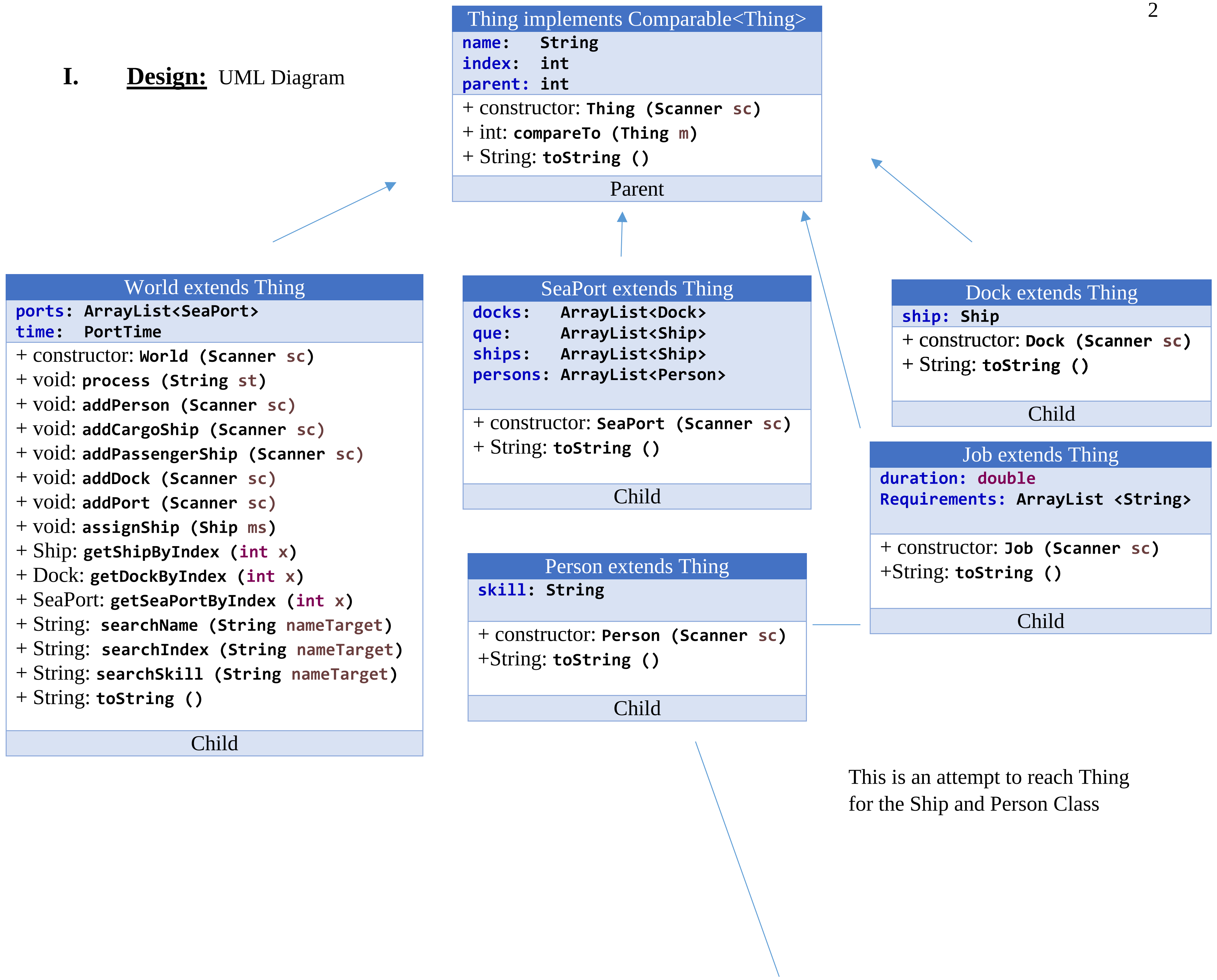
Anthony Borza

Due Date: 11/06/16

Project 1: SeaPorts

CMSC 335 7980

## I. **Design:** UML Diagram



```
Thing implements Comparable<Thing>
name:   String
index:  int
parent: int
+ constructor: Thing (Scanner sc)
+ int: compareTo (Thing m)
+ String: toString ()
Parent
```

```
World extends Thing
ports: ArrayList<SeaPort>
time:  PortTime
+ constructor: World (Scanner sc)
+ void: process (String st)
+ void: addPerson (Scanner sc)
+ void: addCargoShip (Scanner sc)
+ void: addPassengerShip (Scanner sc)
+ void: addDock (Scanner sc)
+ void: addPort (Scanner sc)
+ void: assignShip (Ship ms)
+ Ship: getShipByIndex (int x)
+ Dock: getDockByIndex (int x)
+ SeaPort: getSeaPortByIndex (int x)
+ String:  searchName (String nameTarget)
+ String:  searchIndex (String nameTarget)
+ String: searchSkill (String nameTarget)
+ String: toString ()
Child
```

```
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
```

```
Job extends Thing
duration: double
Requirements: ArrayList <String>
+ constructor: Job (Scanner sc)
+String: toString ()
Child
```

```
Person extends Thing
skill: String
+ constructor: Person (Scanner sc)
+String: toString ()
Child
```

