Anthony Borza

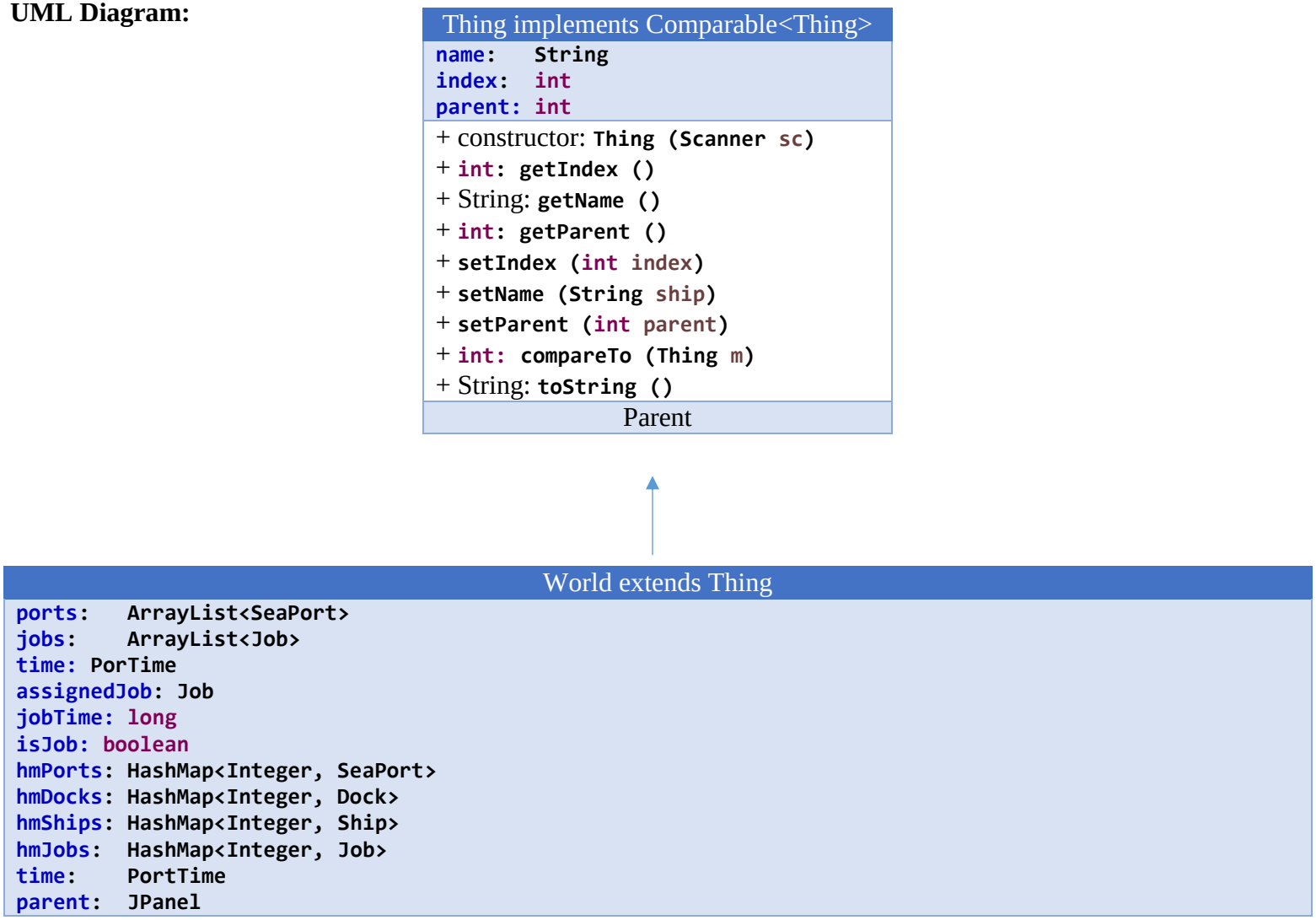
Due Date: 12/16/16

Final Project 4: SeaPorts: (Extends off Project 3)

CMSC 335 7980

I. Design: (Updated from Project 3)

UML Diagram:



+ constructor: `World (Scanner sc)`
+ void: `process (String st, HashMap <Integer, SeaPort> hmPorts, HashMap<Integer, Dock> hmDocks)`
+ void: `addPerson (Scanner sc, HashMap <Integer, SeaPort> hmPorts)`
+ void: `addCargoShip (Scanner sc, HashMap <Integer, SeaPort> hmPorts, HashMap<Integer, Dock> hmDocks)`
+ void: `addPassengerShip (Scanner sc, HashMap <Integer, SeaPort> hmPorts, HashMap<Integer, Dock> hmDocks)`
+ void: `addDock (Scanner sc, HashMap <Integer, SeaPort> hmPorts, HashMap<Integer, Dock> hmDocks)`
+ void: `addPort (Scanner sc, HashMap <Integer, SeaPort> hmPorts)`
+ void: `addJob (Scanner sc)`
+ void: `assignJob (Job job)`
- void: `assignJobs ()`
- void: `assignPersonsToJobs ()`
+ void: `assignShip (Ship ms, HashMap <Integer, SeaPort> hmPorts, HashMap<Integer, Dock> hmDocks)`
+ Ship: `getShipByIndex (int x, HashMap <Integer, Ship> hmShips)`
+ Dock: `getDockByIndex (int x, HashMap <Integer, Dock> hmDocks)`
+ SeaPort: `getSeaPortByIndex (int x, HashMap <Integer, SeaPort> hmPorts)`
+ String: `searchName (String nameTarget)`
+ String: `searchIndex (String nameTarget)`
+ String: `searchSkill (String nameTarget)`
+ void: `setSortParameter (int param)`
+ String: `Sort ()`
+ String: `toString ()`

Child

Classes: SeaPort, Ship, Dock, Person, and Job Extend Thing. The Arrows represent these classes linking to Class Thing.

SeaPort extends Thing

`docks:` `ArrayList<Dock>`
`que:` `ArrayList<Ship>`
`ships:` `ArrayList<Ship>`
`persons:` `ArrayList<Person>`

+ constructor: `SeaPort (Scanner sc)`
+ ArrayList<Dock>: `getDocks ()`
+ ArrayList<Ship>: `getQue ()`
+ ArrayList<Ship>: `getShips ()`

Dock extends Thing

`ship:` `Ship`

+ constructor: `Dock (Scanner sc)`
+ Ship: `getShip ()`
+ void: setShip `(Ship ship)`
+ String: `toString ()`

