**Report on patience card game**

**Patience-game Table of**

Contents

[Introduction 1](#_Toc102121478)

[UML-DIAGRAM 3](#_Toc102121479)

[Design 4](#_Toc102121480)

[Testing 6](#_Toc102121481)

[Evaluation 7](#_Toc102121482)

[What remains to be done 10](#_Toc102121483)

[Mark I should get 10](#_Toc102121484)

Introduction

The game is implemented from the Game.java file. The Game.java file has a main class Game which is where the game is played.

Different libraries are imported in the Game.java file to assist in the implementation of the game. This libraries are for the Graphical, command line interfaces, accessing files and other functionalities.

Different custom classes have been created to fulfill other functionality. The getfile.java class has been created to access the graphical images of the cards and return an array list containing the string of names of the cards.

Some functions have limitations. Such as the function created to make a move can only run if the user has been dealt any cards otherwise the function should not run.

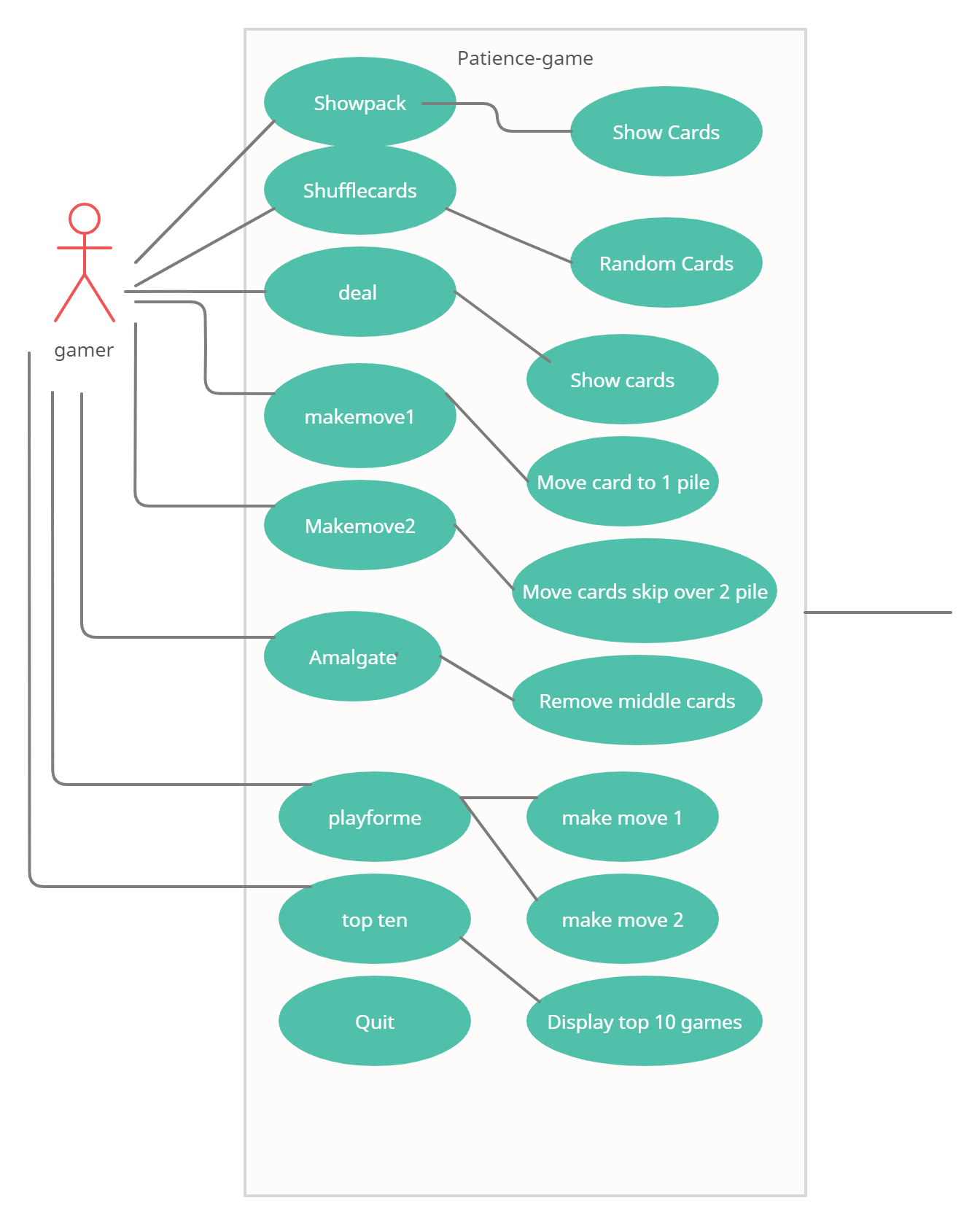
This limitations call for implementation of the WrongPlayException.java class that is called once this limitations are meant.

The menu.java file is the main entry point where the user can interact with the game.

