**Card Game**

*Student A Candidate Number: 014485*

*Student B Candidate Number: 054530*

*(Marks to be allocated 50:50)*

**Table of Contents**

[Development Log 3](#_Toc55848955)

[23/10/2020 – 12:45pm (5 hours) [*Signed: 014485, 054530]* 3](#_Toc55848956)

[25/10/2020 – 12:30pm (3 hours 30 minutes) [*Signed: 014485, 054530]* 3](#_Toc55848957)

[28/10/2020 – 12:30pm (1 hour 30 minutes) [*Signed: 014485, 054530]* 3](#_Toc55848958)

[2/11/2020 – 1:00pm (2 hours) [*Signed: 014485, 054530]* 3](#_Toc55848959)

[Design Analysis 4](#_Toc55848960)

[Test Design 5](#_Toc55848961)

# Development Log

## 23/10/2020 – 12:45pm (5 hours) [*Signed: 014485, 054530]*

Started with A as driver and B as navigator. Created *CardGame.java*, and added input for number of players and location of pack to load. Created objects for the list of cards in the pack, list of players, hand of each player, and deck of each player. Added exceptions for non-integer inputs, non-existing file inputs, and packs with invalid values or the wrong number of cards.

Swapped roles (B as driver, A as navigator), then added a method *dealCards* to deal cards to the players, and a method *countFrequencies* to check if the pack of cards is winnable. Created *Card.java* with setter and getter methods for card values and holders. Created *CardDeck.java* with getter methods for the owner, deck, deck card, and deck size, a setter method for deck card, and methods to remove from deck and add to deck.

## 25/10/2020 – 12:30pm (3 hours 30 minutes) [*Signed: 014485, 054530]*

Started with B as driver and A as navigator. Created threaded class *Player.java*, and created method *identical* to check if the player has four cards of the same value to win the game.

Swapped roles (A as driver, B as navigator). Student A optimised *Player.java* by creating method *play* for players to continually draw and discard cards until a winner is found. A new method *isWinner* was created in *CardGame.java,* replacing *identical*, to perform checks for a win condition and declare the winner of the game.

## 28/10/2020 – 12:30pm (1 hour 30 minutes) [*Signed: 014485, 054530]*

Started with A as driver and B as navigator. Student A created a new method *random* to randomise the index of which card from the deck to take from, and end the game if there are no cards left. The *play* method was updated to utilise this new random method.

## 2/11/2020 – 1:00pm (2 hours) [*Signed: 014485, 054530]*

Started with B as driver and A as navigator. Student B created the method *newStrat* to help the player decide what to do with a newly drawn card depending on cards in their hand, with auxiliary functions such as *keep*, *remove*, and *discard* to act based on the scenario.

Swapped roles (A as driver, B as navigator). Student A created a method *createFile* to create an output file for each player’s actions, and a method *writeToFile*, which writes to these files when a game action has occurred.

## 5/11/2020 – 1:00pm (4 hours) [*Signed: 014485, 054530]*

# Design Analysis

At the start of the game, the program asks for the user to input the number of players in the *main* method. It uses the *validateInput* method to check if an integer greater than two is inputted, and then it iterates this number of times to create a player object and a deck object for that player. Creating deck objects for each player makes it easier to decompose problems later when it comes to drawing and discarding cards. When loading the pack of cards in the *importPack* method, the program checks that the values are all positive integers, and that there are enough cards for the number of players. The pack of cards is implemented as an ArrayList, because this allows us to dynamically add and remove cards from the pack as cards are transferred to players’ decks.

In *dealCards*, the method keeps track of the size of players’ hands through the getter method *getHandSize* to ensure that it continues dealing directly to their hands until every player has four cards. After that, it sequentially adds cards to players’ decks with the *addToDeck* method.

The *genHashMap* method is used to generate a HashMap of key-value pairs of the cards in play, where the key represents the value of the card, and the value represents the number of these cards in the game. This is used in the *countFrequencies* method, which iterates through the players, and analyses their chances of winning based on the number of cards with their preferred value. If the player has fewer than four cards of their preferred value, then a winner isn’t guaranteed from the game, and it may stagnate.

A method *isWinner* is used to check whether a player has the appropriate cards in their hand to win the game. It iterates through their hand by combining the getter methods *getHandCard* and *getValue*. If a winner has been found, it returns the winning player.

We created a class *Player.java* to handle the decisions of each player in the game, as this helps to segregate decisions made by the game (like a referee managing the game), and the players.

To draw a card from the player’s deck, the method *draw* takes the last card in thedeck object, removes it form the deck, and then returns the card which was taken. After every draw, the player’s hand is printed using the method *seeHand*, which looks at the player object and iterates over the hand to check each value.

The method *newStrat* is used to decide what the player should do with their newly drawn card according to the scenario. If the drawn card is their preferred card, it always keeps the card, then discards the first non-preferred card in the player’s hand. Otherwise, it will perform several checks.

Firstly, it will iterate through the cards in the player’s hand; if their hand contains cards with values of other players’ preferred cards, and will discard the first one of these cards so it can make room for the newly drawn card.

Next, it will keep the newly drawn card if there are existing cards in their hand with the same value, and discard the first card in their hand which doesn’t have the same value to make room for this card.

The player will discard the newly drawn card if none of the above conditions occur.

Auxiliary functions *keep*, *remove*, and *discard* are used for the player to perform the appropriate actions according to the game scenario. The *keep* method takes the card as an argument, then adds it to the player object’s hand. The *remove* method takes the card as an argument, then removes it from the player object’s hand. The *discard* method takes the card as an argument, then discards it to the deck of the next player’s deck.

To track the activities of players as actions in the game occur, we made a method *createFile*, which creates an output .txt file for every player in the game in the format *player[NUMBER]\_output.txt*. The additional method *writeToFile* is used to write strings into the text file on new lines after an action in the game is made.

# Test Design