More on GUIs

CS211

Fall 2000

Three Uses of Interfaces

- Declaring the characteristics of an ADT
  - Examples: Collection, Set, List
- Declaring a single characteristic that is shared by several top-level, stand-alone classes (e.g., String, Integer)
  - Examples: Comparable, Cloneable
- Declaring a single helper method that is thus *attached* to an object; the object can be stored or passed as an argument
  - Examples: Comparator, various Event Listeners

Comparable vs. Comparator

- In general, a class that implements Comparable has many other methods
- In general, a class that implements Comparator contains just the single method compare( )
- This isn't quite true because a single class can be made to serve multiple purposes (i.e., you can define a single class that implements Comparable, Comparator, Set, and ActionListener)

Comparing Objects x and y

- Intuition

```
if (x < y)…
```

- When you want the natural order

```
// Can throw ClassCastException
Comparable cx = (Comparable)x;
if (cx.compareTo(y) < 0)…
```

- When using a Comparator (com)

```
// Can throw a ClassCastException
if (com.compare(x,y) < 0)…
```

- When you *might* be using a Comparator

```
if (com == null) {
    Comparable cx = (Comparable)x;
    if (cx.compareTo(y) < 0)…
    }
else
    if (com.compare(x,y) < 0)…
```

Creating a Comparator

- java.awt.Point defines a 2D point with integer coordinates
- If p is a Point then
  - p.x is the x-coordinate
  - p.y is the y-coordinate
- Goal: use lexicographic ordering (i.e., the first coordinate determines the order, but if they're tied, use the second coordinate)

```
import java.awt.Point;

class MyPointComparator implements
    Comparator {

public int compare (Object a, Object b) {
    Point pa = (Point) a;
    Point pb = (Point) b;
    if (pa.x <p b.x) return –1;
    if (pa.x > pb.x) return 1;
    if (pa.y < pb.y) return –1;
    if (pa.y > pb.y) return 1;
    return 0;
    }
}
```

Storing Points in a SortedSet

- If we store Points in a SortedSet then it's easy to print them in lexicographic order
- Assume we start with an array p of Points
- Attempt #1:

  SortedSet s = new SortedSet(java.util.Arrays.asList(p));

  - This fails (throws a ClassCastException) because Points are not Comparable
- Attempt #2:

  SortedSet s = new SortedSet(new MyPointComparator());

  s = s.addAll(java.util.Arrays.asList(p)); // Bulk add

  - This succeeds because we provided a Comparator
  - If I try to put a non-Point into s then my compare() method throws a ClassCastException

Recall: Graphical User Interfaces

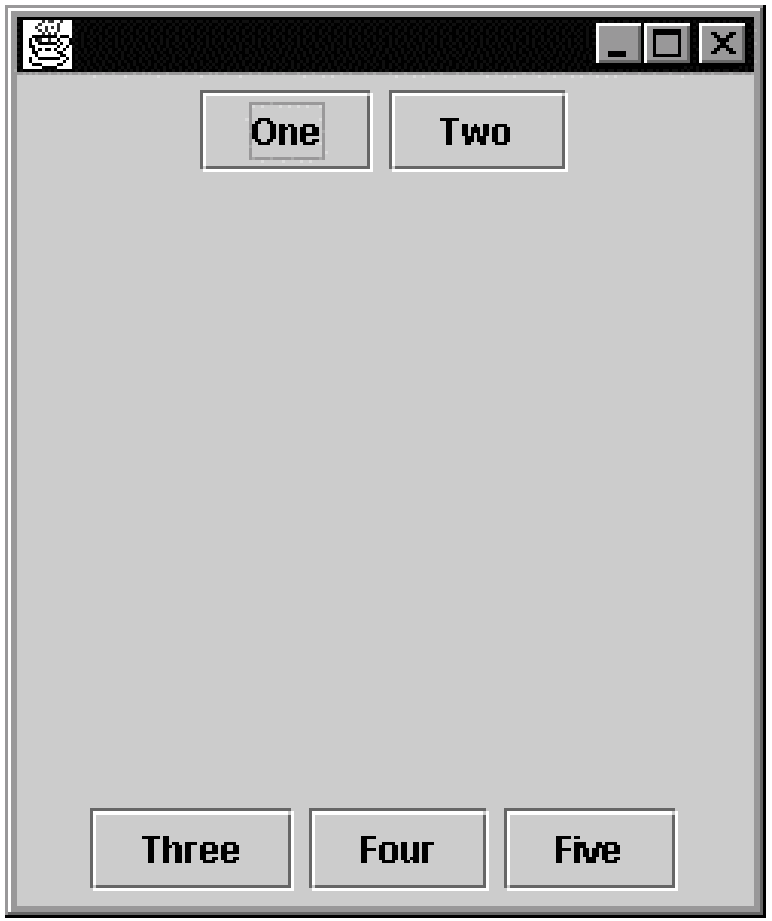
- Layout
  - How items are arranged
  - There are *lots* of predefined GUI items

