Project “Object-Oriented Programming”
Part 1 of 3

Prof. Eric Steegmans

Academic Year 2018-2019

1 Introduction

This text describes the first part of the project for the course ‘Object-Oriented Programming’. The project acts as the exam for the course. Typically, the project is carried out in teams of 2 students. If it is impossible to form a team, it is allowed to do the project individually. In that case, a slightly reduced portion of the assignment needs to be worked out. Note that this first part is still identical for teams and for students that work individually. This will no longer be the case in the second and third parts of the assignment. When, during the project, conflicts between team members arise that make further cooperation impossible, each team member must work out the project individually. It is allowed to look for a different partner after the first part of the assignment. These kinds of changes need to be reported immediately to `Eric.Steegmans@cs.kuleuven.be`.

In this project, part of the core of a *navigation application* is developed. All graphical aspects concerning the user-interface of the application are left out. The project strictly focuses on a subset of the elements that are found in the core of the application. The assignment describes a number of requirements on how these elements should be supported in the envisioned system. In the first place, a set of functional requirements is specified. These functional requirements are high level descriptions of the expected functionality of the system. We expect you to work out all basic functionality (such as getters and setters for properties) that is required to realize the functional requirements. Secondly, at some points in the assignment, a number of non-functional requirements concerning the qualities of the system (such as efficiency, reusability and adaptability) are described.

2 Assignment

The first part of the assignment focuses on one single class, namely the class *Road*. A set of interconnected roads is the essential requisite of a navigation application. Hence, the Road class will serve as a basis for all following parts of the assignment.

2.1 Road

A road is a connection between two locations, that can be traversed by the user of the application (e.g. a street, a motorway). Several of these roads might later be combined to form a longer route, which will take the user from a starting point to the desired destination point. Roads are characterized by the following properties.

2.1.1 Identification

Each road has a unique identification, meaning that no two roads can have the same identification. The identification of each road starts with a capital letter and is followed by 1 or 2 digits. Examples of legal road identifications are `E14`, `A2` and `X99`. Identifications of roads will always have this structure, be it that more and/or other characters might be allowed after the starting upper case letter in the future. he identification of a road can change in time, but still needs to be unique. In this first version, you may safely assume that no more than 100 roads will be created.

All aspects concerning the identification of roads must be worked out in a defensive way. *Tip: Have a look at the class `Character` in the Java API for useful methods on individual characters.*

2.1.2 End points

Every road spans in between two separate end points, lying on the surface of the earth. Once a road has been constructed, its end points can no longer change. Each end point is described with two coordinates (latitude followed by longitude) in order to express its geographical location. The coordinates are represented by *decimal degrees*, using two double precision floating-point numbers (Java‘s *double* type). The area in which the coordinates are located is limited to the positive quadrant and do not exceed a maximum of 70 degrees. It should not be possible to change this maximum value of the latitude and longitude during the run-time of the application. However, it must be possible to change the maximum in future versions of the application without having to change any other classes. The maximum value always applies to all coordinates of any road.

The methods related to coordinates are worked out nominally.

2.1.3 Length

Each road has a length, which stands for the travelling distance between its end points. The length of a road is always expressed in meters. Its value is a positive integer number. You may assume that the length of each road is at all times much smaller than Java's largest value of type `int`. The length of a road may change during its lifetime.

All methods concerning the length must be worked out totally.

2.1.4 Speed

Different roads can be traversed at different speeds. For each road there is a speed limit that should not be exceeded when traversing it. Speed limits are always represented in *meters per second*, as a floating point number (Java's `float` type). If there is no explicit speed limit, a limit of `19.5 m/s` applies. A road's speed limit is always a positive number, that can both increment and decrement over time. However, the speed limit can never exceed the speed of light (`299792458.0 m/s`).

Each road also has an average speed that is obtained while traversing it under standard conditions. It can be taken into account later to determine optimal trajectories. As for the speed limit, the average speed is represented in *meters per second*, as a floating point number (Java's `float` type). Its value is non-negative and cannot exceed the speed limit imposed on the road.

