
# HOMEWORK 4 - Spring 2023

---

**HOMEWORK 4 - due Tuesday, March 21st no later than 7:00PM**

**Reminders:**

- **Submit your files only via CodeGrade in the Content page on Brightspace. Access CodeGrade by clicking "HW4 - CODEGRADE SUBMISSION LINK" for your submission.**
- **Use of a package is optional. If you wish to use it, make sure to name it "hw4" (all in lower case). Otherwise, you will lose points.**
- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **You are allowed to use any Java API Data Structure classes such as LinkedList, ArrayList or Vector to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

---

## Assignment

Simulations can be useful to test how random processes are affected by different initial conditions. For example, simulating the arrival of vehicles at a busy intersection can help to determine the optimal timing mechanism controlling how long the stoplight should remain green for a particular road. In this scenario, a simulation can proceed step by step - each step representing a specific unit of time. During each time step, vehicles may arrive in any of the lanes of the intersection with a fixed a probability, and vehicles may pass through the intersection if the light is green in their direction. By changing the arrival probability and stoplight timing, expected average waiting times for vehicles passing through the intersection can be generated and compared for optimization.

In this assignment, you will be required to write a Java program to simulate vehicles passing through a busy intersection. The intersection between Route 216 and 320 Road is currently serviced by an old fashioned timer based stop light that frequently causes traffic jams. CSE Associates, Inc. has hired you to create a simulation of how their new Stop Light (model # 214) would be able to better serve this intersection. Your simulation must keep track of the vehicles that arrive at the intersection, and show the state of the vehicles queued up at the light, and describe what happens in every time step.

### Simulation Procedure

The primary simulation procedure should be contained within a `static` function called `simulate()` inside the `IntersectionSimulator` class. This method should begin by initializing the intersection based on a few parameters (*listed below*), and enter a loop which executes the time steps.

Your program will simulate the arrival of vehicles on up to **four** roads of an intersection for a specified simulation time. You may assume that each road has two traveling directions (*forward and backward*), with each traveling direction having three lanes (a *left turn* lane, a *middle* lane, and a *right turn* lane), for a total of `2 roads * 2 directions per road * 3 lanes per direction = 12 lanes`. The simulation will take five parameters at the onset:

1. **`simulationTime (int)`:** Indicates the desired total simulation time.

2. **`arrivalProbability (double)`:** Indicates the probability that a vehicle will arrive at *any* lane during a time step.
3. **`numRoads (int)`:** Indicates the number of two-way roads that meet at this intersection.
4. **`names (String[])`:** An array of size `numRoads` indicating the name of each road.
5. **`greenTimes (int[])`:** An array of size `numRoads` indicating the green time for each road.

Note: The two array parameters (*names and greenTimes*) should be equal in size and have corresponding indices (*i.e. names[i] corresponds to a road having a green time of geenTimes[i]*).

**Initialization**

Before the first time step, your program should create a new array of `TwoWayRoads` equal in size to the array parameters. A new `TwoWayRoad` object should be created at each index of this new array, initializing each new road with the `name` and `greenTime` at the corresponding index of the array parameters (*names and greenTimes, respectively*). Once this array has been created and initialized, it should be used to construct an `Intersection` instance, which will be used to keep track of the light during simulation and allow vehicles to pass through the intersection. Lastly, a `BooleanSourceHW4` object should be created (*initializing it's probability member variable to arrivalProbability*) which will be used to determine if vehicles have arrived during simulation.

During execution, the 'light' will be simulated using the `lightIndex` member variable of the `Intersection` instance. This variable indicates the index of the road in the `roads` array which currently has the active light (*the current active light may be in the GREENstate or LEFT_TURN state, as described by the LightValue enum below*). When the `Intersection` instance is constructed, it initializes the `lightIndex` member variable to `0` and the `countdownTimer` to the `greenTime` member variable of `roads[lightIndex]`. After each time step, the timer is decremented by 1. Once the timer reaches `0`, the `lightIndex` is incremented, returning back to `0` if it equals the size of the `roads` array (*i.e. modular arithmetic*), and the `countdownTimer` is again set to `roads[lightIndex]`. This process repeats continually until the simulation ends.

**Time Steps**

On each time step, the program should determine if a vehicle has arrived for *each* lane in the intersection *(all 12)*. This can be accomplished by calling the `occursHW4()` method on the `BooleanSourceHW4` object for each lane. If a vehicle has arrived (*i.e. occursHW4() returns true*), the program should create a new `Vehicle` object, initialize it's `timeArrived` member variable to the current time step value, and enqueue the vehicle onto the appropriate lane.

After all lanes have been considered for arrival, `roads[lightIndex]` should pass vehicles through the intersection - one vehicle per lane, but **only** if the lane is allowed to proceed (*see the LightValue enum below for more detail on the light rules*). When a vehicle is dequeued from a lane, the program should add the vehicle's wait time to the total wait time for the simulation, and increment the number of cars passed through the intersection. The vehicle's wait time can be calculated by subtracting it's `arrivalTime` from the current time step value. If all lanes are empty after having been dequeued (*or ignored if they were empty*), then the program should preempt the countdown timer and switch the light to the next road.

