

Bài 11

# Đồ họa và xử lý sự kiện

# Nội dung

1. Giao diện đồ họa người sử dụng
2. AWT
3. Xử lý sự kiện
4. Swing



# 1

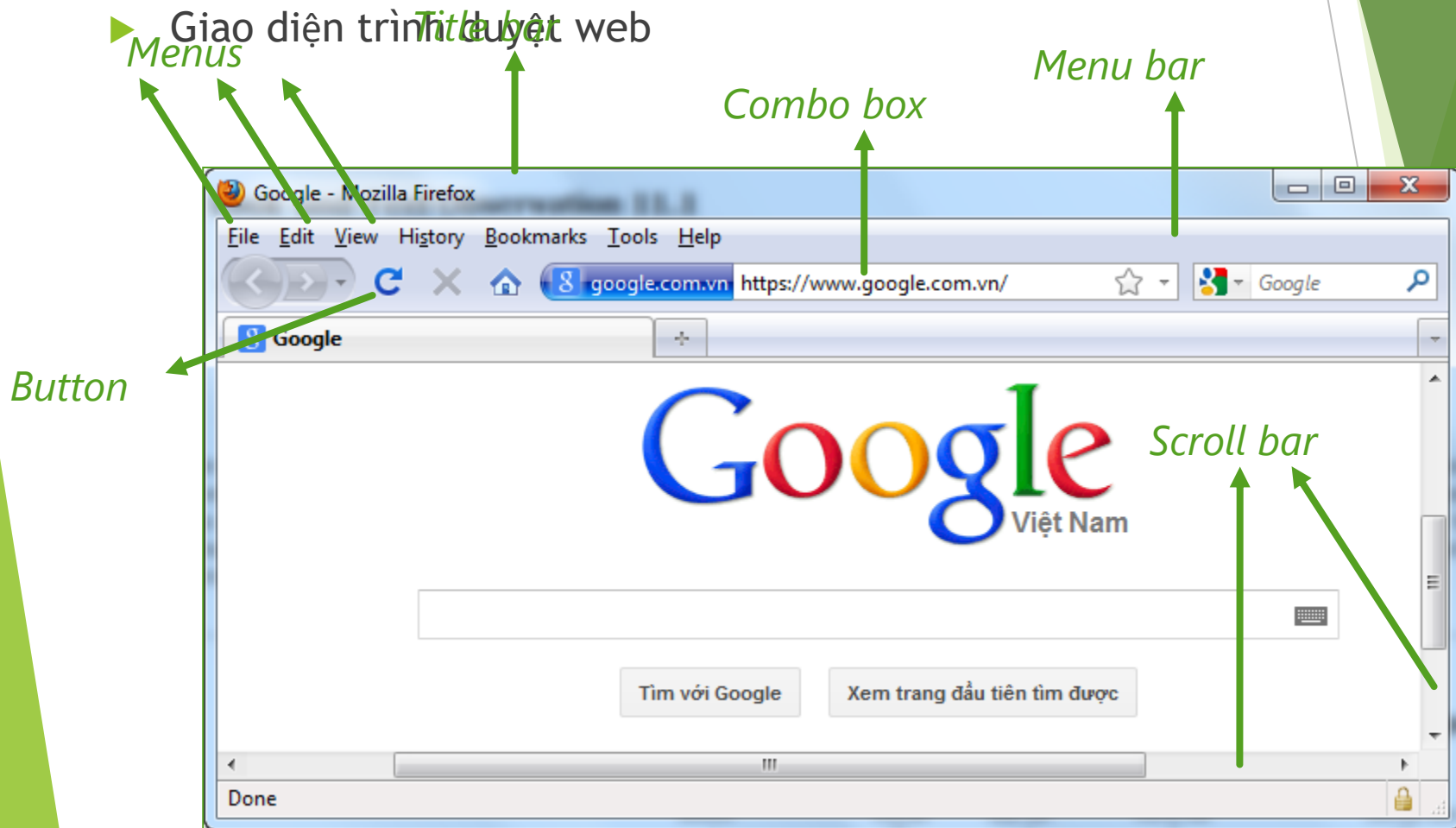
## Giao diện đồ họa người sử dụng

Graphical User Interface (GUI)

# Giao diện đồ họa người dùng

- ▶ Giao diện đồ họa người sử dụng (Graphical user interface - GUI)
- ▶ Giúp tạo ra các ứng dụng có giao diện đồ họa với nhiều các điều khiển như: Button, Textbox, Label, Checkbox, List, Tree...

# Ví dụ



# Lập trình GUI trong Java

- ▶ Java cung cấp hai thư viện đồ họa
  - ▶ AWT
  - ▶ Swing
- ▶ AWT
  - ▶ Được cung cấp trong Java 1.0
- ▶ Swing
  - ▶ Nâng cấp các thành phần giao diện của AWT
  - ▶ Được tích hợp trong Java 1.2

# Lập trình GUI trong Java

- ▶ Một số loại giao diện khác
  - ▶ Eclipse's Standard Widget Toolkit (SWT)
  - ▶ Google Web Toolkit (GWT)
  - ▶ Các thư viện đồ họa như Java bindings for OpenGL (JOGL) hay Java3D.

# 2

**AWT**

Advanced Widget Toolkit



# AWT

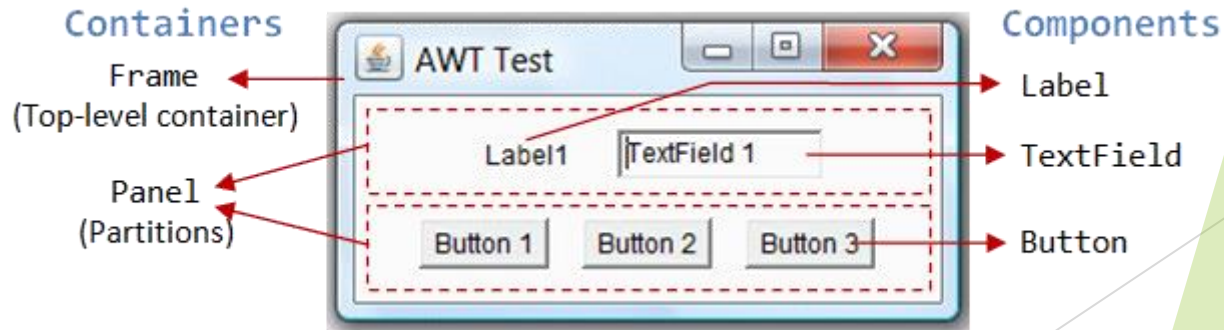
- ▶ AWT - Advanced Widget Toolkit
- ▶ Các lớp AWT được Java cung cấp trong 12 gói
  - ▶ Các gói `java.awt` và `java.awt.event` được sử dụng chủ yếu
  - ▶ Độc lập nền và độc lập thiết bị
- ▶ Các lớp cơ bản
  - ▶ Các thành phần GUI (vd. `Button`, `TextField`, `Label`...)
  - ▶ Các lớp GUI Container (vd. `Frame`, `Panel`, `Dialog`, `ScrollPane`...)
  - ▶ Layout managers (vd. `FlowLayout`, `BorderLayout`, `GridLayout`...)
  - ▶ Các lớp đồ họa (vd. `Graphics`, `Color`, `Font`...)

# AWT (tiếp)

- ▶ Các lớp xử lý sự kiện
  - ▶ Các lớp Event (vd. `ActionEvent`, `MouseEvent`, `KeyEvent` and `WindowEvent`...)
  - ▶ Các giao diện Event Listener (vd. `ActionListener`, `MouseListener`, `KeyListener`, `WindowListener`...)
  - ▶ Các lớp Event Listener Adapter (vd. `MouseAdapter`, `KeyAdapter`, `WindowAdapter`...)

