



**BỘ MÔN CÔNG NGHỆ PHẦN MỀM
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**




LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 10. Đồ họa và xử lý sự kiện

Nguyễn Thị Thu Trang

Nội dung

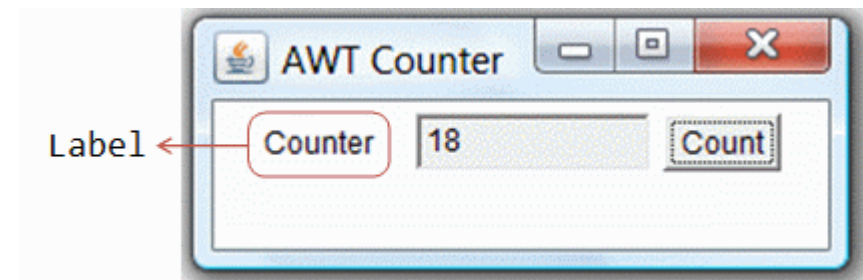
- 
1. Lập trình đồ họa và AWT
 2. Xử lý sự kiện
 3. Quản lý bố cục (layout)
 4. Java Swing

Lập trình đồ họa trong Java

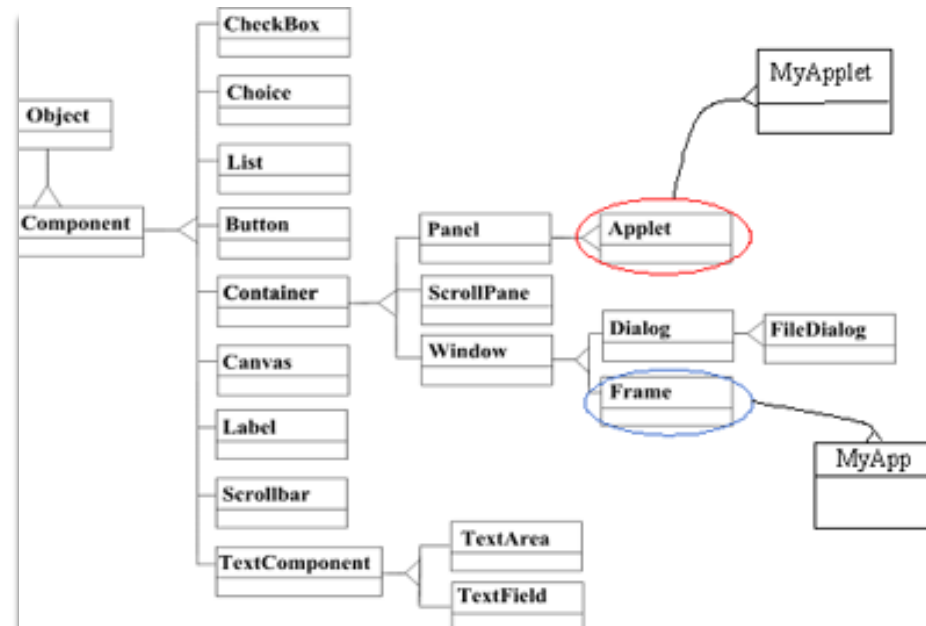
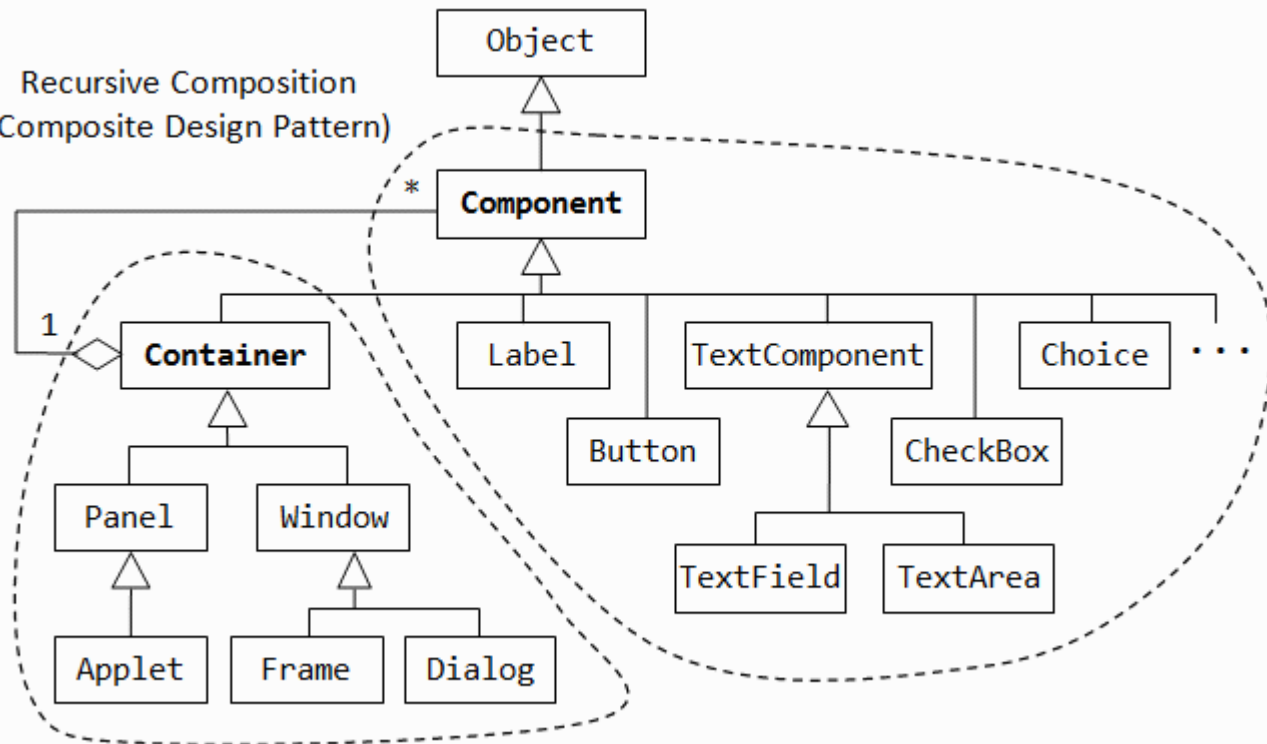
- GUI Programming
- Giúp tạo ra các ứng dụng có giao diện đồ họa với nhiều các điều khiển như: Button, Textbox, Label, Checkbox, List, Tree...
- Java cũng cấp 2 thư viện lập trình đồ họa
 - `java.awt`
 - `javax.swing`

Các thành phần giao diện đồ họa AWT có sẵn

- **Buttons** (java.awt.Button)
- **Checkboxes** (java.awt.Checkbox)
- **Single-line text fields** (java.awt.TextField)
- **Menus** (java.awt.MenuItem)
- **Containers** (java.awt.Panel)
- **Lists** (java.awt.List)



Recursive Composition (Composite Design Pattern)



Windows và Layout Manager

- Container
 - Hầu hết các cửa sổ đều là một Container có thể chứa các cửa sổ khác hoặc các thành giao diện đồ họa khác. Canvas là một ngoại lệ
- Layout manager
 - Container có một LayoutManager tự động thay đổi kích thước và vị trí của các thành phần trong cửa sổ
 - Có thể thay đổi các hành vi của layout manager hoặc vô hiệu nó hoàn toàn
- Event
 - Cửa sổ và các thành phần có thể nhận các sự kiện bàn phím hoặc chuột
- Popup Windows
 - Một vài cửa sổ (**Frame** và **Dialog**) có thanh tiêu đề và viền của riêng chúng và có thể đặt ở một vị trí bất kỳ trên màn hình
 - Các cửa sổ khác (**Canvas** và **Panel**) chỉ có thể được nhúng trong cửa sổ đã có

Lớp Canvas

- Mục đích chính
 - Một vùng để vẽ
 - Một Component được tùy biến không cần chứa các Component khác (ví dụ như một image button)
- Default Layout Manager - None
 - Canvas *không thể chứa bất kỳ một* Component nào
- Tạo và sử dụng
 - Tạo Canvas

```
Canvas canvas = new Canvas() ;
```

Hoặc tạo ra lớp con của Canvas đã sửa việc vẽ thông qua phương thức paint:

```
SpecializedCanvas canvas =  
    new SpecializedCanvas() ;
```

Lớp Canvas (2)

- Tạo và sử dụng (2)

- Kích thước của Canvas

```
canvas.setSize(width, height);
```

- Thêm Canvas vào cửa sổ Window hiện tại

```
add(canvas);
```

hoặc phụ thuộc vào layout manager để định vị cho Canvas

```
add(canvas, BorderLayout.Region_Name);
```

Nếu tạo ra một cửa sổ riêng biệt (ví dụ như Panel) rồi đặt Canvas vào cửa sổ thì sử dụng:

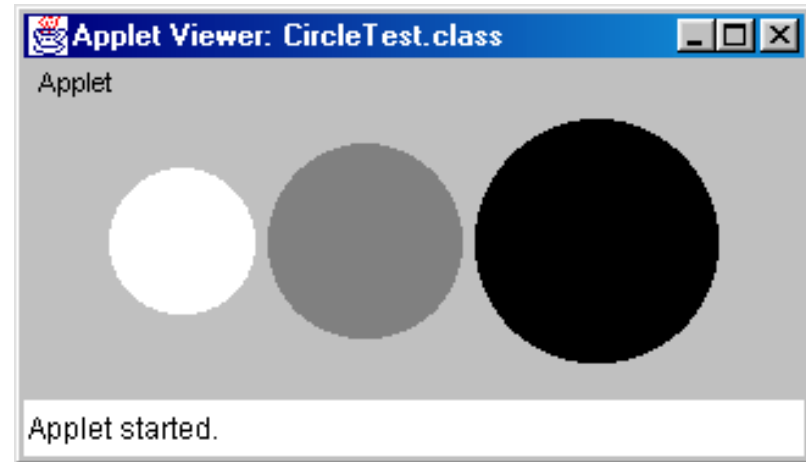
```
someWindow.add(canvas);
```


Ví dụ về Canvas

```
import java.awt.*;

/** A Circle component built using a Canvas. */

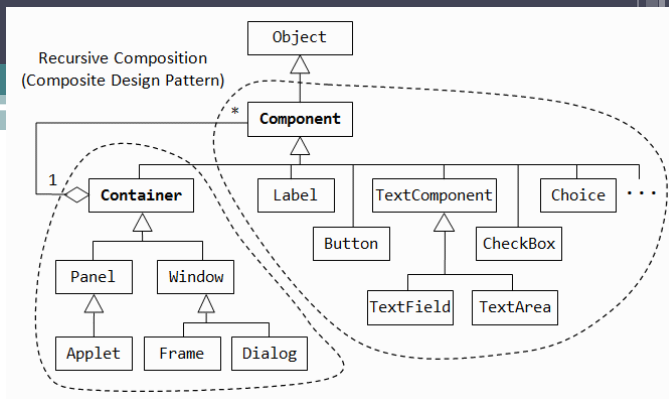
public class Circle extends Canvas {
    private int width, height;
    public Circle(Color foreground, int radius) {
        setForeground(foreground);
        width = 2*radius;
        height = 2*radius;
        setSize(width, height);
    }
    public void paint(Graphics g) {
        g.fillOval(0, 0, width, height);
    }
    public void setCenter(int x, int y) {
        setLocation(x - width/2, y - height/2);
    }
}
```



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

public class CircleTest extends Applet
{
    public void init() {
        setBackground(Color.lightGray);
        add(new Circle(Color.white, 30));
        add(new Circle(Color.gray, 40));
        add(new Circle(Color.black, 50));
    }
}
```

Lớp Component



- Lớp cha trực tiếp của lớp Canvas
- “Tổ tiên” (Ancestor) của tất cả các loại Window
- Các phương thức hay được dùng
 - `getBackground/setBackground`
 - `getForeground/setForeground`
 - Thay đổi hoặc lấy về màu vẽ mặc định
 - Color được kế thừa từ đối tượng Graphics của component
 - `getFont/setFont`
 - Trả về hoặc thiết lập font hiện tại
 - Được kế thừa từ đối tượng Graphics của component
 - `paint`
 - Được gọi khi người dùng gọi repaint hoặc khi component bị che khuất rồi được hiển thị trở lại.

Lớp Component (2)

- Các phương thức hay được sử dụng
 - `setVisible`
 - Hiển thị (`true`) hoặc ẩn (`false`) component
 - Especially useful for frames and dialogs
 - `setSize/setBounds/setLocation`
 - `getSize/getBounds/getLocation`
 - Physical aspects (size and position) of the component
 - `list`
 - Prints out info on this component and any components it contains; useful for debugging
 - `invalidate/validate`
 - Tell layout manager to redo the layout
 - `getParent`
 - Returns enclosing window (or `null` if there is none)

Lớp Panel

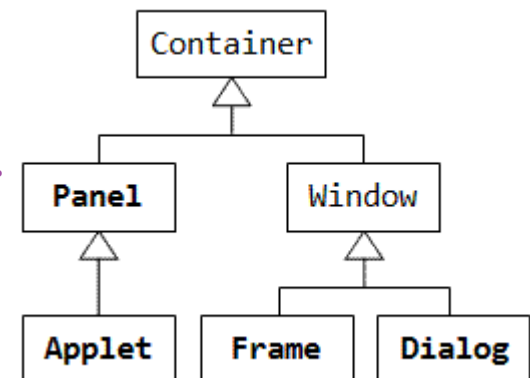
- Mục đích chính
 - Để nhóm/tổ chức các thành phần
 - Một thành phần yêu cầu các thành phần khác nhúng bên trong
- Default Layout Manager - FlowLayout
 - Co các thành phần theo **preferred size**
 - Đặt chúng từ trái sang phải theo các hàng được căn giữa
- Tạo và sử dụng
 - Tạo Panel

```
Panel panel = new Panel();
```

- Thêm các Components vào Panel

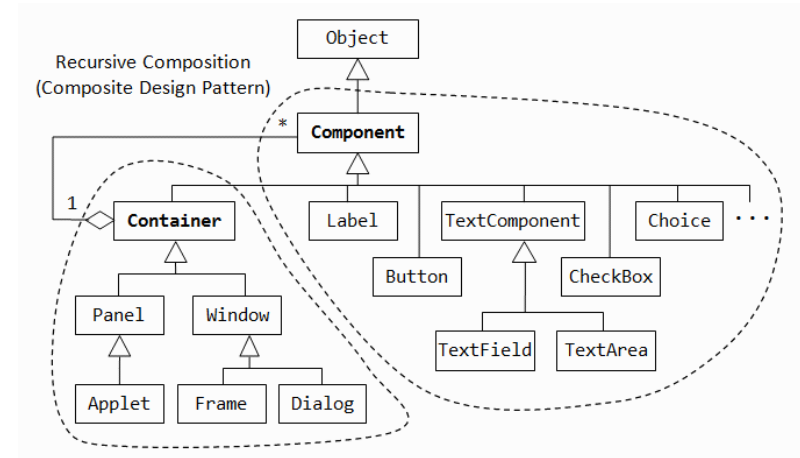
```
panel.add(someComponent);
```

