core WEB programming

AWT Components

Agenda

- Basic AWT windows
  - Canvas, Panel, Frame, Dialog
- Creating lightweight components
- Closing frames
- Using object serialization to save components to disk
- Basic AWT user interface controls
  - Button, checkbox, radio button, list box, scrollbars
- Processing events in GUI controls

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.

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` an `Panel`) are embedded into existing windows only

Canvas Class

- **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` *cannot* contain any other `Components`
- **Creating and Using**
  - Create the `Canvas`

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

Or, since you typically create a subclass of Canvas that has customized drawing via its `paint` method:

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

Canvas (Continued)

- **Creating and Using, cont.**
  - Size the `Canvas`

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

  - Add the `Canvas` to the current `Window`

    ```
    add(canvas);
    ```

    or depending on the layout manager you can position the Canvas

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

    If you first create a separate window (e.g. a Panel), then put the Canvas in the window using something like

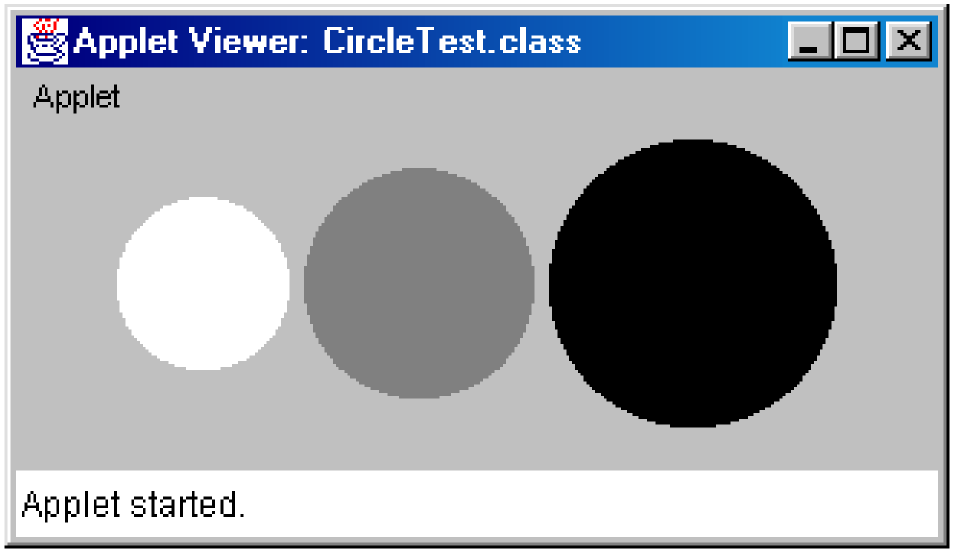
    ```
    someWindow.add(canvas);
    ```

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);
  }
}
```

Canvas Example (Continued)

```
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));
  }
}
```

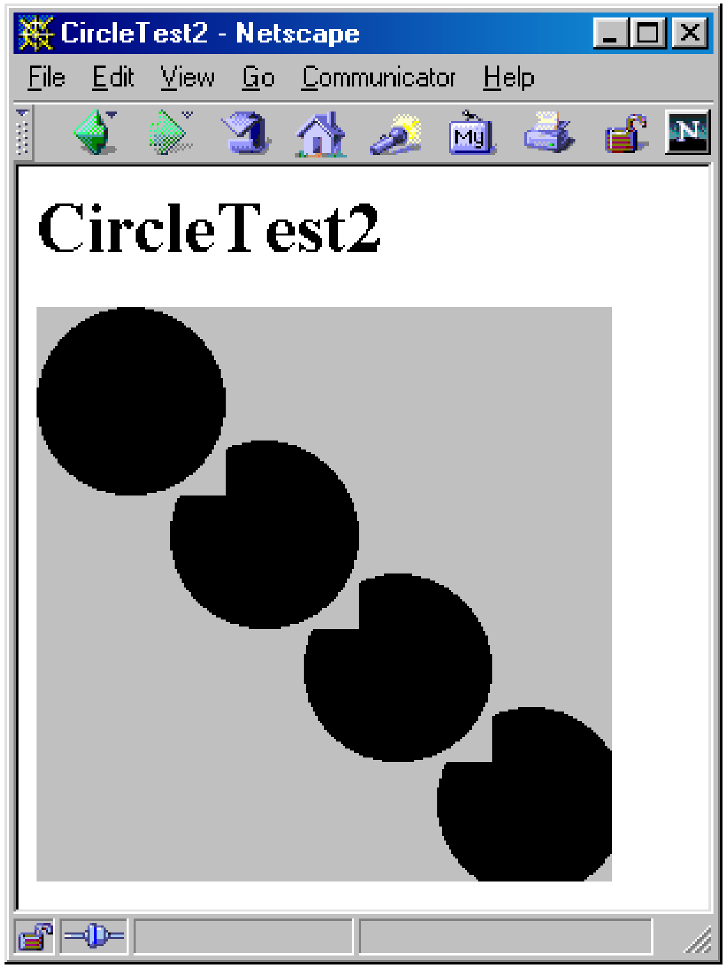


Applet started.

Canvases are Rectangular and Opaque: Example

