Anthony Borza

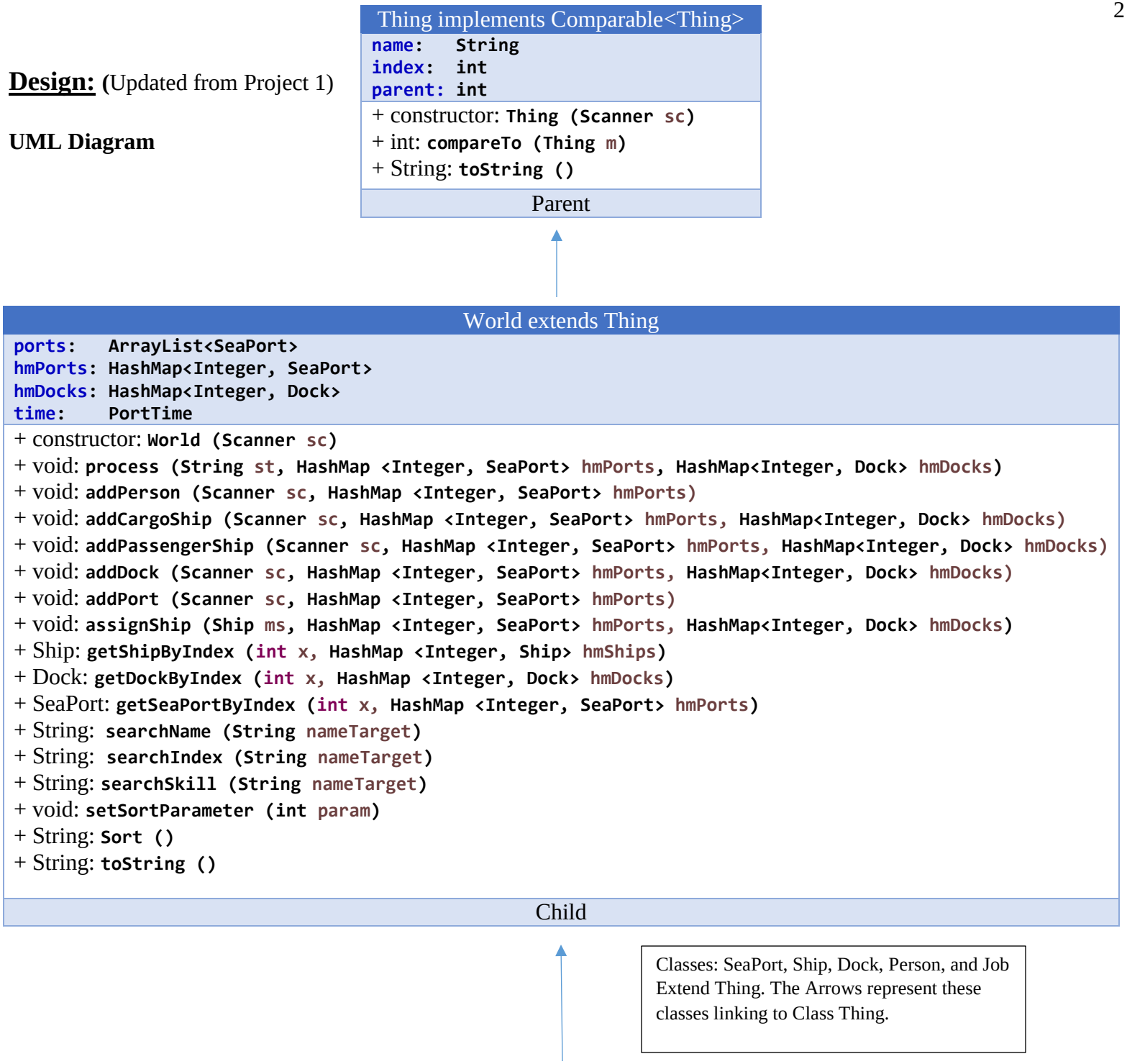
Due Date: 11/20/16

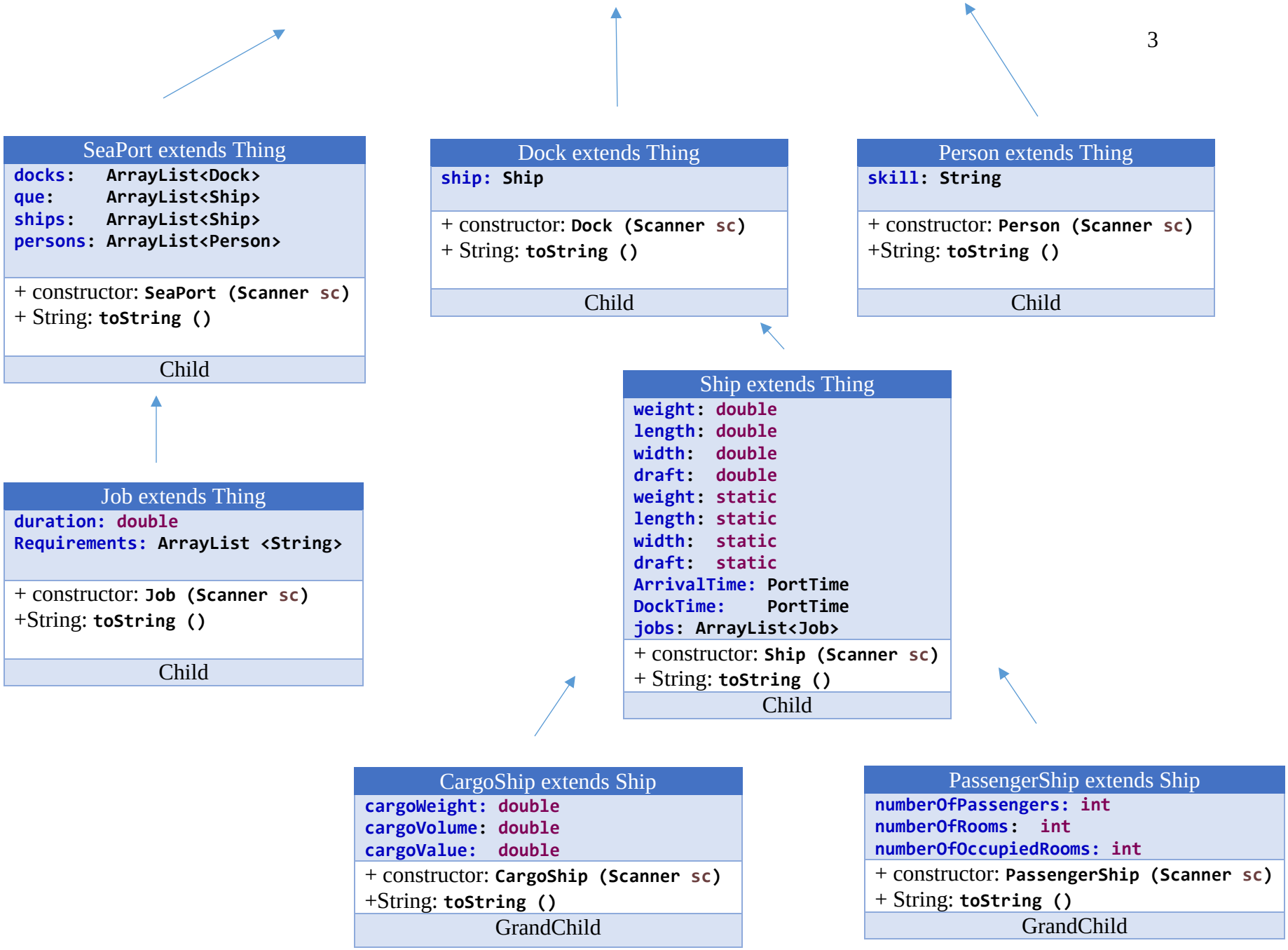
Project 2: SeaPorts: (Extends off Project 1)