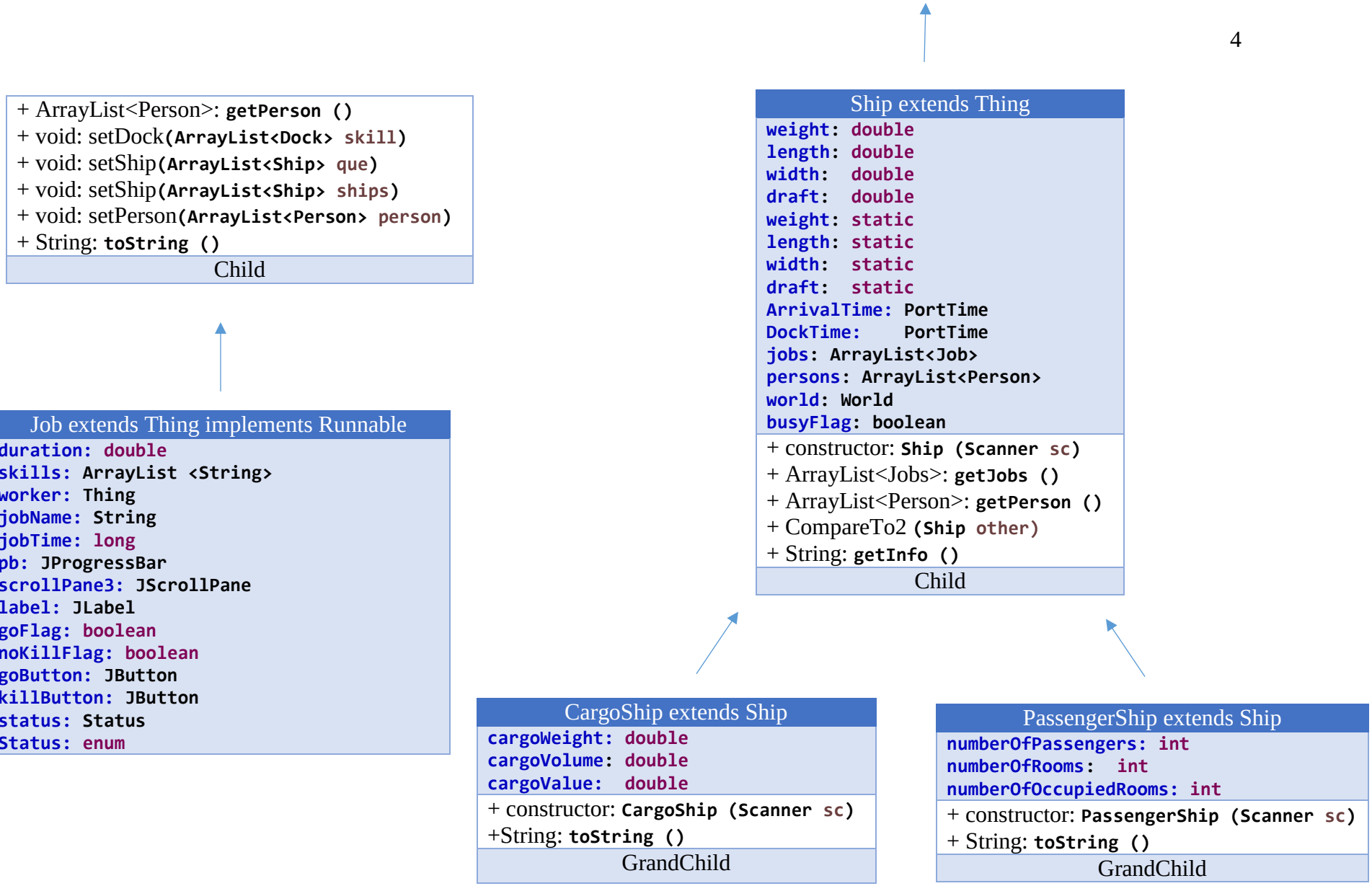
Child

Person extends Thing

`skills:` `ArrayList <String>`
`busyFlag:` `boolean`

+ constructor: `Person (Scanner sc)`
+ ArrayList<Skill>: `getSkill ()`
+ void: setSkill `(String skill)`
+ `boolean`: `capableOf()`
+String: `toString ()`

Child


+ ArrayList<Person>: getPerson ()
+ void: setDock(ArrayList<Dock> skill)
+ void: setShip(ArrayList<Ship> que)
+ void: setShip(ArrayList<Ship> ships)
+ void: setPerson(ArrayList<Person> person)
+ 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>
persons: ArrayList<Person>
world: World
busyFlag: boolean
+ constructor: Ship (Scanner sc)
+ ArrayList<Jobs>: getJobs ()
+ ArrayList<Person>: getPerson ()
+ CompareTo2 (Ship other)
+ String: getInfo ()
Child
Job extends Thing implements Runnable
duration: double
skills: ArrayList <String>
worker: Thing
jobName: String
jobTime: long
pb: JProgressBar
scrollPane3: JScrollPane
label: JLabel
goFlag: boolean
noKillFlag: boolean
goButton: JButton
killButton: JButton
status: Status
Status: enum
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

+ constructor: **Job (HashMap<Integer, Ship> hmShips, JPanel parent, Scanner sc)**
+ void: **run ()**
+ void: **actionPerformed (ActionEvent e)**
+ double: **getDuration ()**
+ void: **addSkill ()**
+ ArrayList<String>: **getSkill ()**
- JPanel: **StatusPanel ()**
- showStatus: **(Status st)**
+ void: **run ()**
+ void: **isResourcesAvailable ()**
+ void: **markResources ()**
+ **boolean**: **requires ()**
+ String: **toString ()**

Child

PortTime.java

-time: int

+ constructor: **PortTime (int t)**
+ **int**: getTime ()
+ void: setTime **(int time)**
+ compareTo: **PortTime (PortTime other)**
+ String: **toString ()**

Skill.java

+ name: String

+ constructor: **skill (String n)**
+ String: **getName ()**
+ String: **setNmae (String n)**

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 cancelButton;
-JButton suspendButton;
-JButton resumeButton;
-JButton readFile;
-JPanel panel1, panel2, panel3, panel4, panel5;
-JProgressBar pb
Scanner sc
World world
Job job

+ constructor: **SeaPortProgram ()**
+ JPanel: **panel1 ()**
+ JPanel: **panel2 ()**
+ JPanel: **panel3 ()**
+ JPanel: **panel4 ()**
+ JPanel: **panel5 ()**
+ void: **inputFile ()**
- ActionListener: **Search ()**
- ActionListener: **Sort ()**
+ void: **populateTree (SeaPort port, DefaultMutableTreeNode top)**
+ void: **createNodes (SeaPort port, DefaultMutableTreeNode top)**

II. User's Guide: (Updated from Project 3)

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

Steps for using my program:

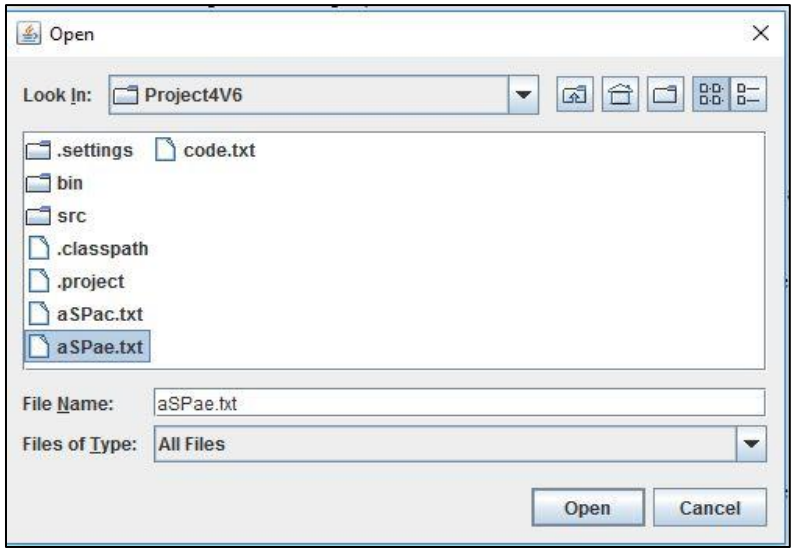
1.) Open a IDE of your choice. (Ex: Eclipse will be used in this Demonstration).