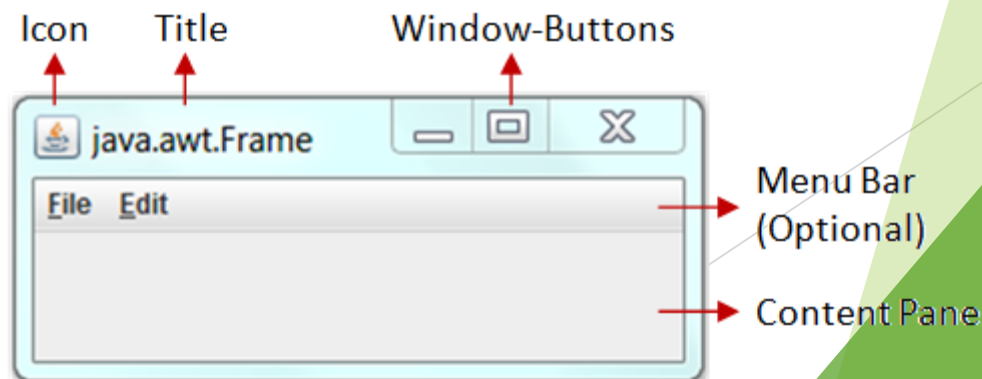
# Các thành phần của AWT

- ▶ 2 loại thành phần chính
  - ▶ *Component*: các thực thể GUI cơ bản (Button, Label, TextField.)
  - ▶ *Container* (Frame, Panel and Applet): Chứa các thực thể GUI. Một container cũng có thể chứa các container khác.



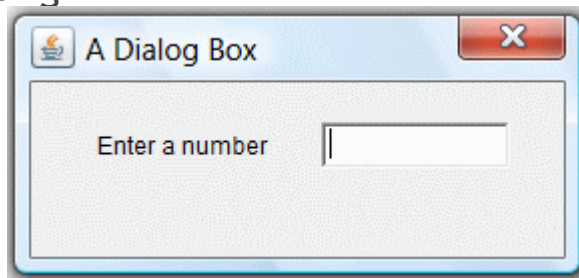
# Top-level AWT container

- ▶ **Top-level container:** Frame, Dialog và Applet.
- ▶ **Frame:** Cung cấp cửa sổ chính cho ứng dụng, chứa:
  - ▶ title bar (chứa biểu tượng, tiêu đề, các nút minimize, maximize & close)
  - ▶ menu bar
  - ▶ vùng hiển thị nội dung



# Top-level AWT container

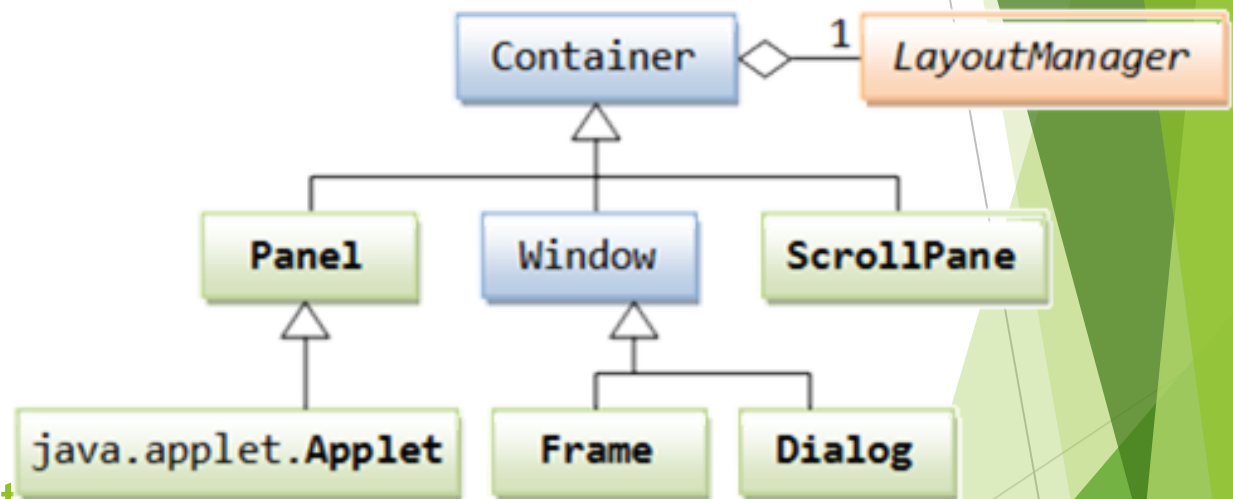
- ▶ Dialog: còn gọi là "pop-up window", chứa:
  - ▶ title-bar
  - ▶ vùng hiển thị nội dung



- ▶ Applet: Ứng dụng Java có thể chạy trên trình duyệt

# Secondary AWT container

- ▶ **Secondary Containers:** Panel, ScrollPane
  - ▶ Đặt bên trong top-level container hoặc các secondary container khác
- ▶ **Panel**
  - ▶ Một vùng hình chữ nhật nằm bên trong container
  - ▶ Sử dụng để áp dụng một *layout* cho các thành phần bên trong
- ▶ **ScrollPane:** tạo ra một vùng có thể trượt dọc hoặc trượt ngang các thành phần bên trong



## Cây phân cấp kế thừa AWT

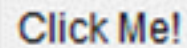
Cây phân cấp kế thừa của các lớp trong AWT

# Các thành phần của AWT

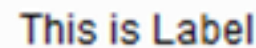
- Button, TextField, Label, Checkbox, CheckboxGroup (radio buttons), List, and Choice

A rectangular text input field with a thin border and the text "Enter your name here" inside.

TextField

A rectangular button with a 3D effect and the text "Click Me!" inside.

Button

A rectangular label with a thin border and the text "This is Label" inside.

Label

A choice component consisting of a dropdown menu showing "Red" and a list box with "Red", "Green" (highlighted), and "Blue".

Choice

Three checkboxes with labels "one", "two", and "three". The "one" checkbox is checked.

CheckBox

Three radio buttons with labels "Alpha", "Beta", and "Charlie". The "Alpha" radio button is selected.

CheckboxGroup

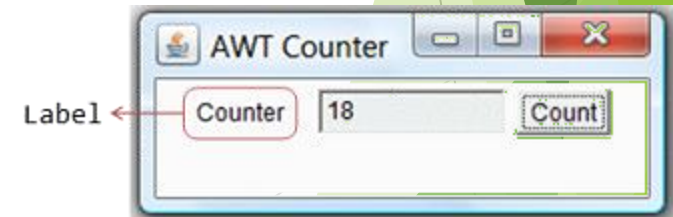
A list component showing a scrollable list of planet names: Mercury, Venus, Earth (highlighted), Mars, Jupiter, Saturn, Uranus, and Neptune.

