CSCI 2120
Homework 2: Observer Pattern

Introduction

For this assignment you will "reverse engineer" the class Observable based on our discussion of how it works and the examples I've provided in lecture. First, write a new interface, called MyObserver, then implement a new class MyObservable that can interact with MyObserver objects.

For this assignment, you will be required to write Javadoc-style documentation for all of your methods, including the test methods.

Procedure

1) Write a JUnit tester that redirects standard output (see JUnit slide deck at the end) so that the Pokemon and PokemonTrainer classes can be tested. You're writing this tester to work with the standard Observer / Observable types that Java provides, so that we can basically use the same tests to work with MyObserver / MyObservable. (Remember : some of the examples used in class only produced output to the screen so you could easily see what was taking place. While this is great for interactive testing, you'll need to send this output "somewhere else" to compare it in a non-interactive JUnit tester. This is know as "redirecting" standard output)

Here's an example of such a JUnit tester:

```
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import static org.junit.Assert.*;
import org.junit.*;

public class RedirectTester {

	private ByteArrayOutputStream output = new ByteArrayOutputStream();
	private ByteArrayOutputStream errorOutput =
                          new ByteArrayOutputStream();

	private String lineSeparator = System.getProperty("line.separator");

	@Before
	public void setup() {
		System.setOut(new PrintStream(output));
		System.setErr(new PrintStream(errorOutput));
	} // end @Before method

	@Test
	public void testStdOut() {
		System.out.println("hello");
		assertEquals("hello" + lineSeparator, output.toString());
	}
```

```
	@Test
	public void testStdErr() {
		System.err.println("good bye");
		assertEquals("good bye" + lineSeparator, errorOutput.toString());
	}


	@After
	public void resetStreams() {
		System.setOut(System.out);
		System.setErr(System.err);
	} // end @After method

} // end class RedirectTester
```

2) In a subdirectory, modify the classes Pokemon and PokemonTrainer so that they are MyObservable and MyObserver subtypes (rather than inheriting from Observable and Observer.

3) Write the MyObserver interface so that it conforms to the following specification:

https://docs.oracle.com/javase/8/docs/api/java/util/Observer.html

4) Write the MyObservable class so that it (mostly) conforms to the following specification:

https://docs.oracle.com/javase/8/docs/api/java/util/Observable.html

A) You may use an array as holder for the MyObserver references that have registered themselves with each object.

B) You'll want to keep track of how may MyObservers have registered

C) You'll want a Boolean instance variable to save the state "hasChanged"

D) Implement every method except for the ones that start with "delete"

5) Copy your tester that you wrote for the Pokemon / PokemonTrainer and make sure the tests you wrote all pass! Make sure to test cases where there's one Observer to one Observable, multiple Observers to one Observable, and one Observer having multiple Observables!

BONUS 1: 15 points – implement the two "delete" methods in MyObserver in the array-based implemention.

BONUS 2: 15 points – re-implement using an ArrayList<MyObservable> instead of an array, AND implement the "delete" methods of MyObserver.

HONORS STUDENTS: You must complete one of the bonuses as a part of the standard implementation – implement a second bonus for actual bonus points!

Submission

You will add, commit, and push your program to Gitlab and also turn in a hard copy of your program code. Label your homework folder "HW2" in your repository. If you attempt bonuses please put them in subdirectories of HW2 called “bonus1” or “bonus2”, respectively.

Grading

The MyObserver, MyObservable, Pokemon, and PokemonTrainer implementations will be 70% of your grade. The tester will be worth 30% of your grade.