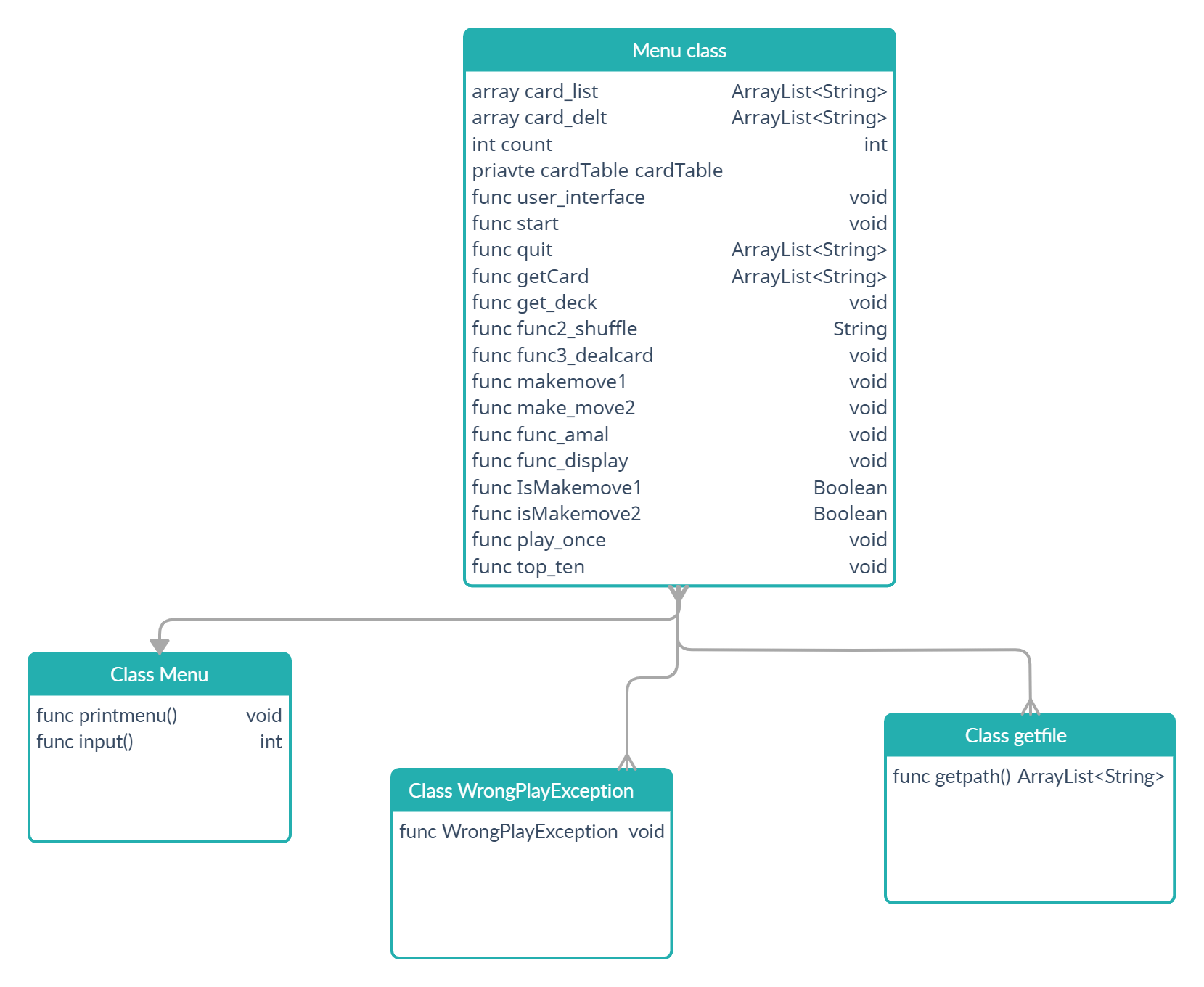
It is used to display the options the user has and accepts user input.

From the combination of the classes, a user can play patience game by interacting with the command-line and a graphical user interface.

# UML-DIAGRAM

****

Design



WrongPlayException.class

This is a public class that extends the Exception class built-in java to handle exceptions.

The class constructor receive a String parameter describing the error that is passed to the exception class to be handled by the in-built functionality of the exception class.

WrongPlayException objects are declared in the game class inside the make\_move1 and make\_move2 functions.

Getfile.class

Since the cards are provide in .gif, there is need to get all the card names and store them in array to be able to move around the cards.

This class is used to retrieve the card names for folder cards and return them as an array.Getfile objects are declared in the game class.

Menu.class

This is where the gamer can interact with the game through the command line. The user is able to view a list of possible options and can therefore make a decision and pick an option by inserting an integer value in the command line. Menu objects are declared in the game object in the constructor.

Game.class

Here we all the other classes are instantiated in this class. The logic of the game is also carried out here.

All the functions of the game to for either to move cards, view cards or close the window are defined and declared here.

It is the entry point where execution of the game begins as the **main** function is declared here.

Function amalgamate

func\_amal(ArrayList card\_delt){

Int size = card\_delt.size()

For(int i=size-1;i>0;i--){

String last\_card = card\_delt.get(i);

For(int j = size-2;j > 0;j--){

String previous\_card = card\_delt.get(j);

If(last\_card.charAt(1) == previous\_card.charAt(1)){

card\_delt.remove(i);

card\_delt.remove(j);

card\_delt.add(j,last\_card);