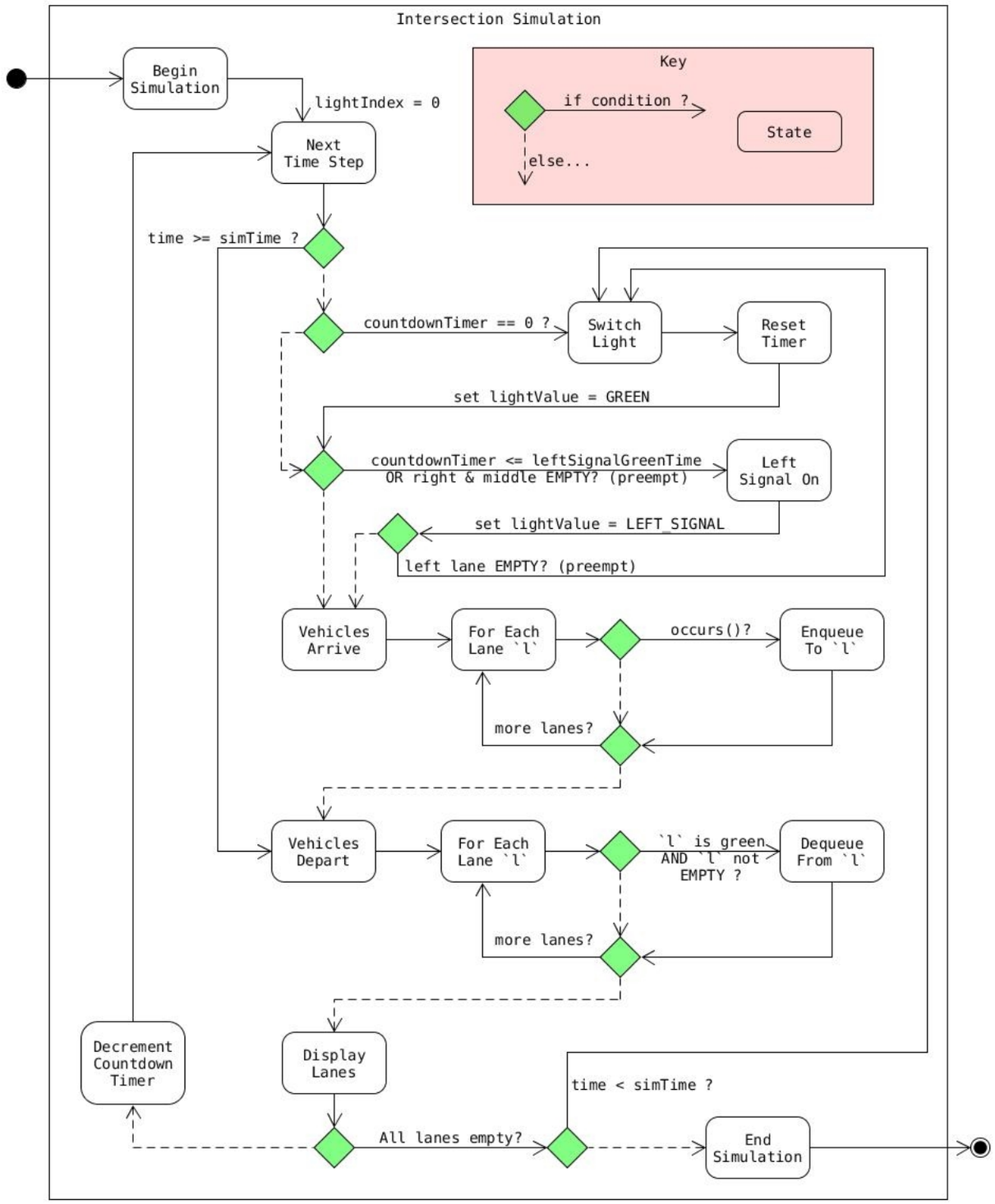
Note: On any particular time step, a **maximum of 6** vehicles may be queued onto lanes of the intersection, and a **maximum of 4** vehicles may pass through an intersection (*if the lightValue is GREEN, then the right and middle lanes can proceed in both directions, but the left lanes cannot*). In other words, any lane can be enqueued and dequeued only **once** on a given time step. A case you should consider is if a lane of the road with the green light is empty at the beginning of the time step, and has an arrival during the time step. If this happens, then the vehicle will be enqueued and dequeued from the lane during the same time step, resulting in a wait time of zero (*this is the equivalent of driving through a green light without having to wait*).

After these operations have been completed, the simulation should output the current state of the intersection to the user. This should include the number of cars on each of the 12 lanes of the intersection, as well as their current wait times. In addition, the current total wait time, total number of cars, and average wait time should be shown in tabular form (*please see the sample I/O for examples*). The program now proceeds to the next time step, repeating the process until the time step value is equal to `simulationTime`.

**Finalization**

Once the time step counter reaches `simulationTime`, no more vehicles should arrive at the intersection. The simulation should proceed as normal until **all** lanes have been cleared and the intersection is empty. Once the intersection is clear of vehicles, the simulation should end and the program should display the results to the user. The output parameters should include the total wait time by all vehicles, the total number of vehicles passing through the intersection, and the average wait time for vehicles during the simulation.

