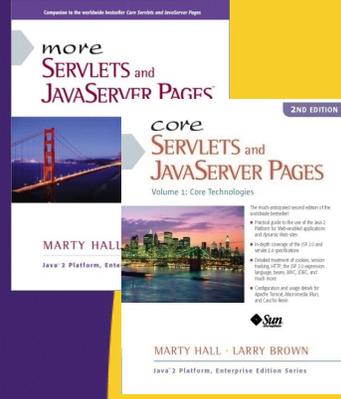




AWT Components: Simple User Interfaces

Originals of Slides and Source Code for Examples:
<http://courses.coreservlets.com/Course-Materials/java.html>

Customized Java EE Training: <http://courses.coreservlets.com/>
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live Java EE training, please see training courses
at <http://courses.coreservlets.com/>.**

**JSF 2, PrimeFaces, Servlets, JSP, Ajax (with jQuery), GWT,
Android development, Java 6 and 7 programming,
SOAP-based and RESTful Web Services, Spring, Hibernate/JPA,
XML, Hadoop, and customized combinations of topics.**



Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact hall@coreservlets.com for details.

Topics in This Section

- **GUI libraries in Java**
- **Basic AWT windows**
 - Canvas, Panel, Frame
- **Closing frames**
- **Processing events in GUI controls**
- **Basic AWT user interface controls**
 - Button, checkbox, radio button, list box

5

© 2012 [Marty Hall](#)



GUI Libraries in Java SE

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

6

GUI Libraries in Java

Part of Java SE

- **AWT (Abstract Window Toolkit)**
 - The original GUI library in Java 1.02. Native Look and Feel (LAF).
 - Covered in this lecture
 - Purposes
 - Easy building of simple-looking interfaces
 - Often for internal purposes only. Not seen by end users.
 - First step toward learning Swing
- **Swing**
 - GUI library added to Java starting in Java 1.1
 - Covered in later lectures
 - Purposes
 - Professional looking GUIs that follow standard
 - GUIs with the same look and feel on multiple platforms

Extensions

- **SWT (Standard Widget Toolkit)**
 - GUI from the Eclipse foundation. Native LAF ala AWT.
 - See <http://www.eclipse.org/swt/>
 - Purposes
 - Higher-performance professional looking GUIs
 - Native LAF
 - Interaction with the Eclipse Rich Client Platform
- **Java FX**
 - GUI library and tools now standardized separately
 - See <http://javafx.com/>
 - Purposes
 - XML-based layout
 - Mobile platforms
 - Rich media: audio, video, etc.

7

© 2012 [Marty Hall](#)



Background

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

8

Windows and Layout Management

- **Containers**
 - Most windows are a `Container` that can hold other windows or GUI components. `Canvas` is the major exception.
- **Layout Managers**
 - Containers have a `LayoutManager` that automatically sizes and positions components that are in the window
 - You can change the behavior of the layout manager or disable it completely. Details in next lecture.
- **Events**
 - Windows and components can receive mouse and keyboard events, just as in previous lecture.

9

Windows and Layout Management (Continued)

- **Drawing in Windows**
 - To draw into a window, make a subclass with its own `paint` method
 - Having one window draw into another window is not usually recommended
- **Popup Windows**
 - Some windows (`Frame` and `Dialog`) have their own title bar and border and can be placed at arbitrary locations on the screen
 - Other windows (`Canvas` and `Panel`) are embedded into existing windows only

10



Foundational AWT Window Types

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

11

Summary

- **Canvas**
 - Purpose:
 - Reusable picture or drawing area. Basis for custom component.
 - Code
 - Allocate Canvas, give it a size, add it to existing window.
- **Panel**
 - Purpose
 - To group other components into rectangular regions.
 - Code
 - Allocate Panel, put other components in it, add to window.
- **Frame**
 - Purpose
 - Core popup window. Main window for your application.
 - Code
 - Allocate Frame, give it a size, add stuff to it, pop it up.

12

Canvas

- **Major purposes**
 - A drawing area
 - A custom component that does not need to contain any other component (e.g., an image button)
- **Default layout manager: none**
 - Canvas is not a Container, so cannot enclose components
- **Creating and using**
 - Allocate it
 - Canvas c = new Canvas();
 - Give it a size
 - c.setSize(width, height);
 - Drop it in existing window
 - someWindow.add(c);

Since Canvas is often the starting point for a component that has a custom paint method or event handlers, you often do

```
MySpecializedCanvas c = new MySpecializedCanvas(...);
```

If this code is in the main window, then "someWindow" is "this" and can be omitted. I.e. the init method of an applet would add a Canvas to itself just with "add(c);".