```
panel.add(someOtherComponent); ..
```



Lớp Panel (2)

- Tạo và sử dụng (2)
 - Thêm Panel vào Container
 - Bên ngoài một container
 - `container.add(panel);`
 - Từ bên trong container
 - `add(panel);`
- Chú ý thiếu phương thức `setSize`
 - Các thành phần bên trong quyết định kích thước của panel; panel không thể to hơn kích thước cần thiết để chứa các thành phần
 - Một panel không chứa thành phần nào có kích thước 0
- Chú ý: Applet là một lớp con của Panel



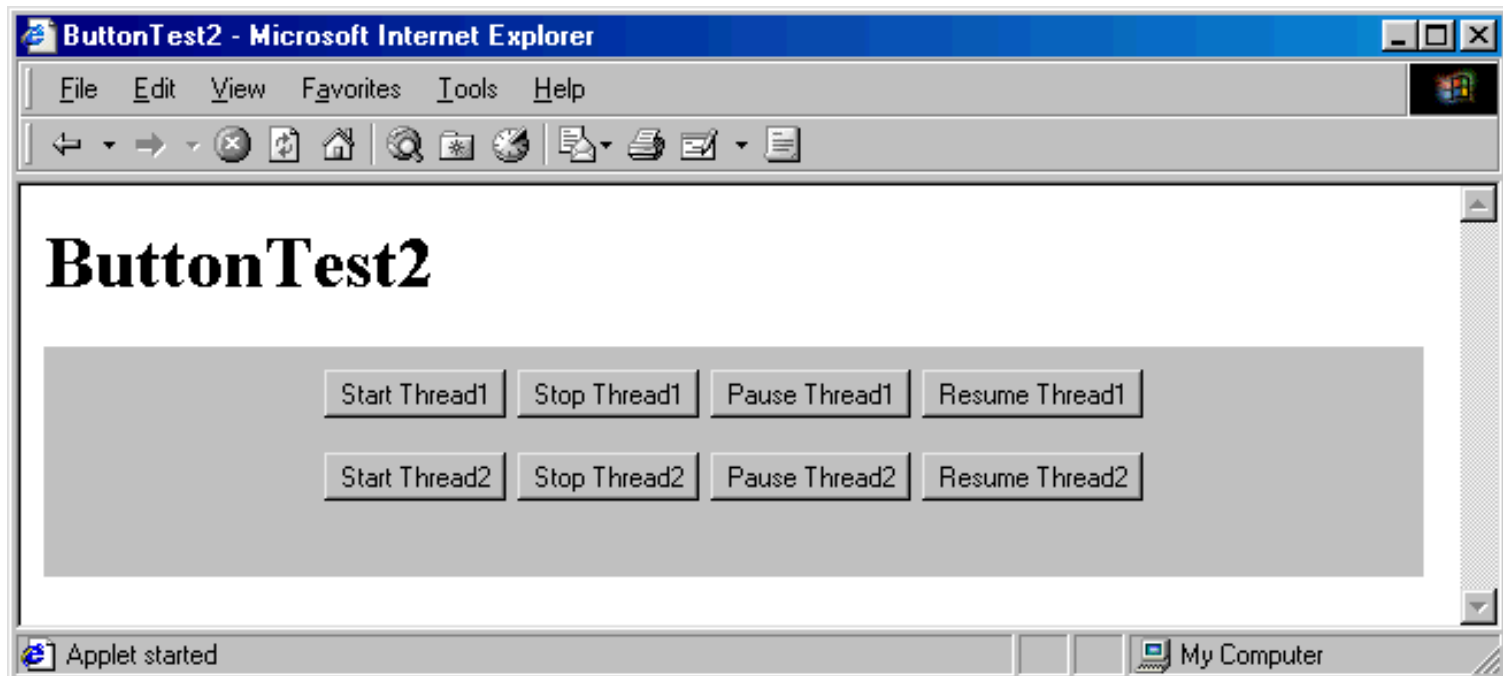
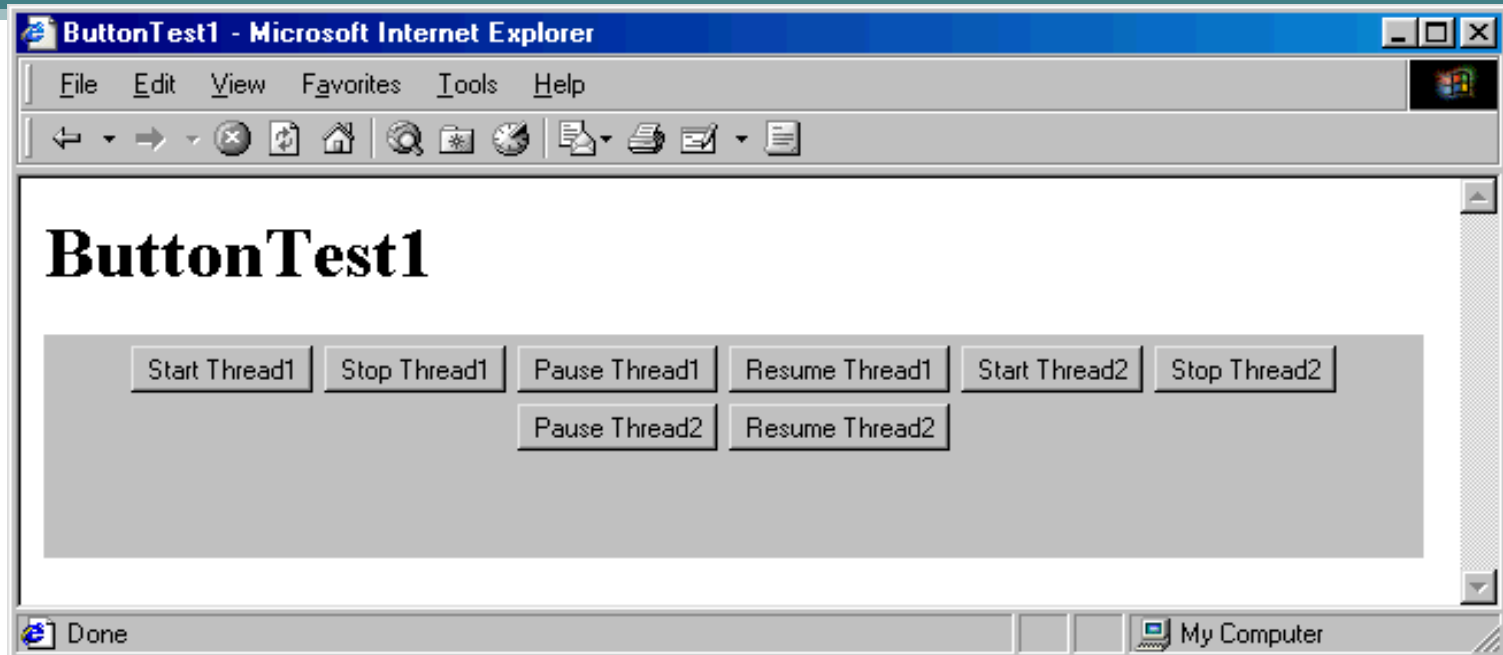
Không có/Có Panel

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

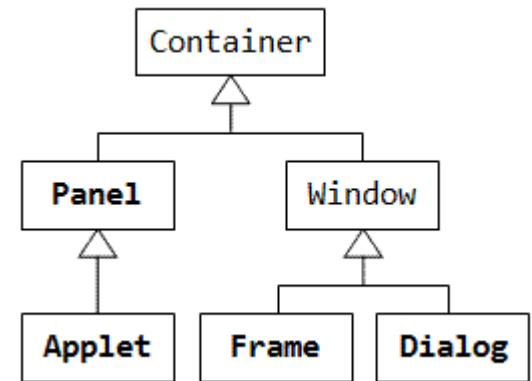
public class ButtonTest1 extends Applet
{
    public void init() {
        String[] labelPrefixes = { "Start",
        "Stop", "Pause", "Resume" };
        for (int i=0; i<4; i++) {
            add(new Button(labelPrefixes[i] +
        " Thread1"));
        }
        for (int i=0; i<4; i++) {
            add(new Button(labelPrefixes[i] +
        " Thread2"));
        }
    }
}
```

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

public class ButtonTest2 extends
Applet {
    public void init() {
        String[] labelPrefixes = {
        "Start", "Stop", "Pause", "Resume" };
        Panel p1 = new Panel();
        for (int i=0; i<4; i++) {
            p1.add(new
        Button(labelPrefixes[i] + "
        Thread1"));
        }
        Panel p2 = new Panel();
        for (int i=0; i<4; i++) {
            p2.add(new
        Button(labelPrefixes[i] + "
        Thread2"));
        }
        add(p1);
        add(p2);
    }
}
```



Lớp Frame và Dialog



- **Frame**

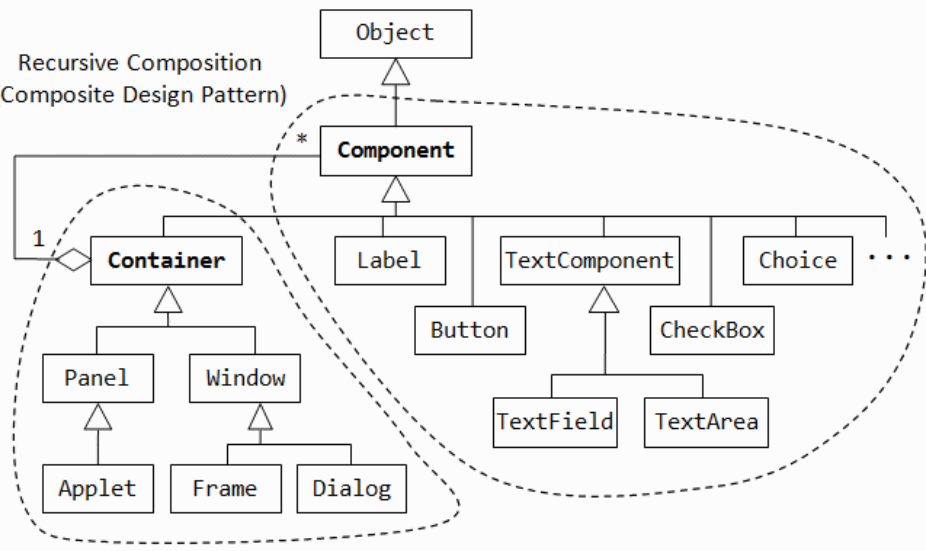
- A stand-alone window with its own title and menu bar, border, cursor and icon image
- Can contain other GUI components

- **Dialog**

- A simplified Frame (no cursor, menu, icon image).
- A modal Dialog that freezes interaction with other AWT components until it is closed.

AWT GUI Controls

Recursive Composition
(Composite Design Pattern)



- Automatically drawn
 - you don't override `paint`
- Positioned by layout manager
- Controls adopt look and feel of underlying window system
- GUI Controls
 - Button, checkbox,
 - radio button, list box, scrollbars

Buttons

- Constructors

- `Button()`, `Button(String buttonLabel)`

- The button size (preferred size) is based on the height and width of the label in the current font, plus some extra space determined by the OS

- Useful Methods

- `getLabel/setLabel`

- Retrieves or sets the current label
 - If the button is already displayed, setting the label does not automatically reorganize its `Container`
 - The **containing window** should be invalidated and validated to force a fresh layout

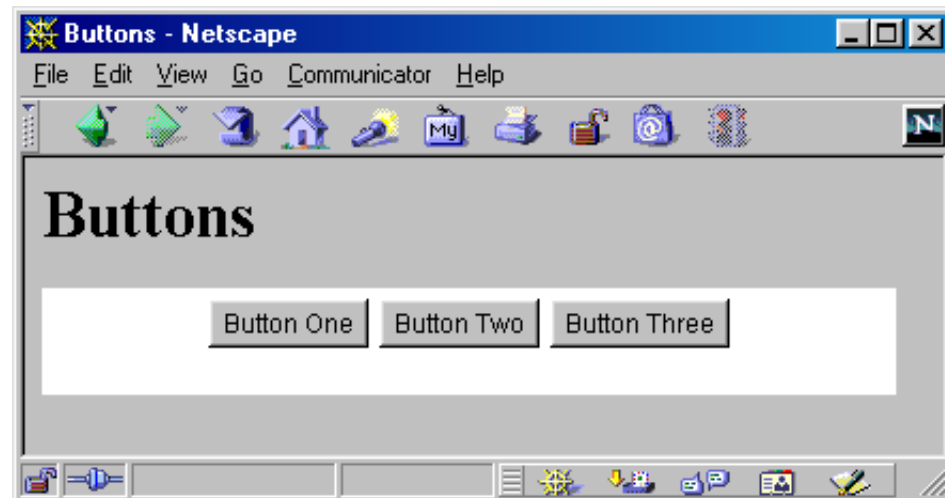
```
someButton.setLabel("A New Label");  
someButton.getParent().invalidate();  
someButton.getParent().validate();
```

Buttons (Continued)

- Event Processing Methods
 - addActionListener/removeActionListener
 - Add/remove an **ActionListener** that processes **ActionEvents** in **actionPerformed**
 - processActionEvent
 - Low-level event handling
- General Methods Inherited from Component
 - getForeground/setForeground
 - getBackground/setBackground
 - getFont/setFont

Button: Example

```
public class Buttons extends Applet {  
    private Button button1, button2, button3;  
    public void init() {  
        button1 = new Button("Button One");  
        button2 = new Button("Button Two");  
        button3 = new Button("Button Three");  
        add(button1);  
        add(button2);  
        add(button3);  
    }  
}
```

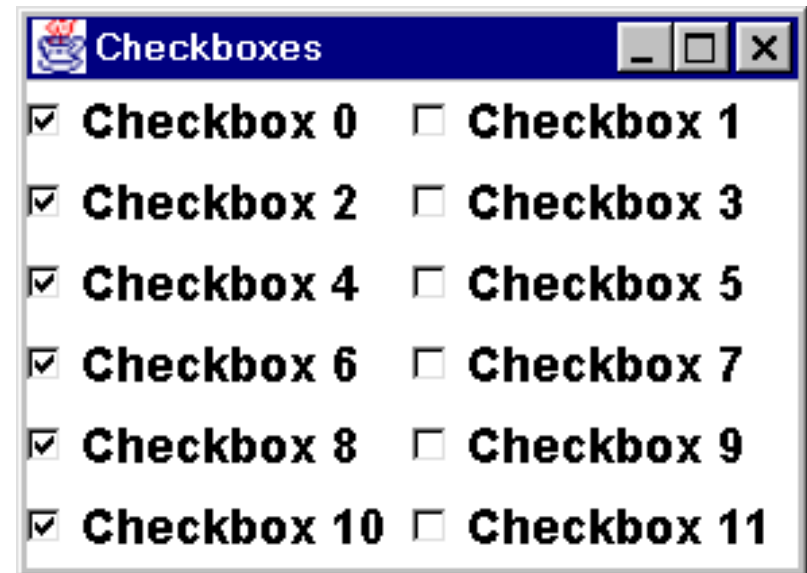


Checkboxes

- Constructors
 - These three constructors apply to **checkboxes that operate independently** of each other (i.e., not radio buttons)
 - `Checkbox()`
 - Creates an initially unchecked checkbox with no label
 - `Checkbox(String checkboxLabel)`
 - Creates a checkbox (initially unchecked) with the specified label; see `setState` for changing it
 - `Checkbox(String checkboxLabel, boolean state)`
 - Creates a checkbox with the specified label
 - The initial state is determined by the boolean value provided
 - A value of true means it is checked

Checkbox, Example

```
public class Checkboxes extends CloseableFrame
{
    public Checkboxes() {
        super("Checkboxes");
        setFont(new Font("SansSerif", Font.BOLD,
18));
        setLayout(new GridLayout(0, 2));
        Checkbox box;
        for(int i=0; i<12; i++) {
            box = new Checkbox("Checkbox " + i);
            if (i%2 == 0) {
                box.setState(true);
            }
            add(box);
        }
        pack();
        setVisible(true);
    }
}
```



Other Checkbox Methods

- getState/setState
 - Retrieves or sets the state of the checkbox: checked (true) or unchecked (false)
- getLabel/setLabel
 - Retrieves or sets the label of the checkbox
 - After changing the label invalidate and validate the window to force a new layout (same as button).
- addItemListener/removeItemListener
 - Add or remove an `ItemListener` to process `ItemEvents` in `itemStateChanged`

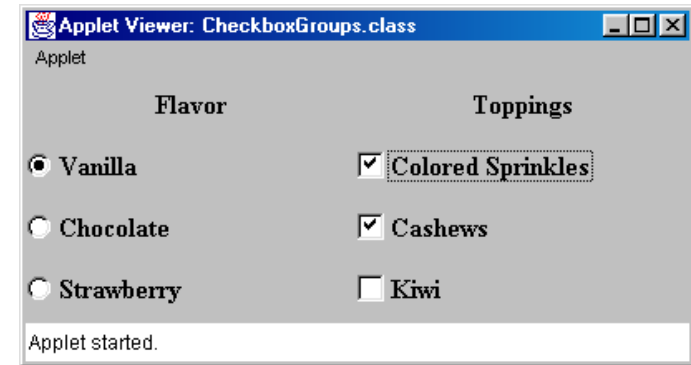
Checkbox Groups (Radio Buttons)

- CheckboxGroup Constructors
 - `CheckboxGroup()`
 - Creates a non-graphical object used as a “tag” to group checkboxes logically together
 - Only one checkbox associated with a particular tag can be selected at any given time
- Checkbox Constructors
 - `Checkbox(String label, CheckboxGroup group, boolean state)`
 - Creates a radio button associated with the specified group, with the given label and initial state

CheckboxGroup: Example

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

public class CheckboxGroups extends Applet {
    public void init() {
        setLayout(new GridLayout(4, 2));
        setBackground(Color.lightGray);
        setFont(new Font("Serif", Font.BOLD, 16));
        add(new Label("Flavor", Label.CENTER));
        add(new Label("Toppings", Label.CENTER));
        CheckboxGroup flavorGroup = new CheckboxGroup();
        add(new Checkbox("Vanilla", flavorGroup, true));
        add(new Checkbox("Colored Sprinkles"));
        add(new Checkbox("Chocolate", flavorGroup, false));
        add(new Checkbox("Cashews"));
        add(new Checkbox("Strawberry", flavorGroup, false));
        add(new Checkbox("Kiwi"));
    }
}
```



