# Multi Threaded Card Game Report

**Amirali Famili 60%, Jamie Elder 40%**

November 28, 2023

## Introduction

This report provides a brief explanation and justification regarding the design choices made for implemented code and tests associated with the project.
This report has three main sections :

1. **Production Code's Design and Performance Issues**
2. **Test Suite's Design and Performance Issues**
3. **Developers Log**

## Production Code's Design & Performance Issues

For developing this card game a V-model approach has taken place , however we also took advantage of the waterfall design approach, by developing a step further and then testing the code that we were sure is correct in functionality and since this was a short project where all our resources were clear it made it easier for both design approaches. This project has 4 main files which we'll look into them with more detail.

### 1. CardGame.java

CardGame class is the core class of this project, it receives a list of integers representing pack and an integer representing the player number, then the constructor of this class will assign those values and deal hands to players, fill the decks with remaining cards and pass the players and decks main lists to their relevant classes.
This class is also responsible for initiating the game with a inner class inside it which extends thread, it would assign each player their own thread and uses an index to identify them.
This class can be considered to be both a builder class and a director (just like waiters in restaurants), it builds the essential components needed for the game and stores them in other classes, then it would run the game.
Hands and decks are distributed in a round robin fashion just like a poker game, the cards are dealt one at a time to each player and then the decks.
The main game has several features, startGame method inside CardGame class is responsible for running the game, it uses a ExecutorService to create a thread pool of size player numbers then it would run a loop player number times to assign each thread (or player) their own task, the run method of this inner class responsible for thread handling has a while loop and within it a playerTurn method which indicates the actions that each player should take within their own turn.
This design choice helped the developers to distinguish the thread related operations from the main class and better organise the code and offer a better understanding for other developer and observers.
For avoiding any race condition, dead locks or conflicts with threads accessing the shared resources, the code is using synchronized blocks (as well as synchronized methods in Card.java and Player.java), lock objects and wait and notify methods, to try to communicate with other threads regarding the shared resources.
One problem with this class is the win condition, although it works just fine but it came with a cost, it uses various methods to flag and communicate with other threads but after several attempts it seemed that threads tend to play their turn at least once after the winner is declared, this could have led to various bugs in the game, developers had no choice to use the system.exit key word to stop the threads from playing after the winner is declared in order to avoid having more than one winner.

### 2. Player.java

Player class is one of our two driver classes, it receives a players linked list from CardGame class and uses that list to communicate with the game and keep the list live with each hand player, this class is also responsible for every player related action, such as detecting the card that they don't need, replacing it with a new card along with other player helper methods.
The weight of the game is split between Player and Card class one holds the hands and the other holds their decks, this split was made possible by the design approach of players and decks, since the

number of nested lists inside each main list represent the number of players in the game which made this split even easier and with less overhead.

### 3. Card.java

Card class is the other driver class of this project, it receives the, decks main list from CardGame class and communicates to the game with it, it also responsible for all deck related operations, such as drawing a card from the player's left deck and discarding a card from their right deck along with helper methods for decks.

### 4. InputOutput.java

InputOutput class is responsible for all input and output management, it appears in CardGame class for capturing players initial hands, writing the player actions and finally declaring winner, informing all player and writing the deck content in their correspongin files at their final state.
This class is responsible for creating, deleting and modifying both players and decks text file, players files can be found in a directory with the name "players" and decks files are stored in "decks" directory. Although most actions made by the InputOutput class are automated by using different constructors for this class, writing players actions such as the card they draw and discard as well as their current hand is done using directly via methods of this class inside CardGame class.

### Some Known Problem

- Since this card game involves deleting, creating and modifying several files for players, it's recommended by the developer that the game shouldn't be played with more than 2000 players doing so will cause memory and cache problems on user's machine.
- As mentioned above a system.exit is used to finish the game to stop the threads from playing one more turn after the winner is declared, developers were aware that this is not a professional approach. Professionalism was sacrificed for functionality.
- When the game is played in two player mode there is a chance that the game might get caught in an infinite loop when both player are looking for the same suite of cards to win (if there is only 4 cards of the same suit in the game for example player 1 has [1,1,3,5] and player 2 has [1,1,2,4], they are both looking for 1).
- If a situation where two players win at the same time with the same hand (which should be impossible due to the nature of the game), the first occurance of the winner inside the main players list is the winner (the one with lower index).
- In the test section the program is being tested with large number of players and constantly printing through the terminal window, this might hold the program back and cause errors with tests (1 out of 10 times), however I left the printing on to demonstrate that the game is being integration tested, by commenting out the single print command in playerTurn method inside PlayerThread class.

