JAVA PROGRAMMING PROJECT

Adam McCarthy

20204693

OVERVIEW

This project involved building a program that enables users to play a series of games on the console. It involves users being able to create new players by selecting their difficulty, their name, and their status. Once a player has been created, they are given the choice between three games: Blackjack, Horse Racing, and Slots. Within each game the player can bet their points on a particular outcome. At any stage they can cash in their points and solidify their place on the leaderboard. However, if they do, they lock themselves out of that player forever, and their score is saved as long as the program is running. It can be accessed via the main menu. If the user loses all of their points and bankrupts themselves, then their player is deleted and their name will not be on the leaderboard.

Upon creating a player, the user must decide between Easy/Medium/Hard difficulties. This simply refers to the amount of points that the player starts with. Additionally, they can choose between a Standard Player and a VIP Player. With VIP status, the player can utilise three powerups. These powerups provide increased odds in the player's favour for a single round. Powerups cannot be replenished, so they must be used tactically. VIP status also means the player's betting limit is increased from 1,000 points to 10,000 points. Lastly, players can only select a name that is not already on the leaderboard. This is done in an effort to differentiate between players.

It is also important to note that players can exit the game in any of the menus. If they are struggling to understand the rules, they can always refer to the instructions which can be accessed via the main menu.

BLACKJACK

Blackjack involves a player being dealt two cards upon the start of the game. These cards range from 2 to 10, with an additional 3 suit cards representing the number 10. There is also an Ace, which can represent either 1 or 11 depending on the player's total card value. The player can 'hit' to be dealt an extra card. The objective is to get as close to 21, or 'Blackjack', as possible without actually going over.

The thing that makes this game interesting is that you are playing against a dealer. The dealer will always 'hit' for another card until they reach at least 18. If the player hasn't gone 'bust', meaning they went over 21, then it is decided by examining whoever is closer to 21. Note that the dealer can go bust too. It's important to note that if both the dealer and the player get the same number, then the player's bet is returned to them.

The VIP powerup for this game is that the dealer's hit number changes from 18 to 19 for a single round. While this might result in more Blackjacks, it also means that the dealer will go bust much more than usual, which will benefit the player significantly.

HORSE RACING

This is a game where the player must bet on a 'horse' of their choice, which is represented by a line. There are 5 horses to choose from. Each horse moves a random amount every 0.2 seconds in an effort to reach a finish line. If the player's horse wins, they get five times their winnings back. However, if there is a tie, the player's bet is returned to them. What makes this game interesting is that the VIP powerup gives your chosen horse a 'boost', where their random amount's bounds are increased for a single round.

SLOTS

In Slots, the player must get matching symbols. There are 6 symbols in total that are randomly selected and placed into three sections. If the player gets a matching pair then they get double their bet. If they get all three matching, then they get 10 times their bet. The powerup for this game is interesting in that it removes a symbol from the selection process, so the player only has to worry about matching with 5 symbols instead of 6. This improves their odds significantly for a single round.

PROGRAMMING CONCEPTS

IF/ELSE IF/ELSE: Used mainly in menu functionality.

METHODS: Used to separate distinct functionalities and provide re-usability of code (for example, 'invalidArgument()').

FOR/FOREACH LOOPS: Often used to loop through arrays such as in Blackjack.

WHILE LOOPS: Often used for infinite 'while (true)' loops and then distrupted with 'break' when condition is satisfied.

ARRAYS/ARRAYLISTS: Used to hold values, for example the ArrayList 'players' stores all saved player information.

STATIC VARIABLES: Used to store objects that must be accessed and manipulated by many elements in a class.

RECURSION: Used to some degree when error checking. Users are caught in function loop until they use a valid input.

CASTING: Used to set and retrieve VIP powerups.

EXTENDING A CLASS: Used to create VIP player type.

GETTERS/SETTERS: Used when changing/accessing properties of a player.

METHOD OVERRIDING: Used for the ‘toString’ method in Player. Further overrides in VIP player to include their VIP tag on the leaderboard.

RANDOM: Used in all games (random cards/slots/speed).

SCANNER: Receives all user input.

COMPARISON OPERATORS: Most prolific in Blackjack formula to determine scores.

CALLING PUBLIC METHODS IN OTHER FILES: Used to call re-usable methods such as ‘betting()’ in the ‘App’ file.

