Object Oriented Programming

CSE1100 – Version A

A. Zaidman / T. Overklift

Date: 09-01-2020 Duration: 3 hours Pages: 5

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)
- The Java API documentation (located on the desktop of the computer)

Set up your computer

Log in using:

Username: `EWI-CSE1100`
Password: `GoodLuck2020`

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 Eclipse already and while Eclipse 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 Eclipse will allow you to "proceed" even when some classes fail to compile. We will still see this as a compilation failure. Ensure that **ALL classes you hand in compile.**
- 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 your project in Eclipse

- Once Eclipse has started go to **File > New > Java Project**
- Type a project name (the exact name doesn't matter) and under "JRE" select **"use default JRE (openjdk-12.0.2)"**.
- Click **Finish**.
- Eclipse will prompt you to create a **module-info.java** file; click **"don't create"**, as this file will prevent you from working in the default package. If you erroneously created it anyway, delete it.
- **Don't use packages** other than the "default package".

Mobile Warriors mobile phone shop

We've all become addicted to our mobile phones and wireless carriers have seen a huge increase in the number of calling minutes, texts and megabytes that the average user consumes on a monthly basis. Mobile Warriors is a vendor of mobile phone subscriptions and pre-paid SIM cards. They are targeting students in particular and often have very good deals for students. They are asking you to come up with a simple textual application that brings order to the chaos that is the diversity of mobile phone subscriptions.

You are given two files, one that contains all the mobile phone subscriptions (`subscriptions.txt`), and another one that contains all the prepaid offerings (`prepaid.txt`). They ask you to consolidate these two sources of information, so that a sales manager can give an overview of the offerings based on the wishes of the customer (e.g., a certain amount of minutes for calling).

subscriptions.txt

```
KPN, Gold, EUR 29, unlimited, unlimited, 9
KPN, Gold, EUR 29, unlimited, unlimited, 9, Student, 20%
KPN, Silver, EUR 16, 200, unlimited, 4
Tele2, Huge, EUR 35, unlimited, unlimited, unlimited
```

This example file "`subscriptions.txt`" is available to you on the S-drive of your computer.

prepaid.txt

```
KPN, Hello You, EUR 10, 60, 250, 2, false, false, true
KPN, Hello You, EUR 10, 60, 250, 2, false, false, true, Student, 10%
T-Mobile, Connected+, EUR 15, 150, 500, 4, false, false, true
T-Mobile, Connected+, EUR 15, 150, 500, 4, false, false, true, Student, 15%
T-Mobile, Connected+Student, EUR 13, 100, 1000, 4, true, true, true, Student, 0%
```

This example file "`prepaid.txt`" is available to you on the S-drive of your computer.

The structure of the file `subscriptions.txt` is:

- Name of the provider
- Name of the subscription formula
- Monthly price
- Number of call minutes
- Number of texts
- Number of gigabytes
- Optional: Student (this subscription can only be offered to students)
- Optional: Student discount rate

The structure of the file `prepaid.txt` is:

- Name of the provider
- Name of the prepaid formula
- Monthly price
- Number of call minutes
- Number of texts
- Number of gigabytes
- Whether the call minutes are carried over to the next month if not depleted
- Whether the texts are carried over to the next month if not depleted
- Whether the gigabytes minutes are carried over to the next month if not depleted
- Optional: Student (this prepaid plan can only be offered to students)
- Optional: Student discount rate

Application requirements

Mobile Warriors asks you to design an application with a textual interface that can do the following;

- Ask the user for two **filenames** that should be read. The first file should depict the subscriptions, the second file should depict the prepaid formulas. Make this order clear to the user
- **Read** the files ("`subscriptions.txt`") & ("`prepaid.txt`") and **store** the information in ***one*** data structure.
- **Show** several different menu options (see next page for more information)
- **Close** the application.

Secondary requirements

- Consider the usefulness of applying **inheritance** and / or **interfaces**.
- Write **unit tests**.
- Ensure it is easy to extend the program in the future (e.g., fixed-line subscriptions).
- Make sure you use a nice **programming style**; code indentation, whitespaces, logical identifier names, reasonable method lengths, number of class attributes, data types, etc.
- Provide **Javadoc** for all non-test code.
- Provide an **equals** method for every class, except the class that contains the `main()` method.

Issues with reading

In the event that you experience trouble with implementing a reader, you are allowed to create hard-coded lists of objects with the information from the .txt files. You can use this to implement the rest of the assignment, and make it possible to get points for a working application. Please refer to the grading sheet on the last page of this exam to view a detailed division of points.

