Bài 6:
GRAPHICAL USER INTERFACE

Nội dung

- Tổng quan về Graphical User Interface (GUI)
- Containment Hierarchy
- Layout Manager
- Swing components

Tổng quan GUI

- AWT: cung cấp các gói cơ bản để lập trình giao diện, được sử dụng trong các phiên bản trước jdk1.2 (java2)
- Swing: có nhiều chức năng hơn AWT, xây dựng chương trình dễ dàng mềm dẻo hơn. Được đưa vào từ jdk1.2
- Swt: được phát triển bởi IBM
- *Swing Components* thường có tên bắt đầu với 'J'
  - AWT có lớp *Panel*
  - Swing có lớp tương ứng là *JPanel*
- Có thể sử dụng plug-in để hỗ trợ:
  - Eclipse: WindowsBuilder Pro
  - Netbean: đã tích hợp

Tổng quan GUI (tt)

- Thư viện hỗ trợ lập trình giao diện trong Java

AWT (Abstract Windows Toolkits)

```
import java.awt.*;
```

Swing (Java Foundation Classes Package)

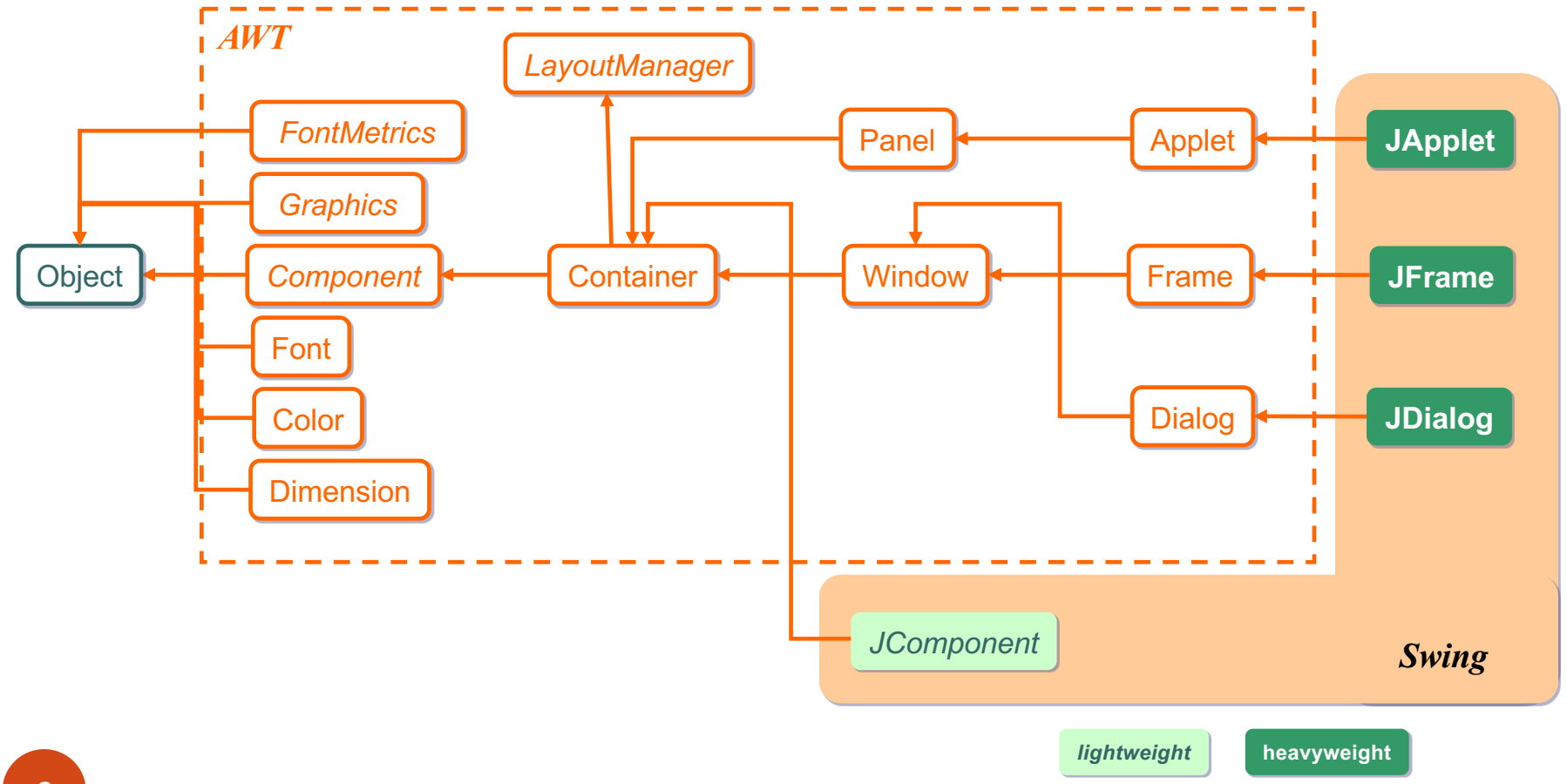
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

Tất cả Swing component có tên bắt đầu với J…

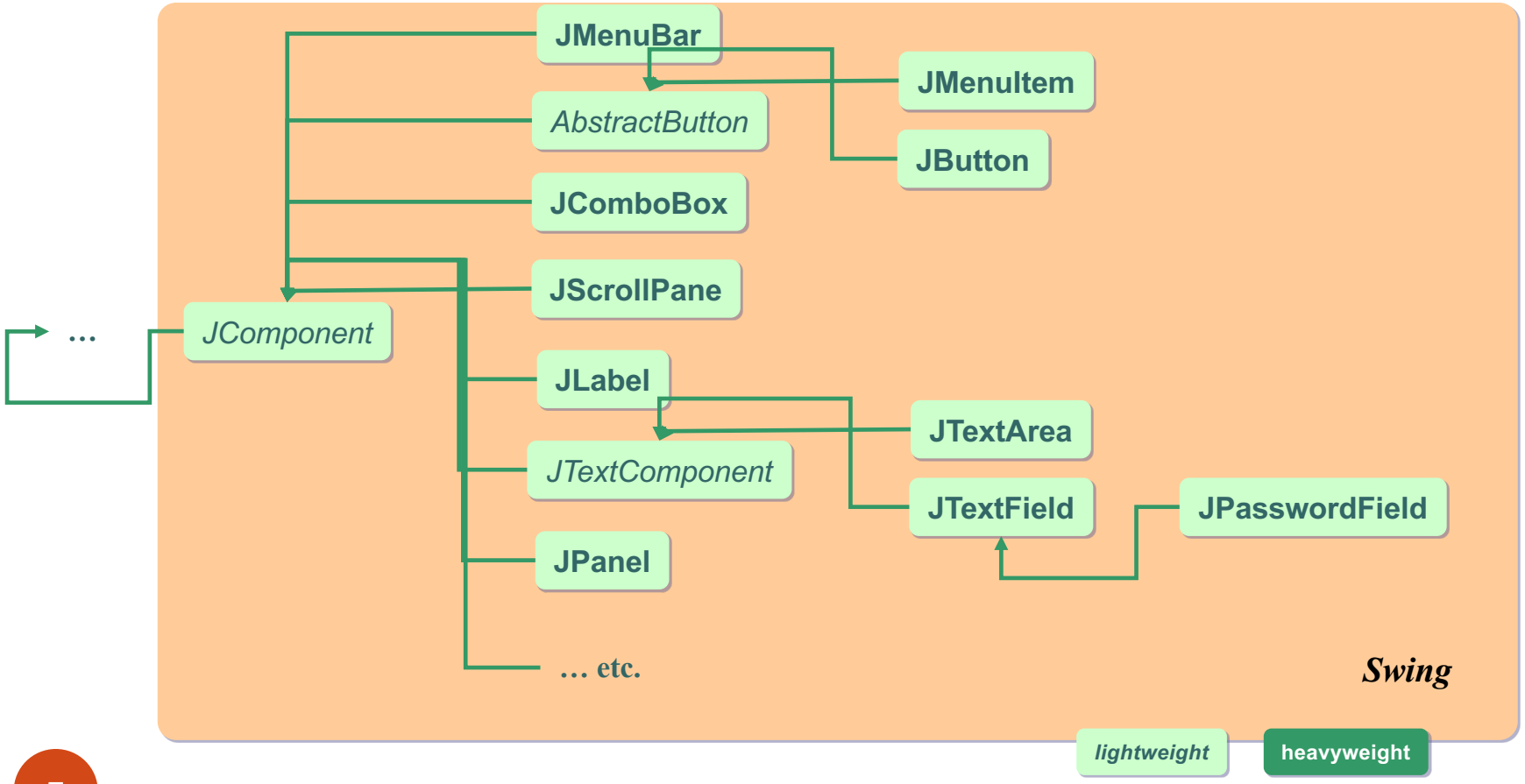
Swing API

- *Java Foundation Classes* (JFC) cung cấp một tập các chức năng giúp xây dựng các ứng dụng GUI.
- Gói Swing là 1 thành phần của JFC
- Sử dụng lệnh: `import javax.swing.*`
- Ưu điểm của Swing so với AWT:
  - Cung cấp thêm các đối tượng mới để xây dựng giao diện đồ họa
  - *look-and-feel:* tùy biến để các thành phần giao diện của Swing nhìn giống như các thành phần giao diện của HĐH
  - Hỗ trợ các thao tác sử dụng bàn phím thay chuột
  - Sử dụng tài nguyên hiệu quả hơn

Java GUI hierarchy



Java GUI hierarchy (tt)



Swing Components

- Swing Components có nhiều mức khác nhau:
  - *frame*, còn gọi main window (JFrame)
  - *panel*, hoặc *pane* (JPanel)
  - *button* (JButton)
  - *label* (JLabel)
- Các mức này qui định các Component trong 1 ứng dụng GUI kết hợp với nhau.
- Cách kết hợp này gọi là Swing *Containment Hierarchy*

Khái niệm container

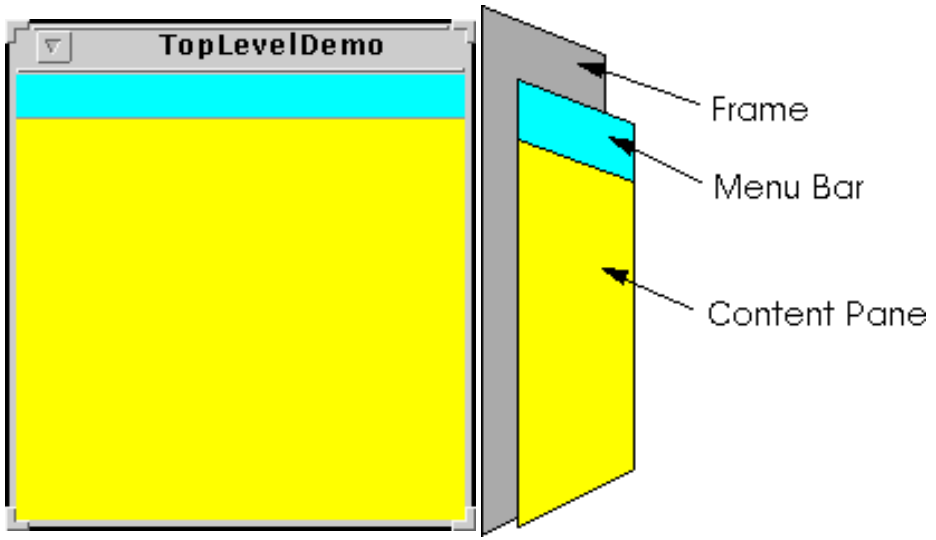
- Là thành phần mà có thể chứa các thành phần khác, có thể vẽ và tô màu.
  - Frame/JFrame, Panel/JPanel, Dialog/JDialog, ScrollPane/JScrollPane, …
- Gắn component vào khung chứa
  - containerObj.add(compObj);
- Lấy thông tin của component
  - objectName.get…( );
- Gán thông tin cho component
  - objectName.set…( );

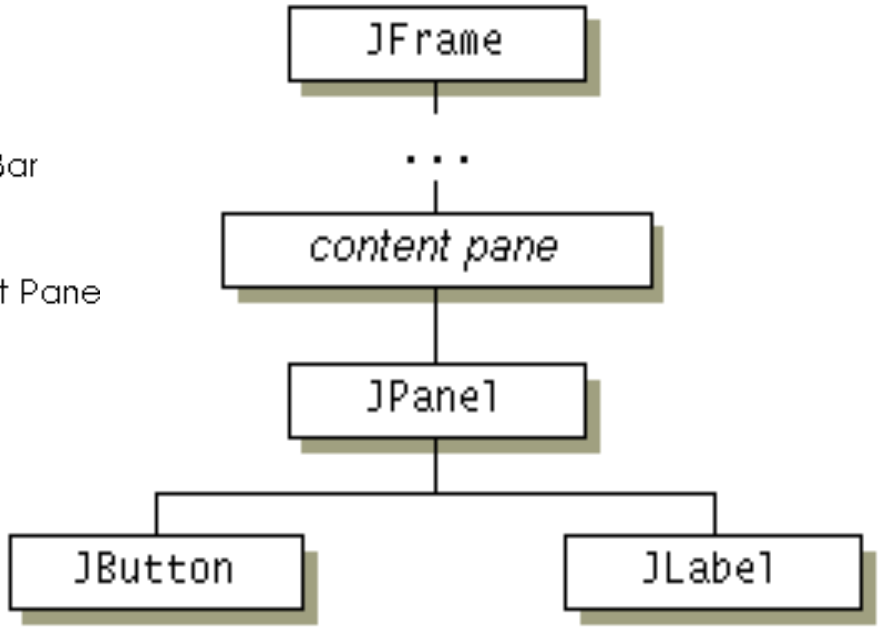
Nguyên tắc xây dựng GUI trong java

- Lựa chọn 1 container: Frame/JFrame, Window/JWindow, Dialog/JDialog, …
- Tạo các điều khiển: (buttons, text areas..)
- Đưa các điều khiển vào vùng chứa
- Sắp xếp các điều khiển(layout)
- Thêm các xử lý sự kiện (Listeners)

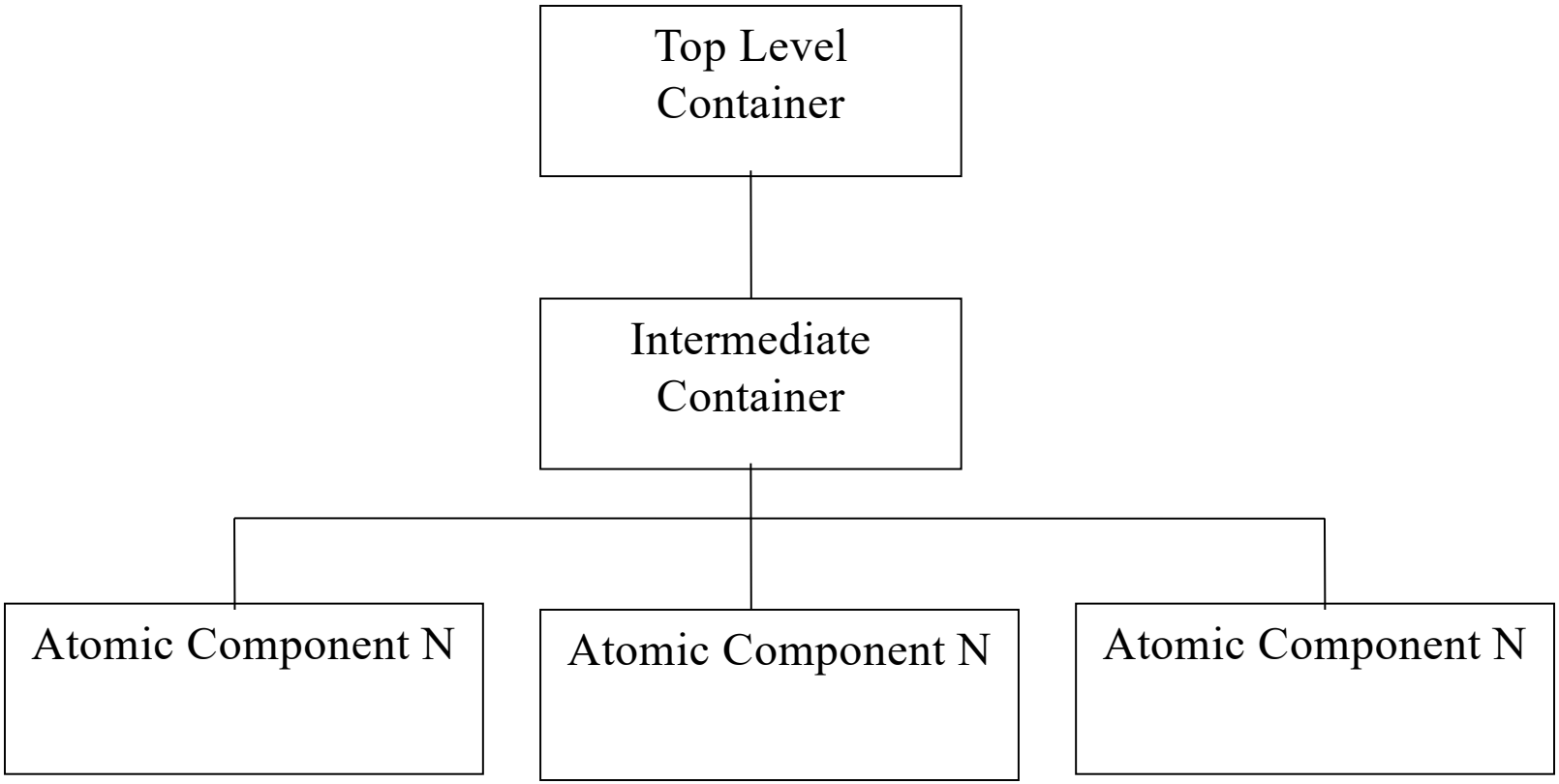
Containment Hierarchy

Frame: Top-level container: JFrame, JDialog, JApplet

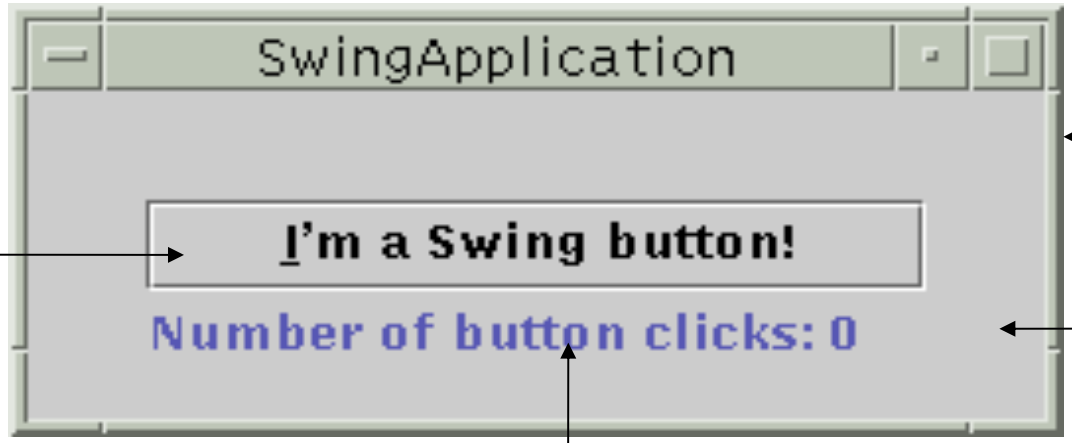




Containment Hierarchy (tt)



Containment Hierarchy (tt)



Top Level

(JFrame)

Atomic

(JButton)

Intermediate

(JPanel)

Atomic

(JLabel)

Containment Hierarchy (tt)

- **Top Level Containers**
  - Là nơi để hiển thị các Component khác
  - Ví dụ:
    - JFrame: sử dụng cho các cửa sổ chính của chương trình
    - JDialog: cửa sổ thông báo
    - JApplet: sử dụng trên trình duyệt

Containment Hierarchy (tt)

- **Top Level Containers: JFrame**
  - Là cửa sổ chính, dùng để chứa các thành phần giao diện khác. Đóng vai trò là một container
  - Hàm khởi tạo
    - JFrame()
    - JFrame(String title)
  - Các thành phần đồ họa được đưa vào content pane, không đưa trực tiếp vào JFrame. Ví dụ:

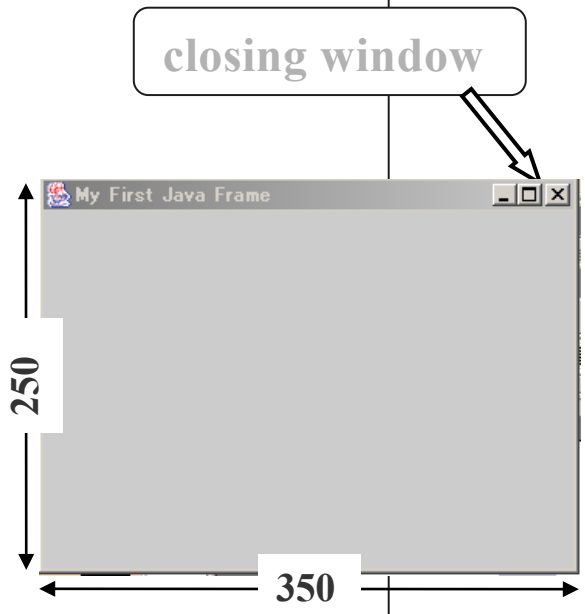
```
frame.getContentPane().add(b);
```

  - JFrame cung cấp 2 phương thức:
    - getContentPane() trả lại một đối tượng ContentPane thuộc lớp Container
    - setContentPane(JPanel): thiết lập nội dung cho Content Pane

Ví Dụ JFrame

AppFrame1.java

```
import javax.swing.*;
class Frame1 extends JFrame {
   /* Construction of the frame */
   public Frame1() {
      this.setSize(350, 250);
      this.setTitle("My First Java Frame");
  }
}
public class AppFrame1 {
   public static void main(String[ ] args) {
      Frame1 frame = new Frame1();
      frame.setVisible(true);
   }
}
```



Containment Hierarchy (tt)

- **Intermediate Containers**
  - Dùng để xác định vị trí của các Atomic Components
  - Hoặc cung cấp cơ chế để tương tác với khung chứa.
  - Ví dụ:
    - JPanel
    - JScrollPane
    - JTabbedPane

Containment Hierarchy (tt)

- **Intermediate Containers: JPanel**
  - Là container trung gian để chứa các Atomic Components
  - Thường dùng để chia các component trong ứng dụng
  - Hàm khởi tạo:
    - JPanel();
    - JPanel(LayoutManager lm);

