Project “Object-Oriented Programming”
Part 3 of 3

Prof. Eric Steegman

Academic Year 2018-2019

1 Introduction

This text describes the third part of the project of the course ‘Object-Oriented Programming’. The project is worked out in the same teams as the second part of the assignment and we no longer accept team changes. If serious problems between team members arise that would make further cooperation impossible, Prof. Steegmans needs to be notified as soon as possible and each team member must work out the rest of the project individually.

The third part of the assignment distinguishes between different types of roads. In addition, it is now possible to use (sub)routes as segments for other routes. The text actually describes the full assignment for the project, including all things that had to be worked out in part1 and in part 2. Some aspects of these previous parts change in this third part. Those changes are in red. New things to be worked out are in blue.

2 Assignment

2.1 Locations

As we have seen in Part 1, each road spans between two separate end points. In Part 2, these points are described more elaborately by their own class *Location*. Locations are characterized by the following properties.

Students working alone on the project must not work out locations. They may stick to end points as introduced in the assignment for the 1st part of the project.

2.1.1 Coordinates

Similar to the specification in Part 1, a location is described with two coordinates (latitude and 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). Both the latitude and the longitude must be finite numbers. They do not change once a location has been created.

The methods related to coordinates are worked out **nominally**.

2.1.2 Address

Every location has a particular address, which consists of at least two characters and can only contain letters, digits, commas and spaces. The address of each location must start with a capital letter. For example, *“Celestijnenlaan 200A, 3001 Heverlee”* is a valid address. It is possible that in future versions of the application the spelling rules will have to be extended to allow additional characters in addresses. However, the characters mentioned above will never be omitted as valid characters. The address of a location may change during its lifetime.

The methods related to addresses are worked out **totally**. **Tip**: Study the method `matches` in class `String`, and regular expressions used in it.

2.1.3 Adjoining Roads

Each location can have adjoining roads. These are all the roads for which the given location is one of the end points. At any point in time, a location can exist without any adjoining roads. Furthermore, new roads can be constructed and existing ones can be demolished. Moreover, locations themselves can also be destroyed.

All public methods regarding adjoining roads should be worked out **defensively**. The non-public methods are worked out **nominally**. Your class must offer at least a method to get all adjoining roads for a location, as well as a method to check whether a given road is one of the adjoining roads for a given location. That method must return its result in (nearly) constant time.

2.2 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 can be combined to form a longer route, which takes the user from a starting point to the desired destination point. Roads are characterized by the following properties.

2.2.1 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. The identification of a road can change in time, but still needs to be unique. There is no limit on the total number of roads. As soon as a road has been destroyed, its identification can be re-used for other roads.

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.*

2.2.2 End points

Every road spans in between two locations, lying on the surface of the earth. Once a road has been constructed, its end points can no longer change. Each location 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 of the end points of roads are located is limited to the positive quadrant and their latitude nor their longitude exceed a maximum of 70 degrees. It should not be possible to change this maximum value 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 point.

The methods related to coordinates are worked out **nominally**. Your class must include a method that returns both end points at the same time.

2.2.3 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**.

2.2.4 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**.

2.2.5 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**.

2.2.6 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.

2.2.7 Classification of Roads

In this final part of the assignment, we distinguish between different types of roads. More specifically, every road is either a *one-way road*, a *two-way road* or an *alternating road*. Every road must be of one of these three types.

1. **One-way Road** One-way roads are traversable in one direction only, i.e. only one of the two end points can act as the starting location for the road, while the other end point is the only valid end location. The role of each end point is fixed at the time one-way roads are constructed. Delays and blockages only apply to the direction in which a one-way road can be traversed. It will be an error to ask for delays or blockages on the non-supported direction.

2. **Two-way Road** Two-way roads are roads that can be traversed in both directions, i.e. both end points can act as starting locations and as end locations. Two-way roads more or less correspond to roads as introduced in the previous parts.

3. **Alternating Road** *Students working alone on the project must not work out alternating roads.* As for one-way roads, alternating roads are traversable in one direction only, i.e. only one of the two end points can act as the starting location for the road, while the other end point is the only valid end location. However, contrary to one-way roads, the role of each of the end points in an alternating road can change in time.

