Static Interface Types

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
- Comp On Hold – Server Crash!
- But, there’s a bit of extra credit for doing all three surveys by their respective deadlines!
- (I think the deadline for this one is March 22, but check on WebCT to be sure.)

Assignment #3
- Will be up REAL SOON (before our next lecture)!
- Please check WebCT tomorrow and over weekend!

Learning Goals
By the end of class today you will be able to...
- Explain what “static” means in Java programs and why you might want to use static methods and fields.
- Explain what an interface is in Java and why they are useful to make reusable code.
- Write simple code that uses pre-defined interfaces.
The Meaning of `static`:
- From our very first Java program, we had: `public static void main (String[] args)`
- But what does `static` mean?

Review: Static Methods
- A method that is tagged as `static`:
  - Can be called on the class, rather than on an object of the class (e.g., `Math.sin(1.0)`)
  - Does not have an implicit parameter `this`
  - Cannot directly access non-static methods or fields (because there is no `this`!)

The Meaning of `static`:
- But what does `static` mean?
- In general, `static` means something that belongs to the class as a whole, and not to a specific object:
  - Static methods have no implicit parameter `this`
  - Static fields belong to the whole class, shared by all objects.

Java Classes
- When you write a class in Java, you describe what goes into each object of the class:
  ```java
  public class UBCStudent {
      String name;
      int studentID;
      public String getName() {...}
      ...
  }
  ```
  That says that each object of type UBCStudent contains a field `name`, and a field `studentID`, and the object knows how to do the `getName` method (applied to itself).

Analogy/Intuition on `static`
- A class is a pattern/blueprint/factory that shows, by example, how to create specific types of objects:
  - Each object has these instance fields...
  - Each object has these methods, that do the following statements...
- Sometimes, you want to talk about the pattern/blueprint itself, not the objects. Those are when you want to use `static`
Real Life Analogy of static Fields
- A factory/blueprint for a car is like the class. The cars are the objects.
- The blueprint shows an engine, battery, etc.
- Each car gets its own engine, battery, etc.
  - These are instance fields.
- But to give each car a serial number, you need a field that lives in the factory.
  - That should be a static field.

Example: UBCStudent with Automatically Assigned Student ID Numbers
- Each object should get a unique student ID number assigned to it.
- We'll keep a variable nextID that keeps track of the next available ID number.
- nextID is one variable for the whole class, so it should be static!

Recap: Static Fields
A field that is tagged as static:
- Is a single field of the class, rather than a separate field of each object of the class
- Is shared by all objects of the class
- Can be accessed without creating an object of the class
- Can be accessed from static methods

Why Use static?
- Static methods are almost never used, with two common exceptions:
  - Numbers aren’t objects in Java, so to have a class of methods that do computation, we’ll often put them in static methods (e.g., the Math class).
  - The main method starts running before there are any objects created, so it has to be static.

Why Use static?
- Static fields are not very common, but are needed when the class as a whole has to coordinate some information, e.g.:
  - Counting how many objects were created
  - Assigning serial numbers
  - Having all objects share or negotiate some information in common.

Recap: Variable Types
- Static variables
  - declared within class
  - associated with class, not instance, indefinite lifetime
- Instance variables
  - declared within class
  - associated with instance
  - accessible throughout object, lifetime of object
- Local variables
  - declared within method
  - accessible within method after declared, lifetime of method
  - (To be more precise, replace “method” by “block”)
- Parameters
  - declared in parameter list of method
  - accessible throughout method, lifetime of method
Questions

Interfaces in Real Life

- What does it mean
  - When a product says it’s “USB compatible”?
  - When a gas station sells “regular unleaded (87 octane) gas”?
  - When you buy a CD that says “Compact Disc Digital Audio”?

- The producer promises that the product has certain features and behaviors.
- If the user uses only those features and behaviors, then everything should work right.

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.

Interfaces Save Effort

- You can buy any USB devices and plug them in, as long as you obey the USB standard.
- You can buy regular gas from any gas station, as long as your car doesn’t demand some other fuel.
- You can buy a CD from any manufacturer, and it will play properly.
- Imagine if you had to negotiate the details of every purchase!

Interfaces in Java Save Effort, Too

- Java has a similar concept of interfaces.
- A class can be declared to implement an interface.
  - This is a promise by the writer of the class that it has certain public methods available that behave a certain way.
- 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!
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(Object x) method:
  - Returns an int < 0 if this < x
  - Returns 0 if this == x
  - Returns an int > 0 if this > x
- Many classes implement Comparable: String, Integer, Double, etc.

Example: Sorting an Array

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

("Sorting" means to put things in order. It makes it possible to find things in an array much faster, and it’s a major task for computers. You’ll learn in CPSC 320 good ways to sort. We’ll just do a simple way...)

Example: Sorting an Array

- We’ll see later/next time how to write our own classes that implement an interface, and how to design our own interfaces.

Questions

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>

Using an Interface (Consumer)

- Declare variables and parameters using interface name instead of class name, e.g.:
  - Comparable smallest = theArray[i];
  - instead of
    - String smallest = theArray[i];
    - or
    - UBCStudent smallest = theArray[i];
- Java lets us use only the methods of the interface:
  - if ( smallest.compareTo(theArray[i]) > 0 ) …
  - but not
    - 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 UBCStudent implements Comparable {
  ...
  }
  ```
- Define all methods needed for interface, e.g.:
  ```java
  public int compareTo(UBCStudent x) {
  ...
  }
  ```

Defining a New Interface
- 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:
  - `public String getFavoriteFood()`
    - Returns the name of the object’s favorite food
  - `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 UBCStudent 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?

instanceof
- 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");
  ```
Midterm Info

- Out of 100: raw avg=56, low=1, high=97
  - Will be scaled (info later)
- Long exam, but not unreasonable:
  - Everyone should be able to do it (with time).
  - Talk to friends, learning centre, WebCT, TAs, prof.
  - See Lecture 17 notes for tips from Beth Simon…
- Solutions posted.
- Protocol for re-mark requests:
  - read solutions first, carefully
  - no re-mark requests accepted until Thursday
  - re-mark requests must be in writing (paper attached to exam) and submitted to instructor
  - entire exam re-marked

Midterm #1

- Raw Score Stats:
  - Average: 56, Low: 1, High: 97 (Out of 100)
- I’ve made ArrayList question extra credit (3pts).
- Scaling Formula: \((\text{raw}/97)^{0.7} * 100\)

<table>
<thead>
<tr>
<th>Raw Score</th>
<th>Scaled Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>27</td>
</tr>
<tr>
<td>30</td>
<td>44</td>
</tr>
<tr>
<td>45</td>
<td>58</td>
</tr>
<tr>
<td>60</td>
<td>71</td>
</tr>
<tr>
<td>75</td>
<td>84</td>
</tr>
<tr>
<td>90</td>
<td>95</td>
</tr>
</tbody>
</table>