Object Oriented Programming

CSE1100 – Version A

A. Zaidman / T. Overklift / S.Y. van den Oever

Date: 05-11-2019 Duration: 3 hours Pages: 6

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: `GoodLuck1100`

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".

Marine Management

A certain marine zoo near Harderwijk wants to start a renovation for their park. Before they start moving animals around they want to get a better overview of all of the different shelters they have and which types of animals the shelters are suitable for. This way they know where they can temporarily relocate certain animals. You will get two TXT files that contain data about shelters and animals. *You may assume the provided files are free of errors.*

Info about the shelters

```
Coastal A.1 470 False 40
Tundra B.1 680 True
Reef C.1 745 True 38
Tundra B.2 1860 False
Coastal A.2 2210 True 280
Coastal A.3 160 True 0
Tundra B.3 1340 True
Reef C.2 390 True 17
Coastal A.4 1185 True 130
```

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

Please note that for the shelters, the first word on each line indicates the **type of the shelter**. Following the type are the characteristics of the different shelters (listed in order of appearance):

Coastal Shelter

- ID (name)
- Volume (in m^3)
- Availability (true / false)
- Land surface area (in m^2)

Constant attributes (not in the input file, the same for all coastal shelters):

- Water type: "*Cool Eutrophic*"
- Climate: "*Temperate*"

Tundra Shelter

- ID (name)
- Volume (in m^3)
- Availability (true / false)

Constant attributes (not in the input file, the same for all tundra shelters):

- Water type: "*Cool Eutrophic*"
- Climate: "*Polar*"

Reef Shelter

- ID (name)
- Volume (in m^3)
- Availability (true / false)
- Number of unique coral types

Constant attributes (not in the input file, the same for all reef shelters):

- Water type: "*Warm Trophic*"
- Climate: "*Tropical*"

Info about the animals

```
Penguin; 300; Tundra, Coastal
Duck; 50; Coastal
Rock Crab; 80; Coastal, Tundra, Reef
Shark; 900; Reef, Coastal
Herring; 40; Tundra, Coastal
Orca; 1400; Tundra
Bottlenose Dolphin; 600; Coastal
```

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

The characteristics of the animals are (in order of appearance in the file):

Animal

- Name
- Volume required (in m^3)
- List of possible shelters, in order of preference, from most preferred to least preferred.

Application requirements

The zoo 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 order of reading is irrelevant, as long as you specify clearly to the user what input you expect.
- **Read** the delivered file format ("`shelters.txt`") & ("`animals.txt`") and **store** it in a 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**. See the grading scheme for more details.
- Ensure it is easy to extend the program in the future (new shelters / properties).
- Make sure you use a nice **programming style**; code indentation, whitespaces, logical identifier names, reasonable method lengths, 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.
    - Since the *water type* and *climate* properties are constant for all shelters of the same type, those **should not be checked** in your equals methods!

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. However, if you do this you **will receive 0 points** for your reader, regardless of its state. 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 6) is selected.
The interface should look like this:

```
Please make your choice:
   1 - Show all shelters
   2 - Show all animals
   3 - Show all shelters suitable for a specific animal
   4 - Show the optimal shelters for a specific animal
   5 - Show the constant properties per shelter type
   6 - Stop the program
```

Option 1 – Showing all shelters

This option should list all the shelters that are held in the data structure (and all their object specific properties). This information should be shown in a user friendly way.

For instance:

Coastal Shelter
`ID: A.1 - Volume: 470 m3 - Available: False - Land surface: 40 m2`

Option 2 – Showing all animals

This option should list all the animals that are held in the data structure (and all their object specific properties). This information should be shown in a user friendly way.

For instance:

`Penguin - Requires: 300 m3 - Preferred shelter: Tundra, Coastal`

Option 3 – Show all shelters suitable for a specific animal

In order to see where they can relocate animals, the zoo wants to be able to see all shelters a specific animal can move to. An animal can move to a shelter if it is of a type that the animal is compatible with, and has a volume that is equal to or greater than the minimum volume the animal requires. A suitable shelter will be shown, regardless of its availability. The zoo wants to be able to enter the name of an animal and get this overview.
If the animal entered is not known, or if there are no suitable shelters available, this should be communicated clearly to the user (without crashing the program).

For instance, entering `Penguin` should return:

Coastal Shelter
`ID: A.1 - Volume: 470 m3 - Available: False - Land surface: 40 m2`
Tundra Shelter
`ID: B.2 - Volume: 1860 m3 - Available: False`
Tundra Shelter
`ID: B.1 - Volume: 680 m3 - Available: True`
Coastal Shelter
`ID: A.2 - Volume: 2210 m3 - Available: True - Land surface: 280 m2`
Tundra Shelter
`ID: B.3 - Volume: 1340 m3 - Available: True`
Coastal Shelter
`ID: A.4 - Volume: 1185 m3 - Available: True - Land surface: 130 m2`

Option 4 – Show optimal shelters for a specific animal

In order to easily find the "optimal" shelter for an animal the zoo wants the following: when the user enters the name of an animal this option should print a filtered and sorted list of shelters. The list should be:

- **Filtered** on shelters that are **large enough** for the animal.
- **Filtered** on shelters that are **available**.
- **Sorted** primarily **descending on the preference of the animal** (shelters of the type that is most preferred by the animal should be listed first; shelters of the type that match the second preference of the animal after that, and so on).
- **Sorted** secondarily **ascending on the volume of the shelter** (the shelter that has the smallest possible size within a shelter type, should be listed first).

If the animal entered is not known, or if there are no suitable shelters available, this should be communicated clearly to the user (without crashing the program).

For instance, entering `Penguin` should return:

```
Tundra Shelter
ID: B.1 – Volume: 680 m3 – Available: True
Tundra Shelter
ID: B.3 – Volume: 1340 m3 – Available: True
Coastal Shelter
ID: A.4 – Volume: 1185 m3 – Available: True – Land surface: 130 m2
Coastal Shelter
ID: A.2 – Volume: 2210 m3 – Available: True – Land surface: 280 m2
```

Option 5 – Show shelter properties

This option should print the *water type* and *climate* properties for a shelter type entered by the user. All properties should be shown in a user friendly way.
If the shelter type entered is not known, this should be communicated clearly to the user (without crashing the program).

For instance, entering `Coastal` should return:

```
Coastal Shelter
Warm Trophic Water – Temperate Climate
```

Option 6 – Quitting the application

This option should terminate the application.

Grade composition

1.5 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.

2.0 *points* **Inheritance**

Proper use of inheritance (*1 point*). Additionally there should be a good division of logic between classes & interfaces (*0.5 points*) 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.

1.5 point **File reading**

Being able to read the user-specified files, and parsing the information into Objects (*0.75 points per well-working file reader*). *A partially functioning reader may still give some points.*

1 points **Code style**

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

0.5 points **Interface**

Having a well-working (looping) interface (*0.2 points*). Having a functioning option 1, 2 & 5 (*0.1 points each*). *A partially functioning interface may still give some points.*

1. 5 points **Data processing**

Having a well-working implementation of options 3 (*0.5 points*) & 4 (*0.7 points*), and clearly informing the user when no shelters / animal was found (*0.3 points*).

1.5 points **Unit testing**

Having one key-class tested completely (*0.6 points*), having all other classes except main completely tested (*0.6 points*) and having tested the methods to read files properly, without actually using the files (*0.3 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 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!