```
public class CircleTest2 extends Applet {
  public void init() {
    setBackground(Color.lightGray);
    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));
  }
}
```

Canvases are Rectangular and Opaque: Result



Standard components have an associated peer (native window system object).

Component Class

- **Direct Parent Class of Canvas**
- **Ancestor of all Window Types**
- **Useful Methods**
  - getBackground/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

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)

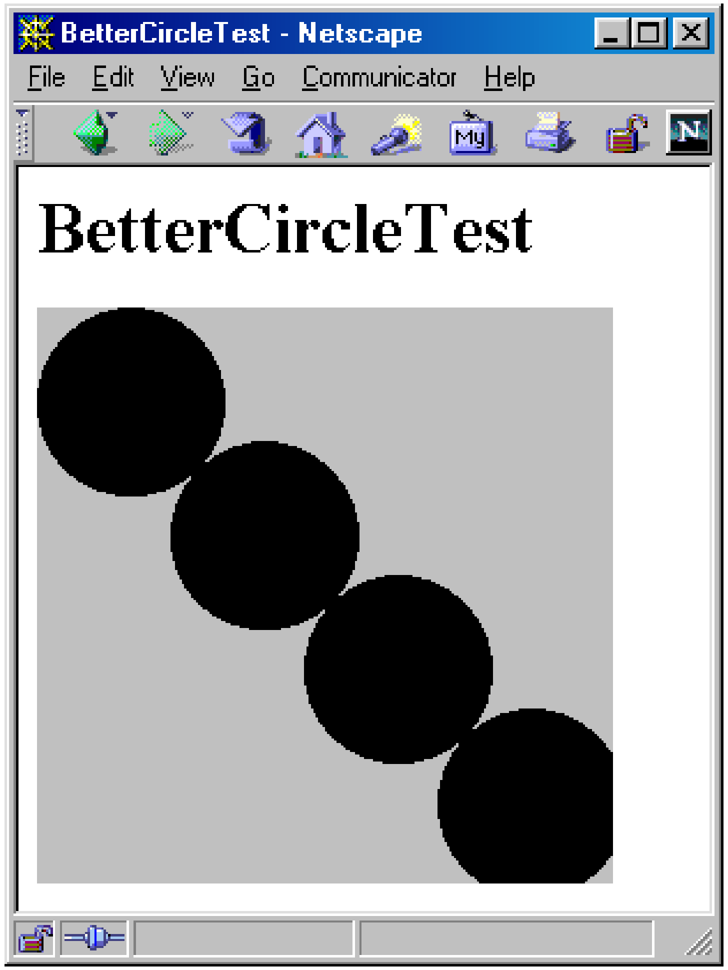
Lightweight Components

- Components that inherit directly from `Component` have no native peer
- The underlying component will show through except for regions directly drawn in `paint`
- If you use a lightweight component in a Container that has a custom `paint` method, call `super.paint` or the lightweight components will not be drawn

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);
  }
  ...
}
```

Lightweight Components: Result


BetterCircleTest - Netscape
File Edit View Go Communicator Help
BetterCircleTest

Lightweight components can be transparent

Panel Class

- **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**
  - Create the Panel

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

  - Add Components to Panel

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

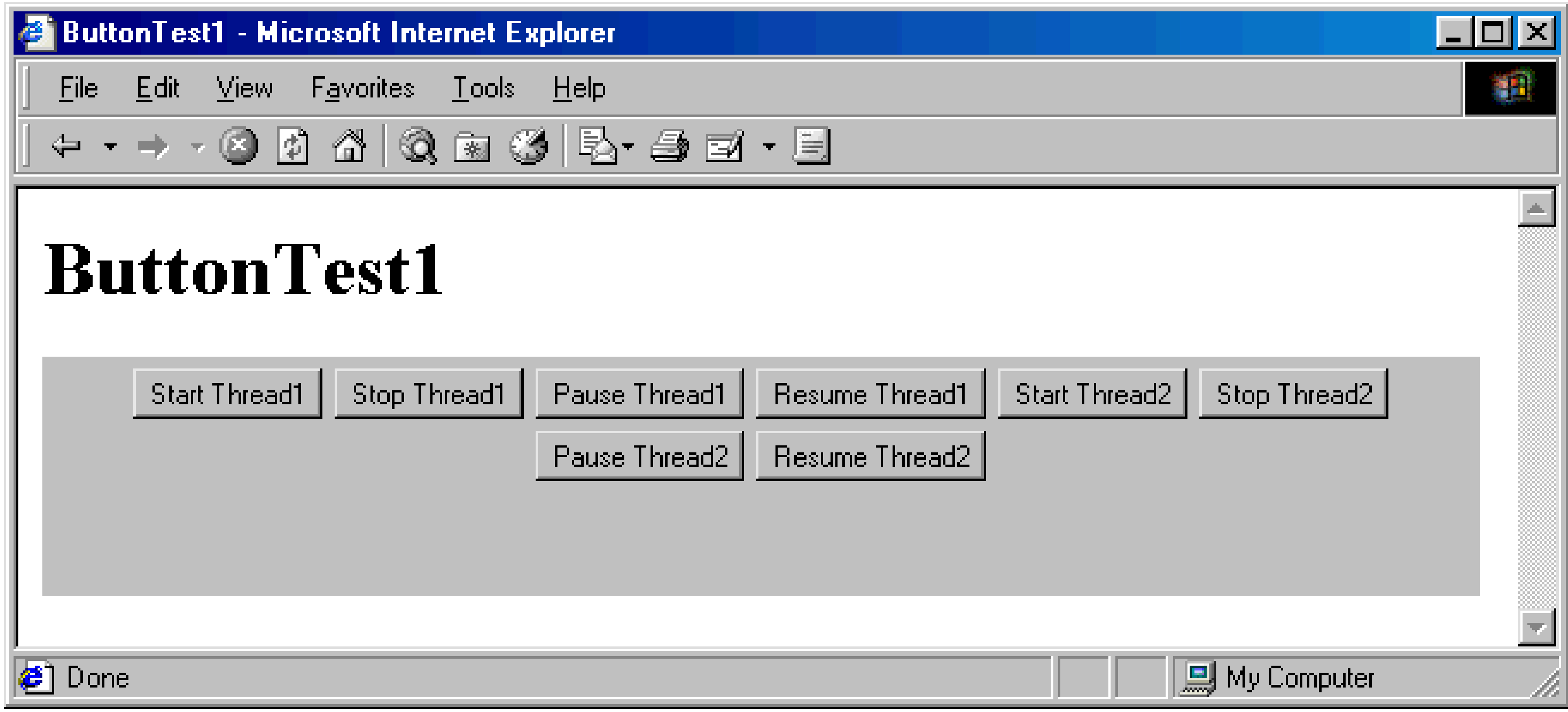
Panel (Continued)

- **Creating and Using, continued**
  - Add Panel to Container
    - To an external container
      - » container.add(panel);
    - From within a container
      - » add(panel);
    - To an external container that is using BorderLayout
      - » container.add(panel,region);
- **Note the lack of an explicit setSize**
  - The components inside determine the size of a panel; the panel is no larger then necessary to hold the components
  - A panel holding no components has a size of zero
- **Note: Applet is a subclass of Panel**

