Object Oriented Programming

CSE1100 – Programming Exam

A. Zaidman / T. Overklift

Date: 09-11-2021 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).
- A local (limited) version of Answers EWI.
- The Java API documentation (located on the desktop of the computer).

Set up your computer

Log in using:

Username: `EWI-CSE1100`
Password: `GoodLuck!1100`

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 IntelliJ and accept the licence agreement. While IntelliJ 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 IntelliJ will allow you to run part of your code even when some classes fail to compile. We will still see this as a compilation failure. Ensure that **ALL classes you hand in compile.** If your solution does not compile, you will fail the exam.
- 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 the template project

- Open an explorer and copy the template project ("`OOP Exam Template`") from the (read only) S-drive and paste it on your P-drive.
- Subsequently open the project on the P-drive in IntelliJ (you can open the project by opening the folder and double-clicking the `.iml` file, or via the open option in IntelliJ).
- Once you have opened the project you can start implementing your exam in this project!

Mech Manual

Aloïse is a game designer from a small Dutch game studio called “Commando Games”. She has a cool idea for a new game with all kinds of mechanical animals that the player has to fight. The animals will all have different attributes, strengths and weaknesses. The game will involve some sort of combat system, but how that works exactly will be decided at some point in the future. She asks you (the intern) to create a prototype for her idea, to see if it is feasible.

Specifically, she asks you to develop an application that can read in the following information about different mechanical animals and equipment she designed:

```
4
MECH - Cloudwing - Bravo - 13568
3ATK - 1DEF - 15HP - 4SPD
strength: wind - weakness: poison
MECH - Howler - Delta - 29845
2ATK - 2DEF - 24HP - 3SPD
strength: fire
MECH - Frosthorn - Omega - 67458
3ATK - 2DEF - 58HP - 1SPD
strength: ice - weakness: fire - weakness: acid
MECH - Acid Crawler - Bravo - 43564
5ATK - 4DEF - 36HP - 5SPD
strength: acid - weakness: poison
4
EQUIPMENT - Rusted Spear - Offensive - 13568
1ATK

EQUIMENT - Leather Armor - Defensive - 43564
1DEF
weakness: wind
EQUIPMENT - Assassins Blade - Offensive - 67458
3ATK
strength: poison
EQUIMENT - Power Armor - Defensive - 29845
3DEF
strength: fire - strength: electric
```

The file **mechs.txt** is available in the template project.

The order of the lines for the mechs (and pieces of equipment is fixed).
The first line of the file contains a number indicating the number of Mechs that will follow.
For each mech you will always first get the name and type of the mech, followed by an ID for a lootdrop. This ID will correspond to the ID of equipment items that follow later in the file.
This information is followed by Attack, Defence, Health Points and Speed of the mech on the second line.
On the final line after that the “strengths” and “weaknesses” of the mech are listed (if there are any, otherwise there will be an empty line).

A **Mech** is characterised by:

- A name
- A category
- A lootdrop ID
- A number of attack points
- A number of defence points
- A number of health points
- A number to indicate its speed
- Zero or more strengths
- Zero or more weaknesses

After the mechs, equipment will follow. Once again the first line of contains a number indicating the number of pieces of equipment that will follow.
On the first line you will get the name and type of the equipment, followed by an equipment ID (this will match with lootdrop IDs from the mechs).
On the line after that you will get the attack bonus or defence bonus (for offensive and defensive equipment respectively) that the equipment provides.
On the final line after that the “strengths” and “weaknesses” of the equipment are listed (if there are any, otherwise there will be an empty line).
More categories of equipment with different properties may be added later, please keep this in mind when designing your solution.

