# Traffic Light Simulator

## *Overview*

The purpose of the project is to demonstrate good architectural design. The main aspects of such design will be:

1. Enumeration: usage of enumeration to create constant that both hold data and functionality.
2. Multi-threading: controlling multiple threads to function as our signals and keep them in proper synchronicity.

## *Constraints*

- Program must stop and start on the use of the enter key.
- No use of GUI
- A written description and/or comments in your code explaining the rationalisation behind your architectural decisions, and what you would have done with more time, is acceptable.

## *Extra points*

- Use of TDD/BDD.
- Storing values in Local storage so that they persist on reload.
- Demonstrating your knowledge of currently trending tools, best practices and design patterns.

## *Background*

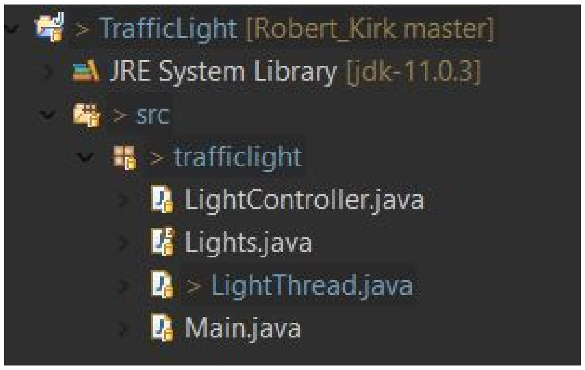
I'm a full stack developer with nearly a one years worth of Java experience about 6 months of which was devoted to the use of Spring Boot.
Considering the amount of time I had for this particular challenge, I tackled it based on what I felt was most important:

1. Neat construction of an enumeration with the proper structure that hold meaningful functionality
2. Creation of a thread that properly times the threads to activate on time to meet the standards set by the projects target
    a. Green light will last 5 seconds
    b. Yellow light will last for 2 seconds
    c. Red light will last for 3 seconds
3. Creation of a method for controlling these elements by the press of the enter key

**Traffic Light Simulator**

## ***Screenshots:***

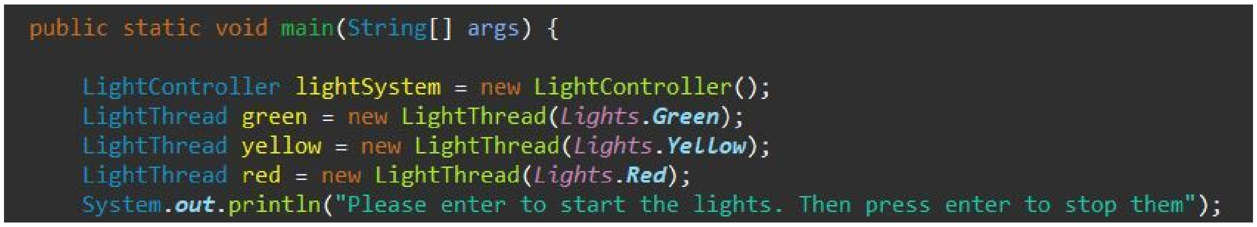
### ***File Structure:***



```
> TrafficLight [Robert_Kirk master]
  JRE System Library [jdk-11.0.3]
  > src
    > trafficlight
      LightController.java
      Lights.java
      > LightThread.java
      Main.java
```

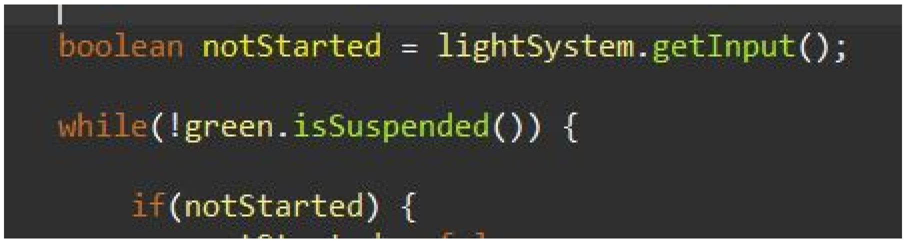
Package trafficlight contains:

- LightController:
  - Holds the functionality of how the enter button controls the functioning of the app.
- Lights:
  - Contains the enumeration and its functionality
- LightThread:
  - Contains the threads necessary to produce the functionality required
- Main:
  - Contains the main method



```java
public static void main(String[] args) {

    LightController lightSystem = new LightController();
    LightThread green = new LightThread(Lights.Green);
    LightThread yellow = new LightThread(Lights.Yellow);
    LightThread red = new LightThread(Lights.Red);
    System.out.println("Please enter to start the lights. Then press enter to stop them");
```

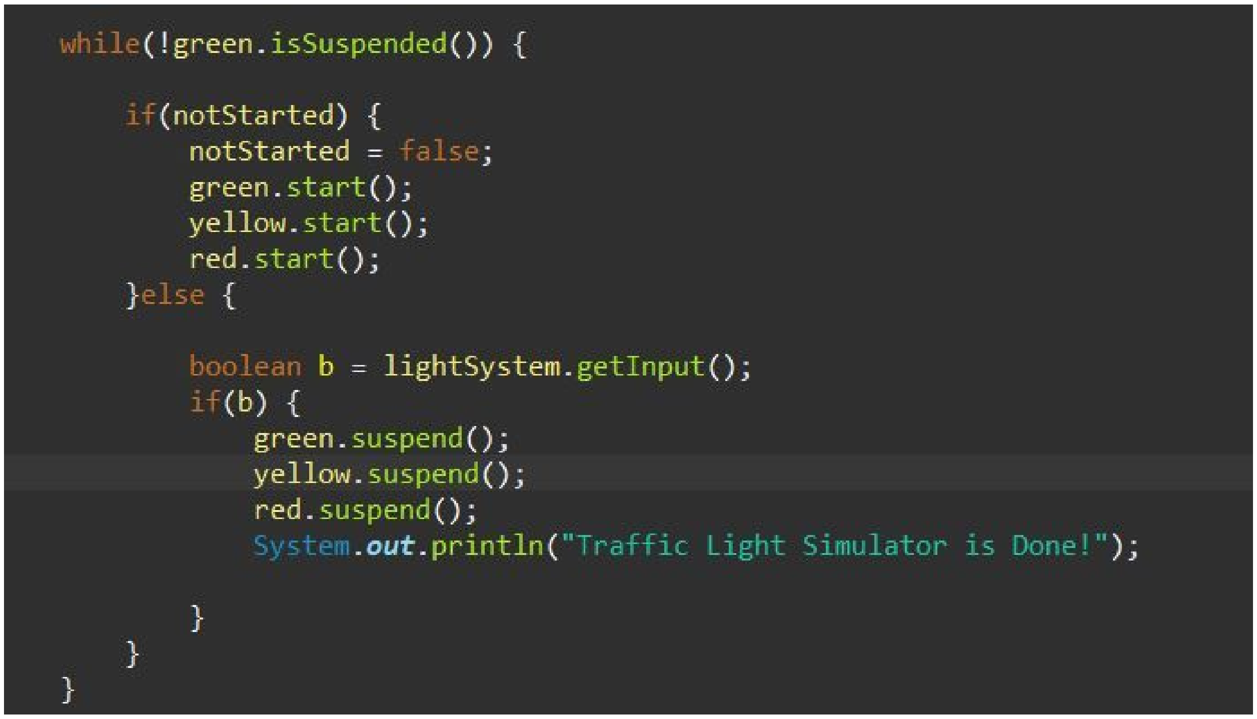
Initial statements to create the threads and control system.



```java
boolean notStarted = lightSystem.getInput();

while(!green.isSuspended()) {

    if(notStarted) {
```

Method get input stops the code and forces the user to press enter to continue.

