ECM2414 CA - Report

Candidate 1:088815

Candidate 2: 002690

50:50 split

Development log

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | Start time | Duration | Candidate 1 role | Candidate 2 role | Signature |
| 20/10 | 11:00 | 3hrs | Driver | Observer | 088815  002690 |
| 22/10 | 11:00 | 3hrs | Observer | Driver | 088815  002690 |
| 26/10 | 11:00 | 3hrs | Driver | Observer | 088815  002690 |
| 28/11 | 11:00 | 3hrs | Observer | Driver | 088815  002690 |
| 30/11 | 11:00 | 3hrs | Driver | Observer | 088815  002690 |
| 2/11 | 11:00 | 3hrs | Observer | Driver | 088815  002690 |
| 4/11 | 11:00 | 3hrs | Driver | Observer | 088815  002690 |
| 6/11 | 11:00 | 3hrs | Observer | Driver | 088815  002690 |
| 9/11 | 11:00 | 3hrs | Driver | Observer | 088815  002690 |
| 11/11 | 11:00 | 3hrs | Observer | Driver | 088815  002690 |

\*between session we also researched any problems we encountered in the previous session

Design

We decided to create a player object on which our threads would run. The player object would implement the runnable interface, and contain private and protected methods such as addCard, checkIfAllElementsAreEqual, isWon, returnHand and ChooseCardToRemove. Also, within the object is the overridden run method with is the method which each of the threads run. We decided that the run method should contain the full functionality so each thread could run concurrently as if it was a different player. When we tryed to get each player to write to the output files we encountered a problem as we couldn’t throw an exception within the run method, instead we used a try catch block and surrounded the body of our code in the try block.

In order to make the player class thread-safe we synchronised it to an external class called monitor, this ensures that any returned values within the thread are correct. The monitor class’ other purpose is to notify the other threads in the event of a player winning the game and if a deck is empty it will notify all other threads when a card is added to it (so it is no longer empty).

The CardGame class is our primary class as it contains our main method. The main method handles all of the user input, input validation, card distribution and thread starting. The Card game class also contains a NumToCard method which reads in the external pack from the text file and returns a list of cards using the pack, this is used in the main method to dealing out the cards to the decks and the players hands. There is also the writeFile method which creates a file of 8n non negative integers, which is then read into the code by our ReadFile method in order for the cards to be dealt. There is also an overridden writeFile method in the testing, this allows for the same pack to be generated every time during the testing instead of a random one.

We decided to implemtent an additional class called CardDeck which contained an identifying integer called deckNum and a list of cards in the deck. The deckNum value is no larger than n which is used to identify the instance of that deck. The methods within the class include the addCardToDeck which adds a card to the back of the deck when a player discards a card, and removeCardFromDeck returns the card at the front of the deck to be added to the players hand, this ensure the cards are picked in order and no cards will not be picked. The final method within the class in the toString method which is simply used to return a string of the values of the deck at end of a game for them to be added to the output files. We decide to implement card deck this way as it would allow the same instance of a deck to be passed to two different player threads, as one player will be discarding to the deck and one player will be picking from the same deck. If this was within the thread the other threads wouldn’t be able to access the decks. This overcome the problem of passing the information between threads

Our final thread-safe class we used is the Card class. This class creates an object for each class using an integer value. The Card objects are used through out the rest of the code.

The only performance issue comes when there is a pack generated without a winning state as our code would just continue, however in the specification is states this is not meant to be accounted for. When there is a valid pack, it run without any performance issues, declaring a winner and producing all the required output files.

Testing

For our testing we used a version of Junit 4 as this is what is provided with the IntelliJ ide.

Our testing covers 100% of our classes and 65% of the lines of code, this includes private methods.

CardDeck class

To test our CardDeck class we generated a list of test card which we assigned the values 1-4. We then added and removed the cards from the deck to make sure the add card and remove card methods were functioning correctly. From the tests we concluded that both of the methods run as expected.

Card class

To test the Card class, we generated a new instance of Card using the constructor card of a card number of 1, and then check to see if the card existed with the same number. This test came back correct.

These two tests were carried out first alongside their respective classes as they are two of the main building blocks of the program as we needed these to be working first before implementing the other classes.

Monitor class

Here we generated the test instance of the monitor class and checked its initial parameters to make sure isWon is initialised to false and winner is 0. We also tested the add winner method which changed the isWon value to true and the winner value to 1, indicating player 1 has won. Again, these tests passed.

Player class

Initially there were 2 issues we encountered when testing the Player class. Firstly, the complex constructor, and secondly its use of private methods. The constructor issue was overcome by creating a method called TestPlayer by creating a full instance of a player with pre-determined decks and starting hand. The second issue was solved using reflection where the private method was passed to a variable called m, where using ‘.setAccessible(true)’ allowed the subsequent test to use ‘m.invoke’ to access the method.

The first test if for the CheckIfAllElementsAreEqual method, as we created a test case where the player stated with a winning hand we just needed to check in the method returned true, which it did. Next the isWon method again should evaluate to true as the player started with a winning hand. For the ReturnHand test we created an expected hand list which each card number is evaluated individually against the test case. For the run method we simply tested that the thread was created correctly by creating a thread and then checking its running using ‘.isAlive’. All of the tests came back without any issues.

CardGame class

The first test for the CardGame class is to test whether a pack file is created therefore after the method is called there is a file called “testPack.txt”

The test for the ReadFile method checks to see if there is a pack with the name given, and should return null if the file cannot be found. In the test “errorPack.txt” was not a file and the test correctly came back null.

Finally, the NumToCard tests to see if that there are the correct number of cards for the given number of players. In our case we had 4 players therefore wee tested there was 32 cards which came back true.

As a result of doing our testing it enabled us to debug our code and made sure it was running as expected even when making changes to other parts of the code.