COMP30280 Java Assignment 1 report

Daniel Danev – 16302043

Introduction

This report accompanies the submitted Java files developed in the implementation of a simple games interface and associated games, played through the Java console upon launching the main file. These files are Player.java; Games.java; Leaderboard.java; Validation.java; and finally, the main interactive file, GameFile.java. These 5 files form a package, as classes/methods are shared across them.

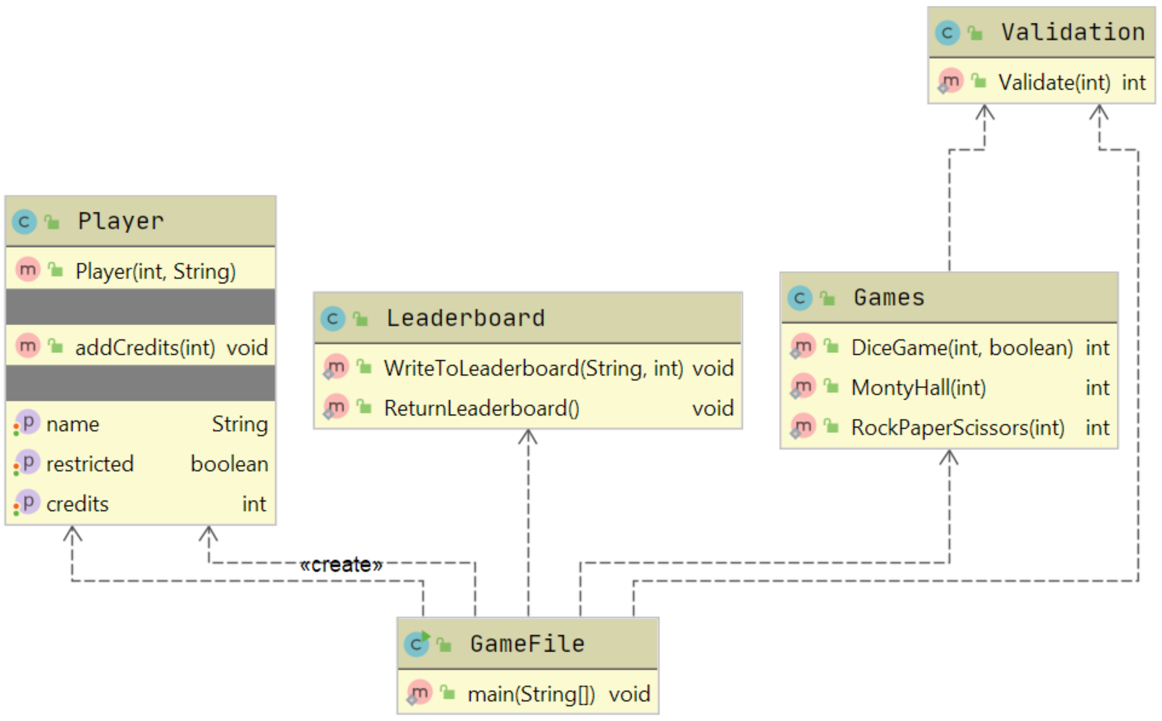
Overview

1. GameFile.java is the primary class, through which all other classes and methods are called and run. It functions as the primary user interface, calling class methods for input validation and game selection as required.
2. Games.java contains the three games available to the user. These are: A simple dice bet wager; the Monty Hall problem game; and rock paper scissors.
   In keeping with the requirements of the report, each of these three games features further interaction from the user after an initial game state is played.
   a. In the dice roll wager game, a user is asked if they wish to go “double or nothing” after an initial success, letting them play again for the chance of doubling their earnings, at the risk of losing what they won in the first roll.
   b. The Monty Hall problem is inherently interactive, as the user is prompted whether they desire to change their initial answer as the game is played.
   c. The rock-paper-scissors game is a best of three match and is played thrice every time.
3. Player.java is the class that creates player variable instances from the main menu; users must enter a player name and account type, after which an associated Player instance is created for said credentials. This class features the getters and setters necessary to view a players’ credit balance; name; account type etc., and the methods therein allow for easy credit payments and payouts. There are three player types:
   a. Normal accounts
   b. VIP accounts, that start with extra credits and get special leaderboard entries.
   c. Restricted accounts, that may only wager 5 credits at a time.
4. Leaderboard.java handles the creation and rendering of leaderboard statuses, which are saved in an accompanying “Leaderboard.txt” file. When the leaderboard is to be updated or displayed, the associated methods in the leaderboard classes perform said functions. If the leaderboard file does not exist, it will be created and written to. The leaderboard is sorted on read, meaning any number of writes can be made to it across numerous game plays and the final sorted leaderboard will always be accurate for all saved name-score combinations when displayed.
5. Validation.java is a simple class with a method designed to validate user inputs. As a significant amount of user inputs are something akin to “please enter 1,2 or 3 to select from the below choices”, it made sense to prevent tedious duplication of input validation checks by abstracting the functionality away to a class method that does the same, being easily invoked with a method call.

Use of programming concepts covered in class

1. Specialised access modifiers were utilized in the Player class, as each player object has a private name; balance; and restricted account Boolean data field. All other classes were public, as they were to be accessed readily from any instance and contained no data fields unique to each occurrence.
2. As previously mentioned, getters and setters were used with the Player class to retrieve and update/modify Player object variables as games were run. For example, when a player wishes to place a wager on a game, the getCredits() getter method is called to check whether the player has sufficient balance in their account to do so. If a player wins a game, the addCredits method is used to update their balance with their winnings.
3. Various datastructures were used throughout this assignment. The most common in a simple Java array, but Lists and ArrayLists were also used, as they offered valuable methods such as shuffling or collection sorting. A set was also utilised for quick and easy membership testing. Utilising the different datastructures available through various JDK imports simplified a great deal of work.
4. Running the project as a package simplified the project structure by allowing different project functionalities to be stored in separate files and easily called with their class/method names as required.
5. Object-oriented programming was used throughout, as evidenced by the interplay of connected classes and objects performing various desired functionalities. For example, the Player class constructor utilises a switch statement to set object variables to their input-dependent values.

UML diagram


Validation
Validate(int) int
Player
Player(int, String)
addCredits(int) void
name String
restricted boolean
credits int
Leaderboard
WriteToLeaderboard(String, int) void
ReturnLeaderboard() void
Games
DiceGame(int, boolean) int
MontyHall(int) int
RockPaperScissors(int) int
«create»
GameFile
main(String[]) void