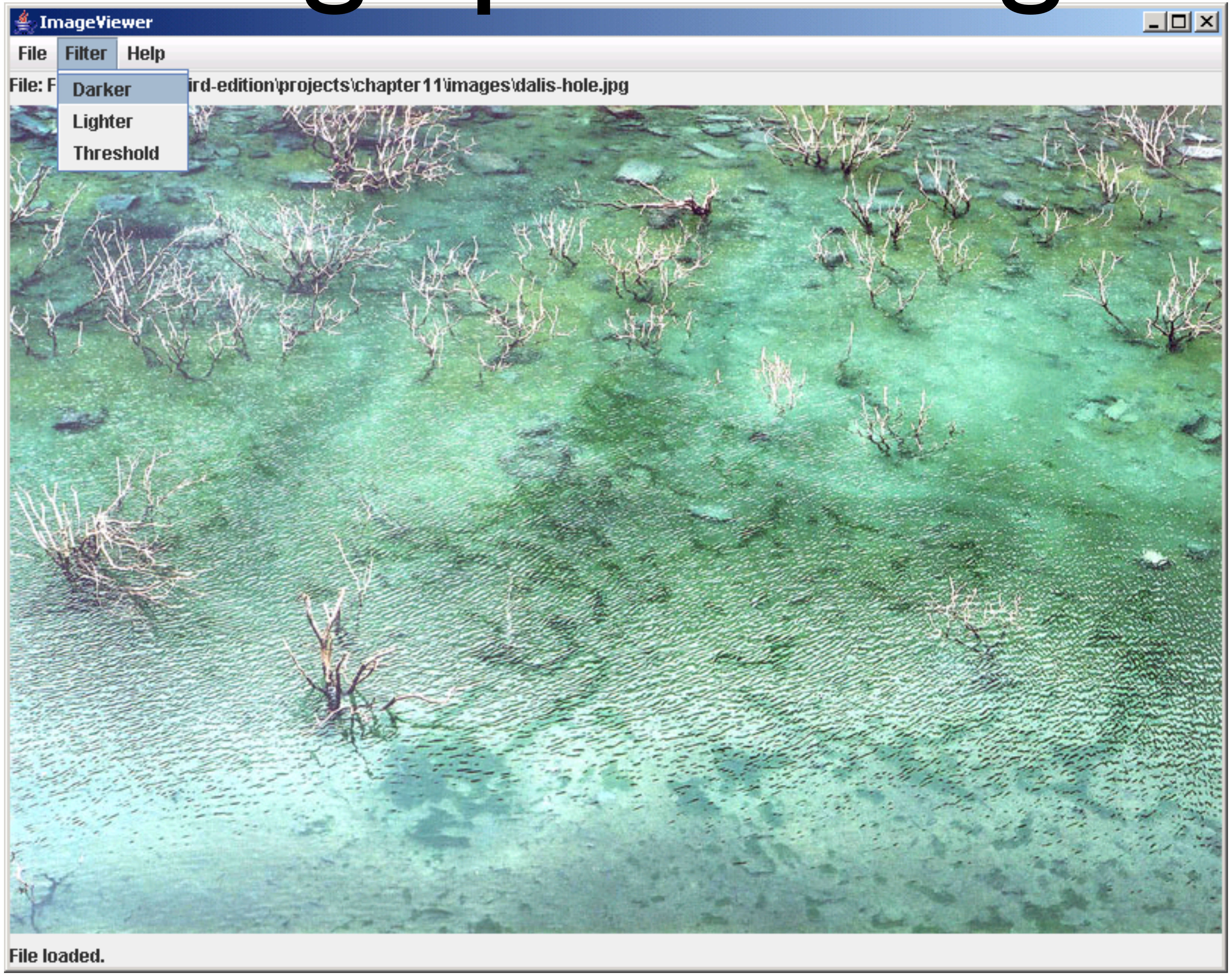
```
public int getColumnCount()

public int getRowCount()

public String getColumnName(int col)

public Object getValueAt(int row, int col)

public boolean isCellEditable(int row, int col)

public void setValueAt(Object value, int row, int col)
```

The imageviewer project



Initialising virtual machine... Done

Image processing



File: F ...ird-edition\projects\chapter11\images\dalis-hole.jpg

File loaded.