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"));
    }
  }
}
```

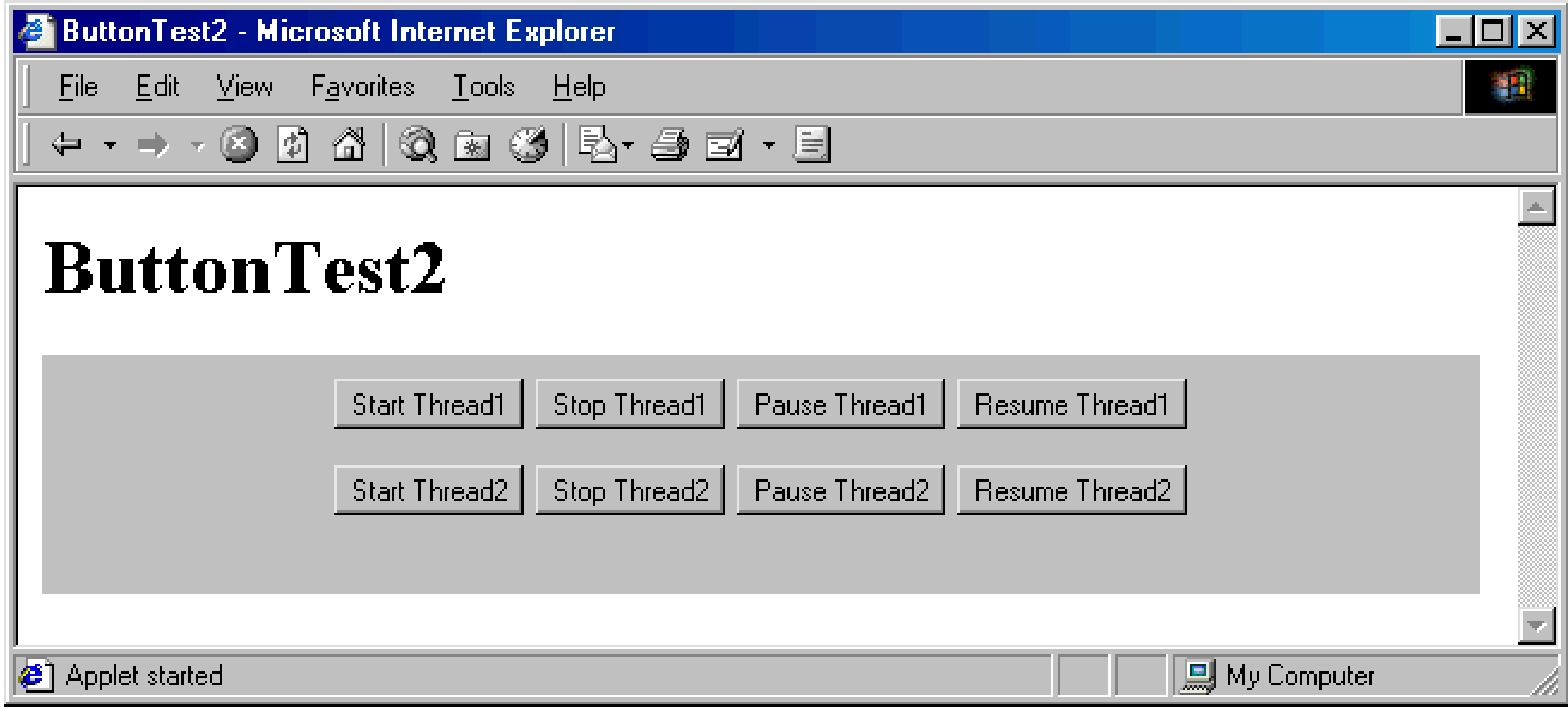
No Panels: Result



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);
  }
}
```

Panels: Result



Container Class

- **Ancestor of all Window Types Except `Canvas`**
- **Inherits all `Component` Methods**
- **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

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 LayoutManager: 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

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)

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);
```

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

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);
  }
}
```

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();
  }
}
```

A Closeable Frame

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

public class CloseableFrame extends Frame {

  public CloseableFrame(String title) {
    super(title);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  }

  public void processWindowEvent(WindowEvent event) {
    super.processWindowEvent(event); // Handle listeners
    if (event.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
}
```

- **If a Frame is used in an Applet, use `dispose` instead of `System.exit(0)`**

Dialog Class

- **Major Purposes**
  - A simplified Frame (no cursor, menu, icon image).
  - A modal Dialog that freezes interaction with other AWT components until it is closed
- **Default LayoutManager: BorderLayout**
- **Creating and Using**
  - Similar to Frame except constructor takes two additional arguments: the parent Frame and a boolean specifying whether or not it is modal

```
Dialog dialog =
  new Dialog(parentFrame, titleString, false);
Dialog modalDialog =
  new Dialog(parentFrame, titleString, true);
```

A Confirmation Dialog

```
class Confirm extends Dialog
              implements ActionListener {
  private Button yes, no;

  public Confirm(Frame parent) {
    super(parent, "Confirmation", true);
    setLayout(new FlowLayout());
    add(new Label("Really quit?"));
    yes = new Button("Yes");
    yes.addActionListener(this);
    no  = new Button("No");
    no.addActionListener(this);
    add(yes);
    add(no);
    pack();
    setVisible(true);
  }
```

A Confirmation Dialog (Continued)

```
public void actionPerformed(ActionEvent event) {
    if (event.getSource() == yes) {
      System.exit(0);
    } else {
      dispose();
    }
  }
}
```

