Homework 2 (HW2) - Interaction Modelling, Application Modelling, and Coding (24 points + BONUS)

Due on 20.11.2017 at 23:59
complete the homework in groups of 2 students.

SUBMISSIONS:

One of the members of the group has to login and submit the assignment using the link “submit” on the course webpage. Please specify in a comment the members of the group.

You have to submit a single .zip file containing:

- A pdf with the documentation of your application that includes:
  o domain model
  o use case
  o sequence diagram
  o application model
- A folder with the source code of your application where a main class plays the role of „boundary class“ (it is the class that invokes the method implementing the functionality that you have developed)

Grading criteria:

- Correct definition of use cases and sequence diagrams (in terms of correct use of the different elements and in terms of compatibility with the interfaces of the classes in the application: the role of each method defined in the interfaces of the classes should be clear from the use cases and sequence diagrams) (5 points)
- Use of the design patterns (at least 1 of the patterns you saw during the lectures has to be included) (3 points)
- Correct definition of the application model and correct use of the methodology that from the domain model allows you to obtain the application model through interaction modeling (3 points)
- Consistency between code and documentation (10 points)
- The application passes a small test (3 points)

Trip Planner - Domain Model [DONE]

Part 1

The following is a map of the four Tallinn tramway routes. A similar map could be drawn for trolleybus and for bus routes.
Task 1. [DONE] Draw a domain diagram for tramway routes such as the one shown above. Note that every tramway stop is associated to a GPS coordinate. Also, note you are not asked a model of the map itself, but only of the routes and location of stops. The domain model should also be able to cover trolleybus and bus routes.

Part 2

Below is a sample schedule of tramway line number 4. The schedule is given starting from one of the terminals of the route, namely Vana-Lõuna. There is a column missing in the picture corresponding to the “Sunday” schedule, which is similar to Saturday but with slightly different departure times. You can view the full schedule here: http://soiduplaan.tallinn.ee/#tram/4/a-b/en
Task 2. **[DONE]** Design a new domain model by adding all information required to generate the schedule of a given tramway line starting from a given stop (you can use the model from the previous question as a basis, but this should be submitted as a separate model). Please assume that the tramway takes a given amount of minutes (e.g. 1, 2 or 3 minutes) to go from a stop to the next one. The number of minutes it takes from the tramway to go from one stop to another is the same regardless of the week, but differs depending on the pair of stops being considered, e.g. between Lubja and Majaka it takes 1 minute, whereas between Majaka and Sikupilli it takes 2 minutes. Note that the domain model produced for this task does not need to represent the concept of “schedule” itself, but only the information required to generate a schedule.

Part 3

A trip planner is an application that allows a user to enter a departure point, a destination point, and a time of departure, and returns a plan for going from the departure to the destination via public transport. The departure and destination points may be an address, a name of a stop or a place of interest (e.g. National museum). If the string entered for the departure (or destination) is incorrect, the application returns the most similar points it knows about (e.g. similar addresses), and allows the user to select one of the proposed alternatives or to type in a new departure or destination address, until the departure and destination points are recognized. If both the departure and destination points are recognized, the trip planner returns a plan.

The plan returned by the trip planner consists of travel legs. Some travel legs involve walking from one point to another (e.g. from an address to a given stop or vice-versa), while other legs involve taking a tramway or bus from one stop to another.
For example, the following is a plan produced by the Berlin travel planner. In this homework, we do not consider anything associated with “fares” (cost of tickets).

