**ECM 2414 Software Development – Pair Coursework**

**Cover Page**

|  |  |  |
| --- | --- | --- |
| Student ID | 700047391 | 700074574 |
| Weightage |  |  |

**Development Log**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Date** | **Time** | **Duration** | **700047391** | **700074574** | **Signed by** | |
| 16/10/21 | 02.45pm | 0.5hr | Discussion | |  |
| 25/10/21 | 05.00pm | 1.5hr | Discussion | |  |
| 26/10/21 | - | - | Creation of this document | N/A |  | |
| 26/10/21 | 01.30am | 1hr | - | PebblesGame |  | |
| 02/11/21 | 01.00pm | 2hr | PebblesGame |  |  | |
| 03/11/21 | 07.45pm | 1hr |  | PebblesGame |  | |
| 07/11/21 | 02.45pm | 2.5hr |  | PebblesGame |  | |
| 07/11/21 | 11.00am | 2hr | Discussion | |  |
| 08/11/21 | 02.00pm | 5hr |  | PebblesGame  Bag |  | |
| 09/11/21 | 07.30pm | 1hr |  | PebblesGame |  | |
| 10/11/21 | 10.00pm | 2hr |  | PebblesGame |  | |
| 11/11/21 | 10.00pm | 2hr | PebblesGame | PebblesGame |  | |
| 13/11/21 | 02.00pm | 1hr | PebblesGame | PebblesGame  Bag |  | |
| 14/11/21 | 10.00 pm | 10 min |  | - |  | |

**Production Code**

**Design Choice**

Our initial goal was to fully appreciate the requirements of the Pebbles Game written in the specification and to fulfil that, we decided to highlight the key notes of this multi-threaded game and choose the structure of this project. We had started off with the creation of PebblesGame.java where we planned to add the main method of the game. Followed by that, we also created Player.java, BlackBag.java & WhiteBag.java – however we ended up re-reading the specification just to notice that the Player class should be nested within PebblesGame.java. We were unsure of having BlackBag.java or WhiteBag.java as 2 separate files but ended up deciding that Bag.java would be better for the assembly of the program.

**Pebbles Game**

The PebblesGame.java file consists of 2 classes – the Pebbles Game class and the nested Player class. We commenced the main method by displaying the welcome message at the beginning of the game and asking the user to enter the number of players in the game by the aid of the scanner. The **getTotalPlayers(Scanner intro)** method ensures that the input from the user would only continue to the next method after being checked if the input is valid by the following criteria:

1. If the input contains “E” or “e” – the program would understand that the user wants to exit and would exit the game.
2. The number of players must be a positive integer and also greater than 1 since this game cannot be played by a single player.

If these following criteria are not satisfied, the method would return an error, inform the user and ask for an input again. After this step of verification, the program proceeds to check the file names being inputted (including the extension .txt or .csv file) by the user by calling the **validateFileName(Scanner intro)** method. This is similar to the **getTotalPlayers()** method if the input contains “E” or “e”, the program would end the game and it will also ensure that the input is a valid file and has a valid directory.

If these conditions are true, the program will proceed to the place the location of the bag into the array locationBag, which will be used later on in the game. At first, we created 3 different methods of validating each file name separately but decided it would be more efficient and time-saving to introduce a for-loop and use the same method for all 3 file names being inputted by the user. When all the black bags are successfully validated, the program would create 3 different files named WhiteBagA.csv, WhiteBagB.csv, WhiteBagC.csv.

// xxx.startGame() is where we initiate the threads and run

Initially we decided that the Player class would be extended by the Runnable class, however we ended up choosing the extension by the Thread Class – as this was preferred over the other.// come back on this again

The nested Player class contains the **playerTakesTenPebbles()** method where the random object chooses a random bag and places the pebble into that selected bag. Originally, the method had been synchronized but we realised that since a specific bag had not been selected and the selection was random, it would have been unnecessary to do so as there was no thread inferences occurring. Within the same method, the pebble would be added to that specific player’s hand and would update the log of the latest pebble being added which was mentioned in the specification.

The **discard()** method also uses the random object to remove a random pebble would also be placed into the pair of the random pebble’s black bag(white bag).Similar to the **playerTakesTenPebbles()** method, this would also be added to the player’s log and would be shown in their file.

Next, we synchronized the **pebbleSum()** method as the sum of pebbles which only should be accessed by one thread at a time since we want to know the sum of an individual’s rather than multiple players at a time.

Followed by the **checkWinningHand(ArrayList<Integer>pebbleHand, int playerId)** method – which also is synchronized and a Boolean method. This method would check if the sum of ten pebbles is hundred. If true, the winner of the game would be announced to all the players to prevent from the game continuing(immediately putting a halt to the other running threads).

As the name of the method suggests, **updateLog(String information)** adds in details whenever a player has discarded a pebble or drawn one and showing the updated pebbles left with the player. We have used BufferedWriter as buffered streams are synchronous – allowing us to implement this for thread safety.

The final method in this class is the run() method, which contains a for-loop to initially select 10 pebbles using the **playerTakesTenPebbles()**method and the players would continue playing by discarding and picking another pebble until a player’s total pebble weight sums up to 100.

**Bag**

The Bag class contains a synchronized method **selectPebble()**to ensure that no conflicts arise during the game where parallel-running threads try to access or change a variable simultaneously.

The last method present in the Bag class is the **checkBagExchangeNeeded(Bag blackBag, Bag whiteBag)** method. Since it involves the use of bags, it is important to note that this method is synchronous as well, avoiding thread interference. The purpose of this method is to ensure that the black bag does not remain empty at any point of the game by adding all the pebbles present in the pair of the white bag.

A document detailing the design choice and reasons with respect to your tests of your production code. You may use either of the JUnit 4.x or 5.x frameworks, but you should explicitly detail which framework you are using in your document. This part of the document should be no more than three sides of A4.