Assignment Taxi

Object Orientation

Spring 2025

1 Goals

After doing this exercise you should be able to:

- Recognize and construct active classes.
- Synchronize concurrent tasks.
- Use an `Executor`.
- Make a concurrent simulation.

It is not required to hand-in the diagrams used, or any other intermediate steps of this approach.

2 Train and Taxi Simulator

In this exercise you will simulate people arriving by train at a railway station and leaving the station by taxi. A sequential simulator is given on Brightspace as a starting point. The simulator uses simple classes to focus on the simulation process. The given simulation contains a single train transporting a random number of people to the station. In the given simulation a train arrives when there are no more people waiting at the station. These people must be picked up by taxis. In the given simulator four taxis are available for the transport. Two with a capacity of four people and two with a capacity of seven people. The simulation stops when the train has arrived ten times and all passengers have been transported by taxis.

Design and Implementation of the Sequential Version

The class diagram in Figure 1 shows the classes used in the given simulator.

Method `step` of the class `Simulation` advances the simulation by one step. A step is defined as follows.

- If there are passengers waiting at the station, the next taxi transports a number of persons from the station.
- If there are no passengers left, the next train brings new passengers.
- The last train closes the station.

The other classes should be self-explanatory.

A possible run of this program reads:

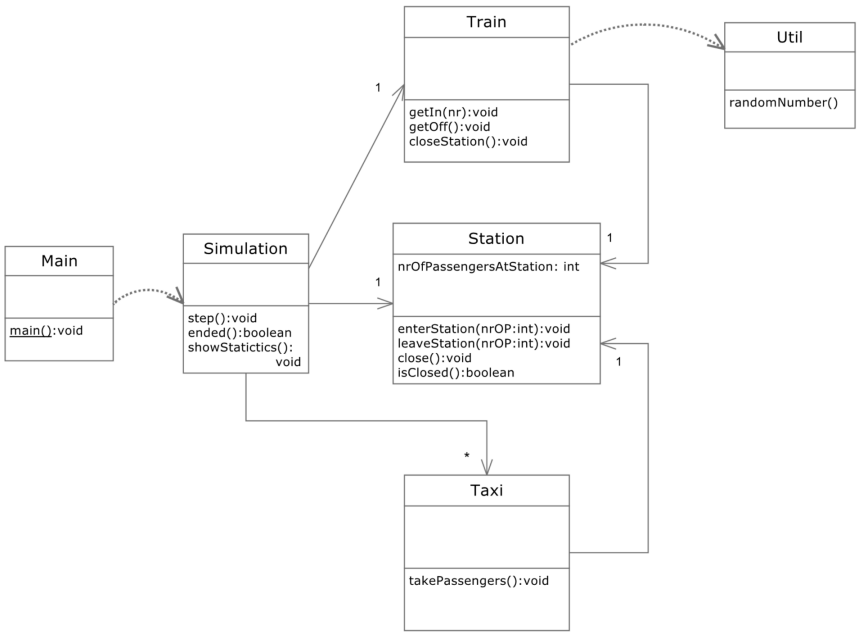

Figure 1: Class Diagram of the sequential version.

```
Taxi 1 created
Taxi 2 created
Taxi 3 created
Taxi 4 created
86 passengers arrived at station
Taxi 1 takes 4 passengers
Taxi 2 takes 4 passengers
Taxi 3 takes 7 passengers
Taxi 4 takes 7 passengers
Taxi 1 takes 4 passengers
...
...
...
Taxi 3 takes 7 passengers
Taxi 4 takes 1 passengers
All persons have been transported
Total transport time in this simulation: 370
Total number of train travelers: 785
Total number of persons transported in this simulation:785
```

3 Your Tasks

Your task is to replace the sequential simulation by a concurrent one, where trains and taxis are threads that bring and take people.

1. Transform `Taxi` and `Train` in active classes and add the required synchronization. You should add some delays to simulate the time trains and taxis need to transport people. Hint: The important variable is the counter of people waiting at the station. All accesses to this variable should be synchronized with the same lock. Always use the lock explicitly. Do **not** use `synchronized`.
2. Make a simulator where the active objects are executed concurrently. Every taxi and train has one thread that they use for the entire duration of the program. Do **not** make a new `Thread` for every **ride** of a taxi or a train.
3. You should use two `Conditions`: One that indicates that there are passengers, and one that there are no passengers. The taxis should only take passengers when there are passengers. The train should only bring new passengers when there are no more passengers. Hint: you need exactly one lock and two conditions in the whole program. Hint: all synchronization should take place in the methods of Station. Taxis and Trains should not use synchronization directly.
4. Remove the functions `Simulation.step()` and `Simulation.ended()`. The function `Simulation.start()` should start all threads and then wait for their termination.
5. After the simulation is done there has to be similar statistics as above. Check these numbers to see that there are no people lost or duplicated in your simulation.
6. Use an `Executor` instead of creating threads manually for the execution of the tasks. This makes it much easier to see when the simulation is finished, or if the simulation takes too long. The active class definitions of `Taxi` and `Train` do not care in which way their task is started.

3.1 Submit Your Project

To submit your project, follow these steps.

1. Find the folder that contains your assignment. In Visual Studio Code, you can find this by going to: File → Open Folder. Add the correct folder to a zip file. At the root level, your zip file should contain only one folder, e.g. `assignment-student-start`, which contain the entire project, i.e. the `app` folder and the Gradle files. *Do not submit only the `.java` files or the `src` folder!*
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.
3. **Do not submit any other format.** Do not submit .rar or .tar.gz or .7z files. Only zip files are allowed.