Sajid Amin

CUS 1156 Project 2

I tested some methods from PedCountRecords and PedCountFileReader class and here are the ones I thought were important for testing to make sure this wasn't a broken program.

- I tested for testFileLoads for checking to see if the program is actually loading the file and returning the data. If it fails, it means none of the other features would even work. I used assertTrue(!pedCounts.isEmpty()) to make sure the data is being read.
- I tested for testRecordCountMatches because after loading all the PedCount objects into the PedCountRecords class, I wanted to confirm that the total number of records matched what was read from the file. It's a good check to make sure the data is being stored correctly.
- I tested for testGetTotalCountByDay because this test checks to see that the method counting pedestrians per day which I used Monday as an example to see it works and doesn't return any weird or negative values. It's important because I use this logic later in the program for stats.
- I tested for testGetAverageByDay because this test checks that averages are calculated correctly for each days like if the method returns a negative number, something's wrong. So assertTrue(avg>=0) helps to catch that
- I tested for testGetWeatherTotalsToManhattan which tests for when the user chooses Manhattan, the program gives a valid map of total counts for each weather type. This map is what is shown in the pie chart
- I tested for testGetWeatherTotalsToBrooklyn for the same reason but to check for Brooklyn as well and not just only Manhattan
- I tested for testInvalidFileNameReturnsEmptyList, this test is something I thought of towards the end of my project because I initally did not even have this issue because I was using a csv that was provided to us, but then I thought what if the program handled a wrong file name? So it should not crash or throw exception.

These tests don't test for every edge case, but I believe I did touch upon the most important features that the menu relies on and what users actually interact with. I also believe that I followed good design practice and made my code readable.

- I split the program into individual classes where PedCountFileReader only reads the file, PedCountRecords hold and processes the data, PedCount which was provide to us, and Main which is for the user to interact with and access the program.
- For directions like manhattan and brooklyn I used constants like TO_MANHATTAN and TO_BROOKLYN and reused the same method getWeatherTotals(direction) for both directions. This was an example of me avoiding writing repeated code and code duplication
- In Main class, I check the user input for valid direction and give a message if its wrong through. That improves the programs user experience and prevents users from being confused on why the program isn't working as intended
- I also used methods like .toUpperCase for user input so when user types in manhattan or brooklyn, it's case insensitive. Same applies for the choices for the cases. For selection, the letter could be upper case or lowercase and it won't interfere with the program.
- My variable names are also straightforward and have meaningful names like "weatherMap", "direction", and "average". My switch statement is also easy to follow and clean and my loop lets the user try again without restarting the program.
- I also did error handling like try and catch around file reading and skipping bad lines. The program doesn't crash if the file is missing or the data isn't clean, it just simply notifies the user through the use of system.err.print and they can read the message and know why the program is not running as intended.