A piece of **Equipment** is characterised by:

- A name
- A category (*offensive* or *defensive*)
- An equipment ID
- An attack bonus (*offensive* equipment only)
- A defence bonus (*defensive* equipment only)
- Zero or more strengths
- Zero or more weaknesses

Next to the mechs and equipment information in the file the program should keep track of a **player**.
The **Player** is characterised by

- A number of attack points
- A number of defence points
- A number of health points
- A number to indicate their speed
- A ‘slot’ for a piece of *offensive* equipment
- A ‘slot’ for a piece of *defensive* equipment

Design and implement a program that:

- **Reads in** the file mechs.txt and has classes and structures to store and represent the information in the file.
- Sets up a **player** with standard values when the program is started. The player should start out with:
    - 3 attack points
    - 1 defence point
    - 30 health points
    - 3 speed points
    - No *offensive* or *defensive* equipment
- Has an **equals() & hashCode()** method for each class (except for the class that contains the main() method).
- Is properly **unit tested**.
- To enable user interaction, please provide a **command line interface** (reading from `System.in` and writing to `System.out`).

Combat system and lootdrops

In the game the player will "fight" against the mechs from the list. The combat system will be adapted at a later point, so that the player can make choices to influence the outcome of the fights, but for now the fights against the mechs from the file will be resolved automatically.

The combat between the player and a (single) mech will be simulated as follows:

- The player and the mech will attack each other one by one. The party with the highest number of *speed points* will start with attacking. In case of a draw the player will start.
- An attack works as follows:
    1. Take the *attack points* of the attacking party *(and add any attack bonus from offensive equipment if applicable).*
    2. If the attacking party has any *strengths* that overlap with *weaknesses* of the defending party, double the attack score (if the defending party doesn't have any matching weaknesses nothing happens).
        - Multiple overlapping strengths / weaknesses do not stack (the attack score can only be doubled once).
        - Don't forget that equipment influences which strengths and weaknesses the player has!
    3. Take the defence score of the defending party *(don't forget to add any bonus from defensive equipment if applicable).* Subtract this defence total from the attack score for the attacking party.
    4. If the remaining attack score is above 0, subtract this much *health points* from the defending party.
- The attacks will keep going back and forth until one of the party's *health point* total reaches 0.

Once the combat has ended there are two options:

1. The player won (the mechs *health points* reached 0 first). In this case:
    - Reset the players' *health point* total to what it was before the battle.
    - Put the equipment piece that corresponds with the lootdrop ID of the mech in the appropriate equipment slot of the player. Any equipment that was already in there is replaced.
2. The player lost (their health reached 0 first).
    - The players' state is reset to the default (they lose any equipment).
    - The game is over.

Combat edge case:

In the event that the player somehow has the same element (e.g. *acid*) as both a strength and a weakness due to different pieces of equipment, the effect cancels out and it should be treated as if the player has neither effect.

Menu and options that should be implemented

```
Please make your choice:
1 – Show all mechs in the system.
2 – Show player stats & equipment.
3 – Fight a mech.
4 – Write current state to file.
5 – Restore state from file.
6 – Quit the application.
```

Option 1:

Show all the mechs that have been read from the file, you do not need to print which equipment the mech drops when it is defeated. Make sure this output is in a human readable format. *This output could for instance look like (example has been shortened):*

```
Cloudwing (Category Bravo)
It has 3 attack, 1 defence, 15 health, and 4 speed.
strength(s): wind, weakness(es): poison

Howler (Category Delta)
It has 2 attack, 2 defence, 24 health, and 3 speed.
strength(s): fire, weakness(es): none.
```

Option 2:

This option should show the stats of the player, as well as their current equipment. There should be a proper way of displaying the case when they do not have any in equipment in their equipment slots (e.g. don't just print null or an empty line).

Option 3:

This option should let the player fight with the mechs following the system described in the section "*Combat system and lootdrops*".
The player will fight against the mechs from the file in order. There should be a message describing whether the player won or lost, which equipment drop they get after the fight, and how many battles they have won in total.

Example output:

```
You fight against a Howler… and win! You get a Power Armor.
You have won 2 battles.
```

If the player wins, they can pick option 3 again to fight against the next mech in the sequence and so on (if the player has defeated all mechs in the list, the next fight will be against the first mech again). If the player loses, their state is fully reset and the next time option 3 is picked, they start by fighting against the first mech again.

Option 4:

This option should write the current state of the game to a file. You may choose the filename. This information should include what equipment the player has and the number of mechs they have defeated. If this information is reimported in option 5 after the application has been restarted, the game should be playable in option 3 as if nothing had happened. You may assume that the information in the text file stays consistent (so you do not necessarily need to store all info about all mechs).