List



# Label

- ▶ **java.awt.Label:** Hiển thị một nhãn văn bản
- ▶ Phương thức khởi tạo
  - ▶ `// Construct a Label with the given text String, of the text alignment`
  - ▶ `public Label(String strLabel, int alignment);`
  - ▶ `public Label(String strLabel); // Construct a Label with the given text`
  - ▶ `public Label(); // Construct an initially empty Label`
- ▶ Phương thức
  - ▶ `public String getText();`
  - ▶ `public void setText(String strLabel);`
  - ▶ `public int getAlignment();`
  - ▶ `public void setAlignment(int alignment);`



# Add component vào container

- ▶ Các bước để tạo một component và add vào container:

- ▶ Khai báo và khởi tạo thành phần đó
- ▶ Xác định container sẽ chứa thành phần này
  - ▶ Sử dụng phương thức `add`
  - ▶ VD: `aContainer.add(aComponent)`

- ▶ Ví dụ

```
Label lblInput;  
lblInput = new Label("Enter ID");  
this.add(lblInput);  
lblInput.setText("Enter password");  
lblInput.getText();
```

# Button

- ▶ **java.awt.Button**: Kích hoạt một sự kiện khi nhấp chuột

- ▶ Phương thức khởi tạo

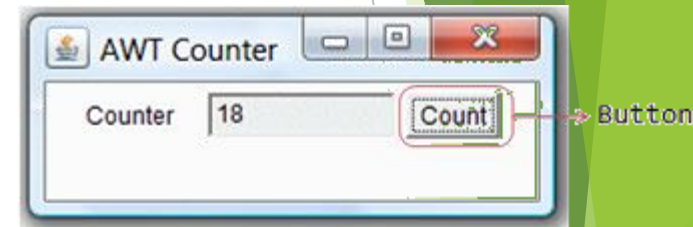
- ▶ `public Button(String buttonLabel);`
- ▶ `public Button(String buttonLabel);`

- ▶ Các phương thức

- ▶ `public String getLabel();`
- ▶ `public void setLabel(String buttonLabel);`
- ▶ `public void setEnable(boolean enable);`

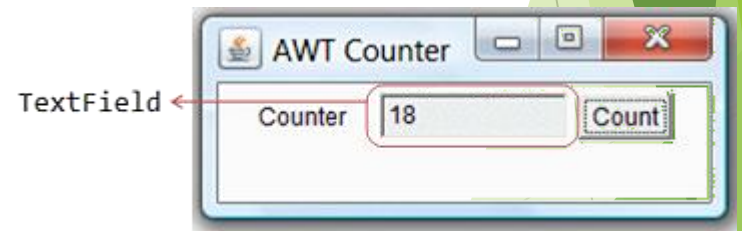
- ▶ Ví dụ

```
Button btnColor = new Button("Red");  
this.add(btnColor);  
...  
btnColor.setLabel("green");  
btnColor.getLabel();
```



# TextField

- ▶ **java.awt.TextField**: Ô văn bản để người dùng có thể nhập liệu trong một dòng (**TextArea**: nhiều dòng)
- ▶ Phương thức khởi tạo
  - ▶ `public TextField(String strInitialText, int columns);`
  - ▶ `public TextField(String strInitialText);`
  - ▶ `public TextField(int columns);`
- ▶ Các phương thức
  - ▶ `public String getText();`
  - ▶ `public void setText(String strText);`
  - ▶ `public void setEditable(boolean editable);`



# Quản lý bố cục

- ▶ Layout: Sắp xếp các thành phần trong container
- ▶ Các Layout manager trong AWT: (trong gói `java.awt`)
  - ▶ `FlowLayout`
  - ▶ `GridLayout`
  - ▶ `BorderLayout`
  - ▶ `GridBagLayout`
  - ▶ `BoxLayout`
  - ▶ `CardLayout`

# Thiết lập Layout Manager

- ▶ Gọi đến phương thức `setLayout()` của container

```
public void setLayout(LayoutManager mgr)
```

- ▶ Các bước để thiết lập Layout trong Container

- ▶ Khởi tạo đối tượng Layout tương ứng, vd. `new FlowLayout()`
- ▶ Gọi đến phương thức `setLayout` với tham số là đối tượng vừa tạo
- ▶ Gọi phương thức `add` của container theo thứ tự tương ứng

- ▶ Ví dụ

```
Panel p = new Panel();  
p.setLayout(new FlowLayout());  
p.add(new JLabel("One"));  
p.add(new JLabel("Two"));  
p.add(new JLabel("Three"));
```

# FlowLayout

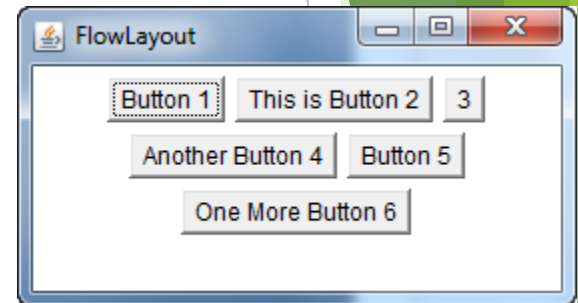
- ▶ Với các container sử dụng FlowLayout:
  - ▶ Các thành phần được sắp xếp lần lượt từ trái sang phải
  - ▶ Khi đầy một dòng -> tạo dòng mới
- ▶ Phương thức khởi tạo
  - ▶ `public FlowLayout();`
  - ▶ `public FlowLayout(int align);`
  - ▶ `public FlowLayout(int align, int hgap, int vgap);`

*Align:*

- ▶ `FlowLayout.LEFT` (or `LEADING`)
- ▶ `FlowLayout.RIGHT` (or `TRAILING`)
- ▶ `FlowLayout.CENTER`

*hgap, vgap:* khoảng cách dọc/ngang giữa các thành phần.

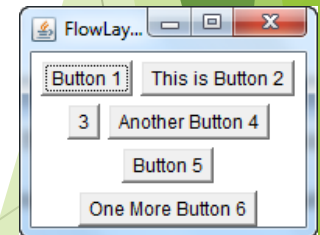
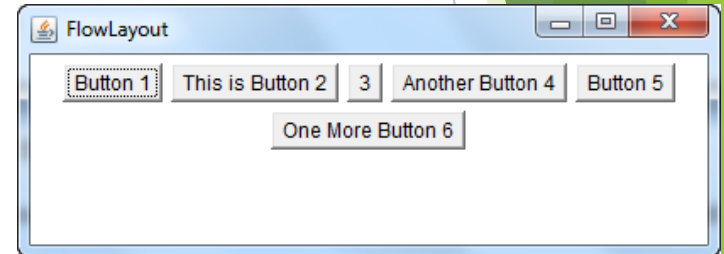
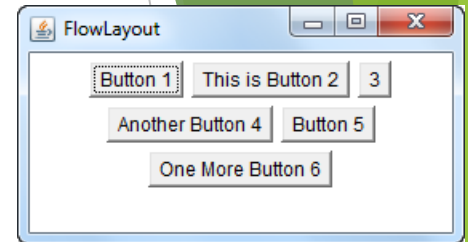
- ▶ mặc định: `hgap=5, vgap=5, align=CENTER`



# Ví dụ FlowLayout

```
import java.awt.*;
import java.awt.event.*;
public class AWTFlowLayout extends Frame {
    public AWTFlowLayout () {
        setLayout(new FlowLayout());
        add(new Button("Button 1"));
        add(new Button("This is Button 2"));
        add(new Button("3"));
        add(new Button("Another Button 4"));
        add(new Button("Button 5"));
        add(new Button("One More Button 6"));

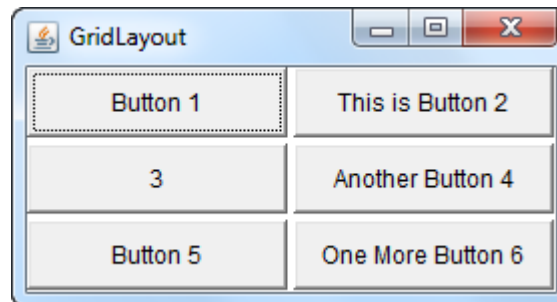
        setTitle("FlowLayout"); // "this" Frame sets title
        setSize(280, 150);      // "this" Frame sets initial size
        setVisible(true);       // "this" Frame shows
    }
    public static void main(String[] args) {
        new AWTFlowLayout(); // Let the constructor do the job
    }
}
```