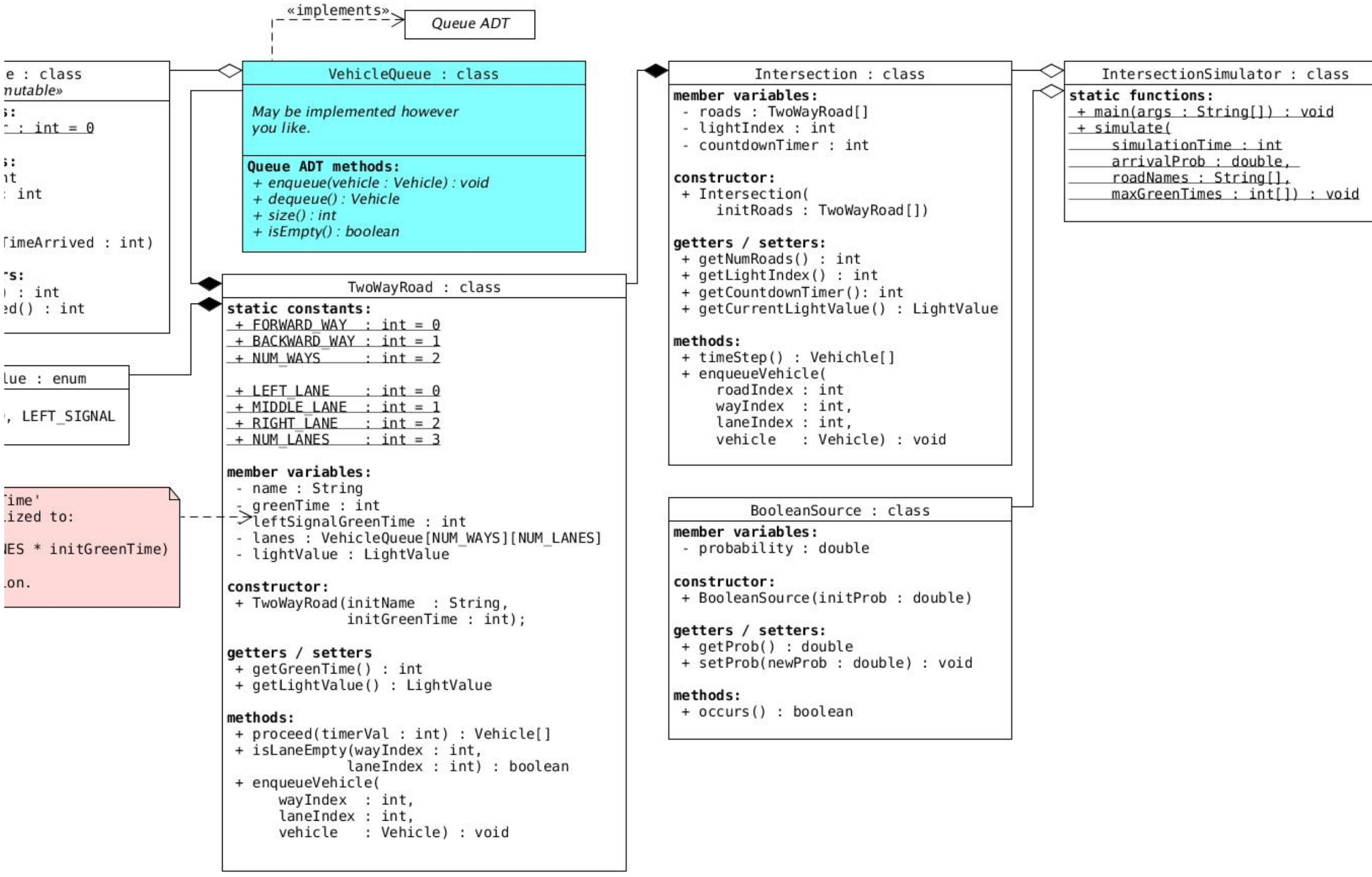
The entire procedure is summarized by the following *Activity Diagram*:




**Activity Diagram**

## Required Classes

The following outlines the required classes you must implement for this assignment. You may write additional classes if you feel they will help you implement this assignment, as well as private helper methods. However, each of the classes and methods below **must** be included if you are to receive full credit. Also note that you should **not** include any method which exposes references to private member variables or allows a user to interact with the class in an undefined way (*if you are unsure, please contact a TA or the instructor*). The following *UML Diagram* outlines the class relationship structure:


«implements»
Queue ADT
e : class
nutable»
s:
: int = 0
s:
nt
: int
TimeArrived : int)
rs:
) : int
ed() : int
lue : enum
, LEFT_SIGNAL
Time'
ized to:
NES * initGreenTime)
on.
VehicleQueue : class
May be implemented however you like.
Queue ADT methods:
+ enqueue(vehicle : Vehicle) : void
+ dequeue() : Vehicle
+ size() : int
+ isEmpty() : boolean
TwoWayRoad : class
static constants:
+ FORWARD_WAY : int = 0
+ BACKWARD_WAY : int = 1
+ NUM_WAYS : int = 2
+ LEFT_LANE : int = 0
+ MIDDLE_LANE : int = 1
+ RIGHT_LANE : int = 2
+ NUM_LANES : int = 3
member variables:
- name : String
- greenTime : int
- leftSignalGreenTime : int
- lanes : VehicleQueue[NUM_WAYS][NUM_LANES]
- lightValue : LightValue
constructor:
+ TwoWayRoad(initName : String,
initGreenTime : int);
getters / setters
+ getGreenTime() : int
+ getLightValue() : LightValue
methods:
+ proceed(timerVal : int) : Vehicle[]
+ isLaneEmpty(wayIndex : int,
laneIndex : int) : boolean
+ enqueueVehicle(
wayIndex : int,
laneIndex : int,
vehicle : Vehicle) : void
Intersection : class
member variables:
- roads : TwoWayRoad[]
- lightIndex : int
- countdownTimer : int
constructor:
+ Intersection(
initRoads : TwoWayRoad[])
getters / setters:
+ getNumRoads() : int
+ getLightIndex() : int
+ getCountdownTimer(): int
+ getCurrentLightValue() : LightValue
methods:
+ timeStep() : Vehichle[]
+ enqueueVehicle(
roadIndex : int
wayIndex : int,
laneIndex : int,
vehicle : Vehicle) : void
IntersectionSimulator : class
static functions:
+ main(args : String[]) : void
+ simulate(
simulationTime : int
arrivalProb : double,
roadNames : String[],
maxGreenTimes : int[]) : void
BooleanSource : class
member variables:
- probability : double
constructor:
+ BooleanSource(initProb : double)
getters / setters:
+ getProb() : double
+ setProb(newProb : double) : void
methods:
+ occurs() : boolean

**UML Diagram**

## 1. `BooleanSourceHW4` class

(Provided as attachment to the specification) A class called `BooleanSourceHW4` which abstracts a random occurrence generator. This class should be constructed with an initial arrival probability ($0.0 < probability \leq 1.0$) which represents the likelihood that a `Vehicle` will arrive at any particular lane at the beginning of each time step. This method should also contain a single method, `occursHW4()` which returns `true` if a vehicle arrives and `false` it does not.

