# Lab Assignment 6 D0010E 2025

## 1. Background

Your task is to write a program that can be used to perform discrete event-driven simulations. Using this program, and some additional code you have to write, you should then simulate a car wash. A large part of the instruction is about the second half but it starts with a general description of the simulator itself.

### 2. A General Discrete Event-Driven Simulator

A discrete event-driven simulator is a program that makes it possible to study how a state S describing a setting to be simulated changes over time as events occur. It starts with an initial state and a start event, where the event is the sole first member of a set of future events the simulator maintains during a run. It is appropriate to represent the set of future events with a priority queue where the priority is the time the events will occur. This is because we need to access future events in the order that they occur. The state should always contain a flag that signals the end of the simulation. Note that since this is a simulation, time is virtual and will be computed using formulas based on assumptions and involving randomness.

Events are implemented as Java classes that have a method that can be called to make them occur. This method typically changes the state S and generates new, future, events that are inserted into the set of future events. When a new event is created, the time in the future when it is to occur has to be computed. This is done in the event.

The actual simulator contains a main loop that repeatedly extracts events from the set of events and makes them occur. The simulation stops when the set is empty or the flag in the state is set.

Note that the simulator is *independent* of the particular simulation it performs and that this has implications on the design. No part of the design of the simulator itself must depend on details of a particular simulation (a simulation of a car wash, for instance). Also, the design should take into account that it should be possible to have two or more simulations running (with separate states) at the same time.

### 3. Simulating a Car Wash

Based on the general simulator program you should then add specific pieces so that it is possible to simulate what happens at a car wash under various circumstances.

The simulated system (the car wash, that is) has a queueing system, $k_1$ fast car-washing machines, and $k_2$ slow car-washing machines. When a customer with a car arrives, and a car washing machine is free, the car is immediately served. Otherwise, the customer has to wait in line with the car in the driveway that has room for at most $k_3$ cars. When a car-washing machine becomes free, the next customer in line is served. If both fast and slow car-washing machines are available, the fast machine is chosen. If the driveway is full of waiting cars, arriving customers will not be able to wait in line, and will therefore go looking for another car wash.

#### 3.1 Time

To be able to simulate this system we need to model when new customers arrive and how long it takes to wash their cars. We assume that all fast machines are equivalent, as are all the slow machines.

In real life the distribution of arrivals depends on time of day, day of the week, and many other things. In our simulation we will assume that the expected value for the number of arrivals during a time interval is

directly proportional to the length of the intervals, and that any two arrivals are two independent events. Statistics tells us that the time between two consecutive arrivals is exponentially distributed with the expected value y where 1/y is the expected value for the number of arrivals per time unit. The time it takes to wash a car could be expected to be normally distributed, but in this experiment we assume that it is uniformly distributed on the interval [a,b]. The only reason for this is that we want to avoid negative times.

It could be interesting to use a simulation like this to calculate the mean queueing time for the cars, how many cars that are denied service when the wait line is full, and the total amount of time that the car-wash machines are idle. To design and interpret such simulations professionally, knowledge in statistics and queueing theory is required.

#### 3.2 Random number generators

The simulation requires some sort of random numbers for arrival times and serving times. It is common to use a simple arithmetic formula to generate a sequence of "random" numbers. Since a formula is being used, the numbers are not random at all except possibly the first number in the sequence that can be chosen depending on the current time. The formula that is normally used is

$$r_{i+1} = (a*r_i + b) \bmod m$$

where a, b, and m are numbers chosen so that the sequence $r_0, r_1, ...$ goes through all integers in the interval [0, m-1] in a way that appears random. That they actually aren't random can be seen by the fact that every other number is odd and every other number is even. It is therefore not possible to use the last binary number in each number to get a random stream of ones and zeroes. The number $r_0$ is called a seed of the sequence. A class representing a random number generator should have two constructors: One that takes an argument as a seed and one that uses the time of the method call as a seed. With a given seed you get a simulation that can be reproduced.

The class java.util.Random is based on such a random number generator with a = 25214903917, b = 11 and $m = 2^{48}$, where the last 16 bits are not returned. Java classes, based on random number generators, for exponential and uniform distributions are available where this instruction can be downloaded. Use these.

#### 3.3 Events

The simulator should be driven by events. The events that can occur are that a car arrives to the system and that a car leaves the system when it has been washed. At these events the *state* of the system should change. The state is described by a time, the number of available car-wash machines, and a queue of customers that waits to be served. An arriving customer should be placed in the queue if all the machines are busy. (If the queue is too long, the customer is rejected.) Otherwise he should be served immediately which means that the number of busy machines of that type (fast or slow) should increase by 1. Using the random number generator that produce serving times it is possible to calculate when the customer will leave the system. This future event should be added to the set of events maintained by the simulator.

When a customer leaves the system there are two options. If the waiting line is empty, the number of available machines should be increased by 1. Otherwise the car that is first in line should be served, and the event representing that car leaving the system should be added to the set of events.

It is possible to generate all event times times before the simulation is started, but it is more rational to generate future events as late as possible during the simulation. This means that the program needs less memory without being less effcient. Implement your program in this more efficient way.

To stop the simulation, you use a stop event. Once the stop event occurs, the simulation should stop immediately even if there are still events waiting in the queue.

The number of available washing machines and the current time should be represented with primitive variables. Remember to distinguish between the fast and the slow machines! The only times that are interesting are those when an event occur. This information should be contained in the priority queue. It can therefore be appropriate to keep track of the time in connection with the priority queue, and have the current time be the same as the time of the last event.