Interface

To enable interaction with the user, a **command line interface** should be provided. Since the interface allows the user to interact with information that you need to read from files, make sure that all information has been read before the menu is shown.
After selecting an option the user should return to the main menu, and the application should continue running until the "stop" option (number 4) is selected.
The interface should look like this:

```
Please make your choice:
   1 – Show all subscriptions and prepaid formulas
   2 – Filter subscriptions and prepaid formulas
   3 – Write to file
   4 – Stop the program
```

Option 1 – Showing all subscriptions/prepaid formulas

When the user selects this option all plans in the system are printed in a user friendly way: this means you print all the attributes with a description of what they mean. Please incorporate any applicable discount in the price of the product.

As an example (for 3 of the items in the input):

```
Subscription plan: KPN Gold, costs 29.00 euro
      includes unlimited calling minutes, unlimited texts and 9GB of data.

Subscription plan for students: KPN Gold, costs 23.20 euro
      includes unlimited calling minutes, unlimited texts and 9GB of data.

Prepaid plan for students: T-Mobile Connected+, costs 12.75 euro
      includes 150 calling minutes, 500 texts and 4GB of data.
      calling minutes carry over: no.
      texts carry over: no.
      internet data carry over: yes.
```

Option 2 – Filtering subscriptions/prepaid formulas

When the user chooses option 2 they should be prompted to make several choices:

- Ask the user whether (s)he want to see: all phone plans, only subscription plans or only prepaid plans.
- Ask the user whether (s)he is a student and would thus be eligible for student plans (don't forget to apply relevant discounts when showing the results!).
- Ask the user to choose between one of the following two options and let them enter the amount:
    - A maximum price that they are willing to pay
    - A minimum amount of calling minutes, texts *and* gigabytes that they want.

Then the application should print the plans that meet the requirements of the user. Don't forget to incorporate any student discounts that apply in the price of the products!

For instance, the requirements: *only subscriptions, not a student and a maximum price of 20 euro's* should give the following result:

```
Subscription plan: KPN Silver, costs 16.00 euro
      includes 200 calling minutes, unlimited texts and 4GB of data.
```

Make sure the application properly informs the user if there are no plans that meet their requirements, and that the application does not crash.

Option 3 – Write to file

In this option, you should create a new file (preferably by asking the user for a filename to make sure you don't overwrite an existing file).

This file should contain all subscriptions, disregarding their type (subscription/prepaid) or their audience (student/non-student), but listed from cheapest to most expensive. Please uphold the file format used in the existing files. *In terms of using the price to sort, please use the base price without any adjustments for students.*

Option 4 – Quitting the application

This option should terminate the application.

Grade composition

1.7 points **Compilation**

Ensuring that your entire project does not contain any errors.
Any compilation error (red error indicator in Eclipse) results in a final grade of 1.

1.5 *points* **Inheritance**

Proper use of inheritance (*1 point*). Additionally there should be a good division of logic between classes & interfaces as well as the proper use of (non-)access modifiers (*0.5 points*).

0.5 points **`equals()` implementation**

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

1.3 points **File reading**

Being able to read the user-specified files, and parsing the information into Objects (*0.65 points per well-working file reader*). *A partially functioning reader may still give some points.*

1.2 points **Code style**

Ensure you have code that is readable. This includes (among others) clear naming, proper use of whitespaces, length, number of arguments and complexity of methods and classes, Javadoc, etc.

0.5 points **Interface**

Having a well-working (looping) interface, that terminates when options 4 is chosen (*0.2 points*). Having a functioning option 1 (*0.3 points*).

1.0 points **Data processing**

Having a well-working implementation of option 3 (*0.8 points*). When there are no suitable plans this is clearly communicated to the user, and the application does not crash (*0.2 points*). *A partially functioning filter may still give some points.*

0.8 points **Writing to file**

Being able to output the contents of both files integrated into 1 file in the correct format (*0.5 points)*. Additionally the output is correctly sorted (*0.1 points)*, the output has been sorted using either Streams or Collections *(0.2 points)*.

1.5 points **Unit testing**

Having all classes except main completely tested (*1.5 points*). A p*artially tested applications may still give some points.*

Penalties

-1 point **Using a package**

Using a package other than the 'default package'.

-0.5 points **Hardcoding the filename(s) of the input file**

Not asking the user to enter file names will lead to a deduction of points.

-0.5 points **Handing in a wrongly named ZIP file**

Double check the filename (and extension! (see below)) you use before handing in.

Handing in your work

To hand in your work you must create a **ZIP** (so not .7z or anything else) file containing all your .**java**-files. Do *not* include your .class-files or anything else.

Go to your private "P-drive" and navigate to your project. You should see your `src` folder. Select this **`src`**-folder, **right**-click, hover "7Zip", and select "Add to *src*.**zip**"
(or in Dutch: "Toevoegen aan *src*.**zip**").

Important: Rename the ZIP file to your student number, e.g. "`4567890.zip`". If you don't see the file name change try refreshing the folder (e.g. press F5).

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.

Please log off when you leave!