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

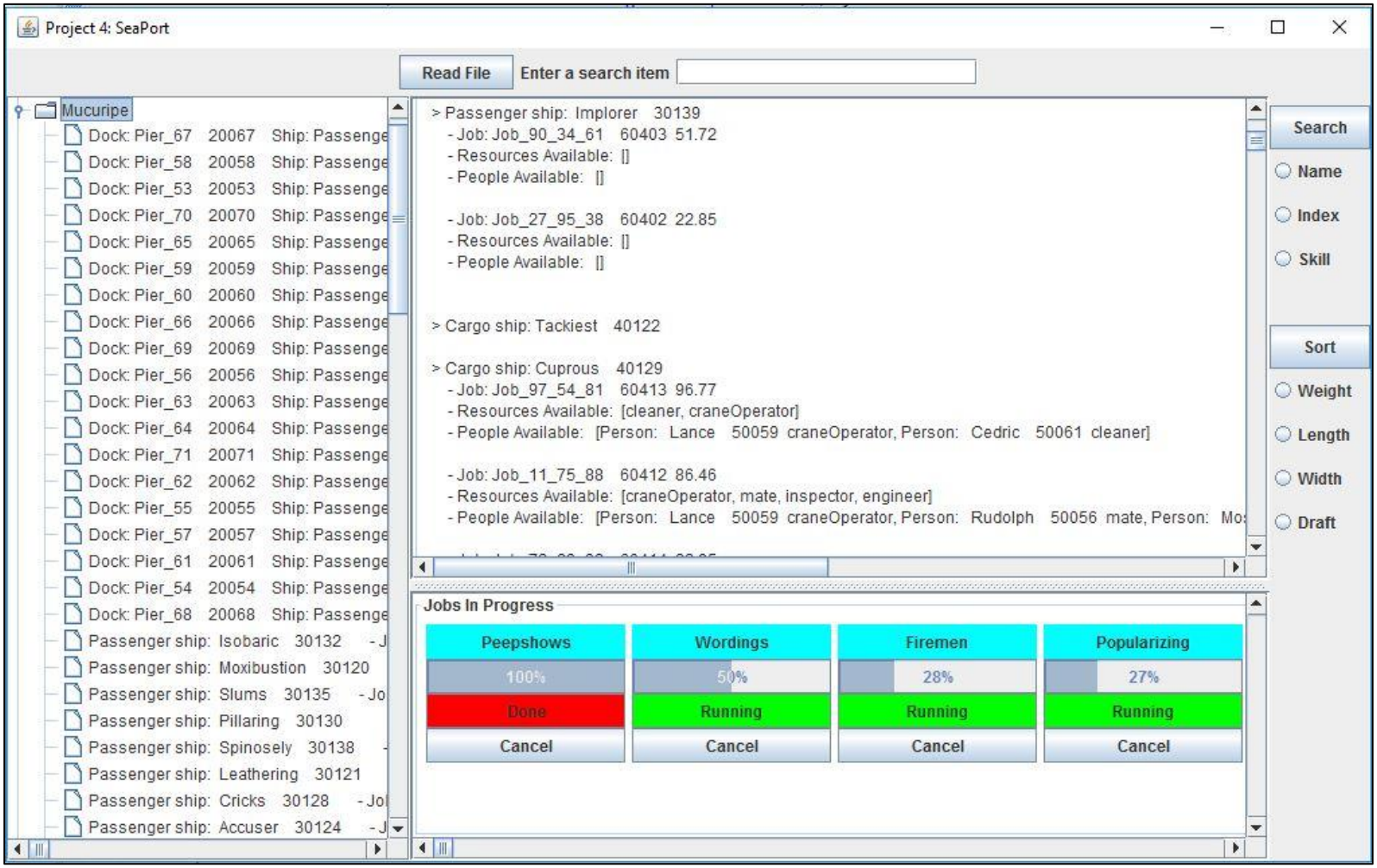
3.) Click on the following button in eclipse to run the program:

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

4.) You will now be prompted to select a txt file. (Ex. aSPae.txt, will be used, you can also use: aSPac.txt)

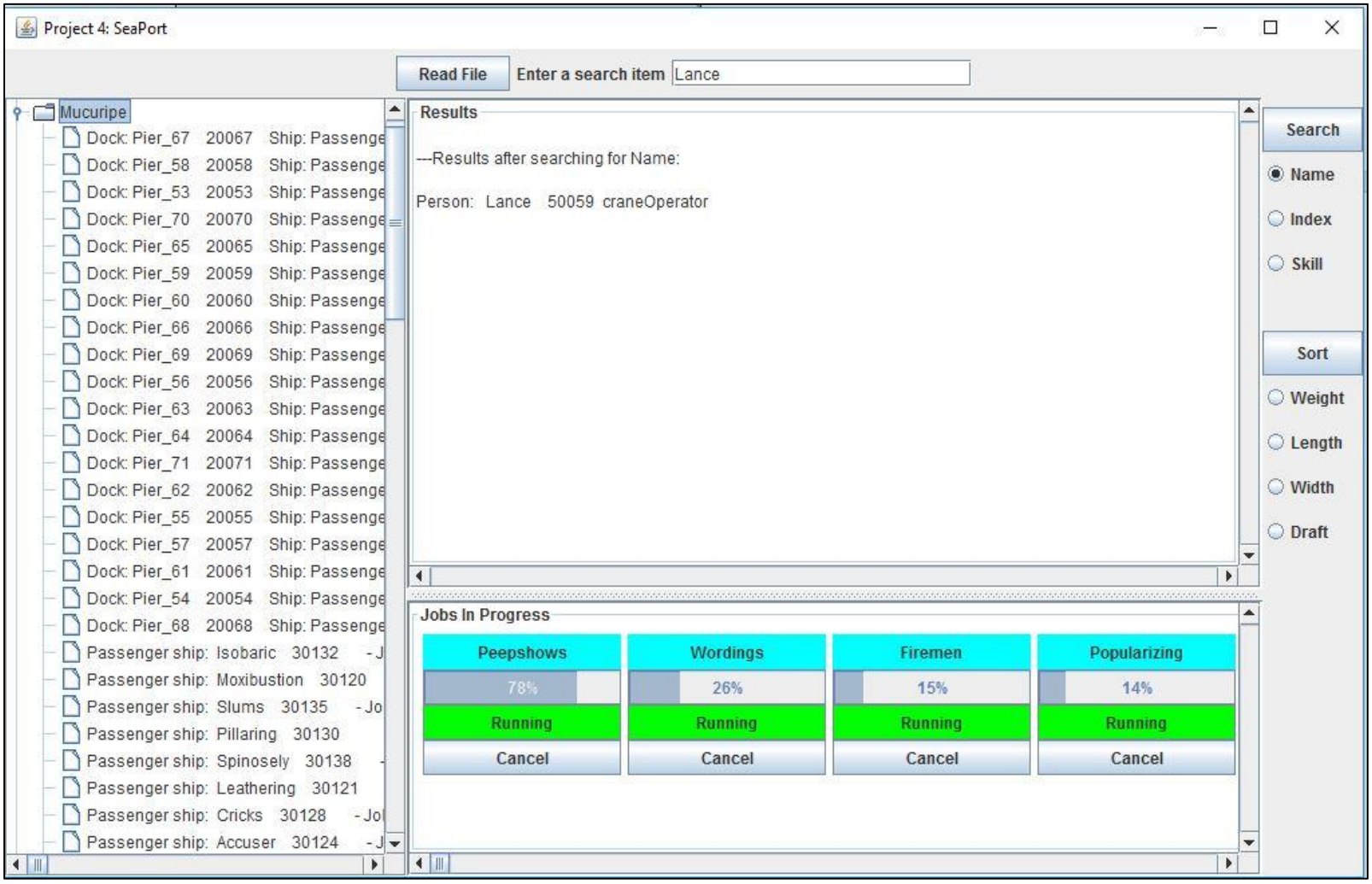


5.) Click the open button, and the following will display.
(**New Features:** JTree, TextArea box for "Jobs in Progress," a running, suspend, waiting, done button, cancel button, and a progress bar)