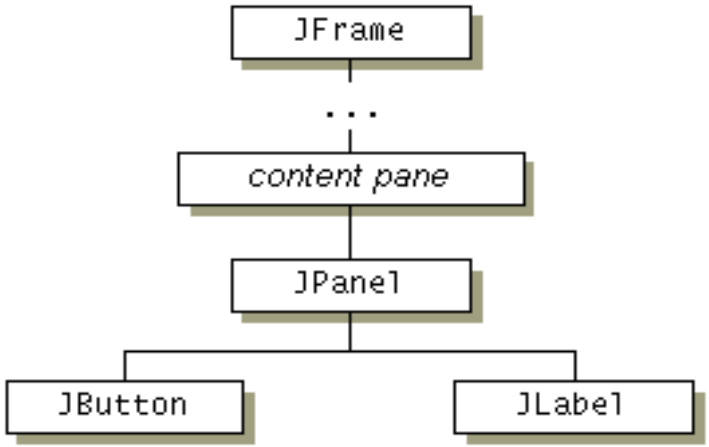
Containment Hierarchy (tt)

- **Atomic Components**
  - Không chứa các Components khác
  - Dùng để cung cấp thông tin đến người sử dụng
  - Hoặc lấy thông tin từ người sử dụng
  - Ví dụ:
    - JButton
    - JLabel
    - JComboBox
    - JTextField
    - JTable

Containment Hierarchy (tt)

- Mọi Top Level Container có 1 intermediate Container gọi là *Content Pane*
- Các atomic component phải gắn vào content pane
  - Ví dụ

```
JFrame frame = new JFrame(…);
JPanel pane = new JPanel();
frame.getContentPane().add(pane);
```



Layout Manager

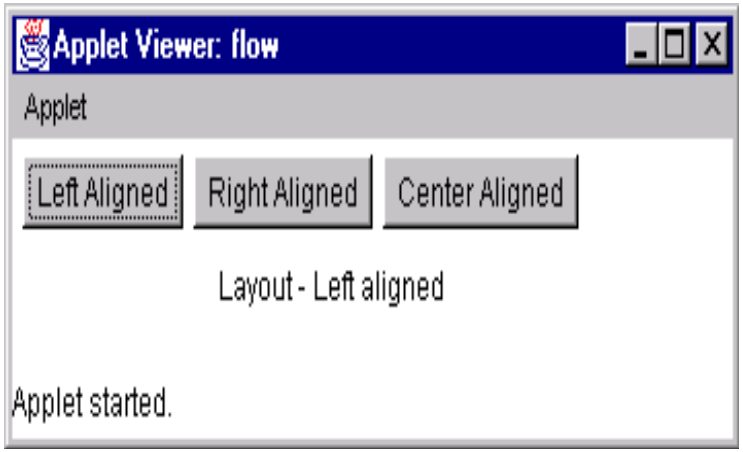
- **Khái niệm:** được dùng điều khiển cách trình bày vật lý của các phần tử GUI như là button, textbox, option button
- Các loại layout khác nhau:
  - Flow Layout
  - Border Layout
  - Grid Layout
  - GridBag Layout
  - Null Layout
  - …
- Trình quản lý layout được thiết lập bằng cách gọi phương thức '**setLayout( )**'

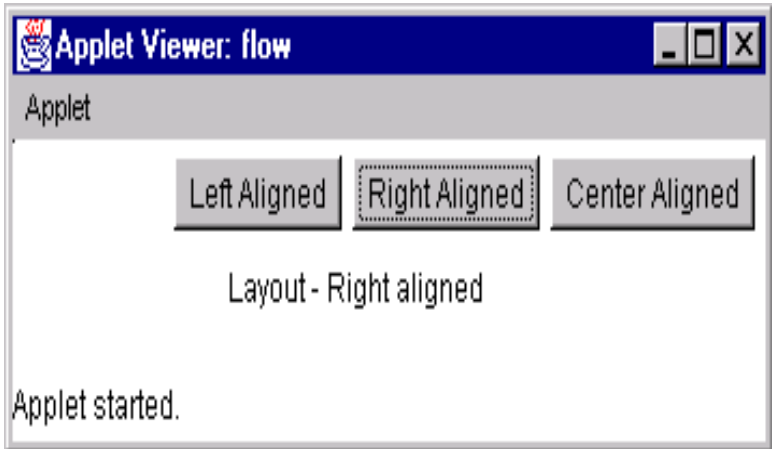
Layout Manager: FlowLayout

- Là trình quản lý layout mặc định cho các panel
- Với FlowLayout các thành phần sẽ được xắp xếp từ góc trái trên đến góc phải dưới của màn hình.
- Các hàm khởi tạo - constructor:
  - public FlowLayout();
  - public FlowLayout(int align); // Canh lề bên phải
  - public FlowLayout(int align, int hgap, int vgap);

- **align:** có các giá trị FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT, FlowLayout.LEADING, hoặc FlowLayout.TRAILING
- **hgap, vgap:** khoảng trống giữa các thành phần

FlowLayout (tt)





Flow Layout – Left and Right Aligned

Layout Manager: BorderLayout

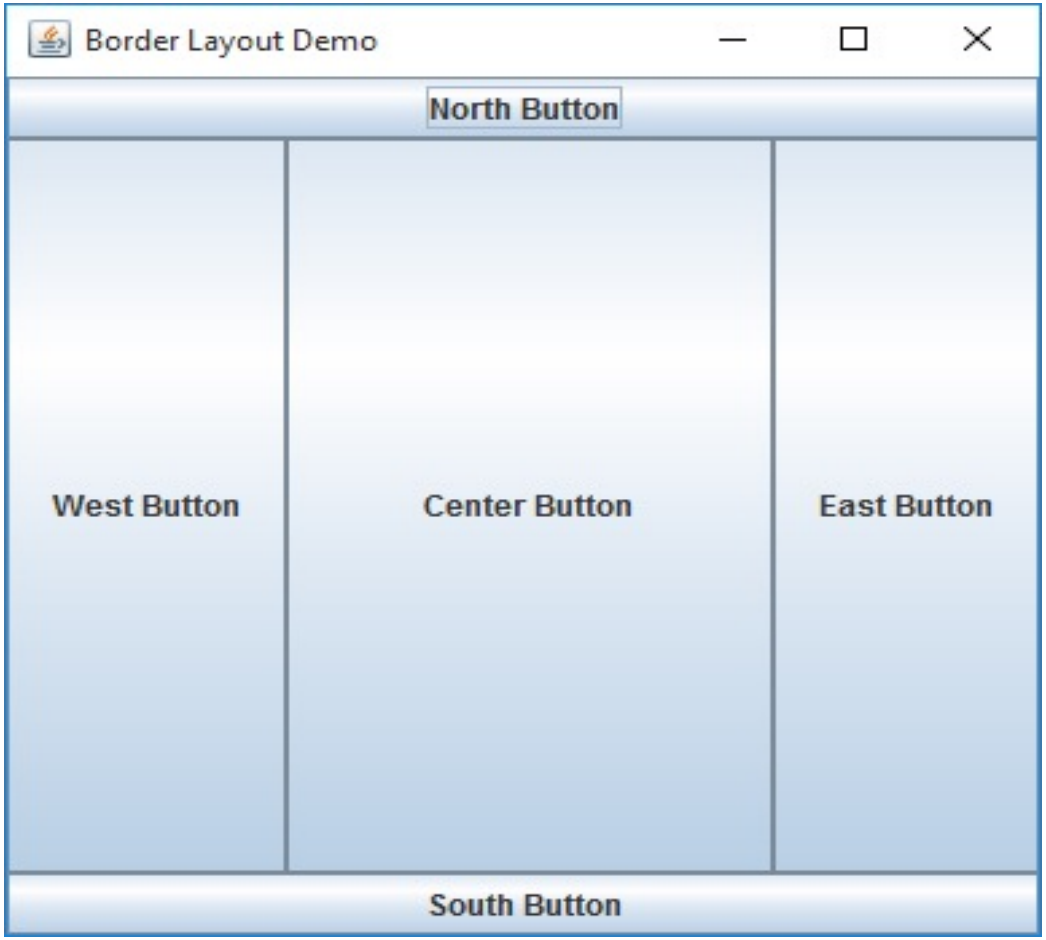
- Là trình quản lý layout mặc định cho Window, Frame và Dialog
- Trình quản lý này có thể xắp xếp đến 5 thành phần trong container
- Các thành phần có thể được đặt vào 5 hướng NORTH, EAST, SOUTH, WEST và CENTER của container
- Ví dụ: Để thêm một thành phần vào vùng North của container

```
JButton b1= new JButton("North Button");
setLayout(new BorderLayout( ));
add(b1, BorderLayout.NORTH);
```

BorderLayout (tt)


```
public void showBorderLayout(){
    JFrame frame = new JFrame("Border Layout Demo");
    frame.setSize(400,400);
    JPanel panel = new JPanel(new BorderLayout());
    panel.add(new JButton("North Button"),BorderLayout.NORTH);
    panel.add(new JButton("South Button"),BorderLayout.SOUTH);
    panel.add(new JButton("West Button"),BorderLayout.WEST);
    panel.add(new JButton("East Button"),BorderLayout.EAST);
    panel.add(new JButton("Center Button"),BorderLayout.CENTER);
    //frame.getContentPane().add(panel);
    frame.add(panel);
    frame.setVisible(true);
```

BorderLayout (tt)



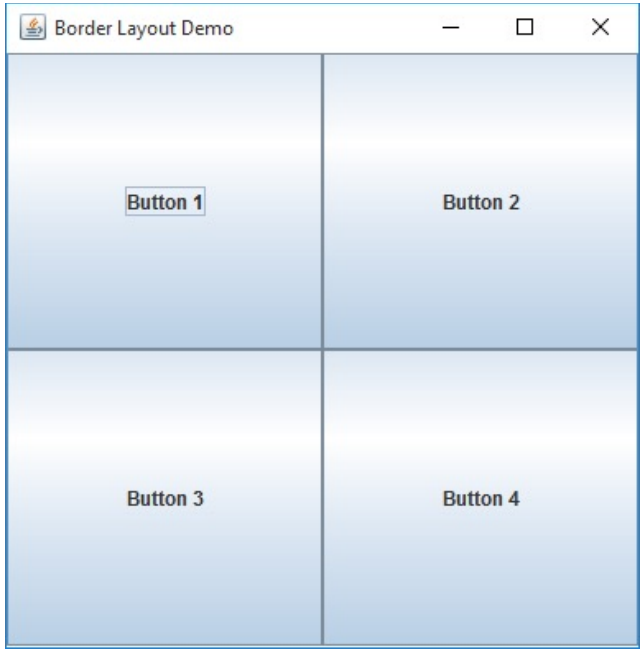
Layout Manager: GridLayout

- Hỗ trợ việc chia container thành một lưới
- Các thành phần được bố trí trong các dòng và cột
- Một ô lưới nên chứa ít nhất một thành phần
- Kiểu layout này được sử dụng khi tất cả các thành phần có cùng kích thước

```
GridLayout layout = new GridLayout(no. of rows, no. of columns);
containerObj.setLayout(layout);
```

GridLayout

```
public void showGridLayout(){
        JFrame frame = new JFrame("Border Layout Demo");
        frame.setSize(400,400);
        JPanel panel = new JPanel();
        GridLayout gridLayout = new GridLayout(2,2);
        panel.setLayout(gridLayout);
        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        panel.add(new JButton("Button 3"));
        panel.add(new JButton("Button 4"));
        //frame.getContentPane().add(panel);
        frame.add(panel);
        frame.setVisible(true);
    }
```



Layout Manager: GridBagLayout

- Bố trí các thành phần một cách chính xác
- Các thành phần không cần có cùng kích thước
- Các thành phần được xắp xếp trong một lưới chứa các dòng và các cột
- Thứ tự đặt các thành phần không tuân theo hướng từ trái-sang-phải và trên-xuống-dưới
- Hàm constructor

```
GridBagLayout gb = new GridBagLayout( );
```

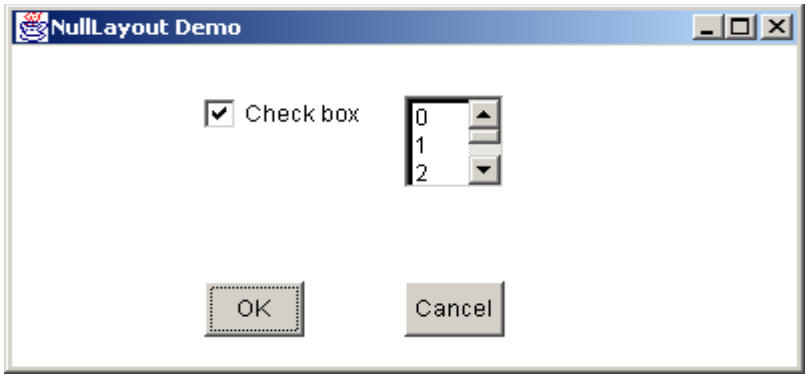
- Lớp GridBagLayoutConstraints lưu trữ tất cả các thông tin mà lớp GridLayout yêu cầu: Vị trí và kích thuớc mỗi thành phần

GridBagLayout

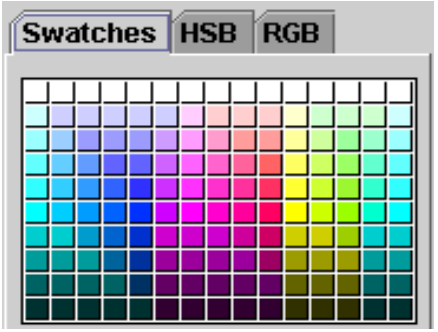
Layout Manager: NullLayout

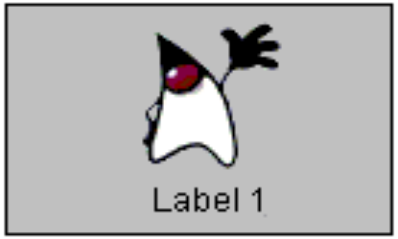
- Tự do trong việc định vị trí và kích thước của các components
- Giải pháp này dùng để xác định hoàn toàn kích thước và và vị trí của component.

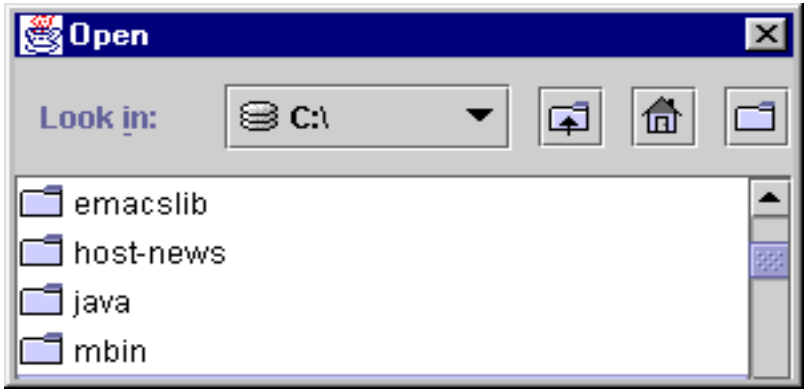
```
Frame fr = new Frame("NullLayout Demo");
fr.setLayout(null);
```

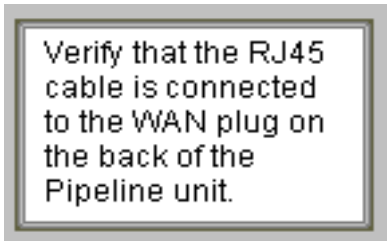


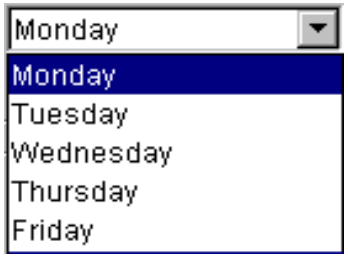
Swing Components











Sử Dụng Swing Component

javax.swing

Class JComponent

java.lang.Object
  └ java.awt.Component
    └ java.awt.Container
      └ **javax.swing.JComponent**

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable

Direct Known Subclasses:

AbstractButton, BasicInternalFrameTitlePane, Box, Box.Filler, JColorChooser, JComboBox, JFileChooser, JInternalFrame, JInternalFrame.JDesktopIcon, JLabel, JLayeredPane, JList, JMenuBar, JOptionPane, JPanel, JPopupMenu, JProgressBar, JRootPane, JScrollBar, JScrollPane, JSeparator, JSlider, JSpinner, JSplitPane, JTabbedPane, JTable, JTableHeader, JTextComponent, JToolBar, JToolTip, JTree, JViewport

Sử dụng Swing Components

- Có các phương thức add, set, get
- Phương thức dùng để gắn components:
  - objectName.add(…);
- Phương thức dùng để lấy thuộc tính:
  - objectName.getxxx( );
- Phương thức dùng để gán thuộc tính:
  - objectName.setxxx( );

JLabel

- *Label* dùng để hiển thị một chuỗi văn bản, hình ảnh nhằm mô tả thêm thông tin cho các đối tượng khác.
- Các constructor của JLabel:

```
JLabel()
JLabel(String text)
JLabel(String text,int hAlignment)
JLabel(Icon icon)
JLabel(Icon icon, int hAlignment)
JLabel(String text,Icon icon,int hAlignment)
```

hAlignment: có các giá trị LEFT, CENTER, RIGHT,LEADING, hoặc TRAILING

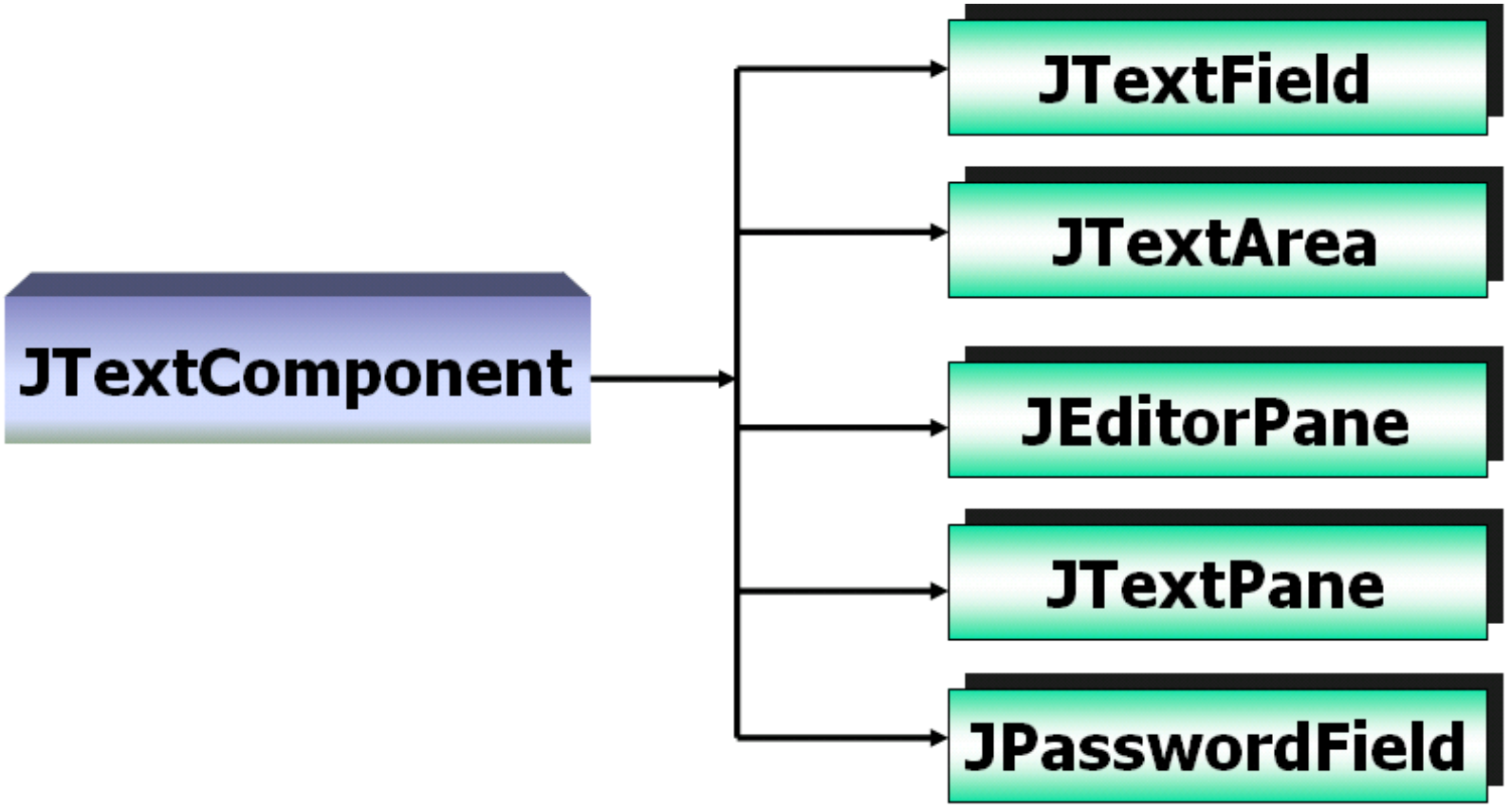
Các thuộc tính JLabel

- text
- icon
- horizontalAlignment
- verticalAlignment

Ví Dụ

JTextComponent

Đây là lớp cha của tất cả các lớp hiển thị văn bản trong Swing



JTextField

- *Textfield* là ô nhập dữ liệu dạng văn bản trên 1 dòng.
- Các constructor của JTextField:

```
JTextField()
```

- Tạo một text field trống có số cột xác định.

```
JTextField(int columns)
```

- Tạo một text field với văn bản có sẵn.

```
JTextField(String text)
```

- Tạo một text field với văn bản có sẵn và số cột xác định.

```
JTextField(String text, int columns)
```

Các thuộc tính JTextField

- `text`
- `horizontalAlignment`
- `editable`
- `columns`

Các phương thức JTextField

- `getText()`

  Trả về chuỗi ký tự trong text field.
- `setText(String text)`

  Đặt chuỗi ký tự trong text field.
- `setEditable(boolean editable)`

  Cho phép hoặc vô hiệu hóa soạn thảo trong text field. Mặc định *editable* là true.
- `setColumns(int)`

  Thiết lập số cột trong textfield. Chiều dài của textfield có thể thay đổi.