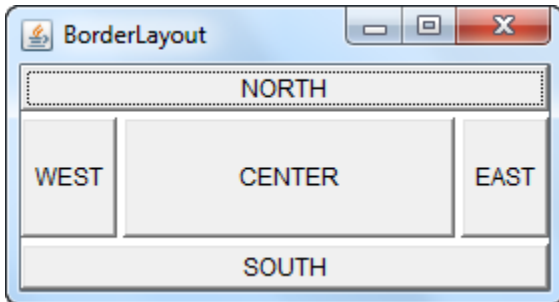
# GridLayout

- ▶ Với các container sử dụng FlowLayout:
  - ▶ Các thành phần được sắp xếp theo hàng và cột
- ▶ Phương thức khởi tạo
  - ▶ `public GridLayout(int rows, int columns);`
  - ▶ `public GridLayout(int rows, int columns, int hgap, int vgap);`
- ▶ mặc định: `rows=1, cols=0, hgap=0, vgap=0`



# BorderLayout

- ▶ Với BorderLayout, container được chia làm 5 phần: EAST, WEST, SOUTH, NORTH, CENTER
- ▶ Phương thức khởi tạo
  - ▶ `public BorderLayout();`
  - ▶ `public BorderLayout(int hgap, int vgap);`
  - ▶ mặc định `hgap=0, vgap=0`



- ▶ Khi thêm vào một thành phần
  - ▶ `aContainer.add(aComponent, aZone)`
    - ▶ `aZone`:
      - ▶ `BorderLayout.NORTH` (or `PAGE_START`)
      - ▶ `BorderLayout.SOUTH` (or `PAGE_END`)
      - ▶ `BorderLayout.WEST` (or `LINE_START`)
      - ▶ `BorderLayout.EAST` (or `LINE_END`)
      - ▶ `BorderLayout.CENTER`
  - ▶ `aContainer.add(aComponent)`: thêm thành phần vào CENTER
- ▶ Không bắt buộc phải thêm đủ thành phần vào cả 5 vùng

# 3

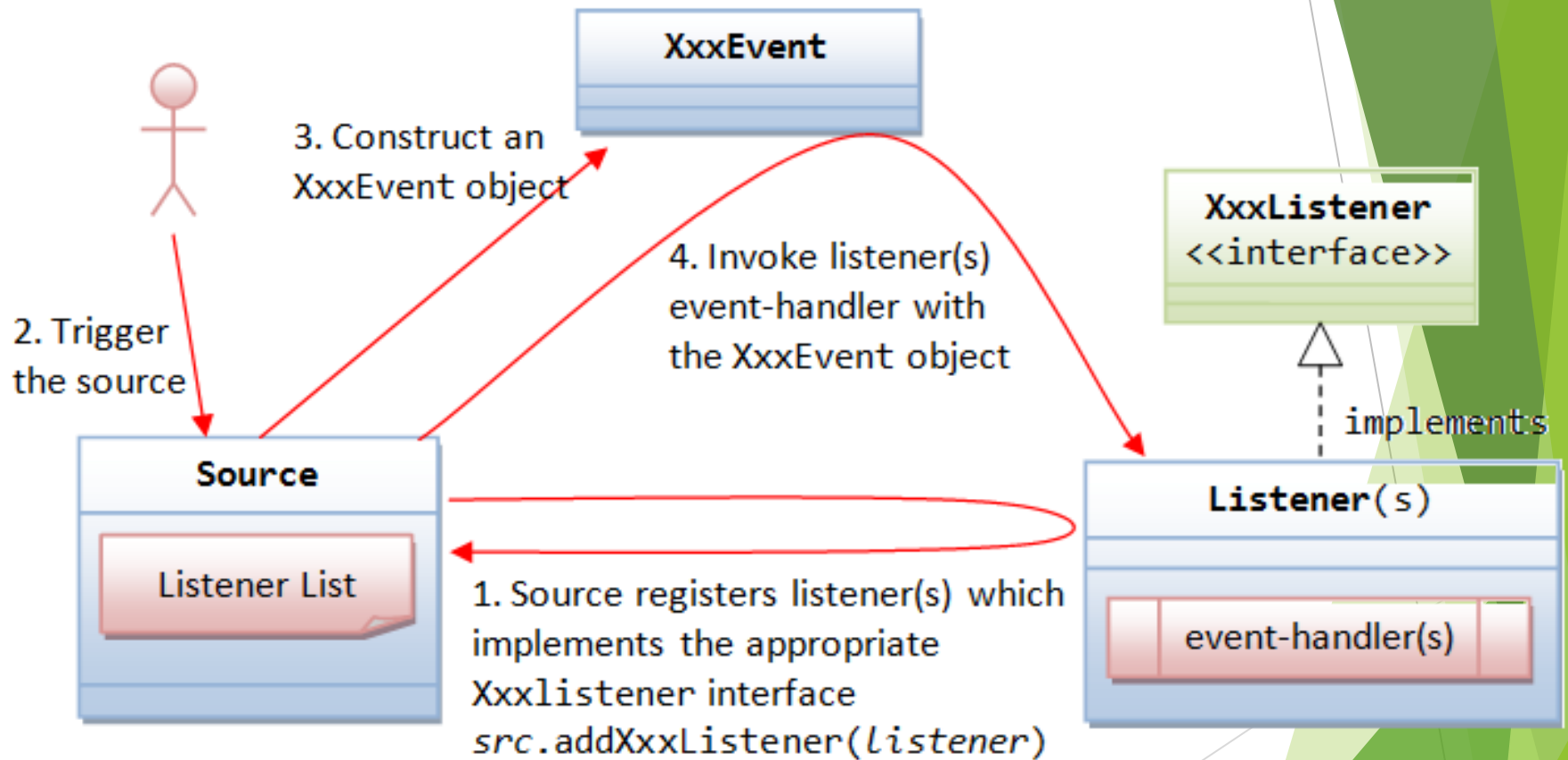
## Xử lý sự kiện

Các giao diện listener và các cài đặt

# Tổng quan

- ▶ Xử lý sự kiện theo mô hình "hướng sự kiện" (event-driven)
  - ▶ Khi một sự kiện xảy ra -> gọi đến mã xử lý tương ứng
- ▶ Các đối tượng: **source**, **listener**, **event**
  - ▶ **source** là đối tượng sinh ra các sự kiện (**event**) (ví dụ: khi người dùng tác động vào)
  - ▶ **event** thông điệp được gửi đến các đối tượng **listener** đã được đăng ký
  - ▶ Các phương thức xử lý sự kiện tương ứng của **listener** sẽ được gọi để xử lý các sự kiện
- ▶ Các **listener** phải được đăng ký với các **source** để quản lý các sự kiện nhất định có thể xảy ra tại các **source**.

# Xử lý sự kiện



# Chiến lược xử lý sự kiện

- ▶ AWT cung cấp các lớp xử lý sự kiện trong `java.awt.event`

<i>Các giao diện listener</i>	<i>Phương thức đăng ký</i>
ActionListener	addActionListener
AdjustmentListener	addAdjustmentListener
ComponentListener	addComponentListener
ContainerListener	addContainerListener
FocusListener	addFocusListener
ItemListener	addItemListener
KeyListener	addKeyListener
MouseListener	addMouseListener
MouseMotionListener	addMouseMotionListener
TextListener	addTextListener
WindowListener	addWindowListener

# Một số listener cơ bản

## ▶ ActionListener

- ▶ Xử lý các nút và một số ít các hành động khác
  - ▶ `actionPerformed(ActionEvent event)`

## ▶ AdjustmentListener

- ▶ Áp dụng khi cuộn (scrolling)
  - ▶ `adjustmentValueChanged(AdjustmentEvent event)`

## ▶ ComponentListener

- ▶ Xử lý các sự kiện dịch chuyển/thay đổi kích thước/ẩn các đối tượng GUI
  - ▶ `componentResized(ComponentEvent event)`
  - ▶ `componentMoved (ComponentEvent event)`
  - ▶ `componentShown(ComponentEvent event)`
  - ▶ `componentHidden(ComponentEvent event)`