13

Canvas Example

```
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);
    }

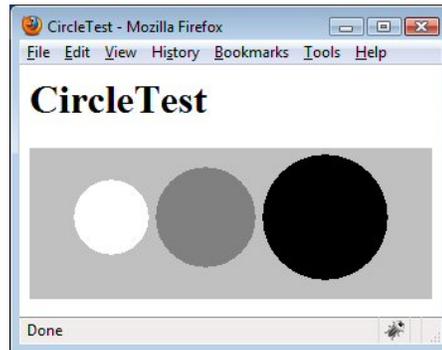
    public void setCenter(int x, int y) {
        setLocation(x - width/2, y - height/2);
    }
}
```

14

Canvas Example (Continued)

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

public class CircleTest extends Applet {
    public void init() {
        setBackground(Color.LIGHT_GRAY);
        add(new Circle(Color.WHITE, 30));
        add(new Circle(Color.GRAY, 40));
        add(new Circle(Color.BLACK, 50));
    }
}
```

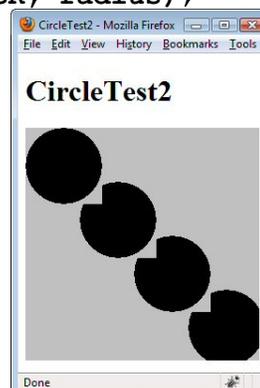


15

Canvases are Rectangular and Opaque: Example

```
public class CircleTest2 extends Applet {
    public void init() {
        setBackground(Color.LIGHT_GRAY);
        setLayout(null); // Turn off layout manager.
        Circle circle;
        int radius = getSize().width/6;
        int deltaX = round(2.0 * (double)radius / Math.sqrt(2.0));
        for (int x=radius; x<6*radius; x=x+deltaX) {
            circle = new Circle(Color.BLACK, radius);
            add(circle);
            circle.setCenter(x, x);
        }
    }

    private int round(double num) {
        return ((int)Math.round(num));
    }
}
```



16

Lightweight Components

- **Idea**

- Regular AWT windows are native windows behind the scenes. So, they are rectangular and opaque.
- You can make “lightweight components” – components that are really pictures, not windows, behind the scenes.
 - These don’t have the rectangular/opaque restrictions, but building them is usually more trouble than it is worth in the AWT library. The Swing library makes it simple with a “setOpaque” method.

- **Code**

- If you really want to do it yourself in AWT, you have to tell Java how to calculate the minimum and preferred sizes (see later section on layout managers).
 - Even so, it can have tricky interactions if the enclosing window has a custom paint method. Use Swing instead!

17

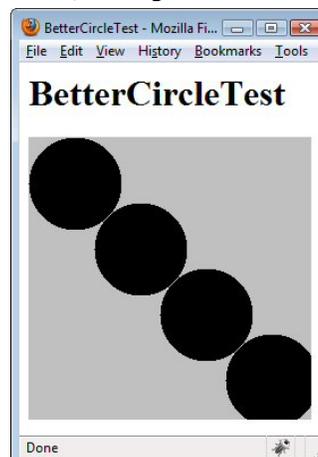
Lightweight Components: Example

```
public class BetterCircle extends Component {
    private Dimension preferredDimension;
    private int width, height;

    public BetterCircle(Color foreground, int radius) {
        setForeground(foreground);
        width = 2*radius; height = 2*radius;
        preferredDimension = new Dimension(width, height);
        setSize(preferredDimension);
    }

    public void paint(Graphics g) {
        g.setColor(getForeground());
        g.fillOval(0, 0, width, height);
    }

