**DELFT UNIVERSITY OF TECHNOLOGY**
**Faculty of Electrical Engineering, Mathematics and Computer Science**

TU Delft

**CSE1100 Object-Oriented Programming (computer exam)**
**October 31st, 2018 09:00-12:00 (total duration: 3 hours)**

**Exam created by A. Zaidman and checked by F. Mulder**

---

You can make use of the following during this exam:

- One book on Java, e.g., Java in Two Semesters (Charatan & Kans) or Introduction to Java Programming and Data Structures (Lang).
- The slides are available in PDF format on the S-drive of your computer
- The Java API documentation (javadoc) is available on the S-drive of your computer
- The file you need to read in is also available on the S-drive

This exam contains **1 assignment (10 points)** (total exam: 6 pages).
→ **the detailed scoring roster is on page 6**

**Log in to the computer with the following credentials:**
**Username: EWI-CSE1100**
**Password: Welcome1100**

**After logging in with the above credentials, you will also be asked for your personal NetID/password combination. Enter those too (this will connect the P drive so that you can store your files).**

**HINT 1:** **Start Eclipse and while it starts up, read the <u>entire</u> assignment and only then start implementing**
**HINT 2:** **Look at the last page to get an overview of how your score is built up for this exam.**
**HINT 3:** **When creating a new project, do not create a "module-info.java" file as this will prevent you from working in the default package. (If you created it anyway, you should delete it.)**

A few hints:

- Your program must **compile** (fail to compile == fail this exam)
- When your program is finished, use the 7Zip program to zip your files. Specifically, make a **zip file of your src folder** (the folder that contains your .java files) when your assignment is ready. Give the zip file the following name <studentnumber>.zip, so for example 12121212.zip. Please also put the inputfile into this zip. The class files are not necessary.
- The **workspace directory of Eclipse** is located on the P drive. Use the Windows Explorer to go to the P drive and select "Add to archive.zip" or the Dutch equivalent "Toevoegen aan archive.zip" from the 7-Zip submenu.
- You can leave your zip file in your Eclipse workspace. We will collect it from there.
- Please, do not use specific packages but rather use the **default package** (this makes correcting the exam that much easier for us) → if you do use a specific package, you will lose 1 point.
- All software present on the computer can be used.
- The **network has been disabled**.
- **Mobile phones** remain in your backpack or coat and will not appear on the table. You should switch off your phone. If you have a **smartwatch**, please also put it in your backpack.
- Any attempt at fraud or actual act of **fraud**, will lead to a punishment by the Board of Examiners.

---

Mobility4You is a brand-new start-up that wants to start selling cars over the internet. Just like Amazon did for the book industry or Zalando did for the clothing industry, it wants to offer a wide variety of brands and make it easy for customers to quickly choose their favourite car. Mobility4You has also decided not to offer any diesel-powered cars, as these cars are less environmentally friendly. For now, their focus is solely on cars, but in the future, they might want to extend to electric bikes, etc. (do take that into account when designing your application!).

Specifically, Mobility4You gave you the following file format and wants you to develop an application around it.

An example of such a file looks as follows:
(obviously, you are not allowed to change the file format in any way!)

```
ELECTRIC_CAR Tesla, Model 3, 150KW, 50000Ah, 30000 euro
GAS_CAR Honda, Civic, 1.5L, 80KW, 18000 euro
HYBRID_CAR Toyota, Prius, 1.5L, 50KW, 12000Ah, 24000 euro
```

The complete file **mobility.txt** is in so-called CSV (Comma Separated Value) format and is available on the S drive (the exception to this CSV format being that there is no comma after the first element (e.g., ELECTRIC_CAR)).

The order of the properties (model, engine power, battery capacity, …) is fixed. The properties per element Electric Car, Gas-powered Car, and Hybrid Car are listed below:

An **ELECTRIC_CAR** is characterised by:

- The brand
- The model name
- The power of the engine
- The capacity of the battery
- The price

A **GAS_CAR** is characterised by:

- The brand
- The model name
- The engine displacement
- The power of the engine
- The price

A **HYBRID_CAR** is characterised by[1]:

- The brand
- The model name
- The engine displacement
- The power of the engine
- The capacity of the battery
- The price

Mobility4You asks you to design and implement a program that:

- **Reads in** the file mobility.txt
- **Stores** the read-in data in a suitable data structure
- **Outputs** the entire catalogue to the screen
- Allows to **add** new configurations of existing products (e.g., cars of different brands, cars with more powerful engines, etc.)
- Allows to **write to file** all product information (preserving the file format!).
- Write an **equals()** method for each class (except for the class that contains the main() method)
- To enable user interaction, please provide a **command line interface** with System.out.*. This interface should look like:

```
Please make your choice:
   1 - Show the entire Mobility4You catalogue
   2 – Add a new electric car
   3 – Add a new gas-powered car
   4 – Add a new hybrid car
   5 - Show the entire Mobility4You catalogue sorted by car-type
   6 – Show the entire Mobility4You catalogue sorted by brand
       (alphabetically)
   7 – Write to file
   8 – Stop the program
```

[1] A hybrid car has a traditional combustion engine and an electrical engine.

