Object Oriented Programming

CSE1100 – Mock Programming Exam

A. Zaidman / T. Overklift

Date: 24-10-2023 Duration: 2 hours Pages: 5

Allowed resources

During the exam you are **NOT** allowed to use books or other paper resources. The following resources are available on the exam computer:

- A 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

1. Log in to windows using:
   Username: `EWI-CSE1100`
   Password: `CSE1100Mock!`
2. 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 your final exam submission will be stored.
3. Once this is done, please open **My Computer** and navigate to the **Z-drive**, where you will find your exam template. Open the exam template folder, and then open the "OOP Exam Template" IntelliJ file. While IntelliJ starts, you can read the rest of the exam.
4. Once the project has opened you can start implementing your exam in this project! Ensure that you are working in the project on the **Z-drive**, to avoid any issues when handing in!

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 not get any points.
- **Follow the submission instructions in the last part of the exam carefully**.
- **Stop writing code a few minutes before the end** of the exam so you have time to make sure your submission compiles and to hand in the exam.
- The **grading** is explained on the last page of this exam.

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 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!
```

*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.

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

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).

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.
- Has an **equals() & hashCode()** method for each class that is used in a non-static way.
    - There should be an equals & hashcode 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. The order of the balls in the machines should be the same as well.
- 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 – 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.
```

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 total number of balls in the machine for this.

Option 3:

Allow the user to “draw” a ball out of the machine. The application should take a random ball from the machine and remove it. The application should print which ball was drawn (and the contents if applicable). Below is an example of what the output could look like:

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

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:

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).
- For a good score, your program should also work well, without exceptions.
- Ensure that you write unit tests to test your program. Your final program should have at least **80% line coverage overall, *and* at least the same number of test method as non-trivial methods.** Constructors, getters, and setters are considered trivial methods.
- Take care to have a nice **programming style**. Use CheckStyle to check this during the exam.

Handing in your work

To hand in your work you must create a **ZIP** file containing your project files. To do so, please go to the desktop and click the **"Exam Uploader"**. There you will have to fill in your student number (e.g. 5678901) and click "Upload Exam Data". When this process has completed successfully you can verify whether a zip file with your exam has appeared on the **P-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 storing and handling the information in the file.
- (0.1 point) Working reading from file, displaying (option 1) & writing to file.
- (0.1 point) Properly working logic draw balls from the machine & show the main prize chances.
- (0.1 point) The program is properly tested with Junit tests.
- (0.1 point) Good code quality, by having no CheckStyle warnings / errors.