# Một số listener cơ bản

## ▶ ContainerListener

- ▶ Được kích hoạt khi cửa sổ thêm/gỡ bỏ các đối tượng GUI

- ▶ `componentAdded(ContainerEvent event)`
  - ▶ `componentRemoved(ContainerEvent event)`

## ▶ FocusListener

- ▶ Phát hiện khi nào các đối tượng có/mất focus

- ▶ `focusGained(FocusEvent event)`
  - ▶ `focusLost(FocusEvent event)`



# Một số listener cơ bản

- ▶ `ItemListener`
  - ▶ Xử lý các sự kiện chọn trong các list, checkbox, ...
    - ▶ `itemStateChanged(ItemEvent event)`
- ▶ `KeyListener`
  - ▶ Phát hiện ra các sự kiện liên quan đến bàn phím
    - ▶ `keyPressed(KeyEvent event) // any key pressed down`
    - ▶ `keyReleased(KeyEvent event) // any key released`
    - ▶ `keyTyped(KeyEvent event) // key for printable char released`

# Một số listener cơ bản

- ▶ `MouseListener`

- ▶ Áp dụng cho các sự kiện chuột cơ bản

- ▶ `mouseEntered(MouseEvent event)`
    - ▶ `mouseExited(MouseEvent event)`
    - ▶ `mousePressed(MouseEvent event)`
    - ▶ `mouseReleased(MouseEvent event)`
    - ▶ `mouseClicked(MouseEvent event)` -- Nhả chuột khi không kéo
      - ▶ Áp dụng khi nhả chuột mà không di chuyển từ khi nhấn chuột

- ▶ `MouseMotionListener`

- ▶ Xử lý các sự kiện di chuyển chuột

- ▶ `mouseMoved(MouseEvent event)`
    - ▶ `mouseDragged(MouseEvent event)`

# Một số listener cơ bản

## ▶ TextListener

- ▶ Áp dụng cho các textfield và text area
  - ▶ `textValueChanged(TextEvent event)`

## ▶ WindowListener

- ▶ Xử lý các sự kiện mức cao của cửa sổ
  - ▶ `windowOpened`, `windowClosing`, `windowClosed`,  
`windowIconified`, `windowDeiconified`,  
`windowActivated`, `windowDeactivated`
    - ▶ `windowClosing` đặc biệt rất hữu dụng

# Chiến lược xử lý sự kiện

- ▶ Xây dựng lớp listener tương ứng với sự kiện
  - ▶ Thực thi giao diện XxxListener
- ▶ Cài đặt các phương thức tương ứng với sự kiện của giao diện này
- ▶ Đăng ký các listener với nguồn
  - ▶ `public void addXxxListener(XxxListener l);`
  - ▶ `public void removeXxxListener(XxxListener l);`

# Ví dụ

## ► Giao diện `MouseListener`

```
interface MouseListener {  
    // Called back upon mouse-button pressed  
    public void mousePressed(MouseEvent evt);  
    // Called back upon mouse-button released  
    public void mouseReleased(MouseEvent evt);  
    // Called back upon mouse-button clicked  
    // (pressed and released)  
    public void mouseClicked(MouseEvent evt);  
    // Called back when mouse pointer entered the  
    // component  
    public void mouseEntered(MouseEvent evt);  
    // Called back when mouse pointer exited the  
    // component  
    public void mouseExited(MouseEvent evt);  
}
```

# Xây dựng lớp Listener

```
class MyMouseListener implements MouseListener {  
    public void mousePressed(MouseEvent event) {  
        System.out.println  
            ("Mouse-button pressed at (" + event.getX()  
            + "," + event.getY() + ").");  
    }  
  
    public void mouseReleased(MouseEvent event) {}  
    public void mouseClicked(MouseEvent event) {}  
    public void mouseEntered(MouseEvent event) {}  
    public void mouseExited(MouseEvent event) {}  
}
```

# Đăng ký với button

```
import java.awt.*;

public class ButtonEventExample extends Frame {
    public ButtonEventExample () {
        Button b = new Button("Button");
        add(b);
        b.addMouseListener(new MyMouseListener());

        setTitle("Button Event Example");
        setSize(280, 150);
        setVisible(true);
    }

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

# Adapter

- ▶ Nhược điểm của việc sử dụng giao diện XxxListener
  - ▶ Phải cài đặt tất cả các phương thức của giao diện
  - ▶ Nếu chỉ cần xử lý 1 sự kiện -> tạo ra rất nhiều phương thức rỗng
- ▶ AWT cung cấp các lớp **adapter** cho các listener có nhiều hơn 1 phương thức
- ▶ Để sử dụng các lớp này, ta viết các lớp kế thừa từ các lớp Adapter thay vì thực thi các giao diện



# Các Adapter tương ứng

<i>Các giao diện listener</i>	<i>Các lớp Adapter</i>	<i>Phương thức đăng ký</i>
ActionListener		addActionListener
AdjustmentListener		addAdjustmentListener
ComponentListener	ComponentAdapter	addComponentListener
ContainerListener	ContainerAdapter	addContainerListener
FocusListener	FocusAdapter	addFocusListener
ItemListener		addItemListener
KeyListener	KeyAdapter	addKeyListener
MouseListener	MouseAdapter	addMouseListener
MouseMotionListener	MouseMotionAdapter	addMouseMotionListener
TextListener		addTextListener
WindowListener	WindowAdapter	addWindowListener

# Ví dụ

```
class MyMouseListener extends MouseAdapter {  
    public void mousePressed(MouseEvent event) {  
        System.out.println("Mouse-button pressed at (" +  
            event.getX() + "," + event.getY() + ").");  
    }  
}
```

```
public class FrameEventExample extends Frame {  
    public FrameEventExample () {  
        addMouseListener(new MyMouseListener());  
  
        setTitle("Button Event Example");  
        setSize(640, 480);  
        setVisible(true);  
    }  
}
```

# Lấy đối tượng từ source

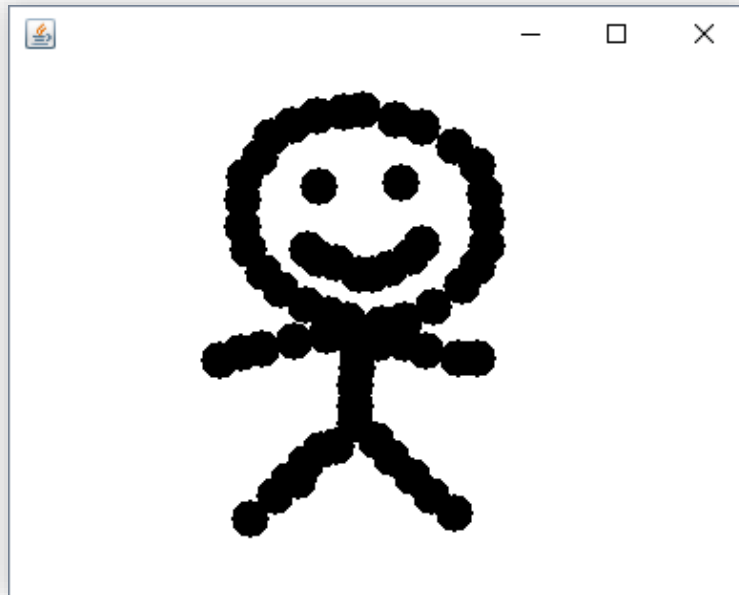
- ▶ Nếu ClickListener muốn vẽ một hình tròn tại vị trí chuột được nhấn?
  - ▶ Tại sao không thể gọi phương thức `getGraphics` để lấy về đối tượng `Graphics` để vẽ
- ▶ Cách 1
  - ▶ Gọi `event.getSource` để lấy về một tham chiếu tới cửa sổ hoặc thành phần GUI tạo ra sự kiện
  - ▶ Ép kết quả trả về theo ý muốn
  - ▶ Gọi các phương thức trên tham chiếu đó

# Cách 1: Sử dụng getSource