**Option 1**
All products are shown on screen in the same format as in the file:

```
ELECTRIC_CAR Tesla, Model 3, 150KW, 50000Ah, 30000 euro
GAS_CAR Honda, Civic, 1.5L, 80KW, 18000 euro
HYBRID_CAR Toyota, Prius, 1.5L, 50KW, 12000Ah, 24000 euro
```

**Option 2, 3 & 4**

- Through questions you ask the user to fill in all the necessary fields that compose an electric-powered (Option 2), gas-powered (Option 3) or hybrid (Option 4) car. Asking the user for a single line with all information is not sufficient.

**Option 5&6**

- Use the same file format as in Option 1 to show the catalogue sorted according to model (for example, first all electrical vehicles, then all hybrids, then all gas-powered cars – Option 5) or brand (Option 6).
- Please note that both sorting options **need to be implemented with a Comparator** (see later on in this assignment for more details)

**Option 7**
The data should be written to file, in the same format so that the application can read in the file again! The old file should be overwritten!

**Option 8**
The application stops.

Some important things to consider for this assignment:

- Think about the usefulness of applying **inheritance**.
- The **filename mobility.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 "command line input")
- Write **unit tests** (look at how your score for this exam is built up at the very end of this document)
- 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.

**Sorting for options 5 and 6 in more detail**

Mobility4You is asking you to implement two ways to sort their catalogue, namely by product (first all electrical vehicles, then all hybrids, then all gas-powered cars) and by brand (alphabetically, so first all cars from Audi, then BMW, …). It might very well be that later on, Mobility4You wants you to implement other types of sorting as well, that is why a flexible way of sorting is important.

Sorting needs to be done with a **Comparator**. While the concept of a Comparator might be new to you, everything you need to understand this concept has been covered during the lectures. The Java API documentation can help fill in the gaps. (*mind you, implementing sorting in a different way will not contribute to your grade*)

Furthermore, sorting can be computationally intensive, certainly if Mobility4You's product range continues to grow. That is why you should also try to have the sorting done with a **thread** so that while the catalogue is being sorted, other stuff can still be done. An important piece of advice here: make sure that you do not copy the data structure containing all products to sort it (work on the "original" one) and make sure that new cars cannot be added during sorting.

Hint: even if you are not able to implement the sorting in the short period of time that you have, you can still implement multi-threading. In that case, have a **separate thread** print "Sorting to be implemented" to the screen.

**Overview of the composition of your grade (total: 10)**

- 2.5 for compilation; if it doesn't compile → final score = 1
- 1 for a well-thought-out application of inheritance
    - Other criteria: are you using inheritance and polymorphism in the right way, is the functionality correctly distributed over the inheritance hierarchy
- 0.6 for correctly implementing equals() methods in all classes except for the class containing the main()
- 0.75 for implementing reading in a file (functioning code that leads to an exception still gives part of the score)
- 0.75 for writing to a file (functioning code that leads to an exception still gives part of the score)
- 0.5 for nicely styled code. Aspects being considered:
    - Length and complexity of methods
    - Length of parameter lists
    - Well-chosen identifier names
    - Whitespace, indentation, …
- 0.5 for a well-working textual interface (including option 1, which prints all products to the screen)
- 0.5 for not hardcoding the filename
- 1.2 for JUnit tests
    - 0.6 for testing a key class completely (depending on how well you test, you get a score between 0.0 and 0.6)
    - 0.6 for testing all other classes (except the class that contains main(), you also get a score between 0.0 and 0.6 depending on how well you test)
    - Do not use files in your tests! (although you can create a String with (part of) the content of a file to test reading in…)
- 1 for implementing threads
    - 0.5 for implementing the thread correctly
    - 0.5 for synchronisation
- 0.7 for implementing sorting with a Comparator

**There is a 1 point deduction if you do not work in the default package. So work in the default package!**