



Polymorphism & A Few Java Interfaces

Rick Mercer


Outline



- ◆ Describe Polymorphism
- ◆ Show a few ways that interfaces are used
 - Respond to user interaction with a GUI with **ActionListener**
 - Compare objects with **Comparator**
 - Tag types to have writeable/readable objects with **Serializable**
 - Create our own icons with **Icon**
 - Play audio files with **AudioClip**

Polymorphism

<http://www.webopedia.com/TERM/p/polymorphism.html>



- ◆ In general, polymorphism is the ability to appear in many forms
- ◆ In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type (class)
- ◆ Polymorphism is considered to be a requirement of any true object-oriented programming language

Polymorphism from mercer



To understand polymorphism, take an example of a workday at Franklin, Beedle, and Associates. Kim brought in pastries and everyone stood around chatting. When the food was mostly devoured, Jim, the president of the company, invited everyone to “Get back to work.” Sue went back to read a new section of a book she was editing. Tom continued laying out a book. Stephanie went back to figure out some setting in her word-processing program. Ian finished the company catalog.

Polymorphism



Jeni met with Jim to discuss a new project. Chris began contacting professors to review a new manuscript. And Krista continued her Web search to find on whether colleges are using C++, Python, or Java. Marty went back to work on the index of his new book. Kim cleaned up the pastries. Rick's was just visiting so he went to work on the remaining raspberries.

Polymorphic Messages



- ◆ 10 different behaviors with the same message!
- ◆ The message “Get back to work” is a *polymorphic message*
 - a message that is understood by many different types of object (or employees in this case)
 - but responded to with different behaviors based on the type of the employee: Editor, Production, Marketing, ...

Polymorphism



- ◆ Polymorphism allows the same message to be sent to different types to get different behavior
- ◆ In Java, polymorphism is possible through
 - inheritance
 - Override `toString` to return different values that are textual representations of that type.
 - interfaces
 - `Collections.sort` sends `compareTo` messages to objects that must have implemented `Comparable<T>`

Polymorphism



- ◆ The runtime message finds the correct method
 - same message can invoke different methods
 - the reference variable knows the type
 - `aString.compareTo(anotherString)`
 - `anInteger.compareTo(anotherInteger)`
 - `aButton.actionPerformed(anEvent)`
 - `aTextField.actionPerformed(anEvent)`
 - `aList.add(anObject)`
 - `aHashSet.add(anObject)`

The Java Interface



- ◆ An interface describes a set of methods
 - NOT allowed: constructors, instance variables
 - static variables and methods are allowed
- ◆ Interfaces must be implemented by a class
 - 646 classes implement ≥ 1 interfaces (in '02)
- ◆ Typically, two or more classes implement the same interface
 - Type guaranteed to have the same methods
 - Objects can be treated as the same type
 - May use different algorithms / instance variables

An interface we'll use soon

- ◆ An **interface**, a reference type, can have
 - **static** variables and method headings with ;
`public int size(); // no { }`
- ◆ Methods are implemented by 1 or more classes
- ◆ Example **interface**

```
public interface ActionListener {  
    public void actionPerformed(ActionEvent theEvent);  
}
```

Multiple classes implement the same interface

- ◆ To implement an **interface**, classes must have all methods specified as given in the interface

```
private class Button1Listener implements ActionListener {  
    public void actionPerformed(ActionEvent theEvent) {  
        // Do this method when button1 is clicked  
    }  
}
```

```
private class Button2Listener implements ActionListener {  
    public void actionPerformed(ActionEvent theEvent) {  
        // Do this method when button2 is clicked  
    }  
}
```

The Comparable interface

A review for most

- ◆ Can assign an instance of a class that implements and interface to a variable of the interface type

```
Comparable str = new String("abc");  
Comparable acct = new BankAccount("B", 1);  
Comparable day = new Date();
```

- ◆ **Some classes that implement Comparable**

```
BigDecimal BigInteger Byte ByteBuffer Character  
CharBuffer Charset CollationKey Date Double  
DoubleBuffer File Float FloatBuffer IntBuffer  
Integer Long LongBuffer ObjectStreamField  
Short ShortBuffer String URI
```

- ◆ Comparable defines a “natural ordering”