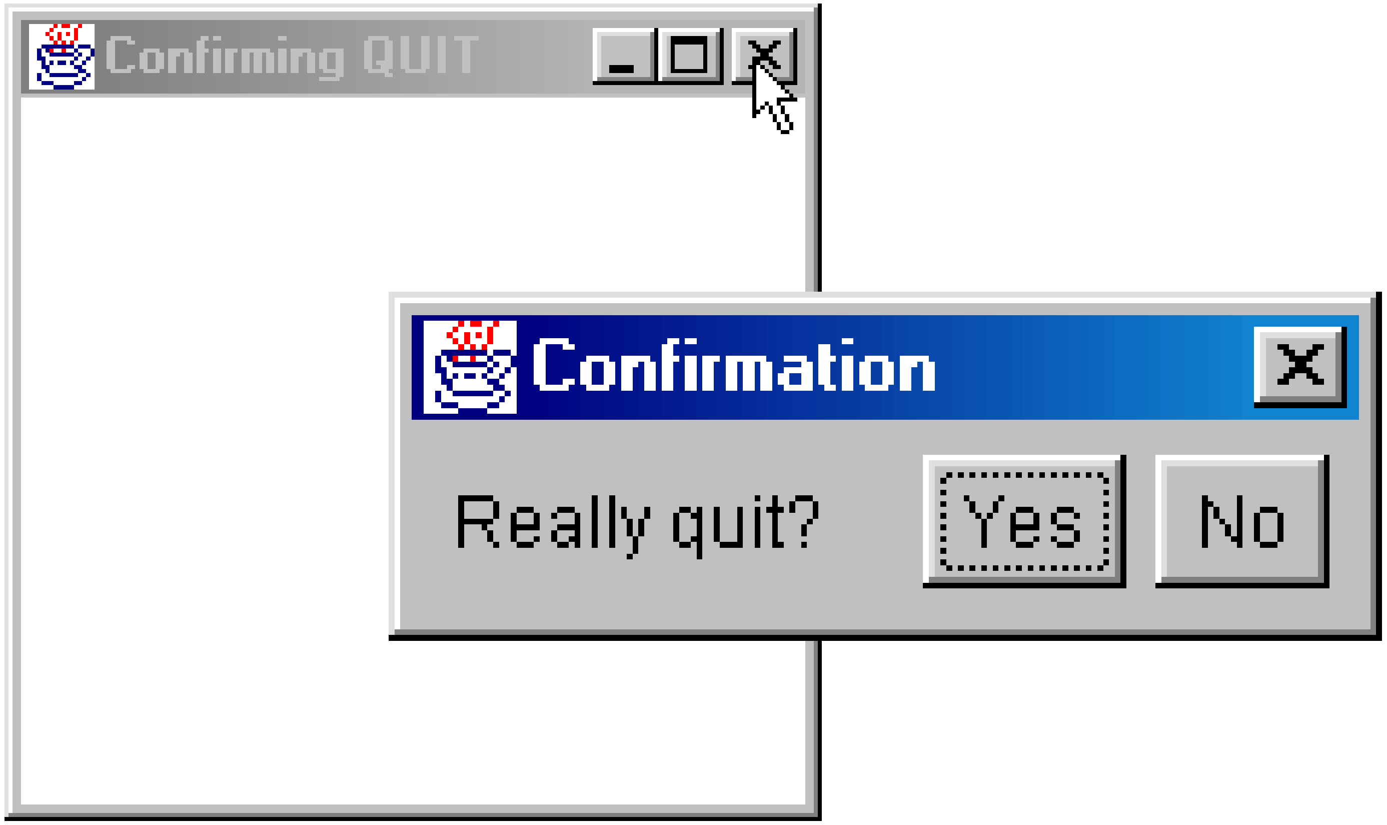
Using Confirmation Dialog

```
public class ConfirmTest extends Frame {
  public static void main(String[] args) {
    new ConfirmTest();
  }

  public ConfirmTest() {
    super("Confirming QUIT");
    setSize(200, 200);
    addWindowListener(new ConfirmListener());
    setVisible(true);
  }

  public ConfirmTest(String title) {
    super(title);
  }
```

Using Confirmation Dialog (Continued)

```
  private class ConfirmListener extends WindowAdapter {
    public void windowClosing(WindowEvent event) {
      new Confirm(ConfirmTest.this);
    }
  }
}
```

A Confirmation Dialog: Result



Modal dialogs freeze interaction with all other Java components

Serializing Windows

- **Serialization of Objects**
  - Can save state of serializable objects to disk
  - Can send serializable objects over the network
  - All objects must implement the `Serializable` interface
    - The interface is a marker; doesn't declare any methods
    - Declare data fields not worth saving as `transient`
- **All AWT components are serializable**

Serialization, Writing a Window to Disk

```
try {
  File saveFile = new File("SaveFilename");
  FileOutputStream fileOut =
    new FileOutputStream(saveFile);
  ObjectOutputStream out =
    new ObjectOutputStream(fileOut);
  out.writeObject(someWindow);
  out.flush();
  out.close();
} catch(IOException ioe) {
  System.out.println("Error saving window: " + ioe);
}
```

Serialization, Reading a Window from Disk

```
try {
  File saveFile = new File("SaveFilename");
  FileInputStream fileIn =
    new FileInputStream(saveFile);
  ObjectInputStream in =
    new ObjectInputStream(fileIn);
  someWindow = (WindowType)in.readObject();
  doSomethingWith(someWindow); // E.g. setVisible.
} catch(IOException ioe) {
  System.out.println("Error reading file: " + ioe);
} catch(ClassNotFoundException cnfe) {
  System.out.println("No such class: " + cnfe);
}
```

AWT GUI Controls

- **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

GUI Event Processing

- **Decentralized Event Processing**
  - Give each component its own event-handling methods
  - The user of the component doesn't need to know anything about handling events
  - The kind of events that the component can handle will need to be relatively independent of the application that it is in
- **Centralized Event Processing**
  - Send events for multiple components to a single listener
    - The (single) listener will have to first determine from which component the event came before determining what to do about it