JTextArea

- TextArea là khung cho phép người sử dụng nhập vào nhiều dòng văn bản.
- Các constructor của JTextArea:

```
JTextArea()

JTextArea(String s)

JTextArea(int rows, int columns)

JTextArea(String s, int rows, int columns)
```

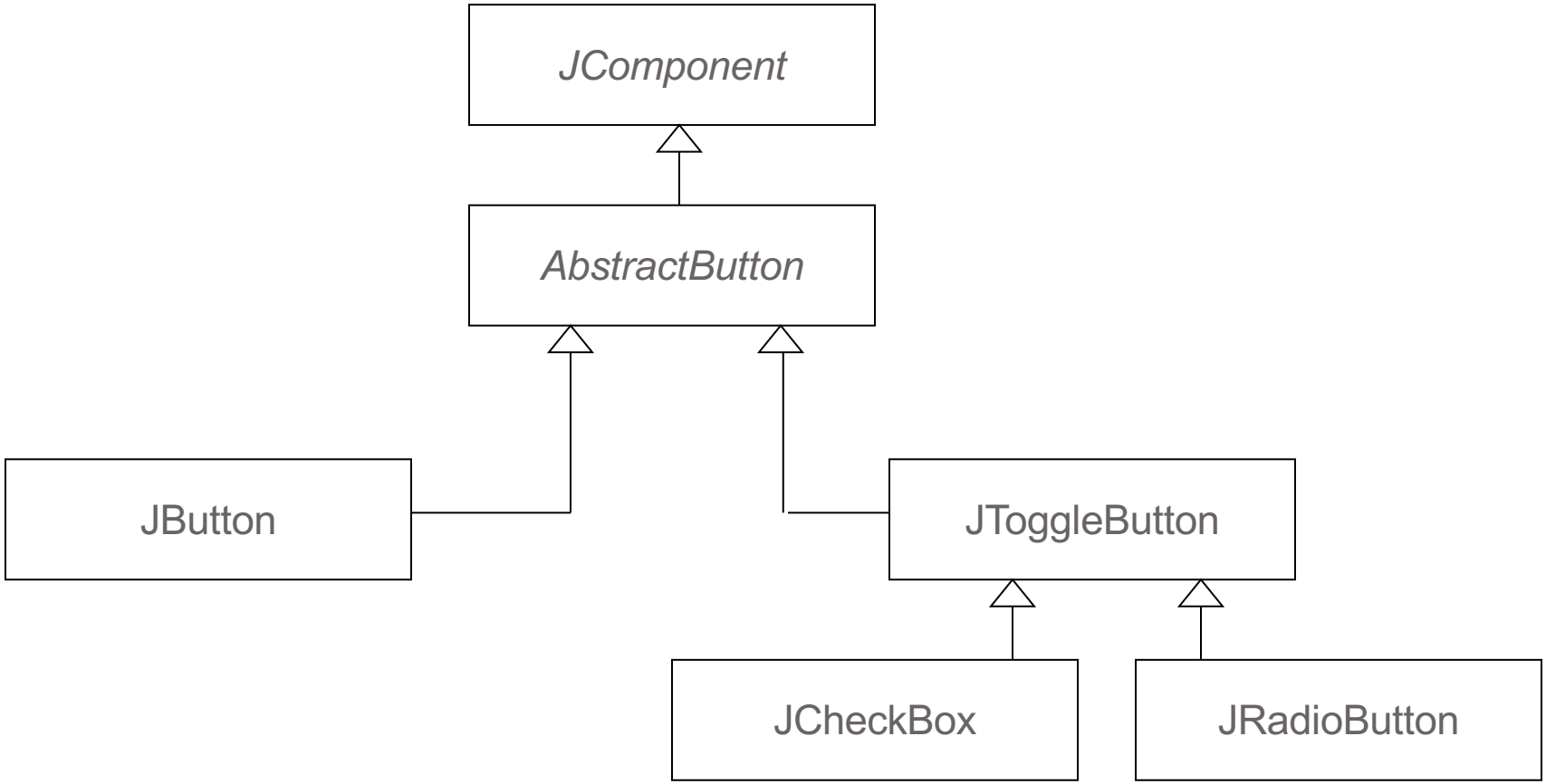
Các thuộc tính JTextArea

- text
- editable
- columns
- lineWrap
- wrapStyleWord
- rows
- lineCount
- tabSize

Các phương thức JTextArea

- `getColumns()`
  Trả về số cột trong textarea.
- `getLineCount()`
  Trả về số dòng trong textarea
- `insert(String str, int pos)`
  Đặt chuỗi vào một vị trí xác định.
- `append(String str)`
  Thêm chuỗi vào cuối textarea
- `paramString()`
  Trả về đoạn văn bản trong textarea

Swing Button Classes



JButton

- *Button* là một thành phần gây ra một sự kiện hành động khi được kích chuột.
- Các constructor của JButton:

```
JButton()

JButton(String text)

JButton(String text, Icon icon)

JButton(Icon icon)
```

Các thuộc tính JButton

- text
- icon
- mnemonic
- horizontalAlignment
- verticalAlignment
- horizontalTextPosition
- verticalTextPosition

JToggleButton

- Các nút lệnh thay đổi trạng thái
  - Nhận các giá trị **on/off** hoặc **true/false**
  - Swing hỗ trợ các kiểu:
    - **JCheckBox**
    - **JRadioButton**

Item Event

- Được tạo ra khi người dùng chọn các mục khác nhau trên JCheckBox, JRadioButton,..
- Các phương thức
  - `Object getItem()`: trả về mục được chọn
  - `int getStateChange()`: trả về trạng thái trạng thái của mục chọn (DESELECTED/SELECTED)

Item Listener

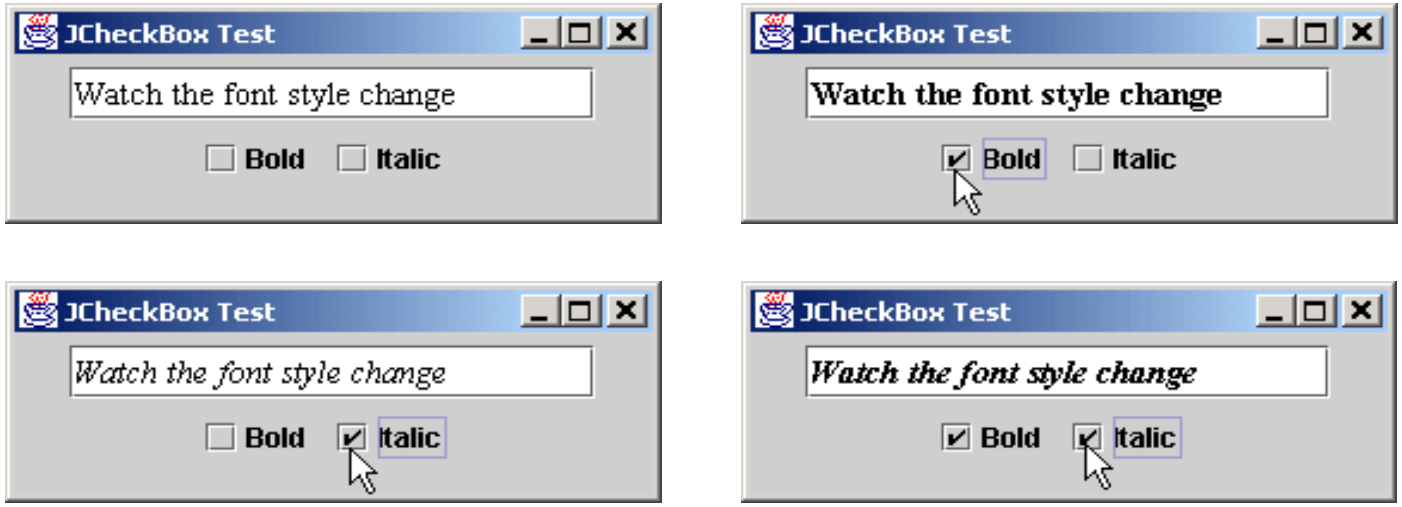
- `void itemStateChanged(ItemEvent e)`: được gọi thi hành khi người dùng chọn hoặc bỏ chọn 1 mục.

JCheckBox

- Checkbox được dùng để cung cấp cho người dùng khả năng lựa chọn
- Các contructor JCheckBox:

```
JCheckBox()
JCheckBox(Icon icon)
JCheckBox(Icon icon, boolean selected)
JCheckBox(String text)
JCheckBox(String text, boolean selected)
JCheckBox(String text, Icon icon)
JCheckBox(String text, Icon icon, boolean selected)
JCheckBox(Action a)
```

JCheckBox (tt)



Line 13

Line 27

Lines 31-35

```
1  // Fig. 12.11: CheckBoxTest.java
2  // Creating Checkbox buttons.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class CheckBoxTest extends JFrame {
12    private JTextField field;
13    private JCheckBox bold, italic;
14
15    // set up GUI
16    public CheckBoxTest()
17    {
18       super( "JCheckBox Test" );
19
20       // get content pane and set its layout
21       Container container = getContentPane();
22       container.setLayout( new FlowLayout() );
23
24       // set up JTextField and set its font
25       field =
26          new JTextField( "Watch the font style change", 20 );
27       field.setFont( new Font( "Serif", Font.PLAIN, 14 ) );
28       container.add( field );
29
30       // create checkbox objects
31       bold = new JCheckBox( "Bold" );
32       container.add( bold );
33
34       italic = new JCheckBox( "Italic" );
35       container.add( italic );
```

Declare two **JCheckBox** instances

Set **JTextField** font to Serif, 14-point plain

Instantiate **JCheckBox**s for bolding and italicizing **JTextField** text, respectively

```
36
37          // register listeners for JCheckBoxes
38          CheckBoxHandler handler = new CheckBoxHandler();
39          bold.addItemListener( handler );
40          italic.addItemListener( handler );
41
42          setSize( 275, 100 );
43          setVisible( true );
44       }
45
46       // execute application
47       public static void main( String args[] )
48       {
49          CheckBoxTest application = new CheckBoxTest();
50
51          application.setDefaultCloseOperation(
52             JFrame.EXIT_ON_CLOSE );
53       }
54
55       // private inner class for ItemListener event handling
56       private class CheckBoxHandler implements ItemListener {
57          private int valBold = Font.PLAIN;
58          private int valItalic = Font.PLAIN;
59
60          // respond to checkbox events
61          public void itemStateChanged( ItemEvent event )
62          {
63             // process bold checkbox events
64             if ( event.getItem() == bold )
65
66                if ( event.getStateChange() == ItemEvent.SELECTED )
67                   valBold = Font.BOLD;
68                else
69                   valBold = Font.PLAIN;
```

Register **JCheckBox**s to receive events from **CheckBoxHandler**

Lines 38-40

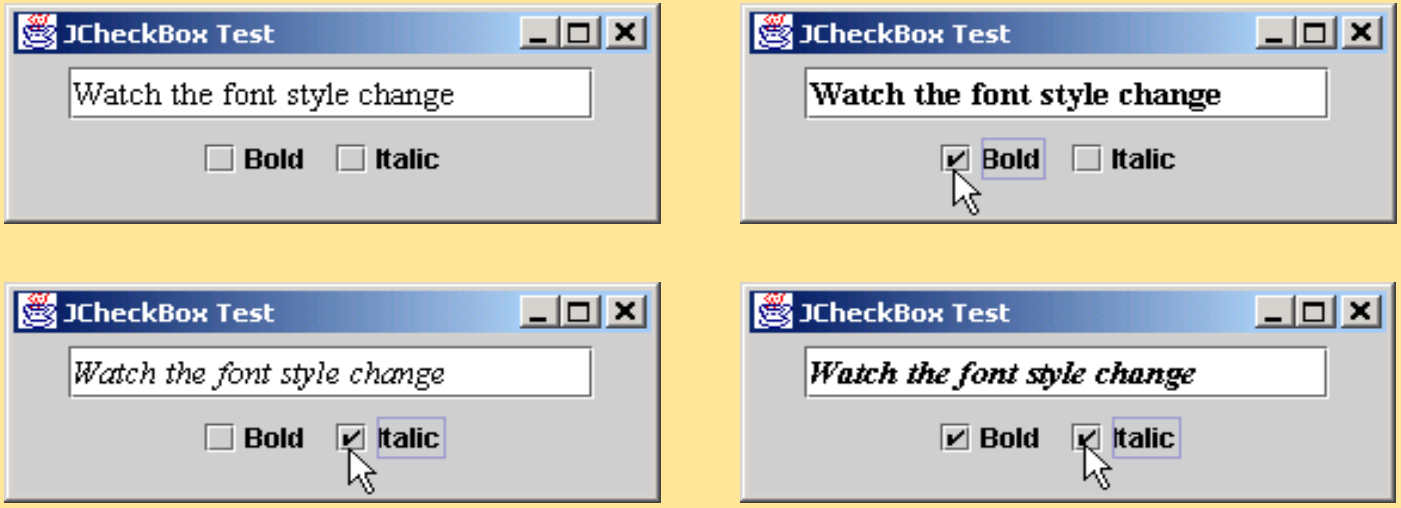
Line 61

When user selects **JCheckBox**, **CheckBoxHandler** invokes method **itemStateChanges** of all registered listeners

```
71          // process italic checkbox events
72          if ( event.getSource() == italic )
73
74             if ( event.getStateChange() == ItemEvent.SELECTED )
75                valItalic = Font.ITALIC;
76             else
77                valItalic = Font.PLAIN;
78
79          // set text field font
80          field.setFont(
81             new Font( "Serif", valBold + valItalic, 14 ) );
82       }
83
84    }  // end private inner class CheckBoxHandler
85
86 }  // end class CheckBoxTest
```

CheckBoxTes Java

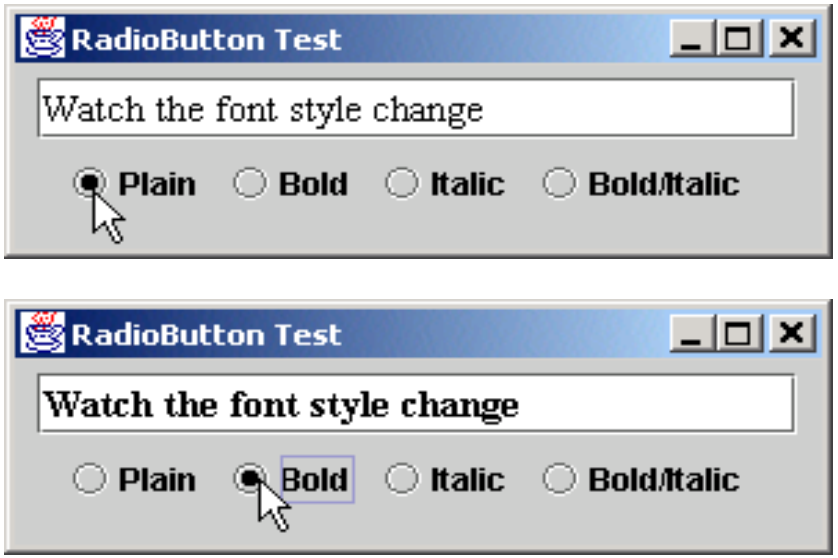
Change **JTextField** font, depending on which **JCheckBox** was selected

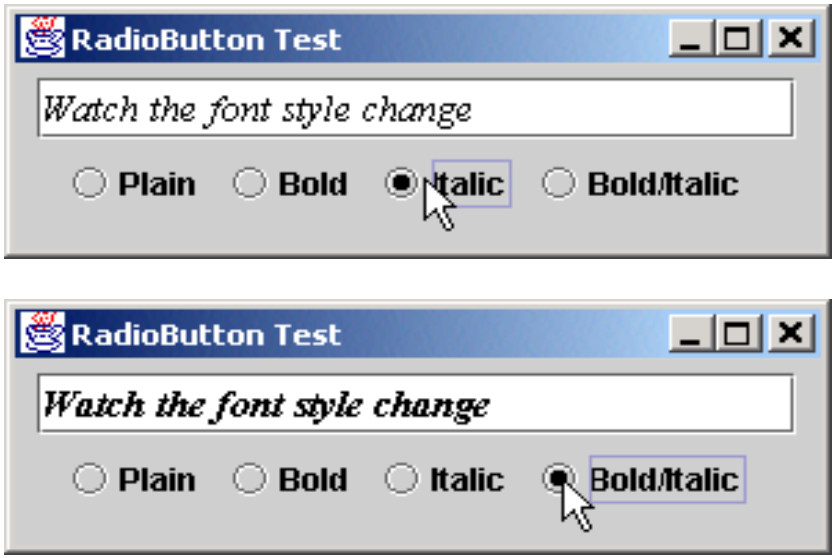


JRadioButton

- Một tập các nút cho phép chỉ lựa chọn được một tại một thời điểm
- Dùng lớp ButtonGroup để tạo ra nhóm
- Các contructor JRadioButton:
  - `JRadioButton(Icon icon)`
  - `JRadioButton(Icon icon, boolean selected)`
  - `JRadioButton(String text)`
  - `JRadioButton(String text, boolean selected)`
  - `JRadioButton(String text, Icon icon)`
  - `JRadioButton(String text, Icon icon, boolean selected)`
  - `JRadioButton(Action a)`

JRadioButton (tt)





```
1  // Fig. 12.12: RadioButtonTest.java
2  // Creating radio buttons using ButtonGroup and JRadioButton.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class RadioButtonTest extends JFrame {
12    private JTextField field;
13    private Font plainFont, boldFont, italicFont, boldItalicFont;
14    private JRadioButton plainButton, boldButton, italicButton,
15       boldItalicButton;
16    private ButtonGroup radioGroup;
17
18    // create GUI and fonts
19    public RadioButtonTest()
20    {
21       super( "RadioButton Test" );
22
23       // get content pane and set its layout
24       Container container = getContentPane();
25       container.setLayout( new FlowLayout() );
26
27       // set up JTextField
28       field =
29          new JTextField( "Watch the font style change", 25 );
30       container.add( field );
31
32       // create radio buttons
33       plainButton = new JRadioButton( "Plain", true );
34       container.add( plainButton );
35
```

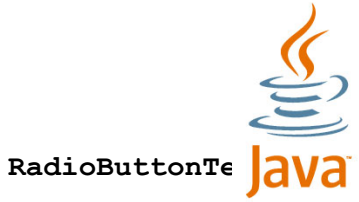
RadioButtonTe Java

Lines 14-15

Line 16

Declare four **JRadioButton** instances

JRadioButtons normally appear as a **ButtonGroup**




```
36       boldButton = new JRadioButton( "Bold", false);
37       container.add( boldButton );
38
39       italicButton = new JRadioButton( "Italic", false );
40       container.add( italicButton );
41
42       boldItalicButton = new JRadioButton(
43          "Bold/Italic", false );
44       container.add( boldItalicButton );
45
46       // register events for JRadioButtons
47       RadioButtonHandler handler = new RadioButtonHandler();
48       plainButton.addItemListener( handler );
49       boldButton.addItemListener( handler );
50       italicButton.addItemListener( handler );
51       boldItalicButton.addItemListener( handler );
52
53       // create logical relationship between JRadioButtons
54       radioGroup = new ButtonGroup();
55       radioGroup.add( plainButton );
56       radioGroup.add( boldButton );
57       radioGroup.add( italicButton );
58       radioGroup.add( boldItalicButton );
59
60       // create font objects
61       plainFont = new Font( "Serif", Font.PLAIN, 14 );
62       boldFont = new Font( "Serif", Font.BOLD, 14 );
63       italicFont = new Font( "Serif", Font.ITALIC, 14 );
64       boldItalicFont =
65          new Font( "Serif", Font.BOLD + Font.ITALIC, 14 );
66       field.setFont( plainFont );
67
68       setSize( 300, 100 );
69       setVisible( true );
70    }
```

Instantiate **JRadioButton**s for manipulating **JTextField** text font

Lines 47-51

Register **JRadioButton**s to receive events from **RadioButtonHandler**

JRadioButtons belong to **ButtonGroup**

```
71
72    // execute application
73    public static void main( String args[] )
74    {
75       RadioButtonTest application = new RadioButtonTest();
76
77       application.setDefaultCloseOperation(
78          JFrame.EXIT_ON_CLOSE );
79    }
80
81    // private inner class to handle radio button events
82    private class RadioButtonHandler implements ItemListener {
83
84       // handle radio button events
85       public void itemStateChanged( ItemEvent event )
86       {
87          // user clicked plainButton
88          if ( event.getItem() == plainButton )
89             field.setFont( plainFont );
90
91          // user clicked boldButton
92          else if (event.getItem() == boldButton )
93             field.setFont( boldFont );
94
95          // user clicked italicButton
96          else if (event.getItem() == italicButton )
97             field.setFont( italicFont );
98
99          // user clicked boldItalicButton
100         else if (event.getItem() == boldItalicButton )
101            field.setFont( boldItalicFont );
102      }
103
104   }  // end private inner class RadioButtonHandler
105
   // end class RadioButtonTest
```

RadioButtonTe Java

Lines 85-104

Lines 88-102

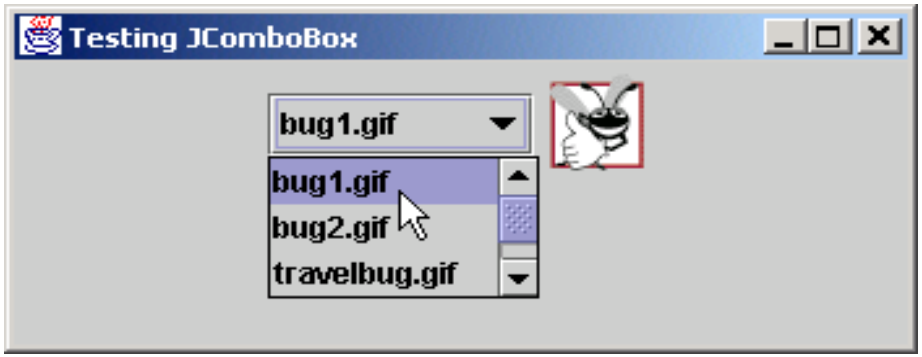
When user selects **JRadioButton**, **RadioButtonHandler** invokes method **itemStateChanged** of all registered listeners

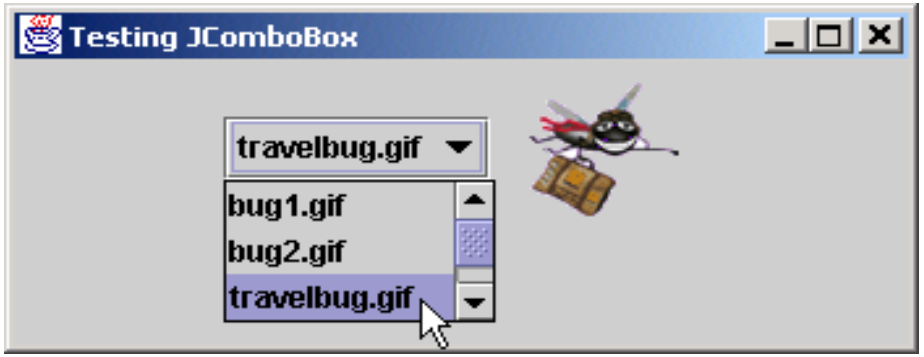
Set font corresponding to **JRadioButton** selected