## Traffic Light Simulator



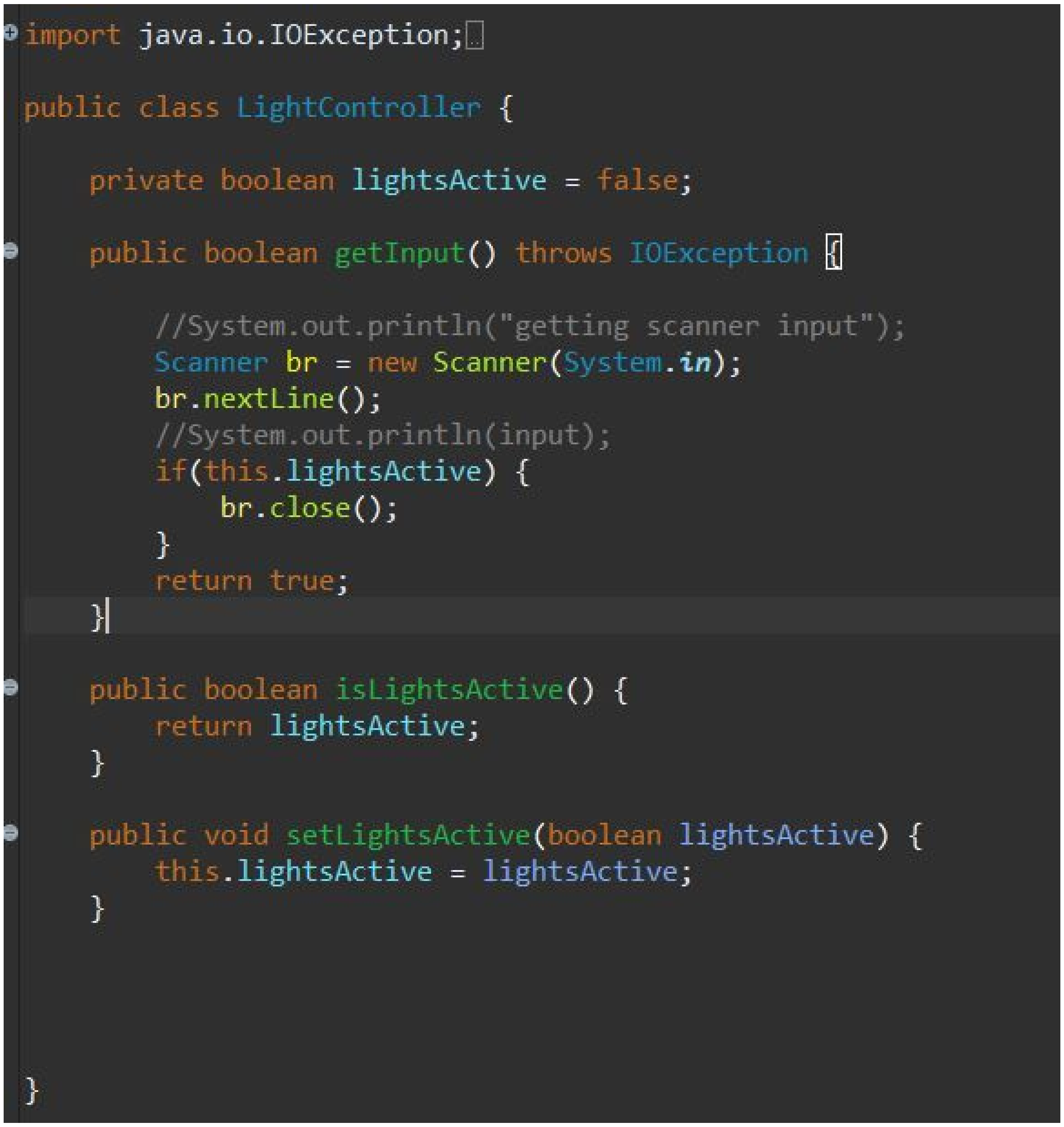
```java
while(!green.isSuspended()) {

    if(notStarted) {
        notStarted = false;
        green.start();
        yellow.start();
        red.start();
    }else {

        boolean b = lightSystem.getInput();
        if(b) {
            green.suspend();
            yellow.suspend();
            red.suspend();
            System.out.println("Traffic Light Simulator is Done!");

        }
    }
}
```

The while loop will only run twice getInput() will force the main thread to stop and wait for the user to press enter.

While statement is conditional on thread green not being suspended. On first pass notStarted is true. This allows for the threads to be started and the flag boolean notStarted to be switched false. Making sure the threads can only be started once.

Once the user presses enter for the second time it will force all of the threads to be suspended and the while loop ends.