    JButton, JLabel, JCheckbox, JList, JScrollbar,…
  - You have to write the code that determines layout
  - In Java, you use LayoutManagers to help with layout
- Event Handling
  - An *event* is (generally) a user input or action
  - The JVM (Java Virtual Machine) takes care of generating events

    Button pushed, text typed, mouse clicked,…
  - You have to write the code that determines how your program responds to an event

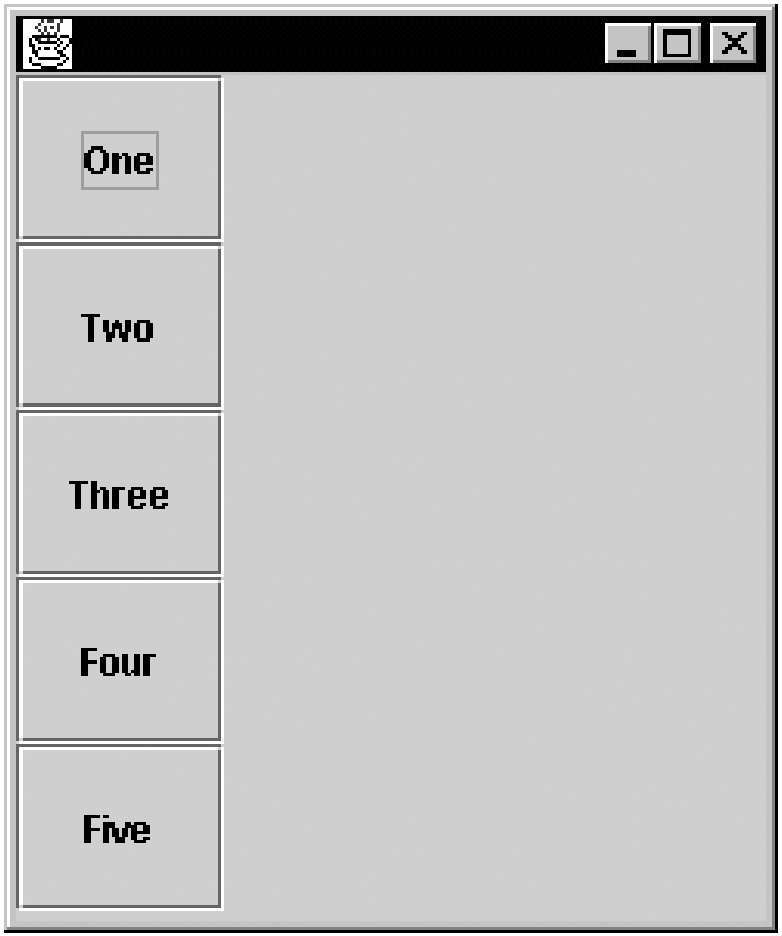
Using Panels to Group Components

```
public static void main (String[ ] args) {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    JPanel topPanel = new JPanel();
    JPanel botPanel = new JPanel();
    topPanel.add(new JButton("One"));
    topPanel.add(new JButton("Two"));
    botPanel.add(new JButton("Three"));
    botPanel.add(new JButton("Four"));
    botPanel.add(new JButton("Five"));
    panel.setLayout(new BorderLayout());
    panel.add(topPanel,BorderLayout.NORTH);
    panel.add(botPanel,BorderLayout.SOUTH);
    frame.getContentPane().add(panel);
    frame.setSize(250,300);
    frame.setVisible(true);
    }
```