- **`private double probability`**
- **`public BooleanSourceHW4(double initProbability)`**
  - *Brief:*
    - Constructor which initializes the `probability` to the indicated parameter.
  - *Parameters:*
    - `initProbability`
      - Probability used to construct this `BooleanSourceHW4` object. The probability should be greater than 0 and less than or equal to 1.
  - *Preconditions:*
    - `0 < initProbability ≤ 1`.
  - *Throws:*
    - `IllegalArgumentException`
      - If `initProbability ≤ 0` or `initProbability > 1`.
  - *NOTE:*
    - This class works with hw4randomNumbers.txt and hw4randomFlags.txt text files that are attached to the homework specification.
- **`public boolean occursHW4()`**
  - *Brief:*
    - Method which returns `true` with the probability indicated by the member variable `probability`.
  - *Preconditions:*
    - `probability` is a valid probability ($0 < probability \leq 1$).
  - *Returns:*
    - Boolean value indicating whether an event has occurred or not.
  - *Note:*
    - This method will utilize a saved random number to generate the random occurrence.

## 2. `LightValue` enum

Write a simple `Enum` named `LightValue`, which lists the phases a particular stoplight lane may be in. These states should include `GREEN`, `RED`, and `LEFT_SIGNAL` (*we are considering yellow to be part of green*). Each `TwoWayRoad` instance (*defined below*) will have its own `LightValue`, which should correspond to the following rules:

1. **`GREEN`** indicates that the ***right*** and ***middle*** lanes may proceed, but the ***left*** lane cannot (*for both directions*).
2. **`RED`** indicates that no lane may proceed (*for both directions*).
3. **`LEFT_SIGNAL`** indicates that ***left*** can proceed, but the ***right*** and ***middle*** lanes cannot (*for both directions*).

Cars in a particular lane may proceed (dequeue one car per time interval) when it is their turn to go, according to the rules above. If you still need help with Java `Enum` types, you can read about them in this Java tutorial.

## 3. `Vehicle` Class

Write a fully documented class named `Vehicle`. This class represents a car which passes through the intersection. Each instance must contain the `serialId` (the first car to arrive at the intersection is `serialId 1`, the second car to arrive is `serialId 2`, the `n`'th car to arrive will have `serialId >n`) and the time it arrived (stored as an `int`). The car must be initialized with a `serialId` and the time it arrived. The serial counter is static and stores the number of vehicles that have arrived at the intersection so far. It is the only variable that is modifiable.

The Vehicle class itself is actually immutable. This means once it has been constructed, no data within the instance can be changed (*the static serialCounter can and should be incremented on each constructor call*). From the UML diagram, note that the public constructor takes all the member variables as arguments. Data can still be read via the getter methods.

- **`private static int serialCounter = 0`**
- **`private int serialId`**
- **`private int timeArrived`**
- **`public Vehicle(int initTimeArrived)`**
  - *Brief:*
    - Default Constructor. You should automatically increment the serialCounter, and set the serialId to its new value.
  - *Parameters:*
    - `initTimeArrived`
      - Time the vehicle arrived at the intersection.
  - *Preconditions:*
    - `initTimeArrived > 0`.
  - *Throws:*

    - `IllegalArgumentException`
        - If `initTimeArrived` ≤ 0.

## 4. `VehicleQueue` class

Lanes in our simulator will be modelled as a `Queue` of `Vehicles`. You may implement a `Queue` of vehicles however you like, and are encouraged to use any Java API class you prefer. Remember that the `VehicleQueue` class **must** implement the following methods in order to comply with the `Queue ADT`:

- **`void enqueue(Vehicle v)`**
- **`Vehicle dequeue()`**
- **`int size()`**
- **`boolean isEmpty()`**

Note: If you decide to use a Java API class to implement `VehicleQueue`, you must use inheritance (extend a Java API class) to simplify the class definition.

## 5. `TwoWayRoad` Class

Write a fully documented class called `TwoWayRoad` that represents one of the roads in our intersection. Each road it bi-directional, with each direction having **three** lanes - a *left turn* lane, a *middle* lane, and a *right turn* lane. Lanes, modelled by `VehicleQueues`, hold the vehicles before they pass through the intersection. These lanes will be stored in a two dimensional array - the first index indicates the *direction of travel* on the road, and the second index indicates the *lane on the road*. In order to access a specific direction, you should use the constants `FORWARD_WAY` and `BACKWARD_WAY` to access the directions of the road. In order to access specific lanes in a particular direction, you should use the second dimension of the array, accessed by the consants `LEFT_LANE, MIDDLE_LANE, RIGHT_LANE`. Your `TwoWayRoad` class must be able to check whether any of the lanes in the road have `Vehicle` objects in them by using the `boolean isEmpty(int wayIndex, int laneIndex)` method. It should also be able to add `Vehicle` objects to the lanes using the `void enqueueVehicle(int wayIndex, int laneIndex, Vehicle vehicle)` method. Furthermore, you should allow vehicles to pass through the intersection, adding the `Vehicles` that have been dequeued to an array to returned to the caller.

