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

TU Delft

TI1206 Object-Oriented Programming (computer exam)
November 2nd, 2016, 13:30-16:30 (total duration: 3 hours)

Exam created by dr. A. Zaidman and checked by dr. G. Gousios

You can make use of the following during this exam:

- Java in Two Semesters (Charatan & Kans)
- The slides from the lectures, either printed on paper or available online through Blackboard + notes
- The Java API document (javadoc) that is available through Blackboard (Blackboard → TI1206 → Assignments → Java 1.8 API)

This exam contains **1 assignment (10 points)** (total exam: 5 pages).
→ **the detailed scoring roster is on page 5**

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 pat hand Eclipse will directly you tot he build path menu. Go to the "Libraries" tab, click "Add Library" and Eclipse will suggest to add jUnit.**

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 H: drive. Use the 7Zip program to go to the H: drive to zip your assignment (do not try to zip your files through Windows Explorer, this might not work).

- 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.
- **Upload** the zip file containing your solution through Blackboard → TI1206 → Assignments → Exam November 2nd, 2016. At that location, you will also find the inputfile that you should use for the exam assignment.
- All software present on the computer can be used.
- The **network has been disabled** so that you can only access Blackboard.
- **Mobile phones** remain in your backpack or coat and will not appear on the table. You should switch off your phone.
- If you make an attempt at fraud or commit **fraud**, you will be punished.

Apple is looking to extend its product range. You've been asked to prepare a new software system that can easily accommodate this new range of products. As Apple is very much aware that in the time given, you can only produce a prototype application, it asks you to focus your implementation on their most successful iOS products, namely the iPhone and the iPad. There is a clear aim to later integrate their Mac line of computers, iPod Touch and Beats audio products. Do take this into consideration!

Specifically, Apple 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!)

```
IPHONE 7, 4.7, A10, GSM, JET BLACK, 32GB, TRUE, 700
IPAD AIR 2, 9.7, A8, TRUE, SILVER, 64GB, 400
```

The complete file **apple.txt** is in so-called CSV (Comma Separated Value) format and is available through Blackboard (the exception to this CSV format being that there is no comma after the first element (IPHONE/IPAD)).

The order of the properties (model, screensize, processor, …) is fixed. The properties per element iPhone, iPad are listed below:

An **iPhone** is characterized by:
- A model name
- A screensize
- A processor
- A type of modem, either GSM or CDMA
- A color
- The amount of memory
- The presence of 3D Touch technology
- The price

An **iPad** is characterized by:
- A model name
- A screensize
- A processor
- The presence of 4G, rather than only wifi

- A color
- The amount of memory
- The price

Apple asks you to design and implement a program that:

- **Reads in** the file apple.txt
- **Output** the entire catalogue to screen
- **Sort** the entire catalogue according to type (so first all iPhones, then all iPads) or price (from low to high)
- Allows to **add** new configurations of existing products (e.g. with higher capacity memory or new colors)
- 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 Apple catalogue
   2 - Add a new iPhone
   3 - Add a new iPad
   4 - Show the entire Apple catalogue sorted by product
   5 - Show the entire Apple catalogue sorted by price (low → high)
   6 - Write to file
   7 - Stop the program
```

Option 1

All products are shown on screen in the following format:

```
Apple iPhone 7 with 32GB of memory
     with an A10 processor and 4.7 inch screen
     700 euros
Apple iPad Air 2 with 64GB of memory
     with an A8 processor and 9.7 inch screen
     having WiFi and 4G technology
     400 euros
```

Option 2 & 3

- Through questions you ask the user to fill in all the necessary fields that compose an iPhone (Option 2) or iPad (Option 3). Use System.in and have a look at the example program 7.6 on page 178 of the 3rd edition of the book.

Option 4&5

- Use the same file format as in Option 1 to show the catalogue sorted according to model (Option 4) or price (Option 5).
- Please note that both sorting options need to be implemented with a Comparator (see later on in this assignment for more details)

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 apple.txt, the old version of the file should be overwritten.
- The **filename apple.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)

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.

Sorting for options 4 and 5 in more detail

Apple is asking you to implement two ways to sort their catalogue, namely by product (which product comes first doesn't matter) and by price (from low to high). It might very well be that later on, Apple 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 might also help.
(mind you, implementing sorting in a different way will not contribute to your grade)

Furthermore, sorting can be computationally intensive, certainly if Apple'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 datastructure containing all products to sort it (work on the "original" one) and make sure that new iPhones or iPads 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 the 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)

- 2.75 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
- 0.5 for correctly implementing equals() methods in all classes except for the class containing the main()
- 0.7 for implementing reading in a file (functioning code that leads to exception still gives part of the score)
- 0.7 for writing to a file (functioning code that leads to 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 trains to screen)
- 0.5 for not hardcoding the filename
- 1.1 for jUnit tests
    - 0.5 for testing the class iPhone (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
- 0.75 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!