CMSC 335 7980

I. Design: (Updated from Project 1)

UML Diagram




SeaPort extends Thing
docks: ArrayList<Dock>
que: ArrayList<Ship>
ships: ArrayList<Ship>
persons: ArrayList<Person>
+ constructor: SeaPort (Scanner sc)
+ String: toString ()
Child
Dock extends Thing
ship: Ship
+ constructor: Dock (Scanner sc)
+ String: toString ()
Child
Person extends Thing
skill: String
+ constructor: Person (Scanner sc)
+String: toString ()
Child
Job extends Thing
duration: double
Requirements: ArrayList <String>
+ constructor: Job (Scanner sc)
+String: toString ()
Child
Ship extends Thing
weight: double
length: double
width: double
draft: double
weight: static
length: static
width: static
draft: static
ArrivalTime: PortTime
DockTime: PortTime
jobs: ArrayList<Job>
+ constructor: Ship (Scanner sc)
+ String: toString ()
Child
CargoShip extends Ship
cargoWeight: double
cargoVolume: double
cargoValue: double
+ constructor: CargoShip (Scanner sc)
+String: toString ()
GrandChild
PassengerShip extends Ship
numberOfPassengers: int
numberOfRooms: int
numberOfOccupiedRooms: int
+ constructor: PassengerShip (Scanner sc)
+ String: toString ()
GrandChild

SeaPortProgram extends JFrame

```
-JTextArea textArea;
-JTextField searchTargets;
-JLabel searchName;
-JRadioButton name;
-JRadioButton index;
-JRadioButton skill;
-JRadioButton weight;
-JRadioButton length;
-JRadioButton width;
-JRadioButton draft;
-JButton searchButton;
-JButton sortButton;
-JButton readFile;
-JPanel panel1, panel2;
```

+ constructor: `SeaPortProgram ()`
+ void: `inputFile ()`
- `ActionListener: Search ()`
- `ActionListener: Search ()`

PortTime.java

```
time: int
```

+ constructor: `PortTime (int t)`
+ compareTo: `PortTime (PortTime other)`
+ String: `toString ()`

II. User's Guide: (Updated from Project 1)

- **How would a user start and run the project?**

Steps for using my program:

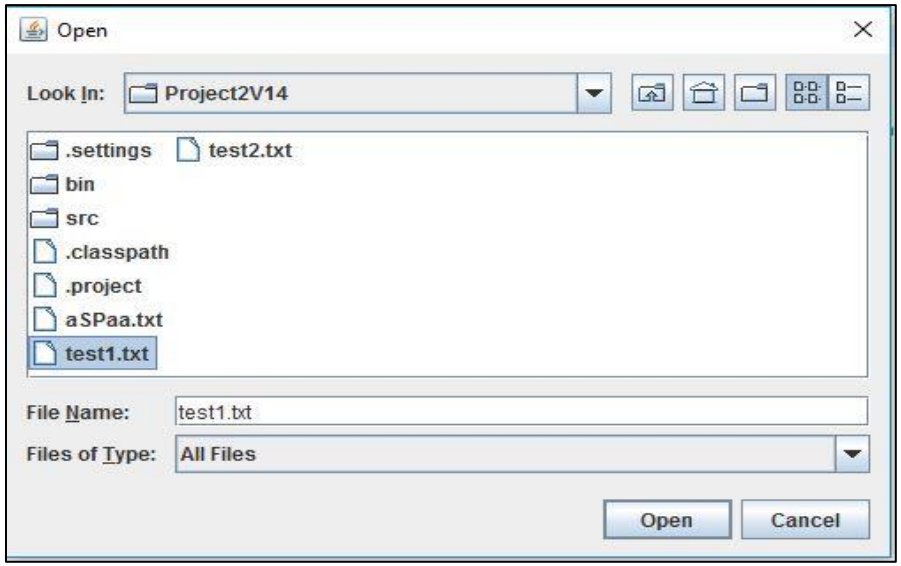
1.) Open a IDE of your choice. (Ex: Eclipse will be used in this Demonstration).

2.) Assuming you have all the files. Open the "SeaPortProgram.java class".

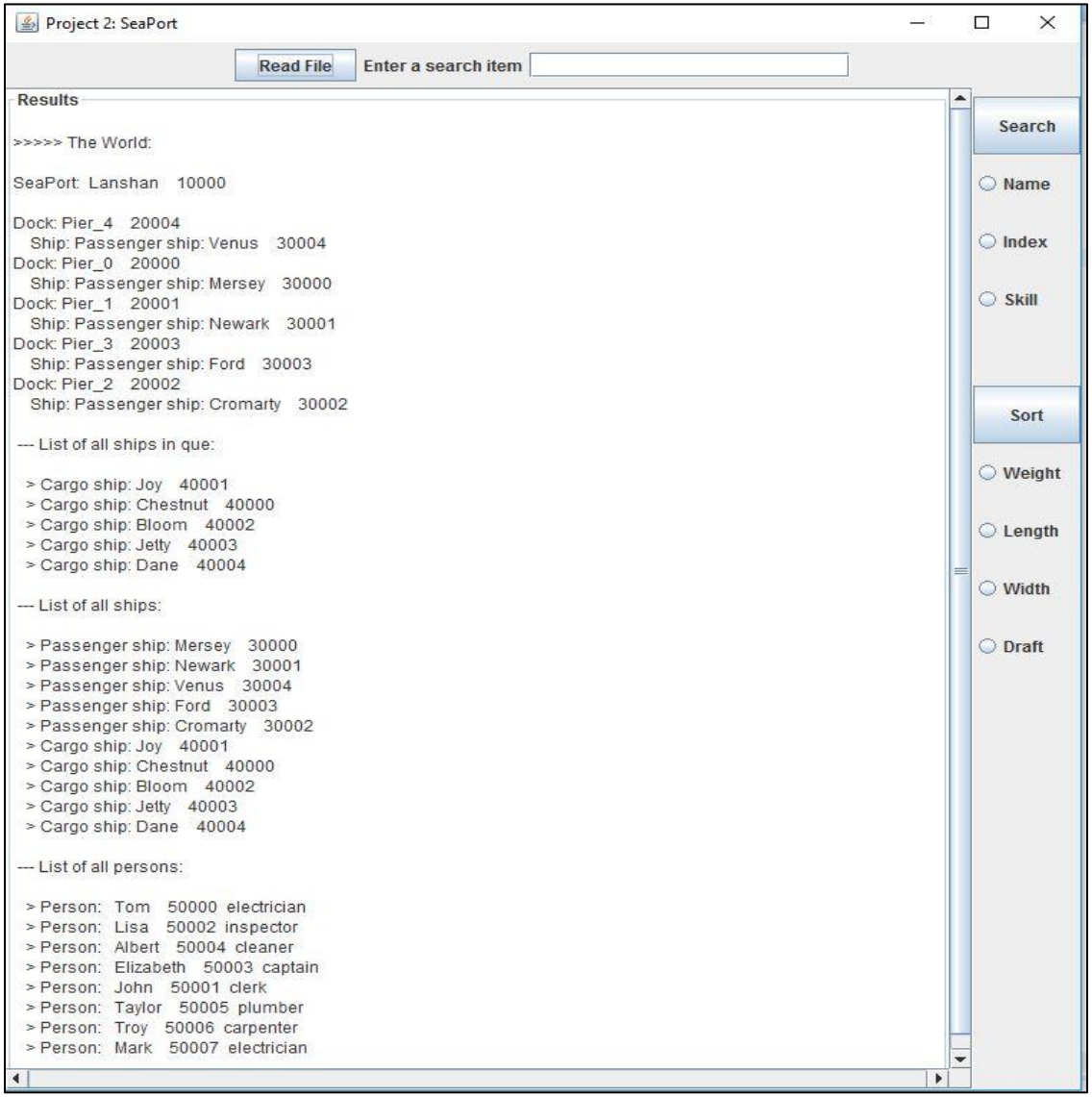
3.) Click on the following button in eclipse to run the program:

(This can be found at the top-left of eclipse)

4.) You will now be prompted to select a txt file. (Ex. test1.txt, will be used)

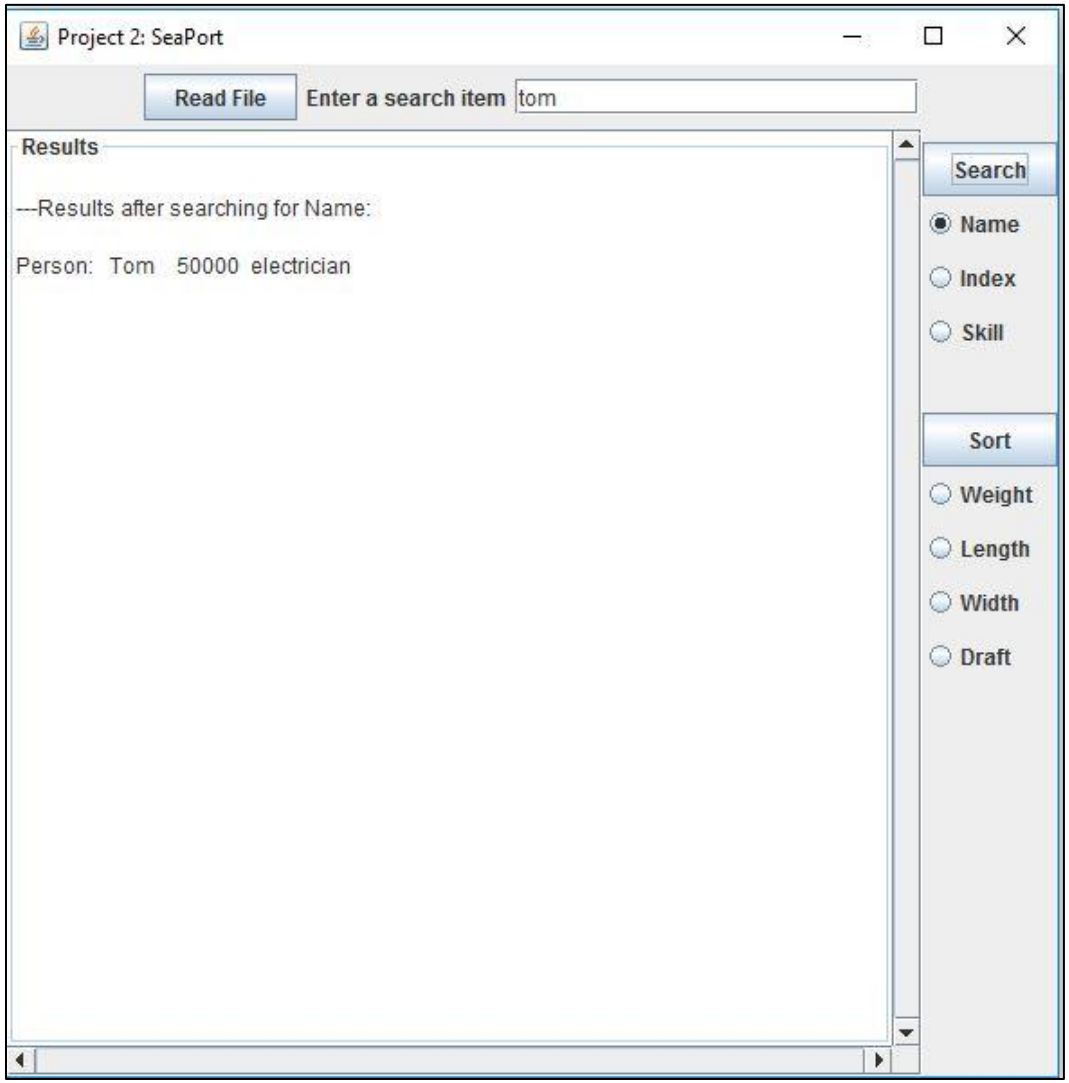


5.) Click the open button, and the following will display:

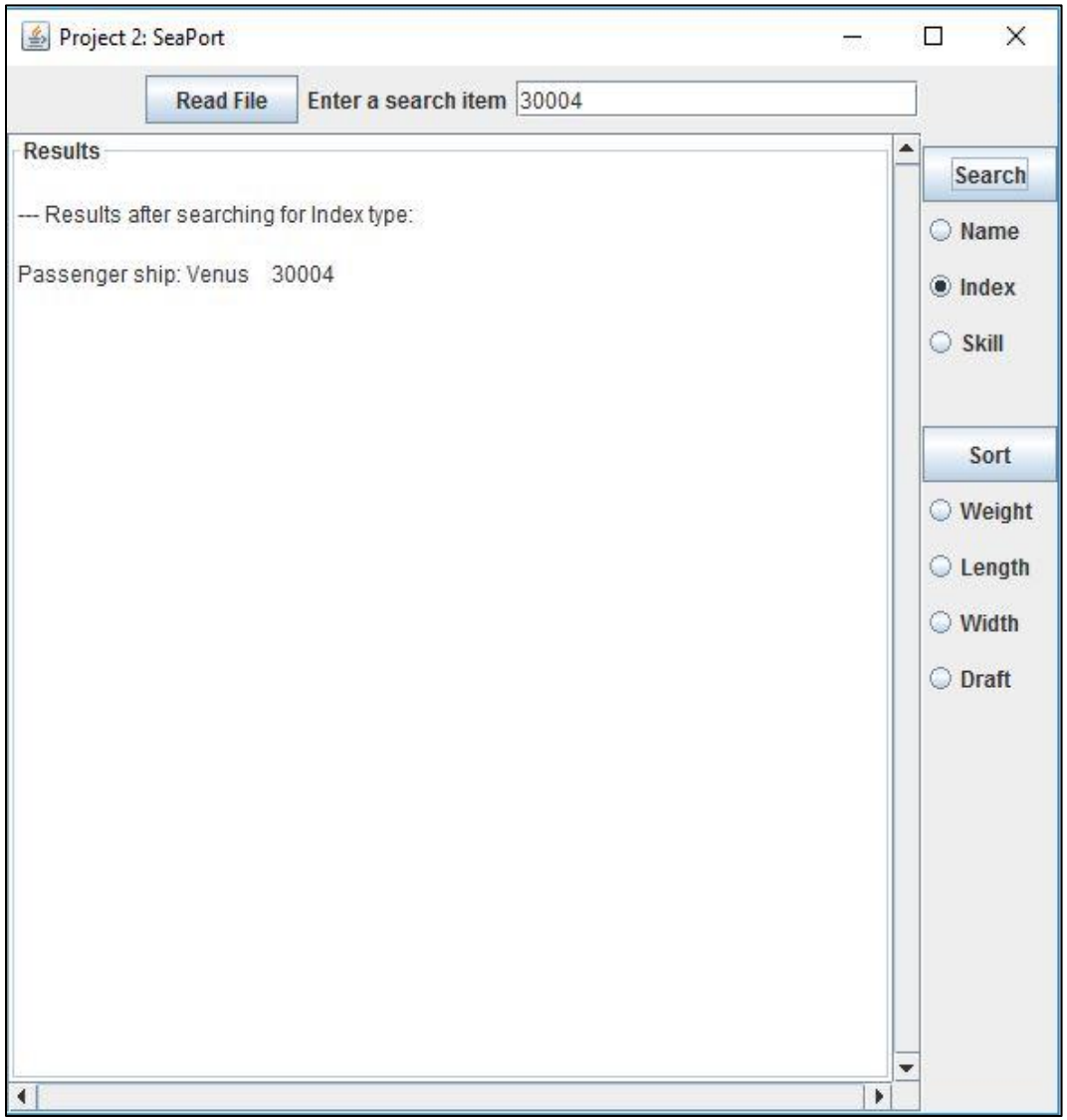


6.) To search for an item in the file, you will have three options: Name, Index, and Skill. (it is not case sensitive)

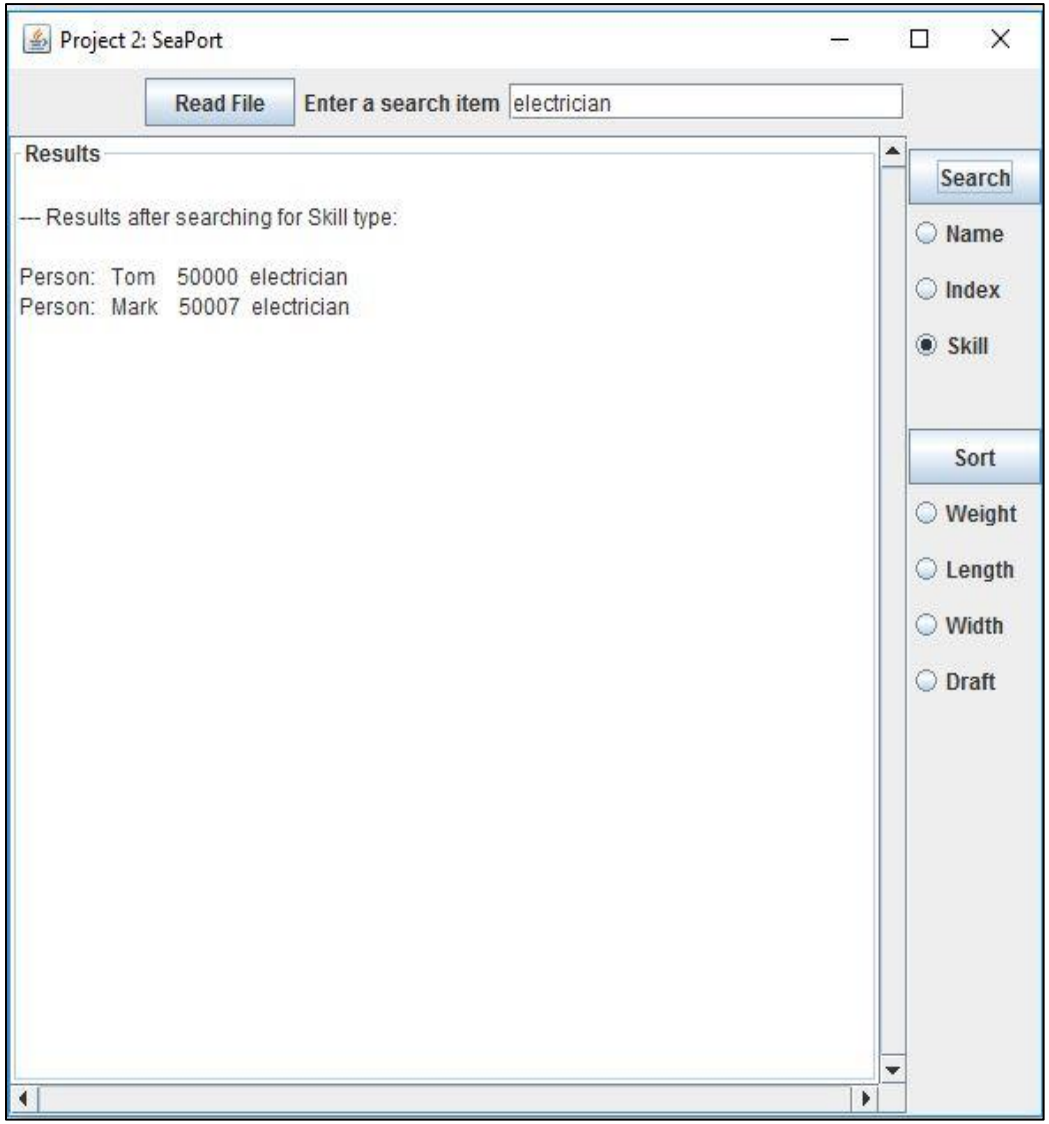
Name Search Example: Search for Tom



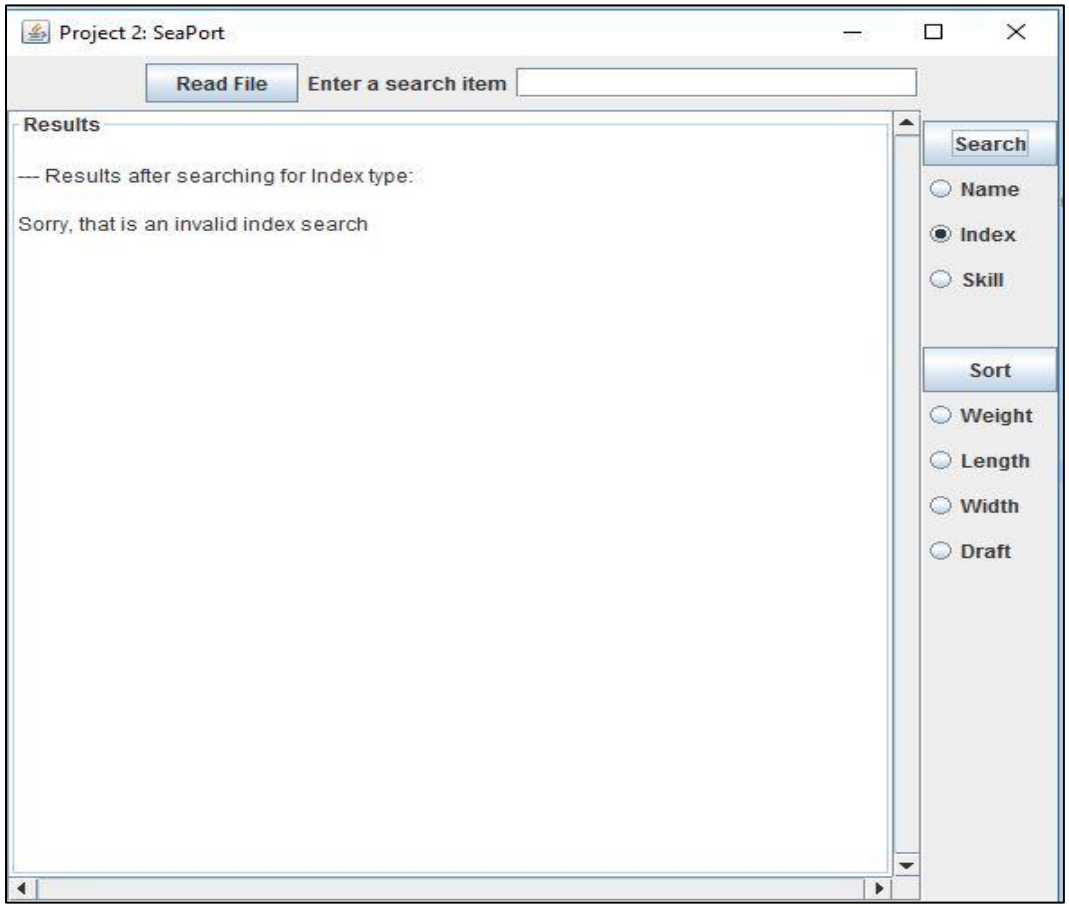
Index Search Example: Search for 30004



Skill Search Example: Search for electrician

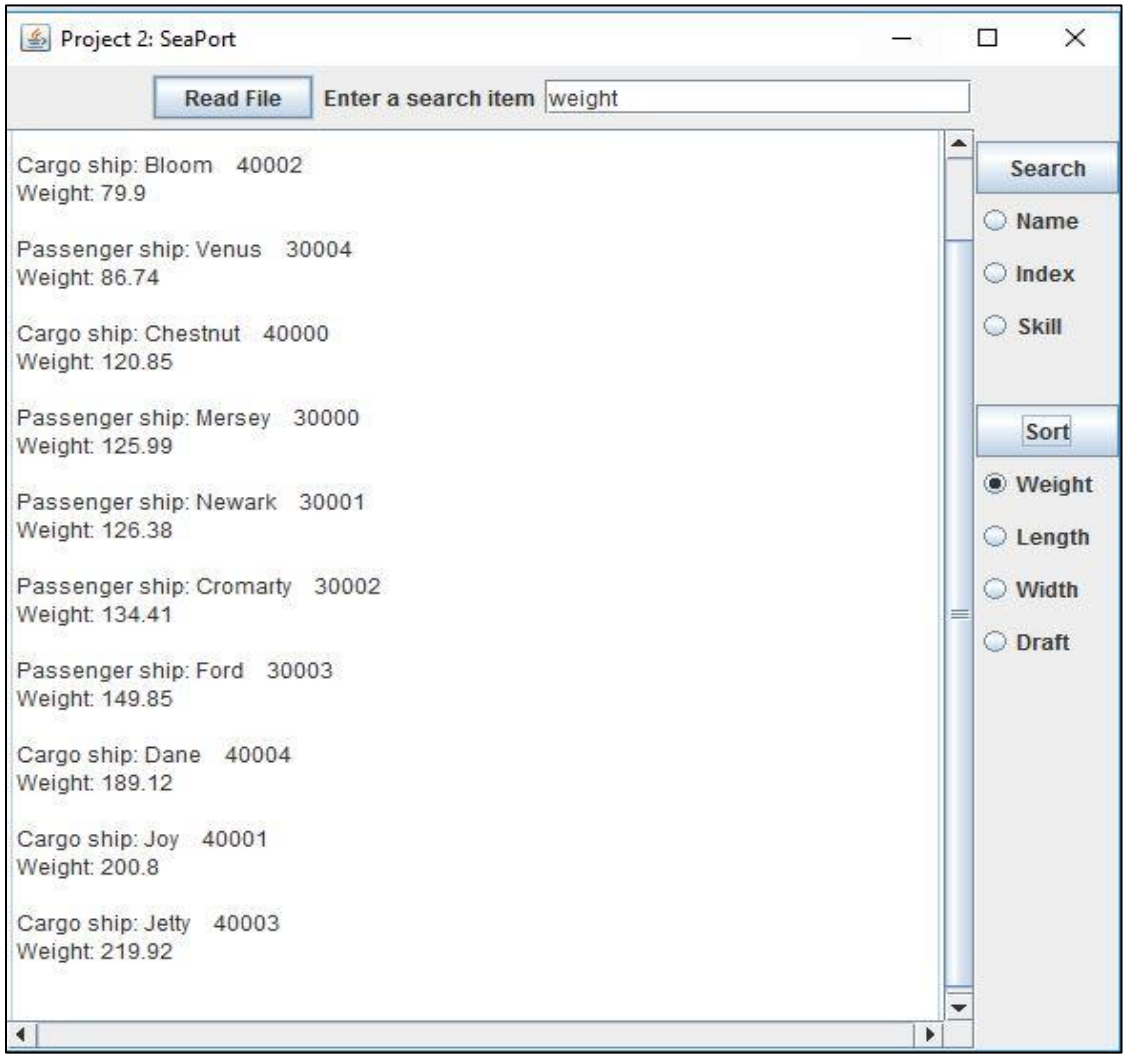


7.) If an invalid search is entered in you will see the following: (Sorry, that is an invalid search. Please try again.)

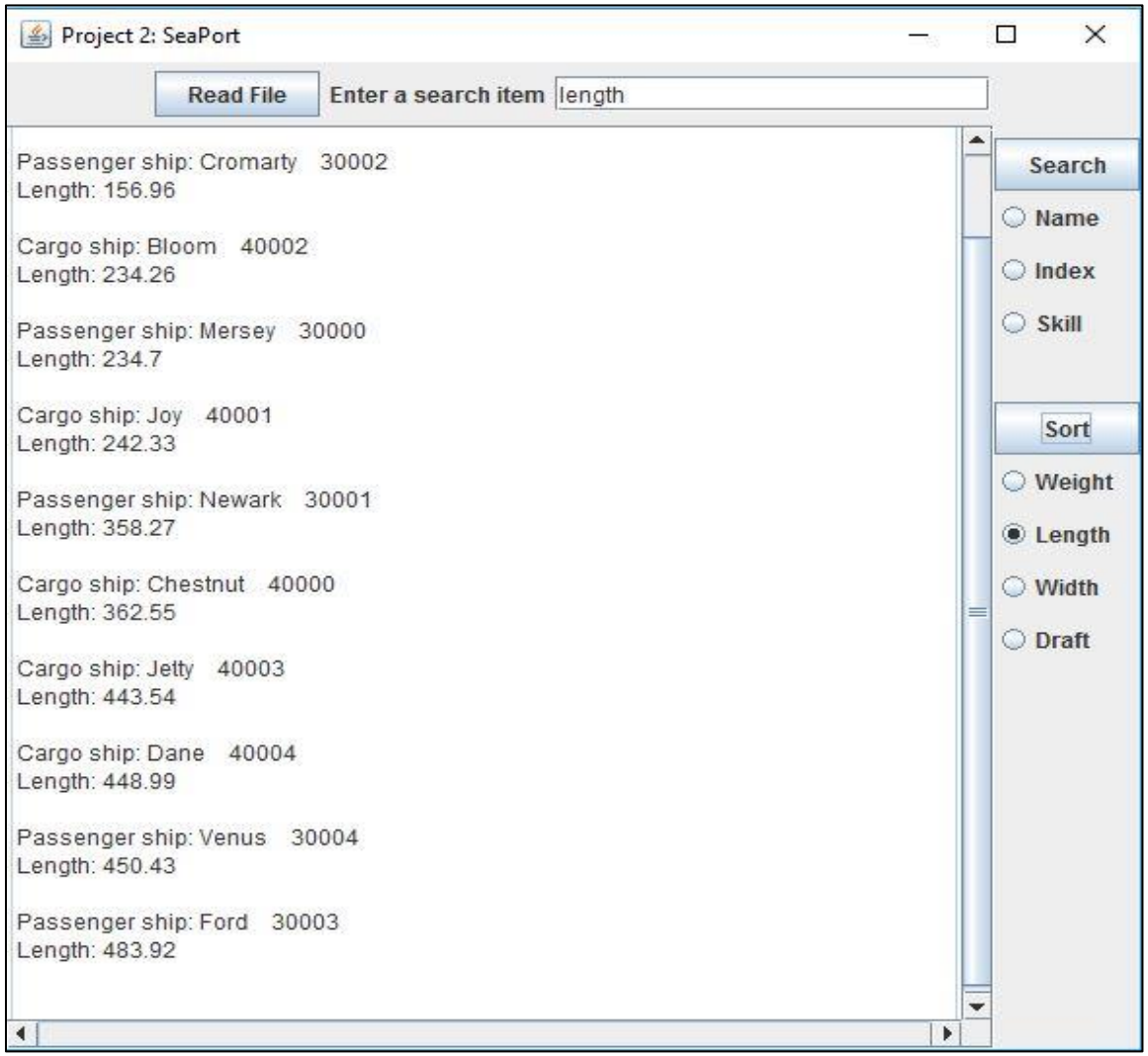


8.) To sort for an item in ascending order you will have three options: Ships Weight, Length, Width, and Draft (it is not case sensitive) **(New step added)**

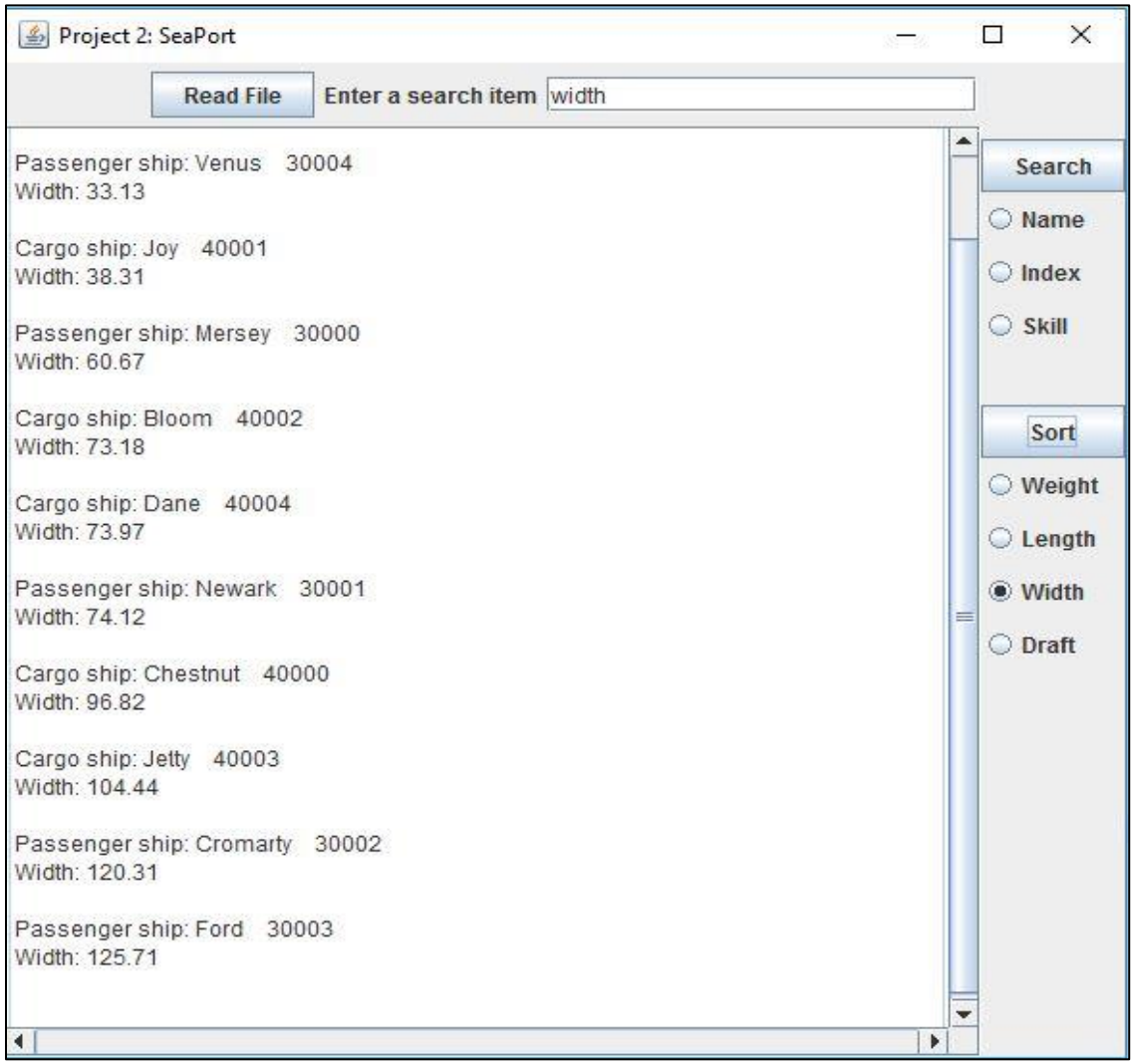
Sort by Ship Weight: Enter in weight



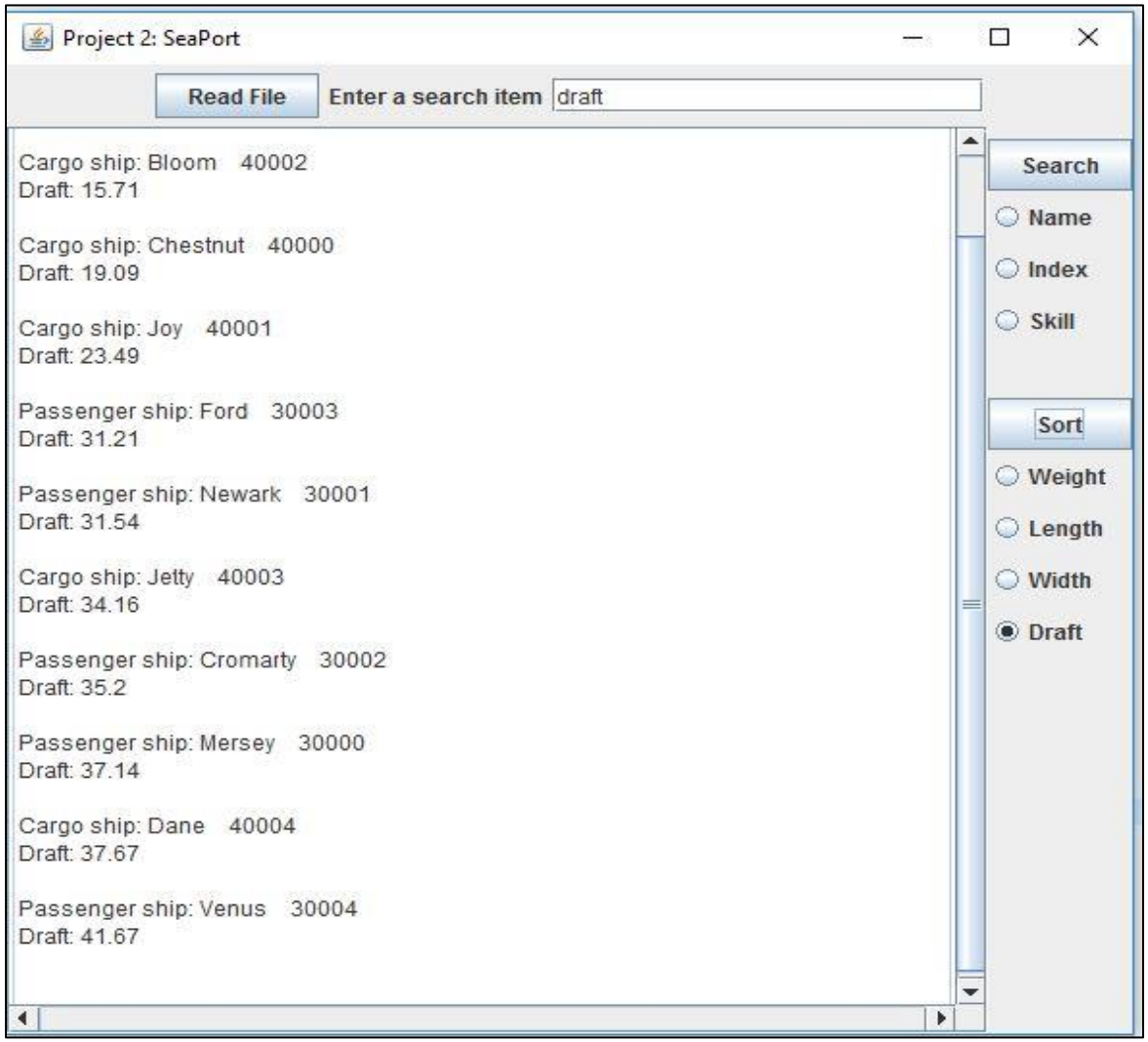
Sort by Ship Length: Enter in length



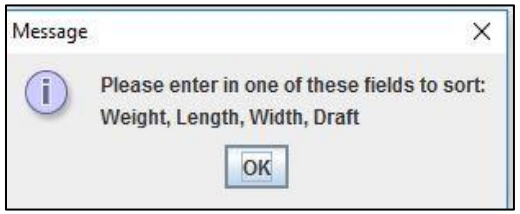
Sort by Ship Width: Enter in width



Sort by Ship Draft: Enter in draft



9.) If an invalid sort is entered in you will see the following:



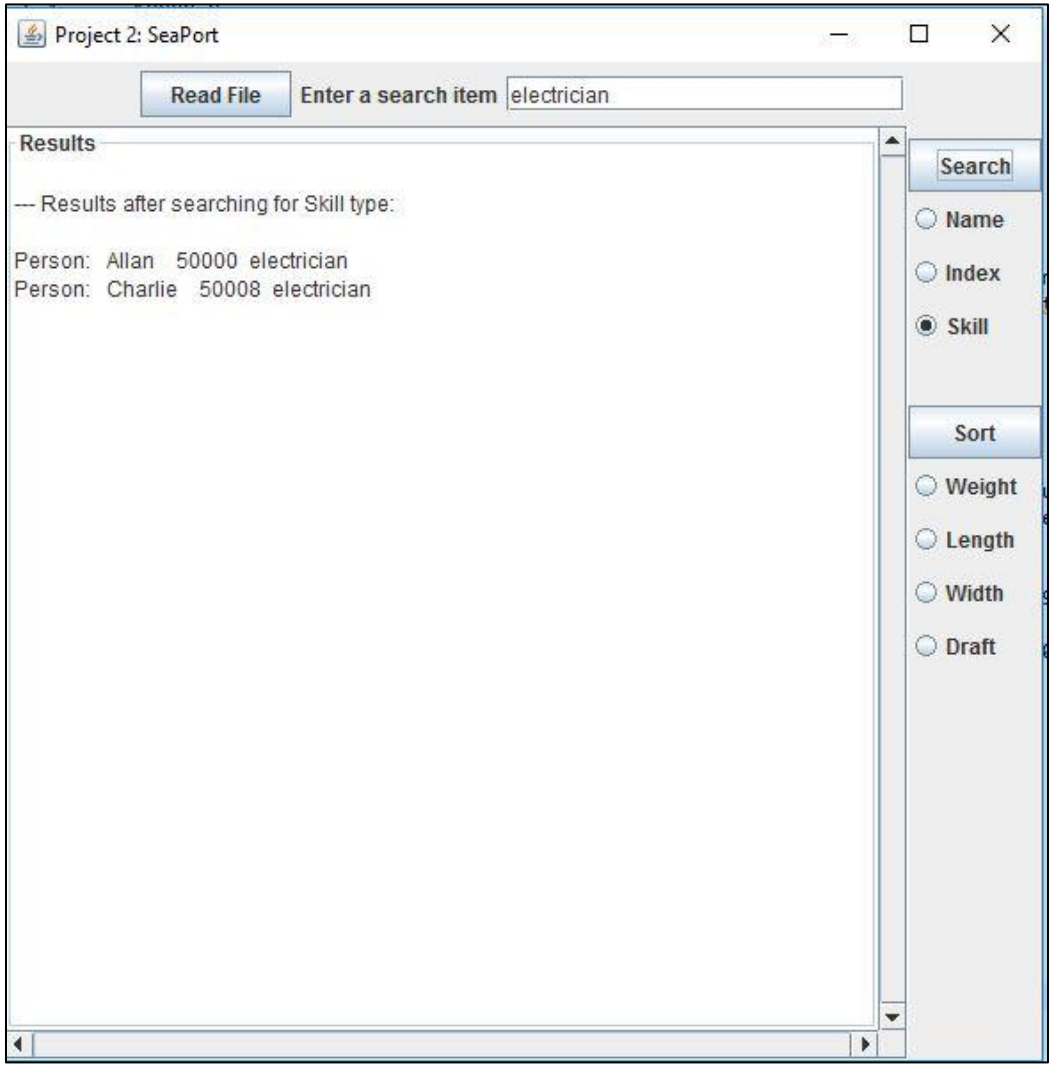
10.) To open and search another file, select the Read File Button. This will display the same dialog box in step 4.