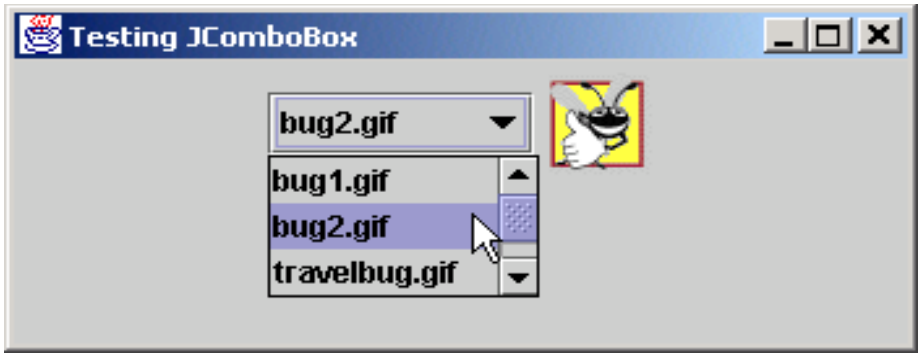
JComboBox

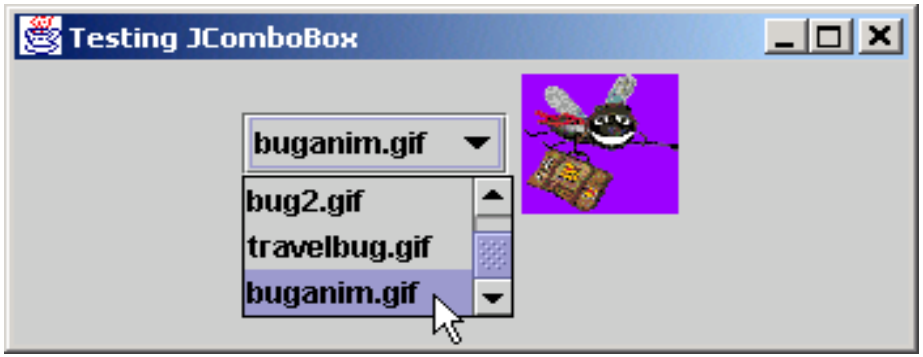
- **JComboBox**
  - Dùng để liệt kê danh sách các mục mà người dùng có thể chọn
  - Còn được gọi là *drop-down list*
  - Phát sinh sự kiện ItemEvent khi người sử dụng chọn 1 mục trong danh sách
- **Các constructor Jcombobox**
  - `JComboBox()`
  - `JComboBox(ComboBoxModel asModel)`
  - `JComboBox(Object []item)`

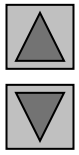
JComboBox (tt)











Out

ComboBoxTes

Lines 31-32

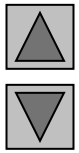
Line 34

```
1  // Fig. 12.13: ComboBoxTest.java
2  // Using a JComboBox to select an image to display.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class ComboBoxTest extends JFrame {
12    private JComboBox imagesComboBox;
13    private JLabel label;
14
15    private String names[] =
16       { "bug1.gif", "bug2.gif",  "travelbug.gif", "buganim.gif" };
17    private Icon icons[] = { new ImageIcon( names[ 0 ] ),
18       new ImageIcon( names[ 1 ] ), new ImageIcon( names[ 2 ] ),
19       new ImageIcon( names[ 3 ] ) };
20
21    // set up GUI
22    public ComboBoxTest()
23    {
24       super( "Testing JComboBox" );
25
26       // get content pane and set its layout
27       Container container = getContentPane();
28       container.setLayout( new FlowLayout() );
29
30       // set up JComboBox and register its event handler
31       imagesComboBox = new JComboBox( names );
32       imagesComboBox.setMaximumRowCount( 3 );
33
34       imagesComboBox.addItemListener(
35
```

Instantiate **JComboBox** to show three **String**s from **names** array at a time

Register **JComboBox** to receive events from anonymous **ItemListener**

```
36          // anonymous inner class to handle JComboBox events
37          new ItemListener() {
38
39             // handle JComboBox event
40             public void itemStateChanged( ItemEvent event )
41             {
42                // determine whether check box selected
43                if ( event.getStateChange() == ItemEvent.SELECTED )
44                   label.setIcon( icons[
45                      imagesComboBox.getSelectedIndex() ] );
46             }
47
48          }  // end anonymous inner class
49
50       ); // end call to addItemListener
51
52       container.add( imagesComboBox );
53
54       // set up JLabel to display ImageIcons
55       label = new JLabel( icons[ 0 ] );
56       container.add( label );
57
58       setSize( 350, 100 );
59       setVisible( true );
60    }
61
62    // execute application
63    public static void main( String args[] )
64    {
65       ComboBoxTest application = new ComboBoxTest();
66
67       application.setDefaultCloseOperation(
68          JFrame.EXIT_ON_CLOSE );
69    }
70
71 // end class ComboBoxTest
```



Out

ComboBoxTes

Lines 40-46

Lines 43-45

When user selects item in **JComboBox**, **ItemListener** invokes method **itemStateChanged** of all registered listeners

Set appropriate **Icon** depending on user selection

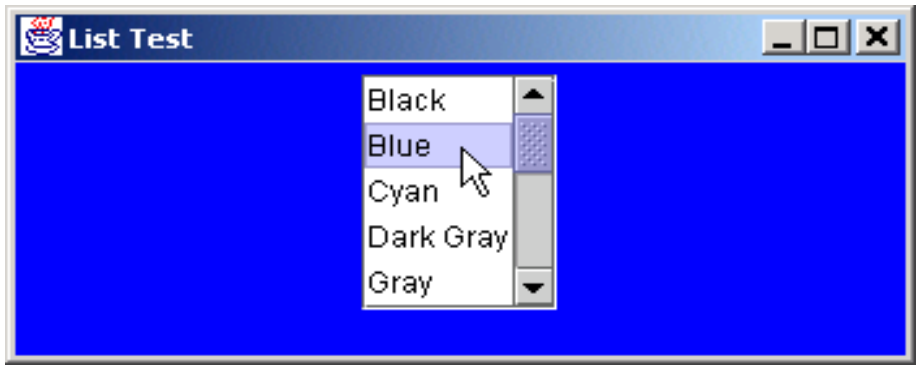
JList

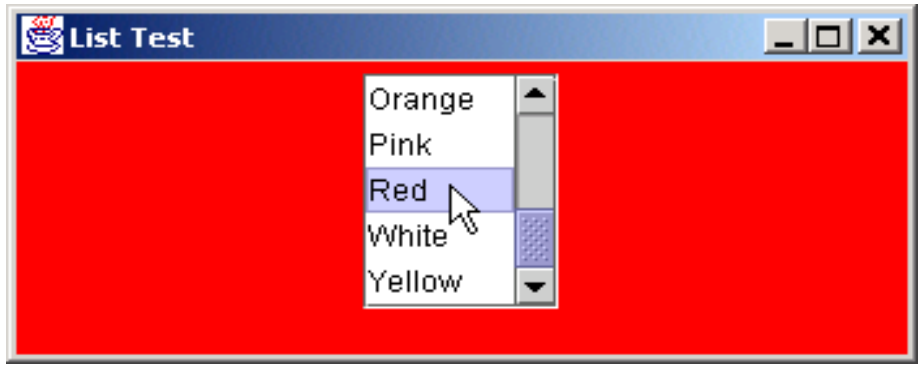
- **Jlist**
  - Danh sách các mục chọn
  - Có thể chọn 1 hoặc nhiều mục
  - Cho phép sắp xếp dữ liệu hiển thị, phân nhóm
  - Có thể hiển thị chuỗi và icon
  - Không hỗ trợ doublclik chuột
  - Phát sinh ListSelectionEvent khi người dùng chọn
- **Các constructor Jlist**
  - `JList()`
  - `JList(ListModel dataModel)`
  - `JList(Object []listData)`

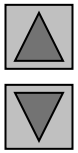
Jlist (tt)

- **Các phương thức**
  - `JList(Object[] listData)`
  - `int getSelectedIndex()`
  - `Object[] getSelectedValues()`
  - `void setListData(Object[] listData)`
  - `void setSelectedIndex(int idx)`
- **ListSelectionListener**
  - `void valueChanged(ListSelectionEvent e)`

Jlist Demo


List Test
Black
Blue
Cyan
Dark Gray
Gray


List Test
Orange
Pink
Red
White
Yellow



Out

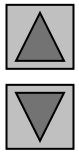
ListTest.: Java

Line 34

```
1  // Fig. 12.14: ListTest.java
2  // Selecting colors from a JList.
3
4  // Java core packages
5  import java.awt.*;
6
7  // Java extension packages
8  import javax.swing.*;
9  import javax.swing.event.*;
10
11 public class ListTest extends JFrame {
12    private JList colorList;
13    private Container container;
14
15    private String colorNames[] = { "Black", "Blue", "Cyan",
16       "Dark Gray", "Gray", "Green", "Light Gray", "Magenta",
17       "Orange", "Pink", "Red", "White", "Yellow" };
18
19    private Color colors[] = { Color.black, Color.blue,
20       Color.cyan, Color.darkGray, Color.gray, Color.green,
21       Color.lightGray, Color.magenta, Color.orange, Color.pink,
22       Color.red, Color.white, Color.yellow };
23
24    // set up GUI
25    public ListTest()
26    {
27       super( "List Test" );
28
29       // get content pane and set its layout
30       container = getContentPane();
31       container.setLayout( new FlowLayout() );
32
33       // create a list with items in colorNames array
34       colorList = new JList( colorNames );
35       colorList.setVisibleRowCount( 5 );
```

Use **colorNames** array to populate **JList**

```
36
37       // do not allow multiple selections
38       colorList.setSelectionMode(
39          ListSelectionModel.SINGLE_SELECTION );
40
41       // add a JScrollPane containing JList to content pane
42       container.add( new JScrollPane( colorList ) );
43
44       // set up event handler
45       colorList.addListSelectionListener(
46
47          // anonymous inner class for list selection even
48          new ListSelectionListener() {
49
50             // handle list selection events
51             public void valueChanged( ListSelectionEvent event )
52             {
53                container.setBackground(
54                   colors[ colorList.getSelectedIndex() ] );
55             }
56
57          }  // end anonymous inner class
58
59       ); // end call to addListSelectionListener
60
61       setSize( 350, 150 );
62       setVisible( true );
63    }
64
65    // execute application
66    public static void main( String args[] )
67    {
68       ListTest application = new ListTest();
69
```



Out

JList allows single selection

Lines 38-39

Register **JList** to receive events from anonymous **ListSelectionListener**

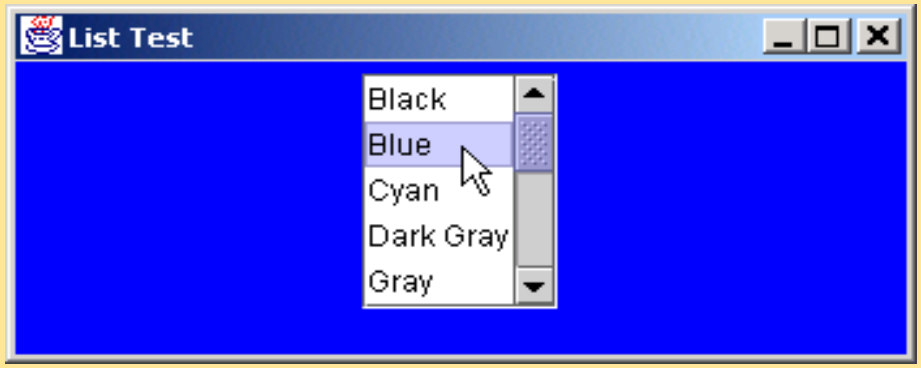
Lines 51-55

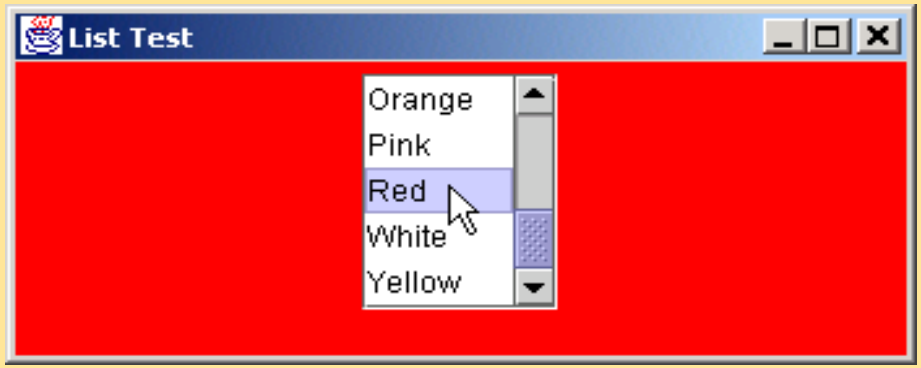
Lines 53-54

When user selects item in **JList**, **ListSelectionListener** invokes method **valueChanged** of all registered listeners

Set appropriate background depending on user selection

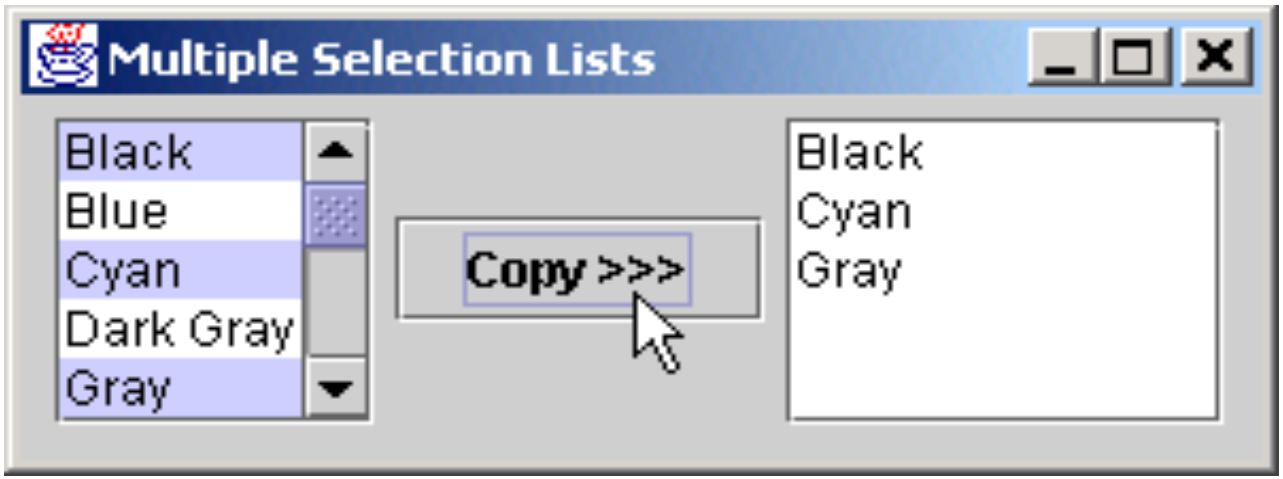
```
70         application.setDefaultCloseOperation(
71            JFrame.EXIT_ON_CLOSE );
72      }
73
74   }  // end class ListTest
```

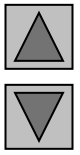




Multiple-Selection Lists

- **Multiple-selection list**
  - Chọn nhiều mục trên **Jlist**





Out

MultipleSelection.java

Line 29

Lines 32-33

```
1  // Fig. 12.15: MultipleSelection.java
2  // Copying items from one List to another.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class MultipleSelection extends JFrame {
12    private JList colorList, copyList;
13    private JButton copyButton;
14
15    private String colorNames[] = { "Black", "Blue", "Cyan",
16       "Dark Gray", "Gray", "Green", "Light Gray",
17       "Magenta", "Orange", "Pink", "Red", "White", "Yellow" };
18
19    // set up GUI
20    public MultipleSelection()
21    {
22       super( "Multiple Selection Lists" );
23
24       // get content pane and set its layout
25       Container container = getContentPane();
26       container.setLayout( new FlowLayout() );
27
28       // set up JList colorList
29       colorList = new JList( colorNames );
30       colorList.setVisibleRowCount( 5 );
31       colorList.setFixedCellHeight( 15 );
32       colorList.setSelectionMode(
33          ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
34       container.add( new JScrollPane( colorList ) );
```

Use **colorNames** array to populate **JList**

JList colorList allows multiple selections

Out

MultipleSelection.java

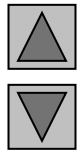
Lines 48-49

Lines 63-64

```
36       // create copy button and register its listener
37       copyButton = new JButton( "Copy >>>" );
38
39       copyButton.addActionListener(
40
41          // anonymous inner class for button event
42          new ActionListener() {
43
44             // handle button event
45             public void actionPerformed( ActionEvent event )
46             {
47                // place selected values in copyList
48                copyList.setListData(
49                   colorList.getSelectedValues() );
50             }
51
52          }  // end anonymous inner class
53
54       ); // end call to addActionListener
55
56       container.add( copyButton );
57
58       // set up JList copyList
59       copyList = new JList();
60       copyList.setVisibleRowCount( 5 );
61       copyList.setFixedCellWidth( 100 );
62       copyList.setFixedCellHeight( 15 );
63       copyList.setSelectionMode(
64          ListSelectionModel.SINGLE_INTERVAL_SELECTION );
65       container.add( new JScrollPane( copyList ) );
66
67       setSize( 300, 120 );
68       setVisible( true );
69    }
```

When user presses **JButton**, **JList copyList** adds items that user selected from **JList colorList**

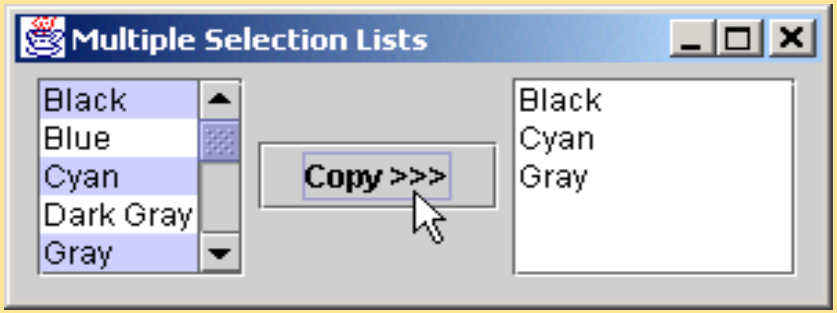
JList colorList allows single selections



Out

MultipleSelection.java

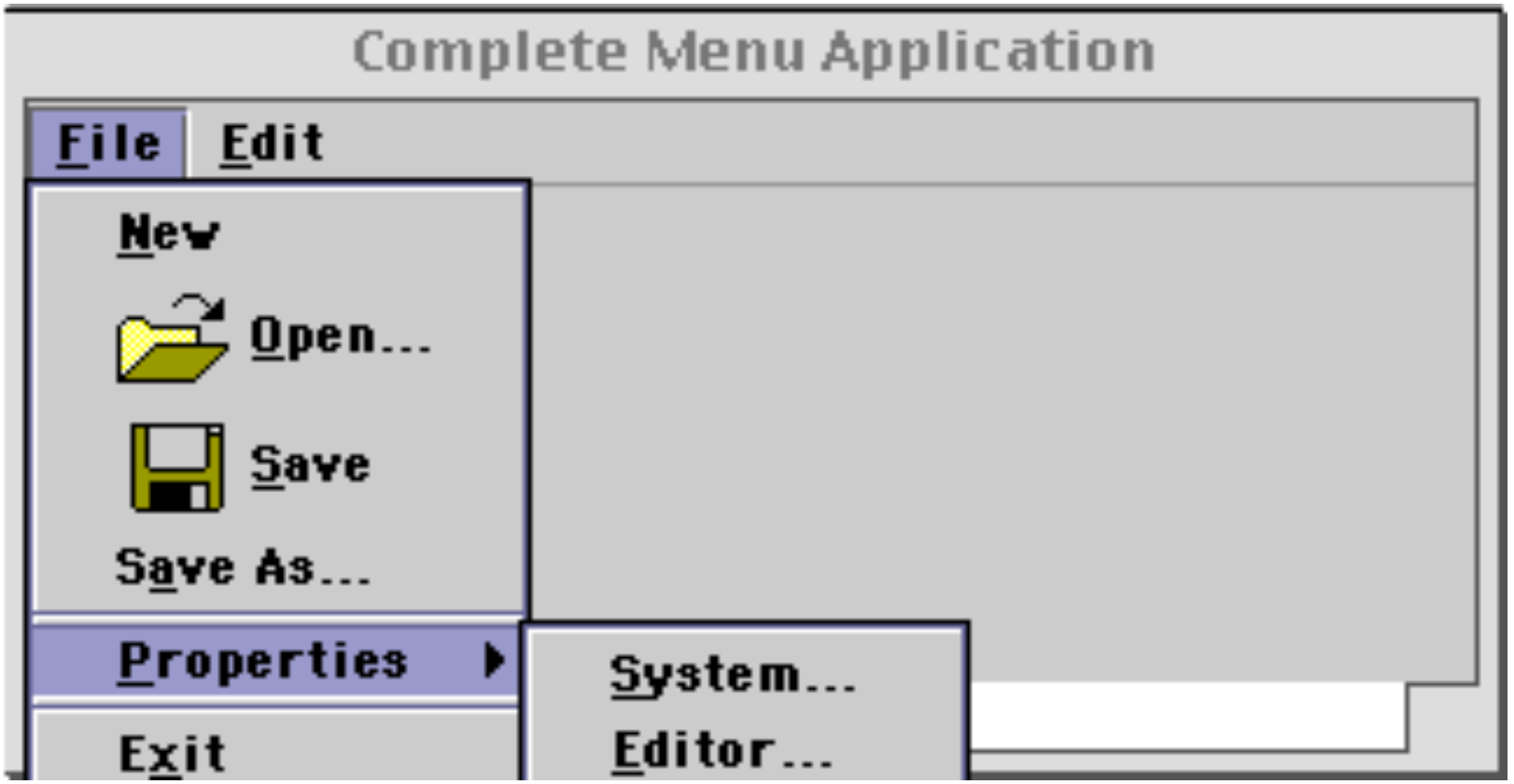
```
71    // execute application
72    public static void main( String args[] )
73    {
74       MultipleSelection application = new MultipleSelection();
75
76       application.setDefaultCloseOperation(
77          JFrame.EXIT_ON_CLOSE );
78    }
79
80 }  // end class MultipleSelection
```



Menus

- Java cung cấp một số lớp - `JMenuBar, JMenu, JMenuItem, JCheckBoxMenuItem, và JRadioButtonMenuItem` - để thực thi menu trong một frame.
- Một JFrame hoặc JApplet có thể chứa một *menu bar* trên đó có gắn các *pull-down menu*. Các menu chứa các *menu item* để người dùng lựa chọn (hoặc bật/tắt). Menu bar có thể được xem như một cấu trúc để hỗ trợ các menu.

Menus (tt)


Complete Menu Application
File
Edit
New
Open...
Save
Save As...
Properties
Exit
System...
Editor...