    public Dimension getPreferredSize() {
        return(preferredDimension);
    }
    public Dimension getMinimumSize() {
        return(preferredDimension);
    }
    ...
}
```



18

Component Class

- **Idea**

- Ancestor of all graphical components in Java (even Swing). So, methods here are shared by all windows and controls.

- **Useful methods**

- setBackground/setBackground
- getForeground/setForeground
 - Change/lookup the default foreground color
 - Color is inherited by the Graphics object of the component
- getFont/setFont
 - Returns/sets the current font
 - Inherited by the Graphics object of the component
- paint
 - Called whenever the user call repaint or when the component is obscured and reexposed

19

Component Class (Continued)

- **Useful methods**

- setVisible
 - Exposes (`true`) or hides (`false`) the 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)

20

Panel

- **Major purposes**
 - To group/organize components
 - A custom component that requires embedded components
- **Default layout manager: FlowLayout**
 - Shrinks components to their preferred (minimum) size
 - Places them left to right in centered rows
- **Creating and using**
 - Allocate it
 - `Panel p = new Panel();`
 - Put stuff into it
 - `p.add(someButton);`
 - `p.add(someOtherWidget);`
 - Drop the Panel in an existing window
 - `someWindow.add(p);`

Note the lack of an explicit `setSize()`. The size of a Panel is usually determined by a combination of what the Panel contains and the layout manager of the window that contains the Panel.

21

No Panels: Example

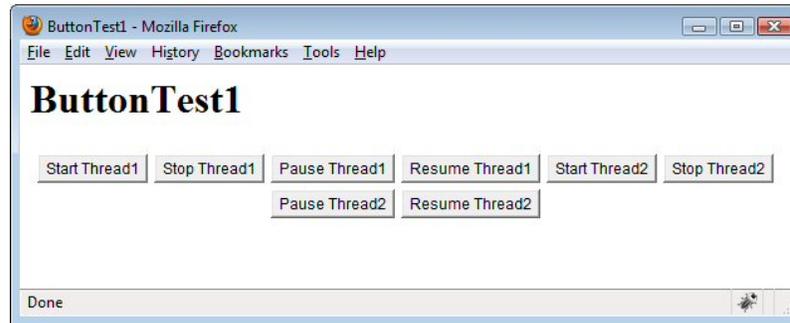
```
import java.applet.Applet;
import java.awt.*;

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"));
        }
    }
}
```

22

No Panels: Result



23

Panels: Example

```
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);
    }
}
```

24

Panels: Result



25

Container Class

- **Idea**
 - Ancestor of all window types except Canvas. So, these methods are common among almost all windows.
- **Useful Container methods**
 - add
 - Add a component to the container (in the last position in the component array)
 - If using BorderLayout, you can also specify in which region to place the component
 - remove
 - Remove the component from the window (container)
 - getComponents
 - Returns an array of components in the window
 - Used by layout managers
 - setLayout
 - Changes the layout manager associated with the window

26

Frame Class

- **Major Purpose**
 - A stand-alone window with its own title and menu bar, border, cursor, and icon image
 - Can contain other GUI components
- **Default layout manager: BorderLayout**
 - BorderLayout
 - Divides the screen into 5 regions: North, South, East, West, and Center
 - To switch to the applet's layout manager use
 - `setLayout(new FlowLayout());`
- **Creating and using – two approaches:**
 - A fixed-size Frame
 - A Frame that stretches to fit what it contains

27

Creating a Fixed-Size Frame

- **Approach**

```
Frame frame = new Frame(titleString);
frame.add(somePanel, BorderLayout.CENTER);
frame.add(otherPanel, BorderLayout.NORTH);
...
frame.setSize(width, height);
frame.setVisible(true);
```
- **Note: be sure you pop up the frame last**
 - Odd behavior results if you add components to a window that is already visible (unless you call `doLayout` on the frame)

28

Creating a Frame that Stretches to Fit What it Contains

- **Approach**

```
Frame frame = new Frame(titleString);
frame.setLocation(left, top);
frame.add(somePanel, BorderLayout.CENTER);
...
frame.pack();
frame.setVisible(true);
```

- **Note**

- Again, be sure to pop up the frame *after* adding the components

29

Frame Example 1

- **Creating the Frame object in main**

```
public class FrameExample1 {
    public static void main(String[] args) {
        Frame f = new Frame("Frame Example 1");
        f.setSize(400, 300);
        f.setVisible(true);
    }
}
```

30

Frame Example 2

- Using a Subclass of Frame

```
public class FrameExample2 extends Frame {
    public FrameExample2() {
        super("Frame Example 2");
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new FrameExample2();
    }
}
```

The "main" method that instantiates the Frame need not reside in FrameExample2. The idea is that you make a reusable Frame class, and then that class can be popped up various different ways (from main, when the user clicks a button, when certain events occur in your app, etc.)

31

A Closeable Frame

- CloseableFrame.java

