ECM2414 Software Development

Pair Programming CA Design Document

In this project we have been using Java 8 and the JUnit 4 framework.

We decided to make a class for both the black bag and the white bag. We also made a file helper class which both of these classes extended. Since the only methods that the two bag classes shared would also be used by the player class, we thought it was necessary to make the file helpers their own class.

For all the bags and the players hand, we chose to store the pebbles in their own separate file instead of a variable such as an array list. This ensured that the data would be safely stored and atomic, since two players would not be able to change the data from the file at the same time. This also was another reason for having the file helper class, since for every action, file needed to be loaded and then saved.

When taking a random pebble from the black bag, we needed a way to relay to the program that the black bag was empty and could not be replenished by the white bag. This was done by returning a negative number, and since all pebbles must be positive numbers, when a negative is returned, we know that the bags must be empty and to choose another bag.

In the main setup for the game, the black bags are stored in an array of black bags, as were the white bags. This was done since it was a simple and elegant solution instead of alternatives.

When replenishing pebbles to the black bag, the program copies all the pebbles from the white bag file to the black bag and then empties the white bag. This is done by taking the bag letter of the black bag and finding the associated white bag. This was the chosen method since it was simple and easy to implement.

The initial set up for the player involves the player drawing from the chosen black bag 10 times in order to start their hand. For this we just used the drawFromBag function inside a for loop since it was the easiest way to implement.

The doneFlag is an Atomic Boolean used to notify the other threads that one of the threads has finished. This is done by passing the doneFlag through as a parameter in the player constructor. The same doneFlag is passed into each of the players and so when one thread sets the done flag to true, all of the threads will stop running.

Buffered Reader was used instead of scanner since buffered reader is synchronised whereas scanner is not. This means scanner is not thread safe, which buffered reader is. While scanner has some useful features such as parsing the data as it is read, this does not out weight the benefits of having a thread safe way of reading files. Also, both methods are as efficient as each other.

The user inputs how many users there are and the file names for the ranges to be read from. Every time the user enters an input, the input is checked. If it is an ‘E’ then the program will automatically exit.

In order to remember what the last bag that pebbles were drawn from, we made a variable in the nested player class which holds this information. This would then be updated every time a pebble is drawn from a bag and used to ensure that the pebbles were discarded to the correct bag associated with the one drawn from last.

**Design choices and reasons for tests**

For the unit tests, we used JUnit 4 framework

We chose to write at least 3 tests for each module; a normal data test, and extreme data test and erroneous data test. If they all pass, then the module is strong and works properly.

**File Helper tests**

Before the tests, we make a new FIleHelper object which is used throughout all the tests

Normal tests

Load pebbles test:

For this method we have to provide a file name which we have previously filled my data, we then make sure that the data loaded is the same as the contents of the file.

Save pebbles tests:

To test this method, we make a new array list and add a bunch of items to it. We then use the save pebbles method providing the new array list and the file name of where we are saving to. We then use the load pebbles method and compare the output of the load pebbles method to the previous array we made to ensure that they are the same, which proves that the save pebbles method works.

Extreme tests

Load pebbles with an empty file:

This test ensures that if we attempt to load an empty file which can happen if the one of the bags are empty, then we will not encounter an error, instead it will just output an empty array

Load pebbles with multiple lines:

This test ensures that we will be able to read all of the data in the file even if it spans multiple lines. This could happen when initially loading the range of the data to be using.

Save pebbles with an empty array:

This ensures that the file will be saved as an empty file. It saves the empty array and then checks that what is loaded from that file is still an empty array. This will be used in cases such as when the black bag or white bag is empty.

**Black bag tests**

Before each test, we set the pebbles in the bag to be the pebbles in the test\_range file. This is so that we know exactly what pebbles are in the bag before each test

We also use a mock object for the random object so we can predict what random numbers will be produced and work out what outputs from certain functions will be produced with those predictable random numbers.

Normal tests

Take Random Pebble test:

For this method, we just ran the module multiple times with an assertEquals against what the expected result would be

Replenish pebbles test:

For this method, we had to fill a white bag, record the state of both the white bag and black bag before and after calling the replenish pebbles method. If the initial white bag matched final black bag. Then the black bag has been replenished correctly, but we need to check also white bag has also been emptied

Get total number of pebbles test:

For this method, we provide a black bag with a known number of pebbles, call the getTotalNumPebbles() method and check that it is the expected number.

Extreme tests

Take a random pebble from a black bag with 1 pebble:

This is used to make sure that there will be no problems when the black bag is almost empty and that the take random pebble method still works as it is supposed to.

Take a random pebble from an empty black bag, but it will be replenished by a white bag:

This may happen in the running of the program so it is important that the process works.

Replenish pebbles with 1 item in the white bag:

With this test we are ensuring that the method of replenishing pebbles is robust and there are no off-by-one errors.

Erroneous tests

Take a random pebble from an empty black bag, but the associated white bag is also empty:

This should return a negative value which will then be handled by the program it is returned to by making the program choose another black bag to take from.

Replenish pebbles when the white bag is empty;

This should make the black bag file empty, which is already is, so realistically will have no effect, but also will throw no errors, so the program must handle this earlier in the call stack. If the replenish pebbles method has been called and the black bag is still empty, the program should ask for another black bag.

**White bag tests**

Before the tests, we make a new white bag object which will be used for each of the tests

Normal tests

Add a pebble to an empty white bag:

This test

Add a pebble to a white bag which already has pebbles

**Player tests**

Normal tests

Draw a pebble from a black bag

Discard a pebble from the players hand to a white bag

Check that the total weight of pebbles matches 100 with pebbles that sum to 100

Check that the total weight of pebbles does not match 100 with pebbles that do not sum to 100

Extreme tests

Draw a pebble from a black bag which has only 1 pebble left

Draw a pebble from an empty black bag but a filled white bag

Discard a pebble from the players hand to an empty white bag

Check that the total weight of pebbles does not match 100 with pebbles that do not sum to 101

Check that the total weight of pebbles does not match 100 with pebbles that do not sum to 99

Erroneous tests

Draw a pebble from an empty black bag and an empty white bag