ECM2414 Pair Programming CA

Tomas Premoli, Lucas Proudhon-Smith, mark split 50:50

# Development Log

* A development log, which includes date, time and duration of pair programming sessions, and which role(s) developers took in these sessions, with each log entry signed by both members (using your candidate number as your signature). This part of the document should be no more than one side of A4.

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Lucas contribution** | **Tomas contribution** | **Signed** |
| Oct 15, 2021  (1 Hour) | Helped setup GitHub repo, discussed how to go about doing the coursework. | Setup GitHub repo, discussed how to go about doing the coursework. | Lucas: 700074221  Tomas: |
| Oct 17, 2021 | Created and worked on UML to plan methods and classes. | Worked on UML to plan methods and classes, started work on basic outline of the program. | Lucas: 700074221  Tomas: |
| Oct 18, 2021 | Added documentation, worked on generation of bags, general bag methods, general code fixes and optimisation. | Added documentation, worked on generation of bags, general bag methods, worked on main game method. | Lucas: 700074221  Tomas: |
| Oct 20, 2021 | Worked on takeRandomPebble(), getTotalPebbleWeight() and bag creation, general code fixes and optimisation. | Worked on takeRandomPebble(), getTotalPebbleWeight(), improved encapsulation for PebbleGame, general code fixes/optimisation. | Lucas: 700074221  Tomas: |
| Oct 25, 2021 | Worked on player constructor, getRandomBlackBag(), swapPebble(), getCounterPart(). General code fixes/optimisation, documentation | Worked on error handling, logging player moves to a file and swapPebble(). General code fixes/optimisation, documentation. | Lucas: 700074221  Tomas: |
| Oct 26, 2021 | Worked on swapContents(), threading aspects of code, general code fixes/optimisation | Fixed file writing, worked on swapContents(), threading, general code fixes/optimisation. | Lucas: 700074221  Tomas: |
| Oct 29, 2021 | Worked on write up | Implemented run(), implemented exit on ‘e’ input, changed swapPebble(int pebbleWeight) for swapRandomPebble() for better encapsulation between objects. | Lucas: 700074221  Tomas: |
| Nov 07, 2021 | Worked on testing/write up | Finalizing and cleaning up code | Lucas: 700074221  Tomas: |
| Nov 08, 2021 | Final code testing/optimisation, submitting project | Final code testing/optimisation, submitting project | Lucas: 700074221  Tomas: |

# Design choices

## Production code

Timeline

Description automatically generated with low confidenceTo start off with, after an initial discussion and quick read through of the project spec, we decided to make a rough UML so we could outline the basic structure of the code, figure out which methods we’d need and further understand the specification provided.

As we continued working on the project, we went slightly off the original UML’s plan as we figured out more efficient/better ways to create the program. We also found several issues with the UML due to parts of the spec that we had misunderstood, such as changing the swapPebble(int pebbleWeight) to swapRandomPebble() as originally we thought that a player would need to input the pebble they wanted to swap.

In the final program, there were 4 main classes:

* Main
  + A class with methods to:
    - Run the actual game, get user inputs for pebble file locations and player count
    - Create player threads
    - Generate bag objects
* PebbleGame
  + A class which holds the attributes for:
    - An atomic Boolean value used to check if the game is finished or not
    - A lastBag attribute so the program knows what the last bag drawn from was
    - A hashmap of bags and an arraylist of player objects.
  + Contains methods to get/set attributes, create black bags and end the game
  + Also contains a nested ‘Player’ class which holds attributes for
    - An int playerID
    - An array of pebbles that the player currently has
    - A string which stores the location of the player’s output file
  + The Player class contains all the necessary methods needed for a player thread.
* Bag
  + A class which holds attributes for:
    - A hashmap of Bags
    - A bagType value
    - The string of the fille location where a bags content is stored
    - An arraylist of a bag’s pebbles
    - The bag’s corresponding character (bagName)
  + The class also holds the bagType enum used to distinguish between black and white bags
  + The class holds all the methods needed to work with a bag and it’s contents
* PebbleErrors
  + A class used for processing errors including:
    - IllegalPlayerNumberException - thrown when the provided player number is not positive.
    - IllegalBagTypeException - thrown when a method is used on a bagType that the method cannot be used on.
    - NotEnoughPebblesInFileException - thrown when there aren't at least 11x the number of valid pebbles as number of players.
    - NegativePebbleWeightException - thrown when a pebble weight in a file is negative.
    - Empty Bag Exception - thrown when a player attempts to draw from an empty bag.
* PebbleGameTest
  + Methods used for testing, covered in next section

