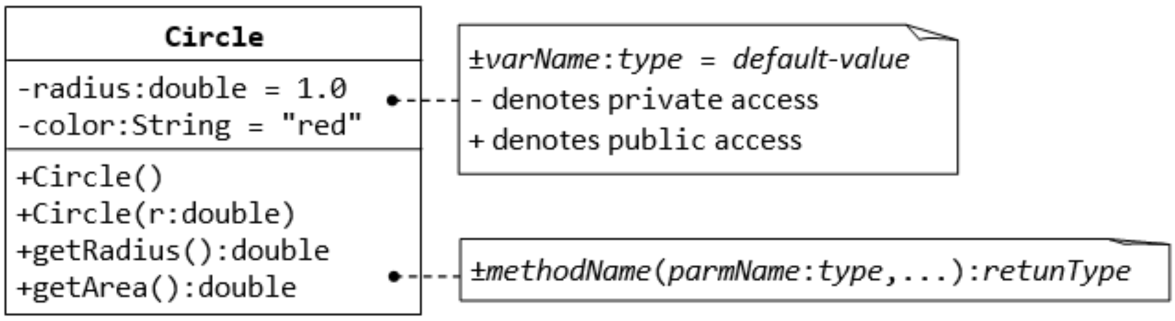
Exercise 06

a) Design a class called **circle** (as shown in the following class diagram). It contains:

- Two private instance variables: radius (of the type double) and color (of the type String), with default value of 1.0 and "red", respectively.
- Two *overloaded* constructors - a *default* constructor with no argument, and a constructor which takes a double argument for radius.
- Two public methods: getRadius() and getArea(), which return the radius and area of this instance, respectively.



b) Write a *test program* called TestCircle (in another source file called TestCircle.java) which uses the Circle class. Create some test circles in the class to test the get radius and get area methods of your circle object.

c) **Constructor:** Modify the class Circle to include a third constructor for constructing a Circle instance with two arguments - a double for radius and a String for color. Modify the test program TestCircle to construct an instance of Circle using this constructor.

d) **Getter:** Add a getter method for variable color for retrieving the color of this instance. Modify the test program to test this method.

e) **public vs. private:** In TestCircle, can you access the instance variable radius directly (e.g., System.out.println(c1.radius))??; or assign a new value to radius (e.g., c1.radius=5.0)?? Try it out and explain the error messages. (Hint: you should not be able to access them directly)

f) **Setter:** Is there a need to change the values of radius and color of a Circle instance after it is constructed? Yes it might be. Therefore, add two public methods called *setters* for changing the radius and color of a Circle instance. Modify the TestCircle to test these methods.

g) **Keyword "this":** Instead of using variable names such as r (for radius) and c (for color) in the methods' arguments, it is better to use variable names radius (for radius) and color (for color) and use the special keyword "this" to resolve the conflict between instance variables and methods' arguments. If you did not already implement it like this, change all your methods and constructors to use the this keyword. Test it again in the test class.

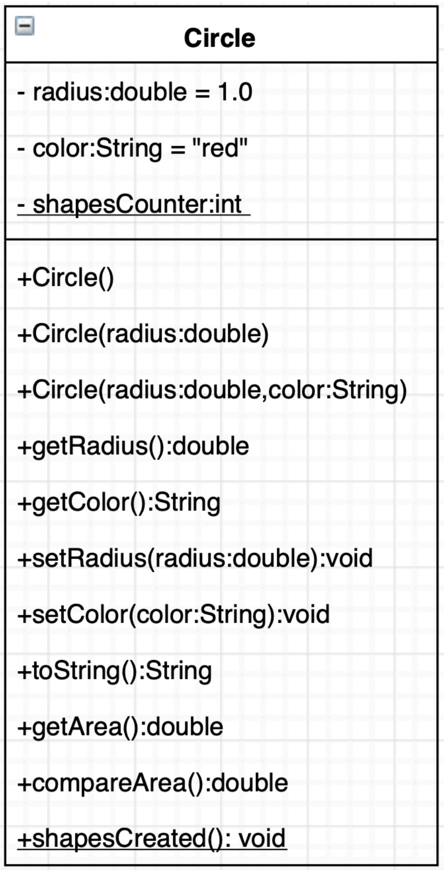
h) Write a method called compareArea() that compares the area of the current circle to another one. Return the difference between the two. Test it in your test program.