```
public class CircleListener extends MouseAdapter
{
    private int radius = 10;
    public void mousePressed(MouseEvent event) {
        Frame app = (Frame)event.getSource();
        Graphics g = app.getGraphics();
        g.fillOval(event.getX()-radius,
                    event.getY()-radius,
                    2*radius, 2*radius);
    }
}
```

# Cách 1: Sử dụng getSource

```
public class FrameEventExample extends Frame {  
    private int radius = 10;  
    public FrameEventExample() {  
        addMouseListener(new CircleListener());  
        setSize(640, 480);  
        setVisible(true);  
    }  
}
```



## Cách 2: Thực thi giao diện listener

- ▶ Giúp frame hiện tại đóng vai trò như một listener

```
import java.applet.Applet;  
import java.awt.*;  
import java.awt.event.*;  
  
public class FrameEventExample extends Frame  
    implements MouseListener {  
    private int radius = 10;  
    public FrameEventExample() {  
        addMouseListener(this);  
    }  
}
```

## Cách 2: Thực thi giao diện listener

```
public void mouseEntered(MouseEvent event) {}  
public void mouseExited(MouseEvent event) {}  
public void mouseReleased(MouseEvent event) {}  
public void mouseClicked(MouseEvent event) {}  
public void mousePressed(MouseEvent event) {  
    Graphics g = getGraphics();  
    g.fillOval(event.getX()-radius,  
               event.getY()-radius,  
               2*radius, 2*radius);  
}  
}
```

# Cách 3: Sử dụng inner class

- Viết lớp listener *bên trong* lớp frame

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class FrameEventExample extends Frame {
    public FrameEventExample() {
        addMouseListener(new CircleListener());
    }
    private class CircleListener extends MouseAdapter {
        ...
    }
}
```



## Cách 4: Sử dụng anonymous inner class

```
public class FrameEventExample extends Frame {  
    public FrameEventExample() {  
        addMouseListener  
            (new MouseAdapter() {  
                private int radius = 25;  
                public void mousePressed(MouseEvent event) {  
                    Graphics g = getGraphics();  
                    g.fillOval(event.getX()-radius,  
                             event.getY()-radius,  
                             2*radius, 2*radius);  
                }  
            });  
    }  
}
```