![BVG](image)

**Results**

<table>
<thead>
<tr>
<th>Station / Stop</th>
<th>Time</th>
<th>Duration</th>
<th>Conn.</th>
<th>with</th>
<th>Fare*</th>
</tr>
</thead>
<tbody>
<tr>
<td>15738 Zeuthen, Nürnberg Str. 8</td>
<td>14:39 dep</td>
<td>1:01</td>
<td>1</td>
<td></td>
<td>3.20 €</td>
</tr>
<tr>
<td>U Karl-Marx-Str. (Berlin)</td>
<td>15:40 arr</td>
<td></td>
<td></td>
<td></td>
<td>2.30 €</td>
</tr>
</tbody>
</table>

**Figure 3. Plan produced by the Berlin trip planner from Nürnberg Str. 8 to Karl-Marx-Platz**

**Task 3. [DONE]** Design a new domain model for a trip planner (you can use the model from the previous question as a basis, but this should be submitted as a separate model). The model should capture all the information required to generate a plan, and it should capture the generated plan itself. You may assume that for every address or place of interest, the corresponding GPS coordinates are available. You may also assume that given two GPS coordinates, we can calculate the distance and the walking time between these two coordinates.

**Trip Planner - Functionalities**

**Part 4**

The trip planner should provide one of the following functionalities:

a. Given departure stop, arrival stop, date and time as inputs, the trip planner checks that there exists at least one direct route from the departure stop to the arrival stop on the input date and after the input time. If there is at least one direct route satisfying the input constraints, the trip planner returns the next bus available, the departing time of the bus (this must be later than the input time) and the arrival time. Otherwise, it returns an error message (e.g. “there are no direct routes from departure stop to arrival stop”).

   **NOTE**: you are not required to output a trip plan for this functionality. **[0.60]**

b. Given departure stop, arrival stop, date and time as inputs, the trip planner checks that there exists at least one trip plan from the departure stop to the arrival stop on the input date and after the input time. If there is at least one trip plan satisfying the input constraints, the trip planner returns one of the trip plan available. Otherwise, it returns
an error message. NOTE: the output trip plan must be a composition of travel legs (as in Figure 3). The output trip plan does not have to be optimal (i.e. the trip plan with the shortest travel time). [0.80]

c. Given departure stop, arrival stop, date and time as inputs, the trip planner checks that there exists at least one trip plan from the departure stop to the arrival stop on the input date and after the input time. If there is at least one trip plan satisfying the input constraints, the trip planner returns the optimal trip plan. Otherwise, it returns an error message. NOTE: the output should be composed of travel legs. In this case, we assume the optimal trip plan is the shortest trip plan, i.e. the trip-plan with the shortest travelling time (including waiting times). [1.00]

d. Given a pair of GPS coordinates (departure and arrival), date and time as inputs, the trip planner fetches for the optimal trip plan. NOTE: the GPS coordinates do not necessarily refer to bus stops. The output should be composed of travel legs. In this case, we assume the optimal trip plan is the shortest trip plan, i.e. the trip-plan with the shortest travelling time (including waiting times). [1.20]

The travelling time is the length of time between input time and estimated arrival time.

A sample trip planner can be found here: http://soiduplaan.tallinn.ee/#plan/

To fill your database, refer to the following dataset:

- http://www.peatus.ee/gtfs/gtfs.zip

Information about GTFS format can be found here: https://developers.google.com/transit/gtfs/reference

Task 4. Choose the functionality you want to provide (a, b, c, or d) and:

- Write the Use Case description and draw the sequence diagram of the functionality
- Starting from your domain model, draw the application model
- Develop a command line java application implementing the functionality. In the application, a main class plays the role of „boundary class“ (it is the class that invokes the method implementing the functionality that you have developed)

[Hints]

- The sequence diagrams should contain all the operations that you provide in the class interfaces. You can abstract away from methods that extract information from lists without going into implementative details for that
- The input stop can be given as stop ID or stop address or stop GPS coordinates.
- The functionalities can be implemented incrementally.

[Grading]

Functionality a is the simplest one, whilst functionality d is the most complex. Each functionality has a decimal coefficient (within squared brackets). The final score of this homework will be equal to the points earned for the Tasks 4 multiplied by the functionality
coefficient. I.e., if you complete Tasks 4 for functionality \( b \), your final score will be equal to the sum of the points earned in Tasks 4 multiplied by the coefficient 0.80 (e.g. \( 10 \times 0.80 = 8 \)).