Brief:

- **`public final int FORWARD_WAY = 0;`**
- **`public final int BACKWARD_WAY = 1;`**
- **`public final int NUM_WAYS = 2;`**
- **`public final int LEFT_LANE = 0;`**
- **`public final int MIDDLE_LANE = 1;`**
- **`public final int RIGHT_LANE = 2;`**

- **`public final int NUM_LANES = 3;`**
- **`private String name`**
- **`private int greenTime`**
  - The **maximum total** number of steps this road can be active (*this number is **inclusive** of the leftSignalGreenTime* ).
- **`private int leftSignalGreenTime`**
  - The number of steps this road remains in the `LEFT_SIGNAL` state.
  - Should be initialized to `1.0/NUM_LANES * greenTime` in the constructor.
- **`private VehicleQueue lanes[NUM_WAYS][NUM_LANES]`**
- **`private LightValue lightValue`**
- **`public TwoWayRoad(String initName, int initGreenTime)`**
  - *Brief:*
    - Default Constructor. You should automatically initialize the array and all of its member objects, as well as initializing leftSignalGreenTime to `1.0/NUM_LANES * initGreenTime`.
  - *Parameters:*
    - `initName`
      - The name of the road.
    - `initGreenTime`
      - The amount of time that the light will be active for this particular road. This is the total of the time the light should display green for cars going forward/turning right, as well as for cars going left.
  - *Preconditions:*
    - `initGreenTime > 0`.
  - *Postconditions:*
    - This road is initialized with all lanes initialized to empty queues, and all instance variables initialized.
  - *Throws:*
    - `IllegalArgumentException`
      - If `initGreenTime ≤ 0 or initName=null`.
- **`public Vehicle[] proceed(int timerVal)`**

- *Brief:*
  - Executes the passage of time in the simulation. The `timerVal` represents the current value of a countdown timer counting down total green time steps. The light should be in state `GREEN` any time the timerval is greater than leftSignalGreenTime. When timerVal is less than or equal to `leftSignalGreenTime`, the light should change to `LEFT_SIGNAL`. After the execution of timerVal == 1, or if there are no vehicles left the light should change state to `RED`.
- *Parameters:*
  - `timerVal`
    - The current timer value, determines the state of the light.
- *Preconditions:*
  - The TwoWayRoad object should be instantiated.
- *Returns:*
  - An array of `Vehicles` that has been dequeued during this time step.
- *Postconditions:*
  - Any `Vehicles` that should have been dequeued during this time step should be dequeued and placed in the return array.
- *Throws:*
  - `IllegalArgumentException`
    - If `timerval ≤ 0`.

- **`public void enqueueVehicle(int wayIndex, int laneIndex, Vehicle vehicle)`**
  - *Brief:*
    - Enqueues a vehicle into a the specified lane.
  - *Parameters:*
    - `wayIndex`
      - The direction the car is going in.
    - `laneIndex`
      - The lane the car arrives in.
    - `vehicle`
      - The vehicle to enqueue; must not be null.
  - *Preconditions:*
    - The TwoWayRoad object should be instantiated.
  - *Postconditions:*
    - Given that the vehicle specified was not `null`, and the position given was not invalid (and no exception was thrown), the vehicle should be added to the end of the proper queue.
  - *Throws:*
    - `IllegalArgumentException`

        - If `wayIndex &gt 1 || wayIndex &lt 0 || laneIndex &lt 0 || laneIndex &gt 2` or `vehicle==null`.
- **`public boolean isLaneEmpty(int wayIndex, int laneIndex)`**
    - *Brief:*
        - Checks if a specified lane is empty.
    - *Parameters:*
        - `wayIndex`
            - The direction of the lane.
        - `laneIndex`
            - The index of the lane to check.
    - *Preconditions:*
        - The `TwoWayRoad` object should be instantiated.
    - *Returns:*
        - `true` if the lane is empty, else `false`.
    - *Postconditions:*
        - The `TwoWayRoad` object should remain unchanged
    - *Throws:*
        - `IllegalArgumentException`
            - If `wayIndex &gt 1 || wayIndex &lt 0 || laneIndex &lt 0 || laneIndex &gt 2`.

## 6. `Intersection` class

Write a fully documented class named `Intersection`. This class represents a crossing of two or more roads at a stop light in our simulation. The class consists of an array of `TwoWayRoad` objects representing the crossing roads, as well as a countdown timer and a light index. `Intersection` must contain the following private member variables: **`roads`** `(TwoWayRoad[])`, **`lightIndex`** `(int)`, and **`countdownTimer`** `(int)`. The `Intersection` class must also feature the following public methods: **`void timeStep()`**, and **`void enqueueVehicle(int roadIndex, int wayIndex, int laneIndex)`**, as well as a **`display()`** method which prints the intersection to the terminal.

- **`private TwoWayRoad[] roads`**
    - Array of roads which cross at this intersection.
- **`private int lightIndex`**
    - Indicates the road in `roads` with the active light (*either green or left turn signal*).
- **`private int countdownTimer`**
    - Tracks the remaining time steps available for the road currently indicated by `lightIndex`.

- **`public Intersection(TwoWayRoads[] initRoads)`**
  - *Brief:*
    - Constructor which initializes the `roads` array.
  - *Parameters:*
    - `initRoads`
      - Array of roads to be used by this intersection.
  - *Preconditions:*
    - `initRoads` is not `null`.
    - Length of `initRoads`is less than or equal to `MAX_ROADS`.
    - All indices of `initRoads` are not `null`.
  - *Postconditions:*
    - This object has been initialized to a `Intersection` object managing the `roads` array.
  - *Throws:*
    - `IllegalArgumentException`
      - If `initRoads` is `null`.
      - If any index of `initRoads` is `null`.
      - `initRoads.length > MAX_ROADS`.
- **`public Vehicle[] timeStep()`**
  - *Brief:*
    - Performs a single iteration through the intersection. This method should apply all the logic defined in this specification related to the passing of cars through the intersection and switching the selected road (*Note: LightValue changes for a particular road should be handled within the TwoWayRoad class itself and not within this method*). Please refer to the `Simulation Procedure` section above for instructions on how to apply this procedure.
  - *Postconditions:*
    - The intersection has dequeued all lanes with a green light (*if non-empty*) and returned an array containing the `Vehicles`.
  - *Returns:*
    - An array of `Vehicles` which have passed though the intersection during this time step.
- **`public void enqueueVehicle(int roadIndex, int wayIndex, int laneIndex, Vehicle vehicle)`**
  - *Brief:*
    - Enqueues a vehicle onto a lane in the intersection.
  - *Parameters:*
    - `roadIndex`
      - Index of the road in `roads` which contains the lane to enqueue onto.

