Lab: Polymorphism and Interfaces  
CSC 207, “Algorithms and Object-Oriented Design”  
Department of Computer Science  
Grinnell College  
February 17, 2021

Responders

In today’s lab, we’ll look at a variety of objects that implement the same interface. This interface — call it Responder — consists of a single abstract method, respond, which takes one argument, a String, and returns a String. The idea is that a Responder can, in effect, hold up one end of a conversation, speaking whenever it is spoken to; the String that the respond method returns is the object’s reply to the String that that method receives as argument.

1. (For Eclipse users:) In Eclipse, start a new project called conversation, and in that project create the Responder interface.

2. Write an opening comment to introduce the Responder interface, then write the Java code to define that interface. Save the definition. (GNU Emacs users: Run javac to compile it as well.)

Echoes

An Echo is an object that implements the Responder interface with a respond method that always returns the same string that it receives.

3. Document and write a definition for the Echo class.

4. Start a test program, ResponderTester, to test the various kinds of Responders that you will define in this lab. Write tests for the Echo class and then compile and run ResponderTester and confirm that your implementation of the Echo class passes the test.

Bores

A Bore is an object that implements the Responder interface with a respond method that ignores the string it receives and always replies with the same fixed string. The Bore class has a one-argument constructor through which the programmer specifies its constant response.

5. (1 point) Document, write, and test a definition for the Bore class.

Disemvowelers

A Disemvoweler is an object that implements the Responder interface with a respond method that removes all the vowels from the string it receives and returns the result. (Comment moderators in social networks sometimes apply the disemvoweling operation to comments contributed by trolls, so to express a degree of disapprobation without complete censorship.)

6. Document, write, and test a definition for the Disemvoweler class. You To implement the respond method, you may use replaceAll method of String class. This method takes two String parameters. It replaces all occurrences of the first parameter with the second parameter.

Recapitulators

A Recapitulator is an object that implements the Responder interface with a respond method that replies with the null string the first time it is invoked, and replies to any subsequent invocation with the concatenation of all the strings that it has previously received as arguments, each terminated with a newline character.

Suppose obj is an object reference for Recapitulator. Here are the expected outputs after each call to respond method:

System.out.println(obj.respond("hello"));null
System.out.println(obj.respond("world"));hello
System.out.println(obj.respond("!"));helloworld

7. Document, write, and test a definition for the **Recapitulator** class.

**Numberers**

A **Numberer** is an object that implements the **Responder** interface by tweaking the operation of another **Responder** object, **numeraland**. When a **Numberer** receives the **respond** message, it takes the argument and sends **numeraland** a **respond** message with the same argument. **Numberer** then receives **numeraland**'s reply, prepends a serial number to it ("1:" on the first invocation, "2:" on the second, and so on), and returns the result.

Here’s an example of the kind of exchange that takes place when **numeraland** is a **Disemvoweler**:

Hello!
1:Hi!
Is anyone there?
2:s nyn thr?

8. (2 points) Document, write, and test a definition for the **Numberer** class. Note that the constructor needs another **Responder** as its argument. Use the **format** method to display the reply from **Numberer**.

**Composers**

A **Composer** is an object that implements the **Responder** interface by combining the operations of two other **Responder** objects — let’s call them **fore** and **aft**. When a **Composer** receives the **respond** message, it takes the argument and sends **fore** a **respond** message with that argument. When **fore** replies, the **Composer** sends **aft** a **respond** message with that reply as its argument. The **Composer** returns **aft**’s reply as its own response to the **respond** message it received.

If, for instance, **fore** is a **Bore** that replies to any conversational opening with the response "Make America GREAT again!", and **aft** is a **Disemvoweler**, then the **Composer** would respond to the string "Hello -- pleased to meet you." with the string "Mk mrc GRT gn!".

The **Composer** class needs a two-argument constructor through which the programmer can specify **fore** and **aft**.

9. Document, write, and test a definition for the **Composer** class.