It must be possible to retrieve the set of all valid starting locations, as well as the set of all valid end locations of any kind of road. Notice that those sets will have one element each for one-way roads and for alternating roads, and two elements each for two-way roads.

2.3 Route

A route is a traversable trajectory that connects a start location with an end location. You are free to work out the methods for routes in a nominal way, in a total way or in a defensive way. You may also work out some methods (or parts of them) in one way, and other methods (or parts of them) in another way. Routes are characterized by the following properties.

2.3.1 Start Location

Each route has a start location. Once a route has been created, its start location cannot be changed anymore.

2.3.2 Segments

Each route has one or more segments that represent the different pathways that define the complete trajectory. A segment is ether one of the three types of roads described above or a (sub)route. In order for a route to be a valid interconnected chain of segments, one of the starting locations of each segment must be one of the end locations of the previous segment. For the first segment, one of its starting locations must be the start location of the route. The composition of routes must be acyclic, meaning that no route can have itself as a segment, nor can it have a segment that directly or indirectly has the given route as one of its segments.

For an existing route, segments can be added, removed or replaced. However, the set of segments resulting of these alterations must still form a valid route as described above. Changes to an existing route `R` may invalidate other routes that have the route `R` as a direct or indirect segment. Each time a method is applied to a route, you will verify that it still has a legal sequence of segments. Methods on routes will throw `IllegalStateException` if this is no longer the case.

Next to changing the composition of routes, it must be possible (1) to ask for the total length of a route, (2) to ask for the start location and the end location of a route, (3) to ask whether the route is traversable, i.e. whether all roads are unblocked in the direction to be taken, and (4) to ask for the sequence of locations visited by a route when traveled from the start location to the end location.

2.4 Main Program

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

1. Instantiate four distinct locations and create three roads that connect these locations. Each of the three roads must be of a different type (two-way, one-way, alternating). One of the roads is a narrow country road, where the average speed is only 8 m/s. At this point, none of these roads are blocked or have any delays.

2. Create an extra parallel two-way road between the locations that are already connected by the country road. This redundant road is a motorway. Make sure that it is longer, but faster than the country road, because of a higher average speed. For comparison, print the properties and travelling time of the two parallel roads to the standard output stream.

3. Create a route involving all four locations. The route must use the motorway to go from one of the locations to the other. Print the length of this route, and all visited locations on the standard output stream.

4. Block the motorway in the direction taken by the route. Show on the standard output stream that the given route is no longer traversable.

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.

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. All aspects of the class of *locations* must be documented only in a formal way. If you want to get a score of 16 or more, you must document all aspects of the class of *routes* in a formal way. If you do not want to go for such a high score, you must not work out any documentation for the class or routes.

6. In addition to the definition of all classes, you must also work out a UML diagram of the design of your navigation system. This diagram should contain all classes, except for self-defined exception classes and test classes. The diagram must contain the name of each class and all its attributes. The diagram must not contain any methods. You may choose any tool for making the diagram (a hand-drawn diagram is also allowed). You must submit this diagram (in PDF format) along with your code.
7. The major goal of part 3 is to work out a proper hierarchy of classes with as much code as possible at the level of non-leaf classes. In our solution, more than 50% of the code (documentation and implementation) is at the level of abstract classes.

3.1 Handing in

1. The third part of the project must be handed in **by both students of each team** before **May 23, at noon**. 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 all classes that you have developed (including test classes). Mention your name and study programme in the comment field when submitting.
3. The UML diagram must be submitted electronically.

3.2 Defense

1. The project needs to be defended in front of Prof. Eric Steegmans. The defenses will start on May 27. In the beginning of May, a list will be posted on Toledo that contains the possible dates at which the project can be defended. You use it to make a concrete appointment with Prof. Steegmans. More information on this will follow on Toledo.
2. You need to bring a printout of the UML diagram of your design to the defense. Without this diagram, the defense cannot take place! Clearly mention your name and study programme on the diagram.
3. Teams of two students defend their project together.

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

Good luck!