CS116-04 Project

4 person teams, one submission in Blackboard per team
Design due in Blackboard Friday, midnight of week 13 (April 14 11:59 PM)
Code due in Blackboard Friday, midnight of week 15 (April 28 11:59 PM)

Objectives:

1. (20 points) Design a multi-object application.
2. (80 points) Code a multi-object application.

Toll Booth Simulation - The overall goal of the project is to be able to compare different Toll Booth configurations for 6 lanes total of two Toll Booth types:

- Manual Toll Booths - Person operated, Vehicle must come to a complete stop. The length of the delay is estimated to be related to the numbr of wheels for the Vehicle (from 4 for cars to 18 for large truck), one second delay per wheel.
- Automatic Toll Booths- Transponder in Vehicle is "read" by monitor, Vehicle must slow down to 10mph. The length of the delay is on average 1 second for all Vehicles.

Assume a Vehicle will pick the shortest Toll Booth Line for their Toll Booth type (Manual or Automatic) and a Vehicle does not change Toll Booth Lines. There is a maximum length of a Toll Booth Line of 25 Vehicles. The Simulator will allow the user to configure up to 6 total Toll Booths in any combination of Manual and Automatic (with at least one of each). We will be simulating using Vehicle and arrival data collected at two different times of day, rushhour.txt and nonrushhour.txt. Files contain comma delimited Vehicle arrive time, Vehicle toll type ("M" or "A"), and Vehicle wheel count (if "M"), one per line, and are sorted in increasing Vehicle arrive time order. More than one Vehicle can arrive at the same time. GenerateInputFiles.zip contains code to generate more random input files. You may need to try different average inter-arrival times (final int ARRIVAL_MEAN=1; // 1 for rush hour, 5 for non rush hour).

Comparisons will be made using the following statistics (you can add other statistics, but these are required):

1. The maximum length of each of the Toll Booth Lines during the entire simulation.
2. The maximum wait time for each Vehicle of a Toll Booth type (Manual or Automatic) during the entire simulation (wait time is from when a Vehicle arrives until when the Vehicle reaches the Toll Booth).
3. The average wait time for each Vehicle of a Toll Booth type (Manual or Automatic) during the entire simulation (wait time is from when a Vehicle arrives until when the Vehicle reaches the Toll Booth).

You will need to define Vehicle classes (use inheritance or abstract) for both manual and automatic cars. In addition to the data about Vehicles from the input file your classes may have additional attributes that your simulation populates to help calculate the simulation Vehicle statistics.

You will need to define a TollBoothLine class, which should be like a queue, allowing Vehicles to be added to the end of the line and removed from the start of the line. Since one of the required statistics is the max length of each Toll Booth Line it might be good to make an attribute for this.

I suggest a DoneVehicles collection class where you store Vehicles after they leave their TollBoothLine (with all their additional attributes popluated by the simulation). When the simulation is done you can process this data to calculate statistics 2 and 3 above.

The basic idea is the void main() Simulator class will run a simulation loop for a certain number of seconds. Different events can happen during each second of the simulation. Vehicle objects are created one at a time from the input file, and if it is the arrival time for that Vehicle, the Vehicle is placed in the shortest appropriate (by toll type) Toll Booth Line. When a Vehicle gets to the front of it's Toll Booth Line, the Vehicle leave time is calculated based on an observed delay for the toll type per axle. Once a Vehicle leaves, the next Vehicle in the Toll Booth Line moves up to the Toll Booth. When a Vehicle is DONE (and its booth time and leave time has been updated), it is moved to an array to save all Vehicles for calculating statistics after the simulation.

The Simulator class needs to do the following:

- ask user for "rush hour" or "non rush hour" simulation
- ask user for number of manual toll booths and number of automatic toll booths (limited to 6 total, with at least one of each)
- instantiate the correct emptyTollBoothLine objects, or a collection of TollBoothLine objects
- run the discrete event simulator (see below)
- calculate and output the Toll Line Statistics
- calculate and output the DONE Vehicle Statistics

Sample Run #1 (showing console input/output)

```
Input File: rushhour.txt
Manual Toll Booths: 4
Automatic Toll Booths: 2

Manual Line #1 Maximum Length=11
Manual Line #2 Maximum Length=10
Manual Line #3 Maximum Length=10
Manual Line #4 Maximum Length=10
Automatic Line #1 Maximum Length=11
Automatic Line #2 Maximum Length=11
Max Manual Wait = 77
Max Automatic Wait = 9
Avg Manual Wait = 10.383805668016194
Avg Auto Wait = 0.9962915749217754
```