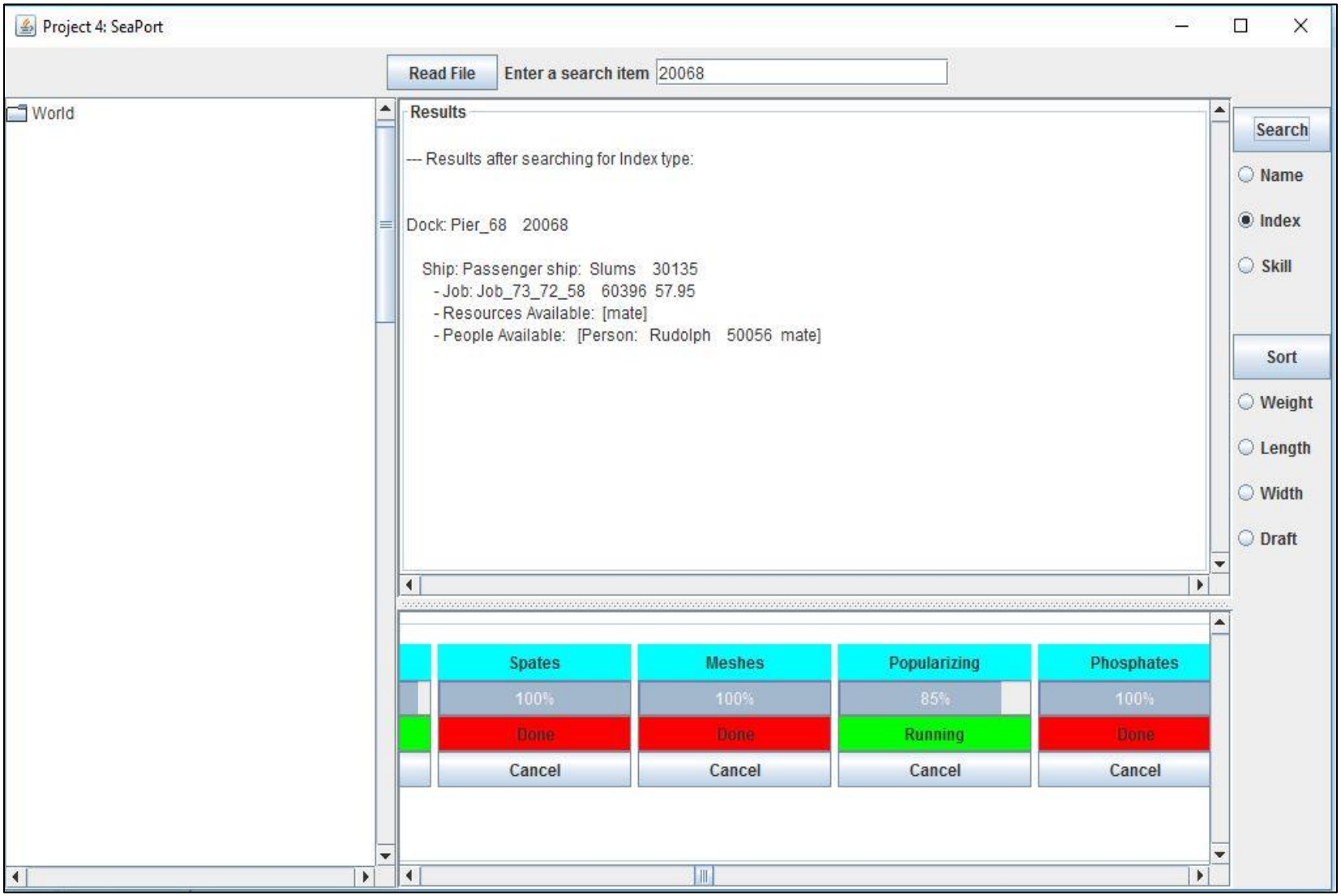


6.) To search for an item in the file, you will have three options: Name, Index, and Skill. (it is not case sensitive)