# 4

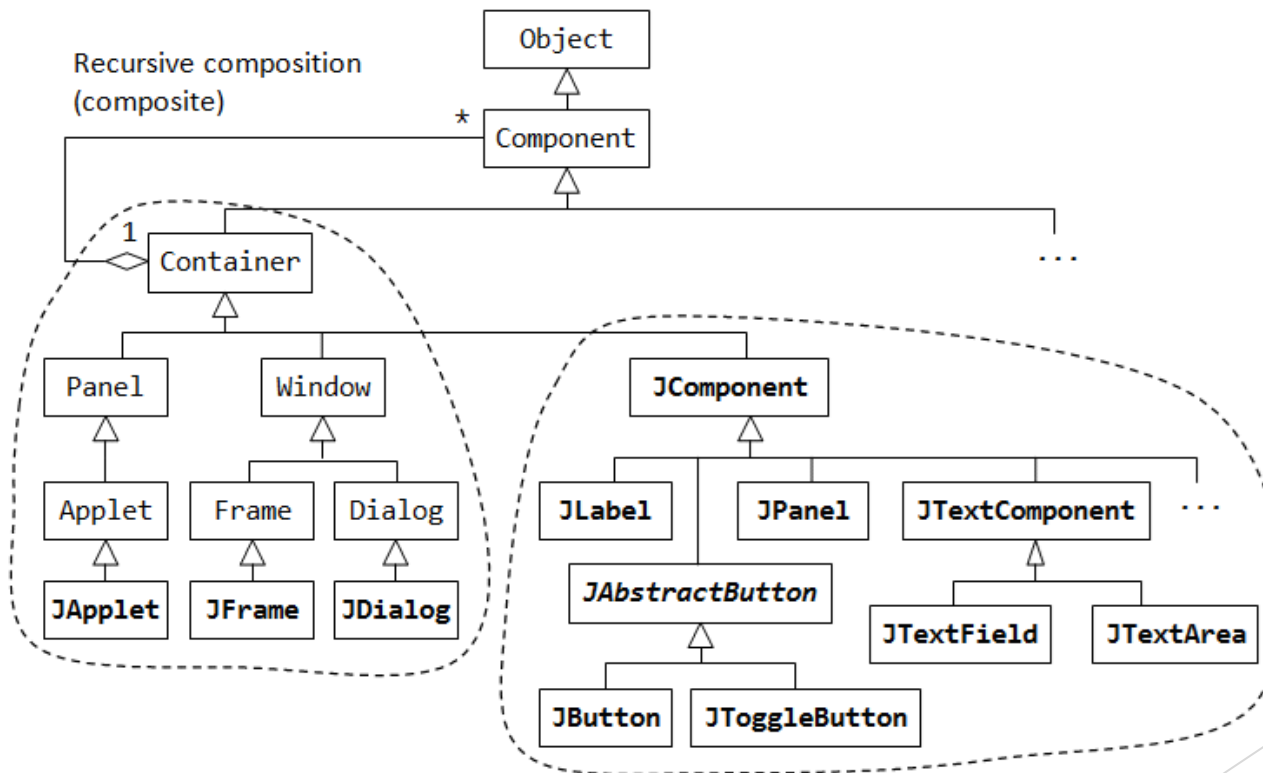
## Swing

javax.swing

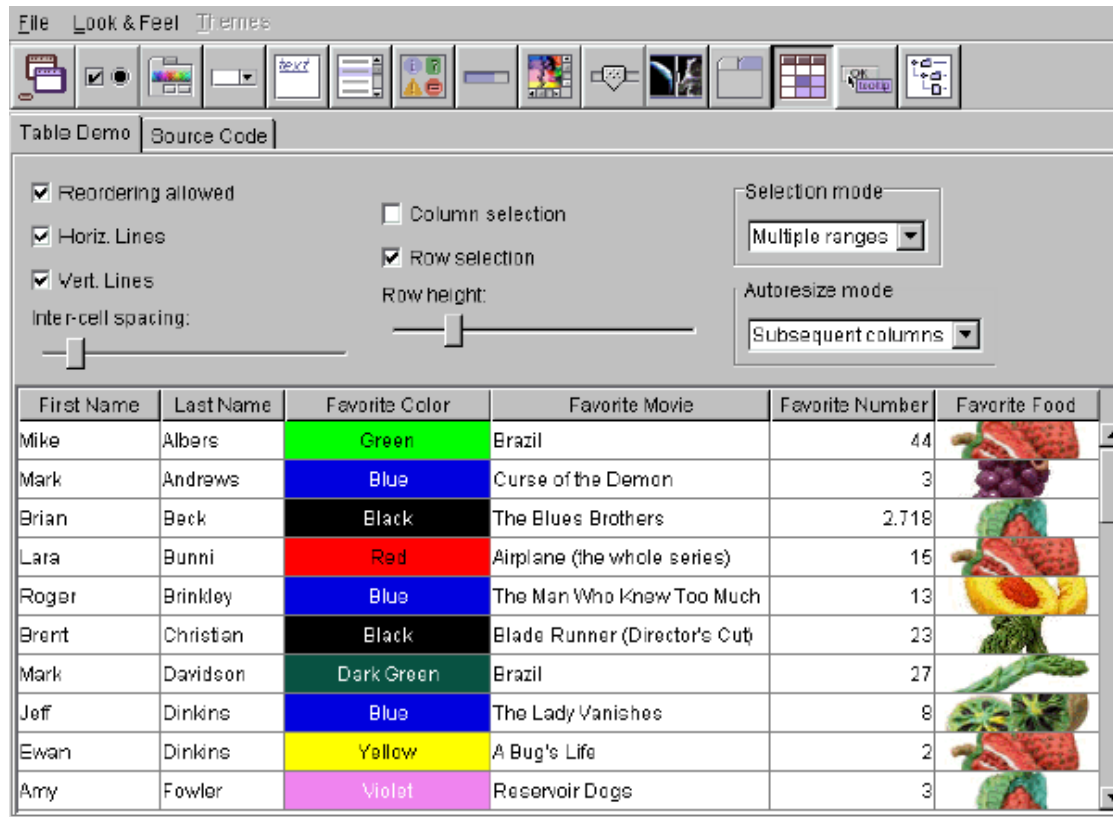
# Swing vs. AWT

- ▶ Cách đặt tên
  - ▶ Tất cả các thành phần trong swing đều có tên bắt đầu với chữ hoa J và tuân theo khuôn dạng JXxxx. Ví dụ: JFrame, JPanel, JApplet, JDialog, JButton,...
- ▶ Các thành phần “nhẹ”? (lightweight)
  - ▶ Hầu hết các thành phần swing đều “nhẹ”, được tạo ra bằng cách vẽ trong cửa sổ cơ sở
- ▶ Look and Feel mới (mặc định)
  - ▶ Có thể thay đổi Look and Feel tự nhiên (native look)
- ▶ Không nên trộn cả swing và awt trong một cửa sổ.

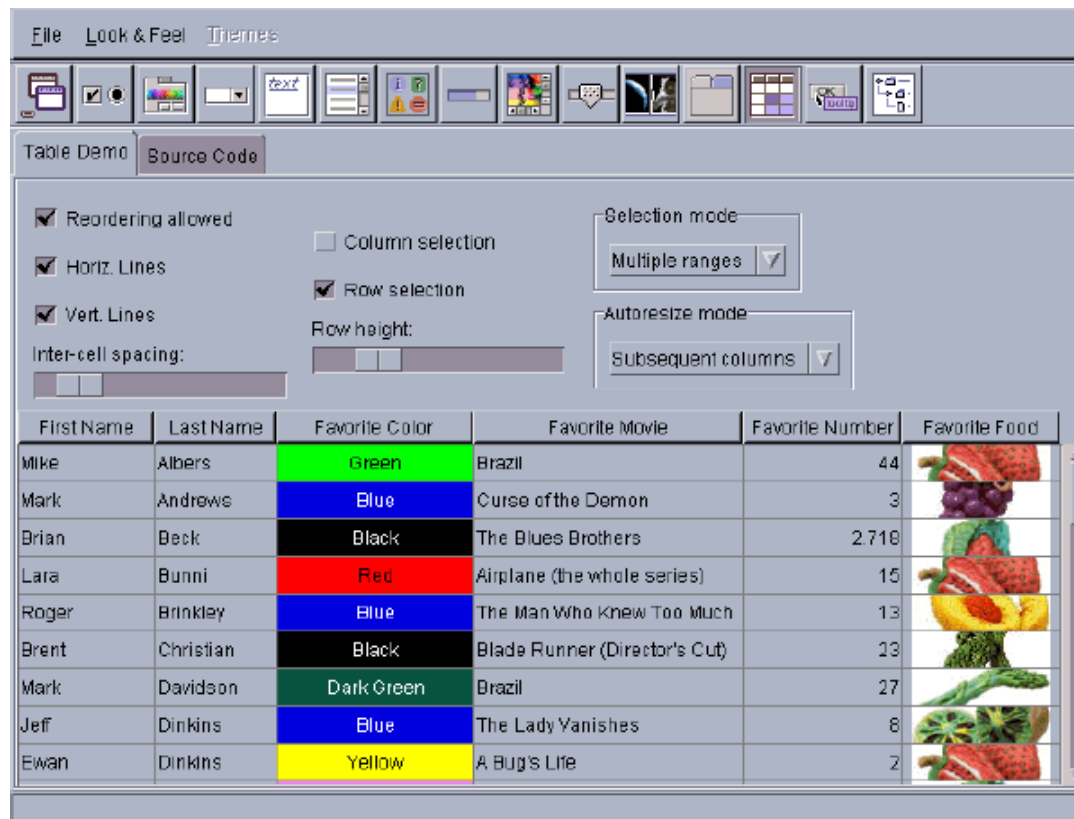
# GUI component hierarchy



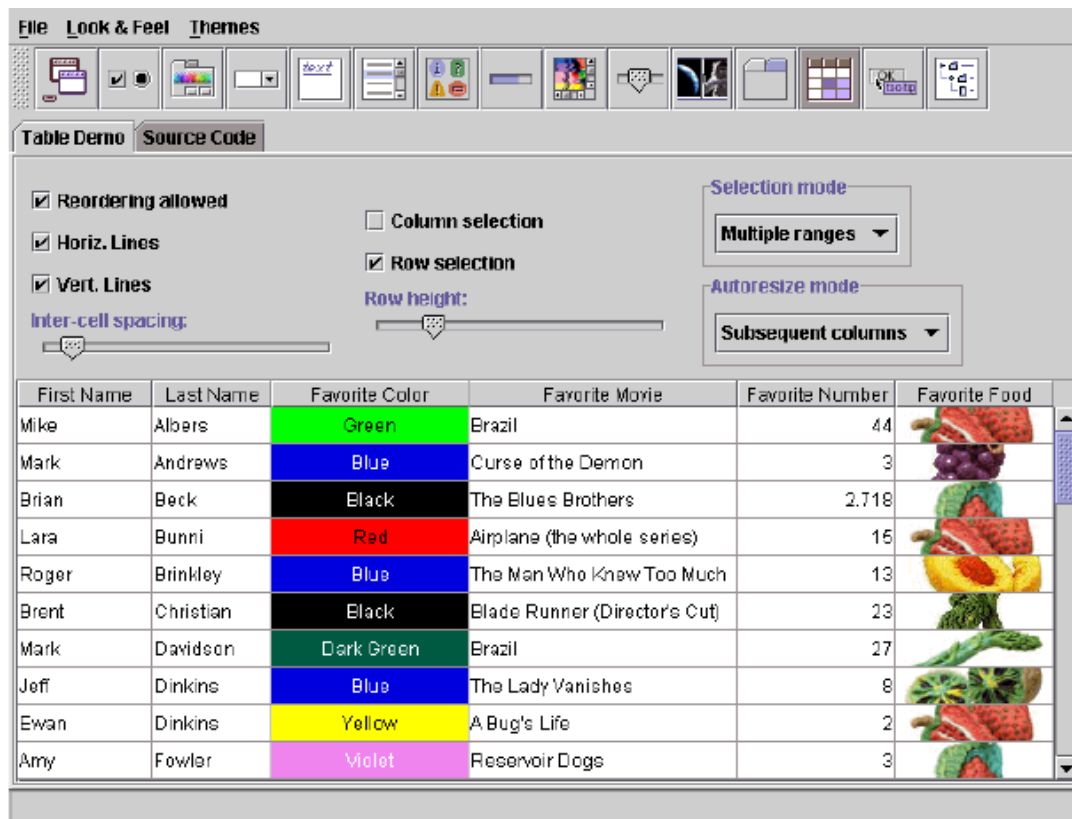
# Windows Look and Feel



# Motif Look and Feel



# Java Look and Feel



# Thay đổi Look and Feel

## ► Gọi phương thức setLookAndFeel

```
public class WindowUtilities {  
    public static void setNativeLookAndFeel() {  
        try {  
            UIManager.setLookAndFeel(  
                UIManager.getSystemLookAndFeelClassName());  
        } catch (Exception e) {  
            System.out.println("Co loi khi thay doi LAF: "+e);  
        }  
    }  
}  
...
```



