CMPS 1600 Introduction to Computer Science II – Spring 2019

Due 11:59pm on Monday February 11

Lab 2

Objectives

- To practice defining and using abstract classes and interfaces.
- To create UML class diagrams.
- To practice generating Javadoc.

Code quality

The code you submit should be easily readable, clean and well-commented. Commented-out sections of non-working code should be removed. Indentations and white spaces should be used to separate logical blocks of the code. Variables, methods and classes should have descriptive names.
Every class and method should have a comment according to Javadoc format. When overriding methods, use `@Override` compiler annotation.

Abstract Shape

Abstract Shape

1. Create an abstract `Shape` class with three abstract methods: `area()`, `perimeter()`, and `getShape()`. `getShape()` will return a `String` that represents a particular shape. (For instance, `getShape()` for a Circle should return `"circle"`). Add a static attribute `shapeCount` that is incremented every time a new shape is created (where you add a method for incrementing `shapeCount` is up to you, but the attribute should be present in `Shape`).

2. Now create three subclasses of `Shape`: `Circle`, `Triangle`, and `Rectangle`. Also create a `Square` subclass of `Rectangle`. The `Rectangle` and `Square` classes from previous assignments can be used with some minor edits. Note: to calculate the area of Triangle using three side lengths, use Heron's formula: check out *http://www.mathopenref.com/heronsformula.html* and the animation there to get the formula and check your results.

3. Make a separate class, `ShapeTester` to test the functionality of the above classes. Make a `Shape` ArrayList that contains at least one of each individual shape. Notice that we cannot construct a `Shape` object since it is an abstract class, but we can create an ArrayList of shapes. Loop through the ArrayList and print out each shape's type using `getShape()`. Then print only circles and their positions in the array. Then output how many shapes were created using `shapeCount` field.

Interfaces

1. Create an interface `Displayable` that contains one void method `display()`.
2. Modify the `Shape` class so that it implements `Displayable`. The `display()` method for `Shape` should print out its type, area, and perimeter, in some format of your choice.
3. In the `ShapeTester` class, make a method `displayArray` that takes as its argument a ArrayList of Shapes. Your method should loop through the array and display each element. Call this method on the array you created above.

Comparable

1. Java has a built-in interface called `Comparable<T>`. It only has one method, `compareTo()`. The `T` in `Comparable<T>` is a generic type. When you implement the interface, you replace `T` with the type that you want to compare to. `compareTo()` returns an integer. For two objects x and y, `x.compareTo(y)` should be negative if $x < y$. It should be positive if $x > y$ and zero if $x = y$.
2. Modify `Shape` so that it implements `Comparable<Shape>`. Notice that we can implement multiple interfaces, but we can only extend one class. For the purpose of this assignment, we will consider one shape to be bigger than another if its area is bigger.
3. In the `main` method of `ShapeTester`, sort the array you have created using the `Collections.sort()` method. Since `Shape` implements `Comparable<Shape>`, the `Collections.sort()` method will know how it should compare two shapes. It will use this to sort the shapes by their area. Display the elements of this array again to make sure the sorting worked properly.

Javadoc

In this and the following labs, we will continue using Javadoc format for comments.

Javadoc Open your project for the above lab in Eclipse, and select Project − > Generate Javadoc. Follow the prompt to initiate generation of the Javadoc documentation. Open `index.html` and examine the documentation that was generated entirely from your code and comments in your code. If there are style or grammar issues, fix them in the comments in this week's lab. *Google and resolve the problems that you may encounter while generating Javadoc.*

UML class diagram

Pay careful attention to specifics of UML notation (public vs. private methods, italic vs. regular font, dashed vs. regular arrows, arrowheads, etc).

Token dispenser

For this assignment think about the attributes and behaviors for a token dispensing machine that you might find in a carwash or in an arcade. Your objective is to create a class that models how a token vending machine works. To simplify the problem, this token dispensing machine only takes quarters, no other coin denominations or bills are allowed. When you put a quarter in the machine, you always get one token in return. Keep in mind that a well designed class will contain some data members (attributes), and some methods (behaviors) that operate on that data. Now, design a class that represents this token dispensing machine.
Create a class diagram using standard UML notation as explained in zybook or any online UML tutorial. You may hand draw your diagram and take a picture of it or scan it, or you can use any drawing software. A big plus would be using special UML software (such as UMLet). Even though the assignment asks you to design one class for the token dispensing machine, your diagram may use several classes if necessary. Convert your class diagram into a PDF file for submission.

Bank

Draw a UML diagram for Bank problem from the previous lab. Illustrate all classes.

Shape hierarchy

Draw a UML diagram for Shape problem from this lab. Illustrate all classes and interfaces that you wrote.

Submission

Upload the final version of Shape.java, Circle.java, Triangle.java, Rectangle.java, Square.java, ShapeTester.java, Displayable.java, UML.pdf files on Canvas and double-check your submission.