CSC171 — Homework 7

Objects: Inheritance, Abstraction, Polymorphism

The goal of this assignment is to give you experience using inheritance, abstraction, and polymorphism in object-oriented programming.

Be sure that your test cases *clearly and completely* demonstrate how your code addresses each part of the assignment. The graders should not have to dig very deeply to figure out what you are doing.

Questions

1. Define the following class hierarchy for shapes:

   (a) Define an abstract class `Shape`. `Shape`s have a color (which can be a string for now) and a boolean indicating whether or not they are filled. These should be properly encapsulated.

   (b) Define an abstract class `Shape2D` that extends `Shape`. This class should define the abstract method `getArea()` that returns the area of a a `Shape2D`.

   (c) Define a class `Rectangle` that extends `Shape2D`. A rectangle has a height and a width. Implement the `getArea()` method appropriately. Use the `@Override` annotation.

   (d) Define a class `Square` that extends `Rectangle`. The class should have an appropriate constructor and also override setter methods as approrpiate to preserve "squareness."

   (e) Define a class `Ellipse` that extends `Shape2D`. An ellipse has a semi-major axis a and a semi-minor axis b (look it up if needed). The area of an ellipse is $A = \pi ab$.

   (f) Define a class `Circle` that extends `Ellipse`. You need appropriate contructor(s) and setter(s) for the class.

   Illustrate all of these with a `main` method in a separate test class.

2. Starting with your class `Person` from Homework 05, define a class `Student` that extends `Person`. Then do the following:

   (a) In addition to the properties of a `Person`, a `Student` has the following properties: a student ID number, a `School` (define a simple class for this), and a major (e.g., "Computer Science"; use a string).

   (b) You will need a constructor for the `Student` class. In Java, your constructor must call the parent class constructor using `super` with appropriate arguments. A student always has an id number and school, so these should be arguments to your constructor, as well as whatever is required for a `Person`.

   (c) Add a method `greeting` to your `Person` class that returns a short greeting that a person might use (e.g., "Hello").

   (d) Override this method in your `Student` class so that students whose major is Computer Science say "Greetings Earthling!" and other students say "Hey."

   Illustrate all of these with a `main` method in either the `Student` class or a separate test class.

3. Define a class `License`. A `License` has a license number and an expiration date (I suggest using `java.time.LocalDate`, but there are other possibilities). Then do the following:

   (a) Think what it might mean for two `License` instances to represent the same real-world license. Write the Boolean `equals` method that takes another `License` as argument and returns `true` if the two `Licenses` are "equal" in this sense. The block comment for your method should explain how you are doing the comparison.

   (b) Write a Boolean method `expired` that returns `true` if the current date is after the expiration date of the `License`, otherwise `false`.

   (c) Define the class `DriversLicense` as a subclass of `License`. In addition to the properties of a `License`, a `DriversLicense` also has the state that issued the license (you can use a string for this).

   (d) Suppose that driver's licenses from different states may have the same license number. Override the `equals` method for `DriversLicense` to do the right thing when testing if two `DriversLicense`s are equal.

   (e) Define the class `TruckDriversLicense` that extends `DriversLicense`. Override the `toString` method to return a meaningful description of an instance.

   (f) Define the class `FishingLicense` which extends `License`. In addition to the properties of a `License`, a `FishingLicense` specifies what types of may be caught (you can use a string for this). Override the `toString` method.

   Illustrate all of these with a `main` method in a separate test class.

Grading Scheme

Equal weight for each part.

Doesn't compile or is trivial	$< 50\%$
Compiles and is non-trivial	$\geq 50\%$
Complete and correct with good style and comments	100%
Incomplete, incorrect, bad style, no comments	$< 100\%$

Submission Requirements

Your submission **MUST** include a file named "`README.txt`" with your name, your NetID, the assignment number, and your lab section. This file should explain anything we need to know about how to build and run your project. In particular, be sure to explain how to run what parts of your submission for each question in the assignment.

Submit your solution as a single ZIP archive to BlackBoard before the deadline.

Late homeworks will not be graded and will receive a grade of 0.

All assignments and activities associated with this course must be performed in accordance with the University of Rochester's Academic Honesty Policy.