A Command Line Game Arcade

MISCHIEF MANAGED

Jennifer Daly

2020526

Mischief Managed – A command line game arcade

Mischief Managed is programmed using the JAVA language and makes use of the object-oriented programming (OOP) features. The arcade allows multiple players to play two games, coin toss and hangman. By playing the games the player can earn and loose points depending on their ability to win the games. The aim is to earn as many points as possible and reach the top of the leader board at the end of the session.

Mischief Managed has 8 classes associated with the application:

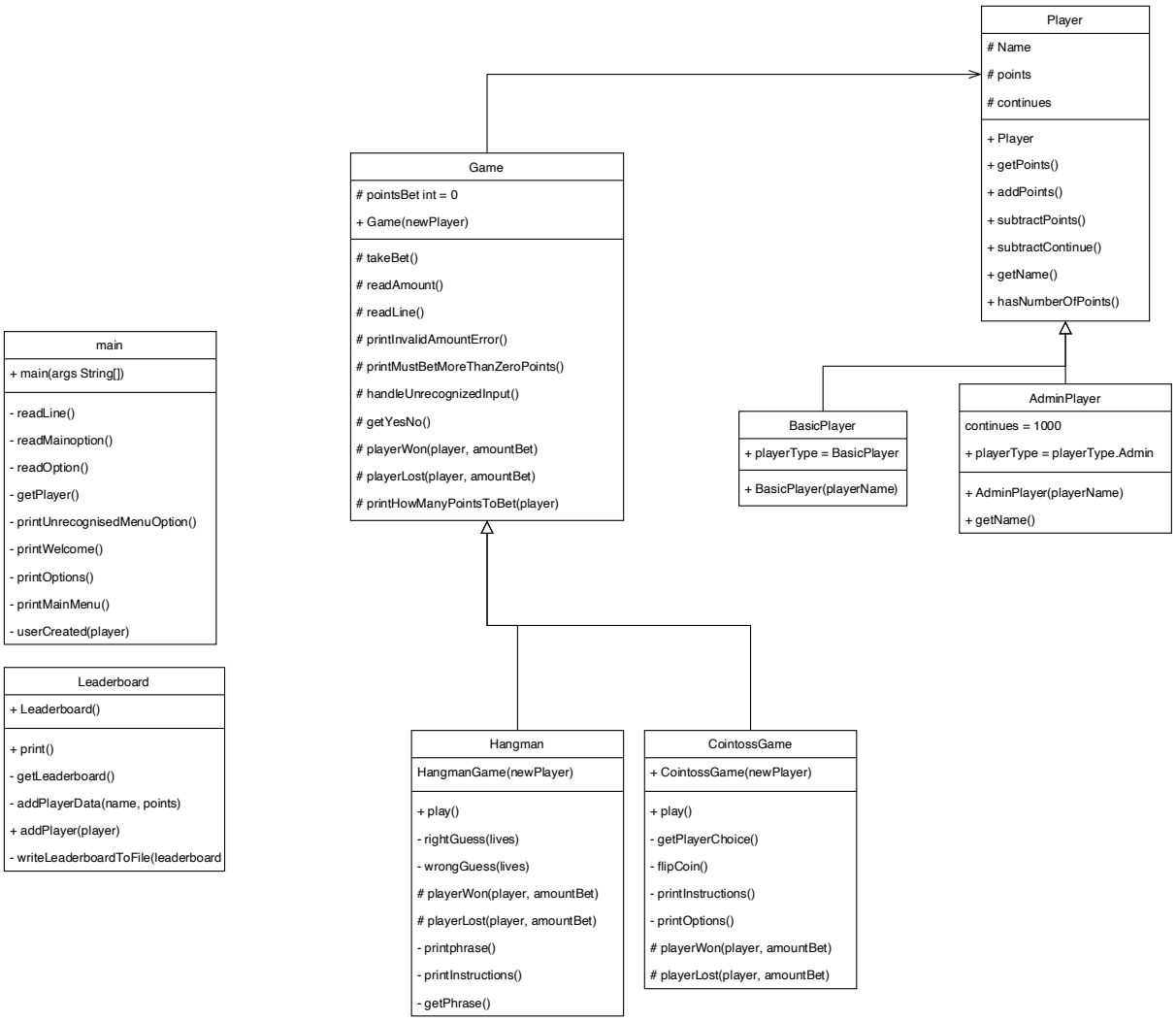
1. Main
2. Leaderboard
3. Player
4. AdminPlayer – sub class of player
5. BasicPlayer – sub class of player
6. Game
7. Cointoss – sub class of game
8. Hangman – sub class of game

