Lab 10: Inheritance
CSE-021-01, Spr 2018 **Points: 20**

Overview

We will use examples given in the lecture to extend the behavior of basic Counters. This lab is meant to familiarize you with the usefulness and concepts of class inheritance. It means traits and behaviors of the parent/super- classes will be passed down to the child/sub- classes.

Before you get started, read chapters 10.1, 10.2, 10.3 and 10.4. Answer activities 10.1.4, 10.2.1, 10.3.1 and 10.4.2 in a Word or text file. Also, answer the Assessment questions as you encounter them in the next section. The prompts for answering assessment questions are placed immediately following the material to which the listed questions relate.

Getting Started

After following the import instructions in the assignment page, you should have a Java project in Eclipse titled Lab 21_10. This PDF document is included in the project in the **doc** directory. The Java files you will use in this lab are in the **src/oop** directory.

Part 1: Test `Counter.java` using `Runner.java`

`Counter` class represents a counter that can be initialized and reset to `0`, incremented by `1`, and asked for its current value.

```
public class Counter {

   private int myCount = 0;

   public void increment() {
      myCount++;
   }

   public int value() {
      return myCount;
   }

   public void reset() {
      myCount = 0;
   }
}
```

It doesn't have a main method; we'll use the `Runner` class to run and manipulate it.

[Answer assessment questions 1 and 2]

Using the provided `Runner` class, along with calls to `Counter` methods, create a counter whose value is `3` (The code should go inside the `testCounter` method).

Create another counter that uses exactly seven calls to the `reset` and `increment` methods to end up with a value of `3` (The code should go inside `testCounter7Statements` method).

Part 2: Fill-in `ModNCounter.java`

"Mod N" counters count to a specified value ("N"), and then cycle back to zero. For example, when N = 3, the mod N counter will count as follows: 0, 1, 2, 0, 1, 2, 0, ...

The mod N counter has an extra instance variable — that we name `cycleLength` — and is initialized with a 1-paramter constructor whose argument is the intended N value. Thus, the following code

```
ModNCounter c = new ModNCounter(2);
System.out.println(c.value());
c.increment();
System.out.println(c.value());
c.increment();
System.out.println(c.value());
c.increment();
```

should print 0, then 1, and then 0.

Your task is to fill in `ModNCounter` so it has the behavior described here. You should only override the `increment` method and making no other changes to produce the desired behavior.

[Answer assessment questions 3, 4, 5 and 6]

Part 3: Create `ModNCounter2.java`

Create this class so it inherits from `Counter`. Objects of this class will behave exactly the same as counters of type `ModNCounter`, but with a different `cycleLength`. For the `ModNCounter2` class, you should create a instance variable, a 1-parameter constructor, and only override the `value` method, and make no other changes to produce the desired behavior. For the class that is creating and using `ModNCounter2`, it should appear to behave exactly the same as `ModNCounter`, but with a different modulus ("N" value). Be sure to include the line "`package oop`" at the top of this file.

[Answer assessment question 7]

Implement a method called `testModNCounter2` to test `ModNCounter2` in `Runner` (and call the method from `main`) so you know it is working. Be sure this new code is included in what you submit for `Runner.java`.

Part 4: Create `DecrementableCounter.java`

One might wish for a counter that allows decrementing its value as well as incrementing it. Create a class `DecrementableCounter` that inherits from `Counter` and provides a `decrement` method. If the counter's value is `0`, a call to `decrement` should have no effect. Otherwise, it should reduce the counter's value by `1`. Don't change the `Counter` class to implement `decrement`. Once again, be sure to include the line "`package oop`" at the top of this file.

[Answer assessment questions 8 and 9]

Implement a method called `testDecrementableCounter` to test `DecrementableCounter` in `Runner` (and call the method from `main`) so you know it is working. Be sure this new code is included in what you submit for `Runner.java`.

Part 5: Fill-in `SeasonCounter.java`

Complete the `SeasonCounter` class that cycles through the four seasons. It will inherit from `ModNCounter`, overriding the `toString` method to return `"spring"`, `"summer"`, `"fall"`, or `"winter"`, depending on whether the current value is `0`, `1`, `2`, or `3`.

[Answer assessment question 10]

Implement a method called `testSeasonCounter` to test `SeasonCounter` in `Runner` (and call the method from `main`) so you know it is working. Be sure this new code is included in what you submit for `Runner.java`.

Part 6: (Assessment) Logic Check and Level of Understanding

1) The Counter class doesn't have a constructor. Why doesn't it need one?
2) The Counter class doesn't have a `main`. Why doesn't it need one?
3) How do we know `ModNCounter` inherits from `Counter` (what is the keyword)?
4) Which method is the constructor inside `ModNCounter` class?
5) Is `cycleLength` variable visible to the parent `Counter` class?
6) What happens when we call `value` and `reset` methods for `ModNCounter` since it is not defined in `ModNCounter`?
7) What happens when we call `increment` and `reset` methods for `ModNCounter2` since it is not defined in `ModNCounter2`?
8) Does `decrement` exist inside `Counter`?
9) Does `increment`'s behavior change for `DecrementableCounter`?
10) Where is `toString` being inherited from?

What to hand in

When you are done with this lab assignment, submit all your work through CatCourses.

Before you submit, make sure you have done the following:

- Verified your solution with your TA or instructor. Note that you may also demonstrate your work after submission, however, any changes following the demo must be resubmitted ***before*** the deadline to receive credit.
- Included answers to activities 10.1.4, 10.2.1, 10.3.1 and 10.4.2, and your answers to Assessment questions (1 – 10) in a Word document or text file named `Part6`
- Attached the `Runner.java` (containing all the test methods and calls), `ModNCounter.java`, `ModNCounter2.java`, `DecrementableCounter.java`, `SeasonCounter.java` and `Part6` files
- Filled in your collaborator's name in the "Comments…" text-box (if any) at the submission page.