Class responsibilities

- `ImageViewer`
  - Sets up the GUI structure.
- `ImageFileManager`
  - Static methods for image file loading and saving.
- `ImagePanel`
  - Displays the image within the GUI.
- `OFImage`
  - Models a 2D image.

OFImage

- Our subclass of `BufferedImage`.
- Represents a 2D array of pixels.
- Important methods:
  - `getPixel`, `setPixel`
  - `getWidth`, `getHeight`
- Each pixel has a color.
  - We use `java.awt.Color`.

Adding an ImagePanel

```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void makeFrame()
    {
        Container contentPane = frame.getContentPane();
        imagePanel = new ImagePanel();
        contentPane.add(imagePanel);
    }

    …
}
```

Loading an image

```
public class ImageViewer
{
    private JFrame frame;
    private ImagePanel imagePanel;

    …

    private void openFile()
    {
        File selectedFile = …;
        OFImage image =
            ImageFileManager.loadImage(selectedFile);
        imagePanel.setImage(image);
        frame.pack();
    }

    …
}
```

Image filters

- Functions applied to the whole image.

```
int height = getHeight();
int width = getWidth();
for(int y = 0; y < height; y++) {
    for(int x = 0; x < width; x++) {
        Color pixel = getPixel(x, y);
        alter the pixel's color value;
        setPixel(x, y, pixel);
    }
}
```

Adding further filters

```
private void makeLighter()
{
    if(currentImage != null) {
        currentImage.lighter();
        frame.repaint();
        showStatus("Applied: lighter");
    }
    else {
        showStatus("No image loaded.");
    }
}
```

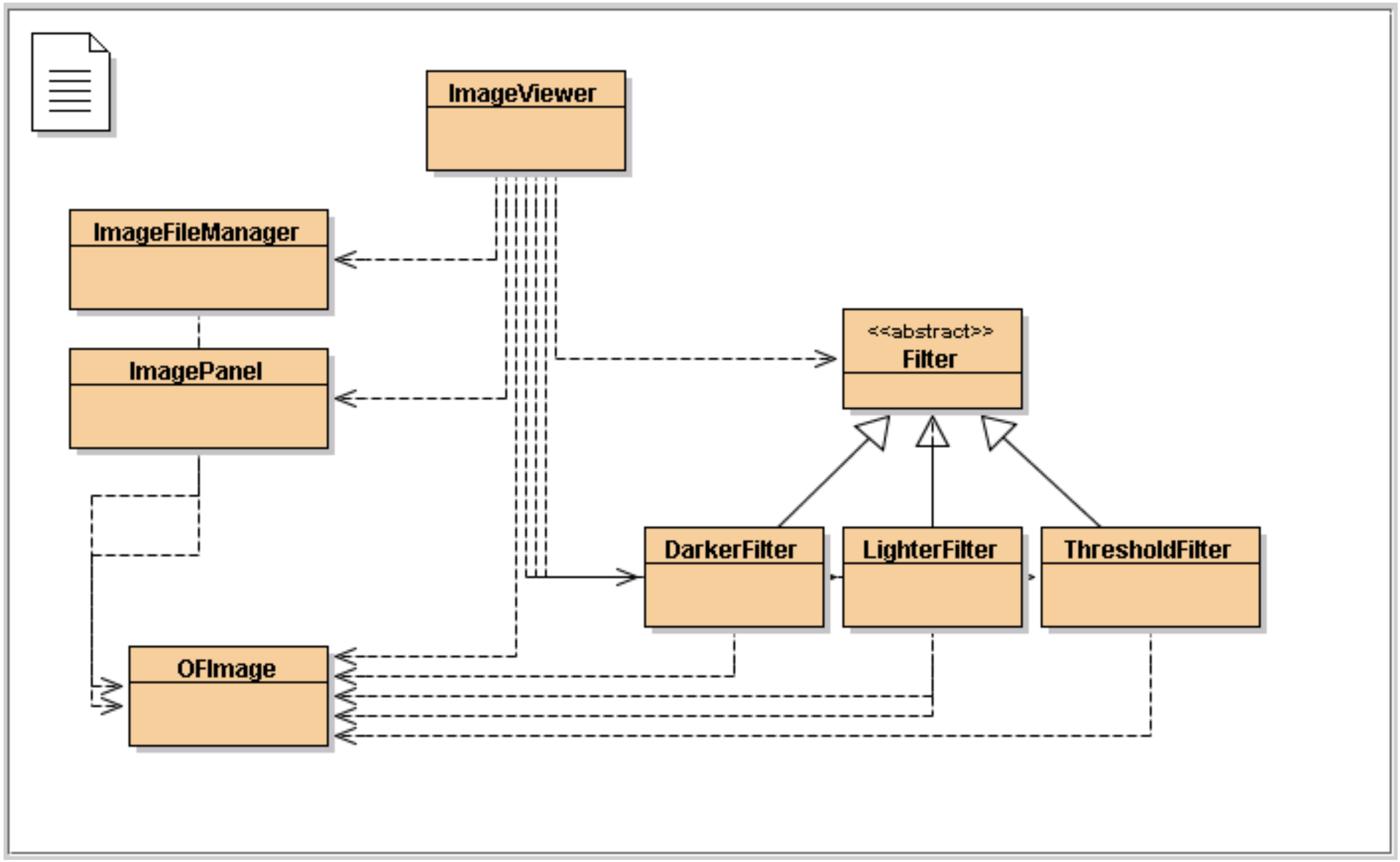
Code duplication?
Refactor!

```
private void threshold()
{
    if(currentImage != null) {
        currentImage.threshold();
        frame.repaint();
        showStatus("Applied: threshold");
    }
    else {
        showStatus("No image loaded.");
    }
}
```