This is an attempt to reach Thing
for the Ship and Person Class

**SeaPortProgram extends JFrame**

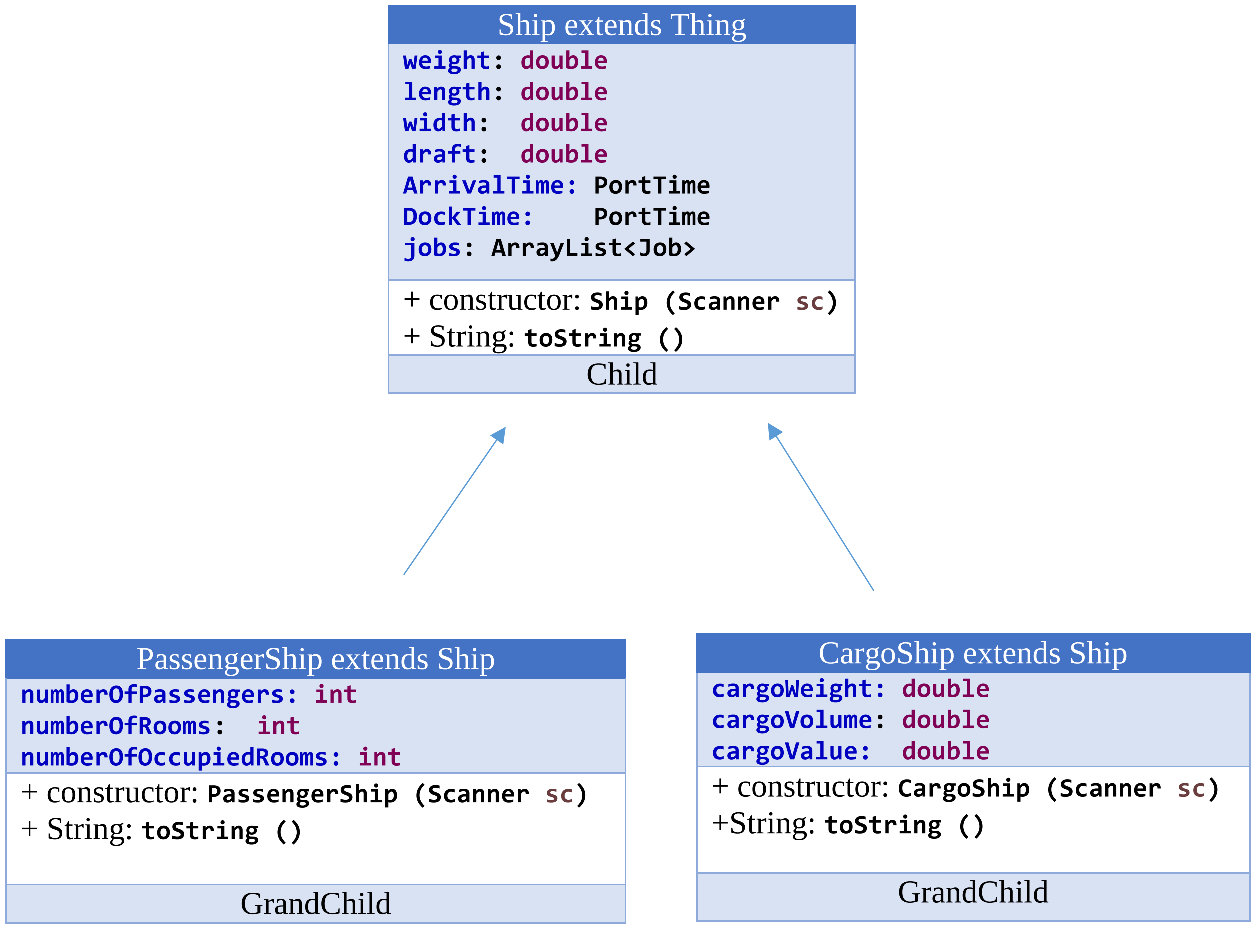
```
-JTextArea textArea;
-JTextField searchTargets;
-JLabel searchName;
-JRadioButton name;
-JRadioButton index;
-JRadioButton skill;
-JButton searchButton;
-JButton readFile;
-JPanel panel1;
```