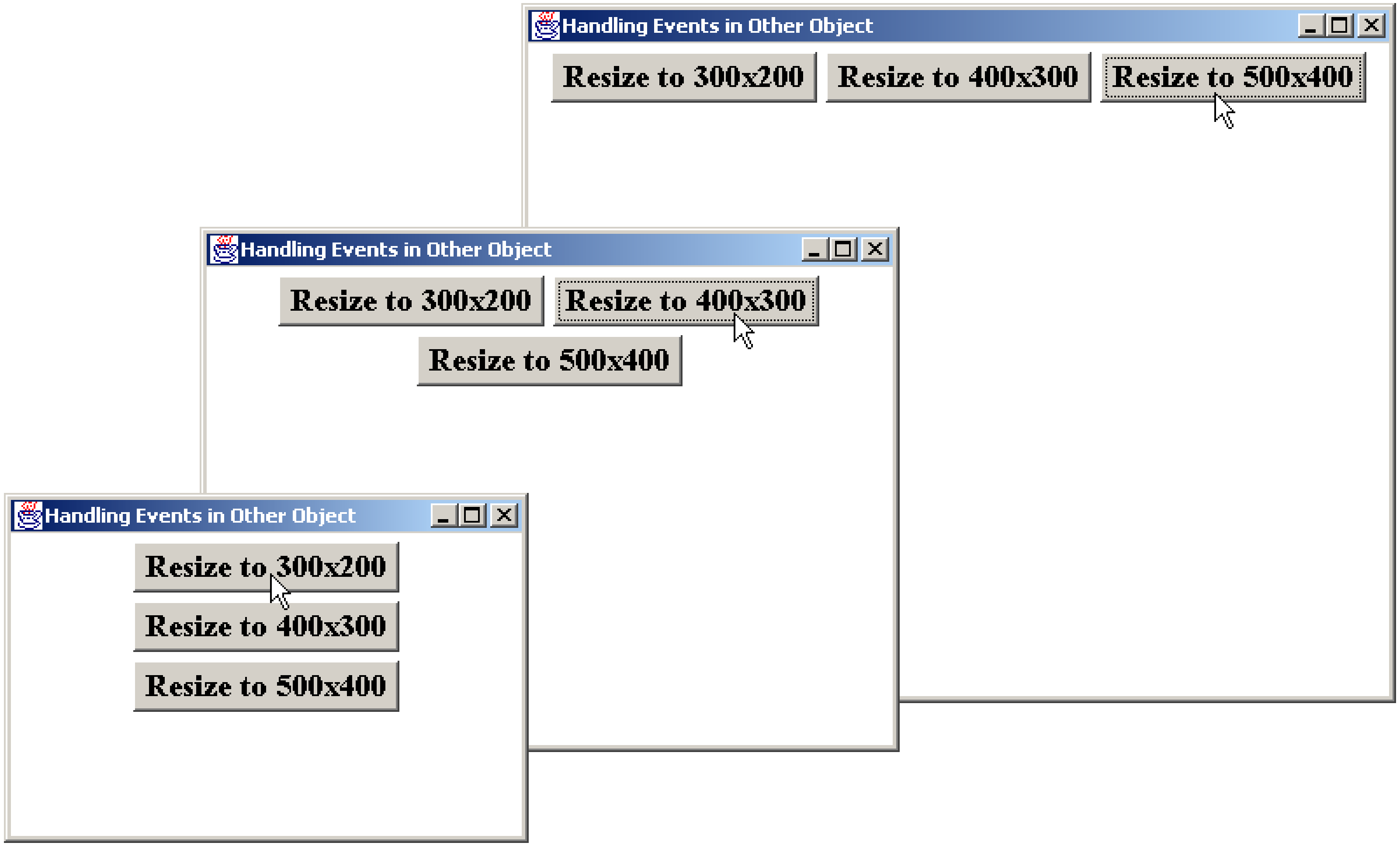
Decentralized Event Processing: Example

```
import java.awt.*;

public class ActionExample1 extends CloseableFrame {
  public static void main(String[] args) {
    new ActionExample1();
  }

  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);
  }
}
```

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();
  }
}
```

Decentralized Event Processing: Result



Centralized Event Processing, Example

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

public class ActionExample2 extends CloseableFrame
                            implements ActionListener {
  public static void main(String[] args) {
    new ActionExample2();
  }

  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);
```

Centralized Event Processing: Example (Continued)

```
    ...
    setSize(400, 300);
    setVisible(true);
  }

  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();
  }
}
```

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

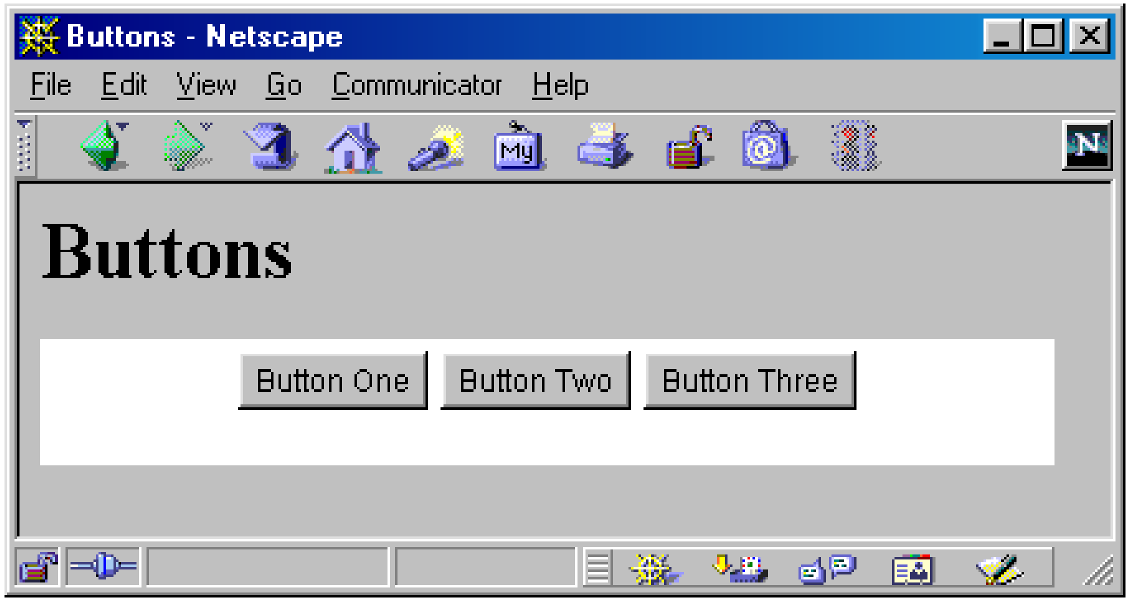
```
someButton.setLabel("A New Label");
someButton.getParent().invalidate();
someButton.getParent().validate();
```