    - `wayIndex`
        - Index of the direction the vehicle is headed. Can either be `TwoWayRoad.FORWARD` or `TwoWayRoad.BACKWARD`
    - `laneIndex`
        - Index of the lane on which the vehicle is to be enqueue. Can either be `TwoWayRoad.RIGHT_LANE`, `TwoWayRoad.MIDDLE_LANE`, or `TwoWayRoad.LEFT_LANE`.
    - `vehicle`
        - The `Vehicle` to enqueue onto the lane.
  - *Preconditions:*
    - `0 ≤ roadIndex < roads.length`.
    - `0 ≤ wayIndex < TwoWayRoad.NUM_WAYS`.
    - `0 ≤ laneIndex < TwoWayRoad.NUM_LANES`.
    - `vehicle != null`.
  - *Throws:*
    - `IllegalArgumentException`
        - If `vehicle` is `null`.
        - If any of the index parameters above are not within the valid range..
- **`public void display()`**
  - *Brief:*
    - Prints the intersection to the terminal in a neatly formatted manner. See the sample I/O for an example of what this method should display.
  - *Note:*
    - The sample I/O shown below requires you to print some of the `VehicleQueues` in reverse. Think about how you might be able to do this using a stack (*e.g. java.util.Stack* ).

## 7. `IntersectionSimulator` class

Write a fully documented class named `IntersectionSimulator`. This class represents the manager of the simulation -- it does the heavy lifting, per se. The main function's responsibility is to get the parameters for the simulation and pass them to the `simulate()` method, either by interactive prompt or command line (See below).

- **`public static void main(String args[])`**
    - Start for application, asks user for following values: `simulationTime (int)`, `arrivalProbability (double)`, `numRoads (int)`, a name for each road, and a "green" time for each road. This method also parses command line for these args. If `args.length < 5`, the above is read in at execution time. Otherwise, refer to the end of this document on how to parse the command line arguments.
- **`public static void simulate(int simulationTime, double arrivalProbability, String[] roadNames, int[] maxGreenTimes):`**
    - This method does the actual simulation. Above, a *Activity Diagram* is presented to demonstrate how the simulation works. This method actually implements the algorithm described by that diagram, using `Intersection`, `BooleanSourceHW4`, and `TwoWayRoad`.

Note: The `simulationTime` is how long cars can 'appear'. The actual simulation can last longer -- long enough for every car to pass the intersection.

Warning: You should make sure that you catch **ALL** exceptions that you throw anywhere in your code. Exceptions are used to indicate illegal or unsupported operations so that your program can handle unexpected events gracefully and prevent a crash. Your program should **NOT** crash from any of the above exceptions (*it should not crash from any exception, but especially not one that you throw yourself*).

---

# Input Format:

The simulator should be run either by prompting the user for input at the beginning of the program or by parsing the command line arguments passed to the program. You may assume that the arguments will be passed in in the proper order and of the correct data type.

**Command Line Arguments**

Command line arguments are text values passed into a program at run time. For example, to compile your java files, you could run javac *.java in a terminal. In this homework assignment, you will be able to start your program with the following options in order:

- `simulationTime`
- `arrivalProbability`
- `numRoads`
- `[numRoads long list of names (Strings)]`
- `[numRoads long list of maxGreenTimes (ints)]`

Example execution command line:

```
java IntersectionSimulator 5 0.25 3 Road1 Road2 Road3 3 4 5
```

To make it easier for you to include command line arguments in your code, here is some sample code for parsing required arguments:

```
if (args.length > 1) {

 int simTime    = Integer.parseInt(args[0]);
 double prob    = Double.parseDouble(args[1]);
 int numRoads   = Integer.parseInt(args[2]);
 String[] names = new String[numRoads];
 String[] times = new int[numRoads];

 for (int i = 0; i < numRoads; ++i) {
  names[i] = args[3 + i];
  times[i] = Integer.parseInt(args[3 + numRoads + i]);
 }

 // process args in simTime, prob, numRoads, names, times.

}
// Else: interactive
```

To test in Eclipse, go to Menu Bar > Project > Properties > Run/Debug settings > Edit the listed run configuration (usually the name of your project) > Arguments tab.

---

## Output Format:

All lists must be printed in a nice and tabular form as shown in the sample output. You may use C style formatting as shown in the following example. The example below shows two different ways of displaying the name and address at pre-specified positions 21, 26, 19, and 6 spaces wide. If the '-' flag is given, then it will be left-justified (padding will be on the right), else the region is right-justified. The 's' identifier is for strings, the 'd' identifier is for integers. Giving the additional '0' flag pads an integer with additional zeroes in front.

```
String name = "Doe Jane";
String address = "32 Bayview Dr.";
String city = "Fishers Island, NY";
int zip = 6390;

System.out.println(String.format("%-21s%-26s%19s%06d", name, address, city, zip));
System.out.printf("%-21s%-26s%19s%06d", name, address, city, zip);

// Output
Doe Jane              32 Bayview Dr.             Fishers Island, NY 06390
Doe Jane              32 Bayview Dr.             Fishers Island, NY 06390
```