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

**PortTime.java**

```
time: int
```

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



**Ship extends Thing**

```
weight: double
length: double
width:  double
draft:  double
ArrivalTime: PortTime
DockTime:    PortTime
jobs: ArrayList<Job>
```

```
+ constructor: Ship (Scanner sc)
+ String: toString ()
```

Child

**PassengerShip extends Ship**

```
numberOfPassengers: int
numberOfRooms:  int
numberOfOccupiedRooms: int
```

```
+ constructor: PassengerShip (Scanner sc)
+ String: toString ()
```

GrandChild

**CargoShip extends Ship**

```
cargoWeight: double
cargoVolume: double
cargoValue:  double
```

```
+ constructor: CargoShip (Scanner sc)
+String: toString ()
```

GrandChild

## II. User's Guide:

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

**Steps for using my program:**

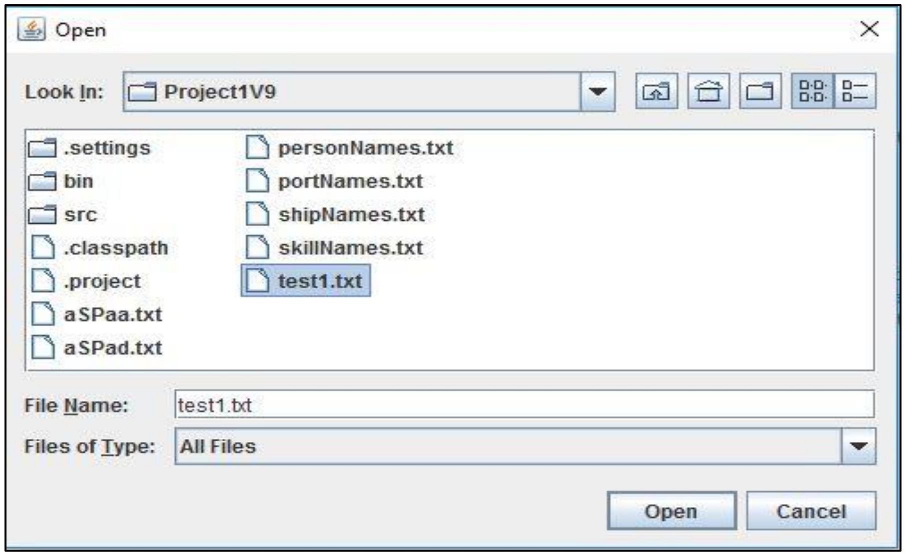
**1.)** Open a IDE of your chose. (Ex: Eclipse will be used in this Demonstration).

