Due Date: March 11th

Lab 08: Inheritance and Design Patterns

For this lab, we'll work on something familiar to what we've been doing in the last 2, but we'll focus on how we can use the Base class of some Derived classes to iterate over an array of Base-type objects.

Part A: Constructor Derivation (15)

Create the following:

- Base
  - Constructor prints "Base constructor."
  - Has two methods called:
    - m1 that increments the number of times m1 is called, then prints that value.
    - m2(String s) that prints the given string with "from Base"
  - A static integer that counts the number of times m1 is called.
- IType
  - An interface with methods for:
    - m2(String s)
    - m3()
- Derived extends Base implements IType
  - Constructor prints "Derived Constructor"
  - Methods:
    - m1() that prints "Derived.m1"
    - m3() that prints "Derived.m3"
- Derived2 extends Derived
  - Constructor prints "Derived2 Constructor"
  - Methods:
    - m2(String s) that prints the given string with "from Derived2"
    - m4() that prints "Derived2.m4"
- Separate implements IType
  - Constructor prints "Separate Contructor"
  - Implement methods for the following:
    - Like Derived:
      - m1
      - m3
    - Like Derived2:
      - m2(String s)

Note how the interface implementation in IType is satisfied with the m2 method definition in Base when it is implemented in Derived. Now initialize a Base, Derived, and then Derived2 and observe the initialization sequence in the terminal. You should see Base (or superclass) constructors being called before subclass constructors.

Call all methods on the objects to see what they do and call m1 in the Base class twice. Lastly, create an ArrayList of Base objects and fill it with an assortment of 10 Base, Derived, and Derived2 objects. Iterate over this list and call the m2 method on all objects. Add some calls for the Separate class as well.

Part B: Strategy Pattern (15)

We've already seen a design pattern, the iterator pattern, that allowed us to iterate over any collection of items without knowing how the collection is actually implemented. In this part of the lab, we'll be looking at another design pattern called the Strategy Pattern. It involves applying a collection of methods (with an interface in common) to a list of objects.

Create an interface like this:

```
public interface OpClass {
   abstract Object op(Object arg);
}
```

Next, create three classes Square, Cube, and SquareRoot that implement this interface. The Square should convert the arg to a Number, square it, then return the Number as an object. The Cube will do the same thing, expect it will cube the number. The same applies to the SquareRoot which returns the square root of the number.

Create a method called apply in another public class called Calculator and implement your main method here. The apply method should accept an ArrayList of Numbers and an OpClass to apply to the list. It should return a new ArrayList with the new values in it. Make use of the ArrayList iterator here as well.

Test this out by building a list of integers, and applying the square, cube, then square root objects on the list. Print out the list after each application.

Grading Criteria:

Style/submission guidelines: https://gmierzwinski.github.io/bishops/cs321/style_guidelines.html

Comments, Formatting, & Readability	**5 Marks**
Submission Guidelines	**5 Marks**
Program	**30 Marks** **See (X) above**
Total	**40 Marks**