Lớp JMenuBar

- *Menu bar* chứa các menu; menu bar chỉ có thể được thêm vào 1 frame. Đoạn code sau tạo và thêm một JMenuBar vào 1 frame:

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

Lớp JMenu

- Gắn các menu vào một JMenuBar. Đoạn code sau tạo 2 menu File và Help, và thêm chúng vào JMenuBar mb:

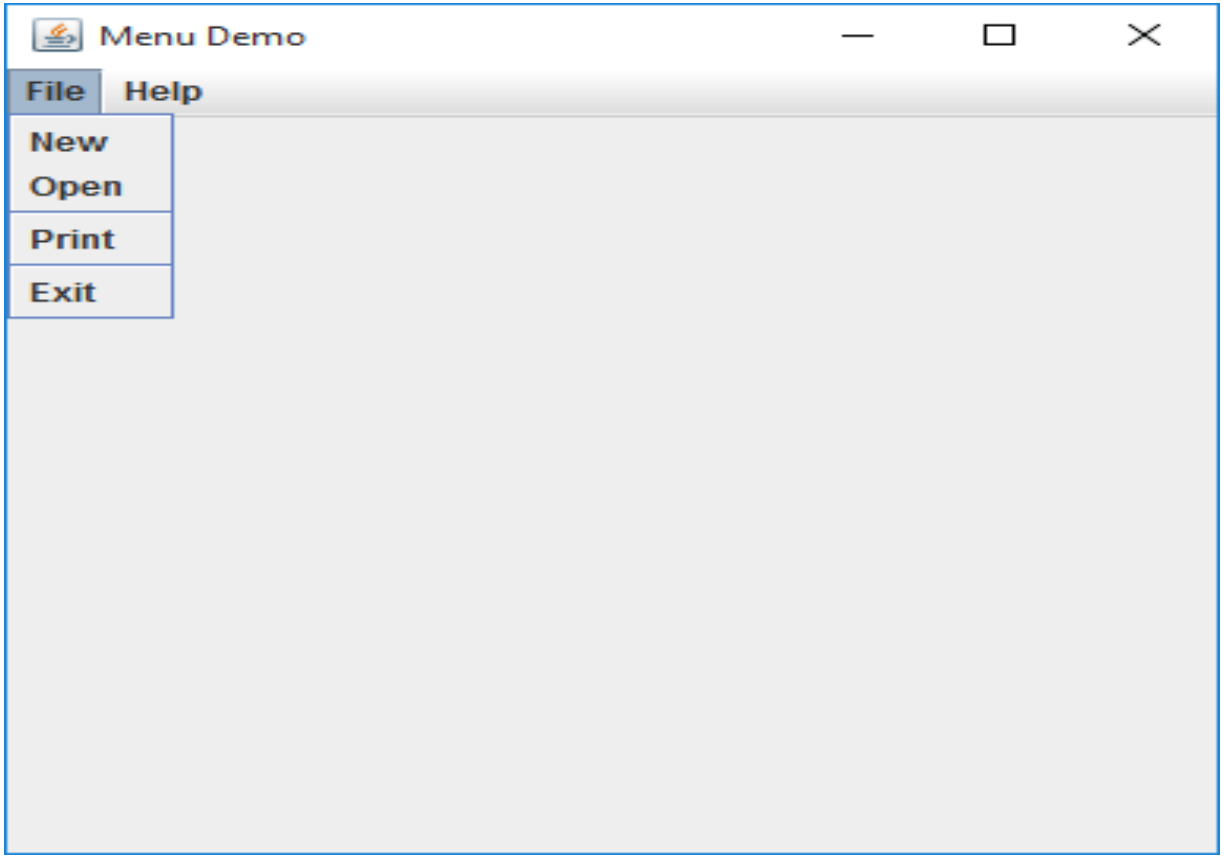
```
JMenu fileMenu = new JMenu("File");
JMenu helpMenu = new JMenu("Help");
mb.add(fileMenu);
mb.add(helpMenu);
```

Lớp JMenuItem

- Đoạn code sau thêm các mục chọn (menu item) và các separator trong menu fileMenu:

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

Menu Demo

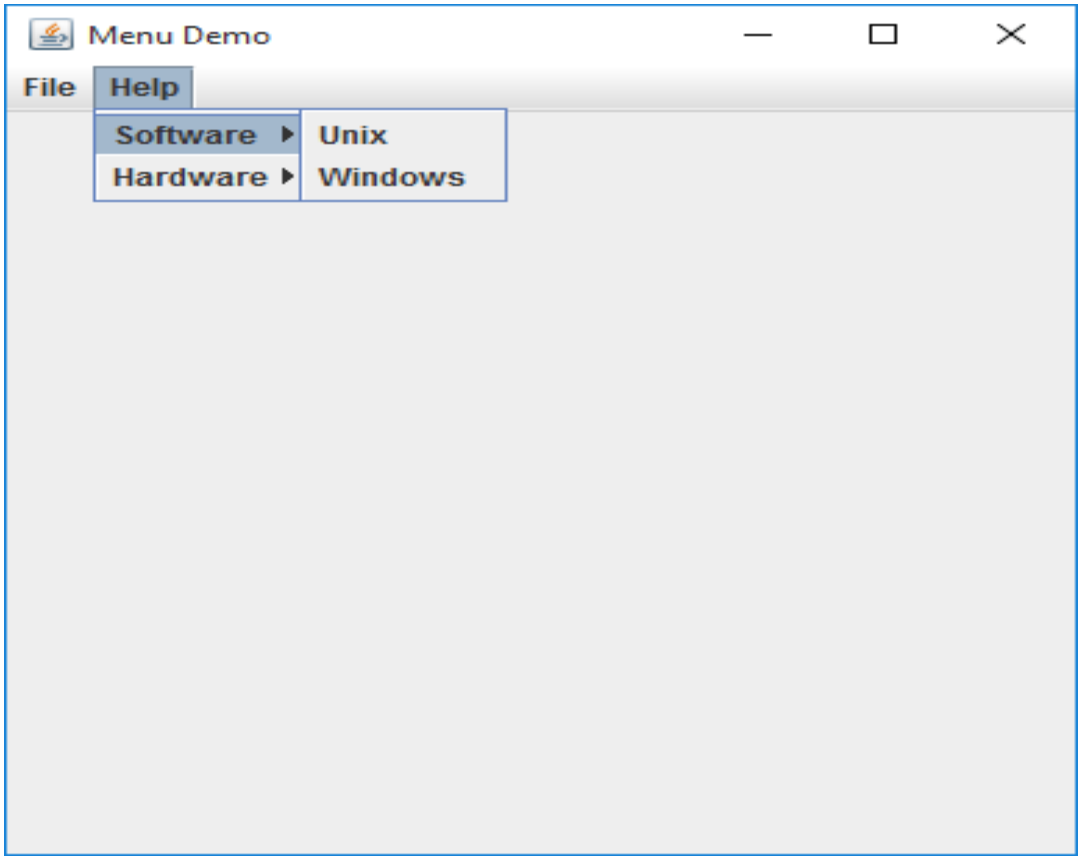


Submenus

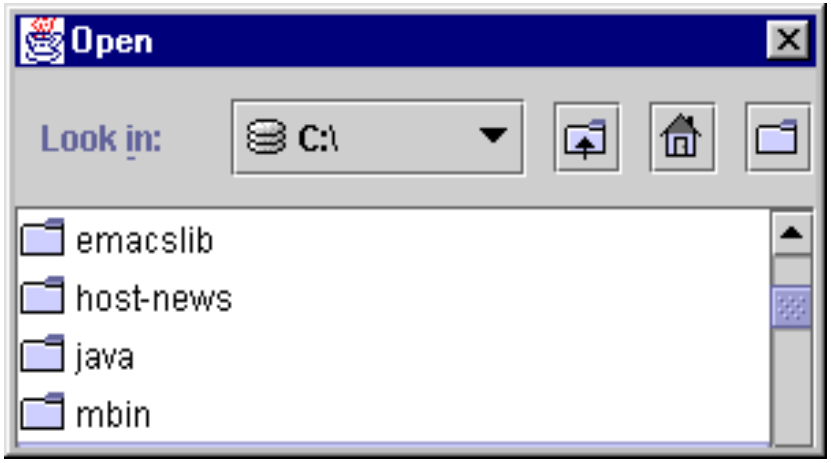
- Thêm các submenus vào các menu item. Đoạn code sau thêm các submenu “Unix”, “NT”, và “Win95” vào trong mục chọn “Software”.

```
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(“Windows"));
```

Submenu Demo



SWING Dialog Boxes

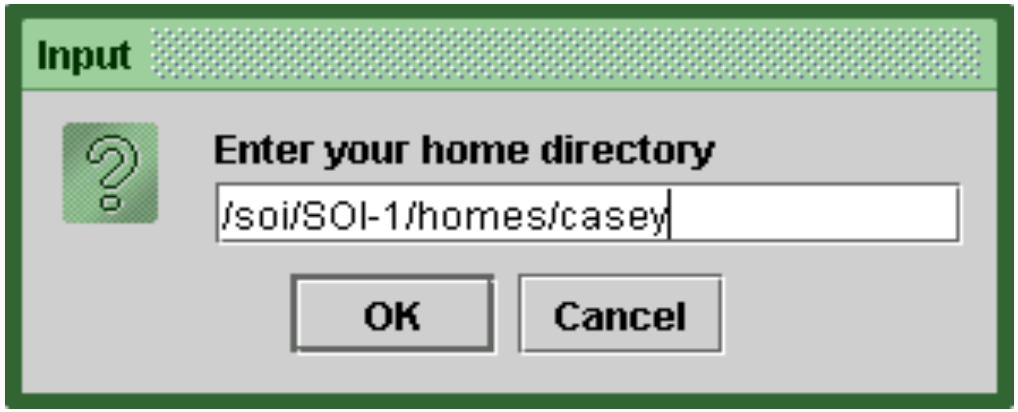


- Nhận thông tin từ người sử dụng
  - Số liệu,…
  - Danh mục tập tin,…
- Hiển thị kết quả
  - Hiển thị thông tin cảnh báo
  - In kết quả lên màn hình,…

JOptionPane

Static method call

```
JOptionPane.showInputDialog("Enter your home directory");
```



JOptionPane (tt)

- 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.

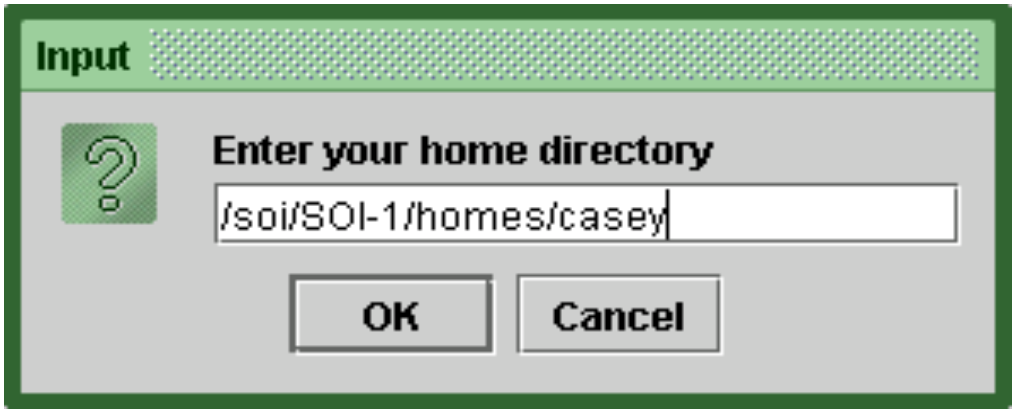
Sử Dụng JOptionPane

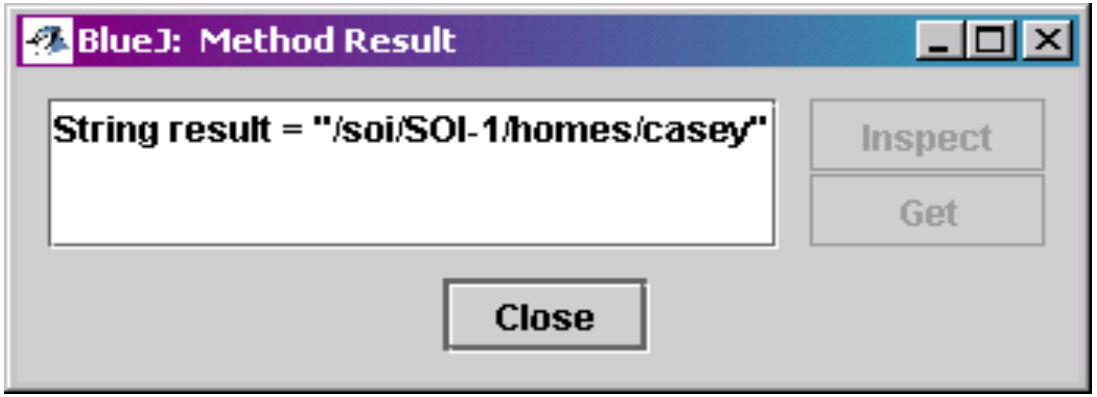

```
import javax.swing.*;  // import JAVA SWING graphical interface libraries

// This class demonstrates use of JOptionPane
public class SimpleDialog
{
    public String inputDialog(String s)
    {
        return JOptionPane.showInputDialog(s);  // static method call
    }
}
```

static int	**showConfirmDialog**(Component parentComponent, Object message, String tit int optionType, int messageType) Brings up a dialog where the number of choices is determined by the optionType para where the messageType parameter determines the icon to display.
static int	**showConfirmDialog**(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) Brings up a dialog with a specified icon, where the number of choices is determined by the optionType parameter.
static String	**showInputDialog**(Component parentComponent, Object message) Shows a question-message dialog requesting input from the user parented to parentComponent.
static String	**showInputDialog**(Component parentComponent, Object message, Object initialSelectionValue) Shows a question-message dialog requesting input from the user and parented to parentComponent.
static String	**showInputDialog**(Component parentComponent, Object message, String title, int messageType) Shows a dialog requesting input from the user parented to parentComponent with the dialog having the title title and message type messageType.
static Object	**showInputDialog**(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue) Prompts the user for input in a blocking dialog where the initial selection, possible selections, and all other options can be specified.
static String	**showInputDialog**(Object message) Shows a question-message dialog requesting input from the user.
static String	**showInputDialog**(Object message, Object initialSelectionValue) Shows a question-message dialog requesting input from the user, with the input value initialized to initialSelectionValue.

JOptionPane

```
JOptionPane.showInputDialog("Enter your home directory");
```

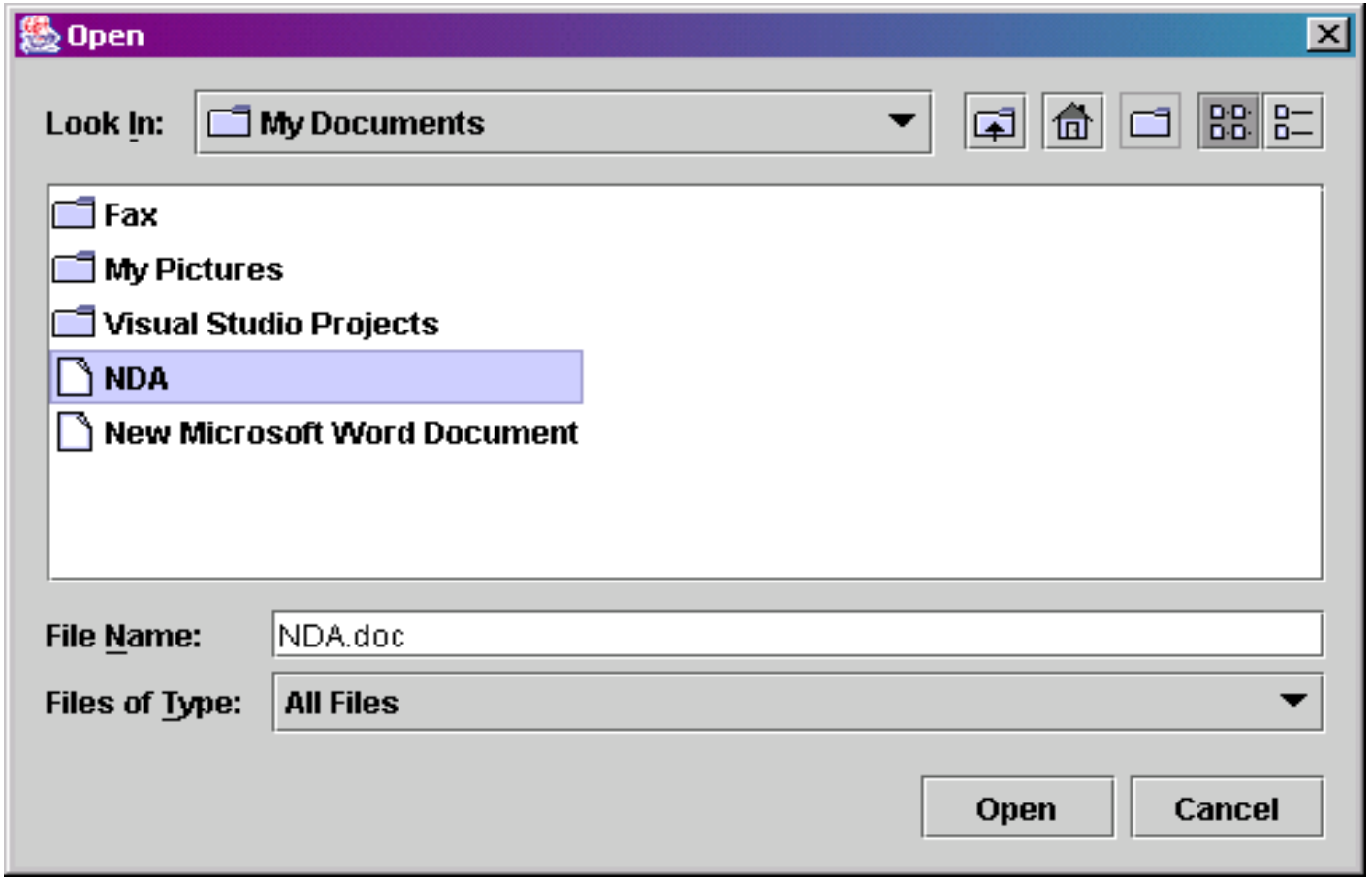




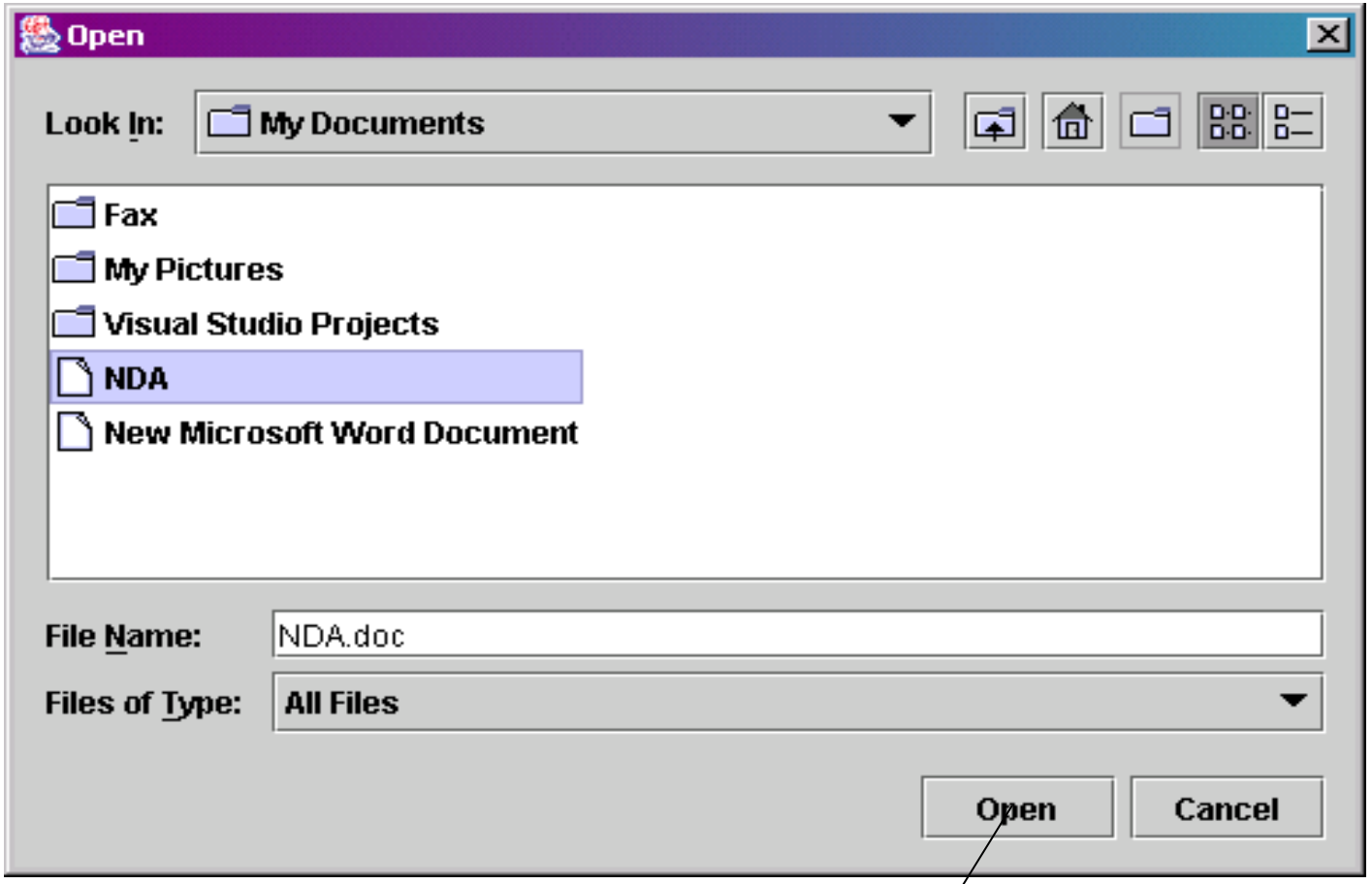
JFileChooser

- int showOpenDialog(Component parent): mở hộp thoại đọc tập tin
- int showSaveDialog(Component parent): mở hộp thoại lưu tập tin
- JFileChooser() : tạo đối tượng dùng để mở hộp thoại ghi/đọc tập tin
- File getSelectedFile(): lấy thông tin về tập tin/thư mục được chọn
  - String getPath()
  - String getName()

JFileChooser (tt)