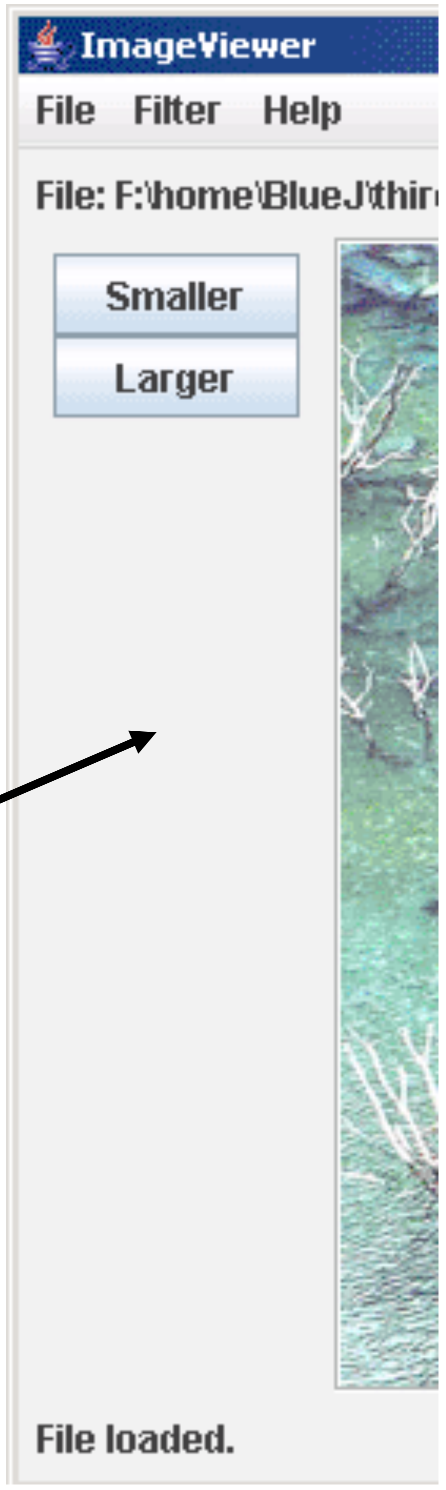
Adding further filters

- Define a `Filter` superclass (abstract).
- Create function-specific subclasses.
- Create a collection of subclass instances in `ImageViewer`.
- Define a generic `applyFilter` method.
- See *imageviewer2-0*.

imageviewer2-0



Buttons and nested layouts



A GridLayout inside a FlowLayout inside a BorderLayout.

Borders

- Used to add decoration around components.
- Defined in `javax.swing.border`
    - `BevelBorder`, `CompoundBorder`, `EmptyBorder`, `EtchedBorder`, `TitledBorder`.

Adding spacing

```
JPanel contentPane = (JPanel)frame.getContentPane();
contentPane.setBorder(new EmptyBorder(6, 6, 6, 6));

// Specify the layout manager with nice spacing
contentPane.setLayout(new BorderLayout(6, 6));

imagePanel = new ImagePanel();
imagePanel.setBorder(new EtchedBorder());
contentPane.add(imagePanel, BorderLayout.CENTER);
```