Lab 10 - Introduction to Objects and Class Implementation

Goals:

- To understand the concepts of classes, objects and encapsulation
- To implement instance variables, methods and constructors
- To design a simple test class for a Java class file

In object-oriented programming, an object is a data type that has structure and state. Objects have data and methods that provide operations that may access or manipulate the state of the object or the data within the object. The grouping of data and operations that apply to that data forms a new data type. Objects of the data type are then used in programs to implement solutions. In this lab activity you will be creating a Java class and then writing a program with that uses an object of that class to test your implementation.

A class definition provides the blueprint used to create objects of that data type. The design of a class supports encapsulation by declaring the instance variables **private** and providing **public** getters and setters (aka accessor and mutator methods), as well as **public** constructors. Private instance variables of a class are visible by all the methods of the class, but are not visible or modifiable outside of the class. Please use these guidelines when building your class.

1) Create a UML diagram of the following class specified below.
   - Class name **`Fan`**
   - Three public static constants named SLOW, MEDIUM, and FAST with the values 1, 2, and 3 to denote the fan speed.
   - A private int instance variable named speed that specifies the speed of the fan (the default is SLOW)
   - A private boolean instance variable named on that specifies whether the fan is on (the default is false)
   - A private double instance variable named radius that specifies the radius of the fan (the default is 5)
   - A private String instance variable named color that specifies the color of the fan (the default is blue)
   - A getter and setter method for all four instance variables
   - A parameterized constructor that creates a fan with a value specified for speed, on, radius and color. Use the *`this.`* construct to assign values to the instance variables.
   - A no-arg constructor that creates a default fan using the *`this()`* constructor call to the parameterized constructor
   - A method named `toString()` that returns a string description for the fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string, such as "fan is 1, blue, and size 5". If the fan is not on, the method returns the string "fan is off".

2) Once you have created your UML diagram, navigate to the **Labs** package in Intelji and create another package named **Lab10**. Create a Java class that is public, without a main method, called **`Fan`** to model the fan specified above.

3) Add Javadoc comments to your code to fully document what you have created. Below is an example:

```
/**
 *This method returns the status of whether the fan on or off
 */
public int isOn(){   }
```

4) Test your class by creating another class called **`FanTest`** within the **Lab10** package. This class will contain a main method. Within the main method create two objects of the class **`Fan`**. The first object will have the maximum speed, radius of 10, a color of yellow, and the fan will be on. The second fan will have medium speed, radius of 20, color of blue and this fan will be off. Display both objects by invoking their toString() method.

5) Run **`FanTest.java`** and take a screenshot of the results.

6) Test your code by running the JUnit test: **`Lab10Test`**. Make sure all tests pass.

7) Take a screenshot of your `Lab10Test.java` tests passing.

8) Upload the **`Fan.java`** file to Gradescope and ensure that all test cases pass.

9) Upload the **`Fan.java`** and **`FanTest.java`** files along with the two screenshots to the submission link in Canvas.