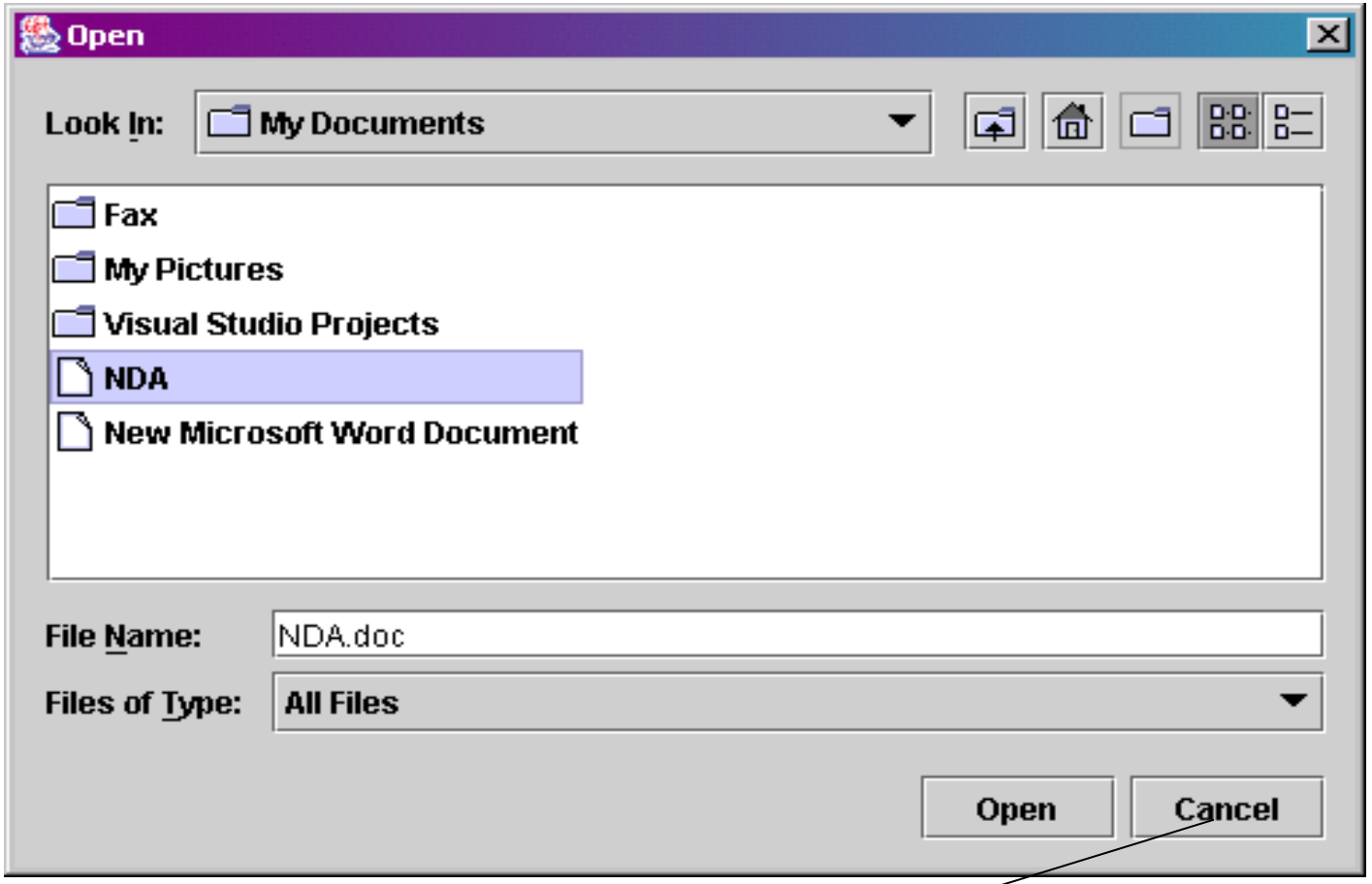


JFileChooser (tt)



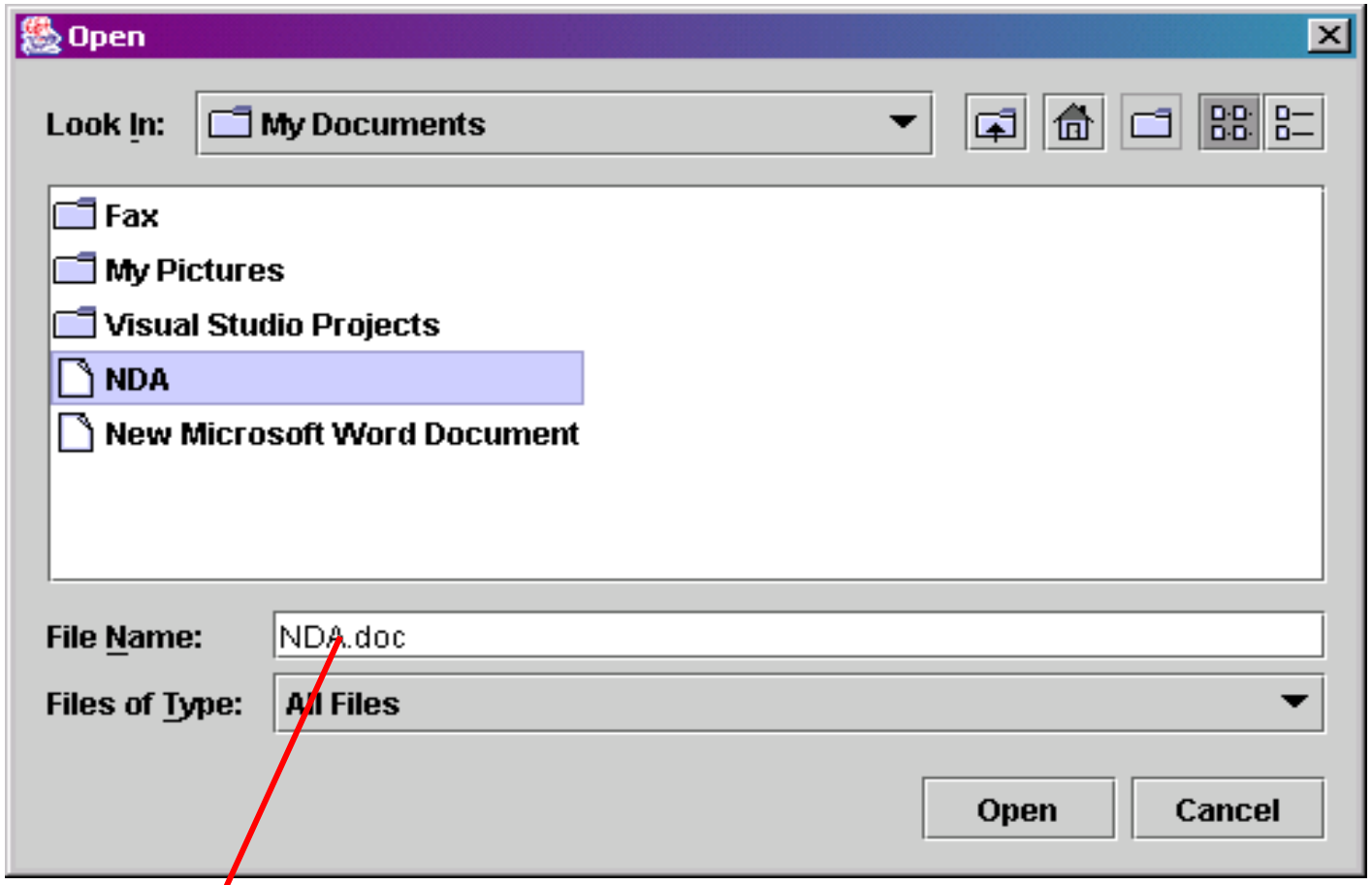
`JFileChooser.APPROVE_OPTION`

JFileChooser (tt)



JFileChooser.CANCEL_OPTION

JFileChooser (tt)




```
if (returnValue == JFileChooser.APPROVE_OPTION)
        filename = fc.getName();
```

showOpenDialog

```
public int showOpenDialog(Component parent)
                   throws HeadlessException
```

Pops up an "Open File" file chooser dialog. Note that the text that appears in the approve button is determined by the L&F.

Parameters:

`parent` - the parent component of the dialog, can be `null`; see `showDialog` for details

Returns:

the return state of the file chooser on popdown:

- JFileChooser.CANCEL_OPTION
- JFileChooser.APPROVE_OPTION
- JFileCHooser.ERROR_OPTION if an error occurs or the dialog is dismissed

Throws:

`HeadlessException` - if GraphicsEnvironment.isHeadless() returns true.

See Also:

`GraphicsEnvironment.isHeadless()`, `showDialog(java.awt.Component, java.lang.String)`

---

showSaveDialog

```
public int showSaveDialog(Component parent)
                   throws HeadlessException
```

Pops up a "Save File" file chooser dialog. Note that the text that appears in the approve button is determined by the L&F.

showOpenDialog

```
public int showOpenDialog(Component parent)
                   throws HeadlessException
```

Pops up an "Open File" file chooser dialog. Note that the text that appears in the approve button is determined by the L&F.

Parameters:

`parent` - the parent component of the dialog, can be `null`; see `showDialog` for details

Returns:

the return state of the file chooser on popdown:

- JFileChooser.CANCEL_OPTION
- JFileChooser.APPROVE_OPTION
- JFileCHooser.ERROR_OPTION if an error occurs or the dialog is dismissed

Throws:

`HeadlessException` - if GraphicsEnvironment.isHeadless() returns true.

See Also:

`GraphicsEnvironment.isHeadless()`, `showDialog(java.awt.Component, java.lang.String)`

---

showSaveDialog

```
public int showSaveDialog(Component parent)
                   throws HeadlessException
```

Pops up a "Save File" file chooser dialog. Note that the text that appears in the approve button is determined by the L&F.

String	**getName**(File f) Returns the filename.
File	**getSelectedFile**() Returns the selected file.
File[]	**getSelectedFiles**() Returns a list of selected files if the file chooser is set to allow multiple selection.
String	**getTypeDescription**(File f) Returns the file type.
FileChooserUI	**getUI**() Gets the UI object which implements the L&F for this component.
String	**getUIClassID**() Returns a string that specifies the name of the L&F class that renders this component.
boolean	**isAcceptAllFileFilterUsed**() Returns whether the `AcceptAll FileFilter` is used.
boolean	**isDirectorySelectionEnabled**() Convenience call that determines if directories are selectable based on the current file selection mode.
boolean	**isFileHidingEnabled**() Returns true if hidden files are not shown in the file chooser; otherwise, returns false.
boolean	**isFileSelectionEnabled**() Convenience call that determines if files are selectable based on the current file selection mode.
boolean	**isMultiSelectionEnabled**() Returns true if multiple files can be selected.

Dùng JFileChooser


```
import java.io.*;       // import JAVA file and stream handling classes
import javax.swing.*;   // import JAVA SWING graphical interface libraries

// This class demonstrates use of JFileChooser
public class SimpleDialog
{
    String fileName;
    File theFile;
    public int fileChooserDialog()
    {
        // Make an instance, call the showOpenDialog method and return result
        JFileChooser fc = new JFileChooser();
        int retval = fc.showOpenDialog(null);
        if(retval==JFileChooser.APPROVE_OPTION){
            theFile=fc.getSelectedFile();
            fileName=fc.getName(theFile);
        }
        return retval;
    }
}
```

a `File` object
a `String`

Lọc Tập Tin Hiển Thị

- FileNameExtensionFilter(String dispc, String filter)
  - FileNameExtensionFilter filter = new FileNameExtensionFilter("JPG & GIF Images", "jpg", "gif");
- JFileChooser :
  - setFileFilter(FileNameExtensionFilter filter)

JTabbedPane

JTabbedPane (tt)

javax.swing

Class JTabbedPane

java.lang.Object
  └ java.awt.Component
    └ java.awt.Container
      └ javax.swing.JComponent
        └ **javax.swing.JTabbedPane**

All Implemented Interfaces:

ImageObserver, MenuContainer, Serializable, Accessible, SwingConstants

JTabbedPane (tt)

- Tạo mới đối tượng JTabbedPane

```
JTabbedPane tabbedPane = new JTabbedPane();
```

- Gắn thêm 1 Tab mới vào đối tượng JTabbedPane

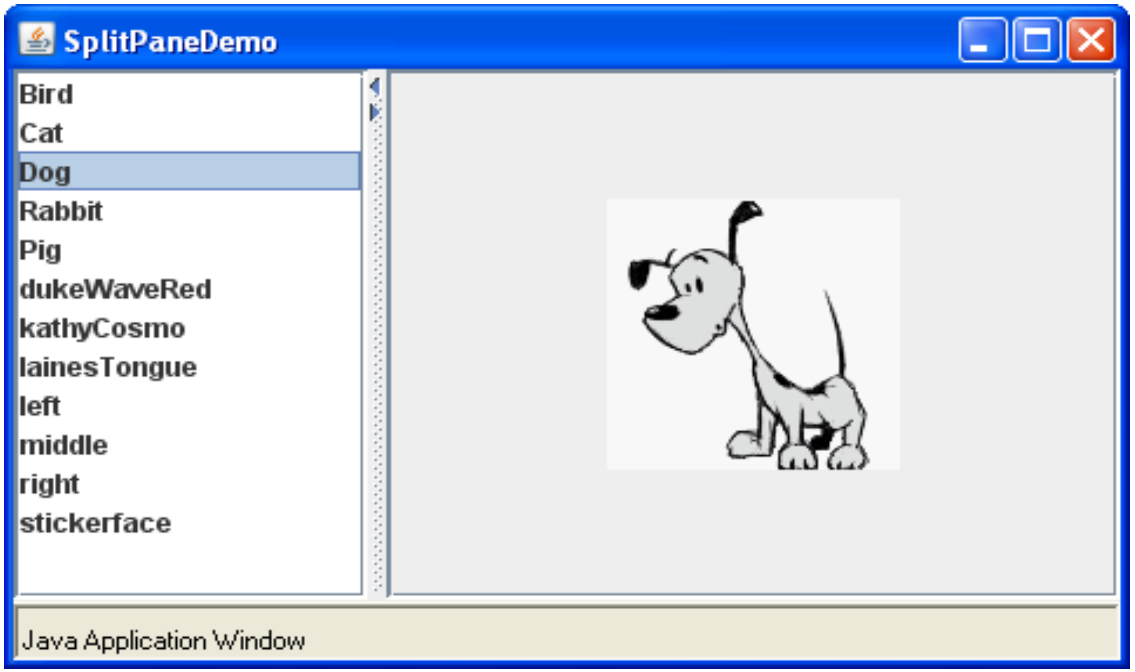
```
tabbedPane.addTab("Tab name", icon, component, "Tooltip");
```

```
JTabbedPane tabbedPane = new JTabbedPane();
ImageIcon icon = new ImageIcon("middle.gif");
JPanel panel1 = new JPanel(new FlowLayout());
panel1.add(new JLabel("Please add more components into the Tab no.1"));
tabbedPane.addTab("Tab 1", icon, panel1, "This is the tab no.1");
tabbedPane.setMnemonicAt(0, KeyEvent.VK_1);

JPanel panel2 = new JPanel(new FlowLayout());
panel2.add(new JLabel("Please add more components into the Tab no.2"));
tabbedPane.addTab("Tab 2", icon, panel2, "This is the tab no.2");
tabbedPane.setMnemonicAt(1, KeyEvent.VK_2);
```

JTabbedPane (tt)


```
JPanel panel3 = new JPanel(new FlowLayout());
panel3.add(new JLabel("Please add more components into the Tab no.3"));
tabbedPane.addTab("Tab 3", icon, panel3, "This is the tab no.3");
tabbedPane.setMnemonicAt(2, KeyEvent.VK_3);

JPanel panel4 = new JPanel(new FlowLayout());
panel4.add(new JLabel("Please add more components into the Tab no.4"));
panel4.setPreferredSize(new Dimension(410, 50));
tabbedPane.addTab("Tab 4", icon, panel4, "This is the tab no.4");
tabbedPane.setMnemonicAt(3, KeyEvent.VK_4);

JFrame fr = new JFrame("JTabbedPane Demo");
fr.setBounds(50, 50, 400, 300);
fr.add(tabbedPane, BorderLayout.CENTER);

fr.setVisible(true);
```

JSplitPane



http://java.sun.com/docs/books/tutorial/uiswing/examples/components/

JSplitPane (tt)


```
JFrame fr = new JFrame("JSplitPane Demo");
fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JSplitPane splitPane;
JLabel picture = new JLabel();;
String[] imageNames = { "Bird", "Cat", "Dog", "Rabbit", "Pig", "dukeWaveRed",
                        "kathyCosmo", "lainesTongue", "left", "middle", "right", "stickerface"};

//Create the list of images and put it in a scroll pane.
JList list = new JList(imageNames);
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
list.setSelectedIndex(0);
MyListSelectionListener listener = new MyListSelectionListener(picture, imageNames);
list.addListSelectionListener(listener);
JScrollPane listScrollPane = new JScrollPane(list);
```

JSplitPane, JScrollPane, JList


```
// Create a split pane with the two scroll panes in it.
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, listScrollPane, pictureScrollPane);
splitPane.setOneTouchExpandable(true);
splitPane.setDividerLocation(150);

// Provide minimum sizes for the two components in the split pane.
Dimension minimumSize = new Dimension(100, 50);
listScrollPane.setMinimumSize(minimumSize);
pictureScrollPane.setMinimumSize(minimumSize);

//Provide a preferred size for the split pane.
splitPane.setPreferredSize(new Dimension(400, 200));

fr.getContentPane().add(splitPane);
fr.pack();
fr.setVisible(true);
```

JSplitPane, JScrollPane, Jlist (tt)

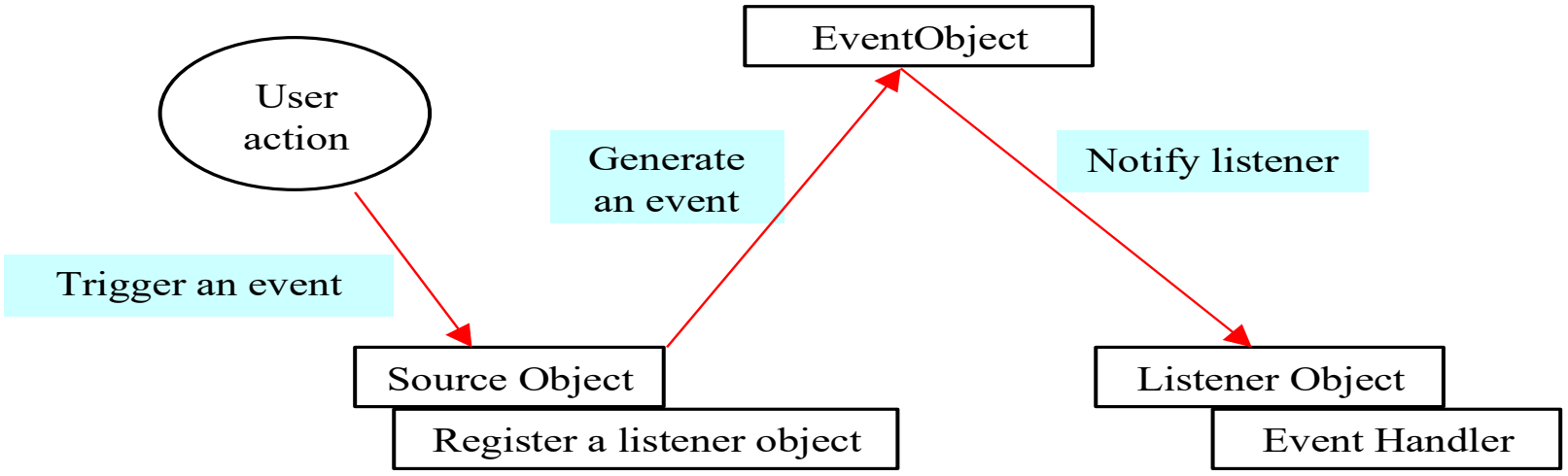

```
public class MyListSelectionListener implements ListSelectionListener {
    String[] imageNames;
    JLabel picture;

    MyListSelectionListener(JLabel pic, String[] images) {
        picture = pic;
        imageNames = images;
    }

    public void valueChanged(ListSelectionEvent e) {
        JList list = (JList)e.getSource();
        ImageIcon icon = new ImageIcon(imageNames[list.getSelectedIndex()] + ".gif");
        picture.setIcon(icon);

    }
}
```

Mô hình xử lý sự kiện

- Mô hình để xử lý tương tác của người dùng với thành phần giao diện
- Miêu tả cách chương trình có thể giao trả lời tương tác của người dùng

Mô hình xử lý sự kiện (tt)

- Có 3 yếu tố quan trọng trong mô hình xử lý sự kiện:
  - Nguồn phát sinh sự kiện (event source)
  - Sự kiện (event object)
  - Bộ lắng nghe sự kiện (event listener)



Mô hình xử lý sự kiện (tt)

- Nguồn phát sinh sự kiện (event source):
  - Thành phần giao diện tạo ra sự kiện
  - VD: Button, mouse, keyboard
- Sự kiện (event object)
  - Tạo ra khi sự kiện xảy ra
  - Chứa tất cả thông tin về sự kiện mà nó xảy ra:
    - Loại sự kiện
    - Nguồn sự kiện
- Bộ lắng nghe sự kiện (event listener)
  - Nhận những sự kiện và xử lý
  - Ví dụ:
    - Hiển thị thông tin cho người dùng
    - Tác vụ tính toán

Mô hình xử lý sự kiện (tt)

- Một Listener được đăng ký và chờ cho tới khi sự kiện xảy ra
- Khi sự kiện xảy ra
  - Một đối tượng sự kiện được tạo ra
  - Đối tượng sự kiện được kích hoạt bằng nguồn đã đăng ký listener
- Mỗi lần listener nhận đối tượng từ sự kiện nguồn
  - Giải mã những thông điệp
  - Xử lý sự kiện mà nó xuất hiện

Xử Lý Sự Kiện

- Khai báo lớp xử lý sự kiện

```
public class MyClass implements<Event>Listener
```

- Cài đặt các phương thức trong listener interface.
- Ví dụ: ActionListener

```
public void actionPerformed(ActionEvent e) {
...//code that reacts to the action... }
```

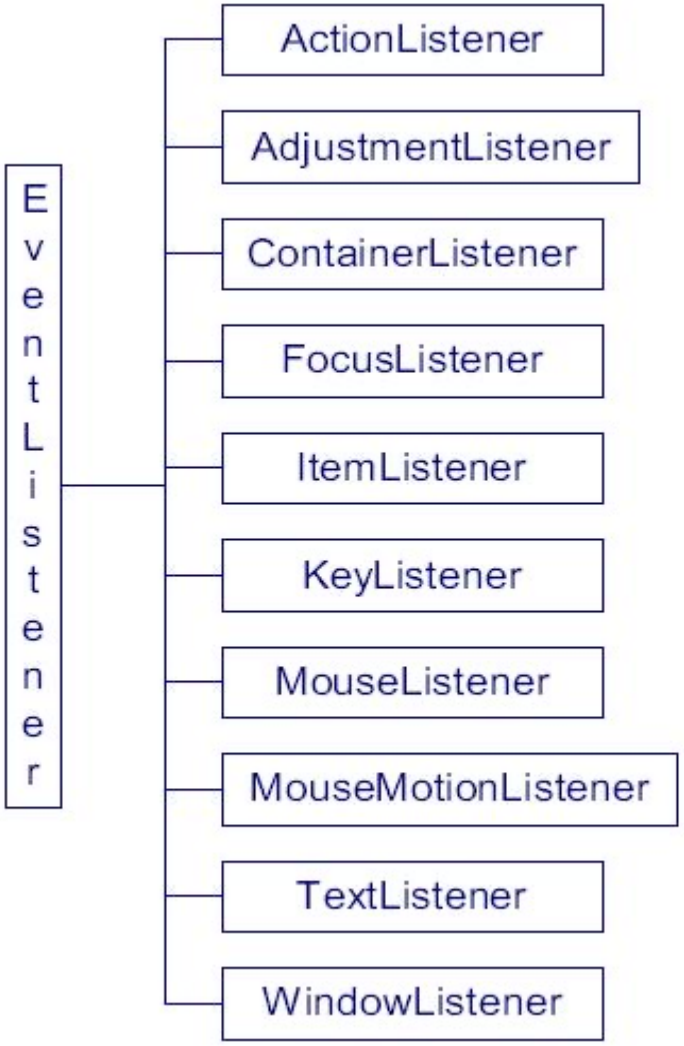
- Gắn bộ xử lý vào

```
componentsomeComponent.add<Event>Listener(
instanceOfMyClass);
```

