**CardGame Report**

Cover Page

Split for coursework – 60:40

Stefan Court, 229401 – 60%

James Barkes, 242734 – 40%

Development Log

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | Time | Duration | Stefan Role | James  Role | Stefan  Signature | James  Signature |
| 29/10/2022 | 18:00 | 0:45 | Driver | Observer | 229401 | 242734 |
| 30/10/2022 | 20:00 | 1:20 | Driver | Observer | 229401 | 242734 |
| 01/11/2022 | 15:30 | 0:45 | Observer | Driver | 229401 | 242734 |
| 02/11/2022 | 17:15 | 2:00 | Driver | Observer | 229401 | 242734 |
| 05/11/2022 | 17:00 | 1:30 | Observer | Driver | 229401 | 242734 |
| 06/11/2022 | 18:00 | 3:00 | Observer | Driver | 229401 | 242734 |
| 09/11/2022 | 18:00 | 2:00 | Observer | Driver | 229401 | 242734 |
| 10/11/2022 | 18:30 | 1:00 | Driver | Observer | 229401 | 242734 |
| 13/11/2022 | 19:00 | 1:20 | Driver | Observer | 229401 | 242734 |
| 15/11/2022 | 18:45 | 2:00 | Observer | Driver | 229401 | 242734 |
| 16/11/2022 | 21:15 | 2:10 | Driver | Observer | 229401 | 242734 |
| 17/11/2022 | 15:30 | 3:00 | Driver | Observer | 229401 | 242734 |
| 19/11/2022 | 13:00 | 3:30 | Observer | Driver | 229401 | 242734 |
| 21/11/2022 | 14:30 | 4:40 | Observer | Driver | 229401 | 242734 |
| 22/11/2022 | 15:00 | 3:00 | Driver | Observer | 229401 | 242734 |
| 23/11/2022 | 12:30 | 2:40 | Observer | Driver | 229401 | 242734 |

Design Choices for code

Setting up CardGame

We had to create a new Scanner object in order to be able to detect inputs from the user. This then takes in an integer value for the number of players and a string value for the location of the file of the pack. We then create the players, create the decks, load the text file into the pack, and then place the cards into the hands of the players and the decks.

Creating the player threads

In order to create the player threads a for loop was used in the CardGame class between the integer 1 and the (maximum number of players + 1), this then creates an object of type playerThread which is then the argument for each player thread. In this for loop the playerThreads are then started.

PlayerThread class

In the player thread class we have implemented the Runnable interface so that each thread has access to the function - public void run() – so that the threads are able to complete their actions. In this class we have a flag and threadNumber. Both flag and threadNumber are private to makes sure that our code is safe and threadNumber is final as it is never changed.

The constructor for each thread takes in a threadNumber to be able to tell which player the thread is running for.

For each thread the run() method will tell the thread what to execute. In our code the run() method contains a while loop for when the flag is false, this is because while the winner has not been announced, all threads will continue through the method. Also, there is an if statement to check if the winner has been announced. If this is the case, then the flag will turn true, and all other threads will cease their tasks. However, if this is not the case, each player will draw and remove a card according to their player number. These two pieces of code are separated by a sleep method which tells the thread to not do any actions for 100 milliseconds. We have written this to ensure that tasks are not carried out after the winner has been announced, because for example if the winner has been announced by thread 1, thread 2 could be about to start the task of drawCard. This causes a slight performance issue, as each thread has to sleep for 100 milliseconds after carrying out a task.

Synchronization

We have decided to synchronize addCardToDeck, removeCardFromDeck, drawCard and removeCard, this is in order to make our HashMaps thread safe, so that there are no inconsistencies with multiple threads accessing the hashmaps and adding values to a key when another thread has not finished using the original value (prevents deadlock).

Atomic Integers

We have decided to use an atomic integer for the playerWinner method in order to see if the players hand contains values of all the same card and make this class thread safe.

Design Choices for testing

CardGame – Class

In testing the first thing that should be done is creating the cardPack then creating the players and then creating the decks. This is surrounded by a try - catch statement which detects an IOException if the filename has been entered incorrectly. Before starting the threads, we should read the text file and add the cards into a pack, then place the cards into the players hands and the decks.

CardPack – Class

inputPack – Method

Here we used three try – catch statements, one to check if the file is of valid file name (FileNotFoundException), one to detect an input/output error (IOException), and one to detect if each line only contains integers in the text file (IllegalArgumentException). This method reads each line of the text file and turns each integer into a card and adds it to a pack. We also added a counter, for later use in the inputCards method, for each line that is read in the text file.