All methods related to speed are worked out defensively.

2.1.5 Current delay

Unfortunately, delays can occur on roads due to e.g., traffic jams or accidents. The navigation application is aware of this, and therefore, each road incorporates a property that indicates, in *seconds*, how much delay is currently experienced when traversing the road. he current delay is at least non-negative and is represented by a floating point number (Java's `float` type). Furthermore, a delay can be infinite, e.g. in case traffic is at a complete and indefinite standstill. The delay distinguishes between the directions in which a road can be traversed. The delay when traveling towards one of the end points can thus differ from the delay when travelling towards the other endpoint.

The methods related to delays are worked out nominally.

2.1.6 Blocked

As a result of roadworks, it may occur that a road is temporarily blocked. This information is vital in order to correctly navigate the user to its destination. A *boolean* property of the roads indicates whether or not it is currently blocked. As with the current delay property, there is a distinction made between the different directions of the road. In other words, the road can be blocked in one direction only or in both directions simultaneously.

2.2 Main Program

The functionality of the developed class of roads needs to be demonstrated by means of the following scenario:

1. Create three road objects using one of the constructors that you have developed. Make sure each road has one overlapping edge point with a previous road, implicitly forming a long route consisting out of three roads. Write the values of all properties of each road to the standard output stream.

2. Set a delay of 100s in the forward direction for the first road, and block the backward direction for that road.

3. Calculate the required cumulative time that is needed to traverse all three roads in the forward direction, taking into account the average speed and the current delays. Print this time span to the standard output stream.

4. Next, an accident occurs on the second road in the forward direction, causing traffic to come to an indefinite standstill. Change the delay of this road to infinity. Afterwards, verify that total cumulative travel time of the three roads is now infinite and write this to the standard output stream.

When you find it useful and/or necessary, you may write more details to the standard output stream. However, all write operations should be located in the main program.

2.3 Facade

We offer a large set of tests to verify the correctness of your class. In order to be able to execute those tests, you must implement all the methods in the interface `Facade`. The purpose of that facade is to build a bridge between your code and the tests. `Facade` includes some typical examples on how to implement each of its methods. Basically, you invoke the equivalent method as defined in your class, and catch any exceptions re-throwing them as exceptions of type `ModelException`.

Your class must not offer methods identical to those of the interface. There are definitely better ways to support each of the characteristics ascribed to roads.

3 Practical Guidelines

1. Only the imposed requirements need to be fulfilled. You will not make a better impression by developing extra elements that are not part of the assignment.

2. Details that are not explicitly specified or mentioned in the assignment may be worked out according to your own inspiration. When doing so, always go for the simplest solution.

3. The project needs to be worked out in Java 10 or higher.

4. When the assignment specifies that a particular property needs to be worked out nominally, totally or defensively, the manipulation of this property in more complex methods such as constructors also needs to be worked out accordingly.

5. All aspects of the class of roads must be documented both formally and informally.

3.1 Registration

You need to register your team, **even if you work alone**. You register by sending an email to `Eric.Steegmans@kuleuven.be` (with your partner in CC) that contains the name and study programme of each team member **before March 10 at 12:00**.

3.2 Handing in

1. The first part of the project must be handed in **by both students of each team before March 30 at 9:00 am**. Deviations from this deadline will only be allowed in exceptional cases.

2. Submitting your solution is done via Toledo. You have to submit your solution as a **JAR** file. Creating a JAR file in the Eclipse environment is performed via the 'Export' function that is found in the 'File' menu. Be sure to select the source files of all classes that you have developed. Mention your name and study programme in the comment field when submitting.

If certain aspects of this assignment are still unclear, further details will be posted on Toledo. In order to avoid contradictory answers, questions concerning the assignment can only be sent to `Eric.Steegmans@kuleuven.be`.

Good luck!