CSE1100: Object Oriented Programming

T. Overklift / A. Zaidman

Date: 05-01-2021 Duration: 3 hours Pages: 5

Allowed resources

The following resources can be used during this exam:

- The slides, videos, assignments, tutorials, and book(s) about CSE1100.
- The Java API documentation.
- Existing internet resources (**if you copy a solution from the internet, copy the URL and put the URL in the Javadoc**)
- Any IDE that you like (e.g., IntelliJ), including its features to generate code like getters, equals methods, etc.

Disallowed resources

- Any information that you obtain from a person, either directly or indirectly.
- Actively asking / answering questions on online platforms.

→ Attempts at fraud will be reported to the Board of Examiners

---

About the exam

- Read the entire assignment and only then start implementing.
- You can find a scoring roster on the final page of this document.
- Hand in your program **before 21:30** (when working with regular time).
    - **Upload** a zip file holding your solution through Weblab.
    - Create your zip file **five minutes ahead of the deadline.**
    - Make sure it includes at least your source files (.java).
    - Name the zip file **[studentnumber].zip**, e.g., 1234567.zip You can find your student number on your student card (7 digits; below your picture).
- Please, do not use specific packages but 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.
- Your project (sources and tests) **must compile** to get a passing grade. Sometimes your IDE will allow you to "proceed" even when some classes do not compile. We will still see this as a compilation failure. Ensure that ALL classes you hand in compile.

Setting up your project in IntelliJ

- Once IntelliJ has started go to File > New > Project
- Select a suitable Java version (we recommend 11 or higher) and click next twice.
- Type a project name (the exact name does not matter).
- Click finish.
- Do not use packages other than the "default package".

PersonalFit web shop

PersonalFit is a young and energetic company that specializes in fitness equipment and related products. With the COVID-19 pandemic they have seen a surge in customers as everyone wants to buy fitness equipment now that sports clubs need to stay closed. To accommodate further growth, they need a new IT system for internal use. You are now tasked with building a prototype of that system.

You need to create a **prototype with a textual interface**.

The prototype application that you should develop reads in a manifest file that contains products that PersonalFit sells. An example manifest file is shown below. The complete file **personalfit.txt** is available through Weblab. Your implementation needs to be able to read the provided file on regardless of your operating system, so make sure you test on the file downloaded from Weblab.

```
HomeTrainerBike Tunturi; T1200; Glutes, Leg Muscles; TRUE; 234 euros
ProteinShake AlbertHeijn; Chocolate; 1000 grams; 23 euros
SpinningBike Tunturi; T2100; Glutes, Leg Muscles, Back, Shoulders; FALSE;
12KG; Magnetic; 499 euros
ProteinShake QNT; Banana; 300 grams; 12 euros
ProteinShake QNT; Vanilla; 600 grams; 20 euros
SpinningBike VirtuFit; Radical2000; Glutes, Leg Muscles, Back, Shoulders;
TRUE; 21KG; Magnetic; 799 euros
```

The 3 main products that PersonalFit sells are the traditional HomeTrainerBike, the more advanced SpinningBike, and the protein shakes. The products are categorized by the following properties:

HomeTrainerBike

- Brand
- Model
- Muscle groups (this is a comma separated list of all muscle groups targeted)
- Whether an electronic display is present that shows key info such as the distance covered (TRUE/FALSE).
- The price

SpinningBike (is a special kind of HomeTrainerBike)

- Brand
- Model
- Muscle groups (this is a comma separated list of all muscle groups targeted)
- Whether an electronic display is present that shows key info such as the distance covered (TRUE/FALSE).
- The weight of the flywheel
- The resistance mechanism (e.g., traditional, magnetic, …)
- The price

ProteinShake

- Brand
- Flavour
- Size in weight
- The price

PersonalFit asks you to design and implement a program that:

- **Reads in** a file as specified by the user, for example the file personalfit.txt. Reading in the file happens at the start-up of the program.
- **Outputs** the entire catalogue of products that PersonalFit has (this includes both protein shakes and fitness bikes).
- Offers a way to **randomly propose a protein shake** to a customer.
- **Add a new protein shake to** the catalogue.
- **Write** all information in the catalogue to a file (preserving the file format!).
- Write an **`equals()`** method for each class (except for the class that contains the `main()` method)
- Make sure that your program is designed in such a way that it is easy to add new product categories, all the while storing all products in an efficient, yet uniform way.