List Boxes

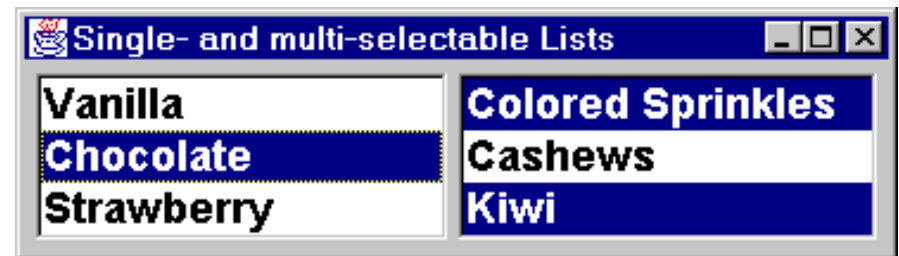
- Constructors

- `List(int rows, boolean multiSelectable)`
 - Creates a listbox with the specified number of **visible rows**
 - Depending on the number of item in the list (`addItem` or `add`), a scrollbar is automatically created
 - The second argument determines if the List is **multiselectable**
- `List()`
 - Creates a single-selectable list box with a platform-dependent number of rows and a platform-dependent width
- `List(int rows)`
 - Creates a single-selectable list box with the specified number of rows and a platform-dependent width

List Boxes: Example

```
import java.awt.*;

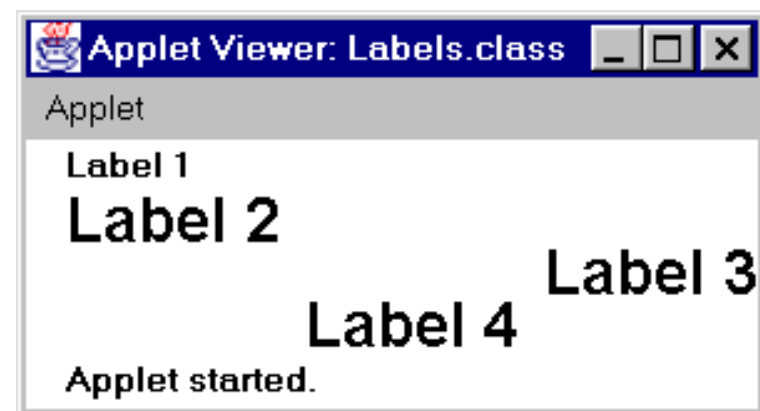
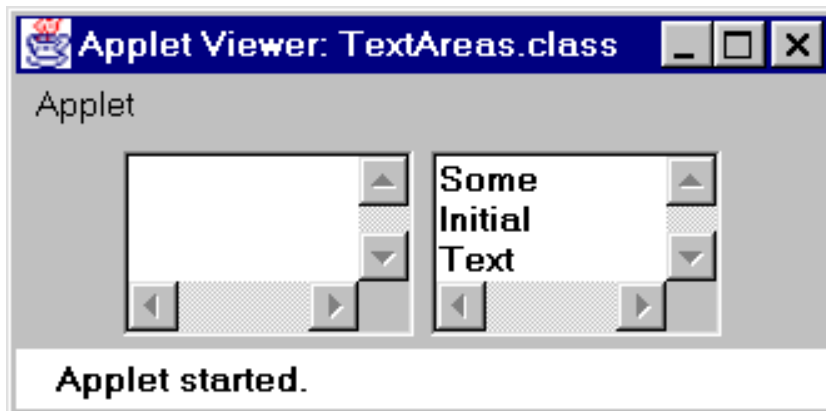
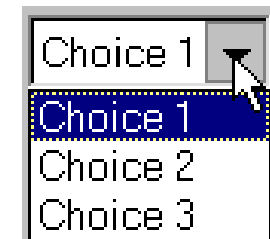
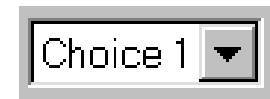
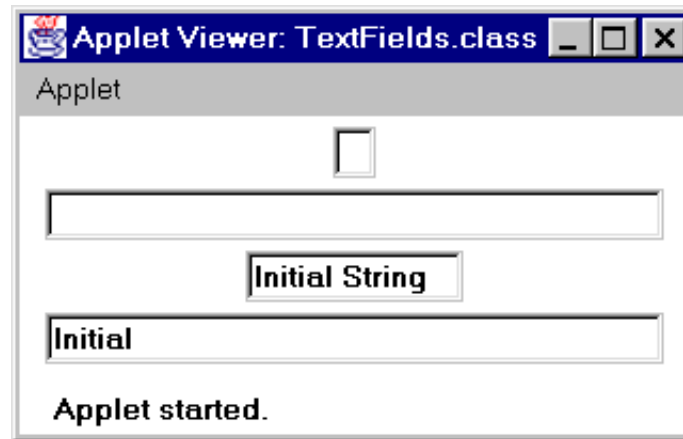
public class Lists extends CloseableFrame {
    public Lists() {
        super("Lists");
        setLayout(new FlowLayout());
        setBackground(Color.lightGray);
        setFont(new Font("SansSerif", Font.BOLD, 18));
        List list1 = new List(3, false);
        list1.add("Vanilla");
        list1.add("Chocolate");
        list1.add("Strawberry");
        add(list1);
        List list2 = new List(3, true);
        list2.add("Colored Sprinkles");
        list2.add("Cashews");
        list2.add("Kiwi");
        add(list2);
        pack();
        setVisible(true);
    }
}
```



A list can be *single-selectable* or *multi-selectable*

Other GUI Controls

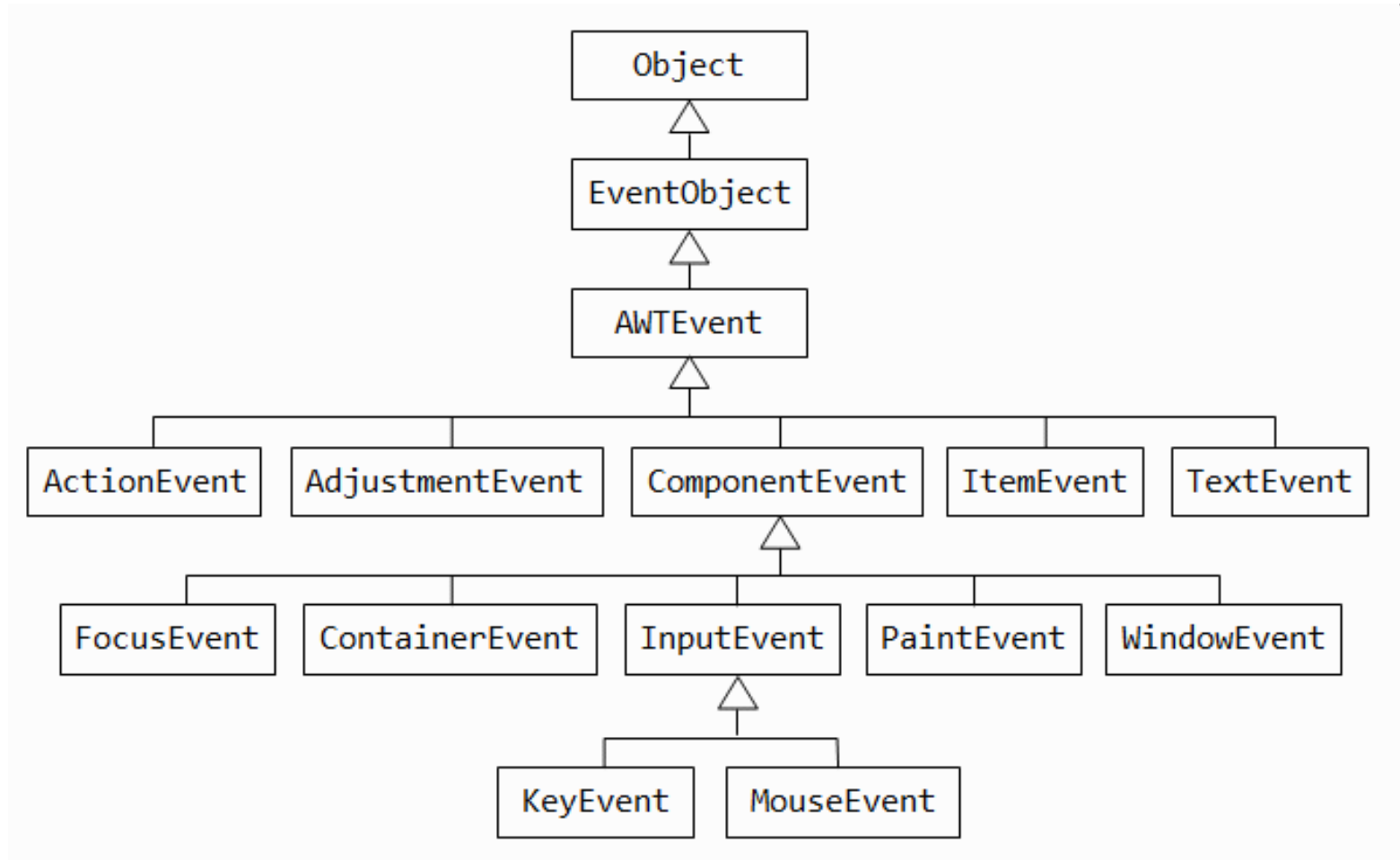
- Choice Lists (Combo Boxes)
- Textfields
- Text Areas
- Labels



Nội dung

1. Tổng quan về đồ họa
- ⇒ 2. Xử lý sự kiện
3. Quản lý bố cục (layout)
4. Java Swing

AWT Event Handling Hierarchy



Chiến lược xử lý sự kiện

- **Xác định loại lắng nghe (listener) cần thiết**
 - 11 loại lắng nghe chuẩn của AWT: ActionListener, AdjustmentListener, ComponentListener, ContainerListener, FocusListener, ItemListener, KeyListener, MouseListener, MouseMotionListener, TextListener, WindowListener
- **Định nghĩa một lớp cho loại lắng nghe đó**
 - Thực thi giao diện tương ứng, VD: KeyListener, MouseListener...
 - Kế thừa lớp tương ứng, VD: KeyAdapter, MouseAdapter,...
- **Đăng ký một đối tượng cho lớp lắng nghe của bạn với cửa sổ**
 - `w.addXxxListener(new MyListenerClass());`
 - Ví dụ `addKeyListener`, `addMouseListener`

Standard AWT Event Listeners

| Listener | Adapter Class (If Any) | Registration Method |
|---|--|--|
| ActionListener AdjustmentListener ComponentListener ContainerListener FocusListener ItemListener KeyListener MouseListener MouseMotionListener TextListener WindowListener | ComponentAdapter ContainerAdapter FocusAdapter KeyAdapter MouseAdapter MouseMotionAdapter WindowAdapter | addActionListener addAdjustmentListener addComponentListener addContainerListener addFocusListener addItemListener addKeyListener addMouseListener addMouseMotionListener addTextListener addWindowListener |

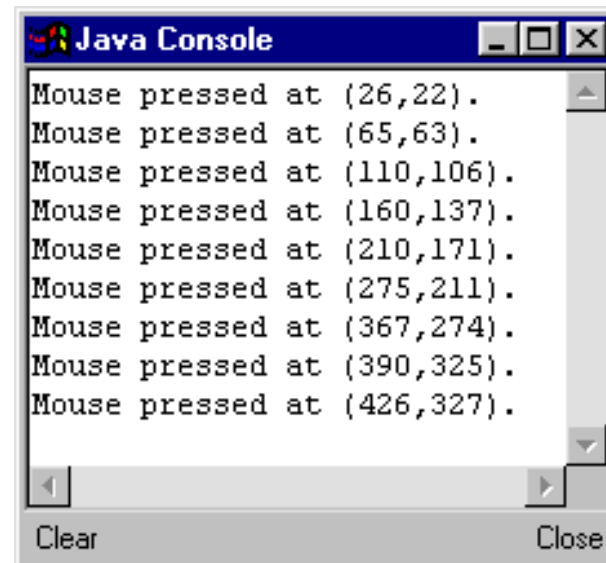
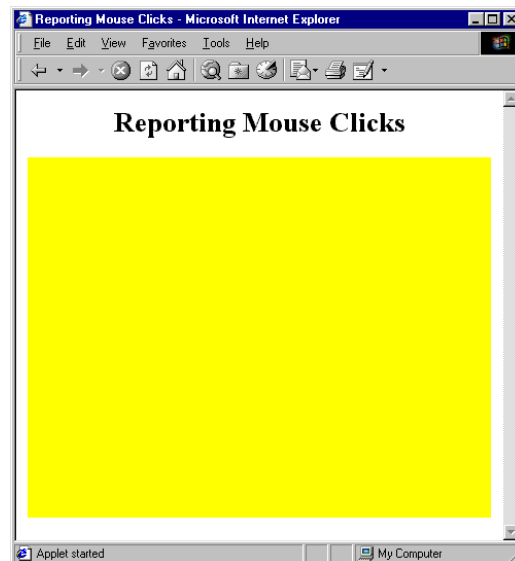
Lớp Listener riêng – VD đơn giản

Listener không cần gọi bất kỳ phương thức nào của cửa sổ mà nó thuộc về

```
import java.applet.Applet;  
import java.awt.*;  
  
public class ClickReporter extends Applet {  
    public void init() {  
        setBackground(Color.yellow);  
        addMouseListener(new ClickListener());  
    }  
}
```

Lớp Listener riêng – VD đơn giản (2)

```
import java.awt.event.*;
public class ClickListener extends MouseAdapter {
    public void mousePressed(MouseEvent event) {
        System.out.println("Mouse pressed at (" +
            event.getX() + "," + event.getY() + ").");
    }
}
```



Tổng quát hóa trường hợp đơn giản

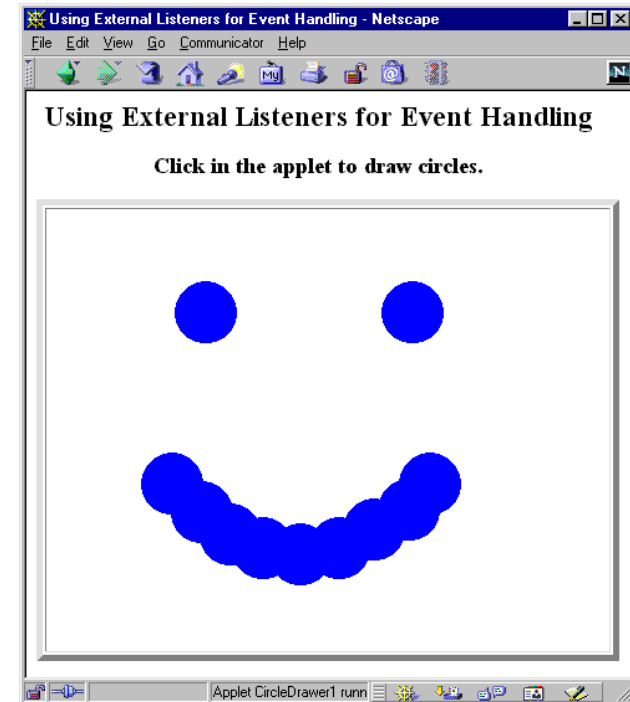
- Nếu ClickListener muốn vẽ một hình tròn tại vị trí chuột được nhấn?
- Tại sao không thể gọi phương thức `getGraphics` để lấy về đối tượng `Graphics` để vẽ
- Giải pháp tổng quát:
 - Gọi `event.getSource` để lấy về một tham chiếu tới cửa sổ hoặc thành phần GUI tạo ra sự kiện
 - Ép kết quả trả về theo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

Lớp Listener riêng – VD tổng quát

```
import java.applet.Applet;  
import java.awt.*;  
  
public class CircleDrawer1 extends Applet {  
    public void init() {  
        setForeground(Color.blue);  
        addMouseListener(new CircleListener());  
    }  
}
```

Lớp Listener riêng – VD tổng quát (2)

```
import java.applet.Applet;  
import java.awt.*;  
import java.awt.event.*;  
public class CircleListener extends MouseAdapter {  
    private int radius = 25;  
    public void mousePressed(MouseEvent event) {  
        Applet app = (Applet)event.getSource();  
        Graphics g = app.getGraphics();  
        g.fillOval(event.getX()-radius,  
event.getY()-radius,  
                2*radius, 2*radius);  
    }  
}
```



Cách 2: Thực thi giao diện Listener

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

public class CircleDrawer2 extends Applet
    implements MouseListener {
    private int radius = 25;
    public void init() {
        setForeground(Color.blue);
        addMouseListener(this);
    }
}
```

Cách 2: Thực thi giao diện Listener (2)