}

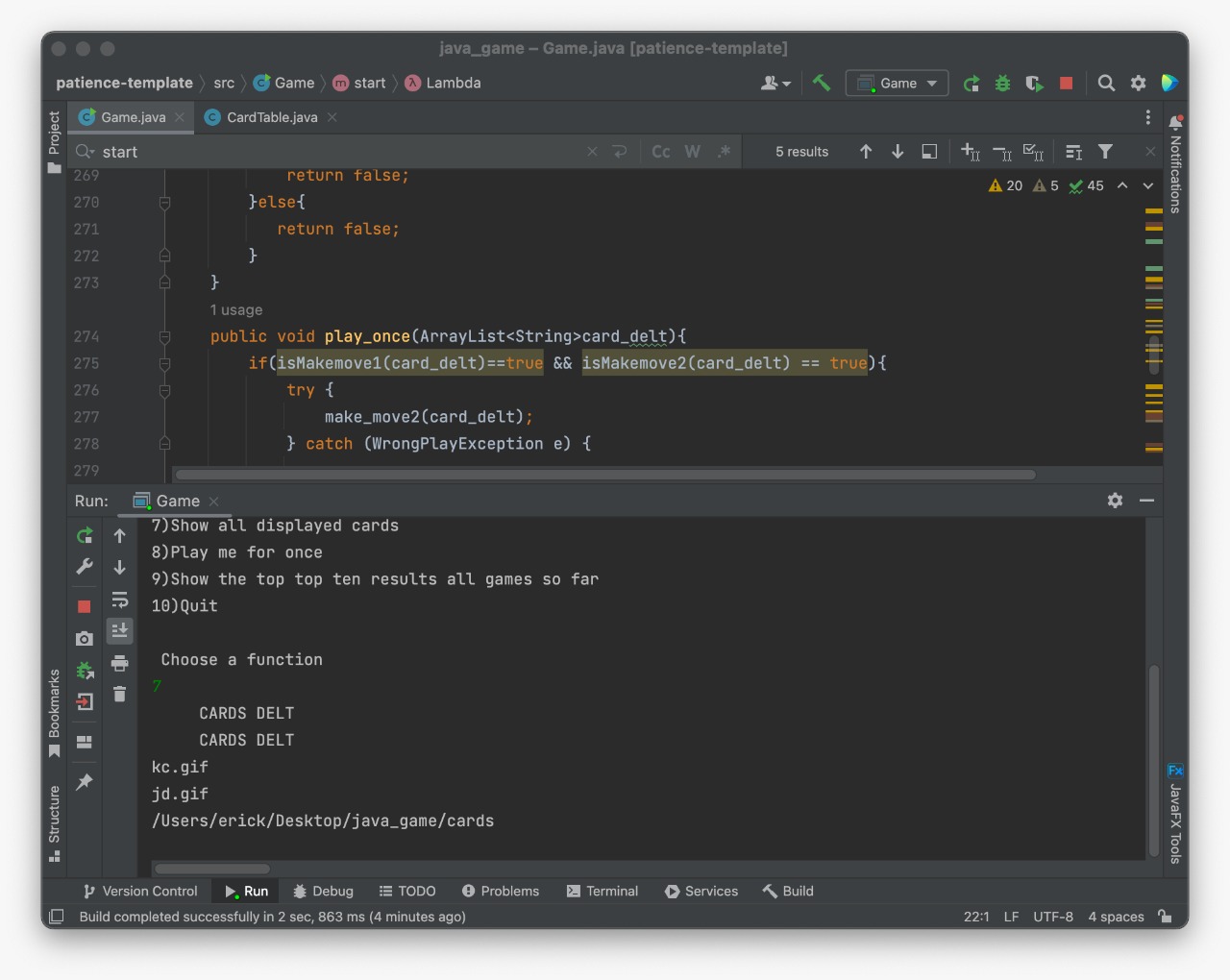
}

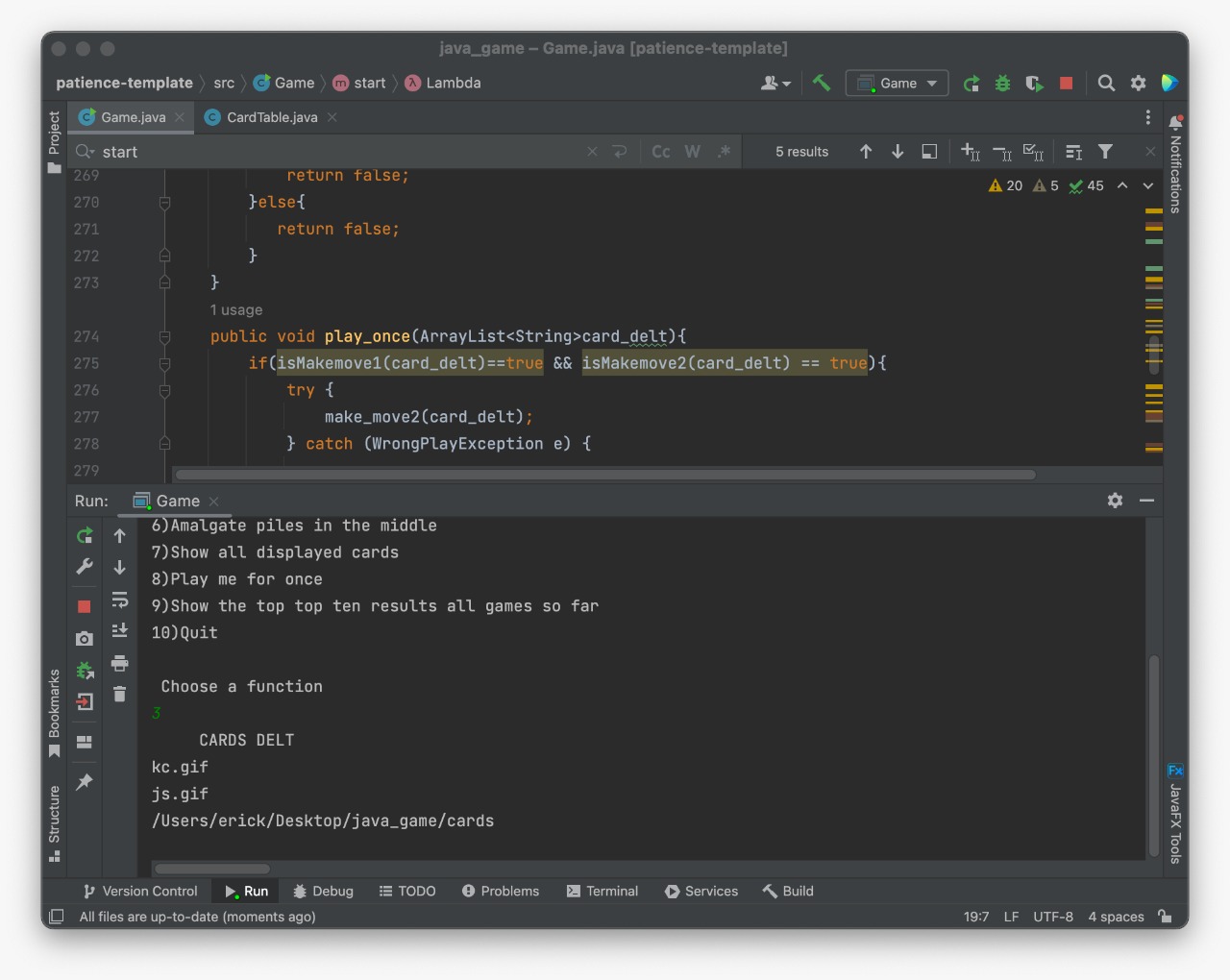
}

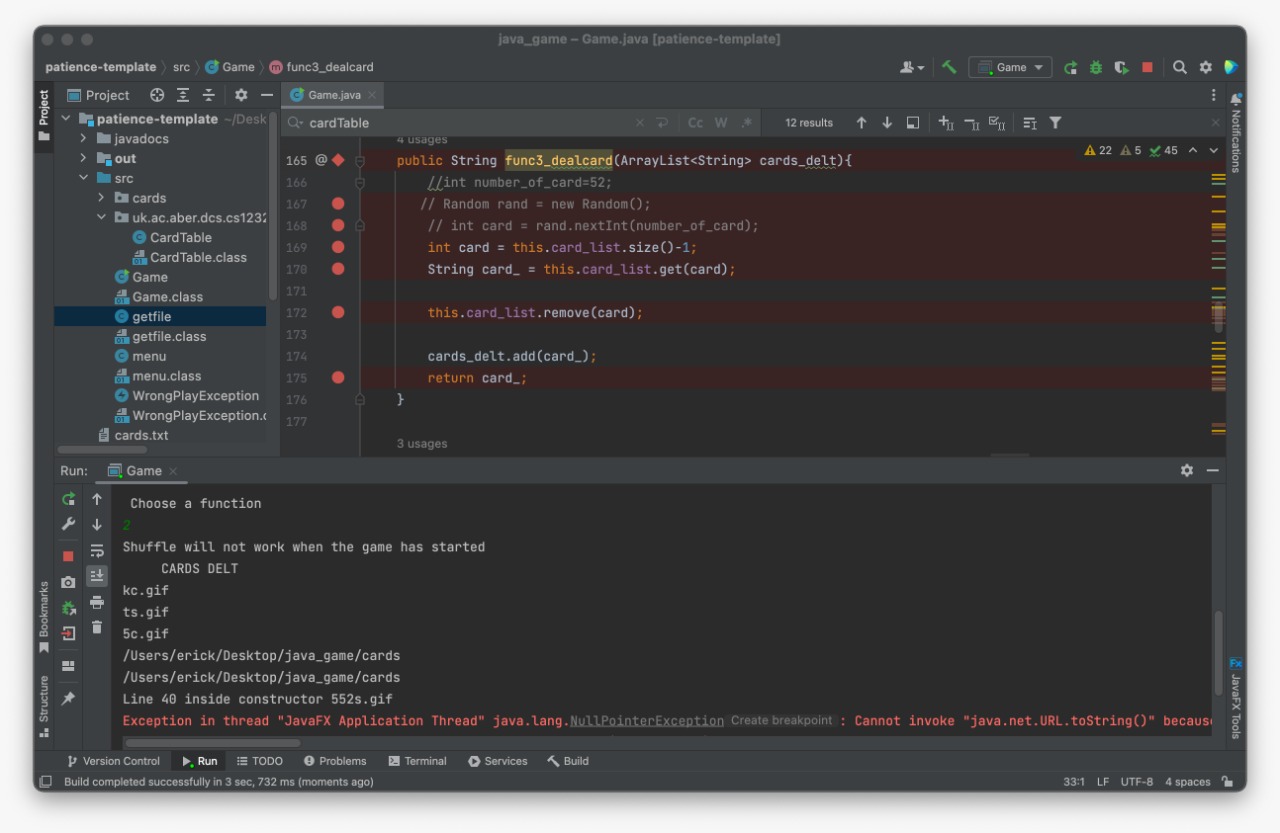
}

# Testing

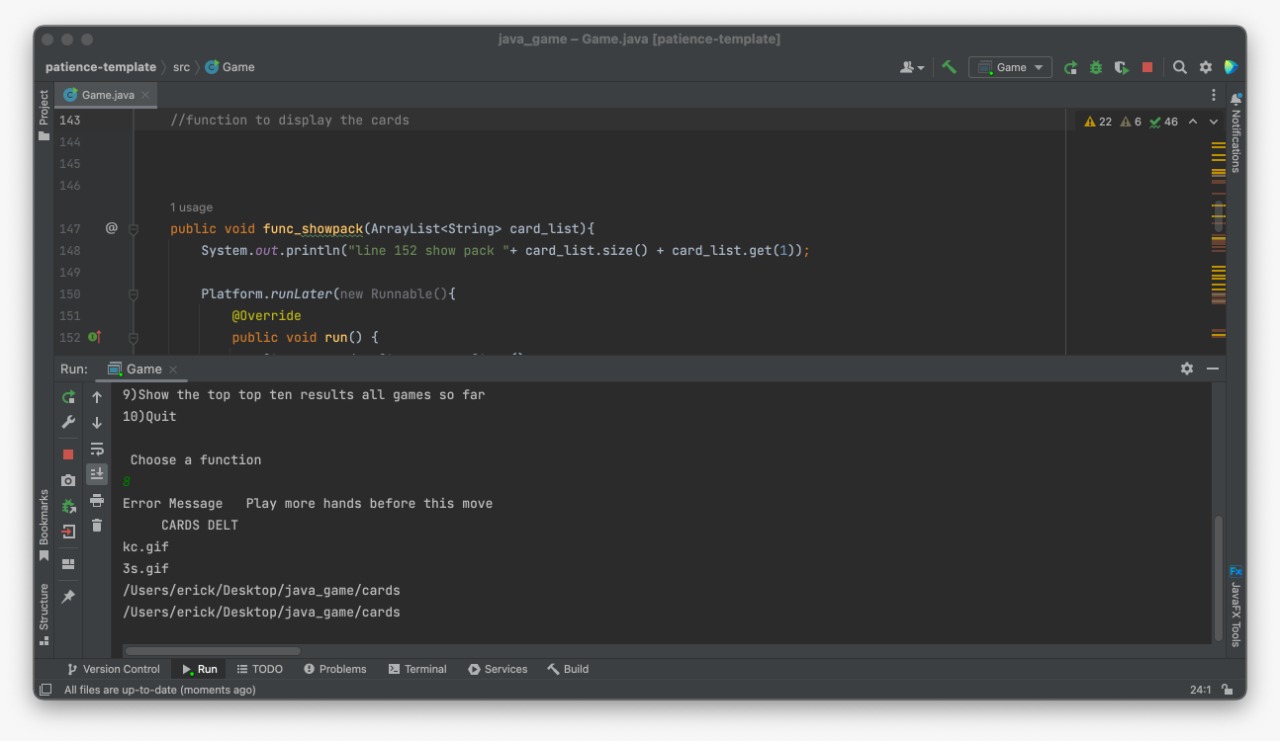
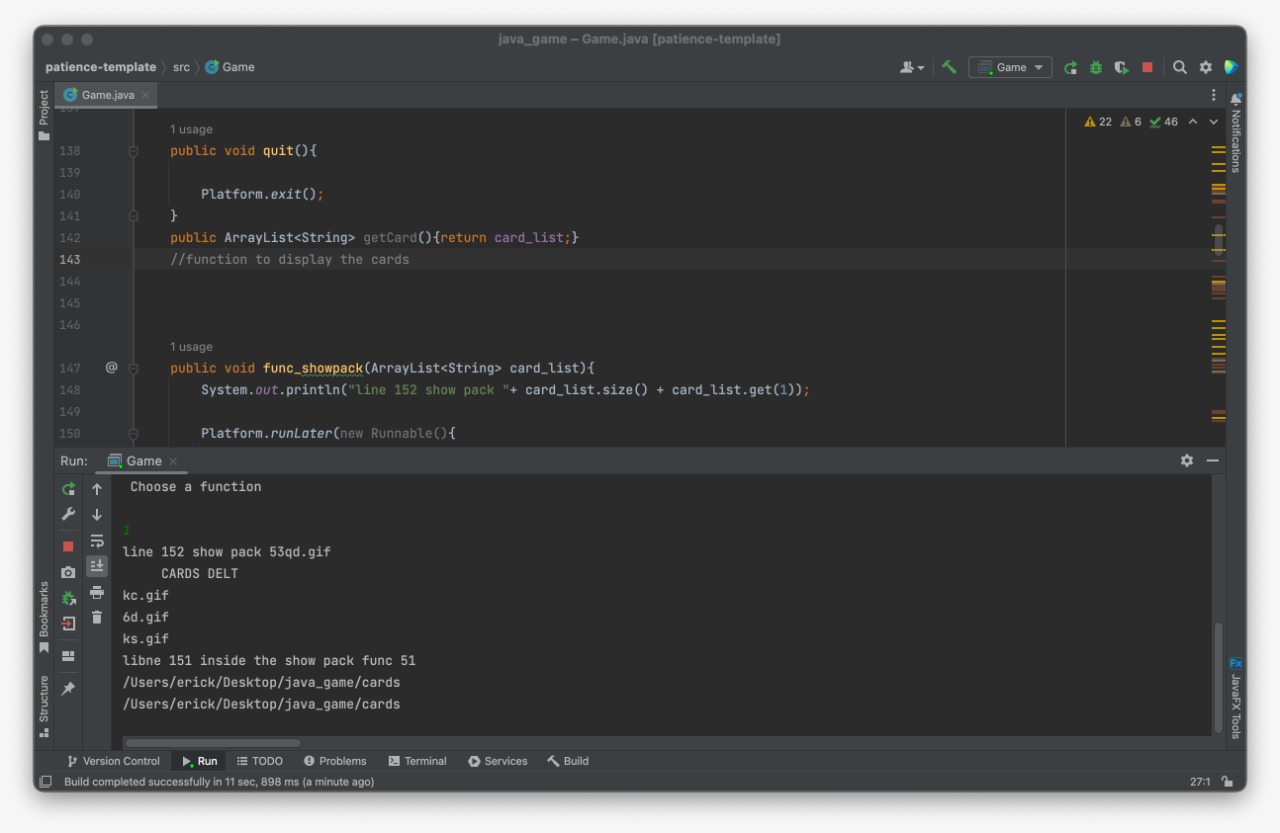
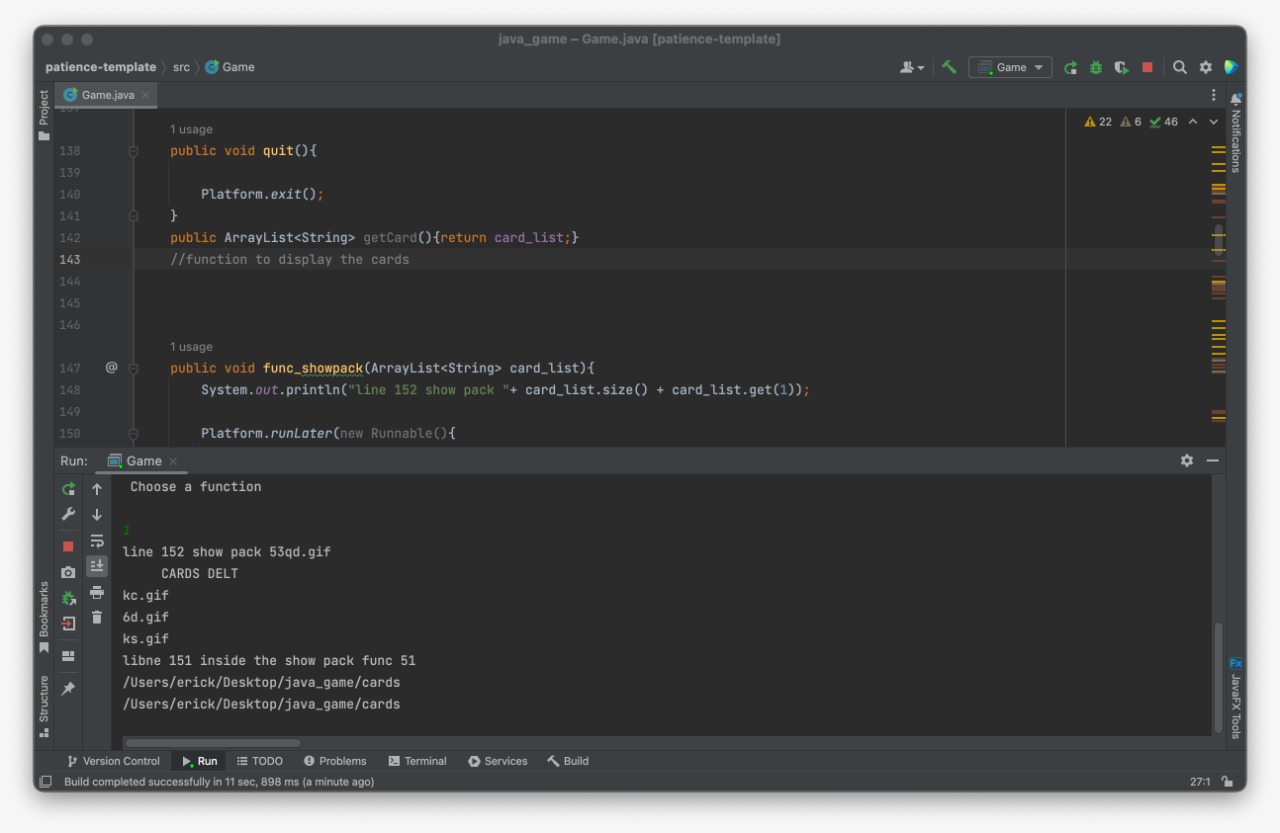
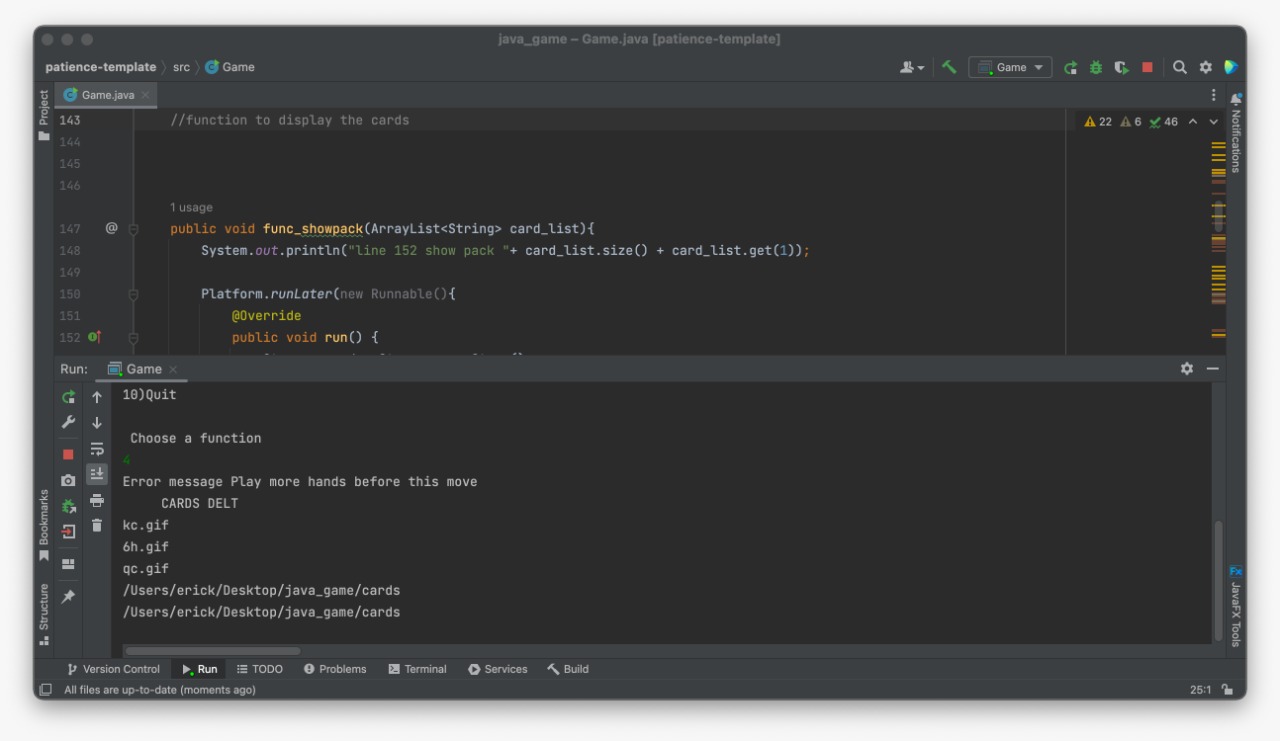
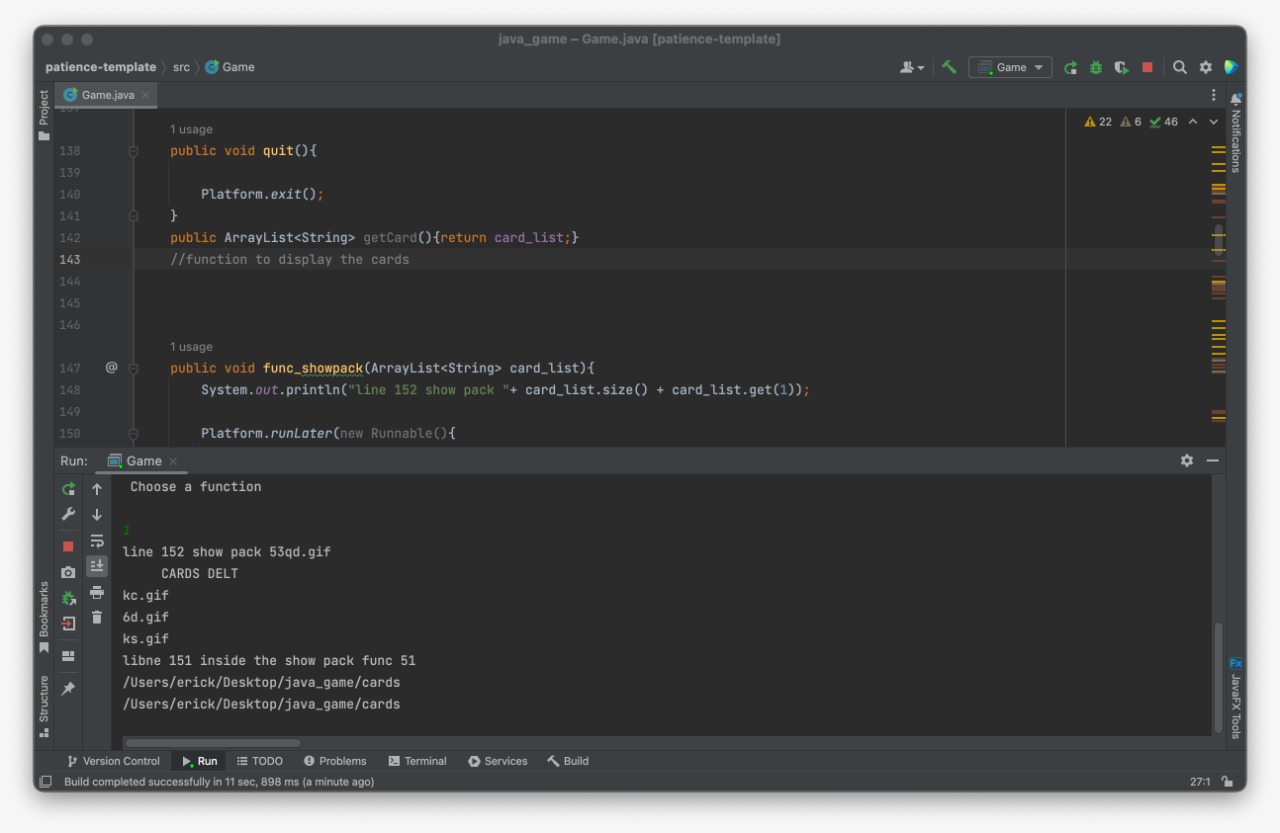
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | **Requirement** | **Description** | **Inputs** | **Expected Outputs** | **Pass/fail** | **Comments** |