Buttons (Continued)

- **Event Processing Methods**
  - addActionListener/removeActionListener
    - Add/remove an **`ActionListener`** that processes **`ActionEvent`**s 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);
  }
}
```



Handling Button Events

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

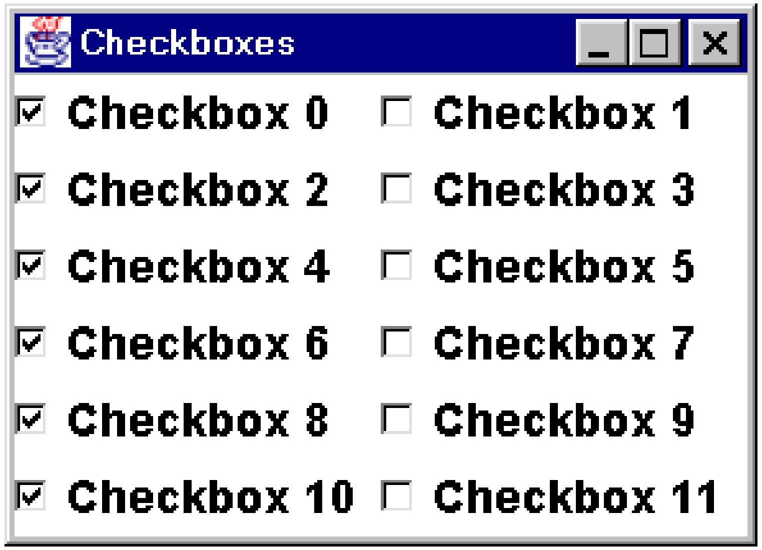
```
public class MyActionListener
             implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    ...
  }
}

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

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);
  }
}
```


Checkboxes

✓ Checkbox 0	☐ Checkbox 1
✓ Checkbox 2	☐ Checkbox 3
✓ Checkbox 4	☐ Checkbox 5
✓ Checkbox 6	☐ Checkbox 7
✓ Checkbox 8	☐ Checkbox 9
✓ Checkbox 10	☐ Checkbox 11

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 `ItemEvent`s in `itemStateChanged`
- **processItemEvent(ItemEvent event)**
  - Low-level event handling

Handling Checkbox Events

- **Attach an `ItemListener` through `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

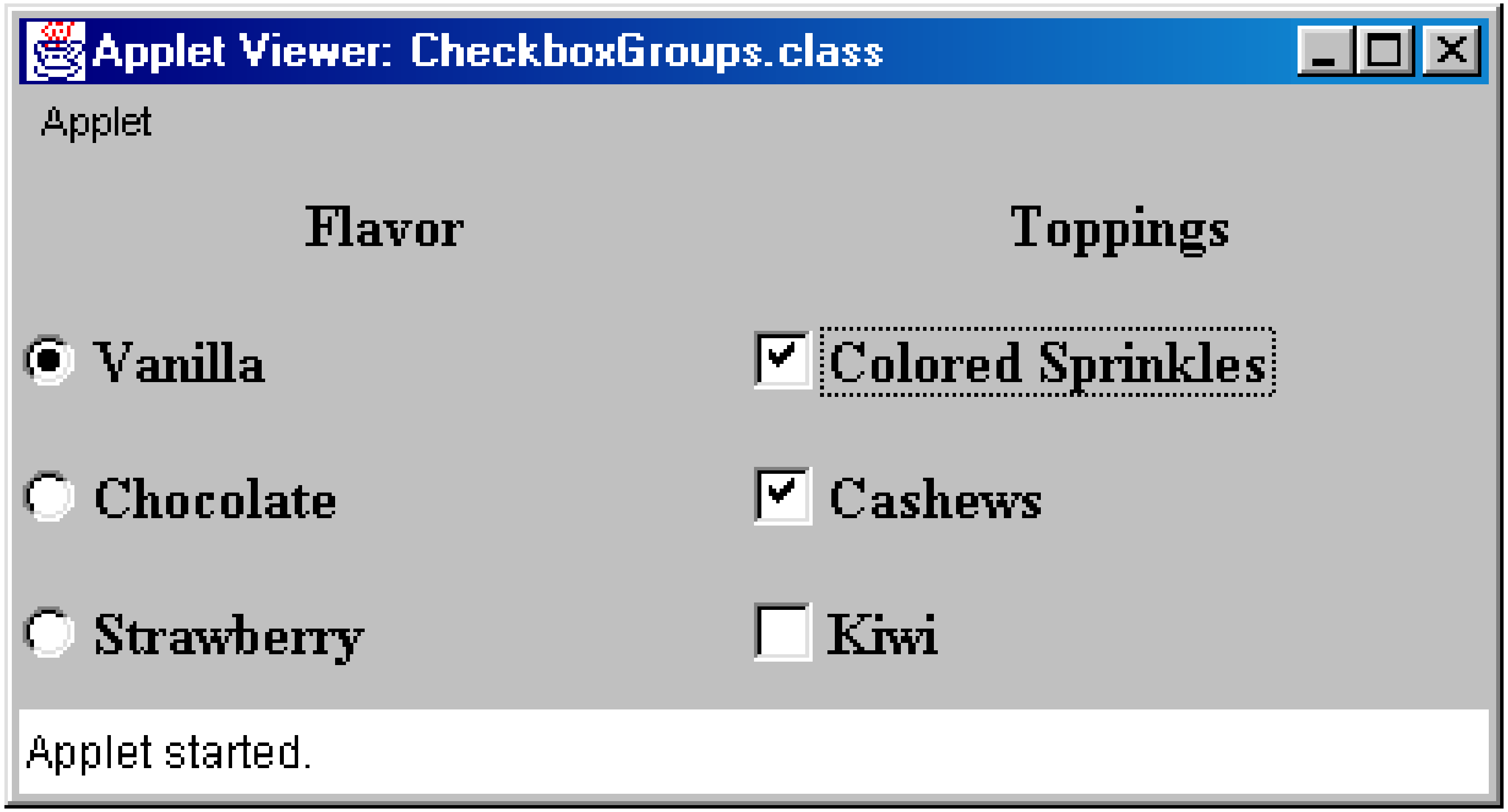
Checkbox Groups (Radio Buttons)

- **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

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"));
  }
}
```