```
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);  
}  
}
```

Chiến thuật xử lý sự kiện

Ưu và nhược điểm

- Tách riêng lớp Listener
 - **Ưu điểm**
 - Có thể kế thừa lớp Adapter và vì thế có thể không cần quan tâm đến các phương thức không sử dụng
 - Sử dụng lớp Listener riêng thì dễ quản lý hơn
 - **Nhược điểm**
 - Cần thêm bước gọi các phương thức trong cửa sổ chính
- Cửa sổ chính thực thi giao diện Listener
 - **Ưu điểm**
 - Không cần có thêm bước gọi các phương thức trong cửa sổ chính
 - **Nhược điểm**
 - Phải thực thi tất cả các phương thức của giao diện, kể cả các phương thức không quan tâm

Standard AWT Event Listeners

(Tổng kết)

| Listener | Adapter Class (If Any) | Registration Method |
|---|--|--|
| ActionListener AdjustmentListener ComponentListener ContainerListener FocusListener ItemListener KeyListener MouseListener MouseMotionListener TextListener WindowListener | ComponentAdapter ContainerAdapter FocusAdapter KeyAdapter MouseAdapter MouseMotionAdapter WindowAdapter | addActionListener addAdjustmentListener addComponentListener addContainerListener addFocusListener addItemListener addKeyListener addMouseListener addMouseMotionListener addTextListener addWindowListener |

Standard AWT Event Listeners

(Chi tiết)

- **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)`

Standard AWT Event Listeners

(Chi tiết – 2)

- **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)`

Standard AWT Event Listeners

(Chi tiết – 3)

- **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

Standard AWT Event Listeners

(Chi tiết – 4)

- **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)`

Standard AWT Event Listeners

(Chi tiết – 5)

- **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` particularly useful

Ví dụ: Simple Whiteboard

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

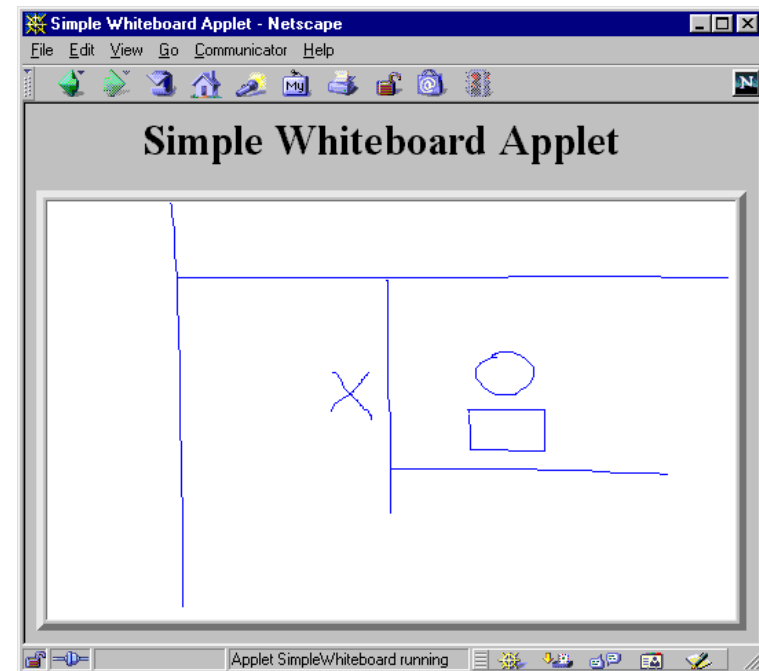
public class SimpleWhiteboard extends Applet {
    protected int lastX=0, lastY=0;
    public void init() {
        setBackground(Color.white);
        setForeground(Color.blue);
        addMouseListener(new PositionRecorder());
        addMouseMotionListener(new LineDrawer());
    }
    protected void record(int x, int y) {
        lastX = x; lastY = y;
    }
}
```

Ví dụ: Simple Whiteboard (2)

```
private class PositionRecorder extends MouseAdapter {  
    public void mouseEntered(MouseEvent event) {  
        requestFocus(); // Plan ahead for typing  
        record(event.getX(), event.getY());  
    }  
  
    public void mousePressed(MouseEvent event) {  
        record(event.getX(), event.getY());  
    }  
}  
...
```


Ví dụ: Simple Whiteboard (3)

```
...
private class LineDrawer extends MouseMotionAdapter
{
    public void mouseDragged(MouseEvent event) {
        int x = event.getX();
        int y = event.getY();
        Graphics g = getGraphics();
        g.drawLine(lastX, lastY, x, y);
        record(x, y);
    }
}
}
```



Whiteboard: Thêm các sự kiện bàn phím

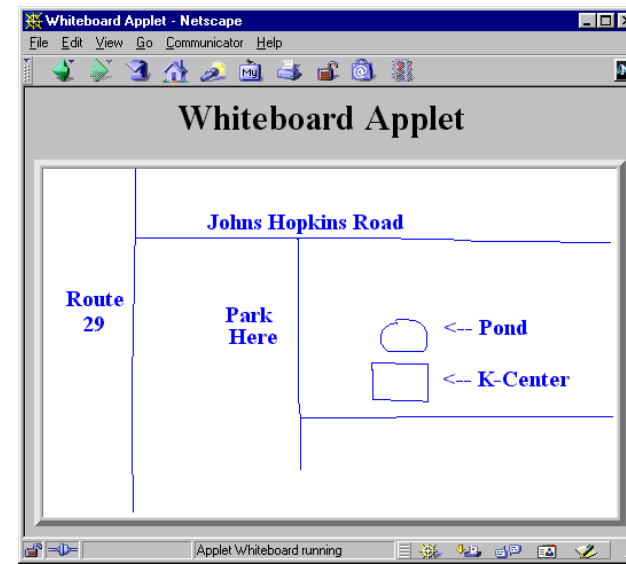
```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Whiteboard extends SimpleWhiteboard {
    protected FontMetrics fm;
    public void init() {
        super.init();
        Font font = new Font("Serif", Font.BOLD, 20);
        setFont(font);
        fm = getFontMetrics(font);
        addKeyListener(new CharDrawer());
    }
}
```

Whiteboard: Thêm các sự kiện bàn phím (2)

...

```
private class CharDrawer extends KeyAdapter {
    // When user types a printable character,
    // draw it and shift position rightwards.
    public void keyTyped(KeyEvent event) {
        String s =
String.valueOf(event.getKeyChar());
        getGraphics().drawString(s, lastX, lastY);
        record(lastX + fm.stringWidth(s), lastY);
    }
}
```



Nội dung

1. Tổng quan về đồ họa

2. Xử lý sự kiện

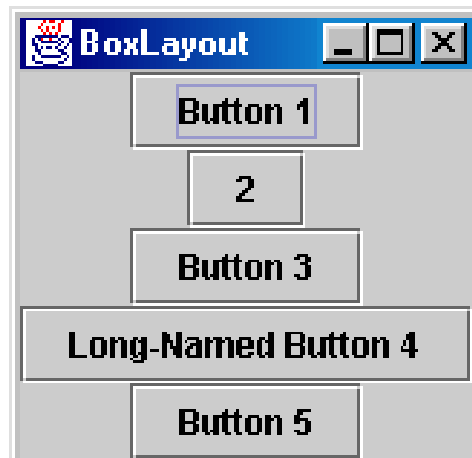
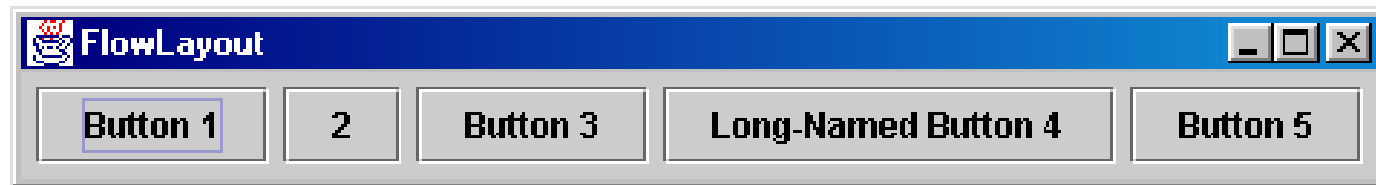
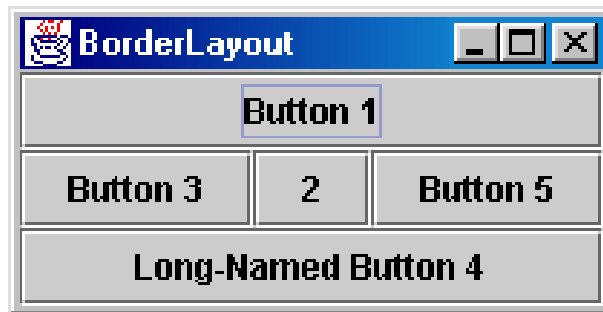
⇒ 3. Quản lý bố cục (layout)

4. Java Swing

Tổng quan về quản lý bố cục

- Làm thế nào để các bộ quản lý layout (layout manager) đơn giản hóa việc thiết kế giao diện người dùng?
- Sử dụng các layout manager chuẩn
 - `FlowLayout`, `BorderLayout`, `CardLayout`, `GridLayout`, `GridBagLayout`, `BoxLayout`
- Định vị trí cho các thành phần bằng tay
- Các chiến thuật để sử dụng layout manager hiệu quả
- Sử dụng các thành phần invisible

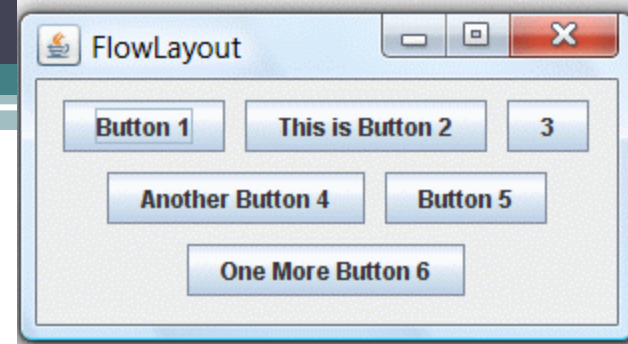
Layout Manager



Layout Manager (2)

- Assigned to each Container
 - Give *sizes* and *positions* to components in the window
 - Helpful for windows whose size changes or that display on multiple operating systems
- Relatively easy for simple layouts
 - But, it is surprisingly hard to get complex layouts with a single layout manager
- Controlling complex layouts
 - Use nested containers (each with its own layout manager)
 - Use invisible components and layout manager options
 - Write your own layout manager
 - Turn layout managers off and arrange things manually

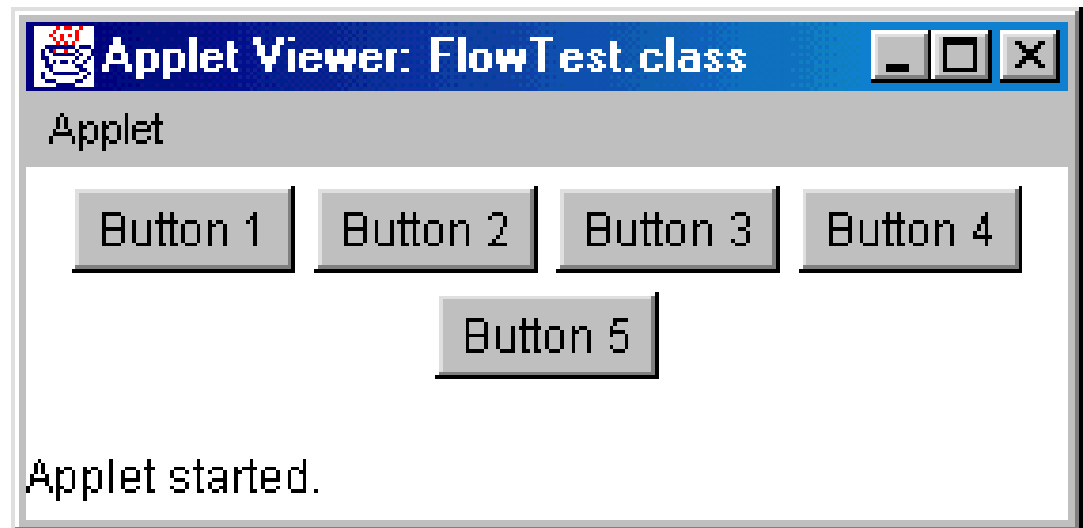
FlowLayout



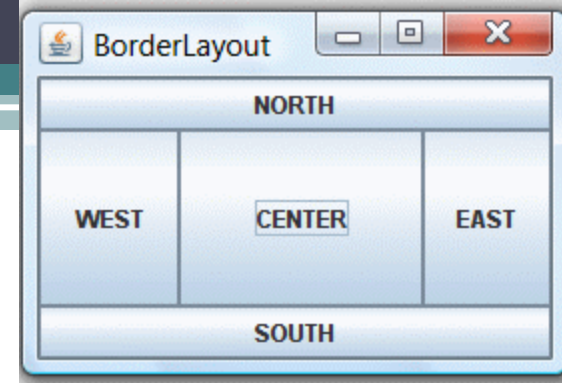
- Là layout mặc định cho **Panel** và **Applet**
- Behavior
 - Đưa các thành phần trở lại kích thước được ưa thích (preferred)
 - Đặt các thành phần theo các dòng từ trái sang phải, từ trên xuống dưới
 - Rows are centered by default
- Constructors
 - **FlowLayout()**
 - Centers each row and keeps 5 pixels between entries in a row and between rows
 - **FlowLayout(int alignment)**
 - Same 5 pixels spacing, but changes the alignment of the rows
 - `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`
 - **FlowLayout(int alignment, int hGap, int vGap)**
 - Specify the alignment as well as the horizontal and vertical spacing between components

FlowLayout: Example

```
public class FlowTest extends Applet {  
    public void init() {  
        // setLayout(new FlowLayout());[Default]  
        for(int i=1; i<6; i++) {  
            add(new Button("Button " + i));  
        }  
    }  
}
```



BorderLayout



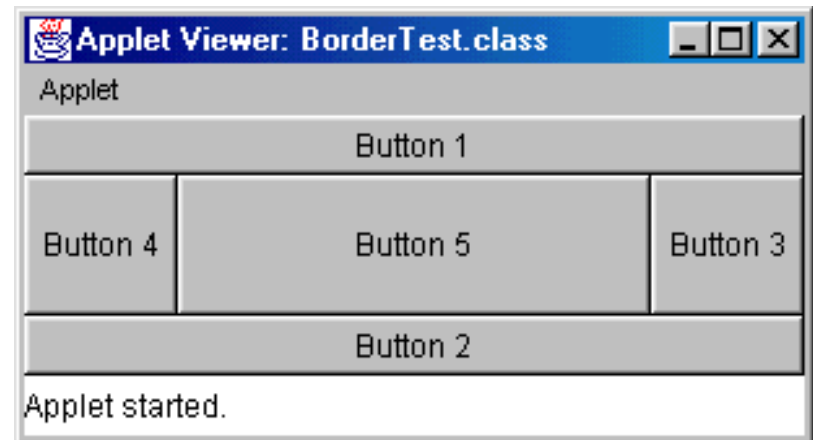
- Là layout mặc định cho **Frame** và **Dialog**
- Behavior
 - Chia Container thành 5 phần
 - Mỗi vùng được xác định thông qua một hằng tương ứng trong BorderLayout.
 - NORTH, SOUTH, EAST, WEST, and CENTER
 - NORTH và SOUTH respect the preferred height of the component
 - EAST and WEST respect the preferred width of the component
 - CENTER is given the remaining space
- Chỉ cho phép gồm tối đa 5 thành phần liệu có quá hạn chế không? Tại sao?

