Java Project 2021

COMP30820
20201662

Introduction

In this report I will discuss my project, how it runs, the data structures used and concepts used. Throughout I will refer to my UML diagram which has been included as an image due to legibility issues when it was imported into the document.

Start Up and Class Instances Creation

The UML document has been split into three section. On the left are the four main classes of the app, in the centre are the miscellaneous class, and in the right are the player classes, along with the leaderboard class.

To being the program run the class Start. This class's main purpose is to create an instance of the player class. The player class is an abstract class that cannot be called itself, instead start will call one of this two subclasses instead. It can call each in one of two way, as a new player or as a returning player.

I will get into more detail on the returning players later, but for the moment it uses the players method to create a ArrayList containing all the names of previous players. newName uses the ArrayList to ensure that new players don't use an existing name by checking it against the ArrayList and appending increasing numbers to it until it find a new version.

Elite players are a paid membership class. They can change their password, they loose half the points of the standard players, and start with thirteen times the amount of points. It enter this class you have to enter your credit card details which are sent to the CardTest class for verification. This has three methods for verifying the card, checkable, which checks if the passed sting contains only digits and is the correct length, prefix, which checks the initial digits of the card match what would be expected, and Luhn, which performs the Luhn text. If it passes all three of these a true boolean is returned and ElitePlayer can be called. Regardless of type, players will also need to set a password at this point.

Both class have their own constructors that call super, but also both have their own overrides. As the standard class is not allowed to change its password it overrides this setter, instead just printing a message to express this limitation. Elite, on the other hand, when it would loose point only losses half of what it should, and it overrides the setPoints method to achieve this.

You will also notice the 1 to 1 relationship Player and Leaderboard have with one another. When an instance is created, player also creates an instance of Leaderboard. While this could have been created separately, nearly everything leaderboard does relies on Player to some degree, so managing it this way made everything run much smother.

Saving

The program saves three types of files in the course of its run. One is the aforementioned player list. The second is the player data, and the third is the leaderboard. All three work rather similarly, reading and writing data to individual lines in the file so the data can be retrieved. For example, in the event of a returning player the load method in Start is called. This pulls data from the players save file, points etc., but also their password. They will only be able to create the class instance, elite or standard, if they are able to enter this correctly.

One thing I will note here is that, while a player can save at any time on the main menu, the game features an autosave. Any alteration to a class instance, such as gaining points, triggers the autosave, and point alterations also trigger a leaderboard update. Players need to be careful though. In addition to the autosave, the game also features premadeath. If a players score drops below 0 their save will be deleted and it will be game over, but their final score will live on on the leaderboard.

The leaderboard file works much the same, however reading and writing is a little more complex. The leaderboard data is stored in two ArrayLists, one for the names, the other for the points, corresponding names and points being stored at the same positions. To update a persons score they are removed from both ArrayLists then added at the end as a new person. The scores ArrayList is then iterated over backwards, comparing the new score to the person before/above them. If they are higher the scores are swapped, as are the names in the corresponding locations, and the score is compared to the next persons. This continues until the next score is higher, at which point the program iterated over them in order, writing one item from each at a time to a new line in the file. This allows names to be followed by their score in the file, supporting easy retrieval.

In all cases as these files are very important exception handling has been used throughly.

Main Menu

Once the player instance is created it is passed, along with scanner, to the class MainMenu. Here players can can choose from two games, CodeBreaker and RockPaperScissorsLizardSpock as well as a number of other options using switch cases. These include changing their password if they are elite, printing their player card (using toSting), printing the leaderboard, saving, or quitting (which performs the latter two actions as part of it).

There is a hidden easter egg on this menu, accessed by entering the Konami Code (UUDDLRLRBA). This will take the player to a new class where they can play a dangerous game of flip the point. This uses loops, getters and setters from the player class, and a random number generator which I will discuss later. Here the player can guess the outcome of the coin flip and their score modifier is doubled if they are right. If they are wrong though they could loose up to half their points, a quarter for the elite, and their modifier is halved. All points the player receives or looses are multiplied by this modifier, allowing them to gain points very quickly if they are luck, or loose them all. When exiting this easter egg the program also prints a random inspirational quote.

Games and Methods

Finally there are two games from the main menu. However, both use a third class called common a lot. This common class contains method that a number of classes use, such as random number generation, overloaded to allow a minimum number to be set in addition to a maximum, a y/n response checker that is used a lot, win and loose methods for the games, as well as a risk method to manage how much the player gambles, and a method to print the rules of a game. The games themselves use these methods a lot to function.

The overloaded RNG methods in common both make use of Math.random, although it is not the only part of the program to use Math, as the the win and loose methods make use of Math.round. The yesNo method is possible the most important though. It is used by every class as a secure way of getting answers, only allowing yes and no for answers, and handling case and some spelling errors.

Both games use loops to manage rounds and use a lot of the common methods to function. RockPaperScissorsLizardSpock is more chance based managing how much you gamble against the odds is important. Codebreaker is quite challenging, but there is a trick to it. It was also tricky to program as it uses three boolean arrays to check the players input for hits and blows. While hits are easy to calculate, complications arise for calculating the blows as a hit cannot be counted as a blow for another position. This was solved through carful comparison of these three arrays that track hits, blows and if a position was used for already.

Conclusion

I apologise that I could not go into more detail on the work here as I could easily have gone on for twice the number of words. Ultimately, I enjoyed working on this project and hoping my learning in Java has come across. Finally, I hope you have fun with the all, its games, and exploring the its features, easter egg, and small references. .