Open Knight's Tour Report

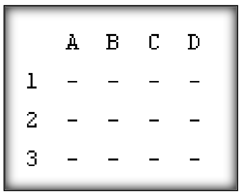
The first thing my teammate and I did was write down pseudo code of how the program should run.
Our pseudo code was this:

- See the map visually, with dashes to fill the spaces
- Instructions for the game stated
- Code to loop the game until the user has filled the dashes or has quit
    - Turn the dashes into user count
- After each move, the map will be redrawn with the new move
- If user completes the game, give congratulatory message or if press quit, ends the game

The project was tested by listing what could be typed in and using a process of elimination to see if the program would break.
We concluded what could be typed in this format (column) (row) was:

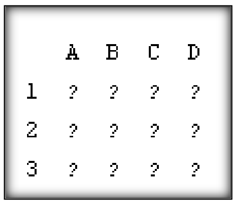
- Not inputting anything and pressing enter
- Entering a letter other than a, b, c, or d, and a number other than 1, 2 or 3.
- Vice versa above
- Inputting a, b, c or d, and a number other than 1, 2 or 3.
- Inputting a letter other than a, b, c, or d, and 1, 2 or 3.
- Inputting double numbers or letters

We first started off by making the 2D array map static so then it becomes a global variable and then all the different methods later used can use it. The code for drawMap was made private so it won't interfere with the rest of the code until it is called.
drawMap was made by using a for loop to loop through the columns and rows until it reaches the end of the array, and since "-" is read in mapBoard, " -" is put there to represent dashes in the table and to finish it off, spaces and "\t" were used to space out the board evenly thus the table would look like this.



```
   A  B  C  D
1  -  -  -  -
2  -  -  -  -
3  -  -  -  -
```

However if " -" was and something else like " ?" was used instead, the map would look like this instead.



```
   A  B  C  D
1  ?  ?  ?  ?
2  ?  ?  ?  ?
3  ?  ?  ?  ?
```

Then the variables for the program were stated and we made a while loop so the code would loop until the user has pressed quit and the board was filled. The instructions were typed up and if statements were used for the rest of the code. If "Q" was pressed the program terminated, if not, switch statements and limitations were used so if the user input was wrong, the program would re-loop.
Afterwards when the user had filled the board, a congratulatory message would show up and since we added a saveFile function, the person could search for the .txt to see the moves made.

We had several issues. We originally used hash-map to write the code for the table because I had looked online for tips and saw that was also an option to use, so I learnt how to use it and started writing the code, but after discussing with the teacher, using 2D arrays was the best so the option of enlarging the map and using the L-shape rule was there.

The other issues were to do with user input and the program broke whenever there was:

1. A random input of letters / numbers above or below 2 characters
2. Only 1 single character entered
3. Switching the inputs around so instead of “A1”, “1A” was put instead
4. An input of a, b, c or d, and a number which was 4 or above
5. An input of a, b, c or d, and the number 0

After each of those inputs, the program read “invalid move” and then the program wouldn’t allow us to enter anything else, or the program broke and gave us an array out of bounds exception.

We looked at our code carefully and concluded that we needed to put limitations on the user input, so we started off with our switch statement. Our switch statement was so if the user had pressed a, b, c or d, as their first character, that would change to corresponding numbers and then the code would read the second character and if the user input was valid, this code would run:

```
if (mapBoard[secondNumber][toNumber] == "-") {
      mapBoard[secondNumber][toNumber] = Integer.toString(userMoves);
      userMoves ++;
}
```

Which would check to see if the location to see if it was still empty, and if it was, it would store the move in the 2D array and add 1 to the userMoves, but if it wasn’t the program would re-loop.

The first and second issue were really similar and so we put this code:

```
      if (Knight.length() > 2 || Knight.length() <= 1)
```

So if anything wasn’t 2 characters, it would be registered as invalid and the program would re-loop.

The next issue was fixed by simply adding:

```
      default:     // If Knight's first character is none of above, toNumber = 0.
            toNumber = 0;
            break;
```

To the end of the switch statement which made it if anything other than the 4 cases was inputted, the program would read it as invalid and re-loop.

The last 2 issues were similar to the first 2 - adding limitations would make it so entering a number 4 or above wouldn’t break the program.

```
      if (secondNumber < 1 || secondNumber > 3)
```

That made it so if any number which wasn’t 1, 2 or 3, would be registered as invalid and the program would re-loop.

Fixing the code was hard because even if there are no errors, it has to be scanned carefully and broken down methodically to find and fix the problem and even after knowing what code to put down, where to put it was another problem in itself to make sure that the program does get fixed and doesn’t read it where you would have liked because the code was put outside a bracket. We listed a multiple of solutions to each problem and from good communication and collaboration, the code finally got fixed. However we were not able to code in the “L-shape” rule because we were not sure how to modify the code given to us to make it work.