BorderLayout (Continued)

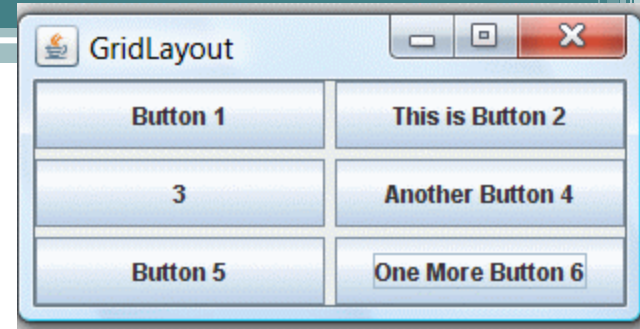
- Constructors
 - `BorderLayout()`
 - Border layout with no gaps between components
 - `BorderLayout(int hGap, int vGap)`
 - Border layout with the specified empty pixels between regions
- Adding Components
 - `add(component, BorderLayout.REGION)`
 - Always specify the region in which to add the component
 - `CENTER` is the default, but specify it explicitly to avoid confusion with other layout managers

BorderLayout: Example

```
public class BorderTest extends Applet {  
    public void init() {  
        setLayout(new BorderLayout());  
        add(new Button("Button 1"), BorderLayout.NORTH);  
        add(new Button("Button 2"), BorderLayout.SOUTH);  
        add(new Button("Button 3"), BorderLayout.EAST);  
        add(new Button("Button 4"), BorderLayout.WEST);  
        add(new Button("Button 5"), BorderLayout.CENTER);  
    }  
}
```



GridLayout



- **Behavior**

- Divides window into **equal-sized rectangles** based upon the **number of rows and columns specified**
- Items placed into cells left-to-right, top-to-bottom, based on the order added to the container
- Ignores the preferred size of the component; each component is **resized to fit into its grid cell**
- Too few components results in blank cells
- Too many components results in extra columns

GridLayout (Continued)

- Constructors

- `GridLayout()`

- Creates a single row with one column allocated per component

- `GridLayout(int rows, int cols)`

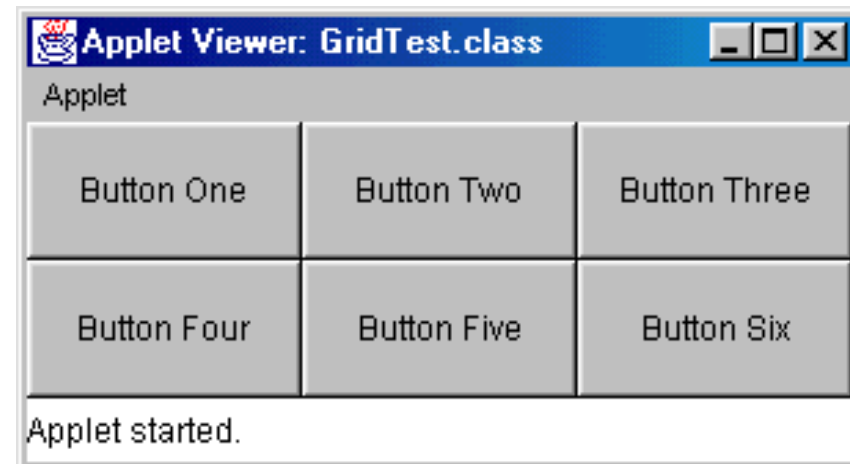
- Divides the window into the specified number of rows and columns
 - Either rows or cols (but not both) can be zero

- `GridLayout(int rows, int cols, int hGap, int vGap)`

- Uses the specified gaps between cells

GridLayout, Example

```
public class GridTest extends Applet {  
    public void init() {  
        setLayout(new GridLayout(2,3)); //2 rows, 3 cols  
        add(new Button("Button One"));  
        add(new Button("Button Two"));  
        add(new Button("Button Three"));  
        add(new Button("Button Four"));  
        add(new Button("Button Five"));  
        add(new Button("Button Six"));  
    }  
}
```



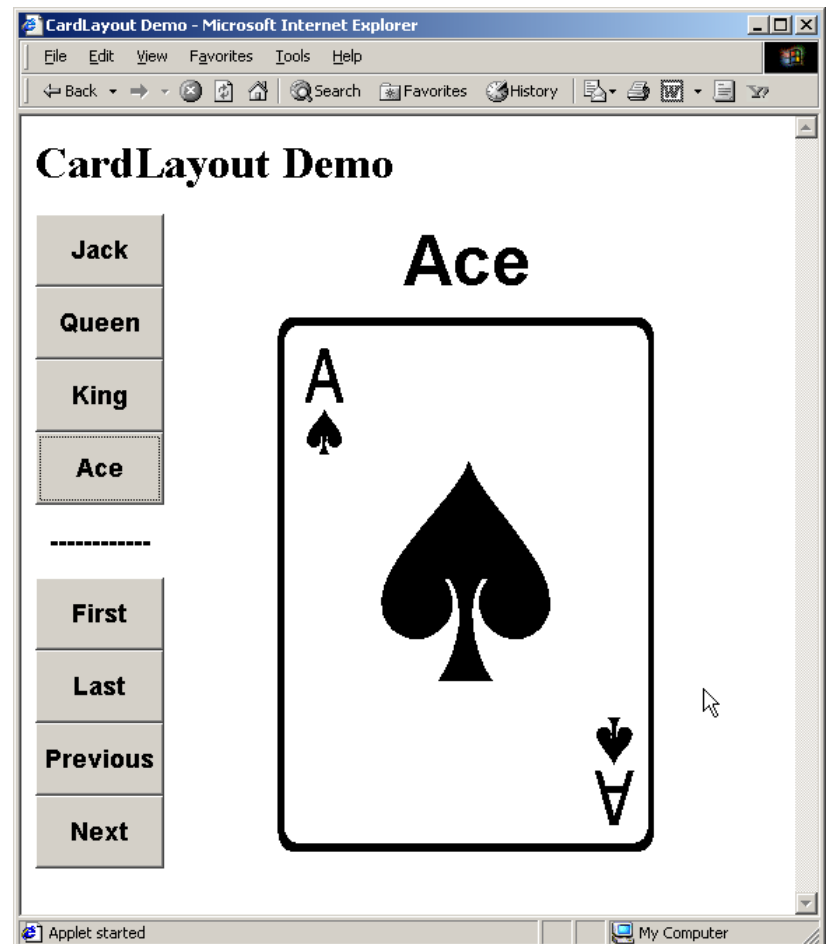
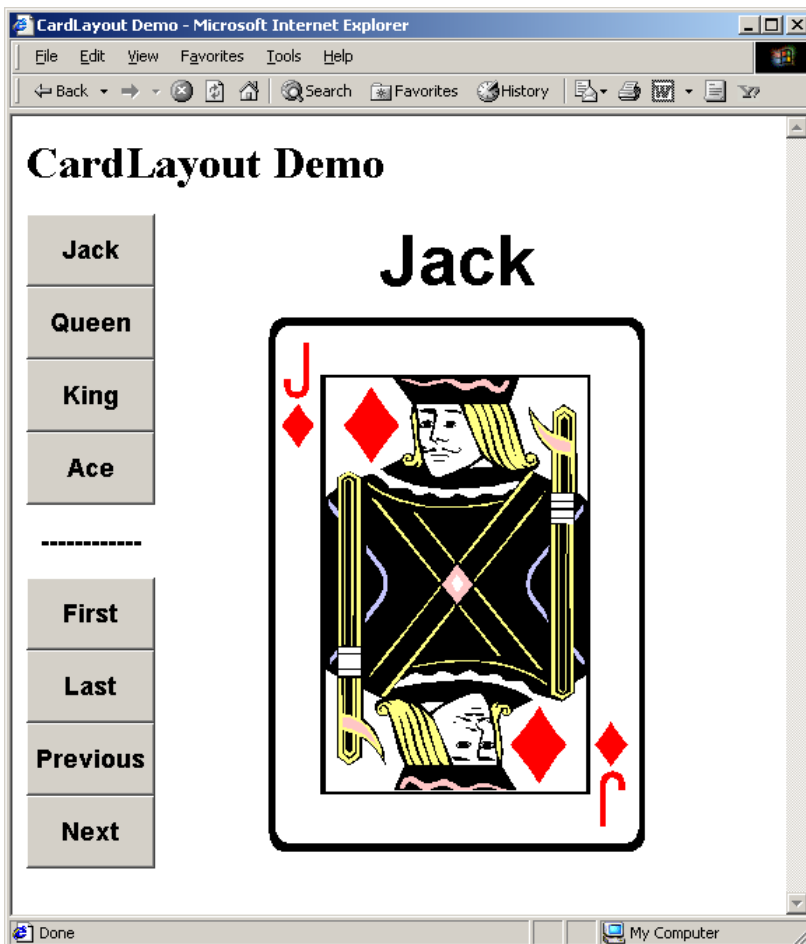
CardLayout

- Behavior

- Stacks components on top of each other, displaying the top one
- Associates a name with each component in window

```
Panel cardPanel;  
CardLayout layout new CardLayout();  
cardPanel.setLayout(layout);  
...  
cardPanel.add("Card 1", component1);  
cardPanel.add("Card 2", component2);...  
layout.show(cardPanel, "Card 1");  
layout.first(cardPanel);  
layout.next(cardPanel);
```


CardLayout, Example



GridBagLayout



- **Behavior**

- Divides the window into **grids**, without requiring the components to be the same size
 - About **three** times more flexible than the other standard layout managers, but **nine** times harder to use
- Each component managed by a grid bag layout is associated with an instance of **GridBagConstraints**
 - The `GridBagConstraints` specifies:
 - How the component is laid out in the display area
 - In which cell the component starts and ends
 - How the component stretches when extra room is available
 - Alignment in cells

GridBagLayout



- Set the layout, saving a reference to it
`GridBagLayout layout = new GridBagLayout();`
`setLayout(layout);`
- Allocate a `GridBagConstraints` object
`GridBagConstraints constraints = new GridBagConstraints();`
- Set up the `GridBagConstraints` for component 1
`constraints.gridx = x1;`
`constraints.gridy = y1;`
`constraints.gridwidth = width1;`
`constraints.gridheight = height1;`
- Add component 1 to the window, including constraints
`add(component1, constraints);`
- Repeat the last two steps for each remaining component

GridBagConstraints

- Copied when component added to window
- Thus, can reuse the `GridBagConstraints`

```
GridBagConstraints constraints = new GridBagConstraints();
constraints.gridx = x1;
constraints.gridy = y1;
constraints.gridwidth = width1;
constraints.gridheight = height1;
add(component1, constraints);
constraints.gridx = x1;
constraints.gridy = y1;
add(component2, constraints);
```

GridBagConstraints Fields

- **gridx, gridy**
 - Specifies the top-left corner of the component
 - Upper left of grid is located at (gridx, gridy)=(0,0)
 - Set to **GridBagConstraints.RELATIVE** to auto-increment row/column

```
GridBagConstraints constraints = new  
GridBagConstraints();  
constraints.gridx = GridBagConstraints.RELATIVE;  
container.add(new Button("one"), constraints);  
container.add(new Button("two"), constraints);
```

GridBagConstraints Fields (Continued)

- **gridwidth, gridheight**

- Specifies the number of columns and rows the Component occupies

constraints.gridwidth = 3;

- **GridBagConstraints.REMAINDER** lets the component take up the remainder of the row/column

- **weightx, weighty**

- Specifies how much the cell will stretch in the x or y direction if space is left over

constraints.weightx = 3.0;

- Constraint affects the cell, not the component (use `fill`)
- Use a value of 0.0 for no expansion in a direction
- Values are relative, not absolute

GridBagConstraints Fields (Continued)

- **fill**

- Specifies what to do to an element that is smaller than the cell size

```
constraints.fill = GridBagConstraints.VERTICAL;
```

- The size of row/column is determined by the widest/tallest element in it
- Can be NONE, HORIZONTAL, VERTICAL, or BOTH

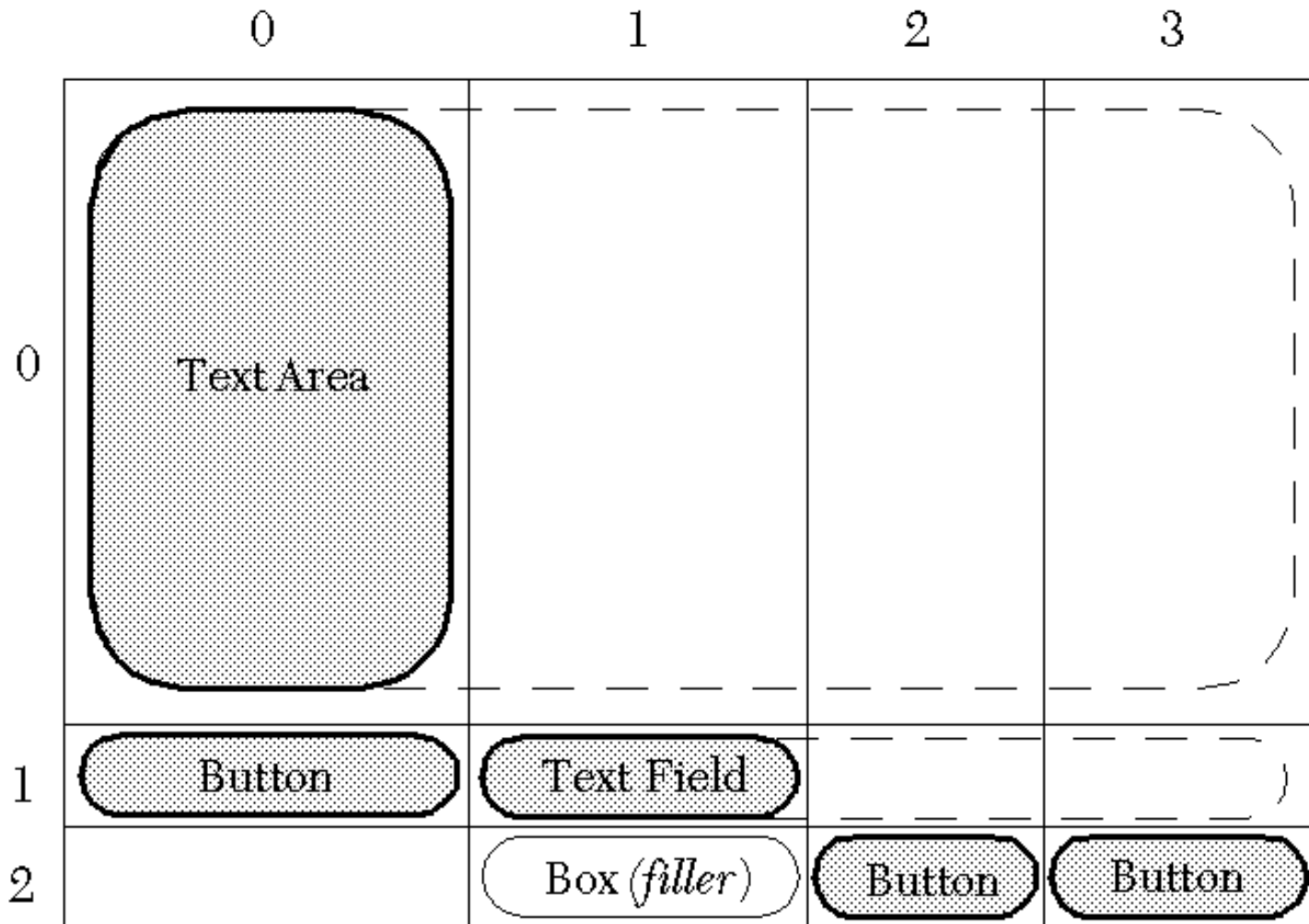
- **anchor**

- If the fill is set to GridBagConstraints.NONE, then the anchor field determines where the component is placed

```
constraints.anchor = GridBagConstraints.NORTHEAST;
```

- Can be NORTH, EAST, SOUTH, WEST, NORTHEAST, NORTHWEST, SOUTHEAST, or SOUTHWEST

GridBagLayout: Example



GridBagLayout, Example

```

public GridBagTest() {
    setLayout(new GridBagLayout());
    textArea = new JTextArea(12, 40);
    // 12 rows, 40 cols
    bSaveAs = new JButton("Save As");
    fileField = new
JTextField("C:\\Document.txt");
    bOk = new JButton("OK");
    bExit = new JButton("Exit");
    GridBagConstraints c = new
GridBagConstraints();
    // Text Area.
    c.gridx      = 0;
    c.gridy      = 0;
    c.gridwidth  = GridBagConstraints.REMAINDER;
    c.gridheight = 1; c.weightx      = 1.0;
    c.weighty    = 1.0;
    c.fill       = GridBagConstraints.BOTH;
    c.insets     = new Insets(2,2,2,2);
    add(textArea, c);
    ...

```

```

// Save As Button.
c.gridx      = 0;
c.gridy      = 1;
c.gridwidth  = 1;
c.gridheight = 1;
c.weightx    = 0.0;
c.weighty    = 0.0;
c.fill       =
    GridBagConstraints.VERTICAL;
add(bSaveAs, c);

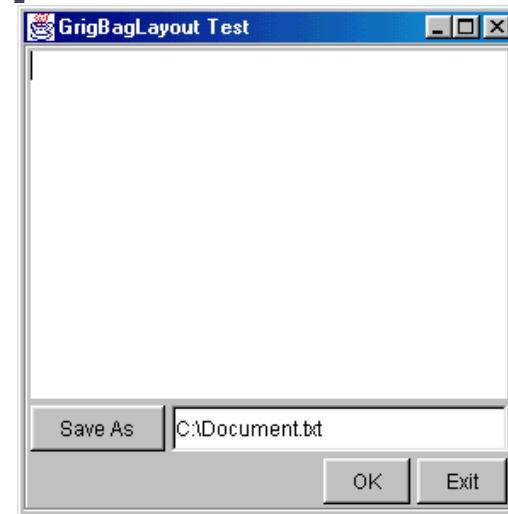
// Filename Input (Textfield).
c.gridx      = 1;
c.gridwidth  =
    GridBagConstraints.REMAINDER
    ;
c.gridheight = 1;
c.weightx    = 1.0;
c.weighty    = 0.0;
c.fill       =
    GridBagConstraints.BOTH;
add(fileField, c);
...