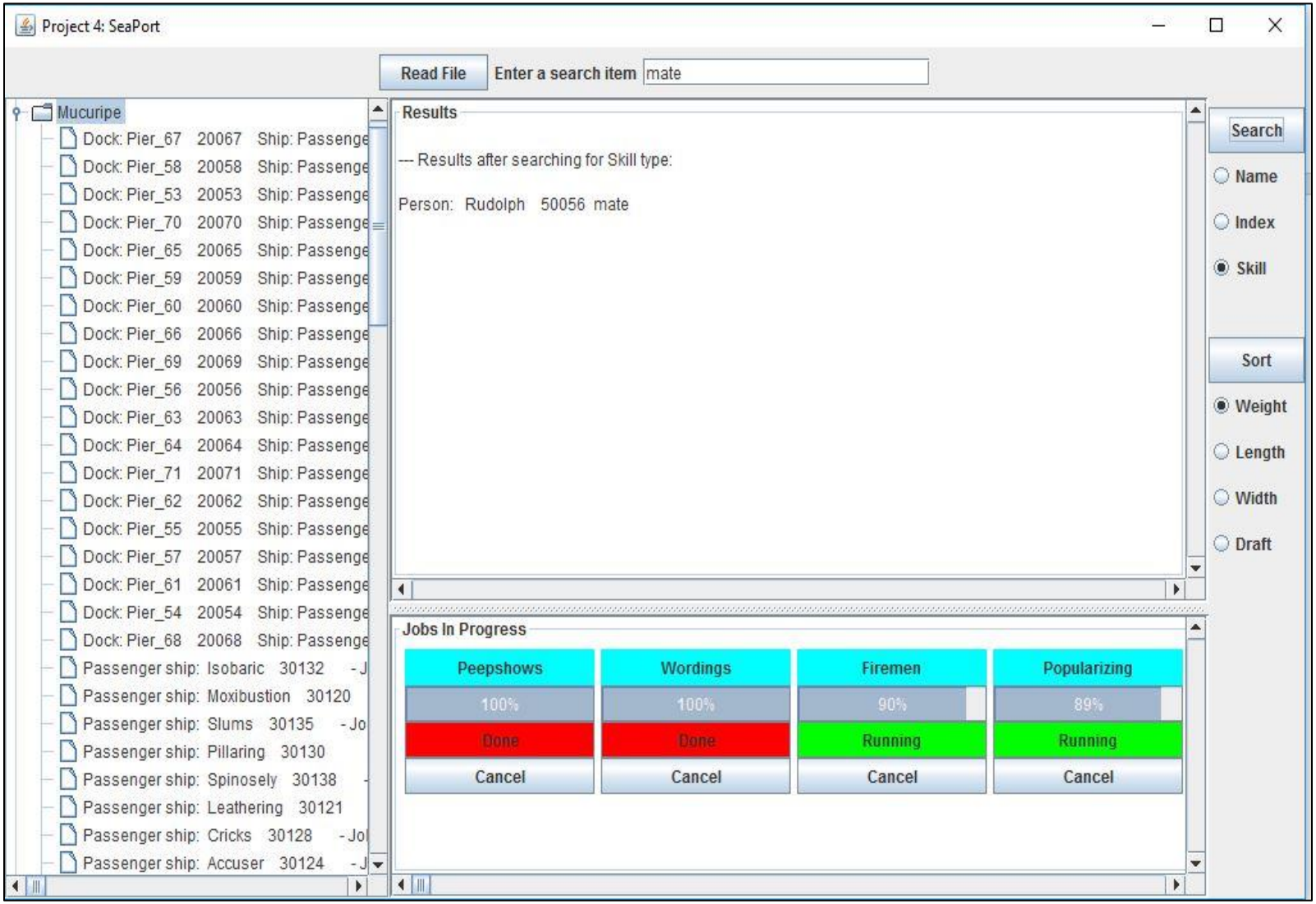
Name Search Example: Search for Lance



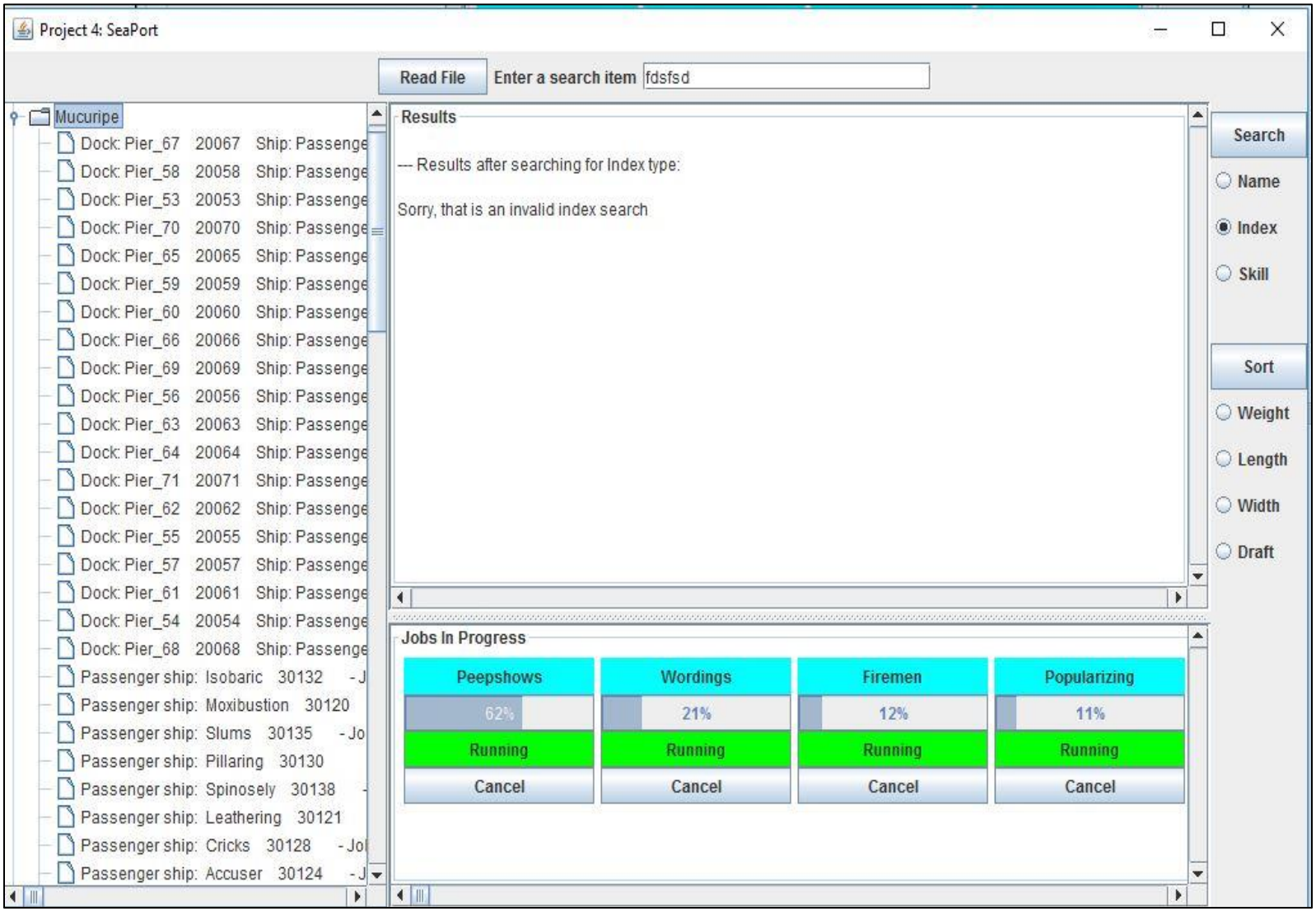
Index Search Example: Search for 20068



Search Skill: Search for mate

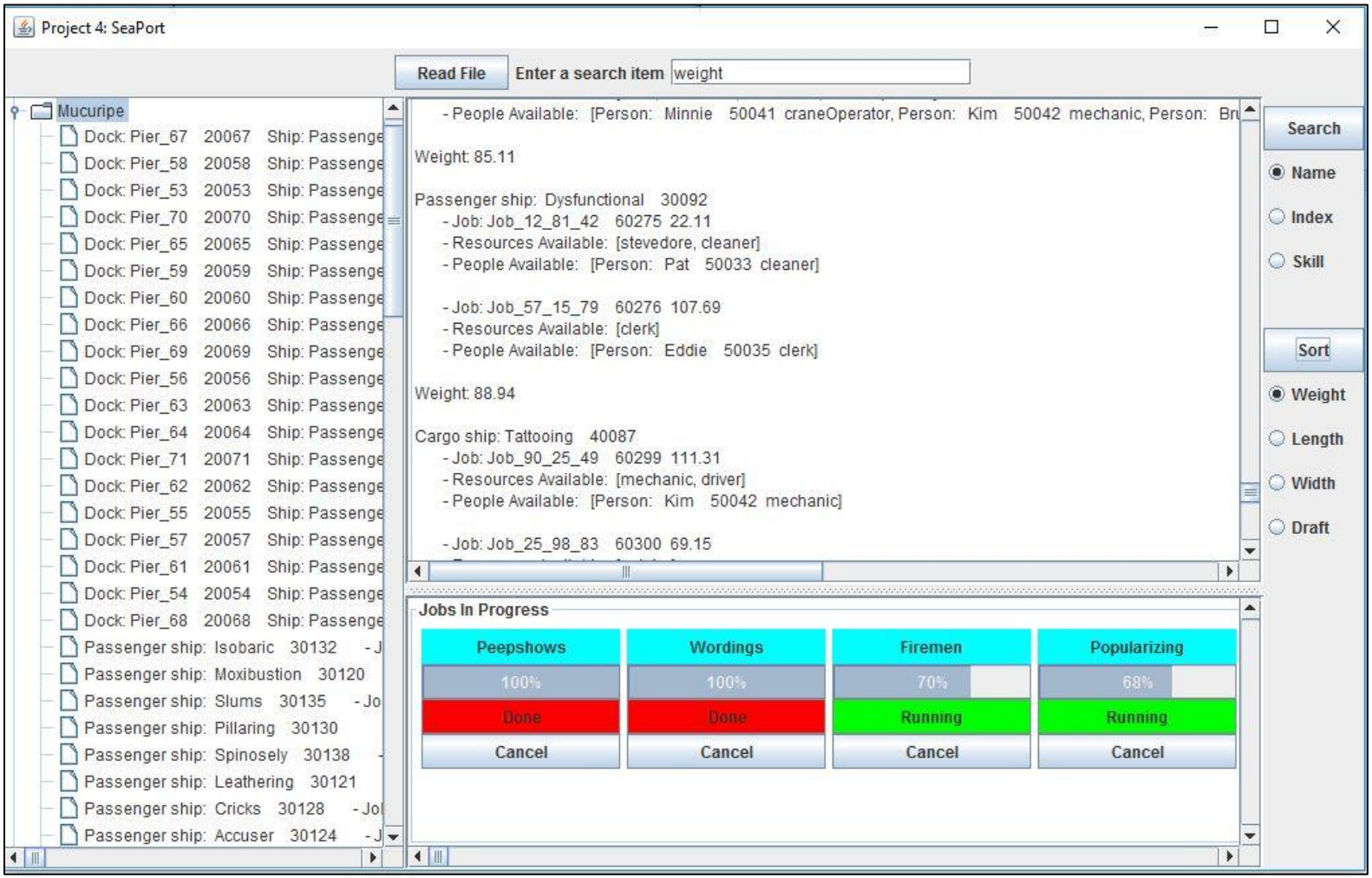


7.) If an invalid search is entered in you will see the following: (Sorry, that is an invalid search. Please try again.)

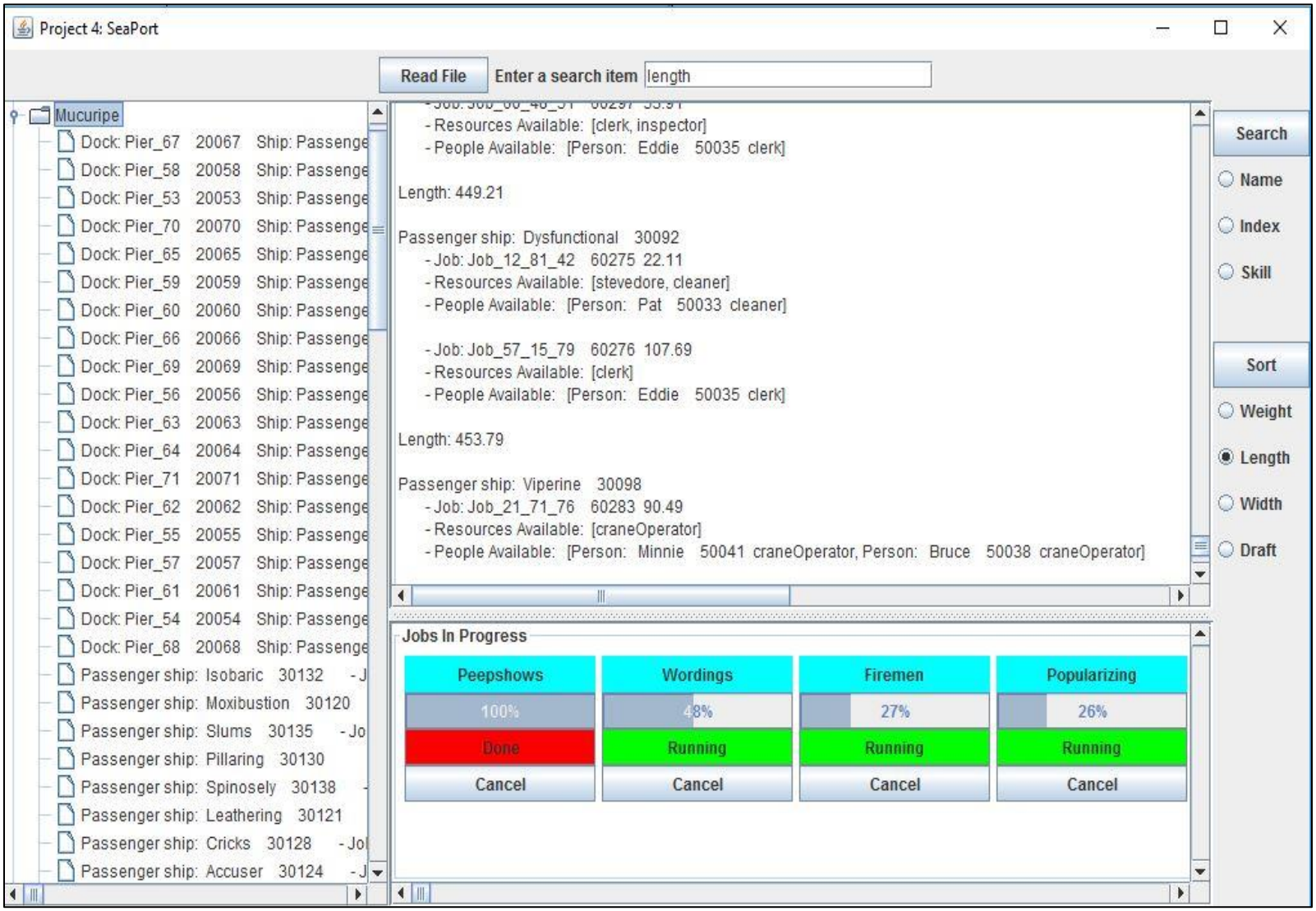


8.) To sort for an item in ascending order you will have three options: Ships Weight, Length, Width, and Draft (it is not case sensitive) **(New step added)**