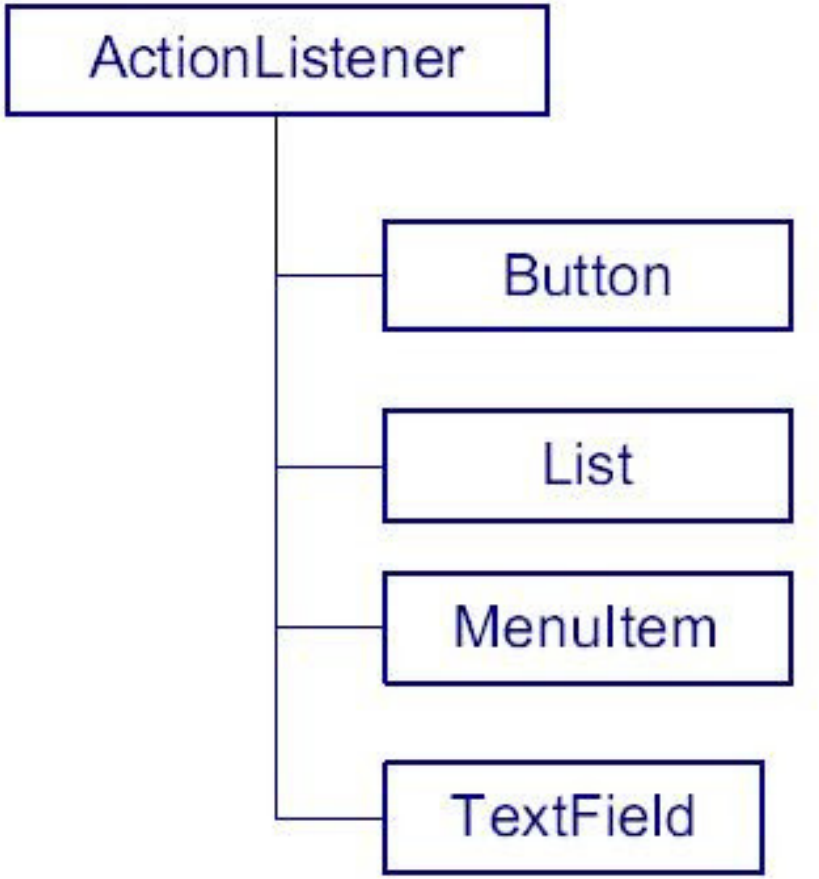
Hành động, sự kiện, lắng nghe

Đối tượng	Sự kiện	Bộ lắng nghe	Hàm
Window, Frame, …	WindowEvent	WindowListener	
Button, MenuItem, …	ActionEvent	ActionListener	actionPerformed(ActionEvent e)
TextComponent, …	TextEvent	TextListener	textValueChanged(TextEvent e)
List, …	ActionEvent	ActionListener	actionPerformed(ActionEvent e)
Checkbox, Radiobutton	ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
	ComponentEvent	ComponentListener	
	MouseEvent	MouseListener	
		MouseMotionListener	
	KeyEvent	KeyListener	

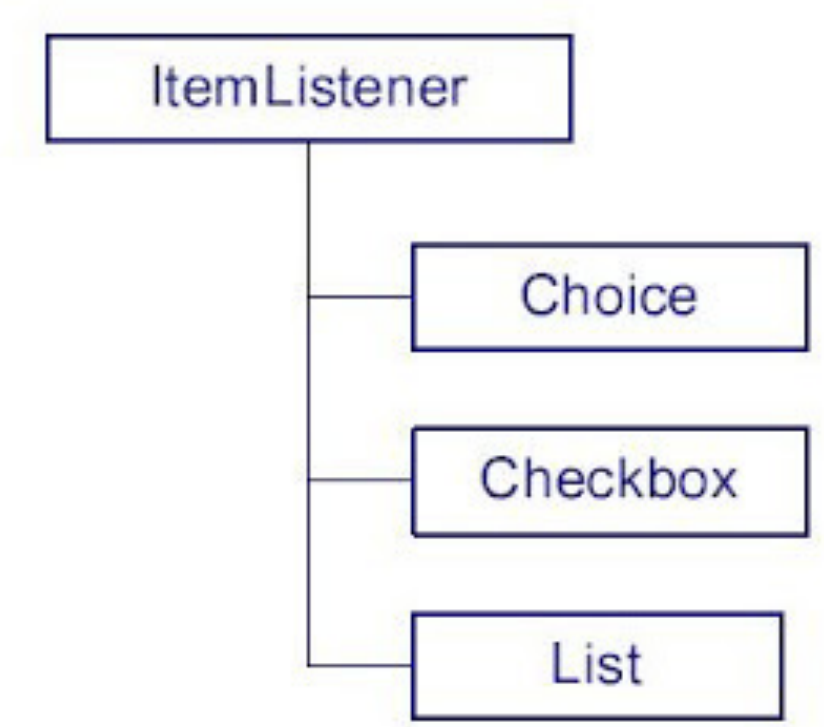
Event Listener



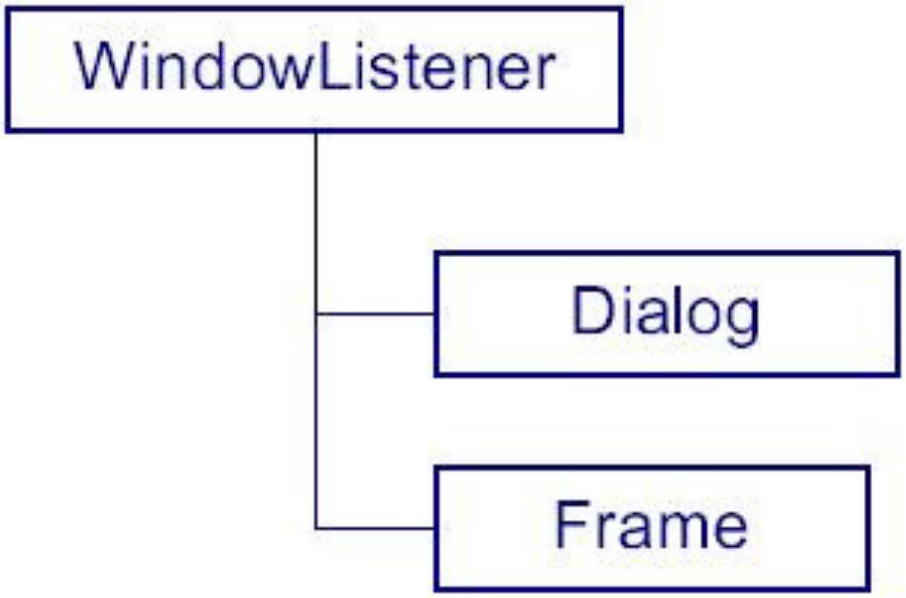
ActionListener



ItemListener



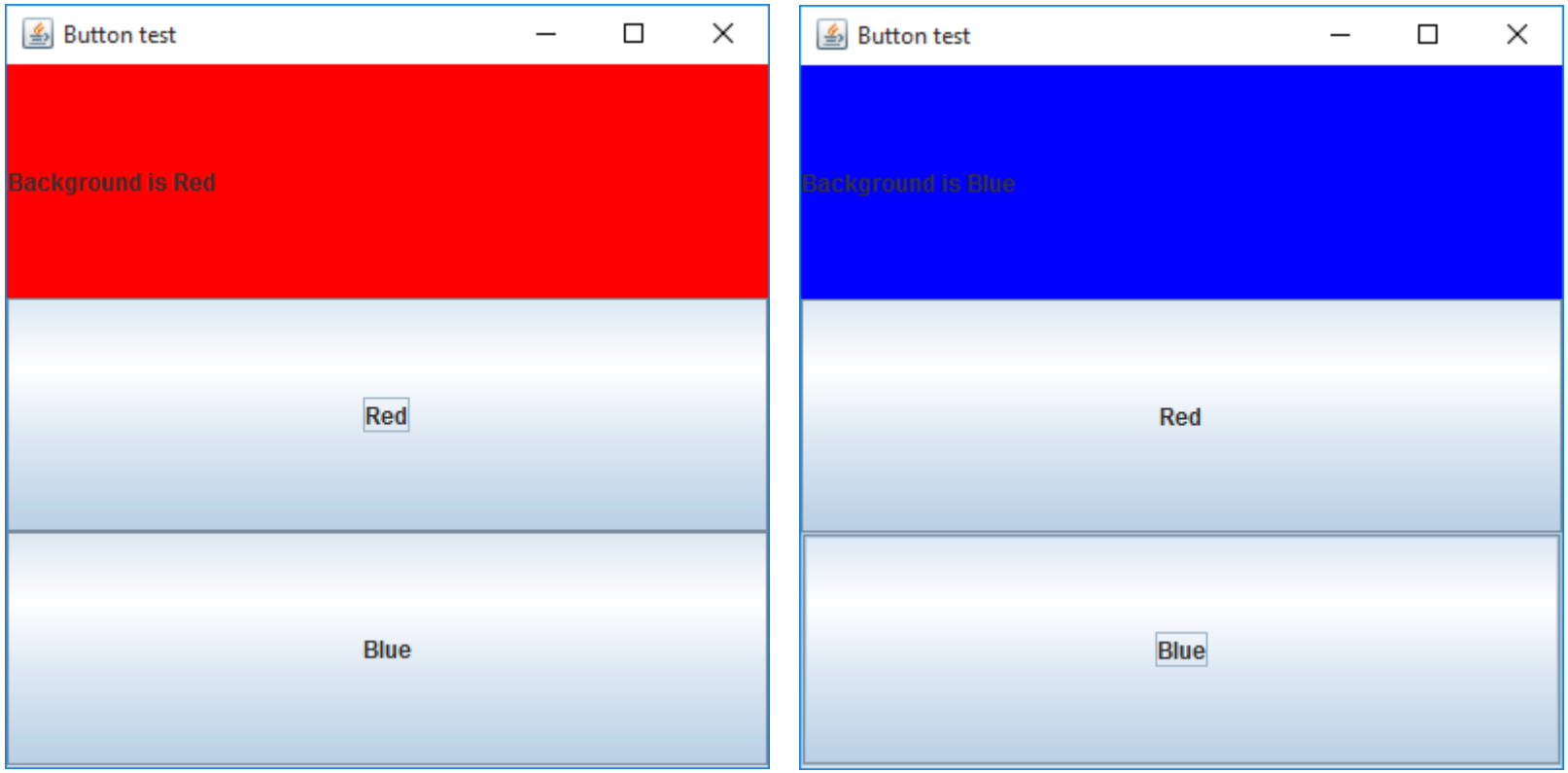
WindowListener



Đáp ứng các sự kiện JButton


```
private JButton createJButton(String title){
        JButton button = new JButton(title);
        button.addActionListener(new MyAction());
        return button;
}
private class MyAction implements ActionListener{
    private void changeBackgroundJLabel(Color bgcolor, String
nameBgcolor) {
        lb.setBackground(bgcolor);
        lb.setText("Background is " + nameBgcolor);
        }
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btnRed){
           changeBackgroundJLabel(Color.red, "Red");}
        if (e.getSource() == btnBlue){
           changeBackgroundJLabel(Color.blue, "Blue");}
      }
}
```

Đáp ứng các sự kiện Jbutton (tt)



Mouse Event Handling

- Event-listener của mouse events
  - MouseListener
  - MouseMotionListener

MouseListener and MouseMotionListener interface methods

MouseListener and MouseMotionListener interface methods	
Methods of interface **MouseListener**	
`public void mousePressed( MouseEvent event )`	Called when a mouse button is pressed with the mouse cursor on a component.
`public void mouseClicked( MouseEvent event )`	Called when a mouse button is pressed and released on a component without moving the mouse cursor.
`public void mouseReleased( MouseEvent event )`	Called when a mouse button is released after being pressed. This event is always preceded by a **mousePressed** event.
`public void mouseEntered( MouseEvent event )`	Called when the mouse cursor enters the bounds of a component.
`public void mouseExited( MouseEvent event )`	Called when the mouse cursor leaves the bounds of a component.
Methods of interface **MouseMotionListener**	
`public void mouseDragged( MouseEvent event )`	Called when the mouse button is pressed with the mouse cursor on a component and the mouse is moved. This event is always preceded by a call to **mousePressed**.
`public void mouseMoved( MouseEvent event )`	Called when the mouse is moved with the mouse cursor on a component.

Fig. 12.16 `MouseListener` and `MouseMotionListener` interface methods.

Lines 25-26

Line 35

```
1  // Fig. 12.17: MouseTracker.java
2  // Demonstrating mouse events.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class MouseTracker extends JFrame
12    implements MouseListener, MouseMotionListener {
13
14    private JLabel statusBar;
15
16    // set up GUI and register mouse event handlers
17    public MouseTracker()
18    {
19       super( "Demonstrating Mouse Events" );
20
21       statusBar = new JLabel();
22       getContentPane().add( statusBar, BorderLayout.SOUTH );
23
24       // application listens to its own mouse events
25       addMouseListener( this );
26       addMouseMotionListener( this );
27
28       setSize( 275, 100 );
29       setVisible( true );
30    }
31
32    // MouseListener event handlers
33
34    // handle event when mouse released immediately after press
35    public void mouseClicked( MouseEvent event )
```

Register **`JFrame`** to receive mouse events

Invoked when user presses and releases mouse button

```
36    {
37       statusBar.setText( "Clicked at [" + event.getX() +
38          ", " + event.getY() + "]" );
39    }
40
41    // handle event when mouse pressed
42    public void mousePressed( MouseEvent event )
43    {
44       statusBar.setText( "Pressed at [" + event.getX() +
45          ", " + event.getY() + "]" );
46    }
47
48    // handle event when mouse released after dragging
49    public void mouseReleased( MouseEvent event )
50    {
51       statusBar.setText( "Released at [" + event.getX() +
52          ", " + event.getY() + "]" );
53    }
54
55    // handle event when mouse enters area
56    public void mouseEntered( MouseEvent event )
57    {
58       JOptionPane.showMessageDialog( null, "Mouse in window" );
59    }
60
61    // handle event when mouse exits area
62    public void mouseExited( MouseEvent event )
63    {
64       statusBar.setText( "Mouse outside window" );
65    }
66
67    // MouseMotionListener event handlers
68
69    // handle event when user drags mouse with button pressed
70    public void mouseDragged( MouseEvent event )
```

MouseTracke Java

Invoked when user presses mouse button

Line 49

Line 56

Invoked when user releases mouse button after dragging mouse

Line 70

Invoked when mouse cursor enters **JFrame**

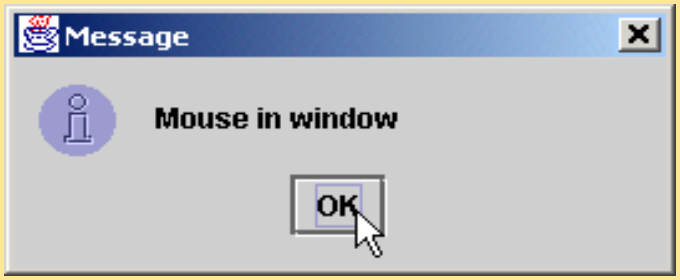
Invoked when mouse cursor exits **JFrame**

Invoked when user drags mouse cursor

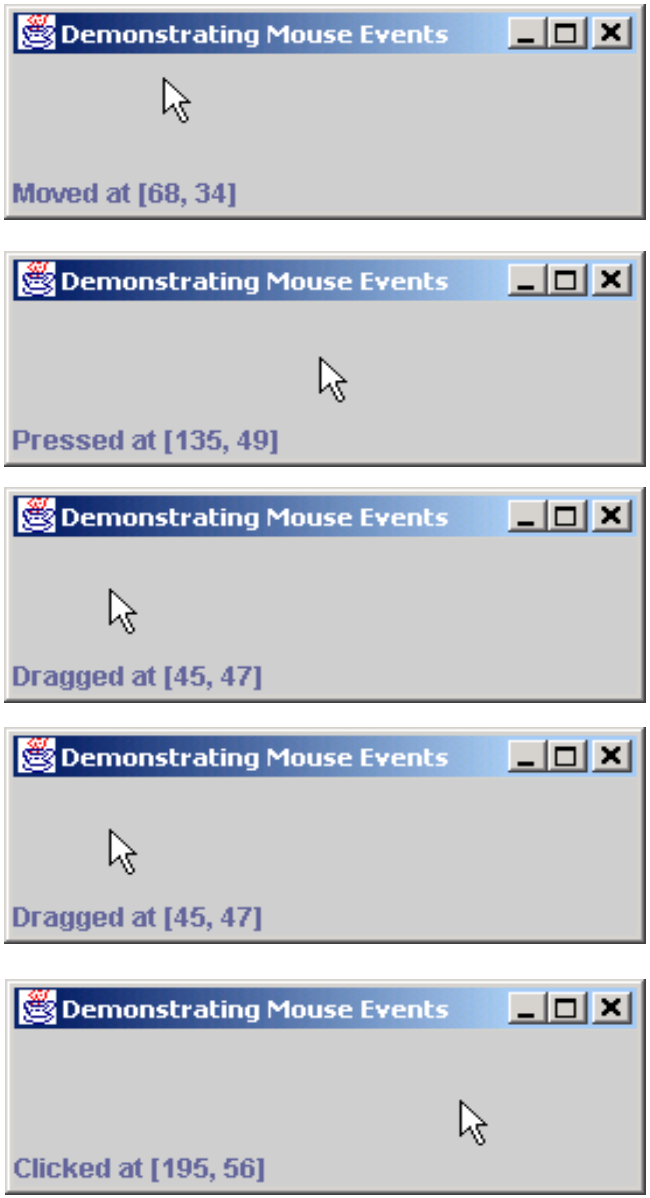
```
71    {
72       statusBar.setText( "Dragged at [" + event.getX() +
73          ", " + event.getY() + "]" );
74    }
75
76    // handle event when user moves mouse
77    public void mouseMoved( MouseEvent event )
78    {
79       statusBar.setText( "Moved at [" + event.getX() +
80          ", " + event.getY() + "]" );
81    }
82
83    // execute application
84    public static void main( String args[] )
85    {
86       MouseTracker application = new MouseTracker();
87
88       application.setDefaultCloseOperation(
89          JFrame.EXIT_ON_CLOSE );
90    }
91
92 }  // end class MouseTracker
```

MouseTracke Java

Invoked when user moves mouse cursor



MouseTracke Java



Line 21

```
1  // Fig. 12.20: MouseDetails.java
2  // Demonstrating mouse clicks and
3  // distinguishing between mouse buttons.
4
5  // Java core packages
6  import java.awt.*;
7  import java.awt.event.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class MouseDetails extends JFrame {
13    private int xPos, yPos;
14
15    // set title bar String, register mouse listener and size
16    // and show window
17    public MouseDetails()
18    {
19       super( "Mouse clicks and buttons" );
20
21       addMouseListener( new MouseClickHandler() );
22
23       setSize( 350, 150 );
24       setVisible( true );
25    }
26
27    // draw String at location where mouse was clicked
28    public void paint( Graphics g )
29    {
30       // call superclass's paint method
31       super.paint( g );
32
33       g.drawString( "Clicked @ [" + xPos + ", " + yPos + "]",
34          xPos, yPos );
35    }
```

Register mouse listener

```
36
37    // execute application
38    public static void main( String args[] )
39    {
40       MouseDetails application = new MouseDetails();
41
42       application.setDefaultCloseOperation(
43          JFrame.EXIT_ON_CLOSE );
44    }
45
46    // inner class to handle mouse events
47    private class MouseClickHandler extends MouseAdapter {
48
49       // handle mouse click event and determine which mouse
50       // button was pressed
51       public void mouseClicked( MouseEvent event )
52       {
53          xPos = event.getX();
54          yPos = event.getY();
55
56          String title =
57             "Clicked " + event.getClickCount() + " time(s)";
58
59          // right mouse button
60          if ( event.isMetaDown() )
61             title += " with right mouse button";
62
63          // middle mouse button
64          else if ( event.isAltDown() )
65             title += " with center mouse button";
66
67          // left mouse button
68          else
69             title += " with left mouse button";
```

MouseDetail

Line 51

Lines 53-54

Invoke method **mouseClicked** when user clicks mouse

Lines 60-61

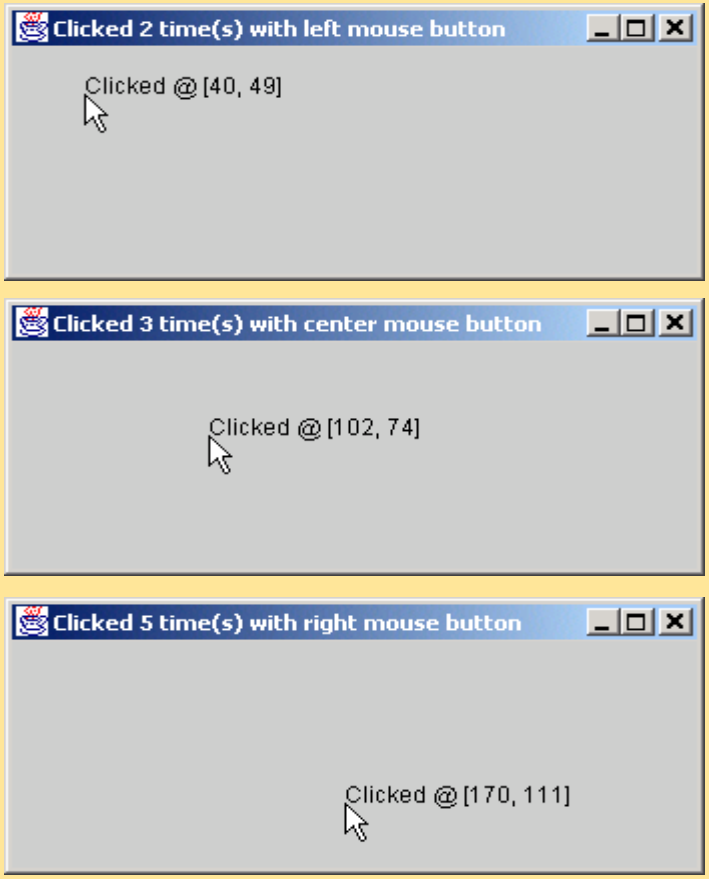
Store mouse-cursor coordinates where mouse was clicked