```

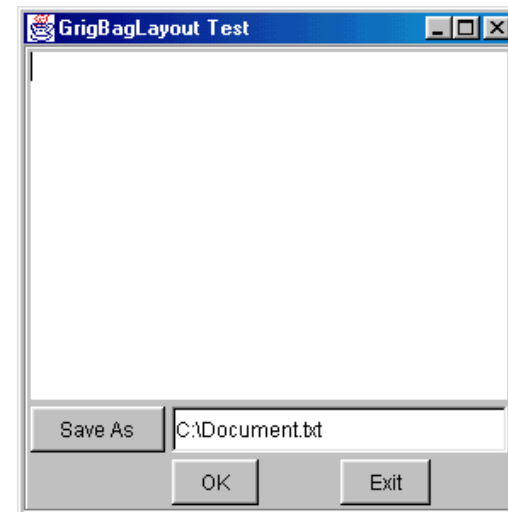
GridBagLayout, Example

```
// Exit Button.
c.gridx      = 3;
c.gridwidth  = 1;
c.gridheight = 1;
c.weightx    = 0.0;
c.weighty    = 0.0;
c.fill       =
    GridBagConstraints.NONE;
add(bExit,c);
```

```
// Filler so Column 1 has
    nonzero width.
Component filler =
    Box.createRigidArea(new
        Dimension(1,1));
c.gridx      = 1;
c.weightx    = 1.0;
add(filler,c);
...
}
```



With / Without Box *filler* at (2,1)

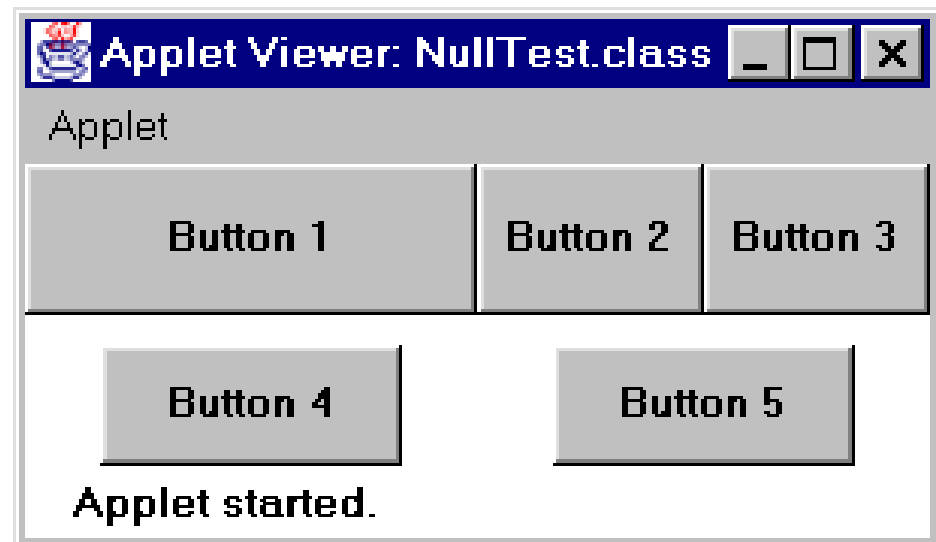


Disabling the Layout Manager

- Behavior
 - If the layout is set to `null`, then components must be sized and positioned by hand
- Positioning components
 - `component.setSize(width, height)`
 - `component.setLocation(left, top)`
- or
 - `component.setBounds(left, top, width, height)`

No Layout Manager, Example

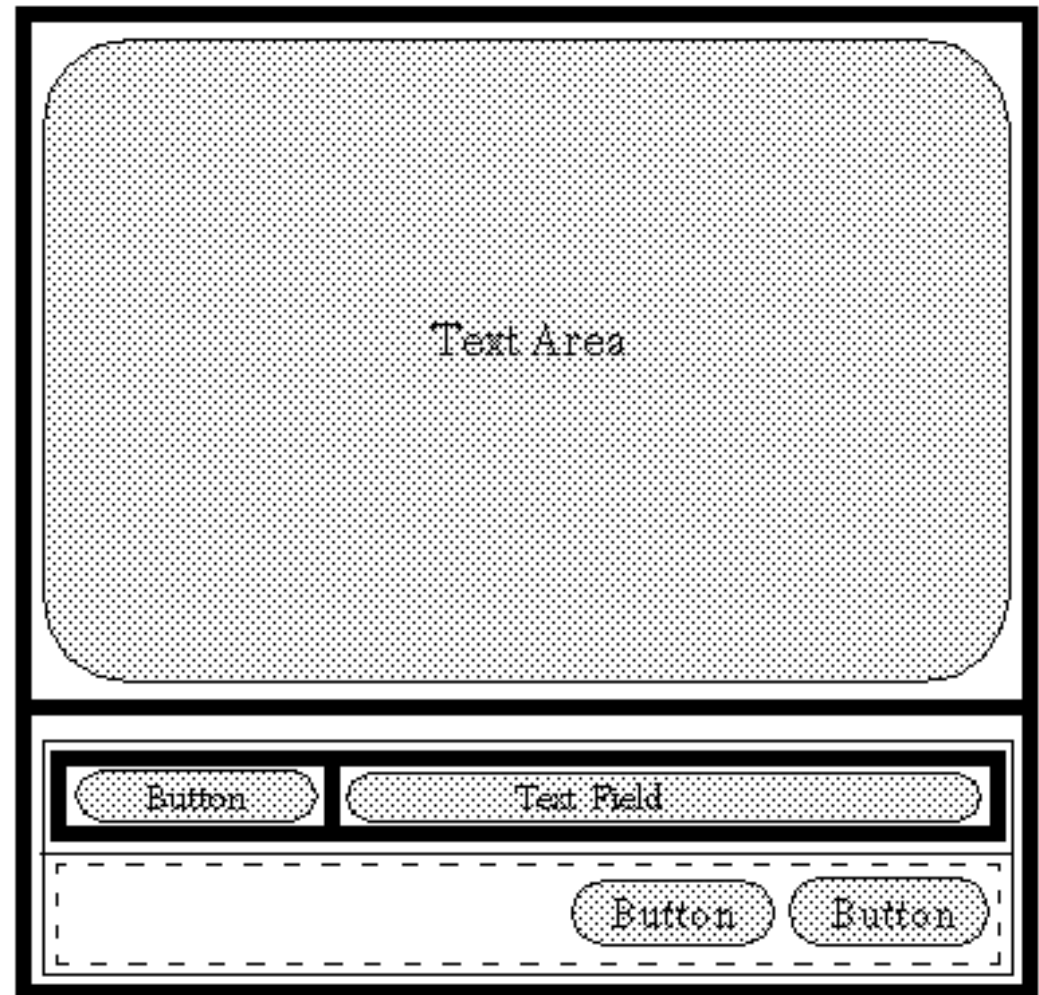
```
setLayout(null);  
Button b1 = new Button("Button 1");  
Button b2 = new Button("Button 2");  
...  
b1.setBounds(0, 0, 150, 50);  
b2.setBounds(150, 0, 75, 50);  
...  
add(b1);  
add(b2);  
...
```



Using Layout Managers Effectively

- Use nested containers
 - Rather than struggling to fit your design in a single layout, try dividing the design into sections
 - Let each section be a panel with its own layout manager
- Turn off the layout manager for some containers
- Adjust the empty space around components
 - Change the space allocated by the layout manager
 - Override `insets` in the `Container`
 - Use a **Canvas** or a **Box** as an invisible spacer

Nested Containers, Example



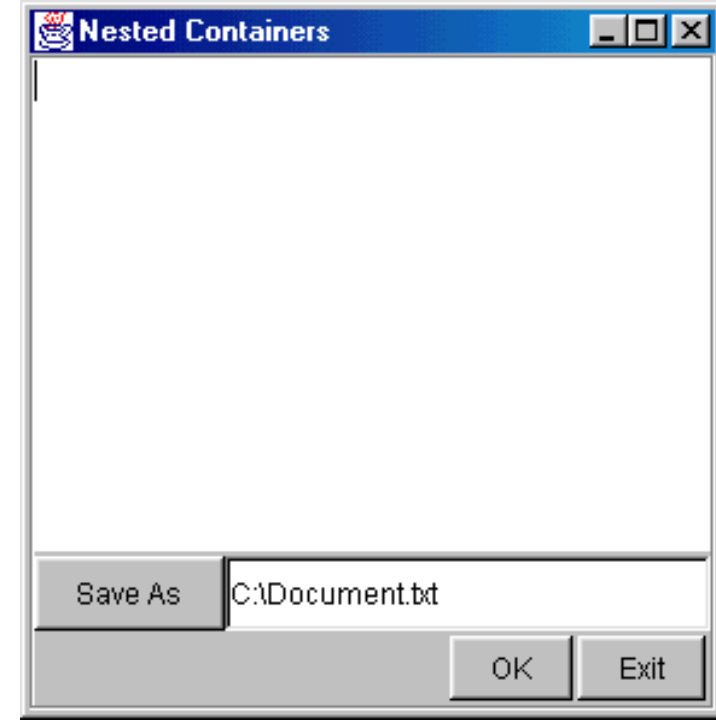
———— BorderLayout
----- FlowLayout
———— GridLayout

Nested Containers, Example

```
public NestedLayout() {  
    setLayout(new BorderLayout(2,2));  
    textArea = new JTextArea(12,40); // 12 rows, 40 cols  
    bSaveAs = new JButton("Save As");  
    fileField = new JTextField("C:\\\\Document.txt");  
    bOk = new JButton("OK");  
    bExit = new JButton("Exit");  
    add(textArea,BorderLayout.CENTER);  
    // Set up buttons and textfield in bottom panel.  
    JPanel bottomPanel = new JPanel();  
    bottomPanel.setLayout(new GridLayout(2,1));
```

Nested Containers, Example

```
JPanel subPanel1 = new JPanel();  
    JPanel subPanel2 = new JPanel();  
    subPanel1.setLayout(new BorderLayout());  
    subPanel2.setLayout(new FlowLayout(FlowLayout.RIGHT,2,2));  
  
    subPanel1.add(bSaveAs,BorderLayout.WEST);  
    subPanel1.add(fileField,BorderLayout.CENTER);  
    subPanel2.add(bOk);  
    subPanel2.add(bExit);  
  
    bottomPanel.add(subPanel1);  
    bottomPanel.add(subPanel2);  
  
    add(bottomPanel,BorderLayout.SOUTH);  
}
```

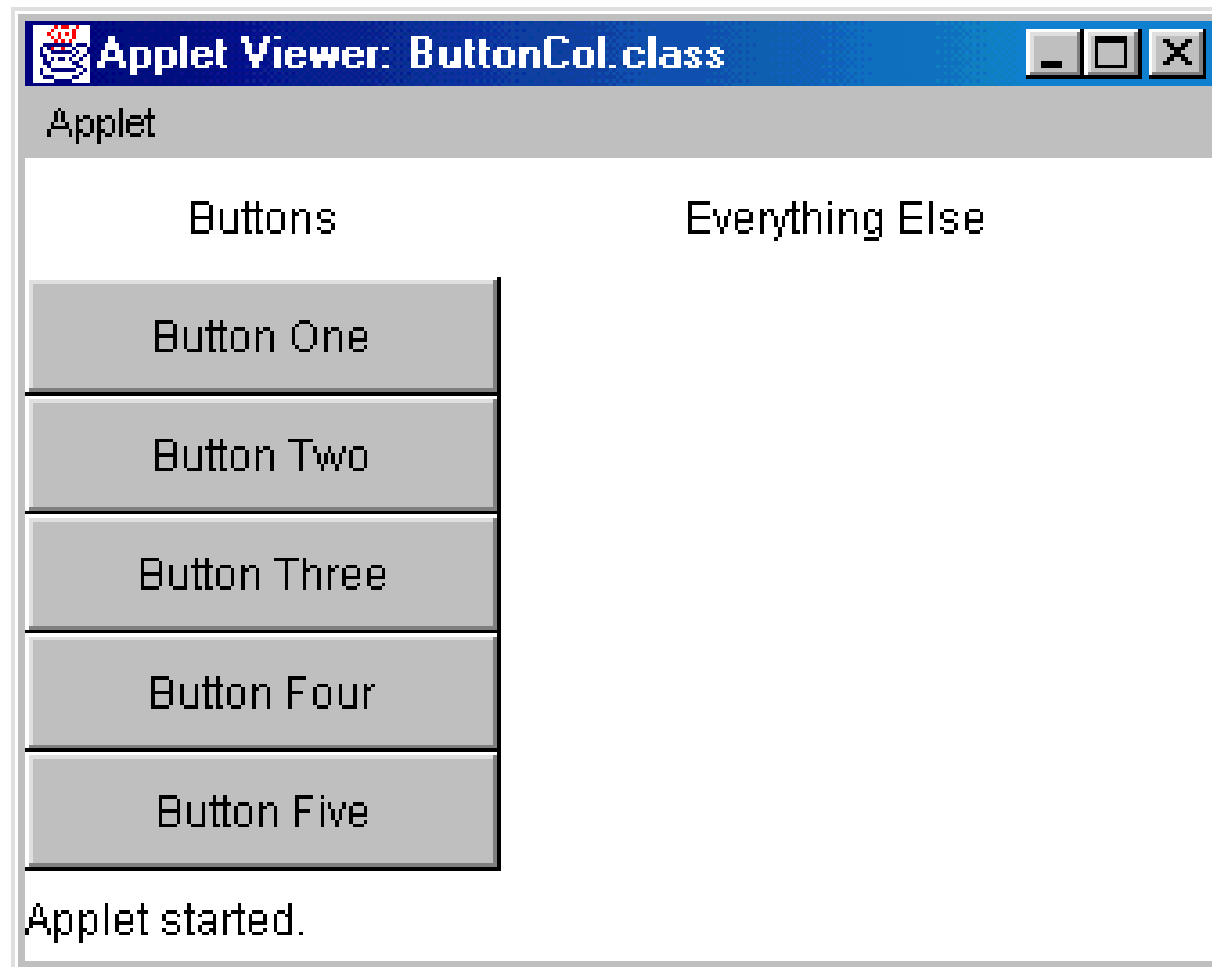


Turning Off Layout Manager for Some Containers, Example

- Suppose that you wanted to arrange a column of buttons (on the left) that take **exactly 40%** of the width of the container

```
setLayout(null);  
int width1 = getSize().width*4/10;  
int height = getSize().height;  
Panel buttonPanel = new Panel();  
buttonPanel.setBounds(0, 0, width1, height);  
buttonPanel.setLayout(new GridLayout(6, 1));  
buttonPanel.add(new Label("Buttons", Label.CENTER));  
buttonPanel.add(new Button("Button One"));  
...  
buttonPanel.add(new Button("Button Five"));  
add(buttonPanel);  
Panel everythingElse = new Panel();  
int width2 = getSize().width - width1,  
everythingElse.setBounds(width1+1, 0, width2, height);
```

Turning Off Layout Manager for Some Containers: Result



BoxLayout

- Behavior
 - Manager from Swing; available only in Java 2
 - Arranges Components either in a horizontal row, `BoxLayout.X_AXIS`, or in a vertical column, `BoxLayout.Y_AXIS`
 - Lays out the components in the order in which they were added to the Container
 - Resizing the container does not cause the components to relocate
 - Unlike the other standard layout managers, the `BoxLayout` manager cannot be shared with more than one Container

```
BoxLayout layout =  
    new BoxLayout(container, BoxLayout.X_AXIS);
```

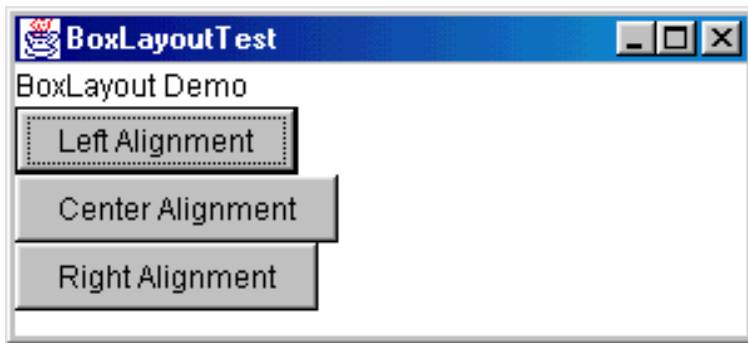
Component Arrangement for BoxLayout

- Attempts to arrange the components with:
 - Their preferred widths (vertical layout), or
 - Their preferred heights (horizontal layout)
- Vertical Layout
 - If the components are not all the same width, `BoxLayout` attempts to expand all the components to the width of the component with the largest preferred width
 - If expanding a component is not possible (restricted maximum size), **BoxLayout** aligns that component horizontally in the container, according to the x alignment of the component

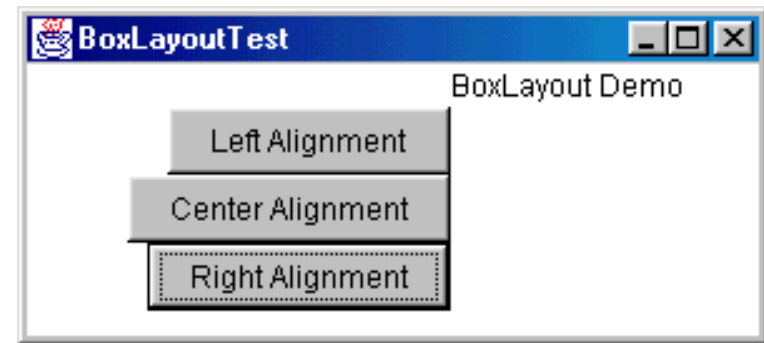
Component Arrangement for BoxLayout (Continued)

- Horizontal Layout
 - If the components are not all the same height, `BoxLayout` attempts to expand all the components to the height of the tallest component
 - If expanding the height of a component is not possible, **`BoxLayout`** aligns that component vertically in the container, according to the y alignment of the component.