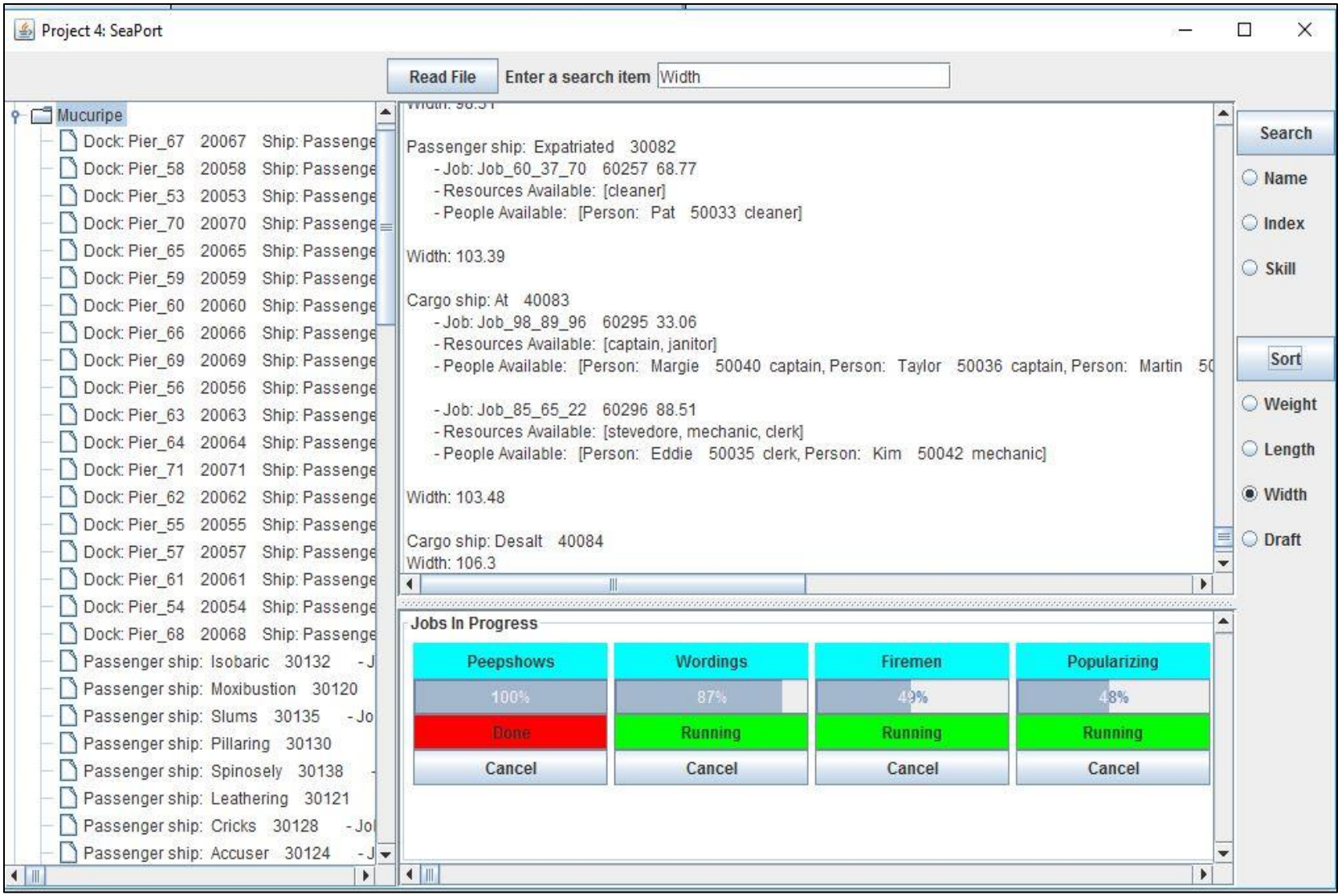
Sort by Ship Weight: Enter in weight



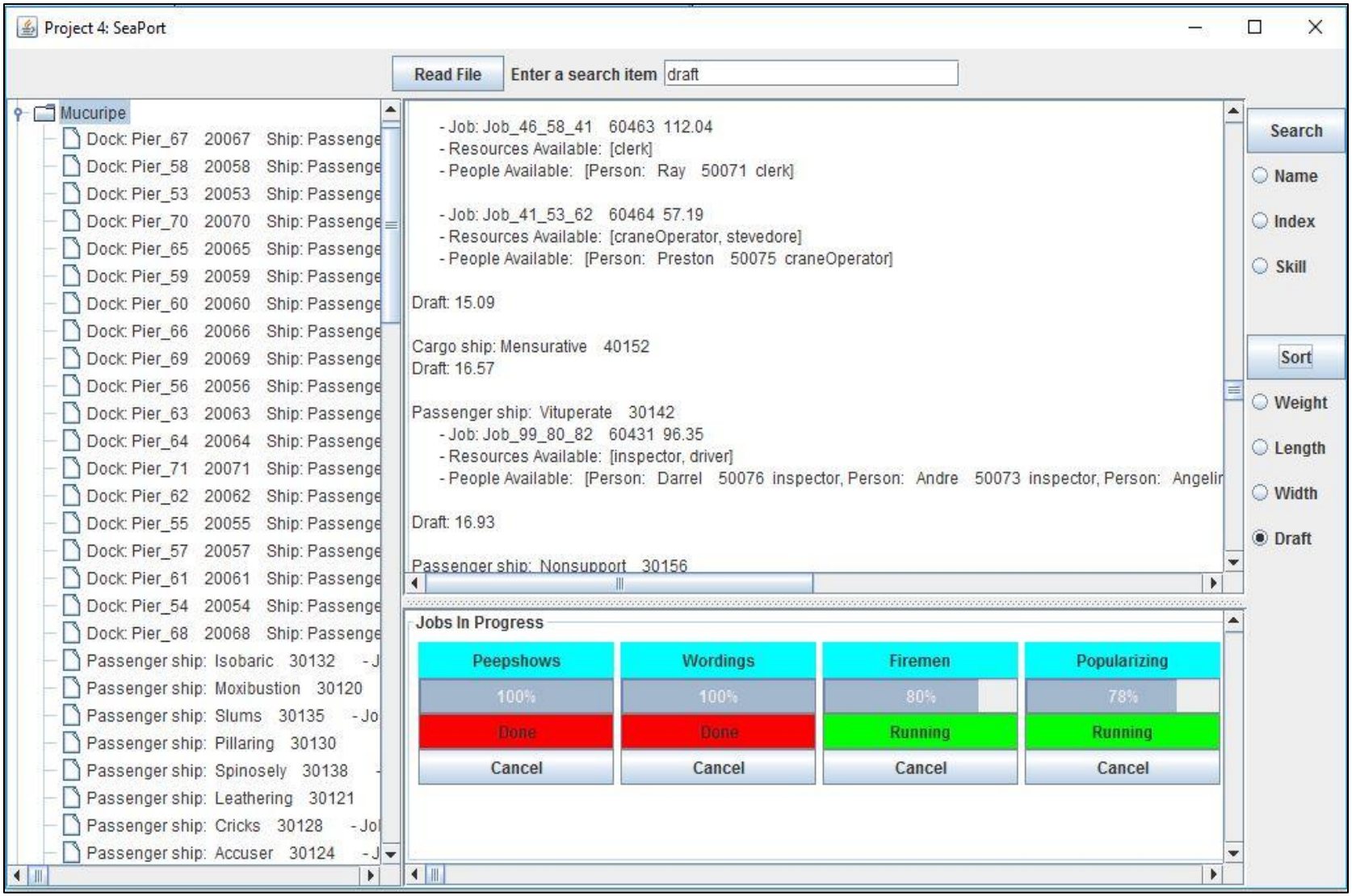
Sort by Ship Length: Enter in length



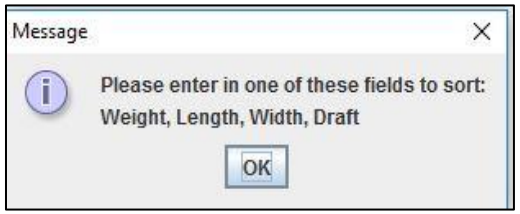
Sort by Ship Width: Enter in width



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.

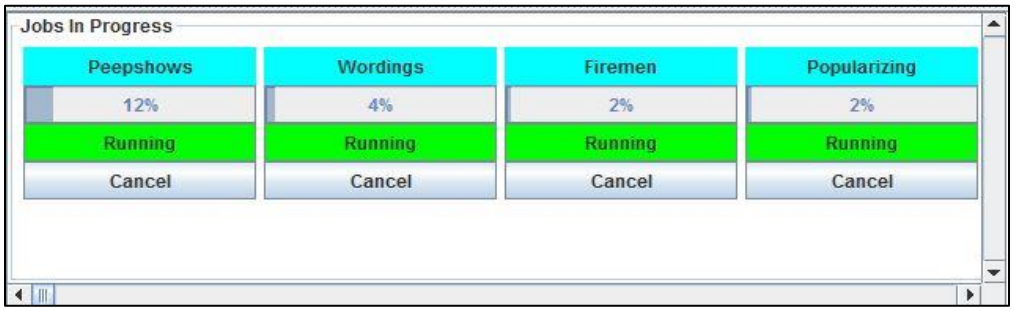
11.) If you followed these steps correctly, you have successfully used my program the right way.

