# Working Document:

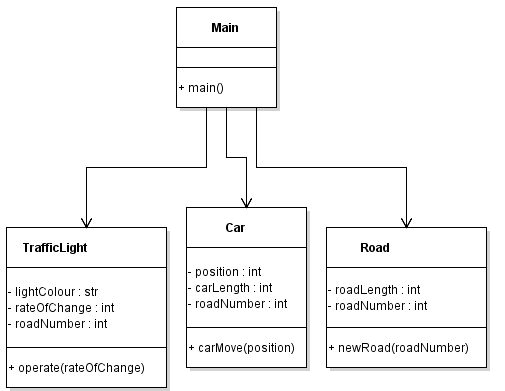
# Problem Specification:

The current problem at hand is to plan, design and start developing a car traffic simulator. This project has been split into two main parts, the first of which is to establish a basic simulator with minimal functionality which can be built upon as the project progresses.

The program will be used to visually represent a typical urban traffic setup complete with vehicles of different sizes, roads with varying lengths and directions as well as traffic lights. For the first part of this project a basic traffic setup is planned consisting of a single car and two road segments separated by a traffic light.

The first part of the program will not behave transiently as described above but more statically. It will run the main simulator with no graphical user interface (GUI) but instead through a series of out print statements showing changes in car position along each road segment as well as random traffic light parameters.

# Problem Decomposition Using UML Class Diagrams:



The first part of the project consists of establishing a basic traffic simulator, based on the above UML diagram it can be seen that the program fundamentals can be achieved with 4 major classes; Main(), TrafficLight(), Car() and Road().

**Main():** The Main() class is the designated class used to run the simulator program. It is used to call all methods from other classes and loop them to simulate arbitrary increments of time, hence why its member fields are set to public.

**Car():** The Car() class contains the objects (position, carLength) required to simulate a single vehicle of length 1 moving along a road segment. Other than the getter and setter methods for the member fields the class uses a carMove() method that gets the position field and updates it to +1.

**Road():** The Road() class contains objects (roadLength, roadNumber) that both relate to unique road objects. As the main() simulator runs the position member field references the roadLength to track when it reaches the last position on that road. The newRoad() method is then run to get the current roadNumber and increase it by a value of +1.

**TrafficLight():** The TrafficLight() class dictates when the car object moves to the next road section. It has an operate() method that references a defined rateOfChange value and runs a random integer function to produce a random number within a specified domain. If the random number is lower than the rateOfChange the lightColour is set to the opposite value; (red to green or vise versa).

# Part 2:

# 

The second part of the project consists of implementing the existing simulator into a functioning graphical user interface. From the above UML diagram it was planned to extend the simulators’ main classes off of a new MainFrame() which would be responsible for the generation and updating of the GUI. A significant change was implemented from the previous design by the removal of the Road() class. Instead of inserting road sections of varying sizes the GUI uses a grid section with selectable boxes that can be selected and stored to form a chain of road segments. These boxes were created to have their own custom coordinate system to allow the positions to be stores in arrays to provide the ability to save and load designs.

**Main():** The Main() class is the designated class used to run the simulator program. It is used to call all methods from other classes and loop them to simulate arbitrary increments of time, hence why its member fields are set to public. It also implements the run() method which repaints the GUI if it detects updates provided by user input.

**Car():** The Car() class contains the objects (position, carLength) required to simulate a single vehicle of length 1 moving along a road segment. Other than the getter and setter methods for the member fields the class uses a carMove() method that gets the position field and updates it to +1. Further updates to this design would have included the visual representation of the car on the grid space by calling (position) and painting in that coordinate on the grid.

**TrafficLight():** The TrafficLight() class dictates when the car object moves to the next road section. It has an operate() method that references a defined rateOfChange value and runs a random integer function to produce a random number within a specified domain. If the random number is lower than the rateOfChange the lightColour is set to the opposite value; (red to green or vise versa). Further design implementation would have also provided a visual representation of this class. The traffic light would be set at any intersection of road sections with a constantly changing state between green and red. It was also planned on allowing the traffic light to randomise the direction the car would follow at the intersection based off an arbitrary value.