Polymorphism through the Java Interface

Rick Mercer
Outline

- Describe Polymorphism
- Show a few ways that interfaces are used
  - Compare objects with Comparator
  - Create our own icons with Icon
  - Play audio files with AudioClip
  - The role of interfaces in Java’s Collection Framework
Polymorphism

- 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 a requirement of any true object-oriented programming language
To understand polymorphism, take an example of a workday at Franklin, Beedle, and Associates. Kim brought in pastries while everyone was 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.
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 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
    - Example: Override `toString()` to return different values that are textual representations of that type.
  - interfaces
    - Example: `Collections.sort(List<?>)` 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)
An interface describes a set of methods
- class variables are allowed (need `static`)
- NOT allowed: constructors, instance variables

```java
public interface IBowlingLine {

    public static final int LAST_FRAME = 10;

    public abstract int scoreAtFrame(int frame);

    public int scoreSoFar();

    // Interface methods are implicitly abstract, so the
    // abstract modifier is not used with interface methods
    // (it could be—it's just not necessary).

    // Also public by default, so this would work
    void pinsDowned(int pins);
}
```
The Java Interface

- Interfaces are to be implemented by a class
  - ~ 33% of classes (about 1,000) in Java’s API implement one or more interfaces

- 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 / data structures
The Comparable interface remember?

- 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 implement Comparable
  - find out how many with Java’s API

- interface Comparable defines the "natural ordering" for collections
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> {
    int compare(T one, 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);

Sort using different Comparators
class OurIcon implements Icon

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

JOptionPane.showMessageDialog(null, "View from
the UofA
Computer Science
Department", "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
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
An interface you need now

- **An interface**, a reference type, can have
  - static variables and method headings with ;
    ```java
    public int size(); // no {} 
    ```

- Many classes may implement an interface
  - Use interface ActionListener to register any number of objects to respond to button clicks, menu selections, and input into a text field
    ```java
    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

```java
private class TheButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent theEvent) {
        // Do this method when a user clicks TheButton
    }
}

private class TheTextFieldListener implements ActionListener {
    public void actionPerformed(ActionEvent theEvent) {
        // Do this code when user presses the enter key in TheTextField when it has focus
    }
}
```