CM2307 – Report – C2040032

1a) Graphical user interface, text

Description automatically generated

In the above screen grab you can see that the code produces the wrong sequence number proving that the code is not thread safe.

(NOTE: this code was not written by me and was provided in the coursework accidentally. An email was sent explaining this to me stating I only needed to show proof of the provided test code working.)

1b)

Text

Description automatically generated

To make the class thread safe I added the keyword *synchronized* to the rollMe() function, this is the equivalent of the getNext() function we covered as an example in the lessons and is the element of the code where the previous value in memory is fetched. The keyword stops a race condition from occurring by synchronizing the threads access to the memory where the data is, stopping the case where the same memory is being fetched at different times of the function’s cycle which can lead to data being fetched before it has been updated.

2ai)

This program is based around playing 2 simple games involving randomness, a card game as defined in the mainCardGame() class and a die based game defined in the mainDieGame() class. The program initially runs the main() function which uses flow control and user input to select either the card or die game to be played. In the card game, it is initialised by an array of strings being used to represent every card in a deck, they are then shuffled to increase fairness of the game utilising a new “shuffled” card list. This shuffling is done by implementing the LinearCongruentialGenerator() class, which returns a random number between 0-1 which is multiplied by 52 to map it to the deck of card indexes. Users are then asked to draw 2 cards, each using the random function to select which card is taken. If either of these cards are an Ace then the player has won the game and the program finishes, this is done using the declareCardGameWinner() method. In the die game the underlying principles are very similar. Instead of the cards array being used we can just use the random number generator and multiply it by 6 and then convert to an integer to have our resultant die roll. We then have a declareDieGameWinner() methods which tests for one of the dies rolled being a 1, if this occurs a winning message is printed and the program finishes being executed.

2aii)

One of the first issues I found with the code provided is the size of the main Game class. It contains nearly all of functions and methods as well as variables that the program requires. This is the case even though the die and card games are relatively similar, they require multiple methods of their own within the main class. This can be seen as a singleton design pattern that is has a large amount of coupling between methods within the class. This means that the games themselves are not modelled around the object orientated framework that java is built around. The use of the linear congruential generator was good however if it was entered as a non-static variable then there would have been more ability to use the random class as a module.

2bi)

The classes that I have decided to improve are

Game – this class will be split into 2 new classes CardGame and DieGame. This will also require the creation of a GameInterface class to allow for the abstraction of those classes. This follows the design pattern of a factory framework.

I will also create a new class named UserInput, this is for passing the userinput to create an instance of the game they want to play and starts playing the game using the playGame function.

In the CardGame and DieGame Classes I have added a play game function to sequentially start each of the methods of the game.

I have also implemented the ThreadSafeDie class to allow for thread safe running of the die rolling in the DieGame

2bii) Diagram

Description automatically generated

2biii) – See code implementation files