ECM2414 CODING REPORT

50:50 split

Date	Time	700031791	700037512
22/10/2021	9:45 – 11:10 1 hour 25 mins	Design decisions & began writing unit test	Design decisions & began writing unit test
25/10/2021	10:30 – 11:30 1 hour	Driver for testing bags then switched for bag creation functionality	Navigator for testing bags then switched for production code
25/10/2021	12:00 – 16:00 4 hours	Implemented unit testing and coded for 2 hours on the pebbleGame class & Bags	Went over testing specifications and coded for 2 hours on pebbleGame began threading.
28/10/2021	10:00 – 12:00 2 hours	Tested methods in pebbleGame related to bags and improved functionality as well as verification	Tested methods in pebbleGame and began to write player methods
28/10/2021	14:00 - 16:00 2 hours	Finished of tests for the PebbleGame methods and cleaned up repeated code	-
01/11/2021	11:00 – 1:00 2 hours	-	Created flowchart and UML to help with threading of players
04/11/2021	11:30-4:30 5 hours	Decided on the final code structure and began debugging and testing threading, had issue with stale data and out of bounds exceptions	Decided on the final code structure and began debugging and testing threading, had issue with stale data and out of bounds exceptions
05/11/2021	10:00 – 5:00 7 hours	Created test suit	Wrote report about testing

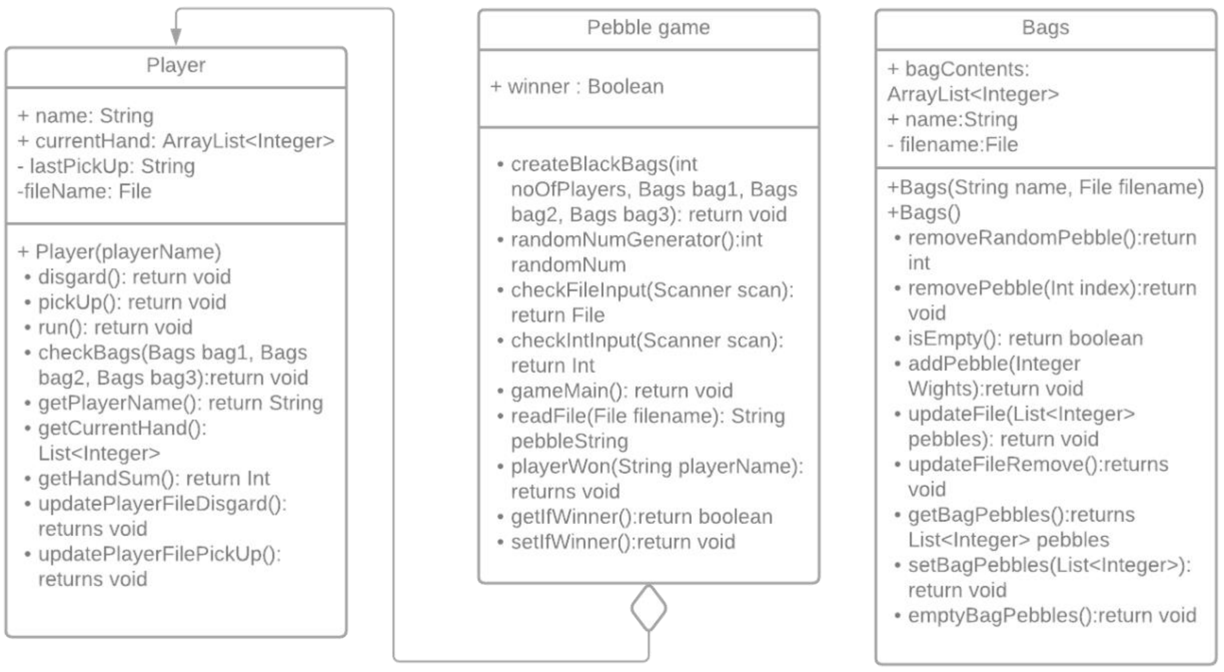
Design Decisions:

Initial thoughts-
Project would be most effectively coded through test driven development so we must have a firm grasp of requirements before proceeding. Understanding how to program in a thread safe manor and ensure all created files are working with production code will be essential to smooth running.
Project requires jar with both byte classes and source file, files include:

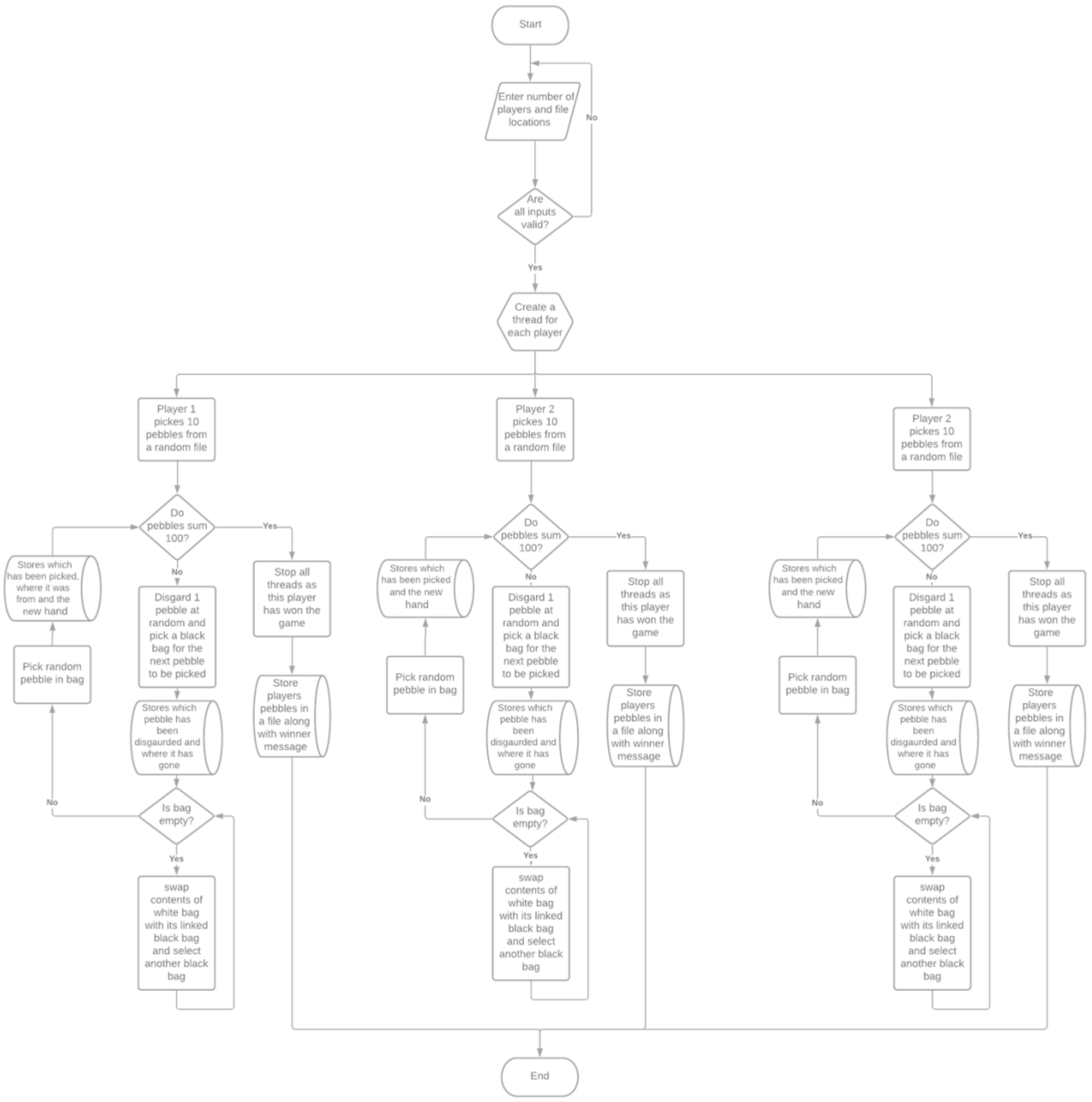
- User output files txt
- Black and white bag csv files
- Nested classes with players (threads and listeners)
- Game set up file (main file)
- Bag creation file

Decided we wanted random numbers between 0 and 25 for the game to work this is to the likelihood of getting 10 numbers that sum to 100 is higher.
We decided we wanted to include sufficient verification and used while loops to check that user inputs were of the correct form.
To ensure the game ran after a valid file was inputted we decided to populate the bags ourselves using the method createBlackBags().

