ECM2414 Continuous Assessment

Tom Shannon & Fedor Morgunov

50:50 Weighting

Development Log

Developer 1: Tom Shannon (candidate no: 099864)

Developer 2: Fedor Morgunov (candidate no: 186033)

Date	Time	Duration	Developer 1 role	Developer 2 role	Signatures
6/11/2023	12:25	1hr	Observer	Programmer	099864 186033
8/11/2023	12:00	1hr	programmer	Observer	099864 186033
10/11/2023	10:35	3hr	50% programmer 50% observer	50% programmer 50% observer	099864 186033
12/11/2023	2:35	1hr	Programmer	observer	099864 186033
13/11/2023	12:35	1hr	programmer	observer	099864 186033
18/11/2023	19:00	2hr	50% programmer 50% Observer (Testing Phase)	50% programmer 50% Observer (Testing Phase)	099864 186033
24/11/2023	10:25	2hr	Compilation and Documentation	Compilation and Documentation	099864 186033

Design Choices

We have chosen to take an iterative approach to the design and implementation of the production code as this will help us to ensure that the program accurately meets the specific requirements.

For the code design we have identified the following classes:

- Card
- Deck
- CardGame
- Player

The Card class is a very basic class containing the value of the card as an attribute and the instantiation method and the getter method.

The Deck class has aggregation with the Card class as it will contain a list of cards, The deck will contain simple getter and setter methods for adding and removing cards, getting the card at the top of the deck, and getting the size and deck number. The Deck class will also contain a writeToFile() method that will write the deck contents to a text file.

The CardGame Class will handle the main execution of the program, it will handle the checking of the deck input file to ensure it is the correct length for the number of players that have been selected. The class will also handle the creation of the decks and players and the starting of the threads. Once the thread execution finishes it will also call the decks writeToFile() methods.

The Player Class will implement the runnable interface as the players will need to act as threads for this game to work. During the run method, the player threads will output each action to their text file. The player class contains getters and setters for the left and right decks. The takeFromDeck() method is synchronized with the removeFromHand() method so that it avoids the threads attempting to access the same card within a deck. The takeFromDeck() method also calls the removeFromHand() method within itself so that the action can be treated as one atomic action to avoid decks and players having less than 4 cards at a time. In the run method, the checkwin() Boolean is used to identify when a thread has won, once this becomes true, the thread notifies all other threads that it has won, and all other threads will stop and write the final lines in their output file.

Known performance issues:

Currently no known performance issues.

Testing

Junit version: 5.8.1

All tests were developed alongside the implementation of the program.

The tests aim to test every part of the program that contribute to the functionality of it.

When executing .jar file make sure to specify the CardGame as the main class e.g (java -cp cards.jar CardGame)

CardTest class:

As this class is very small only one test is needed to cover the whole class, the testGetValue() test creates a card object and sets its value to 5. The expected output when the assertEquals statement is run is True.

DeckTest class:

This class consists of 4 tests for the deck class.

The first test, testSize() creates a new deck and checks that it has a size of 0. It then creates a new card and adds it to the deck (this effectively tests the addCard() method) and then tests to see if the deck size is now 1. It then removes the card (testing the removeCard() method) and tests that again the deck size is now 0. Once again a card is added and then the clearCards() method is tested to ensure it removes all cards within the deck, the expected output of this would be a deck of size 0.

The second test, testGetTopCard() creates a deck, adds a card to it and then checks that the getTopCard() method returns the card that was added previously.

The third test testAddToBottom() creates a new deck with a card already in it. It adds a new card to the bottom of the deck and then ensures that the top card is still at the top after the new card has been added.

The fourth test checks to see if the deck output file contains the correct number of lines and text that is expected once the program finishes its execution.

PlayerTest class

This class contains 4 tests that test both the public and private methods within the player class.

The first test, testPlayerInitialization() creates a new player object and assigns it the number 1, it checks to see if the player object isn't null and the index matches the value assigned to it.

The second test, testAddCardToHand() creates a player and a card and calls addCardToHand(). It then tests that the players hand has a size of 1 and checks that the card in the hand matches the value of the card that should have been added to the hand.

The third test, testSetLeftAndRightDeck() creates a player and 2 decks and then sets these decks as the left and right decks of the player. It then checks to see that the player's left and right deck match the values of the decks that should have been assigned to them.

The fourth test, testTakeFromDeckAndRemoveFromHand() uses java reflection to test the private methods from the player class. It initialises a player 2 decks and cards within the decks and player's hand. It checks to see after the takeFromDeck() method is called that the players hand remains the size of 4. As after takeFromDeck() is called removeFromHand() is called as this is treated as an atomic action. The test also checks to see that the deck to the left of the player has reduced in size after the card has been taken.

Non Junit Tests:

-CardGame test--
In order to test the functionality of the CardGame class we provide you with .txt files to test different scenarios.
The name of the text file references the number of players it's intended to handle (e.g. 11 players expect eleven_players.txt as the deck file name).

Special cases:
four_players_instantwin.txt expects four players and handles a situation where every player wins immediately.
four_players_winner_player1.txt expects four players and player1 should be the only player that can win.
fourPlayerInvalid.txt has an incorrect number of cards which should be handled by the CardGame class.

--Invalid number of players test--
Enter the number of players less than 1 or incorrect data type (not int)
Expected output: The program will warn you and ask you to enter the number of players again.

--Invalid deck test--
Enter the number of players and enter the wrong deck. For example, enter 11 players and try loading the deck from eight.txt or a file that doesn’t exist.
Expected output: The program will warn you and ask you to enter the deck file name again.