BoxLayout: Example



- All components have a 0.0 (left) alignment



- The label has a 0.0 alignment
- The buttons have a 1.0 (right) alignment

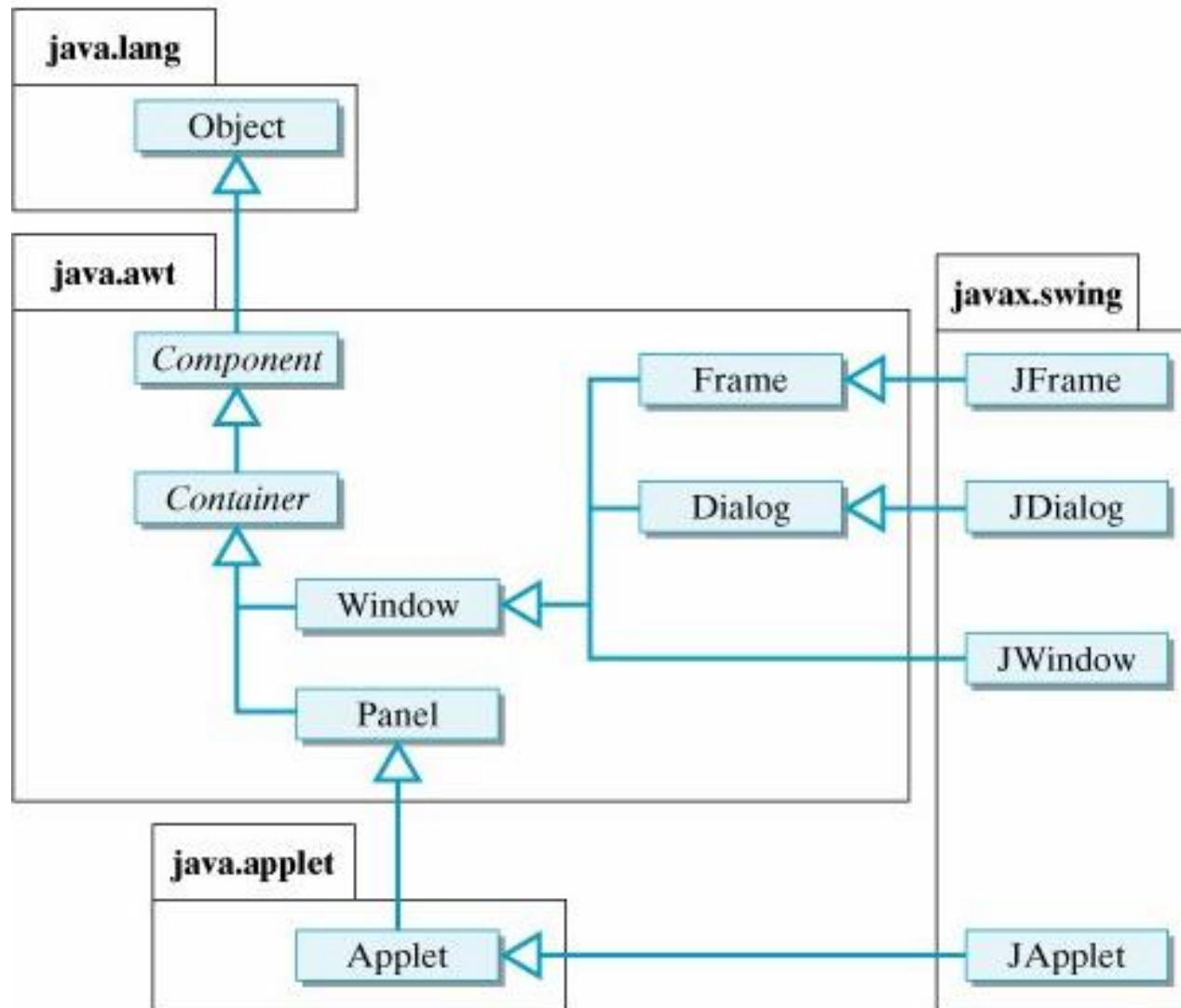
Nội dung

1. Tổng quan về đồ họa
2. Xử lý sự kiện
3. Quản lý bố cục (layout)
- ⇒ 4. Java Swing

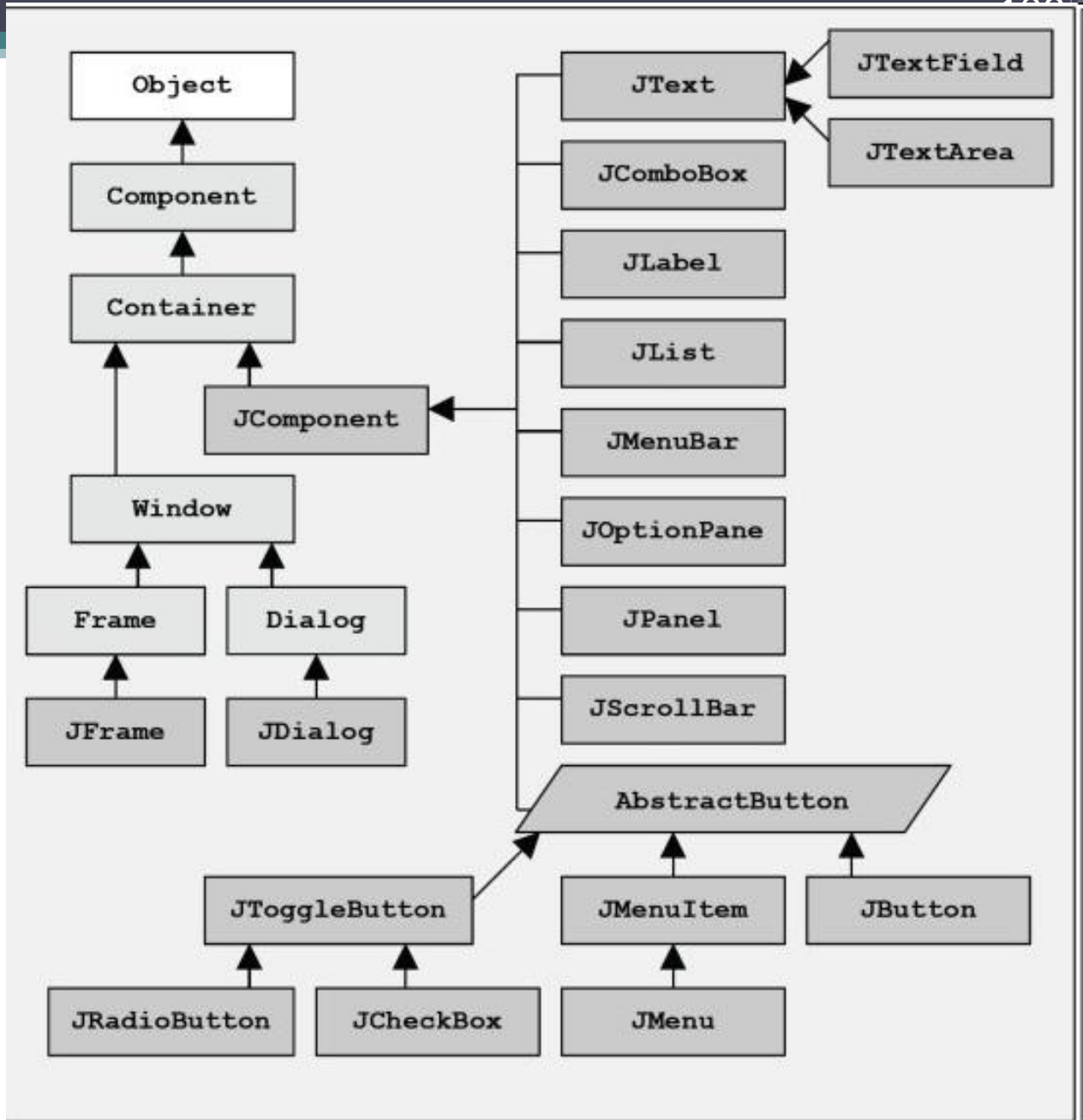
Swing vs. AWT

- 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.

