# Design Choices

For this coursework we coded in the following code editors: Visual Studio Code, IntelliJ and Eclipse. Eclipse was used to develop the tests, while Visual Studio Code and IntelliJ were used to work on the main program. The design has changed since our initial planning session, but the correct UML diagram can be seen in Figure 1.

## A screenshot of a computer program Description automatically generatedCardGame Class

As you can see in Figure 1, we used an ArrayList to store the player objects and cards. We decided to use ArrayList<String> to store the cards since this will be needed to get written to the files and no mathematical operations needed to get done so using a string makes the most sense. The CardGame contains the variables numberOfPlayers and deckLocation, both values for which are inputted by the user.

CardGame also contains all other classes.

Figure - Planning Diagram

## Player Class

Each object of the player class will have a playerID, a startingHand, and their cardPreference. The cardPreference of the player is the same as their playerID. The startingHand is an ArrayList object, containing strings so no conversion is needed from the Pack. Each player also has a Boolean variable to determine if they have won, the default for this value is false.

## Pack Class

The Pack contains a list of strings which will be split into the starting hands and decks. This class inherits all the other classes.

## Deck Class

The deck class is simple since each deck has its own deckID as the deckID is needed to know which deck each player will draw from and discard to. It works out that each deck will always contain 4 cards, as the number of total cards is 8 times the number of players, and each player has 4 cards in their hand at a time. Each deck object also has its own list of strings, which are the values of the cards in its hand.

# How It Works

## Inputs

When you read the code program, the file is read and the input is taken for number of players and the pack, then the error handling is applied on the pack and player to make sure that the inputs are valid. We check that the pack is not empty, that the number of cards in the pack is 8n, and that each number is a valid integer.

## Creating deck and player objects

After the inputs are taken the deck objects are created according to the input and the player objects are also created according to the input with their preferred card being their ID. The ID of the decks and players should be their preferred number, for example: first player has an ID of 1, and first deck has an ID of 1, and so on.

The hand is assigned to the players in a round robin fashion using 2 loops, for example, in a game of 3 players: Player 1 draws cards: 1,4,7,10; Player 2 draws 2,5,8,11; Player 3 draws 3,6,9,12. The decks receive their cards in a similar fashion from the cards remaining, as the decks are assigned cards after the players are.

## Threads

Each thread starts simultaneously for every player where each player has one thread. This thread draws from the deck with the same ID as the player then discards to the deck above. If there is no deck with a deck ID one above the player ID, the player discards to deck 1. If the first card is not the preferred card for the player (with ID1), the player then draws a card from deck ID1. This drawn card is placed between the first and second card in the player's hand - this is so a player does not hold on to a card indefinitely. The player at the same time discards the first card to the button of the deck. Next, the player goes on to the next card in their hand; if the card is preferred, the player goes on to the next card. This cycle repeats until a player has won. Throughout this, the thread is writing to each player’s file accordingly.

## Outputs

A file is created for each player and deck. When a player does an action it will say which deck they drew from, and which deck they discarded to. Once a player has won, this is announced in every player’s output file, this also causes the shared flag to be true and stops the game. Each player’s final hand is added to their output file. Every deck’s final contents is added to its output file.

## Known issues

The game cannot determine if a deck has no winning solution, we have not added this as it was not included in the specification. In such an instance the game would crash. The game may also crash if a player’s preferred card choice is not in the deck as they will continuously discard their whole hand. In rare situation there can be 2 or more winners. This can happen when the players draw the card they need at the exact same time, and due to threading, the game will announce both winners.

# Tests

We used JUnit 4 to develop our tests. We chose this because JUnit 4 is slightly simpler, and we didn’t believe we would need to use the more complex features of JUnit 5. In the program there is a separate file detailing the methods we have tested. Some of these we were unable to test in the same way because they are static, so to test these methods, error handling was added to the main program to make sure everything worked as intended.

In the test file you are able to specify the number of players in the game and the file location for a deck to use. This will then run the specified classes to make sure the inputs you entered are correct.

## testLoadDeckFileFormatted

This test makes sure the deck being used is formatted correctly, that for example, they are all integers, the deck is not empty, and the deck is a multiple of 8. This ensures each deck is in a valid format and means no future errors will occur.

## testNumberOfPlayers

This test makes sure the size of the pack is the number of players multiplied by 8. Also makes sure number of players is not negative.

## testSplitPack

This test makes sure that the pack is split into the same number of decks as there is players.

## testCardsInEachDeck

This test makes sure that each deck contains just four cards.

# Error handling

We have used simple error handling to ensure that the user in unable to input a number of players that is not a positive integer that is not 0, and to ensure that the deck is valid.

# Command Line Testing

We used the command line to see which part of the code runs, and to see if the code outputted the files as intended. The command line also prints the player that wins and provides information on what file was written to and when. We found this to be a very useful tool when testing the program.

# Development Log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Time Start | Time End | Roles | Signed |
| 26/10/24 | 12.30 | 14.00 | Planning and making UML diagram | 022132  030834 |
| 04/11/24 | 13:30 | 15.30 | Student:030834 coded while the other watched | 030834  022132 |
| 11/11/24 | 15:30 | 18:30 | Student:030834 coded the main system and developed some tests. | 030834  022132 |
| 12/11/24 | 19:00 | 21:00 | Student:022132 coded main system | 022132  030834 |
| 13/11/24 | 11:00 | 14:00 | Student:022132 coded main system | 022132  030834 |
| 16/11/24 | 17:30 | 20:30 | Student:030834 coded the main system | 030834  022132 |
| 24/11/24 | 17:30 | 18:30 | Student:030834 coded on the coursework and added comments. | 030834  022132 |
| 25/11/24 | 18:30 | 19:30 | Student:022132 developed tests | 022132  030834 |
| 08/12/24 | 14:30 | 15:30 | Student:022132 developed tests | 022132  030834 |
| 09/12/24 | 10:00 | 12:00 | Student:022132 worked on report | 022132  030834 |
| 09/12/24 | 13:30 | 15:30 | Final work and submission | 022132  030834 |