To enable user interaction, please provide a **command line interface** with `System.out.*`. This interface should have the following menu:

```
Please make your choice:
    1 – Print all bikes & protein shakes
    2 – Add a new protein shake
    3 – Propose a random protein shake to the customer
    4 – Show bikes that train a specific muscle group
    5 – Write to file
    6 – Stop the program
```

Option 1: Print all elements in the catalogue
This option prints all bikes and shakes that are currently in the system in a neat format. *An example print is shown below. Please note: your format does not have to be the same as below, but should be more neat than in the input file.*

```
Bike: Tunturi T1200
      Muscle groups: Glutes & Leg Muscles
      An electronic display that visualizes speed is present.
      The price is: 234 euros.
Protein Shake: Albert Heijn,
      Flavour: chocolate
      Size: 100 grams
      Price: 23 euros.
```

Option 2: Add a new protein shake
Through questions you ask the user to fill in all the necessary data for a protein shake. Subsequently, you add the new protein shake to the data structure.

Option 3: Propose a random protein shake (& print it)
As PersonalFit strongly believes in protein shakes, they want to advertise a protein shake to each of their customers. For this initial implementation, PersonalFit asks you to implement functionality that can propose a **random** protein shake in the application. If this option is selected, you randomly select a protein shake from the protein shakes that are in stock and print the information related to this shake.
To ensure that the program is still responsive while a random protein shake is being selected. Please implement this feature with **multi-threading** and make sure that no additional protein shakes can be added during the random selection of a protein shake to avoid potential concurrency issues. If you can implement multi-threading but not the random shake proposal, please print “Thread started" or something similar from the new thread.

Option 4: Search for a muscle group (& print all bikes that train this muscle group)
You should parse out all muscle groups and store them in a list of muscle groups affected per bike. Then search for bikes that target the muscle group the user wants.
A) If no bike meets the criteria. Report that back to the user.
B) If one or more bikes do train that muscle group, print those bikes to screen like Option 1.

For full points you need to implement option 4 using **functional programming** (e.g. using a stream instead of a for loop).

Option 5: Write to file
Create new file called “output.txt" with the information that is currently in the system in the same file format as the input. The information that you write to file should come from the data structure that you have created.

Option 6: Quit application
The application stops.

Some important things to consider for this assignment:

- The program *and* the tests should **compile**
- Think about the usefulness of applying **inheritance and/or interfaces**.
- The **filename personalfit.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)
- 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 supply Javadoc comments.

Handing in your work

To hand in your work you must create a **ZIP** (so not .7z or anything else) file holding all your **.java**-files, you may also .zip your entire project folder.

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

Double-check the correctness of the number using your campus card.

Finally, upload your .zip archive to Weblab!

Grade composition

2 points **Compilation**
- If your solution does not compile → final score = 1

1.3 points **Inheritance**
- Proper use of inheritance. Additionally, there should be a good division of logic between classes & interfaces, as well as the proper use of (non-)access modifiers.

0.3 points **`equals()` implementation**
- Correct implementation of `equals()` in all classes that are part of your data model.

0.6 points **File reading**
- Being able to read the user-specified files and parsing the information into Objects. *A partially functioning reader may still give some points.*

0.6 points **File writing**
- Being able to write the user-specified files and parsing the information into Objects. *A partially functioning writer may still give some points.*

1 point **Code style**
- Ensure you have code that is readable. This includes (among others) clear naming, use of whitespaces, length and complexity of methods, Javadoc, etc.

0.5 points **User Interface**
- Having a well-working textual interface (including option 1, which prints the catalogue contents). *A partially functioning interface may still give some points.*

0.3 points **Adding new protein shakes**
- For adding new protein shakes and letting the user enter the details (option 2).

1.4 points **JUnit tests**
- *0.6 points* for testing the class SpinningBike (depending on how well you test, you get a score between 0.0 and 0.5)
- *0.6 points* for testing all other classes & methods (except the `main()` method, and methods that depend on external resources such as files, you 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…).
- *0.2 points* for testing with randomness, consider using a *seed (Do not forget to mention where you found the information – the URL!)*

0.6 points **Threads**
- *0.3 points* for implementing the thread correctly.
- *0.3 points* for the correct use of synchronization.

0.5 points **Propose a protein shake** (with randomness, options 3).
- *0.2 points* for a working implementation.
- *0.3 points* for using proper randomness.

0.9 points **Search for bikes that target specific muscle groups.**
- *0.5 points* for a working implementation.
- *0.4 points* for using functional programming.

There is a 1-point penalty if you do not work in the default package.
There is a 0.5-point penalty if you hardcode the filename.