The software has a specific path to follow outlined in the main class. Upon first running the application the user is greeted with a welcoming screen and asked to pick from “New Player” or “Quit”. When the user selects “New player” they are prompted to enter their name and whether they are an admin user or not. After the nature of the player has been determined, they are given the options for hangman, cointoss, see the leader board or return to main menu.

Once a game has been selected the player will be brought into the game and allowed to play until the playing conditions run out, for example if the player loses three times the game is stopped. When a game is stopped by the player or by the game, the player is presented with the game options menu again. From here they can replay the game or return to the main menu. If they elect to return the leader board will be displayed before the main menu is displayed. A new player can then start playing the games with a new name or opt to quit out of the application upon which the leader board will be displayed.

Each time a player returns to the main menu their scores are added to the leader board and it is displayed. The leader board is a persistent file and keeps track of all those who have played the application. This means that when the application is booted back up the leader board remains the same. To reset the leader board the file must be deleted.

Regarding the classes, the idea to use multiple methods in each class and to make each game a class was taken to try and simplify the program. A program with many smaller methods can be easier to read and understand than large cumbersome code. In some of the classes this process was easier than others. Most notable the player and the game class had a great result with this line of thinking whereas the leader board class had a reduced number of functions which where harder to split into smaller functions.



The above diagram details the UML diagram of the classes. The methods and constructors are shown with their access modifiers. The closed arrows indicate a subclass, and an open arrow indicates a link between the two classes, in this case an instance of a game cannot exist without an instance of a player.

Access modifiers

As seen above the application has many classes with different jobs to do. To avoid accidental changes from occurring with the classes, access modifiers were implemented in the classes. For this project, private, protected, and public modifiers where implemented. The private modifiers were used for classes where they did not have any subclasses and as such did not need to pass the information anywhere. Protected was used in the super classes where they needed to keep information private from the other classes but allow the subclasses to call the information. Public was used where methods needed to be called from outside the class or subclass. In the UML diagrams + is public, - is private and # is protected.

Getters/Setters

In this project there are a lot more getters than setters as information is typically initialised and then not changed. Getters are used to retrieve information in the different classes or within the classes to avoid calling the same information twice. An example of two getters in action in the code base is the getPoints() and getName() getter. Both methods are attributed to the player and are used to retrieve the players name and score. The Leader board class makes use of these getters to insert the player information into the leader board.

Data Structures

Throughout the application a wide variety of loops and conditional statements are implemented. In the main class there is a nested while loop structure implementing the two menus so that the player chooses when to exit the loop. The outside while loop controls the display of the main menu and the inner while loop controls the options menu for the player.

Additionally, in the main class there are nested if else if statements which control which class the player needs to access information depending in the options they choose. Although it looks complicated the structure is a great way to include decisions and options in a code base.

For loops are used in the Leader board class to iterate through the ArrayList to sort the players scores. By accessing the players at each index, the new players results can be inserted at the correct position in the array. A for loop is also used to write the ArrayList back to a file.

Alternatively, a while loop could have been used, however, in this case the for loop was more straight forward. The while loop was tasked with the job of taking the information out of the file and inserting it into the ArrayList in the first place.

BufferedReader

For this project, BufferedReader was used instead of Scanner. BufferedReader offered a new technology in Java to learn more in-depth knowledge about than discussed in the class. As discovered during research the BufferedReader is preferable when not dealing with only strings. [1] It was an interesting technology to work with and provided a good challenge. Overall, BufferedReader helped to speed up the tasks and reduced the risks of the leader board becoming too big to be read in due in part to its better multithreaded processing speeds and the larger buffer memory capacity at 8KB versus the scanners 1KB.