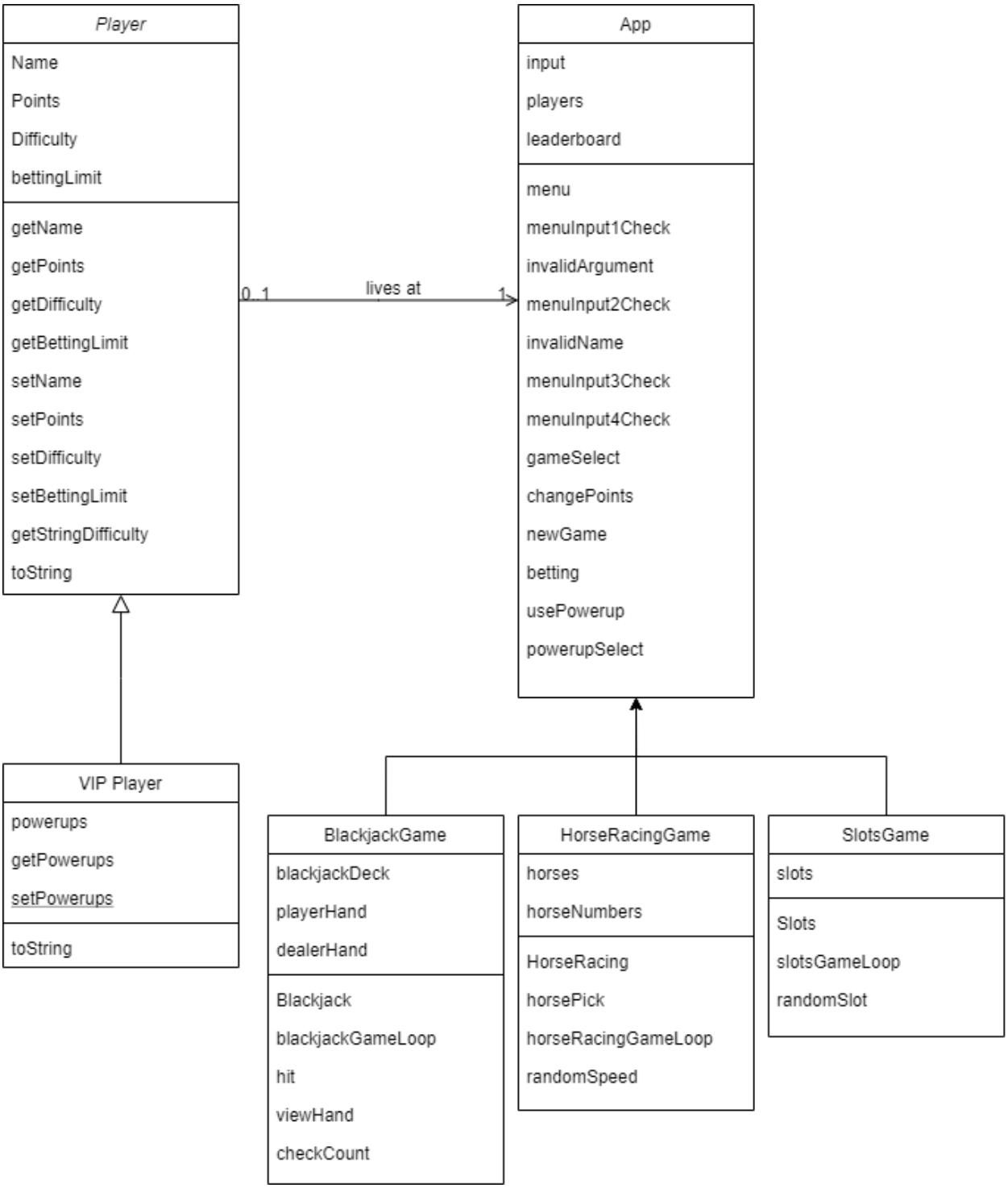
TRY/CATCH: Used when checking if input can be converted to an integer. Also used when making the program wait 0.2 seconds.

INSTANTIATION: Used to create player and VIP player instances for users to play with.

ERROR HANDLING: Prolific in menu functionality to stop invalid inputs.

COMMENTING: Used to make code more readable and understandable.

CLASS DESIGN



CLASS DESIGN DISCUSSION

It is a given that a distinct class was to be made for players. This allows the program to instantiate players and change their individual attributes accordingly. VIP players were extended from this, with the added functionality of powerups. This means that VIP players enjoy all of the attributes of a default player, while also availing of the extra functionality of powerups.

These classes then had to be hooked up to a game loop. The main game loop is called App. This class contains the main menu functionality and various generic methods with functions such as error handling, betting, and powerups. While the App class calls the player/VIP classes to instantiate, the App class is subsequently called upon by its associated games.

I found it unnecessary to extend the three games as children of the App class. Instead, they remain distinct classes that interact with the App class when necessary. These games do not interact with one another. Users are sent back to the App, and then back into another game if they wish to change games. This allows the App to essentially be a central hub of information holding the program's state, which makes data handling much more straightforward.