Determine number of times user has clicked mouse

Determine if user clicked right mouse button

Determine if user clicked middle mouse button

```
70
71          setTitle( title );  // set title bar of window
72          repaint();
73       }
74
75    }  // end private inner class MouseClickHandler
76
77 }  // end class MouseDetails
```



MouseDetail

Keyboard Event Handling

- Interface **KeyListener**
  - Dùng để xử lý key events
    - Phát sinh khi 1 phím được nhấn và thả ra.
    - **KeyEvent**
      - Chứa *virtual key code* đại diện cho các phím

```
1  // Fig. 12.22: KeyDemo.java
2  // Demonstrating keystroke events.
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class KeyDemo extends JFrame implements KeyListener {
12    private String line1 = "", line2 = "";
13    private String line3 = "";
14    private JTextArea textArea;
15
16    // set up GUI
17    public KeyDemo()
18    {
19       super( "Demonstrating Keystroke Events" );
20
21       // set up JTextArea
22       textArea = new JTextArea( 10, 15 );
23       textArea.setText( "Press any key on the keyboard..." );
24       textArea.setEnabled( false );
25       getContentPane().add( textArea );
26
27       // allow frame to process Key events
28       addKeyListener( this );
29
30       setSize( 350, 100 );
31       setVisible( true );
32    }
33
34    // handle press of any key
```

Line 28

Line 35

Register **JFrame** for key events

```
35    public void keyPressed( KeyEvent event )
36    {
37       line1 = "Key pressed: " +
38          event.getKeyText( event.getKeyCode() );
39       setLines2and3( event );
40    }
41
42    // handle release of any key
43    public void keyReleased( KeyEvent event )
44    {
45       line1 = "Key released: " +
46          event.getKeyText( event.getKeyCode() );
47       setLines2and3( event );
48    }
49
50    // handle press of an action key
51    public void keyTyped( KeyEvent event )
52    {
53       line1 = "Key typed: " + event.getKeyChar();
54       setLines2and3( event );
55    }
56
57    // set second and third lines of output
58    private void setLines2and3( KeyEvent event )
59    {
60       line2 = "This key is " +
61          ( event.isActionKey() ? "" : "not " ) +
62          "an action key";
63
64       String temp =
65          event.getKeyModifiersText( event.getModifiers() );
66
67       line3 = "Modifier keys pressed: " +
68          ( temp.equals( "" ) ? "none" : temp );
69
```

Called when user presses key

Called when user releases key

Return virtual key code

Called when user types key

Determine if *modifier keys* (e.g., *Alt*, *Ctrl*, *Meta* and *Shift*) were used

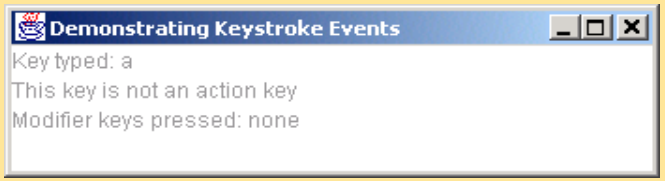
KeyDemo.ja Java

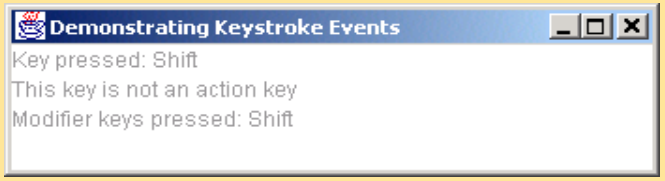
Line 43

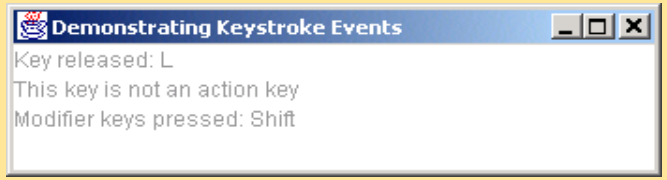
Lines 64-65

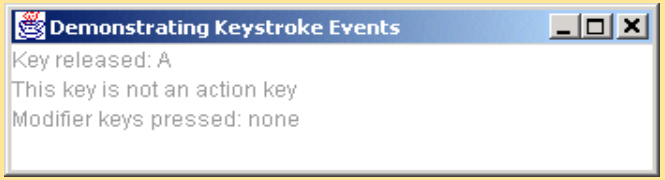
KeyDemo.ja

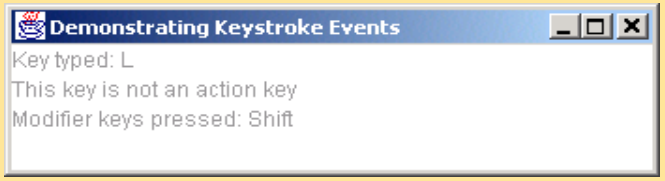

```
70          textArea.setText(
71             line1 + "\n" + line2 + "\n" + line3 + "\n" );
72       }
73
74       // execute application
75       public static void main( String args[] )
76       {
77          KeyDemo application = new KeyDemo();
78
79          application.setDefaultCloseOperation(
80             JFrame.EXIT_ON_CLOSE );
81       }
82
83    }  // end class KeyDemo
```

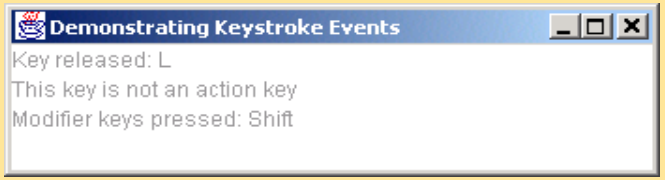


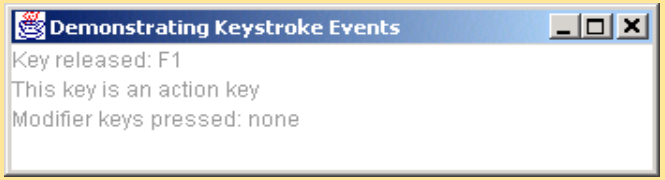












WindowListener

- Chứa những xử lý cho sự kiện cửa sổ được mở
  - `public void windowOpened(WindowEvent e)`
- Chứa những xử lý cho sự kiện cửa sổ chuẩn bị đóng
  - `public void windowClosing(WindowEvent e)`
- Chứa những xử lý cho sự kiện cửa sổ sau khi đóng
  - `public void windowClosed(WindowEvent e)`
- Chứa những xử lý cho sự kiện cửa sổ kích hoạt
  - `public void windowActivated(WindowEvent e)`
- Chứa những xử lý cho sự kiện ẩn cửa sổ
  - `public void windowDeactivated(WindowEvent e)`
- Chứa những xử lý cho sự kiện làm thu nhỏ cửa sổ
  - `public void windowIconified(WindowEvent e)`
- Chứa những xử lý cho sự kiện làm phóng to cửa sổ
  - `public void windowDeiconified(WindowEvent e)`

Class Adapter

- Tại sao phải sử dụng những class Adapter?
  - Thực thi tất cả các phương thức của interface sẽ mất nhiều thời gian.
  - Chỉ cần quan tâm thực thi một vài phương thức
- Những class Adapter
  - Xây dựng bằng Java
  - Thực thi tất cả phương thức của listener
  - Những thực thi của các phương thức là rỗng.
- Một số class Adapter trong GUI
  - WindowAdapter, MouseAdapter, KeyAdapter….

```
private class MouseClickHandler extends MouseAdapter
```

Inner Classes

- Class được khai báo trong class khác
- Tại sao phải sử dụng inner class?
  - Giúp đơn giản chương trình
  - Đặc biệt là trong xử lý sự kiện.

Giấu tên những class inner

- Không đặt tên những class inner
- Tại sao lại giấu tên những class inner class?
  - Làm cho code đơn giản hơn
  - Đặc biệt là trong xử lý sự kiện.
- Ví dụ:

```
this.addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent e) {
            System.out.println("window closing");
            dispose();
            System.exit(1);
});
```

Lambda Expresions

- Biểu thức Lambda Expresions: **(args) -> { body code }**
- Ví dụ:

```
button.addActionListener(new ActionListener(){
    @Override
    public void actionPerformed(ActionEvent e){
        System.out.println("hello");
    }
});
```

- Lambda Expresions:

```
button.addActionListener(e -> System.out.println("hello"));
```

Lambda Expresions (tt)

➢ **Ví dụ 2:**

```
button.addActionListener(new ActionListener(){
      @Override
      public void actionPerformed(ActionEvent e) {
          if (e.getSource() == btnRed){
             lb.setBackground(Color.red);
             lb.setText("Background is " + "Red");
          }
       }
});
```

Lambda Expresions (tt)

- **Lambda Expressions**

```
button.addActionListener(e -> {
    if (e.getSource() == btnRed){
        lb.setBackground(Color.red);
        lb.setText("Background is " + "Red");
    }
});
```

Pluggable Look And Feel

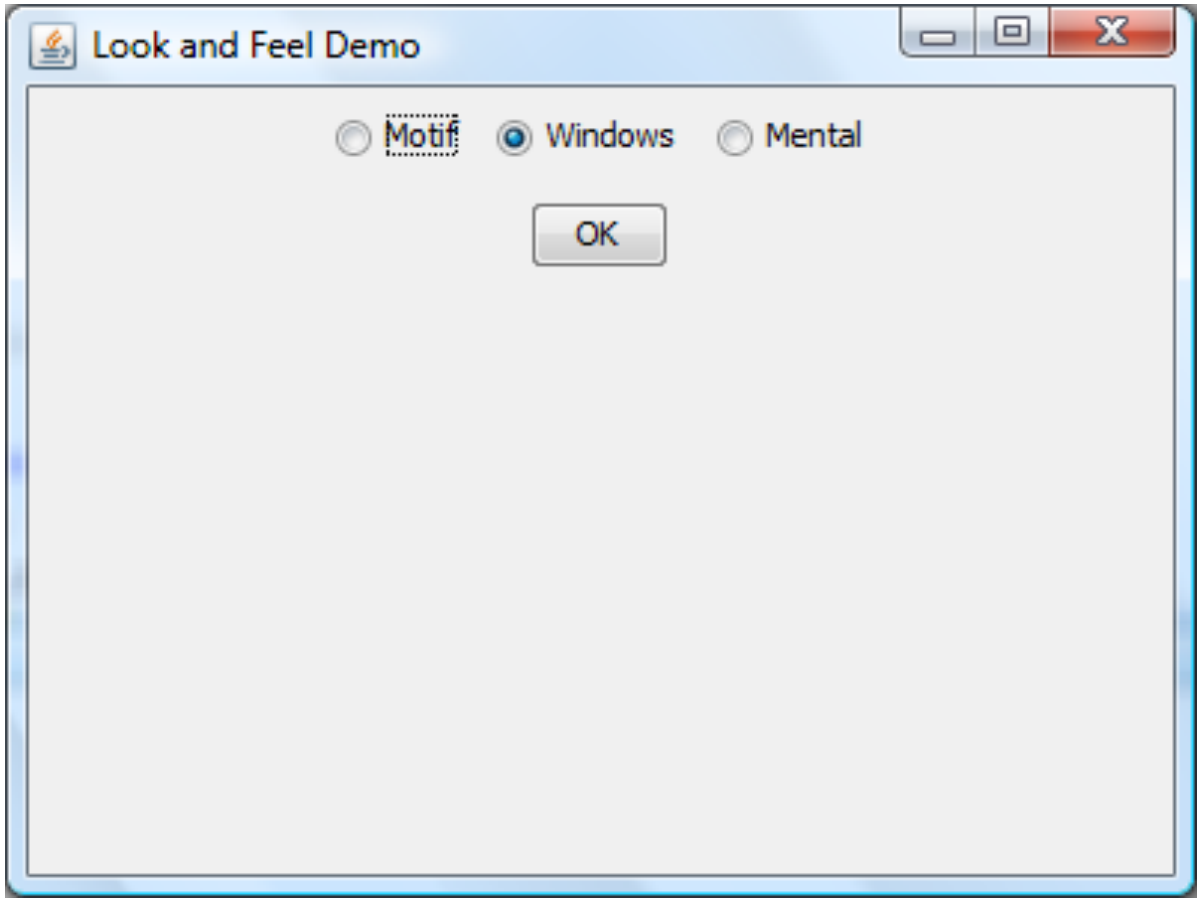
- Swing hỗ trợ pluggable look-and-feel.
- Swing hỗ trợ 3 loại:
  - Motif - "com.sun.java.swing.plaf.motif.MotifLookAndFeel"
  - Windows - "com.sun.java.swing.plaf.windows.WindowsLookAndFeel"
  - Mental (Java platform) - "javax.swing.plaf.metal.MetalLookAndFeel"

Thay Đổi Look and Feel

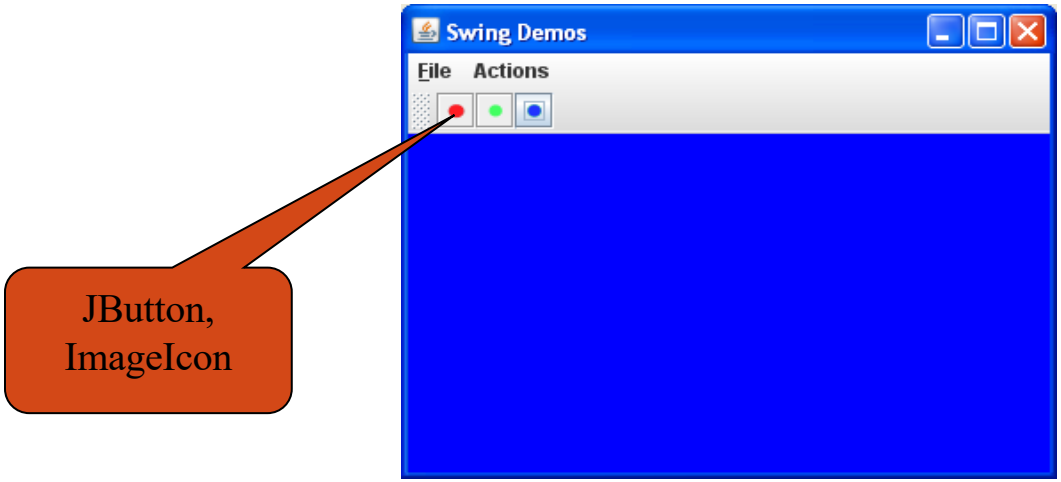
- UIManager.setLookAndFeel(String className) throws UnsupportedLookAndFeelException
- SwingUtilities.updateComponentTreeUI(Component c)

```
try {
    UIManager.setLookAndFeel(
"com.sun.java.swing.plaf.Motif.MotifLookAndFeel" );
}
catch( UnsupportedLookAndFeelException e ) {
}
```

Look And Feel Demo



JToolBar, Icon, ImageIcon




```
// Create toolbar
JToolBar toolbar = new JToolBar();
MainToolBarListener actionListener = new MainToolBarListener(this);
Icon red = new ImageIcon("images/red.png");
redIcon = new JButton(red);
redIcon.addActionListener(actionListener);
toolbar.add(redIcon);
```

JToolBar, Icon, ImageIcon

- Đặt tooltip cho Icon trên thanh toolbar

```
// Create toolbar
JToolBar toolbar = new JToolBar();
MainToolBarListener actionListener = new MainToolBarListener(this);
Icon red = new ImageIcon("images/red.png");
redIcon = new JButton(red);
redIcon.setToolTipText("Set red color for the background");
redIcon.addActionListener(actionListener);
toolbar.add(redIcon);
```

Graphics Context và Object

- **Graphics context**
  - Hỗ trợ thao tác vẽ trên màn hình
  - **Đối tượng Graphics** quản lý graphics context
    - Điều khiển cách vẽ
    - Cung cấp các phương thức để vẽ, chọn font, màu….
  - **Graphics** là 1 lớp trừu tượng!
- **Class Component**
  - Là lớp cơ sở của các thành phần trong **java.awt và javax.swing**
  - Phương thức **paint(Graphics g)**

Lớp Color

- Hỗ trợ các thao tác trên màu sắc.
- Color(int red, int green, int blue)
- Lớp Graphics:
  - void setColor(Color c): chọn màu dùng để vẽ
  - Color getColor(): lấy về màu đang chọn

Lớp Font

- Font(String name, int style, int size)
  - Name: tên font có trong hệ thống
  - Style: FONT.PLAIN, FONT.ITALIC, FONT.BOLD
  - Size: kích thước đơn vị point (1/72 inch)
- Lớp Graphics
  - Font getFont()
  - void setFont(Font f)

Lớp Graphics

- Draw string at **x, y**
  - `drawString(s, x, y)`
- Draw line from **x1, y1** to **x2, y2**
  - `drawLine(x1, y1, x2, y2)`
- Draws rectangle with upper left corner **x1, y1**
  - `drawRect(x1, y1, width, height)`
- As above, except fills rectangle with current color
  - `fillRect(x1, y1, width, height)`
- As above, except fills rectangle with background color
  - `clearRect(x1, y1, width, height)`

Lớp Graphics (tt)

- **draw3DRect(x1, y1, width, height, isRaised)**
  - Draws **3D** rectangle, raised if **isRaised is true,** else lowered.
- **fill3DRect**
  - As previous, but fills rectangle with current color
- **drawRoundRect(x, y, width, height, arcWidth, arcHeight)**
  - Draws rectangle with rounded corners. See diagram next slide.
- **fillRoundRect(x, y, width, height, arcWidth, arcHeight)**
- **drawOvalx, y, width, height)**
  - Draws oval in bounding rectangle (see diagram)
  - Touches rectangle at midpoint of each side
- **fillOval(x, y, width, height)**

Ví Dụ


```
public class LinesRectsOvals extends JFrame {
  private String s = "Using drawString!";

  public LinesRectsOvals()  {
    super( "Drawing lines, rectangles and ovals" );
    setSize( 400, 165 );
    setVisible(true);
  }

  public void paint( Graphics g )  {
    g.setColor( Color.red );
    g.drawLine( 5, 30, 350, 30 );
    g.setColor( Color.blue );
    g.drawRect( 5, 40, 90, 55 );
    g.fillRect( 100, 40, 90, 55 );
    g.setColor( Color.cyan );
    g.fillRoundRect( 195, 40, 90, 55, 50, 50 );
    g.drawRoundRect( 290, 40, 90, 55, 20, 20 );
```

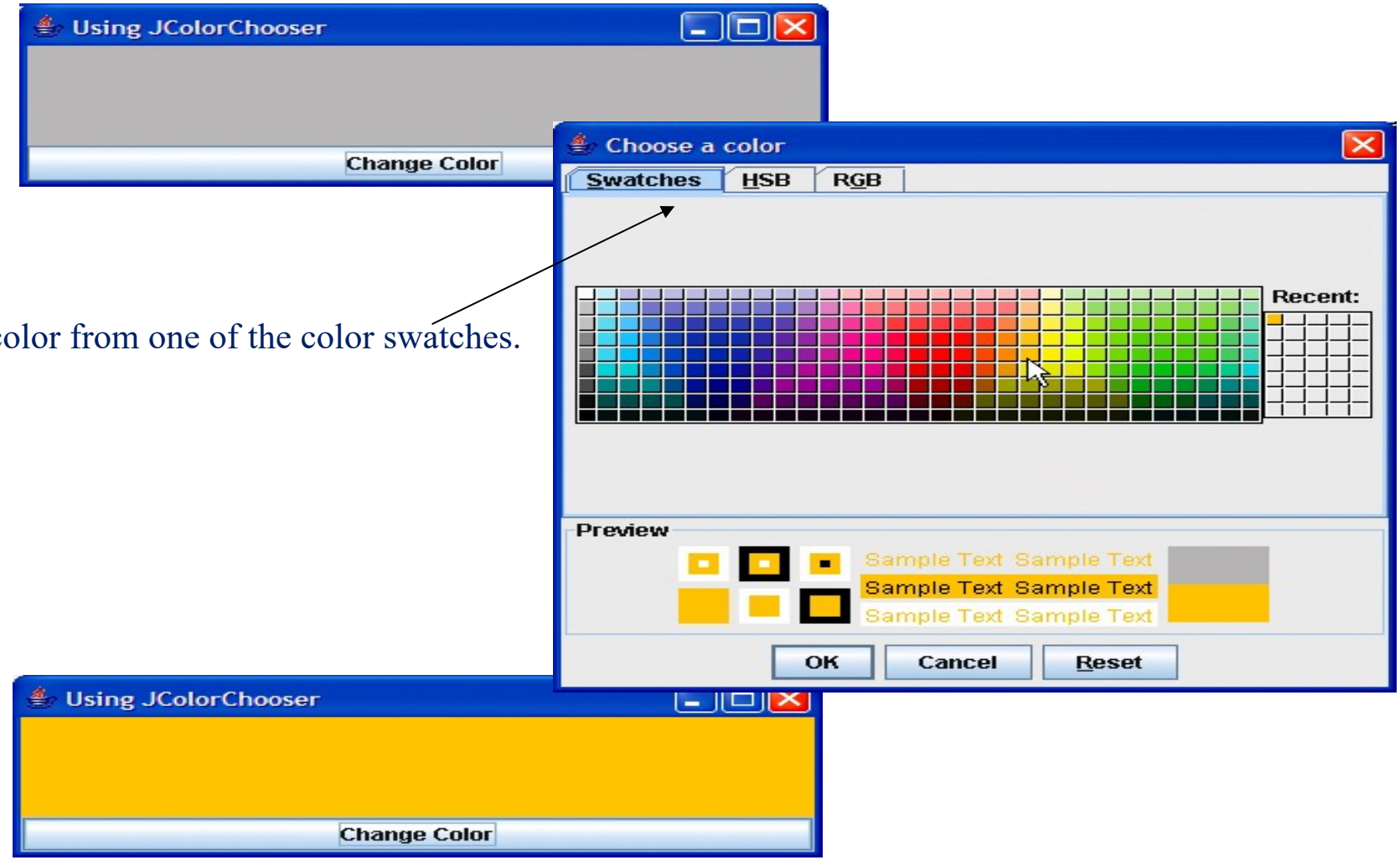
```
    g.setColor( Color.yellow );
    g.draw3DRect( 5, 100, 90, 55, true );
    g.fill3DRect( 100, 100, 90, 55, false );

    g.setColor( Color.magenta );
    g.drawOval( 195, 100, 90, 55 );
    g.fillOval( 290, 100, 90, 55 );
  }

  public static void main( String args[] ) {
    LinesRectsOvals app = new LinesRectsOval
  }
}
```

JColorChooser

- Mở hộp thoại chọn màu
- Trả về đối tượng màu đã chọn

```
static Color showDialog(Component parent, String title,
Color initColor)
```

JColorChooser Demo



Select a color from one of the color swatches.

Q & A