CheckboxGroup, Result



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

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**

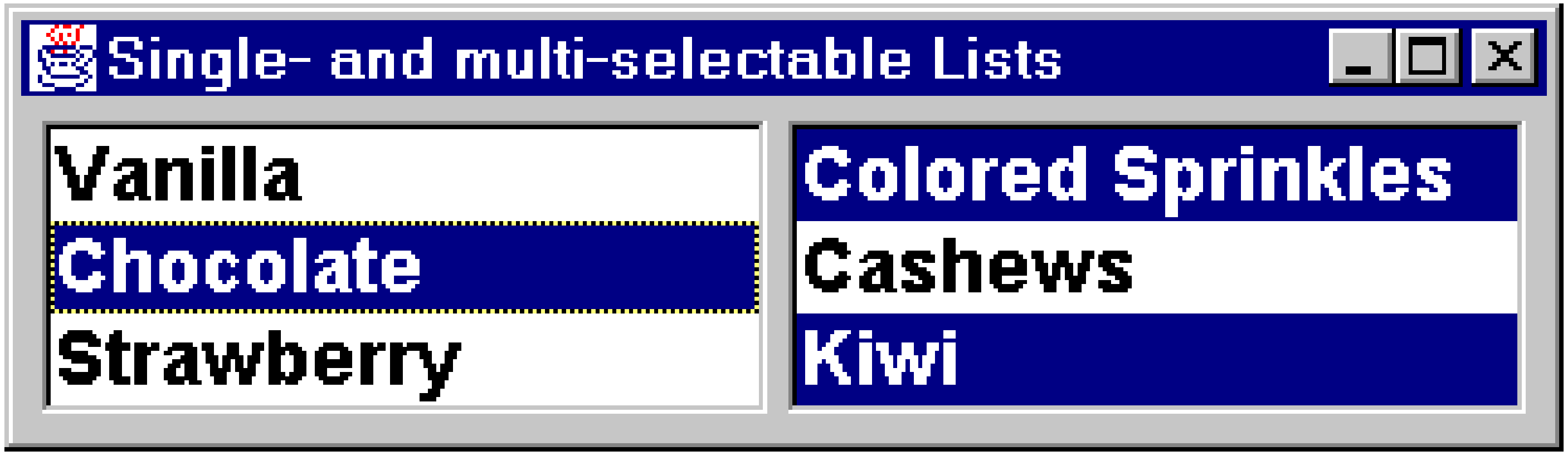
List Boxes

- **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

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);
  }}
```

List Boxes: Result



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

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

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

Handling List Events

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

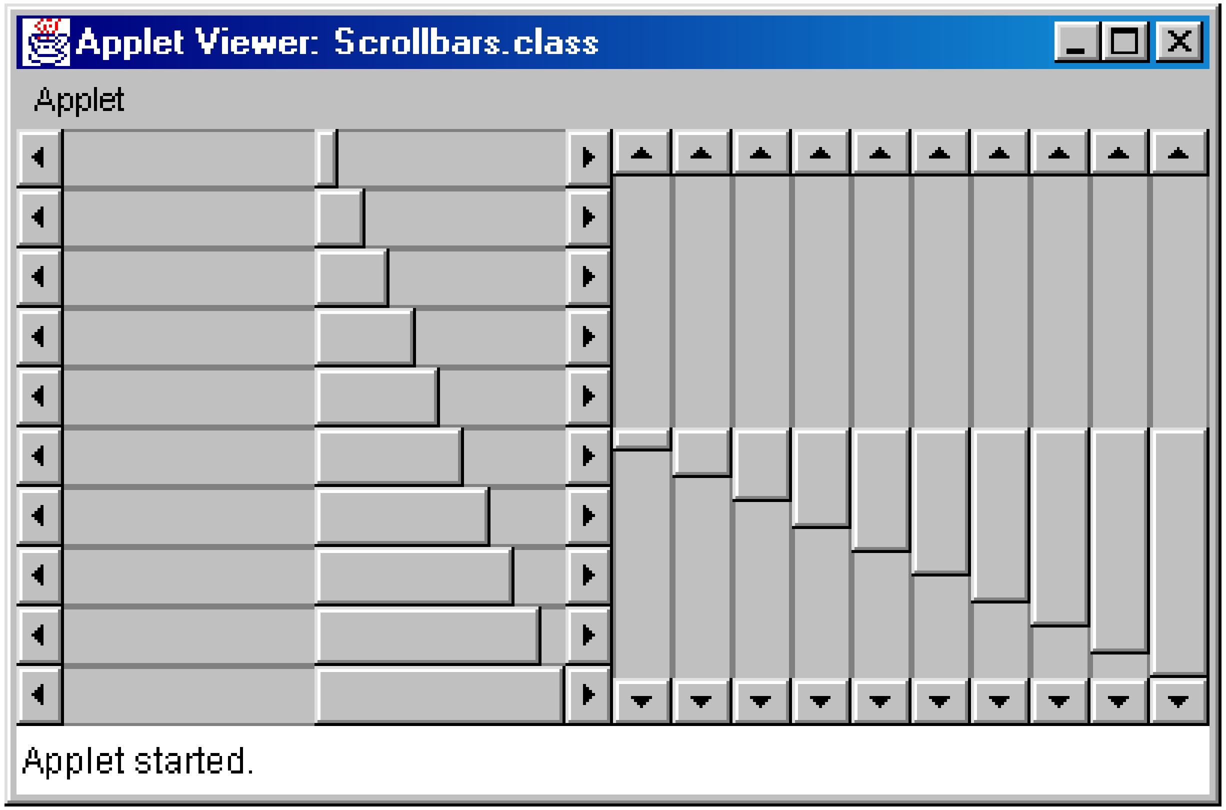
Scrollbars and Sliders

- **Constructors**
  - Scrollbar
    - Creates a vertical scrollbar
    - The "bubble" (or "thumb," the part that actually moves) size defaults to 10% of the trough length
    - The internal min and max values are set to zero
  - Scrollbar(int orientation)
    - Similar to above; specify a horizontal (Scrollbar.HORIZONTAL) or vertical (Scrollbar.VERTICAL) scrollbar
  - Scrollbar(int orientation, int initialValue,
    int bubbleSize, int min, int max)
    - Creates a horizontal or vertical "slider" for interactively selecting values
    - Specify a customized bubble thickness and a specific internal range of values
    - Bubble thickness is in terms of the scrollbar's range of values, not in pixels, so if max minus min was 5, a bubble size of 1 would specify 20% of the trough length

Scollbars: Example