Interface Comparable<T>



- ◆ Any type can implement Comparable to determine if one object is less than, equal or greater than another

```
public interface Comparable<T> {  
    /**  
     * Return 0 if two objects are equal; less than  
     * zero if this object is smaller; greater than  
     * zero if this object is larger.  
     */  
    public int compareTo(T other);  
}
```

interface comparator

```
/**  
 * Compares its two arguments for order. Returns a  
 * negative integer, zero, or a positive integer as the  
 * first argument is less than, equal to, or greater  
 * than the second argument. Equals not shown here  
 */  
public interface comparator<T> {  
    public int compareTo(T other);  
}
```

- ◆ Can specify sort order by objects. In the code below
 - What class needs to be written?
 - What interface must that new class implement?

```
Comparator<BankAccount> idComparator = new ByID();  
Collections.sort(accounts, idComparator);
```

Example



```
import java.util.Comparator;
/**
    A type that can be instantiated and sent as an
    argument to help sort the objects using this
    strategy: acct1 < acct2 if acct1's ID precedes
    acct2's ID alphabetically
 */
import java.util.Comparator;

public class ByID implements Comparator<BankAccount> {

    public int compare(BankAccount b1, BankAccount b2) {
        String id1 = b1.getID();
        String id2 = b2.getID();
        return id1.compareTo(id2);
    }
}
```



```
import java.util.Comparator;
```

```
/**
```

```
    A type that can be instantiated and sent as an
    argument to help sort the objects using this
    strategy: acct1 < acct2 if acct1's balance is less
    than acct2's balance.
```

```
*/
```

```
public class ByBalance implements
```

```
    Comparator<BankAccount> {
```

```
    public int compare(BankAccount b1, BankAccount b2) {
```

```
        double balance1 = b1.getBalance();
```

```
        double balance2 = b2.getBalance();
```

```
        return (int) (balance1 - balance2);
```

```
    }
```

```
}
```


Two sorting strategies

```
// Sort by ID
Comparator<BankAccount> idComparator = new ByID();
Collections.sort(accounts, idComparator);
// First element has the alphabetically first ID
System.out.println(accounts.toString());

// Sort by balance
idComparator = new ByBalance();
Collections.sort(accounts, idComparator);
// First element has the account with the least money
System.out.println(accounts.toString());
```

Output:

```
[A $5.00, B $100.00, C $3,000.00, D $200.00, E $50.00]
[A $5.00, E $50.00, B $100.00, D $200.00, C $3,000.00]
```

class OurIcon implements Icon

```
Icon myIcon = new LiveCamImage("http://www.cs.arizona.edu/  
camera/view.jpg");
```

```
JOptionPane.showMessageDialog(  
    null,  
    "View from\nthe UofA\nComputer Science\nDepartment",  
    "Message",  
    JOptionPane.INFORMATION_MESSAGE,  
    myIcon);
```

- ◆ Notice the 5th parameter type, class or interface?

```
public static void showMessageDialog(Component parentComponent,  
    Object message, String title, int messageType, Icon icon)  
    throws HeadlessException
```

LiveCamImage

```
public class LiveCamImage implements Icon {  
    private BufferedImage myImage;  
  
    public LiveCamImage(String imageFileName) {  
        try {  
            myImage =  
                javax.imageio.ImageIO.read(new URL(imageFileName));  
        } catch (IOException e) {  
            System.err.println("Could not load" + imageFileName);  
        }  
    }  
  
    // Control the upper left corner of the image  
    public void paintIcon(Component c, Graphics g, int x, int y) {  
        g.drawImage(myImage, 2, 2, null);  
    }  
  
    // Icon also specifies getIconWidth and getIconHeight  
    // See file in InterfaceExamples.zip
```

THE UNIVERSITY OF ARIZONA.
TUCSON, ARIZONA


www.cs.arizona.edu/camera
74F/23C 19-Sep-2014 06:35



View from
the UofA
Computer Science
Department


OK

interface AudioClip



- ◆ interface AudioClip has 3 methods
 - loop, play, stop
- ◆ The Applet class implements AudioClip
- ◆ Supports recording, playback, and synthesis of sampled audio and Musical Instrument Digital Interface (MIDI) sequences
 - Can play .au, .aif, .wav, .midi
 - For mp3s, we need something more complex
 - We'll see such a library later

interface AudioClip



```
AudioClip audioClip = null;
URL url = null;

// This assumes songs are in a folder named audio
// Need "file:" unless you are reading it over the web
String baseFolder = "file:" + System.getProperty("user.dir")
    + "/media/";

try {
    url = new URL(baseFolder + "wind.wav");
    audioClip = Applet.newAudioClip(url);
} catch (MalformedURLException e) {
    System.out.println("bad url " + url);
}
audioClip.play();
```