

Name: William Crutchfield

Date: 02/06/19

Course: CMSC 335 – Object-Oriented and Concurrent Programming

Project: 2

1 Source code, data files, and configuration files (if any)

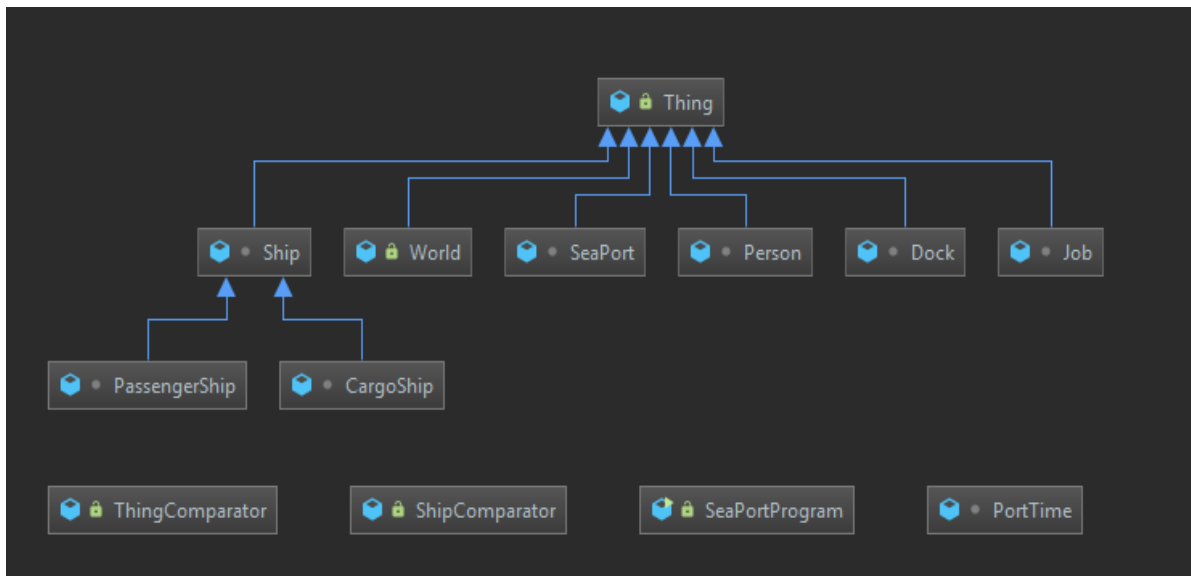
Insert a zipped file of NetBeans ALL project files (so that it could be unzipped and loaded into NetBeans IDE again), zipped file of all data files, and zipped file of configuration files (if any). :



src.zip

2 Design

Insert here UML Class diagram, explain classes, variables, methods, explain how classes tie to the requirements of the project:



The main difference in this UML Diagram compared to the previous project, is that we have two new classes, ThingComparator and ShipComparator. ThingComparator is the default comparator that is used to sort objects by name and index. The ShipComparator is a specialized comparator that is used for Ship objects only. ShipComparator technically extends ThingComparator by calling ThingComparator if the specialized compare methods in ShipComparator are not needed. However, ShipComparator does not actually “extend” ThingComparator due to the need to implement a comparator for Ship objects only.

3

User Guide

Explain how a user starts & runs your project, and any specific features with screenshots:

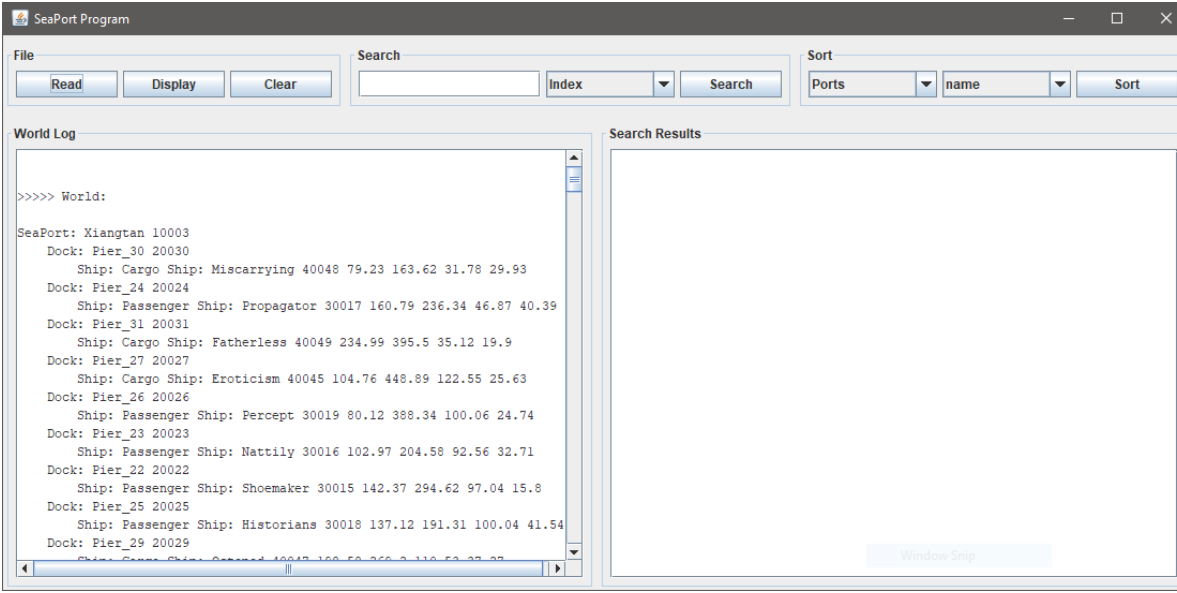
In order to run this project, download the src.zip file and extract. Once extracted, open the src files up in any IDE. Lastly, run the SeaPortProgram.java file.

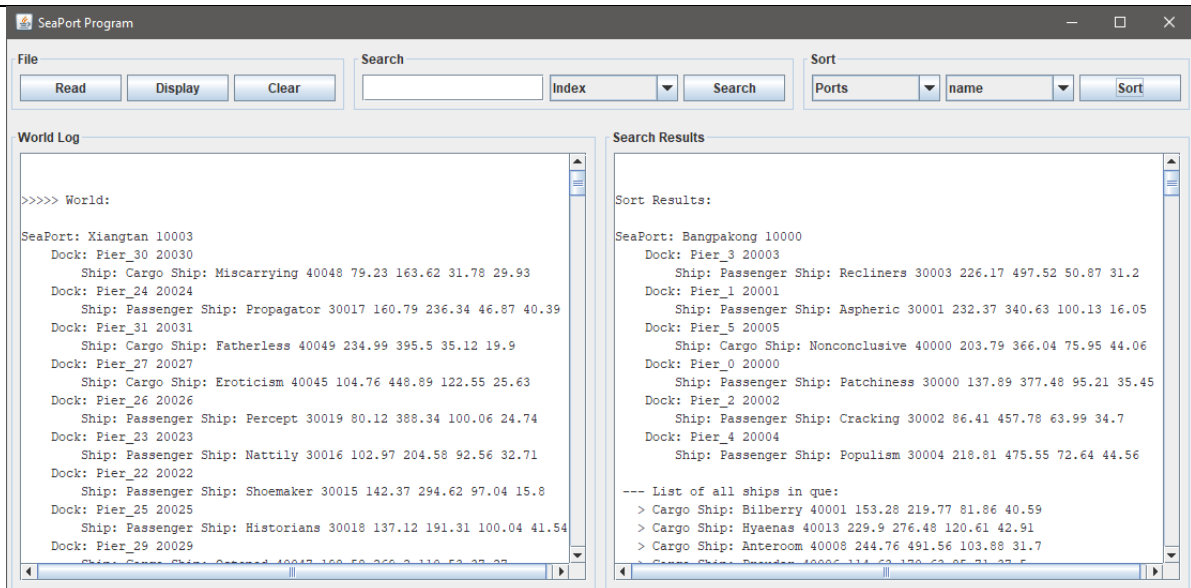
4

Test Plan

Complete this table and extend it with your test cases:

Test Case 1:	
Test Case:	Sort aSPab.txt by Port name
Selected Input:	aSPab.txt
Expected Output:	Bangpakong Majunga Port_Des_Galets Wuchun Xiangtan
Actual Output:	





Pass/Fail: Pass, looking at the screenshots, we can see that the ports are now order alphabetically. I called toString on the sort method so that the user may see the sorted data structure and compare it to the original data structure.

Test Case 2:

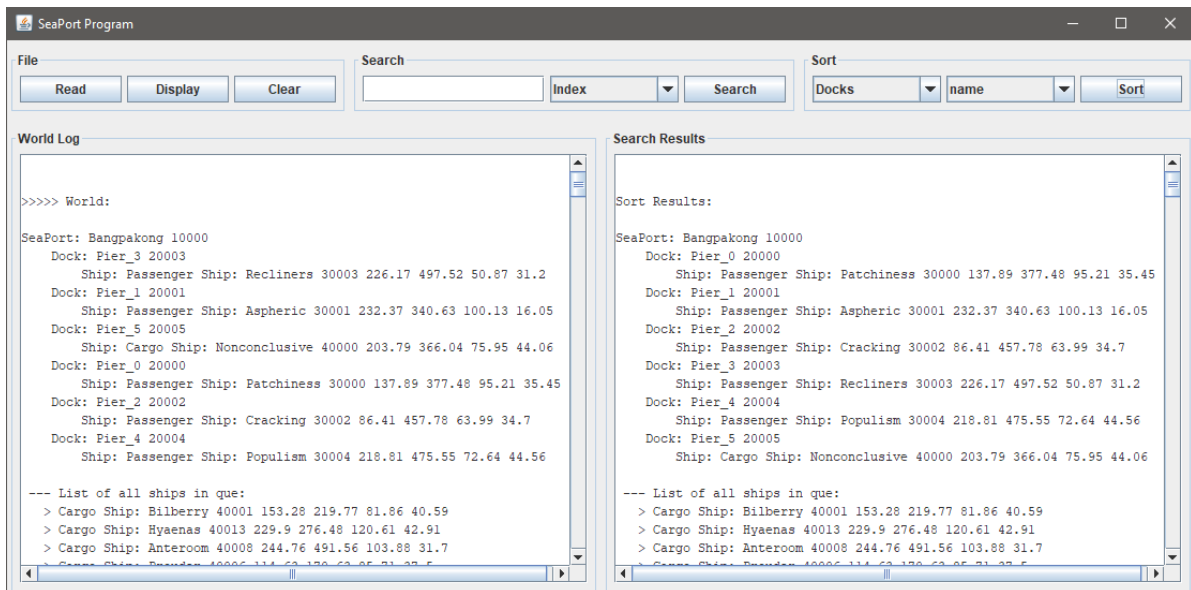
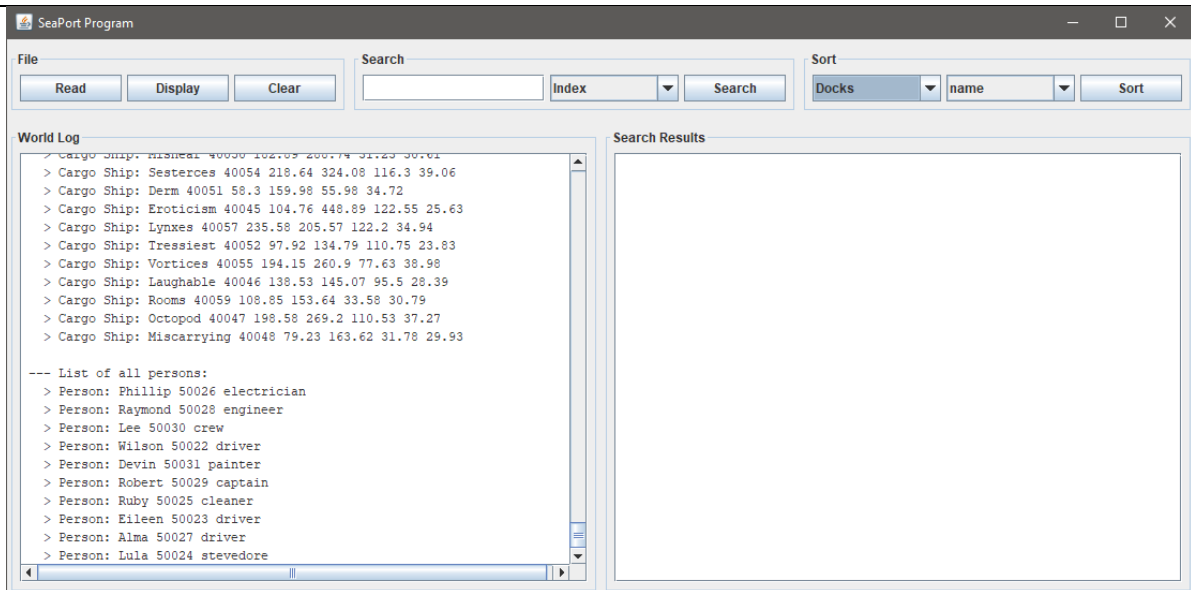
Test Case: Sort aSPab.txt by Dock name (following results of previous test case)

Selected Input: aSPab.txt

Expected Output: Pier_0
Pier_1
Pier_2
Pier_3
Pier_4
Pier_5

Etc.

Actual Output:



Pass/Fail: Pass, looking at the screenshots, we can see that each port now organizes it's docks by name. It is important to note that it is organized by port, so each independent port is sorted by dock name.

Test Case 3:

Test Case: Sort aSPab.txt by Dock name (following results of previous test case)

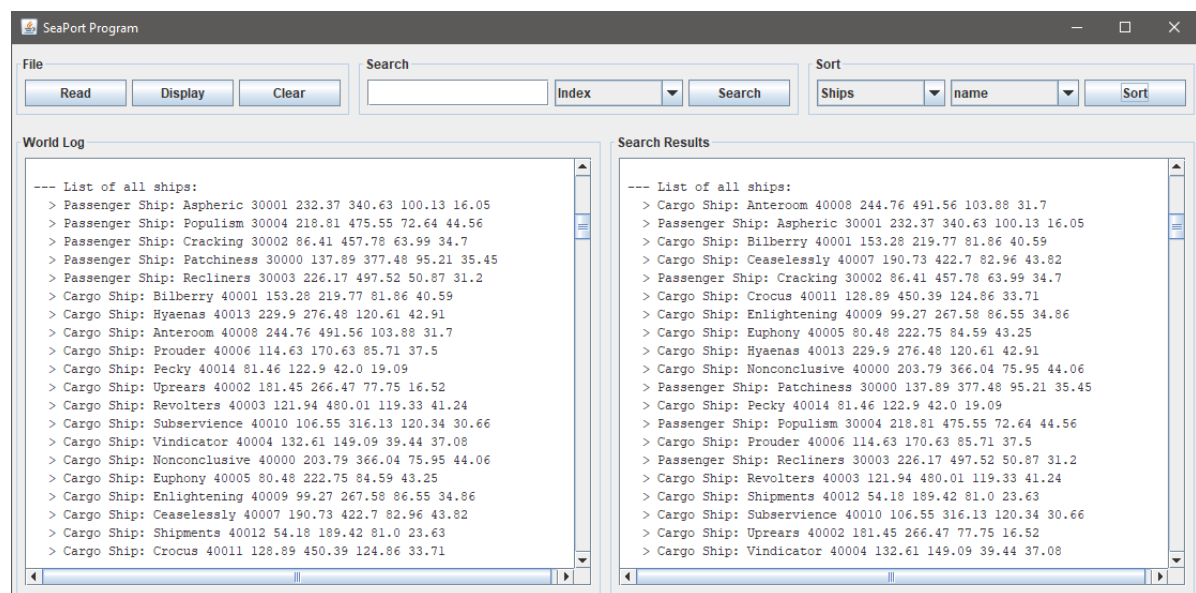
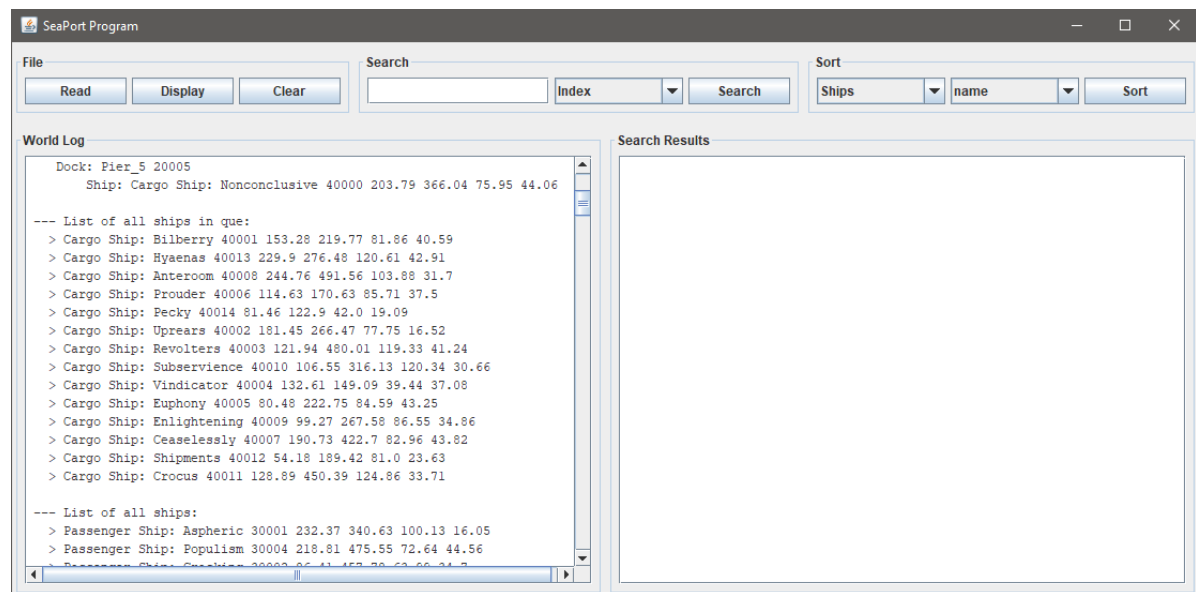
Selected Input: aSPab.txt

Expected Output: (Bangpakong Port)

Anteroom
Aspheric
Bilberry
Ceaselessly
Cracking
Crocus

Etc.

Actual Output:



Pass/Fail: Pass, looking at the screenshots, we can see that each port now organizes ships by name. It is important to note that this does NOT organize ships in the queue by name.

Test Case 4:

Test Case: Sort aSPab.txt by Queue name (following results of previous test case)

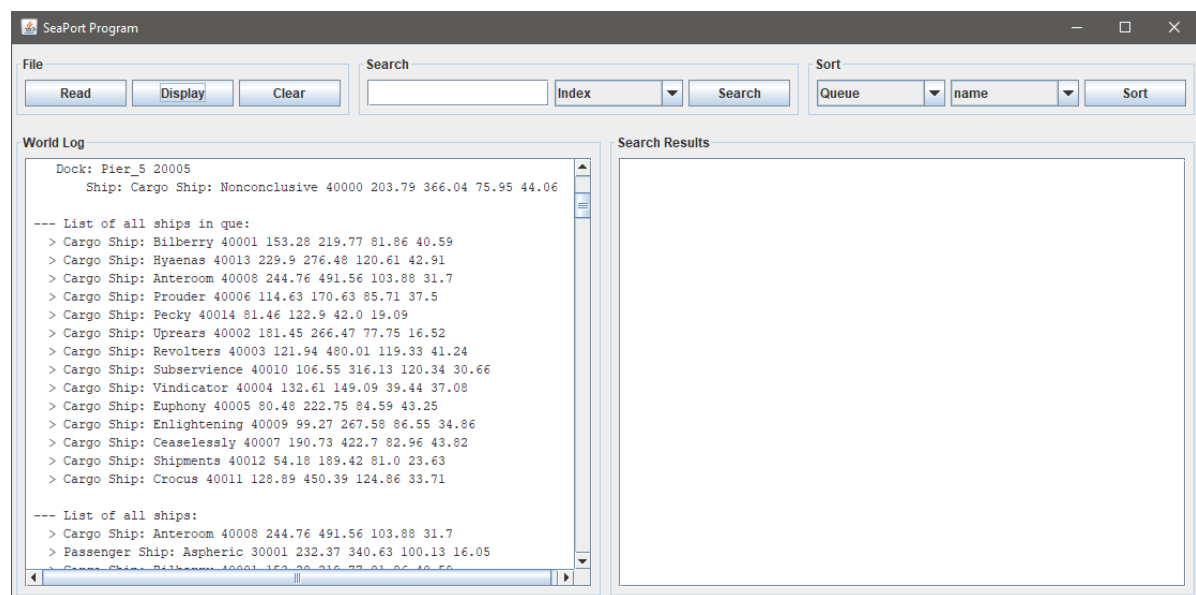
Selected Input: aSPab.txt

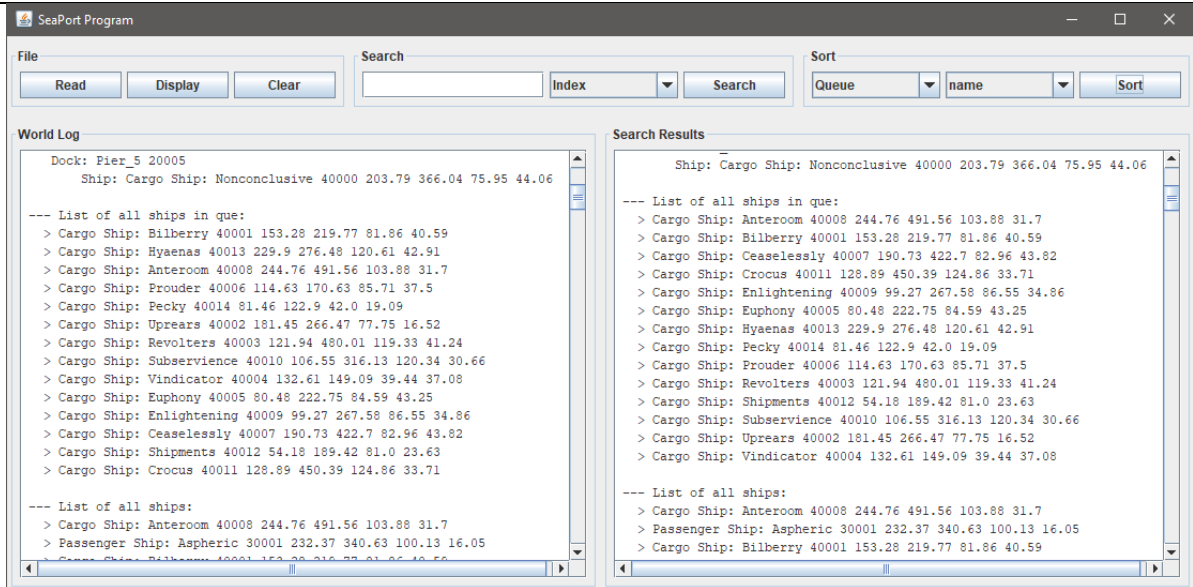
Expected Output: (Bangpakong Port)

Anteroom
Bilberry
Ceaselessly
Crocus
Enlightening

Etc.

Actual Output:





Pass/Fail: Pass, looking at the screenshots, we can see that the ships in the queue are now sorted by name.

Test Case 5:

Test Case: Sort aSPab.txt by Queue weight (following results of previous test case)

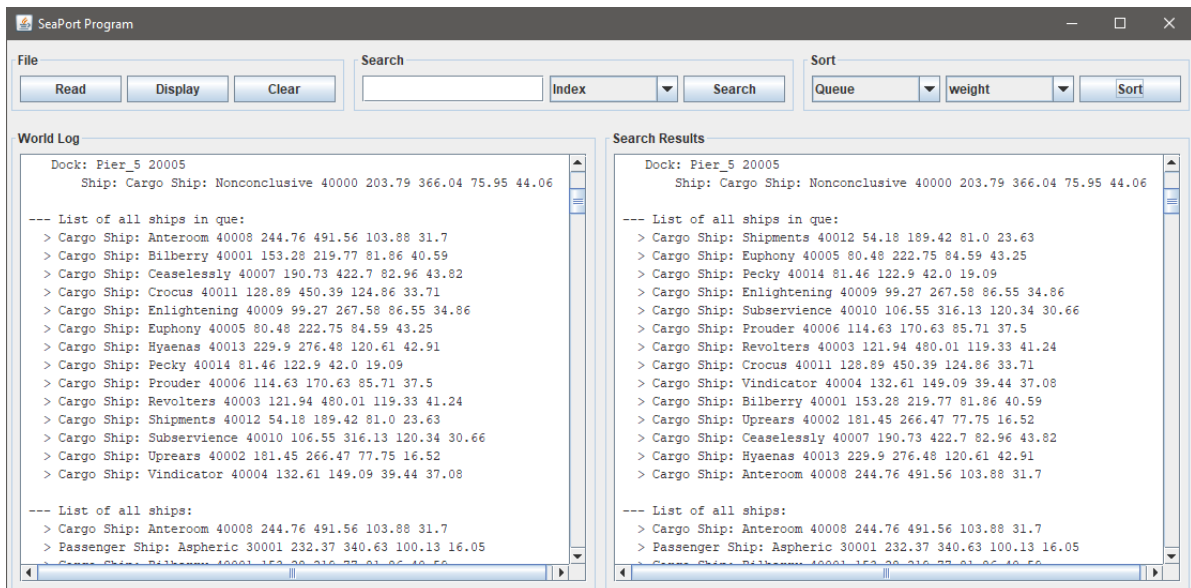
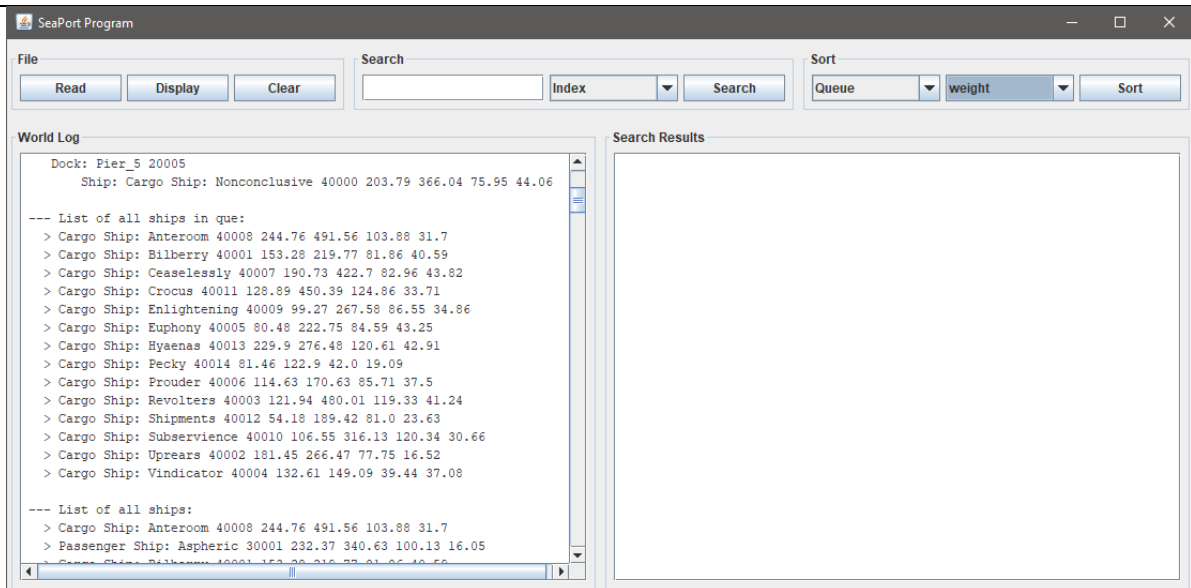
Selected Input: aSPab.txt

Expected Output: (Bangpakong Port)

54.18
80.48
81.46
99.27

Etc.

Actual Output:



Pass/Fail: Pass, looking at the screenshots, we can see that the ships in the queue are now sorted by weight (first decimal number following the ship).

Test Case 6:

Test Case: Sort aSPab.txt by Queue length (following results of previous test case)

Selected Input: aSPab.txt

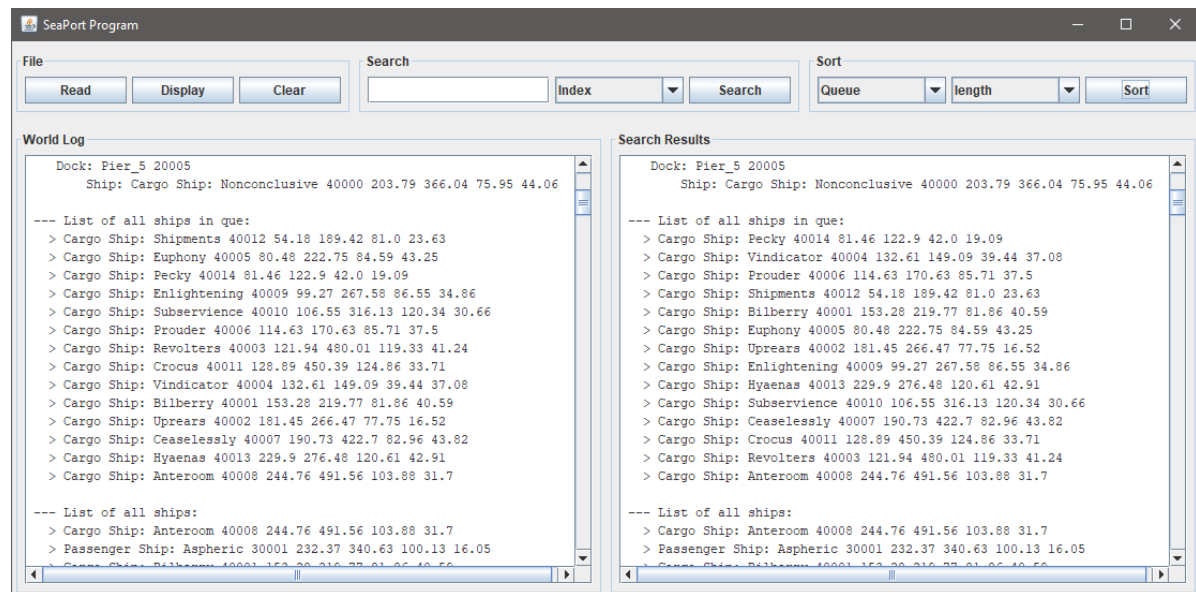
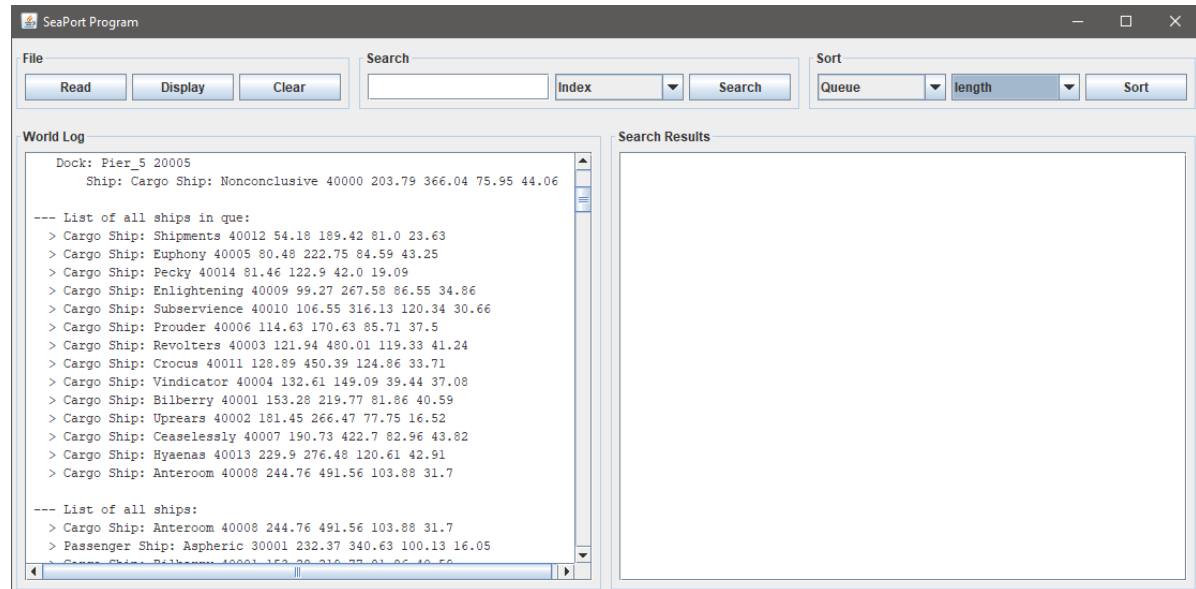
Expected Output: (Bangpakong Port)

122.9

132.61
170.63
189.42

Etc.

Actual Output:



Pass/Fail:

Pass, looking at the screenshots, we can see that the ships in the queue are now sorted by length (second decimal number following the ship).

Test Case 7:

Test Case: Sort aSPab.txt by Queue draft (following results of previous test case)

Selected Input: aSPab.txt

Expected Output: (Bangpakong Port)

39.44

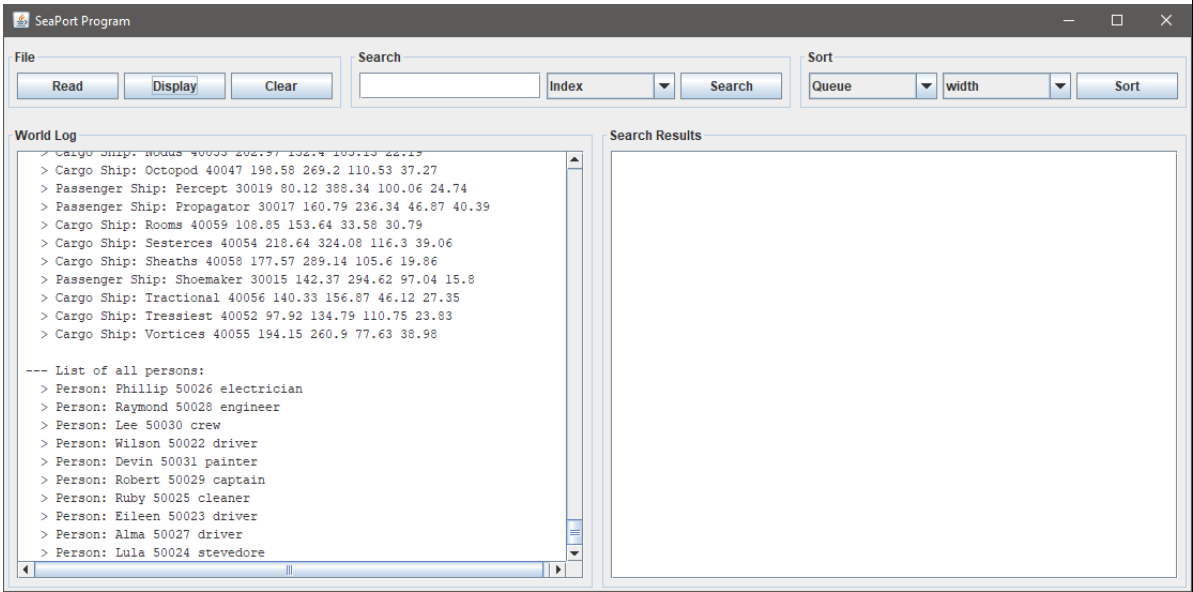
42.0

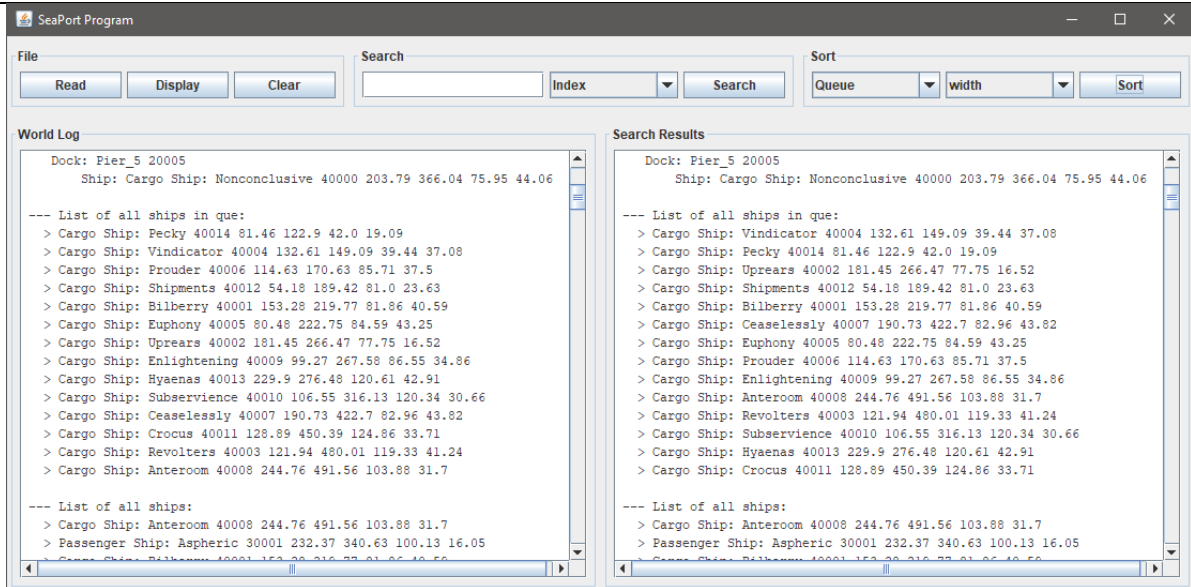
77.75

81.0

Etc.

Actual Output:





Pass/Fail: Pass, looking at the screenshots, we can see that the ships in the queue are now sorted by width (third decimal number following the ship).

Test Case 8:

Test Case: Sort aSPab.txt by Queue draft (following results of previous test case)

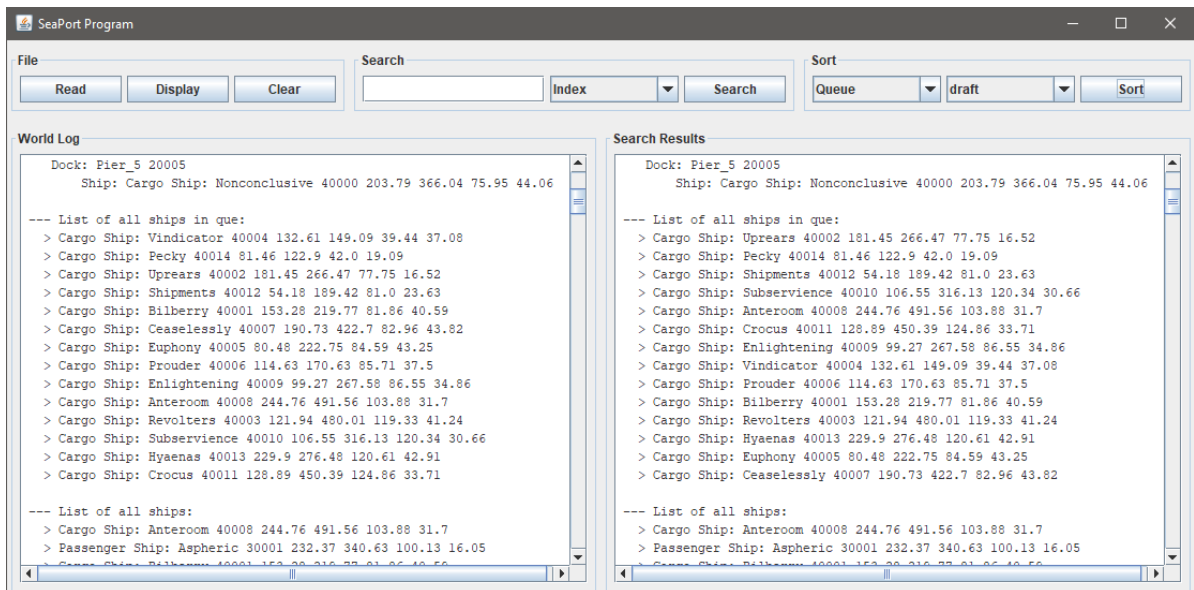
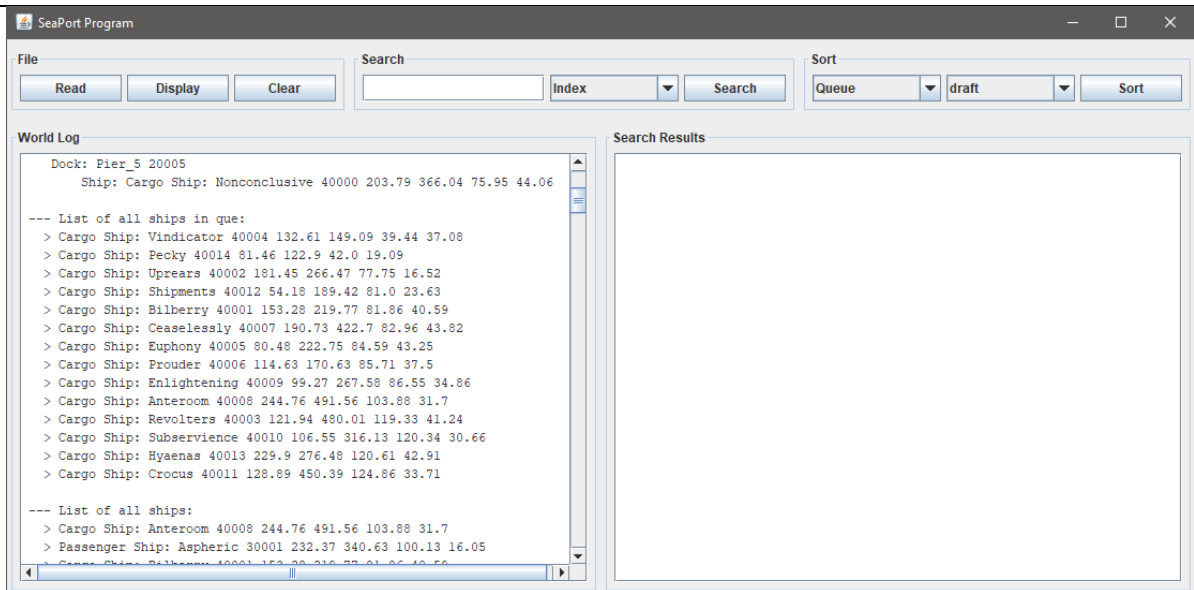
Selected Input: aSPab.txt

Expected Output: (Bangpakong Port)

16.52
19.09
23.63
30.66

Etc.

Actual Output:



Pass/Fail: Pass, looking at the screenshots, we can see that the ships in the queue are now sorted by draft (last decimal number following the ship).

Test Case 9:

Test Case: Sort aSPab.txt by People name (following results of previous test case)

Selected Input: aSPab.txt

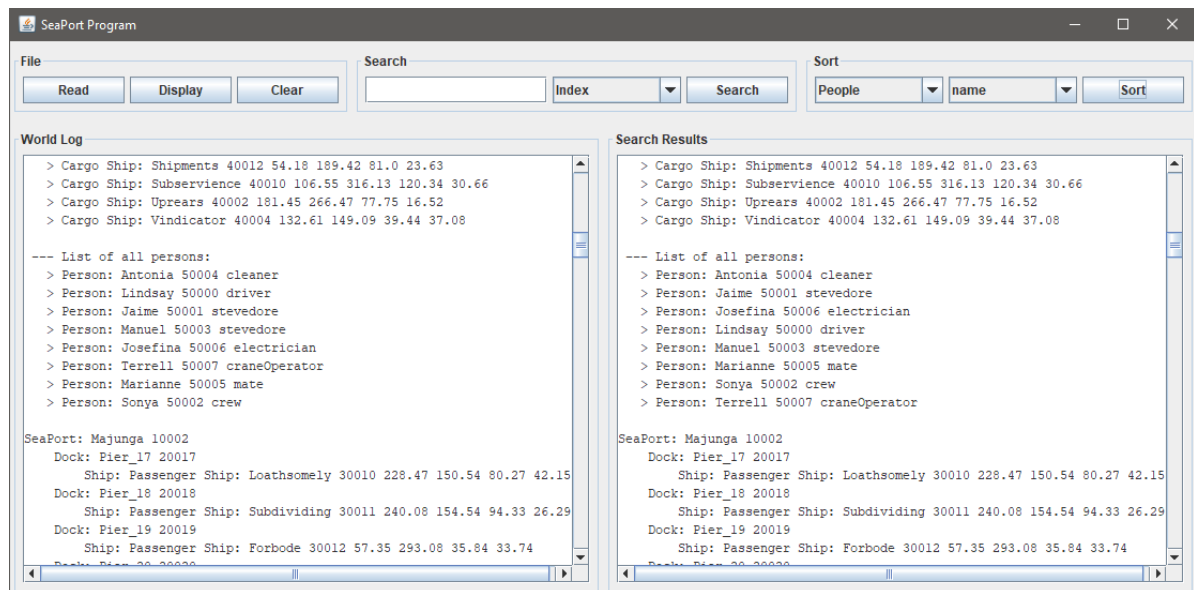
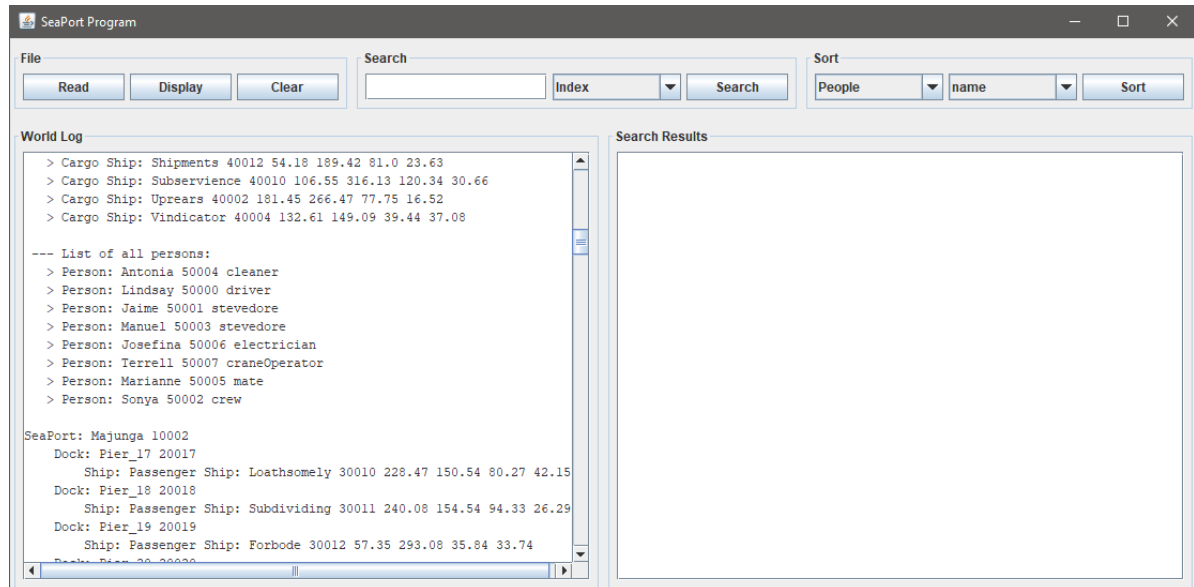
Expected Output: (Bangpakong Port)

Antonia

Jaime
Josefina
Lindsay

Etc.

Actual Output:



Pass/Fail:

Pass, looking at the screenshots, we can see that the people are now sorted by name.

5

Reflection and Lessons Learned

Reflect on your experience completing this project and the lessons you learned:

Overall, I enjoyed this project a lot. I previously had very little experience with comparators in Java. This project forced me to explore the JDK and research the proper way to implement a comparator. However, I believe that I have implemented the comparators correctly. This project provided a great challenge for me, and I learned a lot during the process. Please let me know if there is anything I could improve upon!