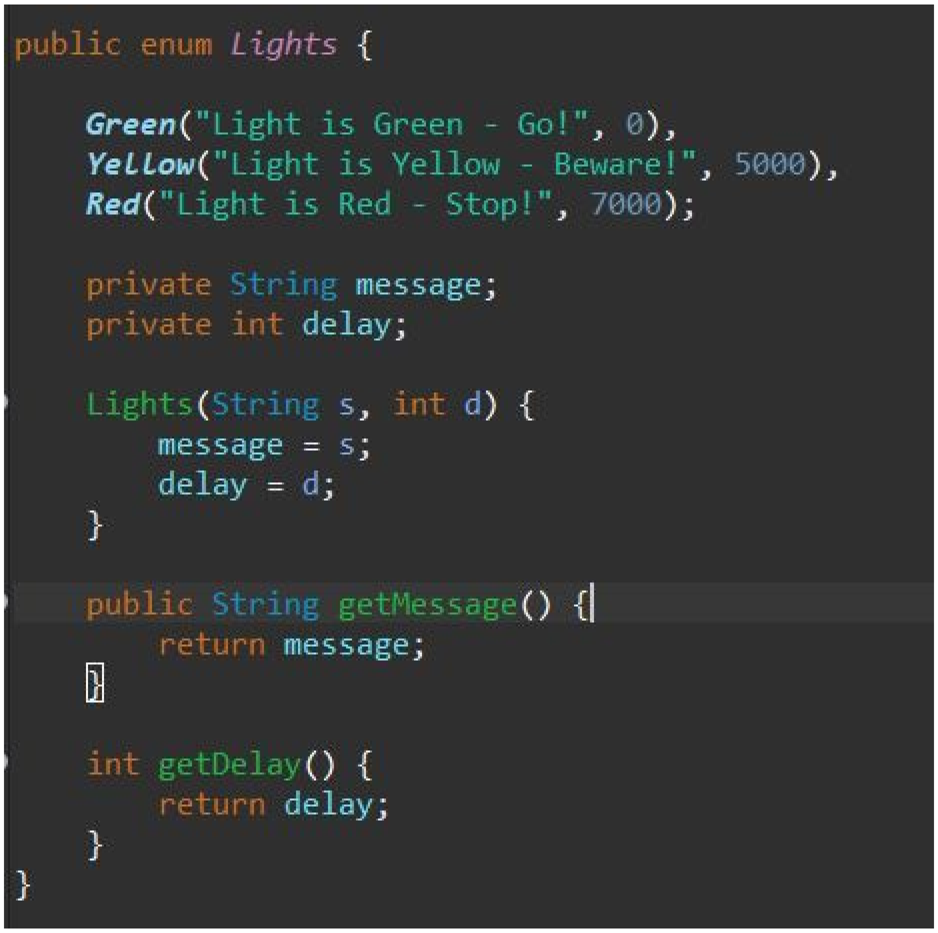
## Traffic Light Simulator



```
import java.io.IOException;

public class LightController {

    private boolean lightsActive = false;

    public boolean getInput() throws IOException {

        //System.out.println("getting scanner input");
        Scanner br = new Scanner(System.in);
        br.nextLine();
        //System.out.println(input);
        if(this.lightsActive) {
            br.close();
        }
        return true;
    }

    public boolean isLightsActive() {
        return lightsActive;
    }

    public void setLightsActive(boolean lightsActive) {
        this.lightsActive = lightsActive;
    }



}
```

The LightController class only contains one variable to ensure for the purposes of closing the scanner when it should be closed, which is the second time the getInput() is called.