---

## Sample Input/Output:

```
// Comment in green, input in red, output in black


Sample I/O:

Welcome to IntersectionSimulator 2021

Input the simulation time: 6
Input the arrival probability: 0.2          // chance of car entering a lane, for all lanes.
Input number of Streets: 2
Input Street 1 name: Route 216              // must be unique names & not the empty string
Input Street 2 name: Route 216              // on duplicate detected, re-prompt for name
Duplicate Detected.
Input Street 2 name: 320 Road
Input max green time for Route 216: 4
Input max green time for 320 Road: 3

Starting Simulation...

################################################################################
```

```
Time Step: 1

    Green Light for Route 216.  // Select first road at start.
    Timer = 4                   // Initialize timer to max green time for road.

    ARRIVING CARS:
        Car[001] entered Route 216, going FORWARD in LEFT lane.
        Car[002] entered Route 216, going BACKWARD in MIDDLE lane.
        Car[003] entered 320 Road, going FORWARD in RIGHT lane.

    PASSING CARS:
        // Car[001] can't pass since 216-FORWARD-LEFT has red light (x).
        Car[002] passes through. Wait time of 0.
        // Car[003] can't pass since 320-FORWARD-RIGHT has red light (x).


    Route 216:
                            FORWARD                 BACKWARD
    ================================    ================================
                              [001] [L] x     [R]
    --------------------------------    --------------------------------
                                  [M]       [M] // [002] passed through.
    --------------------------------    --------------------------------
                                  [R]     x [L]
    ================================    ================================

    320 Road:
                            FORWARD                 BACKWARD
    ================================    ================================
                                  [L] x   x [R]
    --------------------------------    --------------------------------
                                  [M] x   x [M]
    --------------------------------    --------------------------------
                             [003] [R] x   x [L]
    ================================    ================================


    STATISTICS:
        Cars currently waiting: 2 cars
        Total cars passed:      1 cars
        Total wait time:        0 turns
        Average wait time:      0.00 turns

################################################################################

Time Step: 2

    Green Light for Route 216.
    Timer = 3                   // Timer decrements

    ARRIVING CARS:
        Car[004] entered Route 216, going FORWARD in MIDDLE lane.
        Car[005] entered Route 216, going FORWARD in RIGHT lane.
        Car[006] entered Route 216, going BACKWARD in RIGHT lane.
        Car[007] entered 320 Road, going FORWARD in LEFT lane.
```

```
    Car[008] entered 320 Road, going BACKWARD in MIDDLE lane.

PASSING CARS:
    Car[004] passes through. Wait time of 0.
    Car[005] passes through. Wait time of 0.
    Car[006] passes through. Wait time of 0.


Route 216:
                          FORWARD                BACKWARD
================================ ================================
                           [001] [L] x     [R] // [006] passed through.
-------------------------------- --------------------------------
      [004] passed through. // [M]       [M]
-------------------------------- --------------------------------
      [005] passed through. // [R]     x [L]
================================ ================================

320 Road:
                          FORWARD                BACKWARD
================================ ================================
                           [007] [L] x   x [R]
-------------------------------- --------------------------------
                                 [M] x   x [M] [008]
-------------------------------- --------------------------------
                           [003] [R] x   x [L]
================================ ================================

    STATISTICS:
        Cars currently waiting: 4 cars
        Total cars passed:      4 cars
        Total wait time:        0 turns
        Average wait time:      0.00 turns

################################################################################

Time Step: 3

    Green Light for Route 216.
    Timer = 2

    ARRIVING CARS:
        Car[009] entered Route 216, going FORWARD in RIGHT lane.
        Car[010] entered Route 216, going BACKWARD in LEFT lane.
        Car[011] entered 320 Road, going FORWARD in MIDDLE lane.
        Car[012] entered 320 Road, going BACKWARD in RIGHT lane.

    PASSING CARS:
        Car[009] passes through. Wait time of 0.


    Route 216:
                              FORWARD                BACKWARD
    ================================ ================================
```

```
                          [001] [L] x     [R]
    ------------------------------          ------------------------------
                                  [M]       [M]
    ------------------------------          ------------------------------
          [009] passed through. // [R]     x [L] [010]
    ==============================          ==============================

    320 Road:
                           FORWARD                BACKWARD
    ==============================          ==============================
                          [007] [L] x   x [R] [012]
    ------------------------------          ------------------------------
                          [011] [M] x   x [M] [008]
    ------------------------------          ------------------------------
                          [003] [R] x   x [L]
    ==============================          ==============================


    STATISTICS:
        Cars currently waiting: 7 cars
        Total cars passed:      5 cars
        Total wait time:        0 turns
        Average wait time:      0.00 turns

################################################################################

Time Step: 4
Left Signal for Route 216. // Light switches to left arrow.
Timer = 1

ARRIVING CARS:
    Car[013] entered Route 216, going FORWARD in MIDDLE lane.
    Car[014] entered 320 Road, going FORWARD in MIDDLE lane.
    Car[015] entered 320 Road, going BACKWARD in RIGHT lane.

PASSING CARS:
    Car[001] passes through. Wait time of 3.
    Car[010] passes through. Wait time of 1.


Route 216:
                           FORWARD                BACKWARD
==============================          ==============================
                                  [L]     x [R]
------------------------------          ------------------------------
                            [013] [M] x   x [M]
------------------------------          ------------------------------
                                  [R] x     [L]
==============================          ==============================

320 Road:
                           FORWARD                BACKWARD
==============================          ==============================
                            [007] [L] x   x [R] [012][015]
```

```
    ------------------------------                ------------------------------
                         [014][011] [M] x   x [M] [008]
    ------------------------------                ------------------------------
                               [003] [R] x   x [L]
    ==============================                ==============================


    STATISTICS:
        Cars currently waiting: 8 cars
        Total cars passed:      7 cars
        Total wait time:        4 turns
        Average wait time:      0.57 turns

################################################################################

Time Step: 5

    Green Light for 320 Road.
    Timer = 3

    ARRIVING CARS:
        Car[016] entered Route 216, going FORWARD in LEFT lane.
        Car[017] entered Route 216, going BACKWARD in MIDDLE lane.
        Car[018] entered 320 Road, going FORWARD in RIGHT lane.

    PASSING CARS:
        Car[011] passes through. Wait time of 2.
        Car[003] passes through. Wait time of 4.
        Car[012] passes through. Wait time of 2.
        Car[008] passes through. Wait time of 3.


Route 216:
                           FORWARD                BACKWARD
==============================                ==============================
                             [016] [L] x   x [R]
------------------------------                ------------------------------
                             [013] [M] x   x [M] [017]
------------------------------                ------------------------------
                                   [R] x   x [L]
==============================                ==============================

320 Road:
                           FORWARD                BACKWARD
==============================                ==============================
                             [007] [L] x     [R] [015]
------------------------------                ------------------------------
                             [014] [M]       [M]
------------------------------                ------------------------------
                             [018] [R]     x [L]
==============================                ==============================


STATISTICS:
```

```
        Cars currently waiting:  7 cars
        Total cars passed:       11 cars
        Total wait time:         15 turns
        Average wait time:       1.36 turns

################################################################################

Time Step: 6

    Green Light for 320 Road.
    Timer = 2

    ARRIVING CARS:
        Car[019] entered Route 216, going FORWARD in MIDDLE lane.
        Car[020] entered Route 216, going BACKWARD in RIGHT lane.
        Car[021] entered 320 Road, going BACKWARD in MIDDLE lane.

    PASSING CARS:
        Car[014] passes through. Wait time of 2.
        Car[015] passes through. Wait time of 2.
        Car[021] passes through. Wait time of 0.
        Car[018] passes through. Wait time of 1.


    Route 216:
                            FORWARD               BACKWARD
    ================================      ================================
                            [016] [L] x   x [R] [020]
    --------------------------------      --------------------------------
                        [019][013] [M] x   x [M] [017]
    --------------------------------      --------------------------------
                                   [R] x   x [L]
    ================================      ================================

    320 Road:
                             FORWARD               BACKWARD
    ================================      ================================
                             [007] [L] x     [R]
    --------------------------------      --------------------------------
                                   [M]       [M]
    --------------------------------      --------------------------------
                                   [R]     x [L]
    ================================      ================================


    STATISTICS:
        Cars currently waiting:  6 cars
        Total cars passed:       15 cars
        Total wait time:         20 turns
        Average wait time:       1.33 turns

################################################################################

Time Step: 7
```

```
Left Arrow for 320 Road.
Timer = 1

Cars no longer arriving. // Simulation time is up.

ARRIVING CARS:
    // No more cars may arrive.

PASSING CARS:
    Car[007] passes through. Wait time of 5.


Route 216:
                          FORWARD                BACKWARD
================================    ================================
                            [016] [L] x   x [R] [020]
--------------------------------    --------------------------------
                        [019][013] [M] x   x [M] [017]
--------------------------------    --------------------------------
                                   [R] x   x [L]
================================    ================================

320 Road:
                          FORWARD                BACKWARD
================================    ================================
                                   [L]     x [R]
--------------------------------    --------------------------------
                                   [M] x   x [M]
--------------------------------    --------------------------------
                                   [R] x     [L]
================================    ================================


    STATISTICS:
        Cars currently waiting:  5 cars
        Total cars passed:       16 cars
        Total wait time:         25 turns
        Average wait time:       1.56 turns

##############################################################################

Time Step: 8

    Green Light for Route 216.
    Timer = 4

    Cars no longer arriving.

    ARRIVING CARS:

    PASSING CARS:
        Car[013] passes through. Wait time of 4.
        Car[020] passes through. Wait time of 2.
        Car[017] passes through. Wait time of 3.
```