Example: A Column of Buttons

```
import javax.swing.*;
import java.awt.GridLayout;
import java.awt.BorderLayout;
class GUITest {
public static void main (String[ ] args) {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setLayout(new GridLayout(0,1));
      panel.add(new JButton("One"));
      panel.add(new JButton("Two"));
      panel.add(new JButton("Three"));
      panel.add(new JButton("Four"));
      panel.add(new JButton("Five"));
      frame.getContentPane().add(
            panel,BorderLayout.WEST);
      frame.setSize(250,300);
      frame.setVisible(true);
      } }
```



When an Event Occurs…

- The JVM (Java Virtual Machine) determines the event's *source* and *type*
  - The *source* is the component from which the event originated
  - Each source has certain types of events it can generate
- The JVM looks for one or more *event listeners* that have *registered* with the source
  - An *event listener* is an object that implements one of the *Listener interfaces* in **java.awt.event** or in **javax.swing.event**
  - You *register* an event listener by using one of the component's *addListener* methods
- The JVM creates an *event object* using one of the classes in **java.awt.event** or in **javax.swing.event**
- For each registered event listener, the JVM invokes the listener's event-handling method and passes the event object as the parameter

Example: Color Buttons

```
import javax.swing.*; import java.awt.event.*;
import java.awt.Color;
class GUITest {
static String[ ] name = {"red","blue","green","magenta","cyan","yellow"};
static Color[ ] color =
    {Color.red,Color.blue,Color.green,Color.magenta,Color.cyan,Color.yellow};
public static void main (String[ ] args) {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    for (int i = 0; i < name.length; i++) {
            JButton button = new JButton(name[i]);
            panel.add(button);
            button.addActionListener(new MyListener(panel,color[i]));
            }
    frame.getContentPane().add(panel);
    frame.setSize(250,300);
    frame.setVisible(true);
    }
```

java.awt.Color

- Class for color manipulation
- Constants
  - black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow
- Constructors (2 of several)
  - Color (int r, int g, int b); // 0 to 255
  - Color (float r, float g, float b); // 0.0 to 1.0
- Methods
  - lighter( ), darker( ), getRed( ), getGreen( ), getBlue( ), getHSBColor( ),…

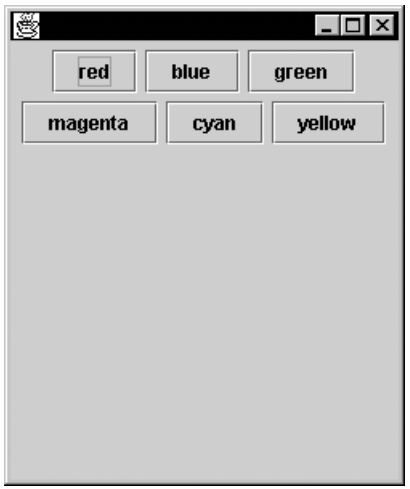
The ActionListener for the Buttons

```
static class MyListener implements ActionListener {

    Color myColor;
    JPanel myPanel;

    MyListener (JPanel panel, Color color) {
        myColor = color;
        myPanel = panel;
        }

    public void actionPerformed (ActionEvent event) {
        myPanel.setBackground(myColor);
        }
    }
}
```



What Actually Happens?

- The user clicks a button
- The JVM examines the button to see if any ActionListeners have registered with the button
  - If the button is supposed to do something, an ActionListener must have registered with the button *prior* to the button-click
  - Use button.addActionListener(…) to register
  - You can register a Listener at any time, but this is typically done when the button is created (as in the example)
- The JVM creates an ActionEvent (call it e)
- For each registered ActionListener x, the JVM calls x.actionPerformed(e)
  - This is where/when your code (telling what should happen when the button is clicked) is executed
  - In the example, the ActionListener remembers the panel and a color when it's created; when the action is performed (i.e., the button is clicked), the panel's background color is changed to myColor

What Do You Have to Code?