To be able to identify individual cars in the program they should be numbered sequentially from 0. The number assigned to a car should be unique. The line of waiting cars in the driveway should of course be represented with a Queue class.

**4. Output**

During the simulation of the car wash the program should report all events and print the mean queueing time, the number of cars denied service and the total time that the machines were idle. (Note that the simulator itself must not be aware of these outputs as simulations in general need not produce such output.) The result should be presented in a table where each row contains information of the event that occured, the time of the event, the number of the car that the event concerns, and the type of car-wash machine it concerns (when relevant). It should be clear if cars that are served come from the queue or if they are served directly upon arrival. Cars that are denied service altogether should of course also be recorded. When a customer is being served the time the serving will take should be shown. An example of an output can be found below in Section 5.

Formatting text output can be done with the `java.util.Formatter` class.

**5. Example**

This is an example of output from a simulation using the following parameters:

`ExponentialRandomStream(2,1234)` for car arrivals,
`UniformRandomStream(2.8,4.6,1234)` for fast machines, and
`UniformRandomStream(3.5,6.7,1234)` for slow machines.

The number of car-washing machines was 2 of each kind, and the stop time was after 15 time units of simulation time. The queue had room for at most 5 cars at a time.

You are not allowed to write the simulator so that it only works with these settings, it should be easy to run the simulation with different settings. Your output should have the same columns and be structured like in the example below.

The output follows:

```
Fast machines: 2
Slow machines: 2
Fast distribution: (2.8, 4.6)
Slow distribution: (3.5, 6.7)
Exponential distribution with lambda = 2.0
seed = 1234
Max Queue size: 5
----------------------------------------
      Time      Event    Id    Fast    Slow    IdleTime  QueueTime  QueueSize   Rejected
      0,00      Start
      0,22      Arrive    0     2       2       0,87      0,00          0          0
      0,24      Arrive    1     1       2       0,95      0,00          0          0
      0,32      Arrive    2     0       2       1,10      0,00          0          0
      0,71      Arrive    3     0       1       1,49      0,00          0          0
      1,26      Arrive    4     0       0       1,49      0,00          0          0
```

```
 2,05      Arrive    5     0       0       1,49      0,80          1          0
 2,58      Arrive    6     0       0       1,49      1,85          2          0
 2,82      Arrive    7     0       0       1,49      2,58          3          0
 2,85      Arrive    8     0       0       1,49      2,69          4          0
 4,18      Leave     0     0       0       1,49      9,35          5          0
 4,76      Leave     1     0       0       1,49     11,65          4          0
 4,92      Arrive    9     0       0       1,49     12,14          3          0
 5,37      Arrive   10     0       0       1,49     13,94          4          0
 5,89      Leave     2     0       0       1,49     16,54          5          0
 6,70      Arrive   11     0       0       1,49     19,78          4          0
 7,25      Leave     3     0       0       1,49     22,55          5          0
 8,20      Arrive   12     0       0       1,49     26,35          4          0
 8,38      Leave     5     0       0       1,49     27,24          5          0
 8,53      Leave     4     0       0       1,49     27,82          4          0
 8,92      Arrive   13     0       0       1,49     28,99          3          0
 9,05      Arrive   14     0       0       1,49     29,54          4          0
 9,72      Arrive   15     0       0       1,49     32,88          5          0
10,33      Arrive   16     0       0       1,49     35,95          5          1
10,80      Arrive   17     0       0       1,49     38,28          5          2
10,98      Arrive   18     0       0       1,49     39,15          5          3
11,69      Leave     9     0       0       1,49     42,73          5          4
11,78      Leave     8     0       0       1,49     43,10          4          4
11,94      Arrive   19     0       0       1,49     43,56          3          4
12,13      Leave     6     0       0       1,49     44,35          4          4
12,22      Leave     7     0       0       1,49     44,61          3          4
12,43      Arrive   20     0       0       1,49     45,03          2          4
14,06      Arrive   21     0       0       1,49     49,92          3          4
14,39      Arrive   22     0       0       1,49     51,22          4          4
14,44      Arrive   23     0       0       1,49     51,49          5          4
14,58      Arrive   24     0       0       1,49     52,19          5          5
15,00      Stop              0       0       1,49     54,29          5          6
--------------------------------
Total idle machine time:  1,49
Total queueing time:     54,29
Mean queueing time:       2,86
Rejected cars:            6
```

Note that the mean queueing time is defined as the total queueing time divided by the number of cars that entered the system. Cars that were rejected because of too long queue are not considered to have entered the system.

### 6. Notes and Restrictions

1. As mentioned, use the given random generators for exponential and uniform distributions to compute different times. You are not allowed to change them but you are allowed to extend them, if you like, to get the functionality you need.
2. **Be careful when designing the simulator.** The simulator should be as general as possible, and it should be easy to add new events or to simulate other systems than the one we have described.
3. Consider that there can be different types of events, but they are all events, and have a lot in common.
4. Let the functionality associated with a class exist in that particular class.
5. Do not export any more functionality from a class than absolutely necessary. Encapsulate data and their methods in classes. Hide away implementation details. Remember that everything is `private` until you can give a good motivation for why it should be accessable in some other way.
6. Your program (classes) should be written in conformance with Java's Code Conventions and be well commented. You should also generate web pages based on **javadoc**-comments.