11.) If you followed these steps correctly, you have successfully used my program the right way.

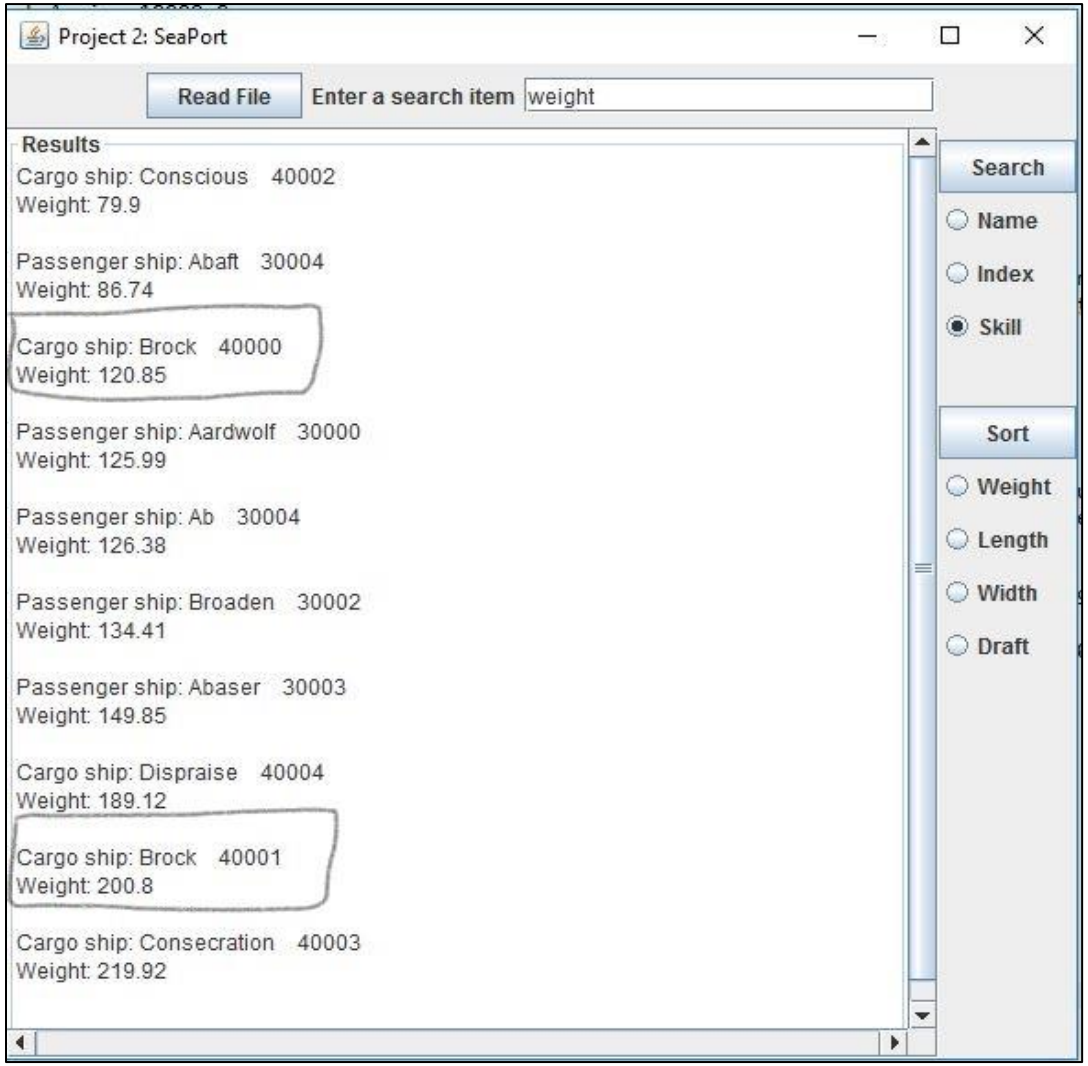
- **Special Features:**
    - User friendly interface
    - Allows you to read in another file without having to run the program again
    - Uses radio buttons to search for the fields in the file
    - Uses radio buttons to sort for the ships weight, length, width, and draft
    - Displays the results nicely
    - Is not case sensitive for what is being search. (Ex: both John, or john, will return the same result).
    - Returns duplicate values for name, index, skill, weight, length, width, and draft (text2.txt contains duplicates)

- **Screen Shots:**
    - See screenshots of the program below for duplicates of both searching and sorting

Example of Searching and Returning Duplicates: (Ex: searching electrician and selecting skill)



Example of Sorting and Returning Duplicates: (Ex: sorting the ships weight and selecting weight)



III. **Test Plan:** (Update from Project 1)

- **What do you expect the project to do?**

  Project 2 was an extension off project 1. The first thing I did before extending on to project 1 was to make sure that I understood what the instructions were asking for. The goal of this assignment was to implement a HashMap around the methods that we created in project 1. This required me to research the best way to implement a HashMap since it was the first time I ever did one. I found Oracles documentation as the best resource for understanding HashMap's. The next thing I did was create a compare method that will sort the ships by weight, length, width, and draft. I then added a sort function in the World class to implement this compare method. Finally, if we go to the UI class SeaPortProgram, I added radio buttons for weight, length, width, and draft. The sorting is done using the sort method in the World class, as well as, the compareWith switch statement in the Ship class. I knew that if I did this I would get the results I want. The program is now able to sort by these fields and display them nicely on the GUI output display.

IV. **Lessons Learned:** (Update from Project 1)

After completing project 2 there are many additional things that were learned from project 1. Like in project 1, I realize how important it is to work with multiple classes using inheritance and polymorphism. This project taught me how to use HashMap's, and compare methods to sort elements in a file. As you can see, the program is now able to sort the ships weight, length, width, and draft in ascending order. Like in previous programming classes, and in project 1, understanding what the problem is asking, and how to implement a solution to solve the problem only comes through trial and error. Project 2 involved a lot of trial and error. I would have to say the hardest part of this project was implementing the HashMap to support efficient linking of the classes that were used in project 1. What made this difficult was getting it to work with the readFile method, or in my case the process method, which is in the World class. After I could get this to work, implementing the comparable interface, which compares values and then returns an int which the tells if the values compare less than, equal, or greater than

was less difficult since I have used this in previous programming classes. All in all, like in project 1, I spent a total of 35 hours on it this week, and factored in about 5 hours a day, or more.