

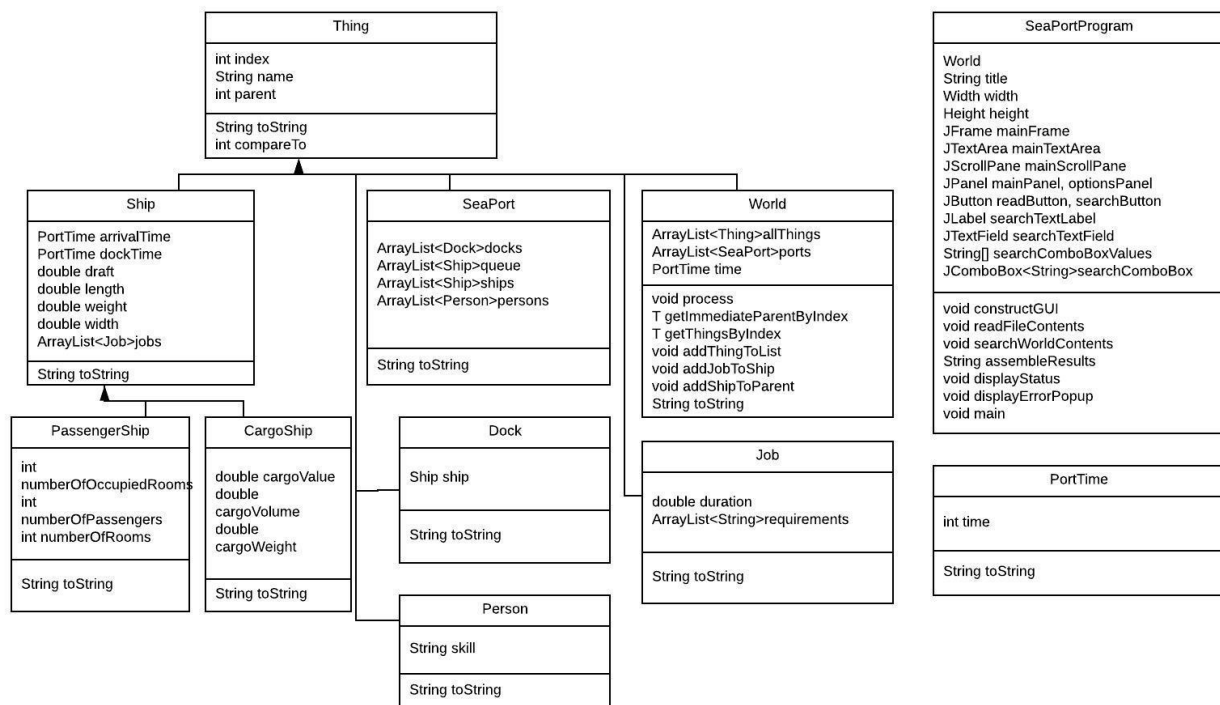
Project 1 Documentation

Shafro Batyrov

9/2/2018

UMUC CMSC 335

Part I: Design

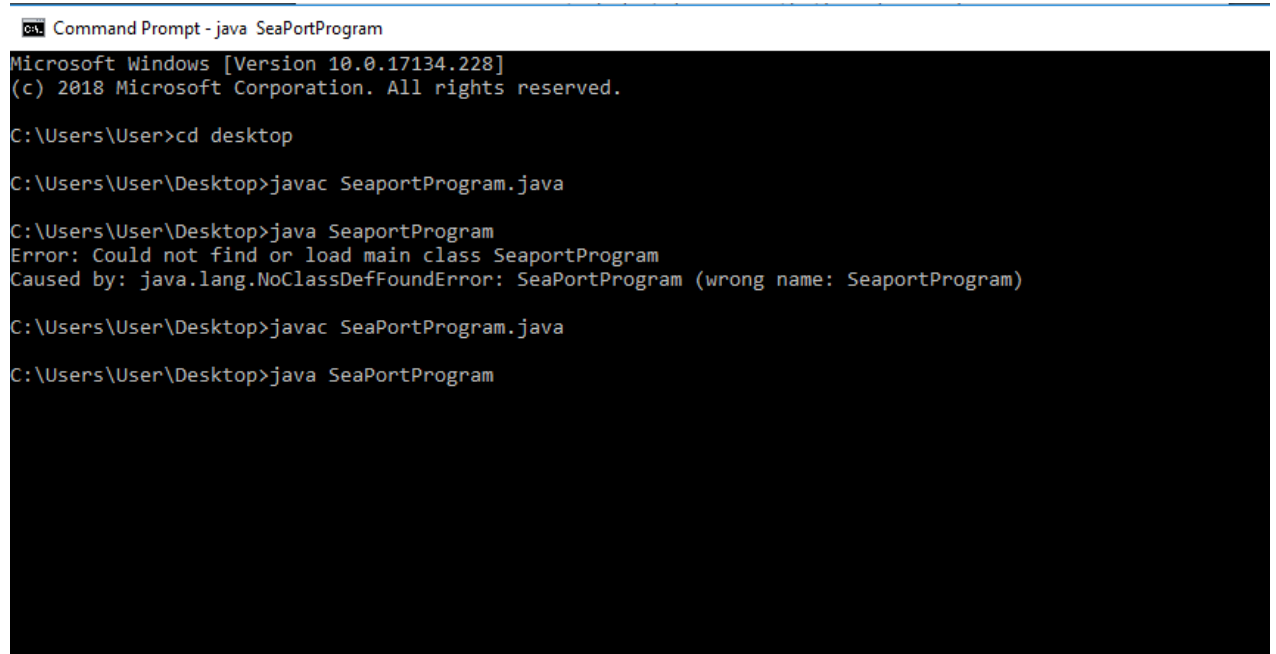


This is the UML diagram for Project1. The only unused class was `PortTime` which was simply included due to its mention on the first page—the author assumes it will be important in the upcoming projects. The only private class field not on the rubric’s listing on fields/methods is `allThings`, an `ArrayList` of `Things` contained within new `World` objects and used to store a quick and convenient listing of all `World` extant objects. This was added to simplify the retrieval process of objects during user searches and provide a comprehensive listing of all objects of all types as they exist in the file.

Part II: User's Guide

Users unfamiliar with the process of running Java programs may have some difficulty running `SeaPortProgram` Project 1 without some prior experience and a set of qualifications.

To run this program, a user would have to save all the source code in one place and direct the Windows command prompt to that destination. They would then enter “`javac SeaPortProgram.java`” and at the second prompt enter “`SeaPortProgram.java`”. The java files are case sensitive so the user has to be sure to input it correctly. The GUI would then pop up.



```
Command Prompt - java SeaPortProgram
Microsoft Windows [Version 10.0.17134.228]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>cd desktop

C:\Users\User\Desktop>javac SeaportProgram.java

C:\Users\User\Desktop>java SeaportProgram
Error: Could not find or load main class SeaportProgram
Caused by: java.lang.NoClassDefFoundError: SeaPortProgram (wrong name: SeaportProgram)

C:\Users\User\Desktop>javac SeaPortProgram.java

C:\Users\User\Desktop>java SeaPortProgram
```

To prevent invalid input, the program will not open non-text files, so users must make an effort to open a file that conforms to the organizational layout of the sample data files. It should be noted that the use of search bar necessitates the inclusion of proper input. User input is case and spelling sensitive, so the program will not find results that are improperly formatted as they appear in the text file.

Part III: Test Plan

Starting out, the author sat down and sought to visualize the underlying processes at work in the text file and the relationships between objects of different types. Laying out the design in a diagram of sorts, the author was more easily able to visualize the connection between types of objects and the relationship each of their classes would possess with others of similar types.