Shuffle – Class

inputCards - Method

Here we use a try – catch statement in order to check that the player number being the argument n allows for all the cards to be added to the players hands and decks. We also decided to put an IllegalArgumentException to catch if the argument of the number of players is not equal to the maxNumPlayers. Another exception we created was the IndexOutOfBoundsException this checks to see if the number of cards in card pack is equal to 8\*n players; we do this by accessing the counter from the CardPack class and checking if it is equal to 8\*n.

PlayerAction – Class

drawCard – Method

This method writes to a player(n)\_output.txt file the card that is drawn from the deck. It then adds the card to the players hand and removes the card from the deck. It is done with the strings getting added to the file first because otherwise the method getFirstCard(playerNumber) will return the card on top of the deck after the action is done which will be a different card then the one that has been already picked. We also decided to put an IllegalArgumentException to catch if the argument of the number of players is inputted negative, 0 or above the maxNumPlayers.

removeCard – Method

This method generates a random integer between 0 – 4 and checks if the card number of the players hand with index random integer is equal to the players number, this is to ensure that we don’t enter a livelock of the same card being removed, if so then this process repeats itself. Otherwise, the player removes this card to the top of the deck to the right of them. It then writes out the action of removing a card to the deck to the right of them to the player(n)\_output.txt file. This is surrounded by a try – catch statement to catch a possible IOException. It then breaks out of the while loop.

We decided to use an IllegalArgumentException to catch if the argument of the number of players is inputted negative, 0 or above the maxNumPlayers.

Winner – Class

playerWinner – Method

Here we check to see if the rest of the cards in the players hand are equal to the first card in their hand, if not the method returns false, otherwise the method writes to a player(n)\_output.txt file the player that won, the player exits, and what the players final hand is. It also writes to a deck(n)\_output.txt file the deck contents at the end of the game for each player. For the other players, a string is written to their respective player(n)\_output that the winner has informed that player that he/she has won. This is surrounded by a try – catch statement that is for an IOException. This method then returns true so that the flag in playerThread can be set to true. We decided to use an IllegalArgumentException to catch if the argument of the number of players is inputted negative, 0 or above the maxNumPlayers.

Using the test suite

For all the tests we have used JUnit 4. When using the test suite under the @BeforeClass decorator we must set the maxNumPlayers from the CardGame class, here we decided to set the number to 4. We then decided to put an input a file name in order to put the cards into pack, here we decided to create an input file of name test.txt. We then need to create the players, create the decks and input the cards from the pack into the hands of the players and the decks.

After the shuffling the players hand and decks contained the following:

Player1 hand: [1, 1, 5, 1]

Player2 hand: [2, 2, 2, 2]

Player3 hand: [3, 7, 13, 5]

Player4 hand: [6, 4, 1, 3]

Deck1 cards: [12, 8, 6, 15]

Deck2 cards: [3, 3, 1, 5]

Deck3 cards: [9, 5, 8, 2]

Deck4 cards: [4, 1, 6, 9]

We have created several tests under the @Test decorator:

* testInputPack– Here we checked if there was an FileNotFoundException/IOException was being thrown we did this by using a fake file name, faketest.txt.
* testInputCards- Here we checked if there was an IllegalArgumentException being thrown, or an IndexOutOfBoundsException being thrown.
* testCreatePlayers - Here we checked if there was an IllegalArgumentException being thrown.
* testCreateDecks – Here we checked if there was an IllegalArgumentException being thrown.
* testDrawCard – Here we checked if the number of cards in the hand of player 1 was equal to 5 and checked the exact cards within the hand according to the contents of the test.txt file. We also checked to see if an IllegalArgumentException was being thrown.
* testRemoveCard – here we checked if the number of the cards in the hand of player 1 was equal to 4 and checked that the exact card removed was not equal to a 1. We achieved this by looking at the top card of deck 2 once the card had been removed from player 1’s hand. We also checked to see if an IllegalArgumentException was being thrown.
* testWinningPlayer – Here we checked that player 2 calling winningPlayer was equal to true and player 3 winningPlayer was equal to false. We also checked to see if an IllegalArgumentException was being thrown.

For the test of testDrawCard and testRemoveCard to work, testDrawCard must be called first.