Option 5:

This option should read the information that was written to file in option 4 back into the system and enable the player to continue where they left off.

Option 6:

The application stops.

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. You can check this with **checkstyle**!
- The textual information should be read into a class containing the right attributes to store the data (so storing all information in a single `String` is not allowed, because this would hinder further development). With regard to the type of attributes used (`int`, `String`, or some other type): you decide!
- Think about your design: consider using *composition*, *inheritance*, *interfaces*, *enums*, *records*, etc.
- The **filename mechs.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 program argument).

Handing in your work

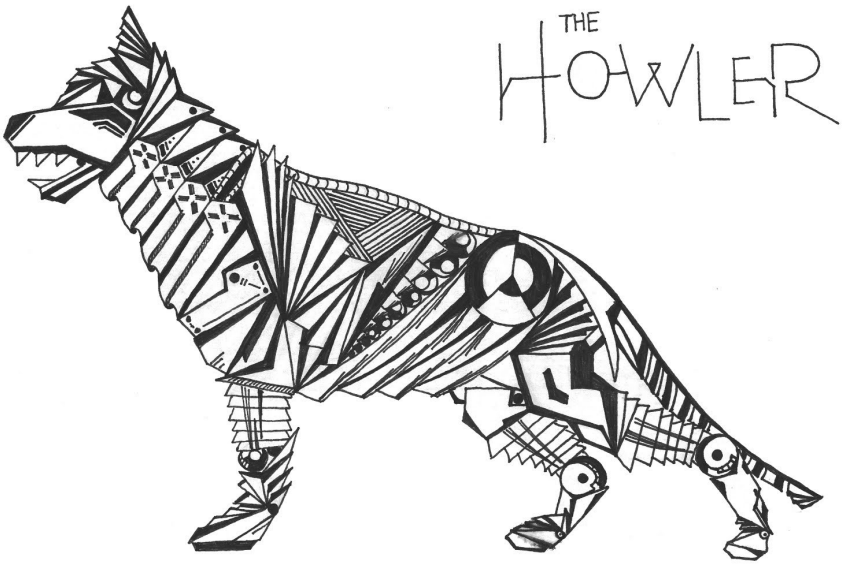
To hand in your work you must create a **ZIP** file containing your project files.

Go to your private P-drive and navigate to your project. You should see your project folder you copied before. Select this folder by left-clicking once, and the, **right**-click, hover "7Zip", and select "Add to 'Folder name'.**zip**" .

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

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.

Good Luck!



Grade composition

1.2 points **Compilation**
- If your solution does not compile your final score = 1.

1.8 points **Class Design**
- Proper use of OOP principles. Additionally, there should be a good division of logic between classes & interfaces, as well as the proper use of (non-) access modifiers (encapsulation).

0.5 points **equals() & hashCode() implementation**
- Correct implementation of equals() & hashCode() in all classes that are part of your data model (any records are exempt).

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

0.8 points **File writing** (**& rereading)**
- Being able to write the state information to a logs file. *A partially functioning writer may still give some points.*

1.5 points **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.

0.6 points **User Interface**
- Having a well-working (looping) textual interface (including option 1 and 2, which prints the mechs and player information to screen). *A partially functioning interface may still give some points.*

1.2 points **JUnit tests**
- *0.3 points* for fully testing a class related to the mechs, player or equipment (depending on how well you test, you get a score between 0.0 and 0.5)
- *0.5 points* for fully testing all other classes related to mechs, player or equipment *Hint: you can for instance test a scanner with a string instead of a file.*
- *0.4 points* for testing the combat functionality, including properly updating the player equipment after a won battle.

1.4 points **Simulating combat with the mechs**
- *0.6 points* for being able to follow the steps in the combat and calculating who wins the battle with only speed, attack, defence and health.
- *0.3 points* for taking strength and weaknesses into account in the calculation.
- *0.3 points* for giving the player the equipment they won, and applying any equipment bonusses correctly afterwards.
- *0.2 points* for printing the combat outcome correctly.

There is a 0.5 point penalty if you hardcode the filename.
There is a 0.5 point penalty if you do not hand in a proper (zip) archive.