```
public class CloseableFrame extends Frame {
    public CloseableFrame(String title) {
        super(title);
        addWindowListener(new ExitListener());
    }
}
```

- ExitListener.java

```
public class ExitListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
        System.exit(0);
    }
}
```

Download these two classes from the source code in the tutorial, then use CloseableFrame wherever you would have used Frame.

32

Frame Example 3

- Using a Subclass of CloseableFrame

```
public class FrameExample3 extends CloseableFrame {
    public FrameExample3() {
        super("Frame Example 3");
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new FrameExample3();
    }
}
```

Same as previous example, but now the Frame closes when you click on the x.

33

© 2012 Marty Hall



AWT GUI Controls and Event Processing

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

34

AWT GUI Controls

- **Characteristics (vs. windows)**
 - Automatically drawn – you don't override `paint`
 - Positioned by layout manager
 - Use native window-system controls (widgets)
 - Controls adopt look and feel of underlying window system
 - Higher level events typically used
 - For example, for buttons you don't monitor mouse clicks, since most OS's also let you trigger a button by hitting RETURN when the button has the keyboard focus

35

GUI Event Processing Strategies

- **Decentralized event processing**
 - Component (e.g., Button) has its own event handler
 - Harder to call methods in the main app, so works best when operations are relatively independent
- **Centralized event processing**
 - Have main app implement listener. Send all events there.
 - Easier for handler to call methods from the main app
 - But, if you have multiple buttons, you will need if/then/else in the event-handler method
- **Semi-centralized event processing**
 - Use inner class for event handling
 - Better than interface if you have many different buttons

36

Decentralized Event Processing: Example

```
import java.awt.*;

public class ActionExample1 extends CloseableFrame {
    public ActionExample1() {
        super("Handling Events in Component");
        setLayout(new FlowLayout());
        setFont(new Font("Serif", Font.BOLD, 18));
        add(new SetSizeButton(300, 200));
        add(new SetSizeButton(400, 300));
        add(new SetSizeButton(500, 400));
        setSize(400, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new ActionExample1();
    }
}
```

37

Very closely analogous to the first approach from the event-handling lecture (separate classes for event handlers).

Decentralized Event Processing: Example (Continued)

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

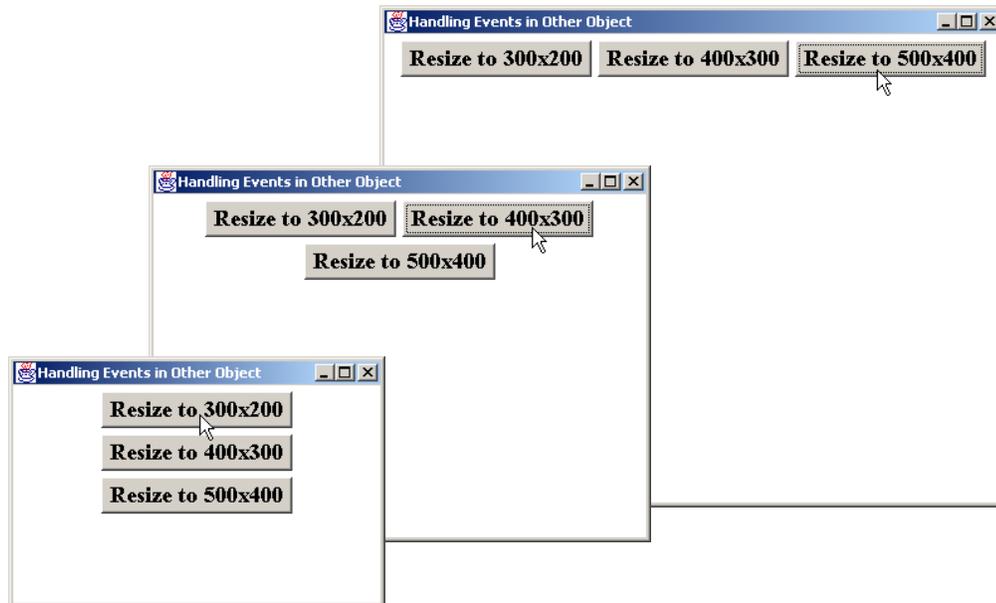
public class SetSizeButton extends Button
    implements ActionListener {
    private int width, height;

    public SetSizeButton(int width, int height) {
        super("Resize to " + width + "x" + height);
        this.width = width;
        this.height = height;
        addActionListener(this);
    }

    public void actionPerformed(ActionEvent event) {
        Container parent = getParent();
        parent.setSize(width, height);
        parent.invalidate();
        parent.validate();
    }
}
```