## Testing

For testing, we used Junit 4 and followed the testing criteria seen below

### main methods

|  |  |  |
| --- | --- | --- |
| Method being tested | Test description | Result expected |
| generateBags() | Covered in main |  |
| main​(String[] args) | Test to see that the game starts and runs with console inputs | black bags and white bags are created, players are created and their threads should be running. |

### pebbleGame methods

|  |  |  |
| --- | --- | --- |
| Method being tested | Test description | Result expected |
| createBlackBag​(char name, String fileLocation) | Use parameters (‘t’, “example-example-input-files/example\_file\_1.csv”) to create a mock black bag | A bag called i is created with it’s contents being the pebbles contained in example\_file\_1.csv |
| finish​(boolean isExit) | Enter ‘e’ or ‘E’ to console | Program immediately exits |
| winCheck​(Player player) | Set a player’s pebble total to 100 but giving an array of pebbles all with weight 10 | Program announces that the player has won, exits |
| getBags(), getPlayerCount(), getPlayers(),  setPlayerCount​(int playerCount) | Ensure get/set methods work as expected using valid inputs | No errors should be thrown, expected returns seen |

### pebbleGame.player methods

|  |  |  |
| --- | --- | --- |
| Method being tested | Test description | Result expected |
| Player constructor | Check that a player object can be created as expected using parameter 1 for the playerID | New player object created with ID 1, an empty pebble array and a output file called “Player 1.txt” |
| getRandomBlackBag() | Try to create a black bag called t using an example pebble file | The test bag should be returned |
| getTotalPebbleWeight() | Check that the correct pebble weight is returned | Should return 0 on a test Player object as no pebbles have been added |
| swapRandomPebble() | Run swapRandomPebble and see if the pebbles array has changed (original array is made of 0s, no 0s are present in bags) | totalPebbleWeight should return > 0 |
| writePebblesToFile(FileWriter writer) | Run the method then check the new file is the same as test\_ writePebblesToFile.txt (expected values) | The files should match |
| writeDiscardToFile(int discardedPebble, char bag) | Run the method and discard a pebble with weight 0 to bag A. Check the new file is the same as test\_ writeDiscardToFile.txt (expected values) | The files should match |
| initialWrite(char initialBag) | Run the method with the initialBag parameter as ‘A’. Check the new file is the same as test\_initialWrite.txt (expected values) | The files should match |
| run() | Check that the initial draw goes as expected, and the game runs and finishes as expected. | No errors should be returned, ttotal pebble weight should be > 0, and the game runs and finishes as expected. |

### Bag methods

|  |  |  |
| --- | --- | --- |
| Method being tested | Test description | Result expected |
| Bag constructor | 1: Testing white bag construction  2: Testing black bag construction | 1, 2: Successful bag construction |
| addPebble​(int weight) | Using pre created bag, random pebble with weight <100 | Number of pebbles in bag increased by 1 |
| takeRandomPebble() | Try to get random pebble from pre created black bag, white bag | Int pebble weight and number of pebbles in bag decreased by 1, IllegalBagTypeException |
| getCounterpart() | Test to see the right counterpart is returned, input bag A | Char value ‘X 00208’ |
| swapContents​(char b) | See if the input bag’s contents have been swapped with the contents of its counterpart bag | Original contents of bag = new contents of counterpart bag |
| setPebbles(), setPebbles(ArrayList<Integer>), getBagName | Ensure get/set methods work as expected using valid inputs  Covered in previous tests |  |

### Exceptions testing

|  |  |  |
| --- | --- | --- |
| Exception being tested | Test description | Result expected |
| IllegalPlayerNumberException | Run the main method’s generatePlayers method with -1 as a parameter (negative player counts aren’t allowed) | IllegalPlayerNumberException thrown |
| IllegalBagTypeException | Run addPebble on a black bag (addPebble can only be run on white bags) | IllegalBagTypeException thrown |
| NotEnoughPebblesInFileException | Can only be tested through console input due to how the exception is handled. |  |
| NegativePebbleWeightException | Can only be tested through console input due to how the exception is handled. |  |

Using the testing criteria above, according to IntelliJ, we covered 83% of the code. Most of the rest off the code that wasn’t covered was stuff that could only be covered by unit tests, but only by console input, which we also tested thoroughly.

Table

Description automatically generated

Graphical user interface, text

Description automatically generated