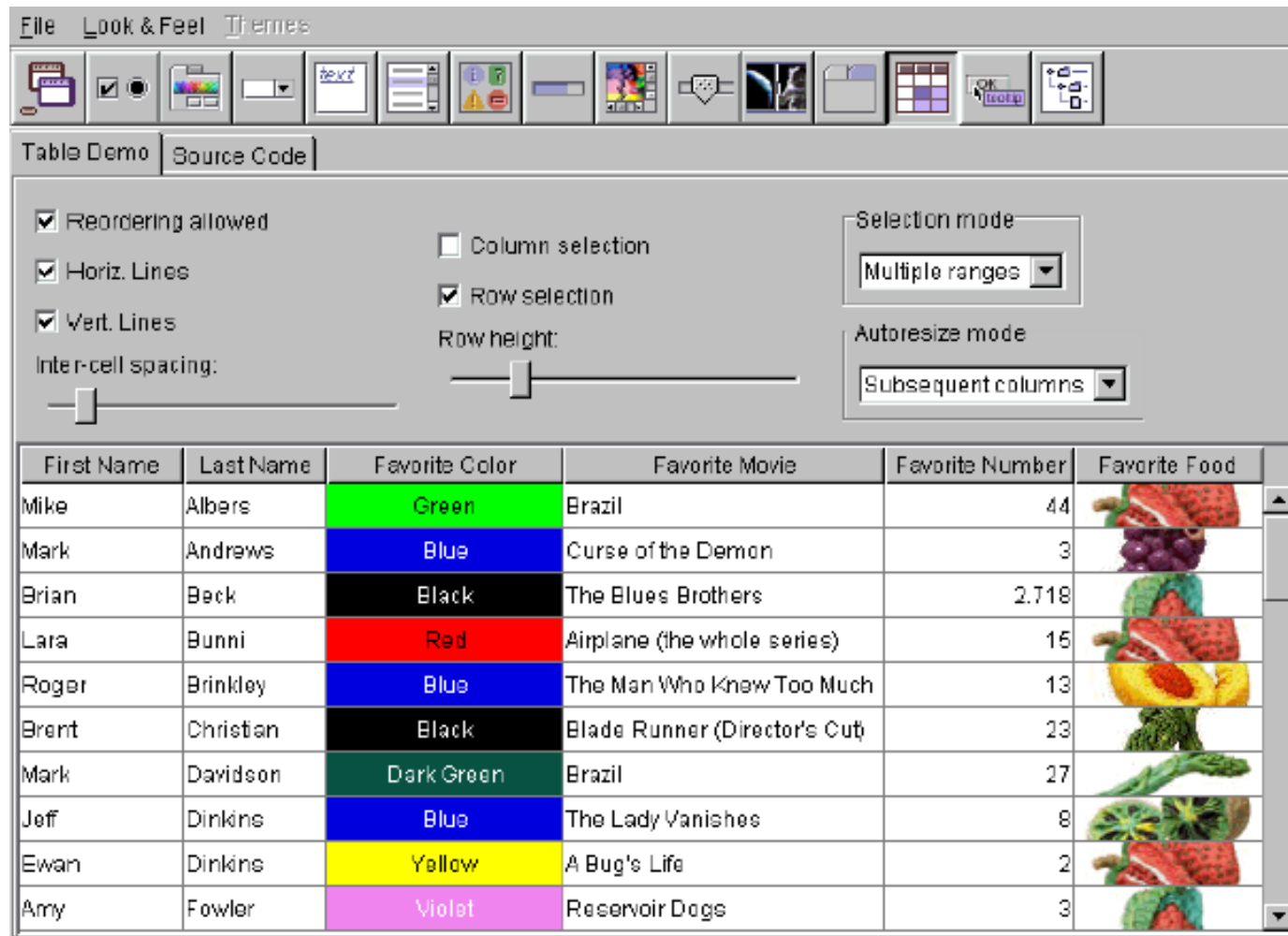
GUI component hierarchy



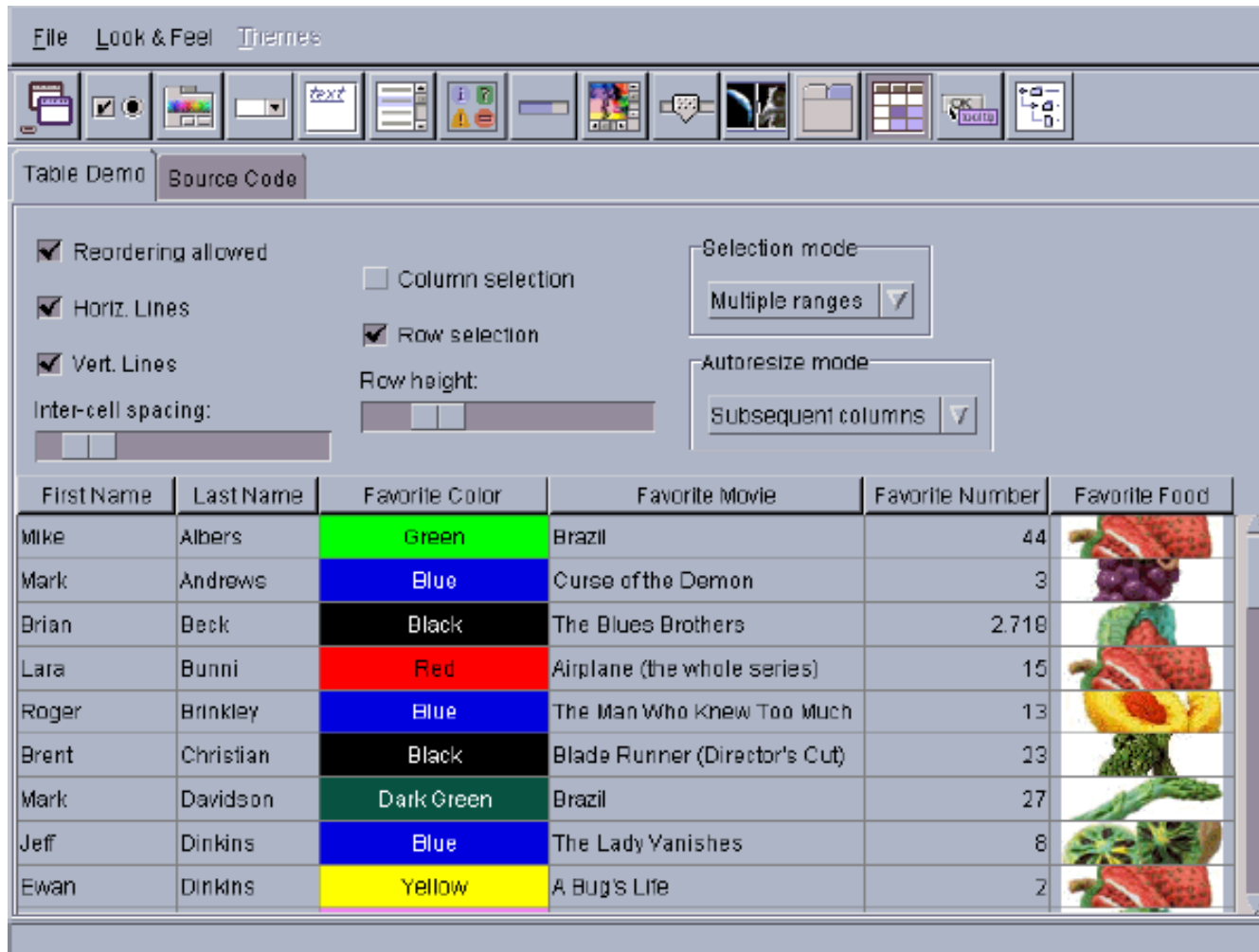
GUI component hierarchy



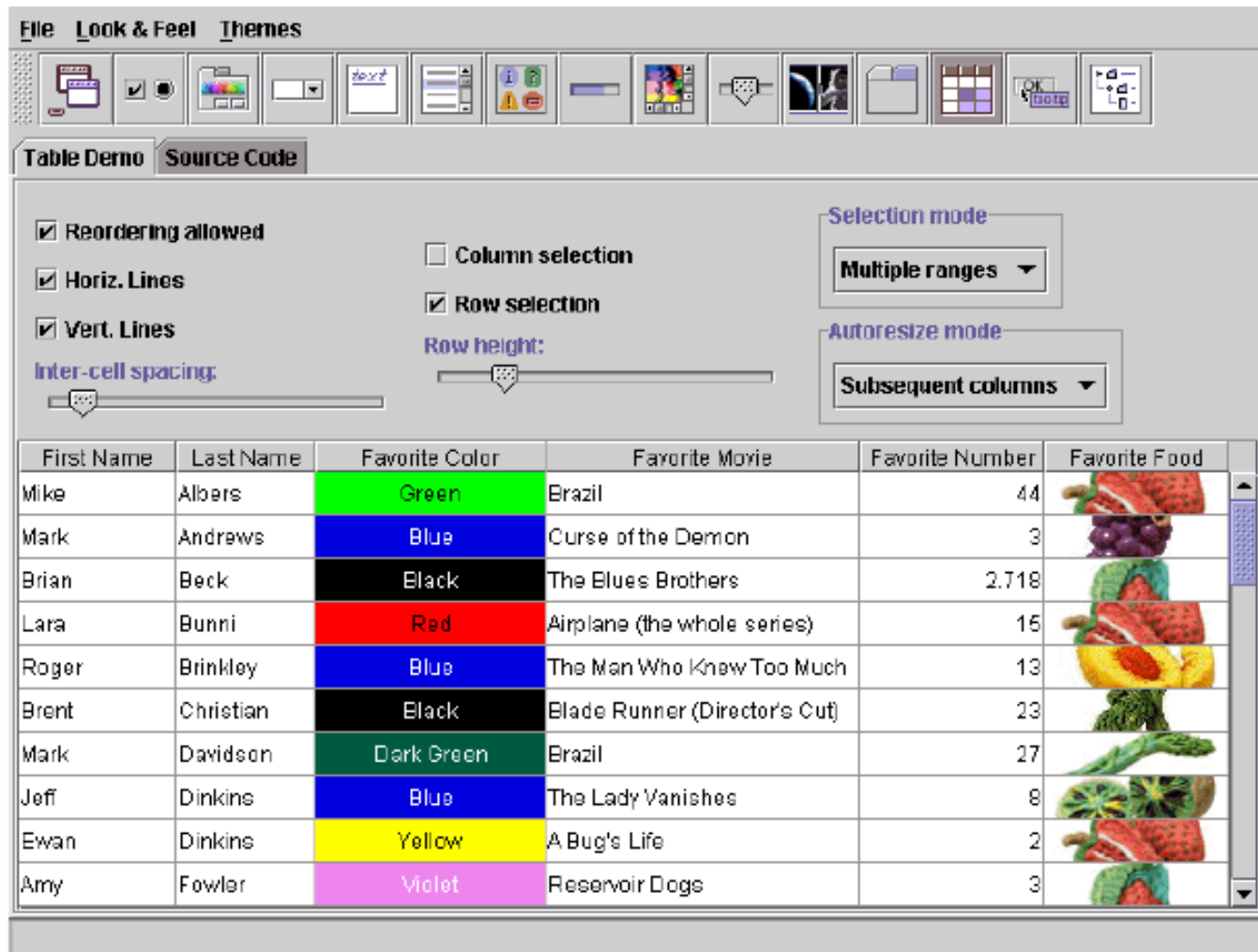
Windows Look and Feel



Motif Look and Feel



Java Look and Feel



Thay đổi Look and Feel

- Hầu hết các ứng dụng Java đều sử dụng native look, chứ không phải Java look.

```
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);  
        }  
    }  
    ...  
}
```

Java Swing – Các thành phần

- Các điểm bắt đầu
 - 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

2.1. JApplet và JFrame

- **Content pane (ô chứa nội dung)**
 - Một JApplet chứa một content pane để thêm các thành phần vào đó.
 - Thay đổi các thuộc tính như layout manager, background color, etc., cũng áp dụng cho content pane.
 - Truy cập vào content pane thông qua phương thức 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.

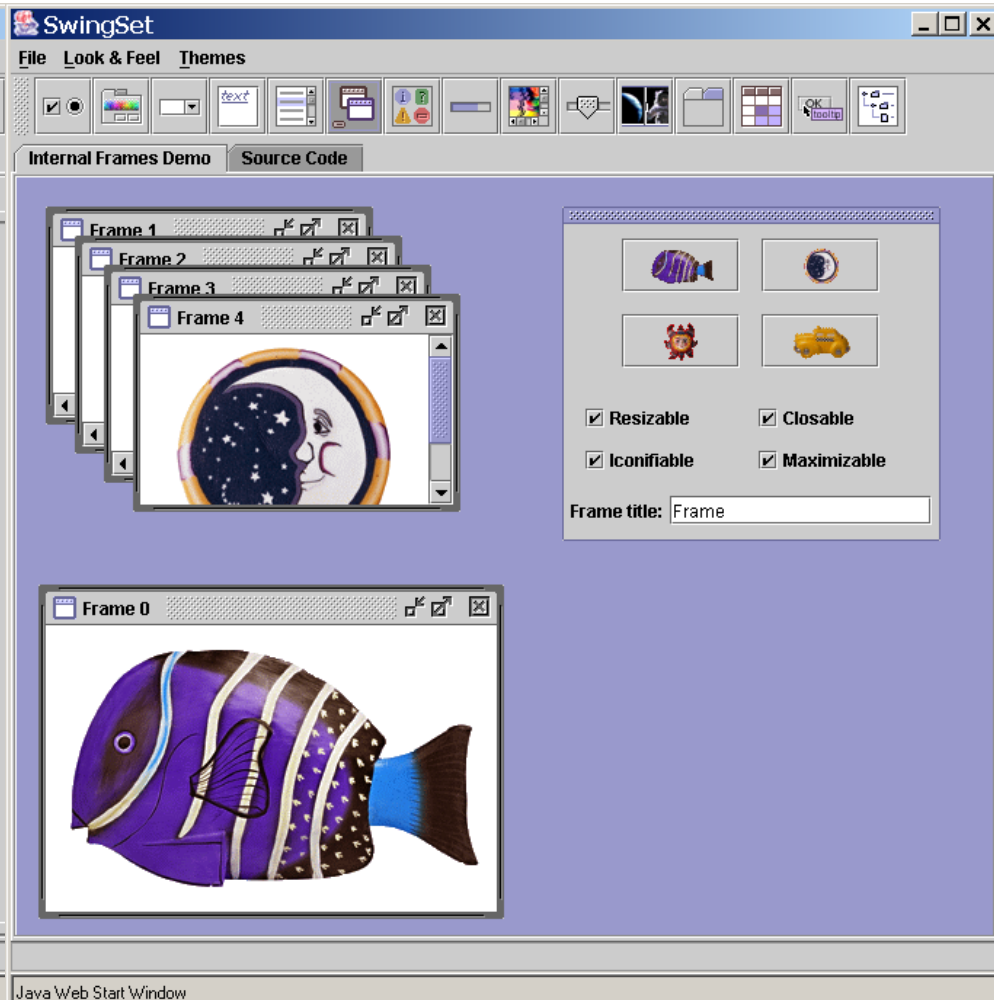
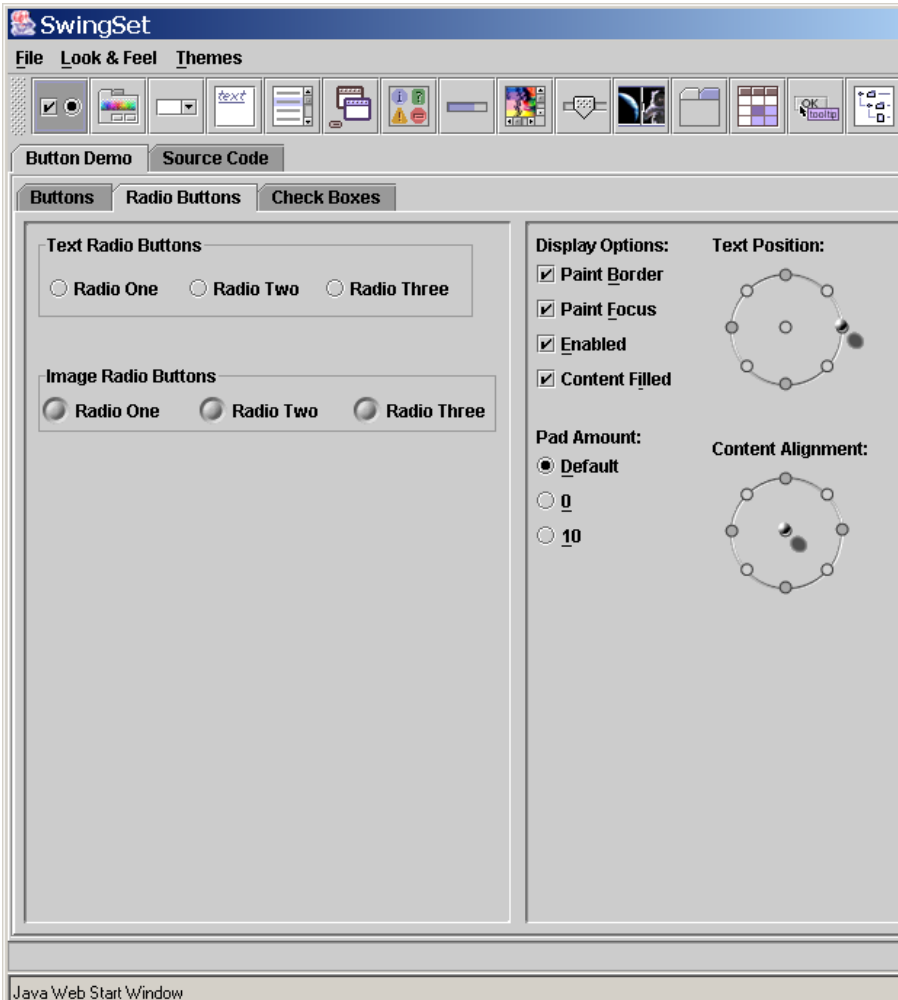
Swing - New Features

- Many more built-in controls
 - Image buttons, tabbed panes, sliders, toolbars, color choosers, HTML text areas, lists, trees, and tables.
- Increased customization of components
 - Border styles, text alignments, and basic drawing features. Images can be added to almost any control.
- A pluggable “look and feel”
- Many miscellaneous small features

Whirlwind Tour of Basic Components

- Starting points
 - JApplet
- Swing equivalent of AWT components
 - JLabel, JButton, JPanel, JSlider
- New Swing components
 - JColorChooser, JInternalFrame, JOptionPane
- Other simple components
 - JCheckBox, JRadioButton, JTextField, JTextArea, JFileChooser

SwingSet - Java Web Start

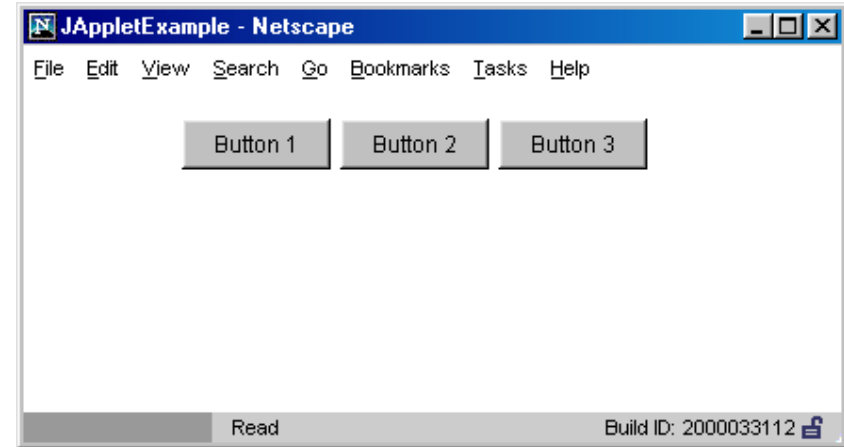


Starting Point: JApplet

- 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: Example Code

```
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"));
    }
}
```



Swing Equivalents of AWT Components

- 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**
 1. Create an ImageIcon by passing the ImageIcon constructor a String representing a GIF or JPG file (animated GIFs!).
 2. 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.

JButton: Example Code

```
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());
    }
}
```


JButton: Example Code (Continued)

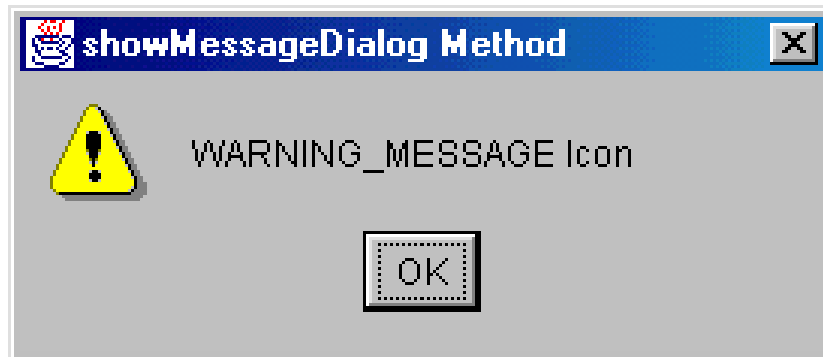
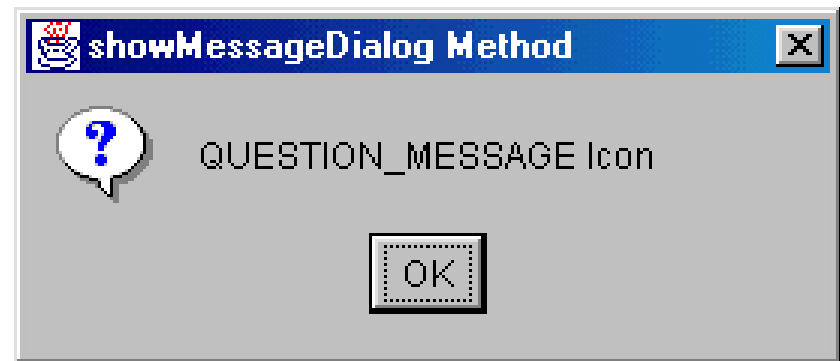
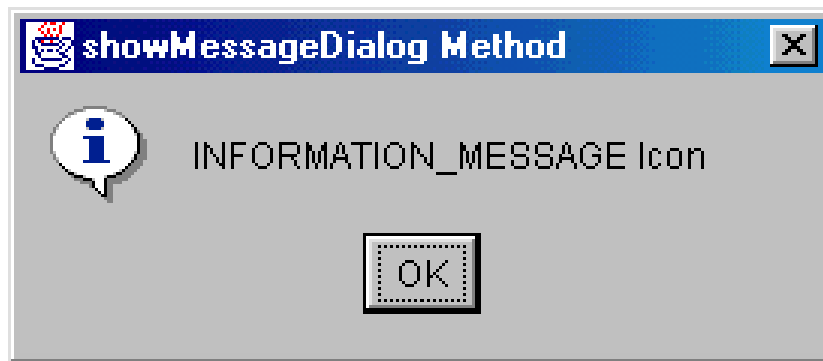
```
JButton button1 = new JButton("Java");  
content.add(button1);  
ImageIcon cup = new ImageIcon("images/cup.gif");  
JButton button2 = new JButton(cup);  
content.add(button2);  
JButton button3 = new JButton("Java", cup);  
content.add(button3);  
JButton button4 = new JButton("Java", cup);  
button4.setHorizontalTextPosition(SwingConstants.LEFT);  
content.add(button4);  
pack();  
setVisible(true);  
}  
}
```



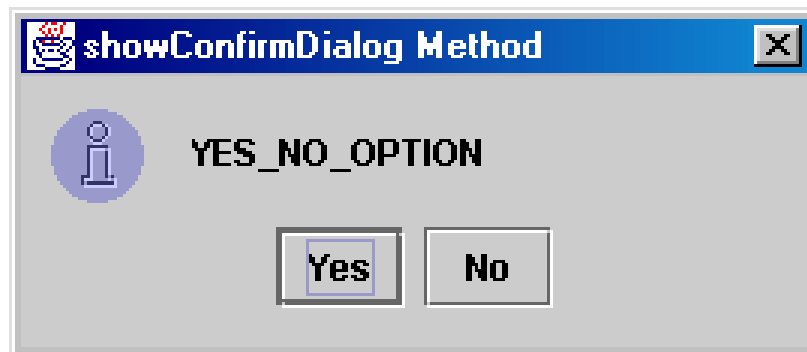
JOptionPane

- 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 (Windows LAF)



JOptionPane Confirmation Dialogs (Java LAF)



Other Simple Swing Components

- **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**