### Overall

Threads are handled with caution in this project and almost all classes are thread-safe and most of the code is surrounded with try and catch clauses to keep the game running.

## Unit Testing & Design Choice

Unit testing the code was one of the most time consuming and challenging aspects of this project, it involved thinking about every possible input and every root that the compiler might take.
However with the help of Junit 4, this process became easier and more engaging with us, Junit 4 was used for test suits because it has one big library and it's perfect for the beginners since everything that we needed was in one place, although we could have ran our tests in parallel as well, on the other hand Junit 4 offered more simplicity for us new developers and met our requirements and also Dr. Majeed mentioned in his lecture notes that for this module it's better to use Junit 4 and we had all examples and resources that we needed there.
For the design of the tests developers decided to create 4 main test classes for every main class that we have within the project we avoided testing the inner class which holds the threads, since there is only a run method and playerTurn method inside that class (PlayerThread Class) which are working together withought any input nor variables and since this class is mainly dependant on the operations of threads and validity of players and decks list (it's in the heart of the project it either work with provided components or it does not), we thought we avoid testing it directly with it's own testClass, however we did test the main game method which includes this class in various aspects to make sure that it's working properly and is functional.
All the 4 testClasses are testing the functionality, validity and robustness of their corresponding classes, in addition there is a testRunner class which is responsible for gathering the result of all the test classes and printing the final approval in the terminal.
There are 4 main files which are operating tests on all classes :

### 1. testCardGame.java

testCardGame class is testing the CardGame class to make sure that builder methods are creating the correct format of data event with wrong values, it also test the start game method which is the main method used for running the game, it tests it for stability (changing the data whilst the game is being played) and Overload (tries to run a game with 2000 players or threads).

### 2. testPlayer.java

testPlayer class is testing the all the methods and objects of Player class with passing every wrong value imaginable to the Player method by using mock objects.

### 3. testCard.java

testCard class is testing the Card class and all it's methods and objects, similarly to testPlayer class it does this by creating mock objects with every wrong value that one can think of and passes them as arguments to check the robustness of the methods.

### 4. testInputOutput.java

testInputOutput class is testing the InputOutput class by passing wrong values as mock objects and testing the constructors of this class, testing this class involves checking the files created by the program for write type of data which has been done so by using the size of the files.

testing the InputOutput class was challenging since it involved testing the methods which required input of the user, the developers addressed this issue by placing a mock object as input value using InputStream and System.setIn, however since the methods that ask for input (number of players and the location of the pack) are constantly calling them selves until they receive the correct input it was very hard (almost impossible) to test them, thus developers have moved this test into another phase namely user acceptance phase, and tested the functionality of the methods with test methods and the validity of the methods with manually entering wrong values. Furthermore, this test class also required checking the text files of players and decks, it does this by checking if the file exists and if it contains the correct format by manually calculating the size of the created data (expected value) and checking the size of the created file (actual value).

### testRunner.java

testRunner class is used to run all the test sequential but at one place using JUnitCore.runClasses, by running the main method of this class all the test will be executed and the final result will be printed to the terminal window.
It's testing all the classes mentioned above (testCardGame.class, testPlayer.class, testCar.class and testInputOutput.class).

### Structure & Design

For the structure we tried to mimic the structure of the source code and keep it simple and to the point, however for the design of the test we specified the methods which are testing the game to run before any other method, to clarify (as mentioned) there are two test methods testing the main game one for stability and one for overload, these methods would run before any other method and basically they try to run two games in the background whilst the other test are being performed, and since the overload method will try to overload the game with 2000 players it's likely that the game will take for ever, thus testing whilst the game is being played in the background is a test it self, so it could be said that the entire 4 test files are performing one large test to show the quality of the code in other words its integration testing the system as a whole. Furthermore, for making sure that the test are valid and we are dealing with our expected values assertEquals and assertTrue methods from Junit 4 are used to make sure that we are receiving the correct data from the methods from classes.