PebbleGame UML


Player
+ name: String
+ currentHand: ArrayList<Integer>
- lastPickUp: String
-fileName: File
+ Player(playerName)
• disgard(): return void
• pickUp(): return void
• run(): return void
• checkBags(Bags bag1, Bags bag2, Bags bag3):return void
• getPlayerName(): return String
• getCurrentHand(): List<Integer>
• getHandSum(): return Int
• updatePlayerFileDisgard(): returns void
• updatePlayerFilePickUp(): returns void
Pebble game
+ winner : Boolean
• createBlackBags(int noOfPlayers, Bags bag1, Bags bag2, Bags bag3): return void
• randomNumGenerator():int randomNum
• checkFileInput(Scanner scan): return File
• checkIntInput(Scanner scan): return Int
• gameMain(): return void
• readFile(File filename): String pebbleString
• playerWon(String playerName): returns void
• getIfWinner():return boolean
• setIfWinner():return void
Bags
+ bagContents: ArrayList<Integer>
+ name:String
- filename:File
+Bags(String name, File filename)
+Bags()
• removeRandomPebble():return int
• removePebble(Int index):return void
• isEmpty(): return boolean
• addPebble(Integer Wights):return void
• updateFile(List<Integer> pebbles): return void
• updateFileRemove():returns void
• getBagPebbles():returns List<Integer> pebbles
• setBagPebbles(List<Integer>): return void
• emptyBagPebbles():return void

Flow Chart for Pebble Game


Start
Enter number of players and file locations
Are all inputs valid?
No
Yes
Create a thread for each player
Player 1 pickes 10 pebbles from a random file
Do pebbles sum 100?
Yes
No
Stop all threads as this player has won the game
Store players pebbles in a file along with winner message
Disgard 1 pebble at random and pick a black bag for the next pebble to be picked
Stores which pebble has been disgaurded and where it has gone
Is bag empty?
Yes
No
swap contents of white bag with its linked black bag and select another black bag
Pick random pebble in bag
Stores which has been picked, where it was from and the new hand
Player 2 pickes 10 pebbles from a random file
Do pebbles sum 100?
Yes
No
Stop all threads as this player has won the game
Store players pebbles in a file along with winner message
Disgard 1 pebble at random and pick a black bag for the next pebble to be picked
Stores which pebble has been disgaurded and where it has gone
Is bag empty?
Yes
No
swap contents of white bag with its linked black bag and select another black bag
Pick random pebble in bag
Stores which has been picked and the new hand
Player 2 pickes 10 pebbles from a random file
Do pebbles sum 100?
Yes
No
Stop all threads as this player has won the game
Store players pebbles in a file along with winner message
Disgard 1 pebble at random and pick a black bag for the next pebble to be picked
Stores which pebble has been disgaurded and where it has gone
Is bag empty?
Yes
No
swap contents of white bag with its linked black bag and select another black bag
Pick random pebble in bag
Stores which has been picked and the new hand
End

Testing Decisions:

Black bags:
- Length of list (correct number of pebbles in bag)
- All the values of the list are positive integers
- Check all numbers are within the range >25 (this is not a rigorous test as it involves random numbers)
- Check all files are of the right format (list of numbers with commas between them)

White bags:
- Check they are empty at the start of the game

Players:
- Have 10 pebbles unless they are in the process of discarding one
- Output file created when thread is started

Started by testing the bag files themselves, then our next step is to write tests for the functionality of creating the bags correctly. To correctly test our functionality, we need to produce some erroneous test files that would check our testing works, e.g. a list containing negatives or floats.
Having created private methods for bags we realised we didn't have a way to access them in the test whist keeping them separate therefore we need to code our public interface and test the public methods which will call the private ones.
We used IntelliJ to create a test skeleton for both the bags class and the pebble game class, we then began to think about the mock objects we would need to properly test our code.

Bags.class tests:

removeRandomPebbleTest: checks that the method correctly returns the element it has deleted from a arraylist

removePebbleTest: checks that the method removes the correct pebble given its index

isEmptyTest: checks that the method if an arraylist is empty

updateFileTest: checks that an arraylist has been added to the associated bag file by asserting if its empty or not

updateFileRemoveTest: checks that the file is emptied when the method is called

PebbleGame.class tests:

getIfWinnerTest: checks that the getter correctly returns false Boolean

setIfWinnerTest: checks that the setter correctly sets the previously false Boolean to true

createBlackBagTest: checks that the method writes to file without throwing an IOException and that the size of the bag is correct

randomNumGeneratorTest: checks that the method generates numbers with a lower bound of 1 and an upper bound of 25

checkFileInputTest: checks that the file is of the right form checkIntInputTest: checks that the only allowed input is a positive integer

discardTest: checks that the current hand size is one less after discarding so the pebble has been removed from the arraylist

pickUpTest: checks the current hand size is one more after picking up so the pebble has been added to the arraylist