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

TUDelft

TI1206 Object-Oriented Programming (computer exam)
November 1st, 2017 13:30-16:30 (total duration: 3 hours)

Exam created by dr. A. Zaidman
and checked by Stefan Hugtenburg and Jesse Donkervliet

You can make use of the following during this exam:

- Java in Two Semesters (Charatan & Kans)
- The slides from the lectures available digitally on the X drive of your computer
- The Java API documentation (javadoc) that is also available on the X drive of your computer

This exam contains **1 assignment (10 points) and 1 bonus opportunity** (total exam: 5 pages).

Log into the computer with the following:

Login:	**ewi_ti1200**
Paswoord:	**Welkom01**

HINT 1:	**Read the entire 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:	**It might be that jUnit 4.0 is not in your Eclipse build path by default. If you add a unit test case through the "New" jUnit wizard, then Eclipse will notice that jUnit is not in the build path and Eclipse will direct you to the build path menu. Go to the "Libraries" tab, click "Add Library" and Eclipse will suggest to add jUnit.**

→ **the detailed scoring roster is on page 5**

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. **Failing to adhere to this naming scheme can cause your submission to not be graded.**
- The **workspace directory of Eclipse** is located on the Y: drive. Use the 7Zip program to go to the Y: drive to zip your assignment (do not try to zip your files through Windows Explorer, this might not work).
- **Leave your zip file on the Y drive, we will collect it 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.**
- There is no need to upload the zip file, we will collect it.
- All software present on the computer can be used; obviously, internet services have been disabled.
- **Mobile phones** remain in your backpack or coat and will not appear on the table. Switch off your phone!
- If you make an **attempt at fraud** or **commit fraud**, you will be punished.

EcoBike is an innovative player in the market of short-distance transportation solutions. They are expanding their business from a traditional shop in downtown Delft to a worldwide presence on the internet. To enable that step, they are asking you to build a barebones software system to help them enable that transition. As they are aware that within the time you are given you can only design a prototype application, they are asking you to focus on their 3 core products, namely: folding bikes, electric bikes and electric scooters. There is a clear aim to expand their online product portfolio in the short term, so do take that into consideration when designing your software system!

Specifically, EcoBike 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!)

```
FOLDING BIKE Brompton; 20; 6; 9283; TRUE; black; 1199
E-SCOOTER Peugeot; 45; 5426; TRUE; blue; 8000; 875
E-BIKE Gazelle; 49; 16455; TRUE; red; 16000; 1499
FOLDING BIKE Dahon; 16; 1; 11109; FALSE; white; 899
E-BIKE Koga; 48; 15488; TRUE; red; 21000; 1899
```

The complete file **ecobike.txt** is available on the X drive of your computer.

The order of the properties (brand, gears, battery capacity, …) is fixed. The properties per folding bike, e-scooter and e-bike are listed below:

A **folding bike** is characterized by:

- A brand
- The size of the wheels (in inch)
- The number of gears
- The weight of the bike (in grams)
- The availability of lights at front and back (TRUE/FALSE) boolean
- A color
- The price

An **e-scooter** is characterized by:

- A brand
- The maximum speed (in km/h)
- The weight of the e-scooter (in grams)
- The availability of lights at front and back (TRUE/FALSE) boolean
- The battery capacity (in mAh)
- A color
- The price

An **e-bike** is characterized by:

- A brand
- The maximum speed (in km/h)
- The weight of the e-bike (in grams)
- The availability of lights at front and back (TRUE/FALSE) boolean
- The battery capacity (in mAh)
- A color
- The price

EcoBike asks you to design and implement a program that:

- **Reads in** the file ecobike.txt when starting the program.
- 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 EcoBike catalogue
    2 - Add a new folding bike
    3 - Add a new e-scooter
    4 - Add a new e-bike
    5 - Find first item of a particular brand
    6 - Write to file
    7 - Stop the program
```

Option 1
All products are shown on screen in the following format:

```
E-BIKE Koga with 15488 mAh battery and head/tail light.
     Price: 1899 euros.
FOLDING BIKE Dahon with 1 gear(s) and no head/tail light.
     Price: 899 euros.
E-SCOOTER Peugeot with 5426 mAh battery and head/tail light.
     Price: 875 euros.
```

Option 2, 3 & 4

- Through questions you ask the user to fill in all the necessary fields that compose either a new folding bike (Option 2), an e-scooter (Option 3) or an e-bike (Option 4). Use System.in (e.g., have a look at the example program 7.6 on page 178 of the 3rd edition of the book).

Option 5
There are 2 distinct ways to implement the search functionality that enables to find the first item (e-bikes, folding bikes and e-scooters) from a certain brand:

1. Implement the search functionality yourself.
2. Implement it using the "binary search" that is available in the class Collections **→ have a look at the final page to understand the consequences of either implementation choice**

As **sorting** can be computationally expensive, you are given the option to use multi-threading to ensure that the application remains responsive during searching.
An important piece of advice here: make sure that you do not copy the datastructure containing all products to search it (work on the "original" one) and make sure that new items cannot be added during sorting.

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

Option 6
The data should be written to file, in the same format so that the application can read in the file again!

Option 7
The application stops.

Some important things to consider for this assignment:

- Think about the usefulness of applying **inheritance**.
- When writing to the file ecobike.txt, the old version of the file should be overwritten.
- The **filename ecobike.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.
- Javadoc is not required, but will lead to a bonus (see below)

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

- 2.5 for compilation; if it doesn't compile → final score = 1
- 1.2 for a well thought-out application of inheritance
    - Specifically: are you using inheritance and polymorphism in the right way, is the functionality correctly distributed over the inheritance hierarchy, is the visibility (public, private, …) of attributes/methods logical, did you consider specifying abstract, final, …
- 0.5 for correctly implementing equals() methods in all classes except for the class containing the main()
- 0.75 for implementing reading in a file (code that leads to exception still receives partial credit)
- 0.75 for writing to a file (code that leads to exception still receives partial credit)
- 0.3 for nicely styled code. Aspects being considered:
    - Length and complexity of methods
    - Length of parameter lists
    - Well-chosen identifier names
    - Whitespace, indentation, …
- 0.4 for option 1, outputting the contents of catalogue of e-bikes, folding bikes and e-scooters to screen
- 0.3 for a well-working textual interface
- 0.5 for not hardcoding the filename
- 1.1 for jUnit tests
    - 0.5 for testing the class "folding bike" (depending on how well you test, you get a score between 0.0 and 0.5)
    - 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 corrrectly
    - 0.5 for synchronization
- 0.7 for sorting:
    - Maximal score of 0.2 if you implement it yourself
    - Maximal score of 0.7 if you implement it using binary search

Bonus opportunity: there is a bonus of maximally 0.4 if you write sufficiently specified JavaDoc comments for 1 or more classes.

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