38

Decentralized Event Processing: Result



39

Centralized Event Processing: Example

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

public class ActionExample2 extends CloseableFrame
    implements ActionListener {
    private Button button1, button2, button3;

    public ActionExample2() {
        super("Handling Events in Other Object");
        setLayout(new FlowLayout());
        setFont(new Font("Serif", Font.BOLD, 18));
        button1 = new Button("Resize to 300x200");
        button1.addActionListener(this);
        add(button1);
        // Add button2 and button3 in the same way...
        ...
        setSize(400, 300);
        setVisible(true);
    }
}
```

40

Centralized Event Processing: Example (Continued)

```
public void actionPerformed(ActionEvent event) {
    if (event.getSource() == button1) {
        updateLayout(300, 200);
    } else if (event.getSource() == button2) {
        updateLayout(400, 300);
    } else if (event.getSource() == button3) {
        updateLayout(500, 400);
    }
}

private void updateLayout(int width, int height) {
    setSize(width, height);
    invalidate();
    validate();
}

public static void main(String[] args) {
    new ActionExample2();
}
}
```

Very closely analogous to the second approach from the event-handling lecture (main class implements interface).

41

Semi-Centralized Event Processing: Example

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

public class ActionExample3 extends CloseableFrame {
    private Button button1, button2, button3;

    public ActionExample3() {
        super("Handling Events in Other Object");
        setLayout(new FlowLayout());
        setFont(new Font("Serif", Font.BOLD, 18));
        button1 = new Button("Resize to 300x200");
        button1.addActionListener(new ResizeHandler(300, 200));
        add(button1);
        // Add button2 and button3 in the same way...
        ...
        setSize(400, 300);
        setVisible(true);
    }
}
```

42

Semi-Centralized Event Processing: Example (Cont)

```
private void updateLayout(int width, int height) {
    setSize(width, height);
    invalidate();
    validate();
}

private class ResizeHandler implements ActionListener {
    private int width, height;

    public ResizeHandler(int width, int height) {
        this.width= width;
        this.height = height;
    }

    public void actionPerformed(ActionEvent event) {
        updateLayout(width, height);
    }
}

public static void main(String[] args) {
    new ActionExample3();
}
}
```

Very closely analogous to the third approach from the event-handling lecture (inner classes for event handlers).

43

© 2012 Marty Hall



Basic AWT GUI Controls

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

44

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();
```

45

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

46

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);
    }
}
```



47

Handling Button Events

- Attach an ActionListener to the Button and handle the event in actionPerformed

```
public class MyActionListener
    implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        ...
    }
}

public class SomeClassThatUsesButtons {
    ...
    MyActionListener listener = new MyActionListener(...);
    Button b1 = new Button("...");
    b1.addActionListener(listener);
    ...
}
```

48

Checkboxes

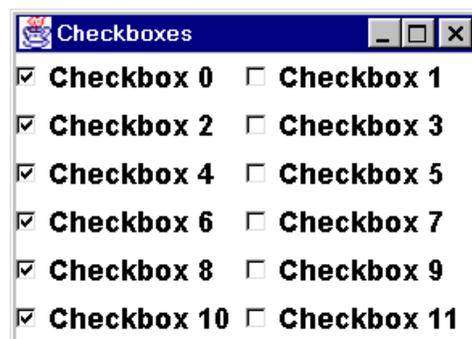
- **Constructors**

- 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

49

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);
    }
}
```



50

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

```
someCheckbox.setLabel("A New Label");
someCheckbox.getParent().invalidate();
someCheckbox.getParent().validate();
```
- **addItemListener/removeItemListener**
 - Add or remove an `ItemListener` to process `ItemEvents` in `itemStateChanged`
- **processItemEvent(ItemEvent event)**
 - Low-level event handling

51

Handling Checkbox Events

- **Attach an ItemListener**
 - Add it with `addItemListener` and process the `ItemEvent` in `itemStateChanged`