| A1.1 | Show the pack | **Function to display the cards that have been dealt** | **none** | **Graphical display of array card dealt** | **pass** | **Function passed successfully** |
| FR2 | Shuffle the cards | **Function to shuffle the crds** | **none** | **Should produce an error unless the game is being instantiated for the first time** | **pass** | **Cards list was shuffled.** |
| FR3 | Deal a card | **Function to deal a card** | **Array Card list** | **Should append the array card dealt** | **pass** | **Adds a card to array list card dealt** |
| **FR4** | Make a move | **Function to pile up the deck of cards** | **Array card list** | **Should append array cards dealt** | **pass** | **Reduces the array card dealt.** |
| FR5 | Make a move 2 | **Function to move the skip over a middle pile and move to the next pile** | **Array card list** | **Should append the array card dealt ad** | **pass** | **Reduces the array card dealt** |
| **FR6** | Amalgamate | **Function to reduce the number of piles** | **Array card dealt** | **Should append the array list card dealt** | **pass** | **Reduces array card dealt** |
| **FR7** | Show all displayed cards in text | **Function to display the number of cards that have been dealt** | **none** | **Should display the cards in the function card dealt** | **Pass** | **Displays a second window of cards dealt** |
| **FR8** | Play for me once | **Function to automatically play once** | **Array Card list** | **Should automatically decide whether to make move 1 or move 2** | **Pass** | **Reduces the array list depending on the move that is available** |
| **FR9** | Show top ten results | **Show the number of games that have the least piles** | **Array games** | **When multiple games are available should show the game with the highest score** | **pass** | **Returns an array list depending on which** |
| **F10** | Quit | **Function to end game** | **none** | **Should close the window** | **pass** | **Closes the window program** |

