sOFTWARE 2 PRACTICAL

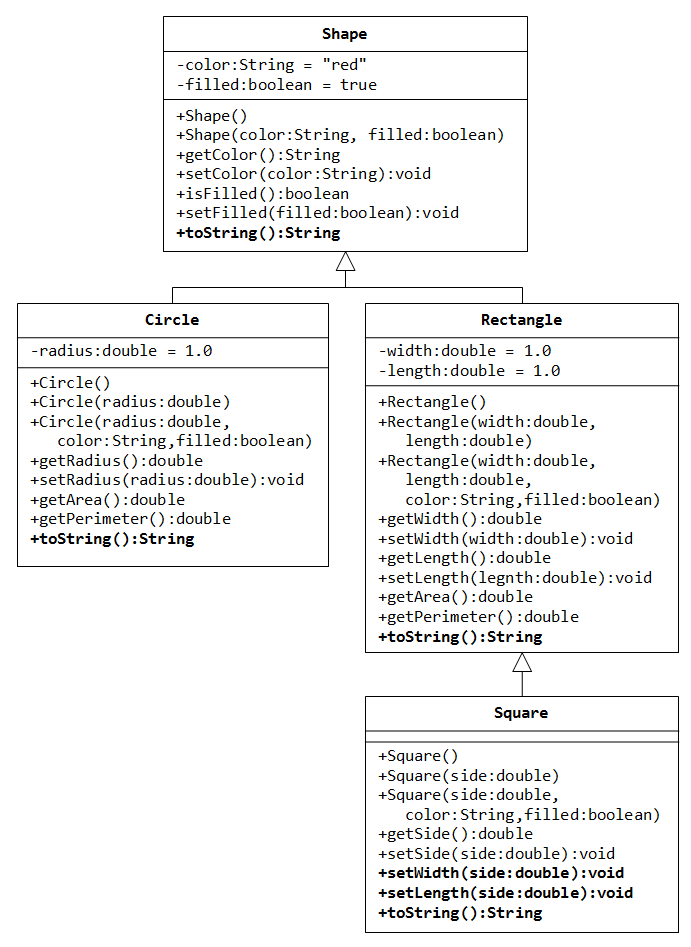
## Simple Inheritance

Week 4 – Practical 4

The aim of this week practical is to understand the basic concept of inheritance in the OOP paradigm. Later in the semester, we will be looking at more advanced concept of inheritance based on today’s practical. So, it is important that you finish this practical. All the classes described here should be in the package geometry.

### Problem:

We want to build a program that draws shapes on a canvas, and in addition provides additional information about the shapes being drawn. We will start with few simple shapes like circles, rectangles and squares. We provide the following UML diagram for the class hierarchy.



Means Square is a subclass of Rectangle.

Methods:  
name (parameters):  
 returned type

Attribute:  
name:type:value  
+ means public  
- means private

Attributes

Class name

Methods

Write a superclass called Shape (as shown in the class diagram), which contains:

1. Two instance variables color (String) and filled (boolean).
2. Two constructors: a no-arg (no-argument) constructor that initializes the color to "red" and filled to true, and a constructor that initializes the color and filled to the given values.
3. Getter and setter for all the instance variables. By convention, the getter for a boolean variable xxx is called isXXX() (instead of getXxx() for all the other types).
4. A toString() method that returns "A Shape with color of xxx and filled/Not filled".

Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.

The Circle class contains:

1. An instance variable radius (type is double).
2. Three constructors as shown. The no-arg constructor initializes the radius to 1.0.
3. Getter and setter for the instance variable radius.
4. Methods getArea() and getPerimeter().
5. Override the toString() method inherited, to return "A Circle with radius=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

The Rectangle class contains:

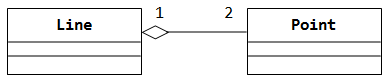
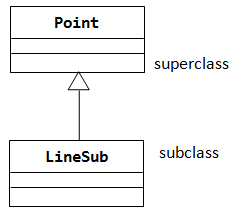
1. Two instance variables width and length (both double).
2. Three constructors as shown. The no-arg constructor initializes the width and length to 1.0.
3. Getter and setter for all the instance variables.
4. Methods getArea() and getPerimeter().
5. Override the toString() method inherited, to return "A Rectangle with width=xxx and length=zzz, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modelled as a subclass of Rectangle. Square has no instance variable but inherits the instance variables width and length from its superclass Rectangle.

1. Provide the appropriate constructors (as shown in the class diagram).
2. Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.
3. Do you need to override the getArea() and getPerimeter()? Try them out.
4. Override the setLength() and setWidth() to change both the width and length, so as to maintain the square geometry.

### Extension 1:

We want to add two additional classes Point and Line, and we need to make a design decision. Should we use ***composition*** or ***inheritance***?

* Composition (has-a relationship):   
   
* Inheritance (is-a relationship):   
   

Choose which design is better suited for our problem and implement the two classes given the UML below:

