Lab 1: schoolsearch

Daniel Deegan & Lauren Hibbs

For our lab, we decided to use Java, as it was a language that we are both familiar with.

An ArrayList was selected as the primary data structure to account for students.txt files of indeterminate length. Both arrays and ArrayLists allow for iterating through the data to find the required values, but an array would either take up extra memory for slots that would never be filled or not have enough space for all the students. The ArrayList consists of a custom Student object, with fields for each value in students.txt.

An alternative we considered was a HashMap so that we could index by the different values that can be searched for in the program. However, this seemed unreasonable because the user can search by almost every attribute on student, so it would require making many hash maps.

For searching through the array, we tried to implement something that would use a single 'query' function to look through the table, so as not to duplicate code unnecessarily. We decided to implement this in a functional way, using Java 8 streams to look through the array of Students and perform operations on each one.

Task Log:

Name: StudentParser
Programmer(s): Daniel Deegan
Time: 9/20 11:30-12:00, 9/23 11:00-11:20
Total person-hours: 0.833

Name: SchoolSearchCmds
Programmer: Lauren Hibbs
Time: (Research on Streams/Strategy) 9/20 11:00-12:00,
(Implementation) 9/23 11:00-12:00 9/26 8:00-9:00
Person hours: 3

Name: SchoolSearch
Programmer: Lauren Hibbs
Time: 9/25 9:00 - 11:00 9/26 11:00-12:00
Person hours: 3

Name: Testing & Bug Fixing
Programmers: Lauren Hibbs & Daniel Deegan
Time: 9/27 12:00 - 2:00
Person Hours: 4

Testing:

Date: 9/23
Tester: Daniel Deegan
Subject: Ensuring the StudentParser read students.txt properly into Student objects
Bugs Found: The teacher's last name would appear at the beginning of the line, overwriting any text that was already there.
Process: By printing each component individually, I ensured that all the data was being stored properly, and the issue was caused by printing it. Additionally, the problem only occured when the last name was at the end of the line, printing the last name before the first name worked just fine. By replacing the last name with a random String, I was able to determine that the problem was not actually caused by the last name, but by the first name preceding it. I discovered that because the teacher's first name is at the end of the line, it was carrying a hidden carriage return character along with it. The Scanner's delimiter was set to only commas or newlines, so the return was being read in as part of the name. The problem was resolved by adding /r/n to the possible delimiters.
Time taken: ~40 minutes.

Date: 9/27
Tester: Lauren Hibbs & Daniel Deegan
Subject: Fixing the Grade High/Low search
Bugs Found : Nothing was being printed for grade high or low
Process: The stream .filter() function returns an empty stream when the comparator is set up incorrectly. There are many different ways to write comparators for streams, but the documentation is complex. I changed from Student implementing Comparable to using something called Comparator.comparing(). This fixed the problem.
Another struggle with Java streams that had to be researched was that there is not an easy way to remove the first element from the stream. .findFirst() returns an optional and I could not figure out how to call functions on an optional stream. To get around this I collected the stream into a list and removed the first element that way.
Time Taken: 45 minutes

Final Notes:

Using streams was a time consuming way to complete the project because we were not very familiar with them. We chose to do it because we were interested in learning how to functionally program in java.
We didn't find many bugs when testing our program against the expected outputs. However, there was a lot of bug fixing to do with streams and it added a lot of time to the process, as mentioned above.