```
public void itemStateChanged(ItemEvent event) {
    ...
}
```

 - The `ItemEvent` class has a `getItem` method which returns the item just selected or deselected
 - The return value of `getItem` is an `Object` so you should cast it to a `String` before using it
- **Ignore the event**
 - With checkboxes, it is relatively common to ignore the select/deselect event when it occurs
 - Instead, you look up the state (checked/unchecked) of the checkbox later using the `getState` method of `Checkbox` when you are ready to take some other sort of action

52

Checkbox Groups (Radio Buttons)

- **CheckboxGroup Constructors**

- CheckboxGroup()

- Creates a non-graphical object used as a “tag” to group checkboxes logically together
 - Checkboxes with the same tag will look and act like radio buttons
 - 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
 - If you specify an initial state of `true` for more than one Checkbox in a group, the last one will be shown selected

53

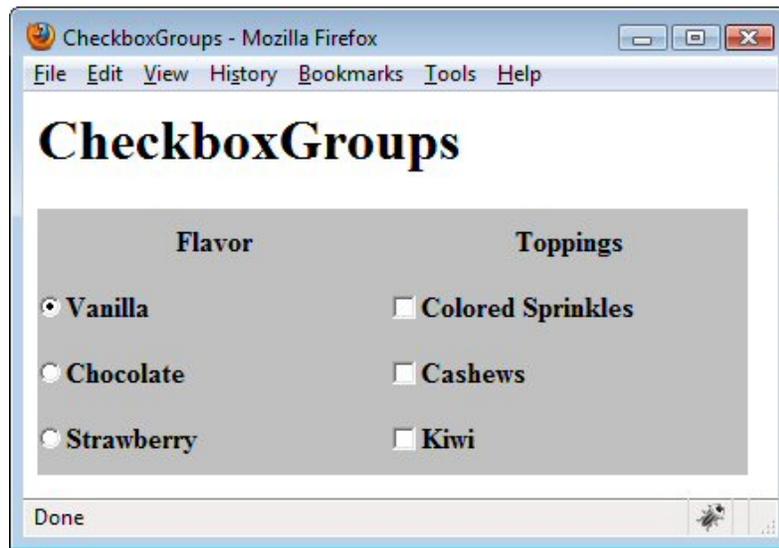
CheckboxGroup: Example

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

public class CheckboxGroups extends Applet {
    public void init() {
        setLayout(new GridLayout(4, 2));
        setBackground(Color.LIGHT_GRAY);
        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"));
    }
}
```

54

CheckboxGroup: Result



By tagging Checkboxes with a CheckboxGroup, the Checkboxes in the group function as radio buttons

55

Other Methods for Radio Buttons

- **CheckboxGroup**
 - `getSelectedCheckbox`
 - Returns the radio button (`Checkbox`) that is currently selected or `null` if none is selected
- **Checkbox**
 - In addition to the general methods described in Checkboxes, `Checkbox` has the following two methods specific to `CheckboxGroup`'s:
 - `getCheckboxGroup/setCheckboxGroup`
 - Determines or registers the group associated with the radio button
- **Note: Event-handling is the same as with Checkboxes**

56

List Boxes

- **Constructors**

- List(int rows, boolean multiSelectable)
 - Creates a listbox with the specified number of **visible rows** (not items)
 - 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**
 - The preferred width is set to a platform-dependent value, and is typically not directly related to the width of the widest entry
- 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

57

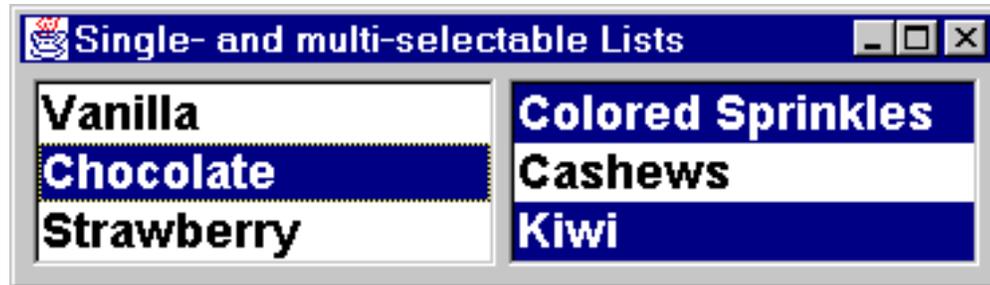
List Boxes: Example

```
import java.awt.*;

public class Lists extends CloseableFrame {
    public Lists() {
        super("Lists");
        setLayout(new FlowLayout());
        setBackground(Color.LIGHT_GRAY);
        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);
    }
}
```