Route 216:

| FORWARD | | | BACKWARD |
|---|---|---|---|
| [016] | [L] x | [R] | |
| [019] | [M] | [M] | |
| | [R] | x [L] | |

320 Road:

| FORWARD | | | BACKWARD |
|---|---|---|---|
| | [L] x | x [R] | |
| | [M] x | x [M] | |
| | [R] x | x [L] | |

STATISTICS:
    Cars currently waiting:  2 cars
    Total cars passed:       19 cars
    Total wait time:         34 turns
    Average wait time:       1.79 turns

################################################################################

Time Step: 9

    Green Light for Route 216.
    Timer = 3

    Cars no longer arriving.

    ARRIVING CARS:

    PASSING CARS:
        Car[019] passes through. Wait time of 3.

    Route 216:

| FORWARD | | | BACKWARD |
|---|---|---|---|
| [016] | [L] x | [R] | |
| | [M] | [M] | |
| | [R] | x [L] | |

    320 Road:

| FORWARD | BACKWARD |
|---|---|

```
    ==============================                ==============================
                                  [L] x   x [R]
    ------------------------------                ------------------------------
                                  [M] x   x [M]
    ------------------------------                ------------------------------
                                  [R] x   x [L]
    ==============================                ==============================


    STATISTICS:
        Cars currently waiting:  1 cars
        Total cars passed:       20 cars
        Total wait time:         37 turns
        Average wait time:       1.85 turns

################################################################################

Time Step: 9

    Left Arrow for Route 216.  // Light preempted - no cars in middle or right lanes.
    Timer = 2

    Cars no longer arriving.

    ARRIVING CARS:

    PASSING CARS:
        Car[016] passes through. Wait time of 4.


Route 216:
                          FORWARD                BACKWARD
==============================                ==============================
                                  [L]     x [R]
------------------------------                ------------------------------
                                  [M] x   x [M]
------------------------------                ------------------------------
                                  [R] x     [L]
==============================                ==============================

320 Road:
                          FORWARD                BACKWARD
==============================                ==============================
                                  [L] x   x [R]
------------------------------                ------------------------------
                                  [M] x   x [M]
------------------------------                ------------------------------
                                  [R] x   x [L]
==============================                ==============================


STATISTICS:
    Cars currently waiting:  0 cars
    Total cars passed:       21 cars
    Total wait time:         41 turns
```

```
        Average wait time:       1.95 turns


################################################################################
################################################################################
################################################################################

SIMULATION SUMMARY:

    Total Time:           9 steps
    Total vehicles:       21 vehicles
    Longest wait time:    5 turns
    Total wait time:      41 turns
    Average wait time:    1.95 turns

End simulation.
```

---