Assignment 7 C212 FA22

Last modified 2022-11-03 10:00 AM

Learning Objectives

After completing this assignment, students will be proficient in adapting provided system specifications in the form of UML Class Diagrams into software. Given a diagram in one of many potential formats, they will be able to design and implement classes to programmatically implement relevant details. They will also be able to write tests to demonstrate the quality and behavior of their work.

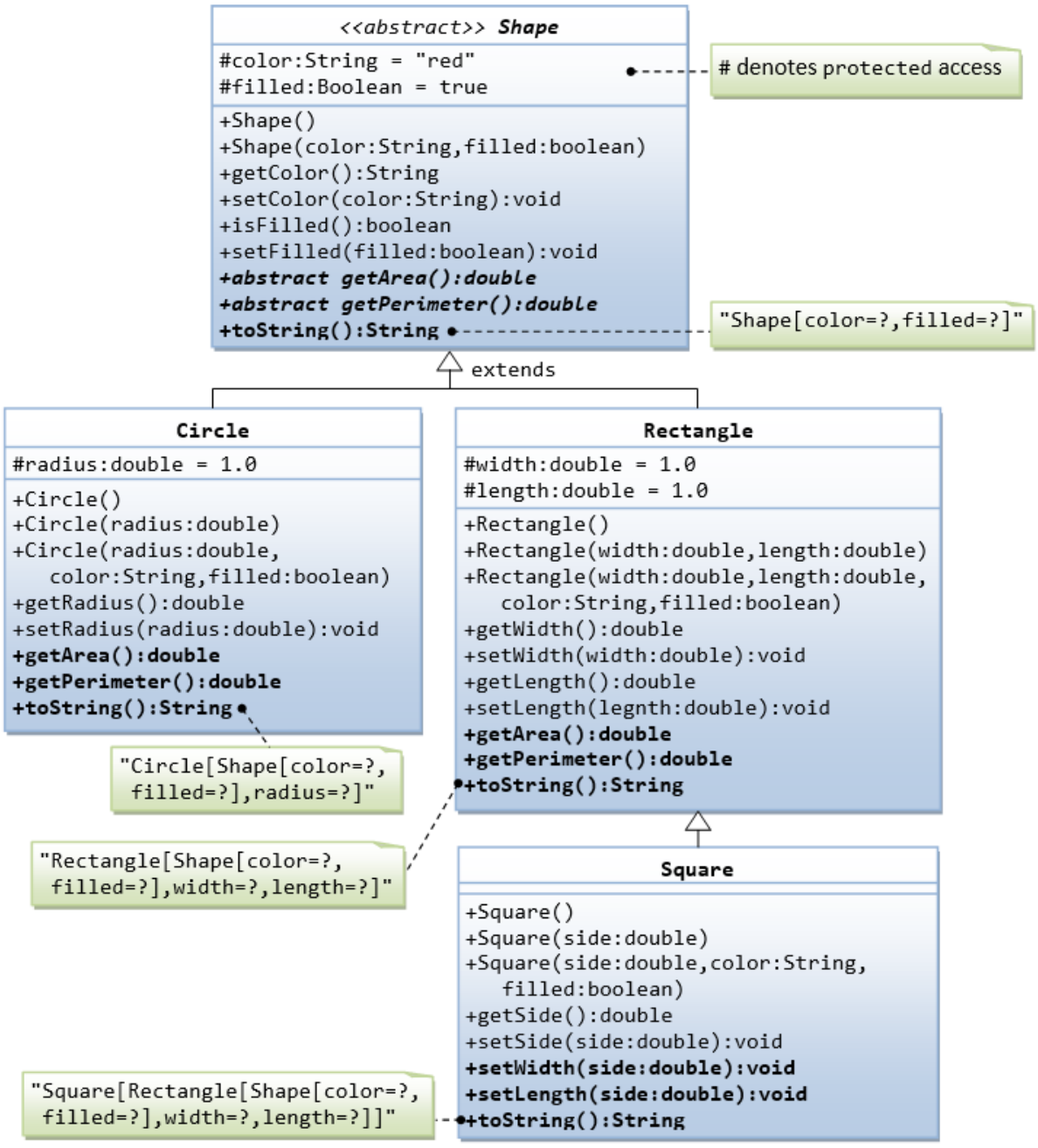
Summary

The gist of this assignment document's contents are as follows:

- Program Description - Adapting a system to code using a UML Class Diagram
    - Shape
    - Circle
    - Rectangle
    - Square
- Rubric
    - Also visible on Canvas assignment page
- Changelog
    - No changes yet

Program Description

Your assignment this week is pretty light on description: you'll just need to implement the system represented by this UML Class Diagram.



In particular, notice that Shape is an abstract class, and it has two abstract methods getArea() and getPerimeter(). As you know, that means its concrete subclasses are required to give implementations to these methods. Also, give all instance variables the protected access modifier, is represented by the # in this diagram. You should also

make sure you mark all methods that override a superclass's method with the @Override annotation.

The instance variables being assigned values in the diagram means that those are the default values if the class is instantiated with the no-args constructor.

Prevent any Shape from having negative dimensions. If ever they are assigned one, throw an IllegalArgumentException.

Notice that Square appears to keep track of a side length but doesn't have a side length instance variable. You should NOT add one! Instead, somehow use a superclass's instance variable and repurpose it to be the Square's side length. This is an example of abstraction. In general, do not add more members to the classes than what is shown in the diagram. You want to make this system as it appears.

Remember to include getters and setters for each instance variable, as well as toString methods. The green boxes describe what each class's toString method should return.

Once you're finished, write JUnit tests to test the behavior of the system and all its classes. If you test something on a parent class, you don't necessarily need to do rigorous testing on the child classes' members inherited from it, unless the child classes override those members. Your tests should encompass all the typical use cases and the edge cases. Furthermore, you should include tests demonstrating the idea of polymorphism – for example, the idea that a Square is a Shape. For this, use upcasting and downcasting. A line of code like `Shape myShape = new Rectangle(2.0, 4.0, "green", false);` would upcast a Rectangle to a Shape, and one like `Rectangle myRectangle = (Rectangle) myShape;` would downcast that same instance.

Rubric

A rubric is also available on the assignment page on Canvas, but here is an outline of the point breakdown for different categories.

- (10pts) Code cleanliness, conventions, and comments
  - (10/10pts) for >95% correctness and all required comments
  - (7.5/10pts) for repeated minor mistakes or one egregious mistake, or for missing either overall or inline comments
  - (5/10pts) for code that is sloppy and hard to understand or is missing all comments
  - (<=2.5/10pts) for code that has no comments, is hard to understand, and consistently does not follow Java conventions
- (20pts) Shape
- (15pts) Circle
- (15pts) Rectangle
- (20pts) Square
- (20pts) JUnit Testing

Here's a rough guideline for grading with respect to testing:

- (0%) for no testing whatsoever
- (33%) for writing some but not much code that resembles testing, but doesn't seem clear what's being tested or why
- (67%) for plenty of testing but the testing doesn't make sense or is missing corresponding expected outputs, OR tests which are reasonable and have corresponding outputs but aren't sufficient for demonstrating coverage
- (100%) for plenty of well-thought out testing that demonstrates coverage of the testing and provides corresponding expected results

Changelog

If any corrections, clarifications, etc. are made to the assignment after its release, they will be listed here. Any changes more substantial than simply fixing typos will get corresponding Canvas announcements. That way, you don't need to keep checking whether updates have occurred.

- No changes yet…