58

List Boxes: Result



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

59

Other List Methods

- **add**
 - Add an item at the end or specified position in the list box
 - All items at that index or later get moved down
- **isMultipleMode**
 - Determines if the list is **multiple selectable** (`true`) or **single selectable** (`false`)
- **remove/removeAll**
 - Remove an item or all items from the list
- **getSelectedIndex**
 - For a single-selectable list, this returns the index of the selected item
 - Returns **-1 if nothing is selected** or if the list permits multiple selections
- **getSelectedIndexes**
 - Returns an array of the indexes of all selected items
 - Works for single- or multi-selectable lists
 - If no items are selected, a zero-length (but non-null) array is returned

60

Other List Methods (Continued)

- **getSelectedItem**
 - For a single-selectable list, this returns the label of the selected item
 - Returns null if nothing is selected or if the list permits multiple selections
- **getSelectedItems**
 - Returns an array of all selected items
 - Works for single- or multi-selectable lists
 - If no items are selected, a zero-length (but non-null) array is returned
- **select**
 - Programmatically selects the item in the list
 - If the list does not permit multiple selections, then the previously selected item, if any, is also deselected

61

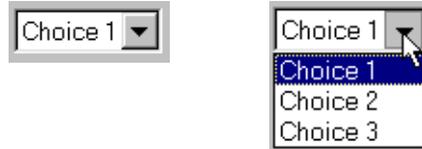
Handling List Events

- **addItemListener/removeItemListener**
 - **ItemEvents** are generated whenever an item is **selected** or **deselected** (single-click)
 - Handle **ItemEvents** in **itemStateChanged**
- **addActionListener/removeActionListener**
 - **ActionEvents** are generated whenever an item is **double-clicked** or RETURN (ENTER) is pressed while selected
 - Handle **ActionEvents** in **actionPerformed**

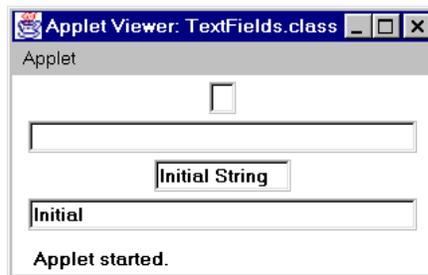
62

Other GUI Controls

- Choice Lists (Combo Boxes)



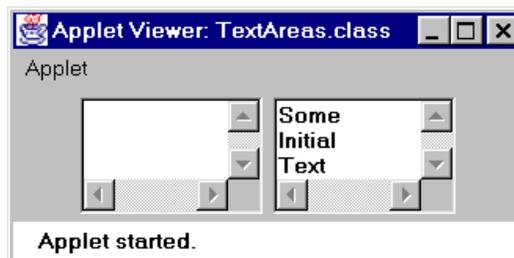
- Textfields



63

Other GUI Controls (Continued)

- Text Areas



- Labels



64

Summary

- **Native components behind the scenes**
 - So, all windows and graphical components are rectangular and opaque, and take look-and-feel of underlying OS.
- **Windows**
 - Canvas: drawing area or custom component
 - Panel: grouping other components
 - Frame: popup window
- **GUI Controls**
 - Button: handle events with ActionListener
 - Checkbox, radio button: handle events with ItemListener
 - List box: handle single click with ItemListener, double click with ActionListener
 - To quickly determine the event handlers for a component, simply look at the online API
 - addXxxListener methods are at the top

65

© 2012 Marty Hall



Questions?

[JSF 2](#), [PrimeFaces](#), [Java 7](#), [Ajax](#), [jQuery](#), [Hadoop](#), [RESTful Web Services](#), [Android](#), [Spring](#), [Hibernate](#), [Servlets](#), [JSP](#), [GWT](#), and other [Java EE training](#).

Customized Java EE Training: <http://courses.coreservlets.com/>

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.

Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

66