****

****

[1.4.1](blob:https://web.whatsapp.com/32896718-e6eb-4f7e-88ed-3d150f02870e) [1.4.1](blob:https://web.whatsapp.com/0eddcbcf-6e81-48bc-b78b-dab0b2f75d56) [1.4.1](blob:https://web.whatsapp.com/32896718-e6eb-4f7e-88ed-3d150f02870e) ****

# 



# Evaluation

Firstly, I carefully read through how the game works to understand the specifications of the game and what major functions the game should have. After reading through the example given everything became clearer but since the game was new to me I kept writing down which classes I should implement and their methods and attributes.

The most important thing I discovered was that I needed to have the string names of the cards. Thus, I wrote the getfile class to access the cards iterate over them and return them as an array.

I wrote down all the requirements in the function and non-functional list. In the example provided it describes how the card names have been encoded to letter and numbers only. Such that 2H will represent 2 of hearts.

This information was very useful especially when writing functions such as make move.

In that function you need to compare card numbers and characters (suit). Since card number always come first and letters second by just selecting the string at index 0 and 1 of the card name different cards can be compared if they are equal to each other.

If they are equal to each other they form a pile (making a move).

In the game class this is where all the heavy lifting has been done. The file contains over 100 lines of code.

Under the game class first I implemented an integer, count. This is static variable meaning that it is no longer a conventional class member. It exist over multiple instances of the class.

This count variable is useful in keeping track of the number of games being played.

Afterwards, I created the array card\_list, which contains all the string names of the cards which are provided a method of the class getfile.

This card\_list is used to keep track of the available number of cards that are to be played in a game.

Another array is needed to keep track of the cards dealt and the sequence of how they were dealt.

This is important in the later stages where functions such as amalgamation and make move are implemented.

Whenever the function deal card is called the card\_dealt array must be appended.

I implemented the function user interface to instantiate the class menu. Through this function methods of menu class are called. These methods are: printmenu and input.

The printmenu function just print a list of options the user has on the console, while the input function creates an instance of class Scanner to be able to read user input from the console.

The user input will decide which function will be executed next.

Thus, inside the get user interface function there is a switch case statement that checks the user input and based on the user input a function is called.

Through javafx we’re able to create a graphical user interface where the user can view which cards have been dealt. The interface can also help the developer debug the functions called whether they behave as expected.

Thus our class extends from javafx application and overrides the function start that starts our graphical user interface.

The function shuffle cards was easy to implement as all I had to do was import function shuffle from java.collections.

Function deal a card was basically manipulating arrays. Remove a card from array cards\_list and apped the card to card\_delt. But in this function I was having an internal battle with myself. I had two methodologies of implementing the function. The first was to use random generator to give me a random index in the range of 52 cards and deal that card. The second was to remove the last card in the card\_list array and just append it to cards\_delt array.

This two approaches really messed with me. The first method of removing a card at random really seamed as a neat implementation at first glance but when I gave it a hard thought the implementation had a few flaws. E.g. if I were to remove a number randomly from the cards\_list should I reindex my whole array presuming the card I got was from the middle of the array. And if I reindex my array it means that there will be a card that will never be dealt that is in the position the same as the latest card to be dealt since random generator I had seeded it not to give duplicate numbers.

In the end I choose to go with dealing the last card in the deck since it is easiest method and most probably I was overthinking things as usual.

Then I implemented function make move to add a card to the latest pile. This function throws an error wrongPlayException when the cards dealt are less than the required minimum which is 2 cards.

Make move compares the card names and if the number or suit of the last pile and the current card match the current card is added to the pile.

Similarly make move by skipping 2 piles was implemented in the same was except that this time round it’s not the latest card but rather two steps back in the card\_delt array.

Function amalgamation was the most difficult to implement. This is mostly because I did not understand the example given on amalgamation of cards. However, I became crafty and from the example I saw a pattern. It is like there is are two loops iterating the card\_delt array from the back.

These are an inner and outer loop. The outer loop holds while the inner loop compares the value of the outer loop to itself. When a match is found a pile is formed.

From this thought process that’s how I actually implemented the function.

Function display was just a simple loop that prints out to the console the contents of the array cards\_dealt.

I created two other functions IsMakemove1 and Ismakemove2. When I was reading the requirements for function play at least once I realized I needed a Boolean expression to determine whether I can make both move 1 and move 2. If both move 1 and move 2 are possible then go for move 2 otherwise move 1.

Hence, I created isMove1 and isMove2 functions to return Boolean. That indicates which of the functions is possible thus from their result decide which move to make.

The implementation of isMakemove1 and isMakemove2 are similar in code to their parent function makemove1 and makemove2. The only difference instead of manipulating arrays inside the body of the if statements, I just returned either true or false.

When implementing these two functions I felt horrible because I had a gut feeling that there is a better way to reuse the code above instead of simply duplicating code. I’m hoping I will learn these neat way soon enough.

Then there is the first function given in the requirements list. Function show cards.

The implementation of quit method was quit easy. I just called a method in the Platform class to close the stage.

I implemented the function get top ten by first declaring a hashtable to keep track of the games that are present and the score of each game.

The score of each game is the size of the array cards\_delt. However, value of the array should be considered in a descending order. i.e. array with value 1 has the highest score.

Every time a new class is instantiated there is an array called games used to keep track of the number of games. I iterate over the games to get each game’s card\_delt array.

The size of card\_delt array is the score of the game. If array is of size one it means there is only one pile in the game and thus the user has completed the game and that game has the highest score.

What I learnt

The best thing I learnt is that I discovered the neat elegance of java its libraries, ease of packaging and generic data types.

I also got a deeper understanding of the javafx library. The terminologies used in javafx such as stage, scene and group. Different ways of creating a javafx stage such as through fxml or hard coding the components in the code as done in my project.

Modularization and OOP was not as embedded in my coding style as the functional method of coding. But after this project I know fully understand and can utilize OOP to harness its full potential and neatly arrange my work.

Java generics was an unfamiliar area before the project, but after the intensive use of arrayList I have developed an interest and come to understand how generics more intuitively. In addition, I have come to appreciate user defined data types and how to use generics and classes to implement this.

# What remains to be done

I’m yet to do the non-functional requirement of creating a text file and recording the game scores of each user.

# Mark I should get

I think I should get 85% of the total mark. I feel confident about my work because I enjoyed every aspect of this assignment and I’m hoping for many more in coming semesters structured in this manner that allows me to explore and discover new things one my own.