# Các thành phần Swing

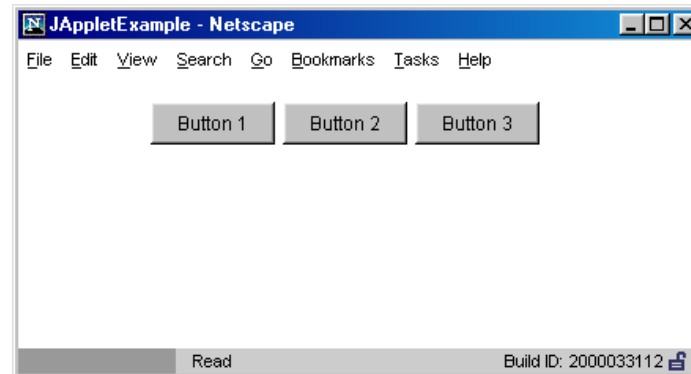
- ▶ Các lớp container
  - ▶ JApplet, JFrame
- ▶ Các thành phần Swing tương đương với các thành phần AWT
  - ▶ JLabel, JButton, JPanel, JSlider
- ▶ Các thành phần Swing mới
  - ▶ JColorChooser, JInternalFrame, JOptionPane, JToolBar, JEditorPane
- ▶ Các thành phần đơn giản khác
  - ▶ JCheckBox, JRadioButton, JTextField, JTextArea, JFileChooser

# JApplet

- ▶ Content pane
  - ▶ A JApplet contains a content pane in which to add components. Changing other properties like the layout manager, background color, etc., also applies to the content pane. Access the content pane through `getContentPane`.
- ▶ Layout manager
  - ▶ The default layout manager is `BorderLayout` (as with `Frame` and `JFrame`), not `FlowLayout` (as with `Applet`). `BorderLayout` is really layout manager of content pane.
- ▶ Look and feel
  - ▶ The default look and feel is Java (Metal), so you have to explicitly switch the look and feel if you want the native look.

# JApplet

```
import java.awt.*;  
import javax.swing.*;  
public class JAppletExample extends JApplet {  
    public void init() {  
        WindowUtilities.setNativeLookAndFeel();  
        Container content = getContentPane();  
        content.setBackground(Color.white);  
        content.setLayout(new FlowLayout());  
        content.add(new JButton("Button 1"));  
        content.add(new JButton("Button 2"));  
        content.add(new JButton("Button 3"));  
    }  
}
```



# Các thành phần tương tự AWT

- ▶ JLabel
  - ▶ New features: HTML content images, borders
- ▶ JButton
  - ▶ New features: icons, alignment, mnemonics
- ▶ JPanel
  - ▶ New feature: borders
- ▶ JSlider
  - ▶ New features: tick marks and labels

# JButton

- ▶ Main new feature: icons
  - ▶ Create an ImageIcon by passing the ImageIcon constructor a String representing a GIF or JPG file (animated GIFs!).
  - ▶ Pass the ImageIcon to the JButton constructor.
- ▶ Other features
  - ▶ HTML content as with JLabel
  - ▶ Alignment: location of image with respect to text
  - ▶ Mnemonics: keyboard accelerators that let you use Alt-someChar to trigger the button.

# Ví dụ JButton

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

public class JButtons extends JFrame {
    public static void main(String[] args) {
        new JButtons();
    }
    public JButtons() {
        super("Using JButton");
        WindowUtilities.setNativeLookAndFeel();
        addWindowListener(new ExitListener());
        Container content = getContentPane();
        content.setBackground(Color.white);
        content.setLayout(new FlowLayout());
    }
}
```

# Ví dụ JButton

```
JButton button1 = new JButton("Java");
    content.add(button1);
    ImageIcon cup = new ImageIcon("images/cup.gif");
    JButton button2 = new JButton(cup);
    content.add(button2);
    JButton button3 = new JButton("Java", cup);
    content.add(button3);
    JButton button4 = new JButton("Java", cup);

button4.setHorizontalTextPosition(SwingConstants.LEFT);
    content.add(button4);
    pack();
    setVisible(true);
}
```

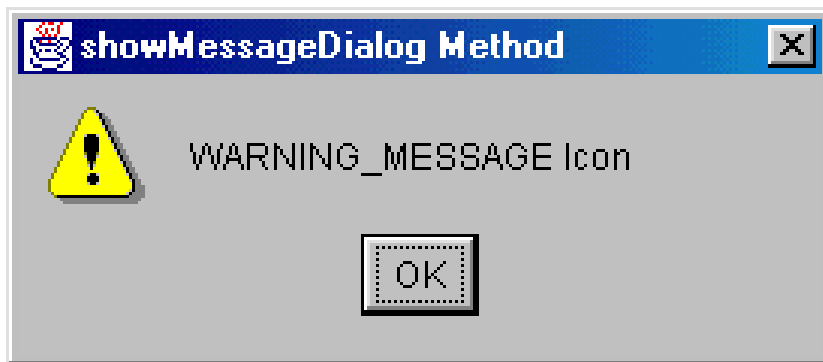
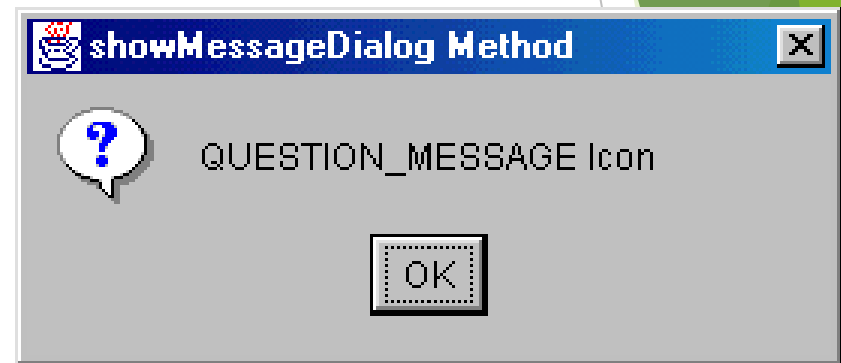


# JOptionPane

- ▶ Very rich class with many options for different types of dialog boxes.
- ▶ Five main static methods
  - ▶ `JOptionPane.showMessageDialog`
    - ▶ Icon, message, OK button
  - ▶ `JOptionPane.showConfirmDialog`
    - ▶ Icon, message, and buttons:  
OK, OK/Cancel, Yes/No, or Yes/No/Cancel
  - ▶ `JOptionPane.showInputDialog` (2 versions)
    - ▶ Icon, message, textfield or combo box, buttons
  - ▶ `JOptionPane.showOptionDialog`
    - ▶ Icon, message, array of buttons or other components

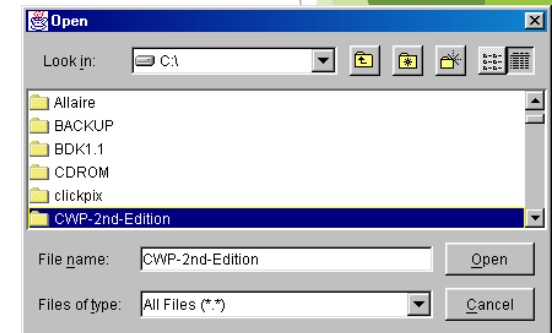


# JOptionPane Message Dialogs



# Các thành phần khác

- ▶ JCheckBox
  - ▶ Note uppercase B (vs. Checkbox in AWT)
- ▶ JRadioButton
  - ▶ Use a ButtonGroup to link radio buttons
- ▶ JTextField
  - ▶ Just like AWT TextField except that it does not act as a password field (use JPasswordField for that)
- ▶ JTextArea
  - ▶ Place in JScrollPane if you want scrolling
- ▶ JFileChooser



The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look. The shapes are concentrated on the right side and bottom of the slide, leaving the top-left area mostly white.

# Thank you!

Any questions?