University of Exeter

Software Development Pair Programming Coursework – Pebble Game

Mark Split: 50:50

Tomas Premoli 700047428

Lucas Proudhon-Smith 700074221

ECM2414 Software Development

Dr. Yulei Wu

11th November, 2021

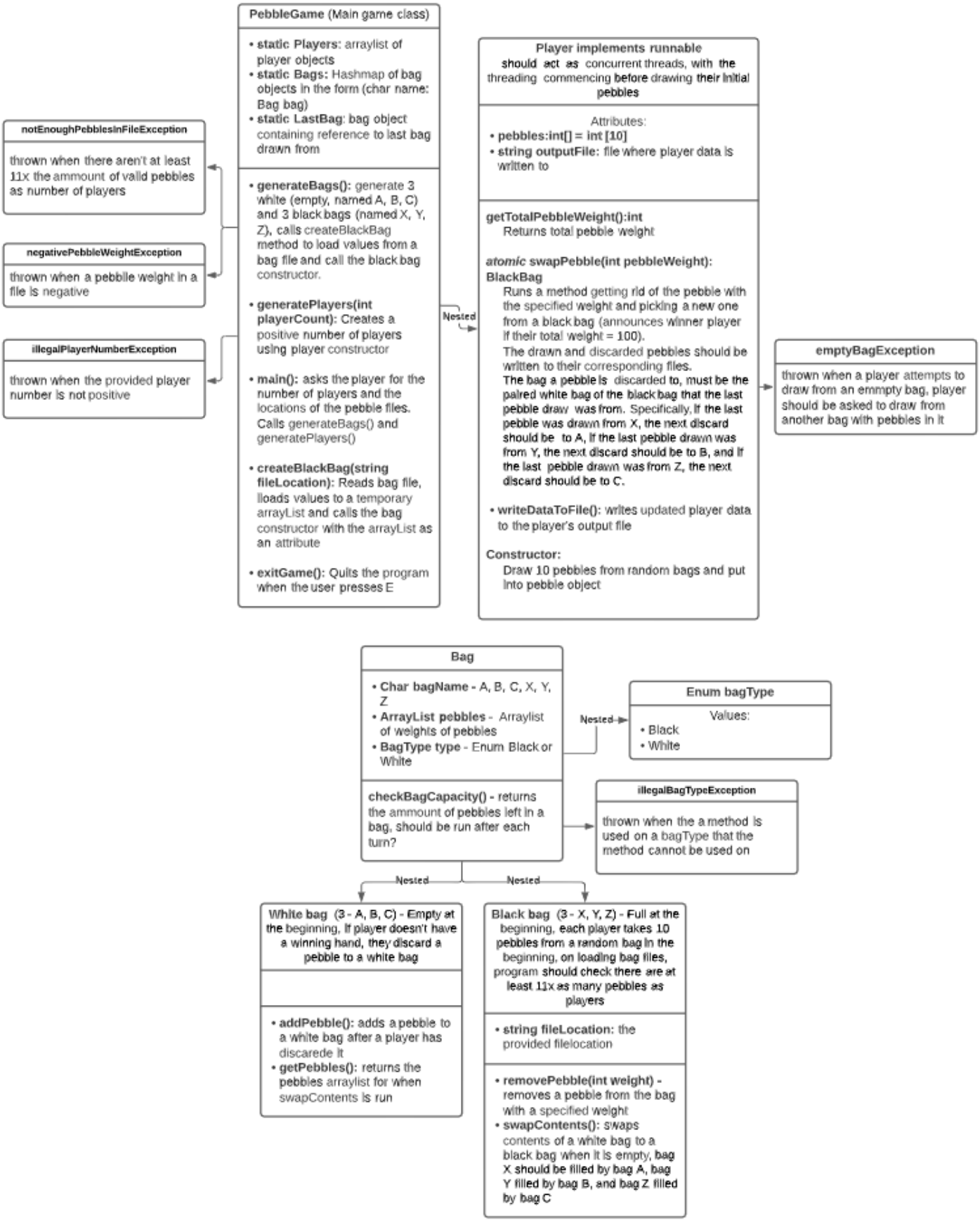
1 DEVELOPMENT LOG

Date	Lucas contribution	Tomas contribution	Signed
Oct 15, 2021 (1 Hour)	Helped set up GitHub repo, discussed how to go about doing the coursework.	Setup GitHub repo, discussed how to go about doing the coursework.	Lucas: 700074221 Tomas: 700047428
Oct 17, 2021 (1.5 hours)	Created and worked on UML to plan methods and classes.	Worked on UML to plan methods and classes, started work on the basic outline of the program.	Lucas: 700074221 Tomas: 700047428
Oct 18, 2021 (1.5 hours)	Added documentation, worked on generating 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: 700047428
Oct 20, 2021 (1.5 hours)	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: 700047428
Oct 25, 2021 (1.5 hours)	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: 700047428
Oct 26, 2021 (1.5 hours)	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: 700047428
Oct 29, 2021 (3 hours)	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: 700047428
Nov 07, 2021 (2 hours)	Worked on testing/write up	Finalising and cleaning up code	Lucas: 700074221 Tomas: 700047428
Nov 08, 2021 (1 hour)	Final code testing/optimisation, submitting project	Final code testing/optimisation, submitting project	Lucas: 700074221 Tomas: 700047428

2 DESIGN CHOICES

2.1 PRODUCTION CODE

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 initially, we thought that a player would need to input the pebble they wanted to swap.

In the final program, there were four 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 that 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 drew 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
    - It also includes a nested 'Player' class which holds attributes for
        - An int playerID
        - An array of pebbles that the player currently has
        - A string that 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 that holds attributes for:
        - A hashmap of Bags
        - A bagType value
        - The string of the file location where the content of a bag 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 its 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

2.2 TESTING

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

2.2.1 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.

2.2.2 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 its 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 by giving the player an array of pebbles, all with weight 10	The 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

2.2.3 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	A new player object created with ID 1, an empty pebble array and an 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, the total pebble weight should be > 0, and the game runs and finishes as expected.

2.2.4 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 a pre-created bag, add a random pebble with weight <100	The number of pebbles in the bag increased by 1
takeRandomPebble()	Try to get random pebble from the pre-created black bag, white bag	Int pebble weight and the number of pebbles in the bag decreased by 1, IllegalBagTypeException
getCounterpart()	Test to see the right counterpart is returned, input bag A	Char value 'X'
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	

2.2.5 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 of 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.