Object Oriented Programming

CSE1100 – Programming Exam

A. Zaidman / T. Overklift

Date: 03-11-2021 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 Stack Overflow.
- The Java API documentation (located on the desktop of the computer).

Set up your computer

Log in using:

Username: `EWI-CSE1100`
Password: `GLHF!`

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!

Gacha games

A company that runs a bunch of gacha (ガチャ) toy vending machines (the ones where you put in a coin and then have to turn the knob to get a random prize inside a plastic ball) wants to try their luck with a digital version of this system now that online gambling has been legalized in the Netherlands. Before they go ahead and apply for a permit they want to test of the feasibility of their plan. They ask you to create a simple implementation of such a toy vending machine that they can use to experiment with different win chances and so on.

Specifically, they ask you to develop an application that can read in the following information about prizes:

```
NORMAL PRIZE BALL [25]
White
This ball contains a normal prize from the animal range.
{Monkey, Rhino, Capibara, Ring-tailed Lemur, Otter}
NORMAL PRIZE BALL [25]
Pink
This ball contains a normal prize from the sea life range.
{Dolphin, Octopus, Oyster, Orca}
RARE PRIZE BALL [10]
Blue
This ball contains a rare prize from the animal range.
{Chimpanzee, Python, Panda, Albatross}
EPIC PRIZE BALL [5]
Purple
This ball contains some pretty epic loot from the shark and duck series.
{Hammerhead Shark, Wild Duck, Rubber Duck, Great White Shark}
LEGENDARY PRIZE BALL [1]
Orange
This ball contains an ultra fabulous animal figurine!
{Max the Marmot}
EMPTY BALL [10]
Red
Oh no! This ball is completely empty. Better luck on the next one!
EXTRA BALLS BALL [5]
Light Green
This ball is special, you get to pull a few more balls!
2
EXTRA BALLS BALL [1]
Bright Green
This ball is special, you get to pull a few more balls!
5
```

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

The order of the lines of a ball specification is fixed. You will always first get the name (type) of the ball and the amount of that type of balls that goes into the machine between the square brackets. This is followed by the color of the ball, and a flavour description. Some balls have additional info that is described below.

A **Normal/Rare/Epic/Legendary Prize Ball** is characterised by:
- A colour
- A description of the ball
- A **single** prize inside

Beware that the input from the file contains a list of prizes per ball type. However, a single prize ball can only contain one prize from the list that is provided.
Since the number of balls and prizes differ, it is possible that there are more balls than prizes. You may reuse prizes if this is the case, but please ensure that each prize appears approximately the same amount of times (small differences are acceptable, but you may not reuse a single prize for all the balls).

An **Empty Ball** is characterised by:
- A colour
- A description of the ball

An **Extra Ball Ball** is characterised by:
- A colour
- A description of the ball
- The number of extra balls that can be drawn as reward.

Design and implement a program that:
- **Reads in** the file gacha.txt and has classes and structures to store and represent the information in the file, and mimic a machine that contains these balls.
    - After the information from the file has been read, a ball machine should have been created with the contents that were stated in the file. e.g. 2x 25 normal balls from the different series, 10 rare balls, etc.
    - 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()** method for each class (except for the class that contains the main() method).
    - There should be an equals method for the class that manages contents of the machine. The contents of the machine are considered equals if the number of balls in the machine, and if the types and content of the balls are equal. However the order of the balls in the machine may be different.
- Is properly **unit tested** (at least 1 test per method, excluding the main method and any method that requires 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 balls currently in the machine.
   2 – Show current chance to win the legendary prize.
   3 – Draw a ball.
   4 – Write debug output to file.
   5 – Reset machine state.
   6 – Quit the application.
```

Option 1:

Show all the balls currently in the machine. To avoid cluttering the output, please do not include the description. This output could for instance look like (example has been shortened):

```
Normal Prize Ball (white), it contains Rhino.
Normal Prize Ball (white), it contains Otter.
Normal Prize Ball (white), it contains Rhino.
Normal Prize Ball (pink), it contains Octopus.
Legendary Prize Ball (orange), it contains Max the Marmot.
Empty ball (red), it contains... nothing.
Extra Ball Ball (light green), it gives 2 extra draws.
```

Option 2:

This option should show the user their percentage chance to win the **legendary prize**, you may round the percentage to an integer value. If the legendary prize has already been drawn from the machine, this should be stated with different message.
Tip: Since there is only one grand prize, you can use the length of the list of balls for this.

Option 3:

Allow the user to "draw" a ball out of the machine. The application should take a random ball from the list and remove it. The application should print which ball was drawn (and the contents if applicable).
Tip: Think about the event that an "extra balls ball" is drawn, the application should automatically resolve this and repeat the draw action for the bonus draws.

Below is an example of what the output could look like:

```
This ball is special, you get to pull a few more balls! You get Extra
Ball Ball (light green), it gives 2 extra draws.

This ball contains an ultra fabulous animal figurine!
You get Legendary Prize Ball (orange), it contains Max the Marmot.

Oh no! This ball is completely empty. Better luck on the next one! You
get Empty ball (red), it contains... nothing.
```

Option 4:

A log of all previous draw actions is written to the file **logs.txt**. The log should contain a chronological record of all things that happened. A single line of the log file could for instance look like:

```
1. A Normal Prize Ball (white) containing Rhino was drawn.
```

Option 5:

Reset machine state. This option should restore the contents of the machine to the original that was read from the file at the start. **Do not** read the file again to do this, but implement a different solution using built in Java functionality (e.g. methods/classes from `java.util`).

Option 6:

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.
- The **filename gacha.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 make use of code indentation, whitespaces, logical identifier names, etc. Also 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.

Good Luck!

Grade composition

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

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

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

0.8 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** (**Logging)**
- Being able to write the logged information to a logs file. *A partially functioning writer may still give some points.*

1.5 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.

0.4 points **User Interface**
- Having a well-working (looping) textual interface (including option 1, which prints the balls to screen). *A partially functioning interface may still give some points.*

0.3 points **Displaying the chance to win the legendary price.**
- *0.2 points* For displaying a correct percentage to win the legendary price.
- *0.1 points* For displaying a different message when the price has already been won.

1.5 points **JUnit tests**
- *0.5 points* for fully testing a class related to the balls (depending on how well you test, you get a score between 0.0 and 0.5)
- *0.5 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 0.5 depending on how well you test). *Hint: you can for instance test a scanner with a string instead of a file.*
- *0.5 points* for writing testing the draw functionality (and properly dealing with its randomness). Include a test that tests an “extra ball ball” being drawn (and the subsequent extra draws).

1.0 point **Drawing balls from the machine**
- *0.3 points* for being able to remove a single ball from the machine and print its information (options 3).
- *0.2 points* for the ball being selected at random.
- *0.5 points* for properly, automatically resolving the effects of drawing an “extra ball ball”.

0.5 point **Reset the machine state**
- Implement a proper reset of the machine without re-reading the input file and using built in Java functionality. *Rereading the file or manually implementing this feature (e.g. looping over all balls to add them) interface may still give some points.*

There is a 0.5 point penalty if you hardcode the filename.