Object Oriented Programming

CSE1100 – Mock Exam

A. Zaidman / T. Overklift / F. Mulder

Date: 20-10-2021 Duration: 2 hours Pages: 4

Allowed resources

During the exam you are **NOT** allowed to use books or other paper resources. The following resources can be used during this exam:

- PDF version of the lecture slides (located on the S-drive of the computer).
- A local (limited) version of Stack Overflow.
- The Java API documentation (located on the desktop of the computer).

Set up your computer

Log in using:

Username: `EWI-CSE1100`
Password: `MockExam!`

Once the environment loads you will be asked to provide your personal NetID and password combination. Entering these will give you access to a private disk ("P-drive") where you can store your assignment. Once this is done, please start IntelliJ and accept the licence agreement. While IntelliJ boots, take the time to read the entire assignment.

Rules

- You are not allowed to use the **internet**.
- **Mobile phones / smartwatches** are not allowed. You must turn them off *and* put them away in your coat / backpack.
- Backpacks remain **on the ground**, not on your desk.
- Attempts at **fraud**, or actual acts of fraud, will be punished by the Board of Examiners.

About the exam

- Your project (sources *and* tests) **must compile** to get a passing grade. Sometimes IntelliJ will allow you to run part of your code even when some classes fail to compile. We will still see this as a compilation failure. Ensure that **ALL classes you hand in compile.**
  - If your solution does not compile, you will fail the real exam. In this case you will get 0 bonus points.
- At the end of the exam there should be a **ZIP archive** on your "P-drive", named `5678901.zip`, where `5678901` is your own student number (the location of your ZIP file doesn't matter). You can find your student number on your student card (7 digits; below your picture).
- **Follow the submission instructions in the last part of the exam carefully**.
- **Stop writing code 5-10 minutes before the end** of the exam so you have time to make sure your submission compiles and to create the .zip file.
- The **grading** is explained on the last page of this exam.

Setting up the template project

- Open an explorer and copy the template project from the (read only) S-drive and paste it on your P-drive.
- Subsequently open the project on the P-drive in IntelliJ (you can open the project by opening the folder and double-clicking the `.iml` file, or via the open option in IntelliJ).
- Once you have opened the project you can start implementing your exam in this project!

Train application

ProRail is the company that is responsible for the railway infrastructure in the Netherlands. They are in dire need of a new software system. They want to keep track of which trains are on the tracks on a certain day to get more insight into the availability and usage of the railway tracks. ProRail has contacted you to make a first prototype of this application. ProRail is well aware that you can only produce a prototype in such a short amount of time, but they will still need some time to finish some of the extra requirements.

Specifically, ProRail wants you to develop an application that reads in a file that specifies which trains are present on the tracks. An example of such a file looks as follows:

```
TRAIN: INTERCITY
DEPARTS: 14.28
ARRIVES: 14.48
IC-STATION: Rotterdam-Centraal
IC-STATION: Schiedam-Centrum
STATION: Delft-Campus
IC-STATION: Delft
STATION: Rijswijk
STATION: Den-Haag-Moerwijk
IC-STATION: Den-Haag-HS
END
TRAIN: SPRINTER
DEPARTS: 14.32
ARRIVES: 15.00
IC-STATION: Rotterdam-Centraal
IC-STATION: Schiedam-Centrum
STATION: Delft-Campus
IC-STATION: Delft
STATION: Rijswijk
STATION: Den-Haag-Moerwijk
IC-STATION: Den-Haag-HS
END
```

The complete file **trains.txt** is available in the template project.

The order of the lines of a train specification is fixed. You will always first get the departure and arrival times and then a sequence of stations along the route of the train. Also important to know: each piece of information is always on a new line.

A **Sprinter** is characterised by:

- The time of departure
- The time of arrival
- A list of stations that are part of the route of this train. The train stops at each station on this route.

