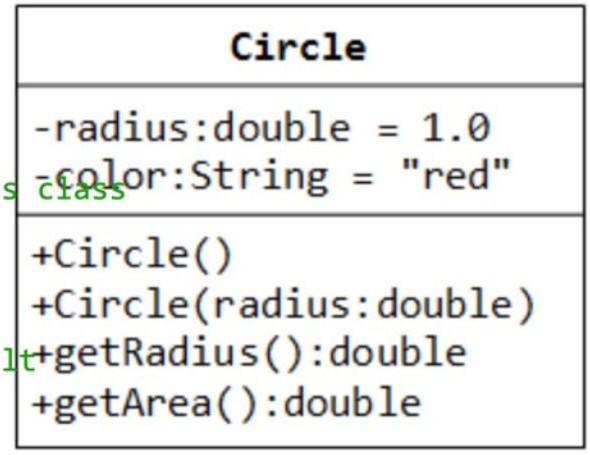
Exercise: The `Circle` Class

A class called `circle` is designed as shown in the following class diagram. It contains:
Two `private` instance variables: `radius` (of type `double`) and `color` (of type `String`), with default value of 1.0
and "red", respectively.
Two *overloaded* constructors;
Two `public` methods: `getRadius()` and `getArea()`.



The source codes for `Circle` is as follows:

```
public class Circle { // save as "Circle.java"
// private instance variable, not accessible from outside this class
private double radius;
private String color;
// 1st constructor, which sets both radius and color to default
public Circle() {
radius = 1.0;
color = "red";
}
// 2nd constructor with given radius, but color default
public Circle(double r) {
radius = r;
color = "red";
}
// A public method for retrieving the radius
public double getRadius() {
return radius;
}
// A public method for computing the area of circle
public double getArea() {
return radius*radius*Math.PI;
}
}
```

Compile "`Circle.java`". Can you run the `Circle` class? Why? This `Circle` class does not have a `main()` method.
Hence, it cannot be run directly. This `Circle` class is a "building block" and is meant to be used in another program.
Let us write a *test program* called `TestCircle` which uses the `Circle` class, as follows:

```
public class TestCircle { // save as "TestCircle.java"
public static void main(String[] args) {
// Declare and allocate an instance of class Circle called c1
// with default radius and color
Circle c1 = new Circle();
// Use the dot operator to invoke methods of instance c1.
System.out.println("The circle has radius of "
+ c1.getRadius() + " and area of " + c1.getArea());
// Declare and allocate an instance of class circle called c2
// with the given radius and default color
Circle c2 = new Circle(2.0);
// Use the dot operator to invoke methods of instance c2.
System.out.println("The circle has radius of "
+ c2.getRadius() + " and area of " + c2.getArea());
}
}
```

Now, run the `TestCircle` and study the results.
TRY:

1. Constructor: Modify the class `Circle` to include a third constructor for constructing a `Circle` instance with the given `radius` and `color`.

```
// Construtor to construct a new instance of Circle with the given radius and color
public Circle (double r, String c) {......}
```

Modify the test program `TestCircle` to construct an instance of `Circle` using this constructor.

2. Getter: Add a getter for variable `color` for retrieving the `color` of a `Circle` instance.

```
// Getter for instance variable color
public String getColor() {......}
```

Modify the test program to test this method.

3. `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.

4. Setter: Is there a need to change the values of `radius` and `color` of a `Circle` instance after it is constructed? If so, add two `public` methods called *setters* for changing the `radius` and `color` of a `Circle` instance as follows:

```
// Setter for instance variable radius
public void setRadius(double r) {
radius = r;
}
// Setter for instance variable color
public void setColor(String c) { ..... }
```

Modify the `TestCircle` to test these methods, e.g.,

```
Circle c3 = new Circle(); // construct an instance of Circle
c3.setRadius(5.0); // change radius
c3.setColor(...); // change color
```

5. 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. For example,

```
// Instance variable
private double radius;
// Setter of radius
public void setRadius(double radius) {
this.radius = radius; // "this.radius" refers to the instance variable
// "radius" refers to the method's argument
}
```

Modify ALL the constructors and setters in the `Circle` class to use the keyword "`this`".

6. 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. For example, include the following `toString()` methods to the `Circle` class:

```
public String toString() {
return "Circle: radius=" + radius + " color=" + color;
}
```

Try calling `toString()` method explicitly, just like any other method:

```
Circle c1 = new Circle(5.0);
System.out.println(c1.toString()); // explicit call
```

`toString()` is called implicitly when an instance is passed to `println()` method, for example,

```
Circle c2 = new Circle(1.2);
```

```
System.out.println(c2.toString()); // explicit call
System.out.println(c2); // println() calls toString() implicitly, same as above
System.out.println("Operator '+' invokes toString() too: " + c2); // '+' invokes toString() too
```