- Create a class that implements the correct *Event Listener* interface
  - Look at the documentation for the type of event; the Swing Tutorial is a better source for this than the API
  - The interface specifies one or more methods that are meant to respond to the event; you write code for these methods
  - Example: class MyListener which contains method actionPerformed
- *Register* an instance of your listener with the source of the (potential) event
  - Example: button.addActionListener(new MyListener(panel,color[i]));

Some Example Events

ActionEvent	User clicks a button, presses *Return* while typing in a text field, or chooses a menu item
WindowEvent	User closes a frame (main window)
MouseEvent	User presses a mouse button or moves the mouse
KeyEvent	User has pressed or released a key or typed a character
ComponentEvent	Component becomes visible
FocusEvent	Component gets the keyboard focus
ListSelectionEvent	Table or list selection changes

Anonymous Inner Classes

- In the the example where we compared Points, we created a class that was only instantiated once (i.e., we only created a single instance of that class)
- In such a situation, you can use an *anonymous inner class*
  - Syntax

```
new nameOfParentClass(constructorArgs) {
        methodAndFieldDeclarations
        }
```

  - Or

```
new nameOfInterface( ) {
        methodDeclarations
        }
```

Using an Anonymous Inner Class

```
SortedSet s = new SortedSet ( new Comparator( ) {
                                    public int compare (Object a, Object b) {
                                            Point pa = (Point) a;
                                            Point pb = (Point) b;
                                            if (pa.x <p b.x) return –1;
                                            if (pa.x > pb.x) return 1;
                                            if (pa.y < pb.y) return –1;
                                            if (pa.y > pb.y) return 1;
                                            return 0;
                                            }
                                } );
s = s.addAll(java.util.Arrays.asList(p));    // Bulk add
```

- Anonymous inner classes show up a lot in GUI code

Example: Button with Counter

```
class CountWindow extends JFrame {
private final String labelPrefix = "Number of clicks: ";
private JLabel label;
private int count = 0;
public CountWindow () {
    JPanel panel = new JPanel();
    panel.setLayout(new GridLayout(0,1));
    JButton button = new JButton("Click here!");
    panel.add(button);
    panel.add(label = new JLabel(labelPrefix + count));
    button.addActionListener(
                    new ActionListener() {
                            public void actionPerformed (ActionEvent e) {
                                    label.setText(labelPrefix + (++count));
                                    }
                            } );
    this.getContentPane().add(panel);
    }
```

Continued: Button with Counter

```
public static void main (String[ ] args) {
      JFrame frame = new CountWindow();
      frame.setSize(150,100);
      frame.setVisible(true);
      }
}
```

Adapters

- Some Event Listeners have several methods
  - MouseListener
    - MouseClicked()
    - MouseEntered()
    - MouseExited()
    - MousePressed()
    - MouseReleased()
  - WindowListener
    - WindowActivated()
    - WindowClosed()
    - WindowClosing()
    - WindowDeactivated()
    - WindowDeiconified()
    - WindowIconified()
    - WindowOpened()
- It's tedious to write all these methods when, in most situations, all but one of them do nothing at all
- Java provides *adapters*
  - An *adapter* is a class that implements all the methods using stubs
  - You can extend the adapter and override the one method you care about
- Examples
  - MouseAdapter
  - WindowAdapter

Using Adapters; a Closeable Window

```
class CloseableWindow extends JFrame {
private JPanel panel;
public CloseableWindow () {
    panel = new JPanel(); this.getContentPane().add(panel);
    this.addMouseListener(new MouseAdapter() {
        public void mouseClicked (MouseEvent e) {
            panel.setBackground(Color.getHSBColor((float)Math.random(),1,1));
        }
    });
    this.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}
public static void main (String[ ] args) {
    JFrame frame = new CloseableWindow(); frame.setSize(150,150); frame.setVisible(true);
}}
```