Sample Run #2 (showing console input/output)

```
Input File: nonrushhour.txt
Manual Toll Booths: 1
Automatic Toll Booths: 5

Manual Line #1 Maximum Length=6
Automatic Line #1 Maximum Length=1
Automatic Line #2 Maximum Length=1
Automatic Line #3 Maximum Length=1
Automatic Line #4 Maximum Length=1
Automatic Line #5 Maximum Length=1
Max Manual Wait = 30
Max Automatic Wait = 0
Avg Manual Wait = 3.5808510638297872
Avg Auto Wait = 0.0
```

Design Requirements - Create the following for the above requirements (see the UML Design Reference for help):

1. (2 points) Create a list of nouns and noun phrases from the above Requirements Document.
2. (2 points) Create Class Diagrams only for the nouns and noun phrases that have significance.
3. (3 points) Show the associations between the classes and any classes that are compositions of other classes.
4. (5 points) Add Attributes to the Class Diagrams created earlier.
5. (5 points) Add Operations to the Class Diagrams created earlier.
6. (3 points) Create test cases for each class.

Coding Requirements

1. (70 points) Code and test your Toll Booth Simulation. Each class needs its own individual test program.
2. (10 points) Analysis (1 page) - Please describe what you think is the opitmal configuration of Manual vs. Automatic Toll Booths (6 total) for the input files provided and why. If there were unlimited number of Toll Booths, how many would you recoommend considering a possible doubling of Vehicles over the next 10 years.What assumptions has our model made?

****

Discrete Event Simulator A Discrete Event Simulator is using a computer program (basically a loop counting some time intervals, minutes for example) to simulate time moving forward and random arrival and handling of some sort of event. So every iteration of the loop is equal to one time interval, and we will randomly generate the time the next event will occur. When we reach that time, we will handle the event (here just count it and output it) and generate the time for the next event. A random exponential integer will tell us how many minutes til the next event, the interval between events time (this time can be zero!).

A basic discrete event simulator will do the following:

- Query, verify, and get from the user, from the console, how many positive integer minutes long should the simulation run be. Use hasNextInt() to verify integer, and check for > 0 too.
- Initialize the current minute to zero, and call RandomData for an exponentially distributed random integer with meanArrival=5 (set a constant for this). Save this random time as what time the first event is.
- Start your simulation loop to count minutes by 1 until the user-enetered simulation length
  - While the current time is the time for an event
    - count the event
    - output the time and the event number
    - call RandomData again for how many minutes until the next event. Add this to the current time to save what time the next event is.
- When the simulation loop is done, output how many events occured

Your Toll Booth event simulator is different because you are reading events (Vehicle arrivals) from an input file instead of generating inter-arrival time. Also you need to do something with the Vehicles you create instead of just writing out the events.

```
public class RandomData {
        public static int getInt(char type, int x, int y) {
                int num;
                switch (type) {
                case 'U': case 'u':
                        num = (int)(x + (Math.random()*(y+1-x)));
                        break;
                case 'E': case 'e':
                        num = (int)(-1*x*Math.log(Math.random()));
                        break;
                case 'N': case 'n':
                        num = (int)( x +
                (y * Math.cos(2 * Math.PI * Math.random()) *
                Math.sqrt(-2 * Math.log(Math.random()))));
                        break;
                default:
                        num = (int)(x + (Math.random()*(y+1-x)));
                }
                return num;
        }
}

import java.util.Scanner;
public class EventSimulator {
        public static void main(String[] args) {
                final int ARRIVAL_MEAN=5;
                int simulationLength=0, minute=0, nextArrivalTime, callCount=0;
                Scanner input = new Scanner(System.in);
                while (simulationLength<=0) {
                        System.out.print("How many minutes long is the simulation? ");
                        while (!input.hasNextInt()) {
                                input.next();
                                System.out.print("Please enter an integer: ");
                        }
                        simulationLength = input.nextInt();
                }

                nextArrivalTime = minute + RandomData.getInt('E', ARRIVAL_MEAN, 0);
```

```
                while(minute<=simulationLength) {
                        while ((minute == nextArrivalTime) && (minute<=simulationLength)) {
                                callCount++;
                                System.out.println("Minute:"+minute+" Event#"+callCount);
                                nextArrivalTime=minute+RandomData.getInt('E', ARRIVAL_MEAN, 0);
                        }
                        minute++;
                }
                System.out.println("Number of events = " + callCount);
        }
}
```