i) **Method `toString()`:** Every well-designed Java class should contain a `public` method called `toString()` that returns a short description of the instance (in a return type of `String`). The `toString()` method can be called explicitly (via *`instanceName`*`.toString()`) just like any other method; or implicitly through `println()`. If an instance is passed to the `println(`*`anInstance`*`)` method, the `toString()` method of that instance will be invoked implicitly. Include a `toString()` method to the `Circle` class that returns the following when called: Circle[radius=?? color=??] (?? current object values). Test your toString method with the code below. What happens if you test the code on the circle object without having the toString() methods implemented?

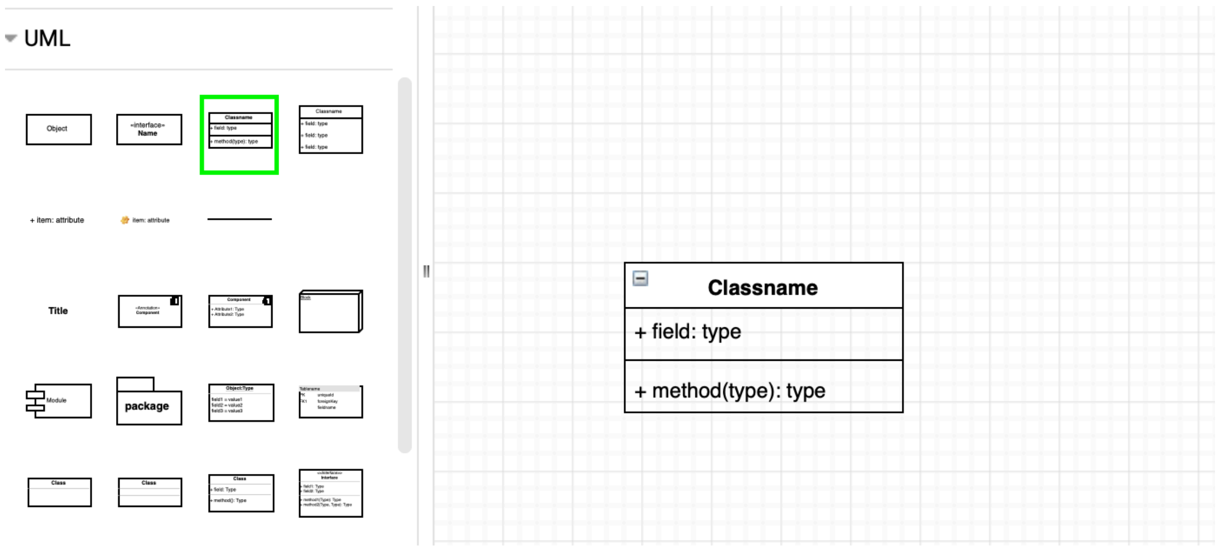
```
Circle c1 = new Circle1(1.2);
System.out.println(c1.toString());  // explicit call
System.out.println(c1);             // println() calls toString() implicitly, same
as above
System.out.println("Operator '+' invokes toString() too: " + c1);  // '+' invokes
toString() too
```

j) **Keyword `"static"`:** Create a static variable shapesCounter that counts the number of circle objects created (add the increment part to all constructors). Add a static method shapesCreated() that returns the number of created circle object by the circle class (you have to call it like this: Circle.shapesCreated()).

The final circle class diagram after solving all tasks should look like this:



k) Implement a similar class for another geometric shape (triangle, rectangle or hexagon). Try to start with drawing the class diagram. You can use draw.io for that. It provides basic UML drawings (click on existing field or method and then ctrl+enter to add a new field or method). Test it in the same test class.


UML
Classname
+ field: type
+ method(type): type