## Traffic Light Simulator

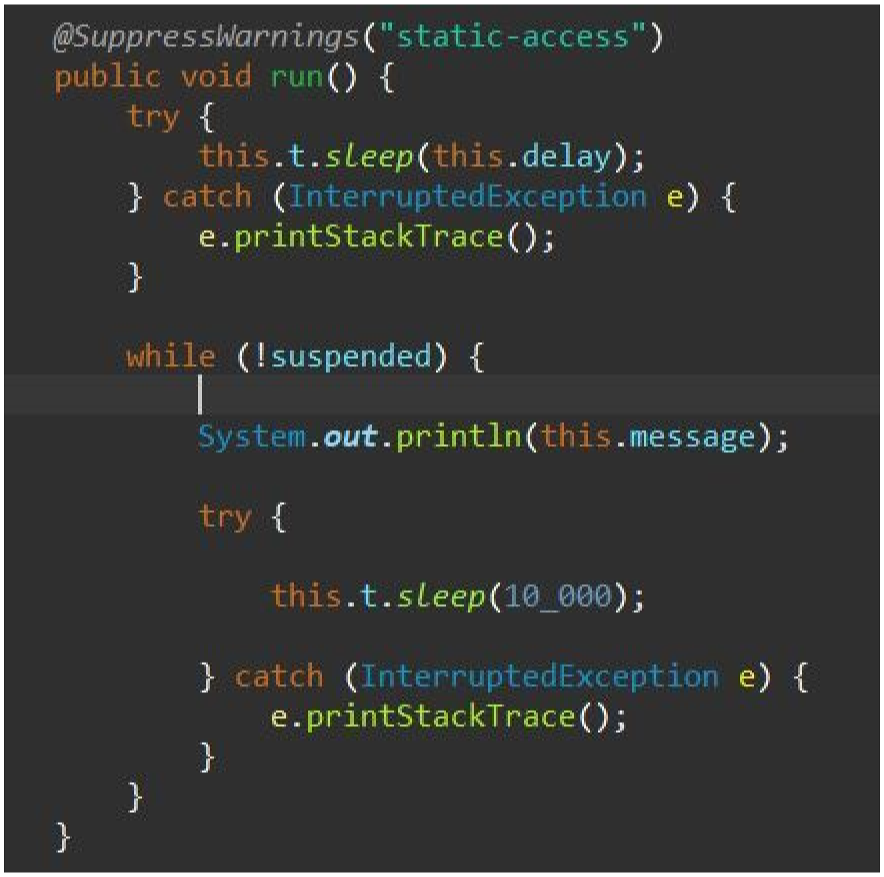


```java
public enum Lights {

    Green("Light is Green - Go!", 0),
    Yellow("Light is Yellow - Beware!", 5000),
    Red("Light is Red - Stop!", 7000);

    private String message;
    private int delay;

    Lights(String s, int d) {
        message = s;
        delay = d;
    }

    public String getMessage() {
        return message;
    }

    int getDelay() {
        return delay;
    }
}
```

The enumeration Lights contains three values Green, Yellow and Red. They each have two associated values

1. The string message that they will be broadcasted to their associated thread when called
2. The delay that will desynchronize the threads so they can fire 10 seconds apart from their initialization.
    a. Green has no delay
    b. Yellow will fire 5 seconds after Green
    c. Red will fire 7 seconds after Green and 2 seconds after yellow

**Traffic Light Simulator**



```
@SuppressWarnings("static-access")
public void run() {
    try {
        this.t.sleep(this.delay);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    while (!suspended) {
        |
        System.out.println(this.message);

        try {

            this.t.sleep(10_000);

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

When the Thread starts to initially run it will hit a period of sleep. This allows the threads to begin their functionality of  printing the required message to the scheduled timing.

Green has no delay and will fire again after 10 seconds. (5 seconds until yellow fires).
Yellow will fire after a 5 second delay then again after 10 seconds ( 2 seconds before red).
Red will fire after 7 seconds then again after 10 seconds.

## ***Potential improvements***

1. I would like to make it so that I'm not just using Scanner.nextLine() as a speed bump for the process having the user press enter and actually take in the users input.
2. I would think it better to not have to use the delay property to cause the lights to be properly staggered as it obscures the true intention of have a 5 seconds of green light, 2 seconds of yellow and 3 seconds of red.

## ***Conclusion:***

In conclusion, the program passes all requirements, but perhaps done in an unorthodox fashion and would require some reworking.

## **Traffic Light Simulator**

Initially I had begun the process without using multi-threading and it cost me a fair amount of time refactoring my code to include that requirement. I should have read the instructions more thoroughly.

I had surplus time to complete the project, but the project probably could have been further expedited by keeping the documentation in sync with the coding aspect.