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

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

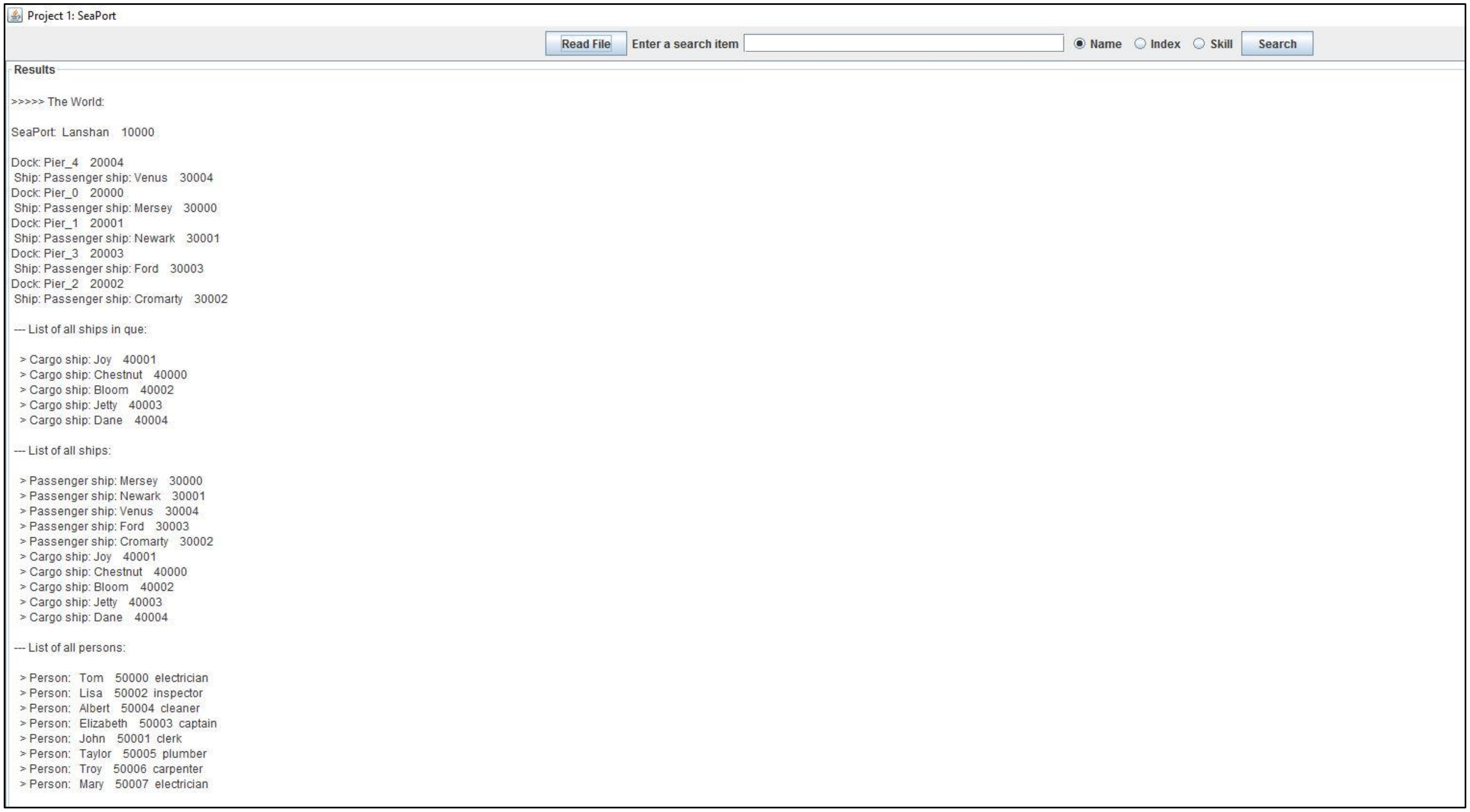
(This can be found at the top of eclipse)

