ICT 4315 Unit 2

Introduction

Raj had been having a difficult time returning to work from his vacation in New Delhi. He had been visiting family for over a month and a half and had not been developing any software during his time away from Levtk, his place of employment. It seems like all it takes to forget basic java principles is a month of vacation. Realizing he had lost some of his sharpness, he broke open an old assignment he had done back when he was in the University. It was a pretty famous problem called The Knights Tour.

Knights Tour Problem

The specific problem he was working on was one where the professor had provided the solution in Java , but it was in a procedural form and he was tasked with creating classes that used the logic provided by the professor with the desired outcome of a truly object oriented program.

The logic itself was trustworthy to Raj as he knew his professor knew what he was doing, but he wasn't a big chess player. Actually, he had never played chess before and this assignment had thrown him for a few loops. He recalled taking that logic the professor had provided and writing out how it would make the knight move about the chessboard.

The classes

Raj started looking at the code and trying to determine what would be good classes. Good classes in this case were classes that were related to the problem domain and provided methods that were applicable to themselves. He had determined that the problem could be developed into OO code with a Knight, Board, Game, and MovementLogic class.

Knight

Raj's Knight class was simple enough. It had three private fields and three public methods. He had followed the guidelines set forth by his professor – all fields should be private and all methods should be public. This guideline ensures that the fields within the classes can't be altered by the consumers of the class.

The knight held an integer that was set at eight which was the total number of different moves a knight could make. It also held two arrays of integers which were a length of 8 each. These arrays each corresponded to the movement of the knight; knights move one left and two up, two left and one up, one left and two down, two left and one down and then four other moves exactly the same, but to the right. This equated to the total number of moves a knight can make. By looking at each index of each array the knight could tell any calling class where it was located on the provided the index as a parameter of each of get methods.

Board
The board class was composed of three private fields and three public methods. The fields were a two dimensional array of integers each of size eight which represented the all spaces on the chessboard. There were two other integer variables which were named rows and columns and these were set to eight as well and were used in the method called legalMove which determined if a move would be possible i.e. if the knight took the move would it still be on the board.

There setBoard method set up the chessboard with every space equal to zero after each place of a game. So it was a zeroing out of the board function to wipe it clean.

Finally, the board was able to print itself out at the end of the games if the knight's tour was successful. This method was called in the Game class and only if the number of moves the knight had taken were equal to sixty-four which would mean that the knight had been in every space on the board.

MovementLogic
Most of the magic of the application was in its logic class. There were three objects instantiated in the MovementLogic class, a Board, a Knight, and a Random number. There were also two private integer variables named currentRow an currentCol that were used in the four methods provided by the class.

The play method was responsible for running a single play of the game and called the nextMove method which iterated over the possible moves and made a call to the instance of board legal move to check if the move was possible. If the move was possible or legal, the nextMove method stores the possible moves in an array which is then checked in the methods additional logic to see if there are more than one possible move. If there is more than one, the random.nextInt method is called to randomly choose one of the legal moves. If there are no legal moves then that game ends and the play method is called again from the Game class until the game either results in a Full Tour or the total number of attempts to solve the knights tour problem are exhausted without success.

Game
The Game class is where the applications main statement lives and where the number of attempts are configured in a for loop. The Game class mostly calls the play method from the MovementLogic class repeatedly until the tour completes or fails in which case a Closed Tour message is printed to the screen. The class also prints out the progress by using the modulus operator to determine when each 100,000 attempts have been

completed so that the user of the program knows something is progressing and the program is still functioning.

Conclusion

Raj used the Knights Tour assignment he had to remind him of proper object oriented design and use of looping in Java. He also had created a few JUnit test for that assignment , but they were poor and he didn't get full credit for them. He did make decent tests for his Board methods , but fell short on the MovementLogic methods as they required the methods to be broken down into simpler ones which could be mocked up and tested individually.

Note to Professor Judd
I am writing this as if I were Raj as I am sure you understand. I know my unit tests are poor. I feel that you've helped to guide me in the right direction this week and appreciate your timely feedback when I requested it. I do have a sense of accomplishment from getting this assignment working, although it was late. I am hopeful I will be able to make better progress on this weeks assignment and produce better tests.