Implementing Interfaces
Defining Interfaces

Readings
This Week: Ch 8.3-8.8 and into Ch 9.1-9.3
(Ch 9.3-9.8 and Ch 11.1-11.3 in old 2nd ed)
Next Week: Ch 9.4-9.5 and into Ch 10.1-10.8
(Ch 11.4-11.5 and into Ch 13 in old 2nd ed)
(Reminder: Readings are absolutely vital for
learning this stuff!)

Labs and Tutorials
This Week: Lab #8 – A catch-up lab…
Next Week: Lab #9 is up on WebCT.

Survey #2
Dr. Ben Yu’s second survey for you all is up
on WebCT now.
Completely optional…
But, there’s a bit of extra credit for doing all
three surveys by their respective deadlines!
The deadline for this one is March 29.
Because of the server crash, if you did
this survey before the crash, you’ll have
to do it again. 😞

Assignment #3
- Will be up REAL SOON (tonight or
tomorrow)!
- Please check WebCT tomorrow and over
weekend!

Learning Goals
By the end of class today you will be able to…
- Write code that uses interfaces.
- Write code that implements interfaces.
- Declare your own interfaces.
Last Time: Sorting an Array

- Using Comparable, we can write the same code, that sorts different kinds of arrays!

Recap: Interfaces as Contracts

- If you buy a "USB" product, but the plug is shaped wrong, who is responsible?
- If you put 87 octane gas in a car that runs on diesel, who is responsible?
- If you buy a CD, but it actually installs secret spyware on your computer, who is responsible?
- Producer promises to supply certain features
- Consumer promises to use only those features.

Recap: Interfaces in Java

- Java has a similar concept of interfaces.
- An interface is a set of methods that must be available and work in a certain way.
- A class can be declared to implement an interface.
  - This is a promise by the writer of the class that it has those public methods available and that they behave properly.
  - Code can declare variables with the interface name (instead of a class name).
  - Java will let you use only the interface methods.
  - But now, your code will run with any class that implements that interface!

Different Roles

<table>
<thead>
<tr>
<th>Role</th>
<th>USB Analogy</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer</td>
<td>Learn what USB devices can do, look for the USB interface, and plug them in.</td>
<td>Read Java API for interfaces, declare variables with interface types, call the interface methods.</td>
</tr>
<tr>
<td>Producer</td>
<td>Understand the USB interface specification and build product that meets it.</td>
<td>Understand the interface declaration and implement its methods.</td>
</tr>
<tr>
<td>Standards Committee</td>
<td>Define USB standard.</td>
<td>Design and declare a new interface type.</td>
</tr>
</tbody>
</table>

Example: the Comparable interface

- Java provides an interface called Comparable.
  - Think of this like the name of a class.
  - Objects that implement Comparable must provide a compareTo() method:
    - Returns an int < 0 if this < argument
    - Returns 0 if this == argument
    - Returns an int > 0 if this > argument
  - Many classes implement Comparable: String, Integer, Double, etc.
Using an Interface (Consumer)
- Declare variables and parameters using interface name instead of class name, e.g.:
  ```java
  Comparable smallest = theArray[j];
  ```
  instead of
  ```java
  String smallest = theArray[j];
  ```
  or
  ```java
  BigInteger smallest = theArray[j];
  ```
- Java lets us use only the methods of the interface:
  ```java
  if ( smallest.compareTo(theArray[j]) > 0 ) …
  ```
  but not
  ```java
  if ( smallest.length() > 0 ) …
  ```

Implementing an Interface (Producer)
- Define a class as usual, but claim that it implements the interface, e.g.:
  ```java
  public class TalkingDoll0 implements Comparable {
      …
  }
  ```
- Define all methods needed for interface, e.g.:
  ```java
  public int compareTo(Object x) {
      …
  }
  ```

Defining an Interface (Standards Committee)
- If you want to create your own interface type, you must decide what public methods a class must provide, in order to implement your interface.

```java
public interface InterfaceName {
    … method signatures …
}
```

Example: Feedable
- Let's create a new interface type, called Feedable.
- All Feedable classes must have the following two methods:
  ```java
  public String getFavoriteFood()
  ```
  Returns the name of the object’s favorite food
  ```java
  public void feedFood(String food)
  ```
  Prints a message based on the food passed in.
- In the interface, leave off the “public” since they're always public.

Multiple Interfaces
- A physical object can have multiple interfaces
  - E.g., a digital camera with a USB port, a Firewire port, an S-video output, a CompactFlash slot, etc.
- A Java object can implement multiple interfaces
- Let’s make TalkingDoll0 implement Feedable

Polymorphism
- Polymorphism is just a fancy word for the idea that the same program text will work for different kinds of objects, adapting to whatever object it’s called on.
- The same call x.feedFood("dog food") can refer to 3 different methods.
- How does Java know which one to call?
Objects Know Their Class

- How did Java know which feedFood method to call?
- Java objects know what class they belong to.

You can check which class an object belongs to using the `instanceof` operator:

```java
if (x instanceof Dog)
    x.feedFood("dog food");
```