Object Oriented Programming

CSE1100 – Programming Resit Exam

A. Zaidman / T. Overklift

Date: 25-10-2022 Duration: 2 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 Stack Overflow.
- The Java API documentation (located on the desktop of the computer).

Set up your computer

Log in using:

Username:	`EWI-CSE1100`
Password:	`Good!Luck!`

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). If you do not properly do this, you will get **a penalty**.
- **Follow the submission instructions in the last part of the exam carefully**.
- **Stop writing code 5-10 minutes before the end time** 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!

Collectable Card Clash

The company Hailstorm Entertainment that builds mobile games (for people who have phones), has noticed the recent surge in popularity of Pokémon® cards and wants to take a stab at creating a digital game with collectable cards. They ask you to create a simple setup with a collection of cards, as well as system that allows players to open cards packs. The company will work out most other gameplay mechanics at a later stage.

Specifically, they ask you to develop an application that can read in information about different (types of) cards. Below is a (shortened) example of the format (the first line of every card has been highlighted in bold, for ease of reading):

```
Unit Card
NORMAL
Knight of the Legion - 4 Energy
4 Attack – 5 Defence
Spell Card
RARE
Frost Ray - 3 Energy - Frost
Freeze all enemy units, rendering them unable to attack for 1 turn.
Weapon Card
RARE
Warhammer - 3 Energy
2 Durability
Spell Card
LEGENDARY
Heroes never die! - 6 Energy - Holy
Revive the most expensive allied unit that died this game.
Unit Card
RARE
Longbow Archer - 3 Energy
4 Attack – 2 Defence
```

The complete input file **playingcards.txt** is available in the template project.

The order of the lines of a card specification is fixed. You will always first get the type of the card on the first line. The remainder of the attributes is defined per card type below; some of the attributes are shared between different types of cards.

A **Unit Card** is characterised by:

- A rarity
- A name
- An energy cost
- An attack value
- A defence value

A **Weapon Card** is characterised by:

- A rarity
- A name
- An energy cost
- A durability value (number of times the weapon can be used before breaking)

A **Spell Card** is characterised by:

- A rarity
- A name
- An energy cost
- A spell type
- A description of the effect

Opening card packs

An important aspect of the application should be the fact that players can build a collection of cards they "own" by opening "card packs". Each card pack should contain **5 randomly selected cards** from the cards that are in the system.
The company has decided up front that **there will only be 4 different types of rarities** for the cards: **Normal**, **Rare**, **Epic** & **Legendary**.
The odds of receiving a specific rarity of card are fixed as well (all odds sum to 100%):

- Normal: 74%
- Rare: 16%
- Epic: 8%
- Legendary: 2%

The **chance of getting a card within a rarity category is equal for all cards** (e.g., if there are 10 rare cards, the chance of getting a specific card is 1.6% (16% for getting a rare card and then 10% for getting that specific card).
Tip: You can get a random number between 0 and 1 by using `Math.random()`

Building a collection

When the player opens cards packs, any cards the player does not yet “own” should be added to their collections. However, any cards the player already has in their collection are not added again (each card can only be collected once). Instead, those cards are converted to gold. Different rarity cards are worth a different amount of gold:

- Normal: 1 gold
- Rare: 2 gold
- Epic: 4 gold
- Legendary: 10 gold

The player should have a gold total (which starts at 0) that is increased whenever they earn gold.

Design and implement a program that:

- **Reads in** the file **playingcards.txt** and has classes and structures to store and represent the information in the file, and can create a “card collection”.
    - After the information from the file has been read, there should be a collection of all known cards from which “card packs” can be drafted.
    - Since reading input from a file is a slow and costly operation, ensure that your application **only reads the file once** (when the application is first started).
- Has an **equals() & hashCode()** method for each class (except for the class that contains the main() method).
- Is properly **unit tested** (100% line coverage, excluding the main method and any method that requires direct interaction with an external file (you *should* test methods that for example use a scanner)).
- To enable user interaction, please provide a **command line interface** (reading from `System.in` and writing to `System.out`). This interface should look like:

```
Please make your choice:
   1 – Show all known cards.
   2 – Show user’s card collection and gold.
   3 – Open a pack of cards.
   4 – Save collection to file.
   5 – Quit the application.
```

Option 1:

Show all the cards available in the system. This output could for instance look like (example has been shortened):

```
Unit: Knight of the Legion (NORMAL), costs 4 Energy.
4 Attack – 5 Defence

Spell: Blinding Light (EPIC), costs 3 Energy.
Offensive - Blind all enemy units, rendering them unable to attack for 1
turn.
```

Option 2:

Show all the cards the player has collected, and the amount of energy they cost. When the app is first started, the collection of the player should be empty, and they should have 0 gold. This output could for instance look like:

```
You have 4 gold and own 2 cards.

Unit: Knight of the Legion (NORMAL), costs 4 Energy.
4 Attack – 5 Defence

Spell: Blinding Light (EPIC), costs 3 Energy.
Offensive - Blind all enemy units, rendering them unable to attack for 1
turn.
```

Option 3:

A “card pack” with 5 random cards is generated and shown to the player. This option should use random chance to get cards, and award cards and/or gold to the player following the rules described in the sections (*opening card packs & building a collection*).

Option 4:

Export collection (and gold).
Write the cards in the collection of the player as well as the amount of gold they have to an external file (you may choose the name and extension of this file, and it is allowed to hardcode this information). You may use any format you want, as long as all data is preserved.

Option 5:

The application stops.

Some important things to consider for this assignment

- 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 the usefulness of applying inheritance & class composition.
- The **filename playingcards.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).
- Write unit tests!

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 your code should not generate any CheckStyle warnings. You are also expected to provide Javadoc comments.

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.

Which things will determine your grade?

For this mock exam you can get at most 0.5 bonus points. You can get these bonus points by queueing during one of the labs after the mock exam and asking a TA for feedback.
The TA will go over your submission, give feedback, and award any bonus points that you have earned.
When you ask for feedback from a TA make sure you show them **a compiling solution. If your solution does not compile you will get 0 bonus.**

You can earn the following points:

- (0.1 point) A well designed class structure for the information in the file.
- (0.1 point) Working reading from file, displaying (option 1) & writing to file.
- (0.1 point) Generating and opening card packs is fully functional.
- (0.1 point) The program is properly tested with Junit tests.
- (0.1 point) Good code quality (e.g. indentation, method length, variable naming, Javadoc).

Good Luck!