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



File Name: test1.txt
Files of Type: All Files

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

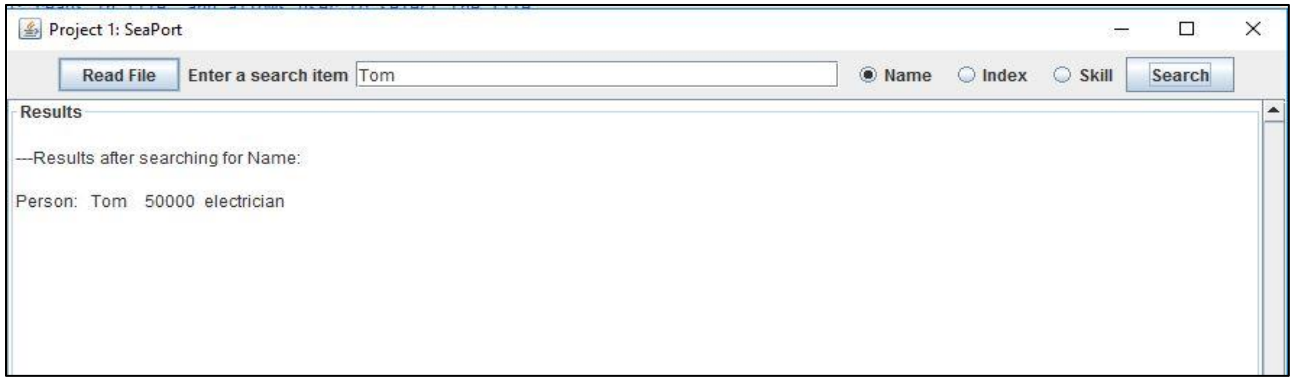


Chosen setting: [✓ Name] Index Skill

```
>>>>> The World:

SeaPort: Lanshan   10000

Dock: Pier_4   20004
 Ship: Passenger ship: Venus   30004
Dock: Pier_0   20000
 Ship: Passenger ship: Mersey   30000
Dock: Pier_1   20001
 Ship: Passenger ship: Newark   30001
Dock: Pier_3   20003
 Ship: Passenger ship: Ford   30003
Dock: Pier_2   20002
 Ship: Passenger ship: Cromarty   30002

 --- List of all ships in que:

  > Cargo ship: Joy   40001
  > Cargo ship: Chestnut   40000
  > Cargo ship: Bloom   40002
  > Cargo ship: Jetty   40003
  > Cargo ship: Dane   40004

 --- List of all ships:

  > Passenger ship: Mersey   30000
  > Passenger ship: Newark   30001
  > Passenger ship: Venus   30004
  > Passenger ship: Ford   30003
  > Passenger ship: Cromarty   30002
  > Cargo ship: Joy   40001
  > Cargo ship: Chestnut   40000
  > Cargo ship: Bloom   40002
  > Cargo ship: Jetty   40003
  > Cargo ship: Dane   40004

 --- List of all persons:

  > Person:  Tom   50000  electrician
  > Person:  Lisa   50002  inspector
  > Person:  Albert   50004  cleaner
  > Person:  Elizabeth   50003  captain
  > Person:  John   50001  clerk
  > Person:  Taylor   50005  plumber
  > Person:  Troy   50006  carpenter
  > Person:  Mary   50007  electrician
```

**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

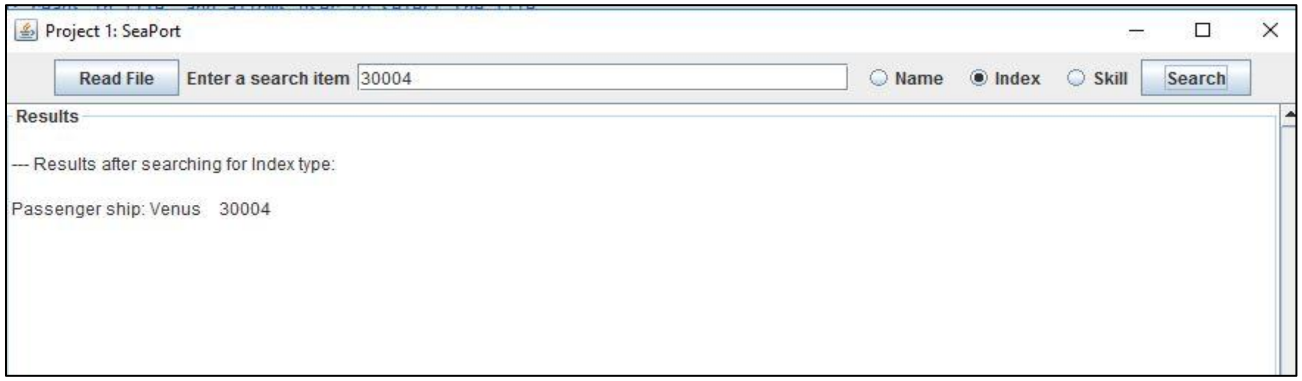


Enter a search item: Tom

Name [✓] Index Skill

```
---Results after searching for Name:

Person:  Tom   50000  electrician
```

**Index Search Example:** Search for 30004

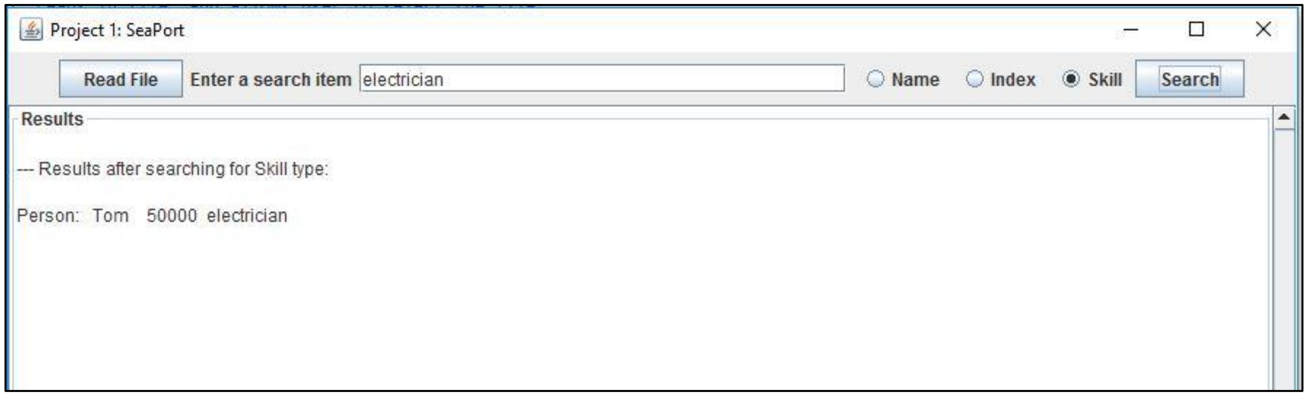


Enter a search item: 30004

Name Index [✓] Skill

```
--- Results after searching for Index type:

Passenger ship: Venus   30004
```

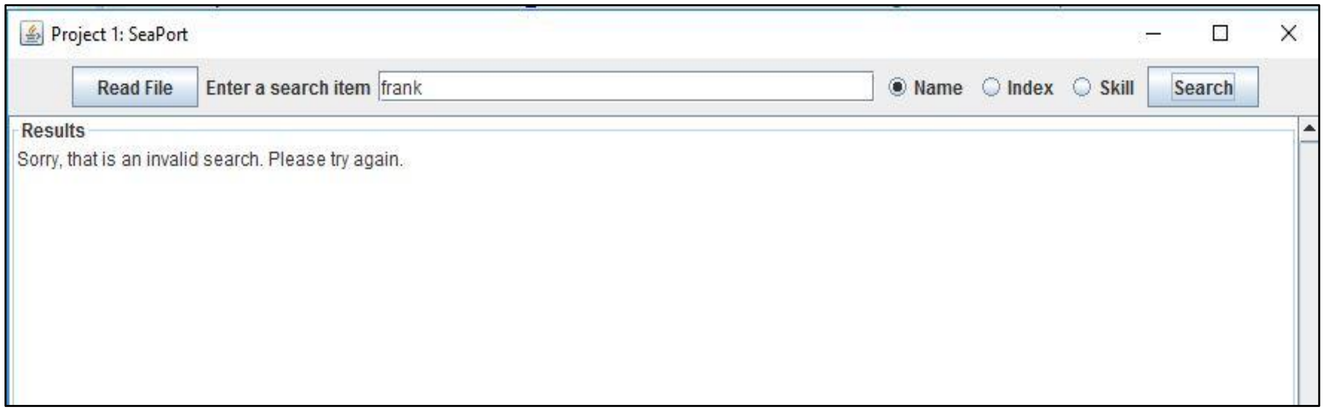
**Skill Search Example:** Search for electrician



Search item: electrician; selected: Name ○ Index ○ Skill ●

```
--- Results after searching for Skill type:

Person:  Tom   50000  electrician
```

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



Search item: frank; selected: Name ● Index ○ Skill ○

```
Sorry, that is an invalid search. Please try again.
```

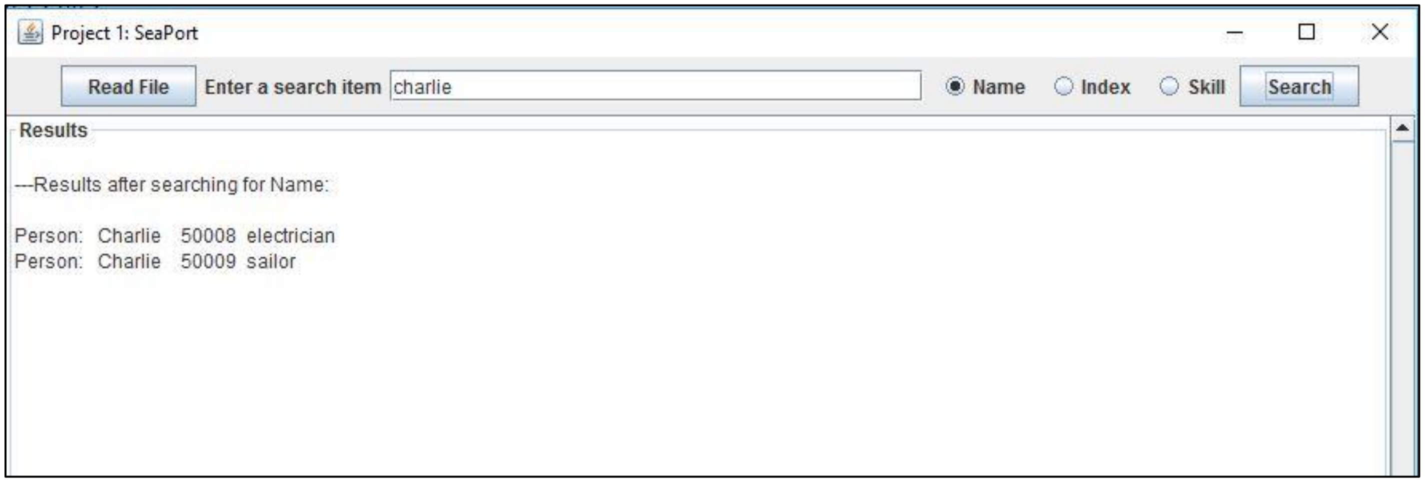
**8.)** To open and search another file, select the Read File Button. This will display the same dialog box in step 4.

**9.)** 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
  - 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, and skill

- **Screen Shots:**
  - Allows duplicates for name, index, and skill (reading in test2.txt)

    **Name:** search for charlie

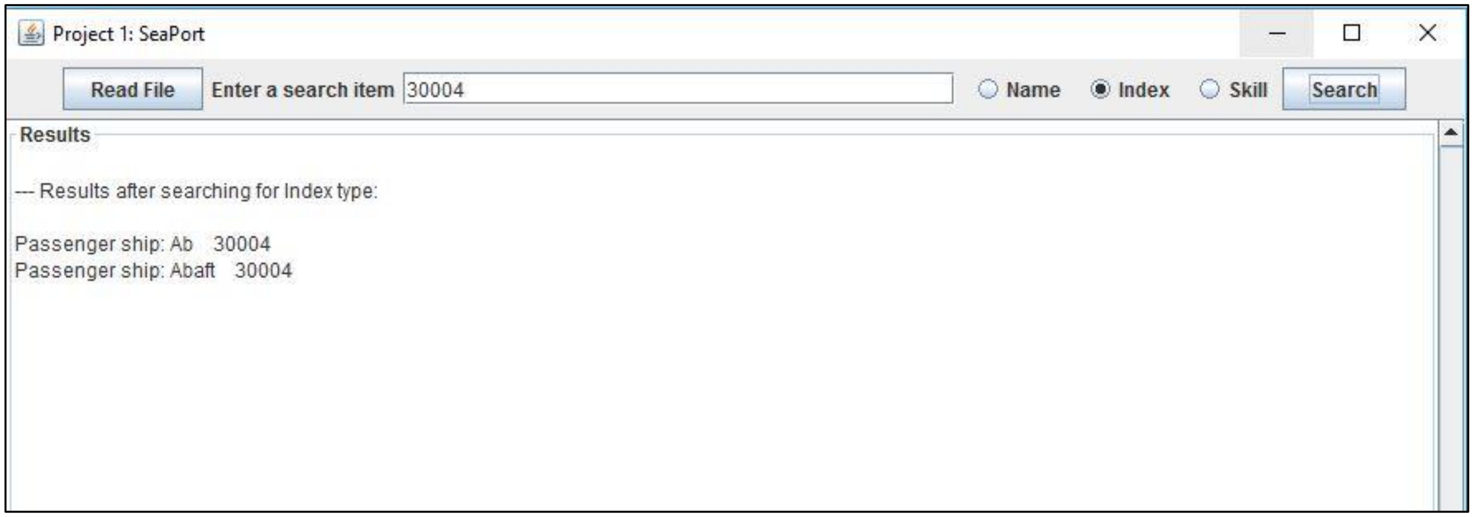


Enter a search item: charlie

(● Name) ○ Index ○ Skill

```
---Results after searching for Name:

Person:  Charlie   50008  electrician
Person:  Charlie   50009  sailor
```