- **Special Features:**
    - User friendly interface with scrollbars implemented on all text areas, as you can see in the screenshots
    - 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 (File used: aSPae.txt)
    - Uses radio buttons to search by name, index, and skill (File used: aSPae.txt)
    - Is not case sensitive for what is being search. (Ex: both John, or john, will return the same result).
    - Displays the results nicely in the text area after user decides what they want to search or sort
    - Displays the available resources for each job in the text area. This tells us whether a job can start.

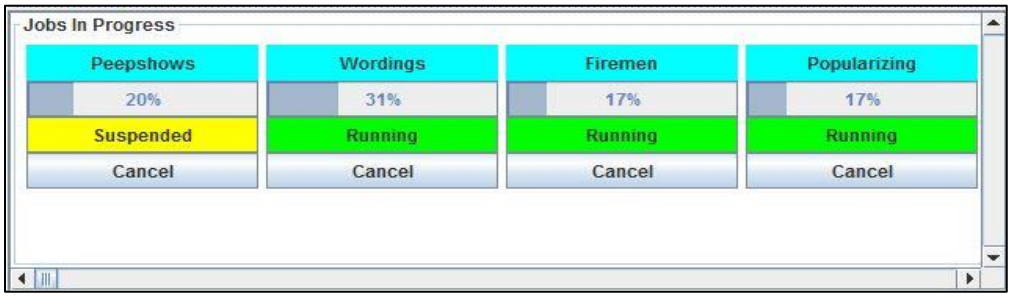
Example: Shows the available resources for Passenger Ship: “Snowcaps.”

```
Dock: Pier_59   20059

  Ship: Passenger ship:  Snowcaps   30126
    - Job: Job_20_59_98   60390  42.01
    - Resources Available:  [engineer]
    - People Available:  [Person:  Moses   50065  engineer]

    - Job: Job_49_39_76   60391  110.88
    - Resources Available:  [mechanic, carpenter, craneOperator]
    - People Available:  [Person:  Lance   50059  craneOperator, Person:  Lewis   50060  mechanic, Person:  Cassandra   50054  carpenter, Person:  Phyllis   50063  mechanic]
```

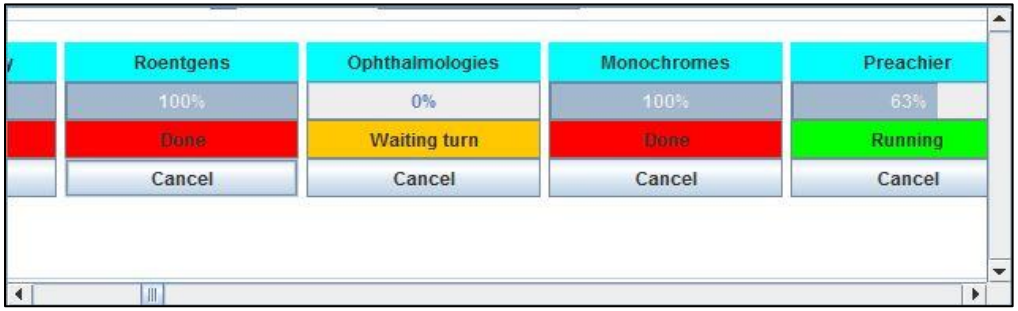
- Uses synchronized and multithreading to detect when jobs start and end.
- Has a Running, Waiting, Suspend, Done, and cancel button that are colored coded (Ideal for users of this program)
- When a Thread is “Running” the resources are available to start that Job. This will display as the following:



- The threads can be “Suspended” individually. For example, if I suspend the ship, “Peepshows”, the following ships will continue completing their jobs. This will display as the following:

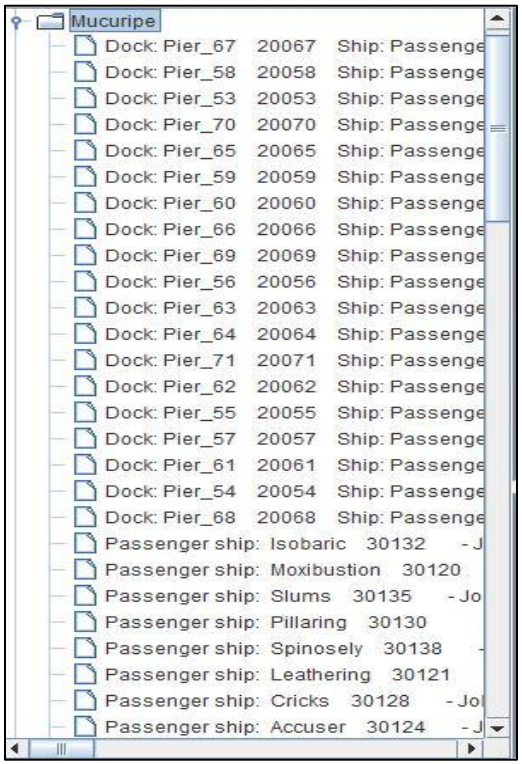


- If a Thread is waiting to “Start” because not all the resources are available yet. This will display as the following:



- When a Thread is “Done,” and all jobs are completed, it will look like the following:

- Nice JTree panel to show all the nodes for each dock, which has a nice scrollPane. Implements Horizontal and vertical scrolling. This displays as the following:



III. **Test Plan:** (Update from Project 3)

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

  Project 4 was an extension off project 3, and is the final project for the main project, "SeaPort Series." The first thing I did before extending on to project 3 was to make sure that I understood what the instructions were asking for. The goal of this assignment was to first display the available resources for each job that were listed in the data file. The second thing was to list each person and their skill at each port, along with supporting assignment to ships and jobs. The third thing was to use the concept of blocking threads from starting until resources become available. The jobs would only start if the ship is at the dock with all the people and their required skills are available. If there are no resources the job cannot be completed; therefore, it will never start. The fourth thing is when all jobs of the ship are done, that ship should leave the dock, and allow for another one to start. Finally, if a job does not have enough skills to start, the job should be cancel, and not progress. I found Oracles documentation, and other sites, and books as the best resources for understanding how to do this.

IV. **Lessons Learned:** (Update from Project 3)

After completing project 4 there are many additional things that were learned from project 1, 2, and 3. Like in project 1, 2, and 3, I realize how important, and difficult it is to work with multiple java classes; as well as implementing threads and synchronization. This project was difficult for me, and I am overall happy with my results, and feel that I met all the requirements for this final project. The GUI is user friendly, and everything works as it should. I understood and successfully implemented all aspects for this assignment. I now understand how threads work, as you will see in my program, I was successfully able to show each job running, suspended, waiting, done, and canceled. You will also now see in my program the available resources for each job on the ship, as well as, the available people. This project took most of the week, and a total of 4-5 hours a day. I feel that this project fulfilled my expectations. All in all, I have learned a lot this semester, and appreciate all of the assistance you provided me with, so that I could be successful in this advanced programming class.