An **Intercity** is characterised by:

- The time of departure
- The time of arrival
- A list of stations that are part of the route of this train. There are IC-stations in this list and regular stations. This type of train stops at the begin/end station and all intermediate IC-stations, while normal stations on the route are just there to show the route (the train doesn't stop at these).

ProRail asks you to design and implement a program that:

- **Reads in** the file trains.txt and has classes and structures to store and represent the information in the file.
- Has an **equals()** method for each class (except for the class that contains the main() method). Two trains should be equal only if their type of train, route followed and departure and arrival times match.
- Is properly **unit tested** (at least 1 test per method, excluding the main method).
- To enable user interaction, please provide a **command line interface** (reading from `System.in` and writing to `System.out`). This interface should look like:

```
Please make your choice:
   1 – Show all trains that are in the in-memory database on screen.
   2 – Show the travel times of all train lines.
   3 – Show all trains that stop at a given station.
   4 – Stop the program
```

Option 1:
All trains are shown on screen in a human readable format. This can for example be:

```
Intercity from Den-Haag-HS to Rotterdam-Centraal
   Departure: 14:28
   Arrival: 14:48
    ---Den-Haag-HS
    ----(Den-Haag-Moerwijk)
    ----(Rijswijk)
    ---Delft
    ----(Delft-Zuid)
    ---Schiedam-Centrum
    ---Rotterdam
```

The output format doesn't have to be exactly the same, but it should show a distinction between stations that are on the route (e.g. Rijswijk) and a station where the train stops (e.g. Delft). Please take into consideration:

- An Intercity stops at the start and stop station and at IC-stations
- A Sprinter stops at all stations on the route

Option 2:

- Show the: train type, departure station, destination station and total travel time (in minutes) for all trains. For instance:
  `Intercity from Den-Haag-HS to Rotterdam-Centraal takes 20 minutes.`

Option 3:

- Ask the user to provide a station (via `System.in`).
- **Show all trains that stop** at the given station (you may print the train information in the same format as option 1).
- Implement this option using **functional programming only**. (So no for loops, but streams in combination with filters and/or maps).

Option 4:
The application stops.

Some important things to consider for this assignment

- The textual information should be read into a class containing the right attributes to store the data (so storing all information in a single `String` is not allowed, because this would hinder further development). With regard to the type of attributes used (`int`, `String`, or some other type): you decide!
- Think about the usefulness of applying inheritance.
- The **filename trains.txt should not be hardcoded** in your Java program. Please make sure to let the user provide it when starting the program (either as an explicit question to the user or as a program argument)
- Write unit tests!

Other things to consider

- The program should **compile**
- For a good grade, your program should also work well, without exceptions.
  Take care to have a nice **programming style**, in other words make use of code indentation, whitespaces, logical identifier names, etc. Also provide Javadoc comments.

Handing in your work

To hand in your work you must create a **ZIP** file containing your project files.

Go to your private P-drive and navigate to your project. You should see your project folder you copied before. Select this folder by left-clicking once, and the, **right**-click, hover "7Zip", and select "Add to 'Folder name'.**zip**" .

Important: Rename the ZIP file to your student number, e.g. "`4567890.zip`".

Double-check the correctness of the number using your campus card.
The location of your ZIP file doesn't matter, as long as it is in your P-drive.
Please ensure that there's only **one** ZIP file on your drive.

Which things will determine your grade?

For this mock exam you can get at most 0.5 bonus points. You can get these bonus points by queueing during one of the labs after the mock exam and asking a TA for feedback.
The TA will go over your submission, give feedback, and award any bonus points that you have earned. When you ask for feedback from a TA make sure you show them **a compiling solution. If your solution does not compile you will get 0 bonus.**
You can earn the following points:

- (0.1 point) A well designed class structure for the information in the file.
- (0.1 point) A working reader that can parse file input and store it & option 1 works.
- (0.1 point) Option 2 ***or*** option 3 works (option 3 must be implemented with functional programming).
- (0.1 point) At least 5 functional test cases.
- (0.1 point) Good code quality (e.g. indentation, method length, variable naming, Javadoc).