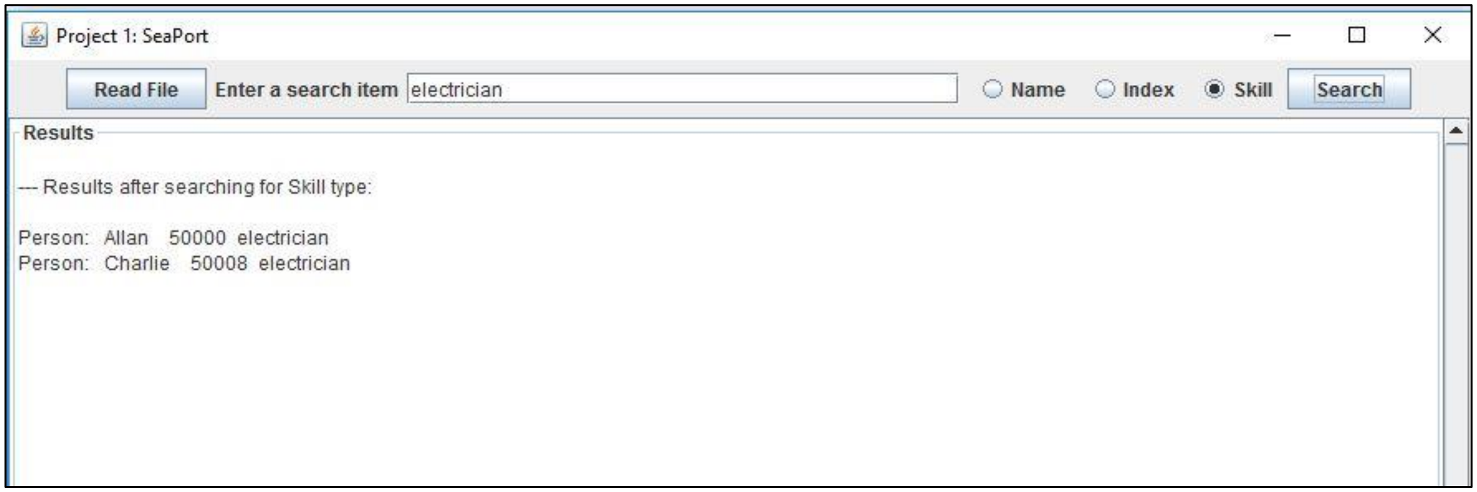
**Index:** search for 30004



Enter a search item 30004 ○ Name [● Index] ○ Skill

```
--- Results after searching for Index type:

Passenger ship: Ab   30004
Passenger ship: Abaft   30004
```

**Skill:** search for electrician



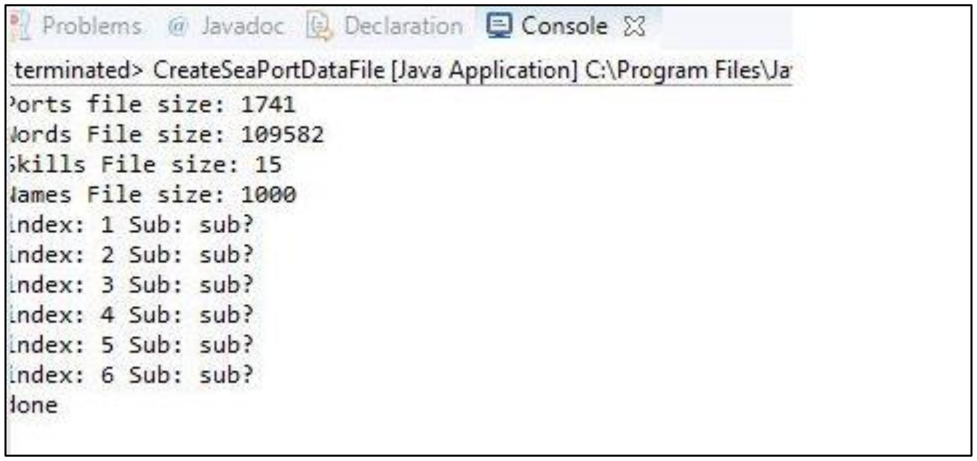
Enter a search item electrician ○ Name ○ Index [● Skill]

```
--- Results after searching for Skill type:

Person:  Allan   50000  electrician
Person:  Charlie   50008  electrician
```

## III. Test Plan:

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

  I expect the code to function as talked about in the project instructions. It seems that a lot of the code is hardcoded to show its functionality. Also, all the classes are within one java file, and as a programmer the first thing I would do is separate the inner classes into their own before even beginning to write any code. Before coding anything, and just running the program as is, the output is this:



```
terminated> CreateSeaPortDataFile [Java Application] C:\Program Files\Ja
'orts file size: 1741
lords File size: 109582
;kills File size: 15
lames File size: 1000
.ndex: 1 Sub: sub?
.ndex: 2 Sub: sub?
.ndex: 3 Sub: sub?
.ndex: 4 Sub: sub?
.ndex: 5 Sub: sub?
.ndex: 6 Sub: sub?
lone
```

The information provided above tells the programmer the file sizes of each txt file.

For example, the files provided are:

- portNames.txt: contains 1741 ports
- shipNames.txt: contains 109582 words
- skillNames.txt: contains 15 skills
- personNames.txt: contains 1000 names

Overall, this is what I expected the project to do. It was designed to get us started, and to think of the ways we could implement what was already given to us by adding new code, deleting unnecessary code, and developing algorithms. By doing this, it would allow us to meet all the requirements for this project.

## IV. <u>Lessons Learned:</u>

After completing this project there are many things that were learned. I realize how important it is to work with multiple classes using inheritance and polymorphism. This project taught me how to used Array List effectively by adding elements, assigning elements, and searching for elements when reading in a file. Like in previous programming classes, understanding what the problem is asking, and how to implement a solution to solve the problem only comes through trial and error. This project involved a lot of trial and error. I like it when I cannot get something to work at first, like in this project because when I do, it is so fulfilling and exciting. The hardest part of this project was developing and algorithm that would search through a file, by name, index, and skill type. Once I figured out how to do that developing the GUI, and calling World class, and search methods was easy. All in all, to complete project 1, I spent a total of 35 hours on it. I factor in about 5 hours a day, or more.