```
public class Scrollbars extends Applet {
  public void init() {
    int i;
    setLayout(new GridLayout(1, 2));
    Panel left = new Panel(), right = new Panel();
    left.setLayout(new GridLayout(10, 1));
    for(i=5; i<55; i=i+5) {
      left.add(new Scrollbar(Scrollbar.HORIZONTAL,
                             50, i, 0, 100));
    }
    right.setLayout(new GridLayout(1, 10));
    for(i=5; i<55; i=i+5) {
      right.add(new Scrollbar(Scrollbar.VERTICAL,
                              50, i, 0, 100));
    }
    add(left);
    add(right);
  }
}
```

Scrollbars: Result



Scrollbars with varying bubble sizes, but constant ranges and initial values, shown on Windows 98

Handling Scrollbar Events

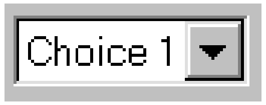
- **AdjustmentListener**
  - Attach an `AdjustmentListener` through `addAdjustmentListener` and process the `AdjustmentEvent` in `adjustmentValueChanged`

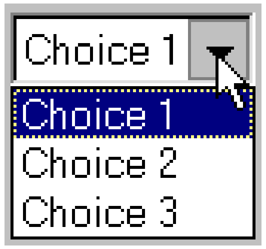
```
public void adjustmentValueChanged
                      (AdjustmentEvent event) {
    ...
}
```

- **Use ScrollPane**
  - If you are using a Scrollbar only to implement scrolling, a ScrollPane is much simpler
- **JSlider (Swing) is much better**

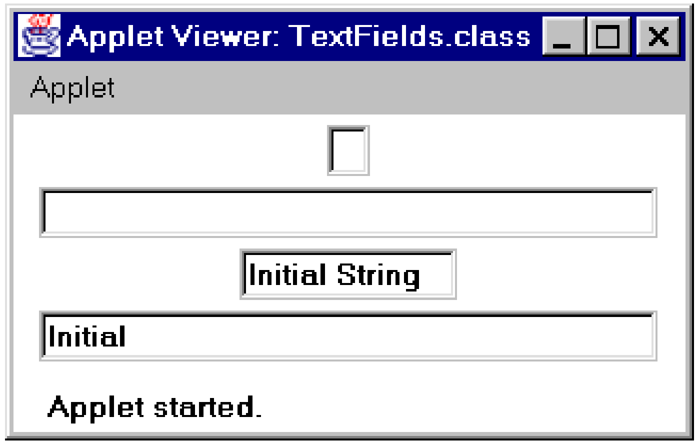
Other GUI Controls

- **Choice Lists (Combo Boxes)**





- **Textfields**



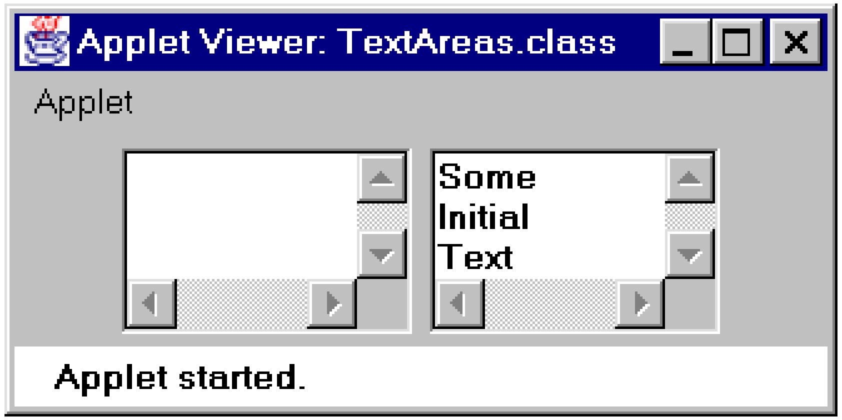
Initial String

Initial

Applet started.

Other GUI Controls (Continued)

- Text Areas



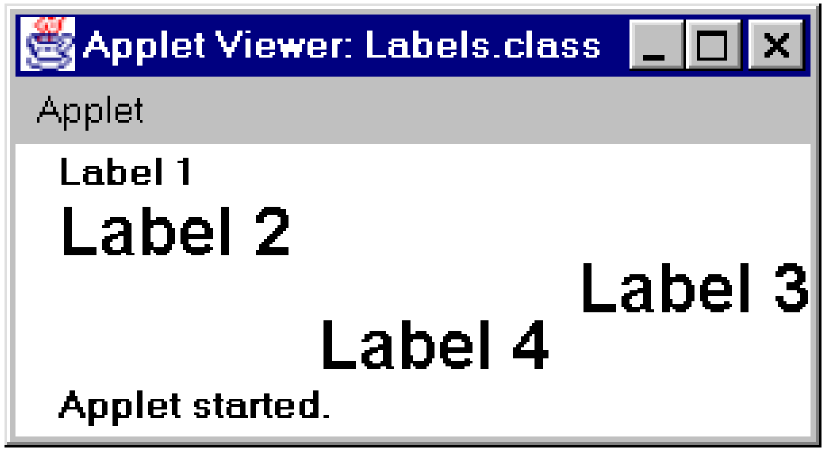
Applet Viewer: TextAreas.class

Applet

Some
Initial
Text

Applet started.

- Labels



Applet Viewer: Labels.class

Applet

Label 1
Label 2
Label 3
Label 4

Applet started.

Summary

- In the AWT, all windows and graphical components are rectangular and opaque
- Canvas: drawing area or custom component
- Panel: grouping other components
- Frame: popup window
- 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
  - add*Xxx*Listener methods are at the top

core WEB programming

Questions?