Cover Page

The final mark for this assessment should be split 50:50

Development log

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Date | Time | Duration | Alex Role | Kerem Role | Alex signiture | Kerem signiture |
|  |  |  |  |  | 700042093 |  |

Design choices

Our PebbleGame code was structed so that you had the game class called PebbleGame and three inner classes:

“Pebble”, this would be a class to represent the pebble in the game, it doesn’t have much use outside of allowing us to have a consistent way of using arrays and arraylists together because you cannot use primitive types in arrayLists

“Player” which would extend Thread and would run on its own thread. It would have its own array of pebbles to represent its hand and would draw and discard from a random slot in its hand every turn it takes

“Bag” which would represent the bags in the game, it would hold an ArrayList of Pebbles. It would then have functionality to return and delete a pebble from its list that would then be used to give to the players who are drawing them. It would also be able to move its entire content to another bag so that we could use the same implementation for both white and black bags.

When it came to actually running the game we would start up all the player threads in a loop and then have a global winner, once a player has a hand that totals to 100 it will change the winner flag to itself from null, this is done synchronously to ensure that only the first player that has won is able to declare themselves the winner

The process of discarding and drawing a pebble is done as follows:

The player will first check if they have won, if they have not won they will then enter a synchronized block where they will discard a stone into the bag they last drew from, they will then draw a stone from a random bag and then exit the synchronised block where they will then start over again from checking if they have won.

The block in which the player discards and draws a stone is synchronized so that the player who is interacting with a bag cannot be interrupted by any other player until they are finished with the bags

During the process of discarding and drawing stones the player will output the value of the pebble that they just drew/discarded and their current hand.

the logic in the bags is as follows:

when a player draws from the bag it will call the takePebble() method which will take a pebble from the bag and return its pointer, this will then be added to the players hand and then the pointer will then be removed from the arrayList in the bag, this means that the pebble now belongs to the player, if the bag is empty when the takePebble() method is called it will get its corresponding white bag and then call fillBag(this) so that the white bag will transfer all the pebbles from the white bag into itself and then the white bag will remove all the pebbles from itself. All of this is done in a synchronized block so that only one player is able to interact with the bag which will prevent two players drawing from the bag at the same time and using any stale data

test design choices

in our testing we used Junit 5 and Visual Studio Code. We decided to use VS code because it was lightweight and we could upload one settings.json file to github and the tests would be able to be performed on either of our machines using the folder structure we had set up which would make it a lot easier to swap driver and observer roles

we chose to use Junit 5 because it’s the newest version and didn’t have a dependacy on hamcrest so it would make it easier to just use the one Junit5 jar file when testing

VS code allowed us to run the tests independently from the ide using the test runner extension

for our testing we tested the bags using their public interfaces (i.e takePebble(), transferpebbles(), etc ), to do this we used Junit assertions where we would make sure to control the state of the program so that we know exactly what the expected outcome would be when we came to asserting

for the player there was a bit more of a challenge as it had a few private methods that would change the state of the hand so we had to use java reflections to test the private methods and make sure we knew what would come out of the random nature of drawing and discarding from the bags. To do this we instantiated bags with only one possible weight and would choose the slot in the hand that would be replaced manually so that we could know where the pebbles should be coming from and where they should be placed in the hand and what their value should be

the bags were tested by making sure we knew what values should be coming out of the bag when we draw from it and making sure we knew when it should be empty, this allowed us to assert these things when it came time that they should happen

using pebbles here was very useful because it meant that we could assert the pointers were the same to ensure that the exact pebble that we want was in the right place, whereas if we had just used ints, so long as the value is the same they would have been treated as equal.