## Final Notes

- Javadoc is used for all the methods and classes in this project to provide guidance on anyone who wishes to understand the code better and modify it for other uses.
- More information about files and guide on the files as well as how to run the tests can be found inside the readme files.
- Whilst running the tests some irrelevant strings might be printed to the terminal (for example : File players/playerN_output.txt does not exists), they mean that a game is being played in the background whilst the test are being executed (since other test are changing the behaviour of the program for their own test purposes these strings might show up).
- All the files regarding tests are stored in a directory named test.

# Developers Log

| Developer | Role | Duration (hrs) | Date - Time | Candidate Number |
|---|---|---|---|---|
| Amirali | Designer | 2hrs | 2023/10/28 - 13:00 to 15:00 | 206898 |
| Amirali | Developer/Designer | 5hrs | 2023/10/30 - 16:30 to 21:30 | 206898 |
| Amirali | Developer | 2hrs | 2023/10/31 - 10:00 to 12:00 | 206898 |
| Amirali | Designer/Developer | 1.5hrs | 2023/10/31 - 12:00 to 13:30 | 206898 |
| Amirali | Designer | 1.5hrs | 2023/11/03 - 15:00 to 16:30 | 206898 |
| Amirali | Developer | 1.5hrs | 2023/11/04 - 15:00 to 18:00 | 206898 |
| Amirali | Developer | 0.5hrs | 2023/11/05 - 18:00 to 18:30 | 206898 |
| Amirali | Developer | 1.5hrs | 2023/11/06 - 12:45 to 14:15 | 206898 |
| Amirali | Developer/Designer | 3hrs | 2023/11/09 - 15:00 to 18:00 | 206898 |
| Amirali | Developer/Designer | 2hrs | 2023/11/10 - 19:45 to 21:45 | 206898 |
| Amirali | Developer/Designer | 2hrs | 2023/11/11 - 14:00 to 16:00 | 206898 |
| Amirali | Developer/Designer | 2hrs | 2023/11/11 - 18:00 to 20:00 | 206898 |
| Amirali | Developer/Designer | 4hrs | 2023/11/12 - 11:00 to 15:00 | 206898 |
| Amirali | Developer/Designer | 3hrs | 2023/11/12 - 17:00 to 20:00 | 206898 |
| Amirali | Developer/Designer | 1.5hrs | 2023/11/13 - 11:30 to 13:00 | 206898 |
| Amirali | Developer/Designer | 2hrs | 2023/11/14 - 16:30 to 18:30 | 206898 |
| Amirali | Designer | 2.5hrs | 2023/11/15 - 19:00 to 21:30 | 206898 |
| Amirali | Developer | 2.5hrs | 2023/11/16 - 15:00 to 17:30 | 206898 |
| Amirali | Developer | 3hrs | 2023/11/17 - 16:00 to 19:00 | 206898 |
| Amirali | Designer | 0.5hrs | 2023/11/17 - 19:00 to 19:30 | 206898 |
| Amirali | Designer | 4hrs | 2023/11/18 - 10:45 to 14:45 | 206898 |
| Amirali | Designer | 2hrs | 2023/11/18 - 18:35 to 20:35 | 206898 |
| Amirali | Developer | 2.5hrs | 2023/11/19 - 18:00 to 20:35 | 206898 |
| Amirali | Developer | 1hr | 2023/11/20 - 13:00 to 14:00 | 206898 |
| Amirali | Developer | 2hrs | 2023/11/21 - 15:00 to 16:00 | 206898 |
| Amirali | Developer | 2hrs | 2023/11/22 - 18:00 to 16:00 | 206898 |
| Amirali | Developer | 2hrs | 2023/11/24 - 16:00 to 18:00 | 206898 |
| Amirali | Developer | 1.5hrs | 2023/11/24 - 21:00 to 22:00 | 206898 |
| Amirali | Developer | 2.5hrs | 2023/11/25 - 11:00 to 14:00 | 206898 |
| Amirali | Designer | 1hrs | 2023/11/26 - 15:00 to 16:00 | 206898 |
| Amirali | Designer | 2hrs | 2023/11/26 - 19:00 to 21:00 | 206898 |
| Amirali | Developer/Designer | 2hrs | 2023/11/28 - 17:00 to 19:00 | 206898 |

### Weight

As mentioned in the cover page the weight of this project is allocated as follows :

> Amirali Famili, with student number 720060845 : 60% contribution
> Jamie Elder, with student number - : 40% contribution

Resulting in a total divergence of (60:40)

# Multi Threaded Card Game Design Pattern