While the process of passing objects to their proper containers during the file reading part of the assignment was straightforward enough, the author had some initial difficulty in deciding how best to search through objects. In the end, the author used a number of generic methods aimed at iterating through `ArrayLists` of `Thing` objects in nested loops. The program makes use of the predefined relationship class structure at play as well as a listing of all encountered `Things`.

Making use of the rubric's suggested methods, the author initially had several different but near-identical methods aimed at either adding objects to their proper classes or retrieving an object by an index. Though this process worked and produced satisfactory results as expected, the sheer amount of needless code repetition made the author consider the possibility of using generics to accept `Things` of all types. As such, the half-dozen or so near-identical methods were all replaced by three generic methods that retained the same purpose as those in the rubric but were slightly more optimized and cleaner in form.

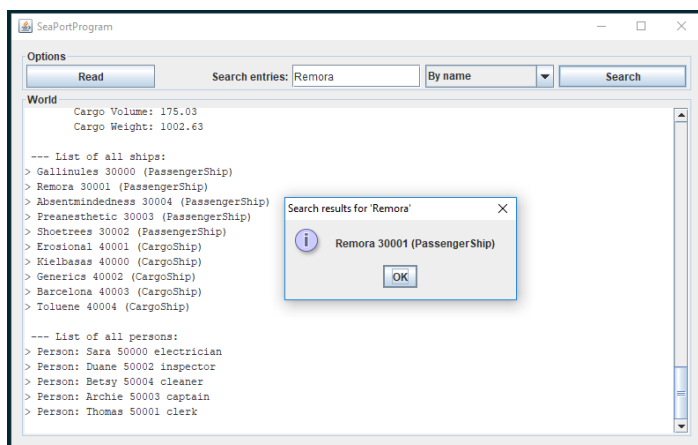
Part III b: Test Cases

The test cases included herein were retrieved from tests run on two data files, namely `aSPaa.txt` from the projects.zip download in the Contents tab, and from `asPad.txt`,

assembled by the author using the `CreateSeaPortDataFile.java` program. The first four tests used the first file, and the last test employed the second file in question.

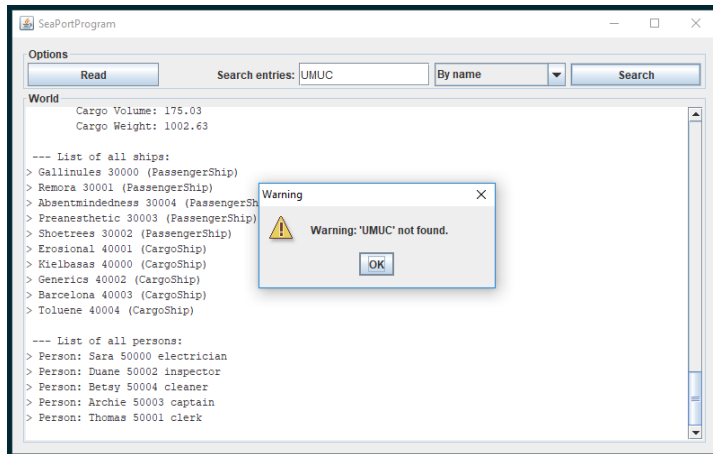
Input	Expected Results	Results
Remora, name	Remora 30001 (PassengerShip)	Remora 30001 (PassengerShip)
UMUC, name	Warning: UMUC not found.	Warning: 'UMUC' not found.
50004, index	Betsy 50004 (Person)	Betsy 50004 (Person)
electrician, skill	Sara (id #50000)	Sara (id #50000)
<no input>	Error: No world initialized. Please try again.	Error: No world initialized. Please try again.
mate, skill	Mark (id #50005) Joe (id #50023) Gabriel (id #50070) Gayle (id #50007) Mack (id #50055) Brendan (id #50068) Patrick (id #50037)	Mark (id #50005) Joe (id #50023) Gabriel (id #50070) Gayle (id #50007) Mack (id #50055) Brendan (id #50068) Patrick (id #50037)

Test Case 1



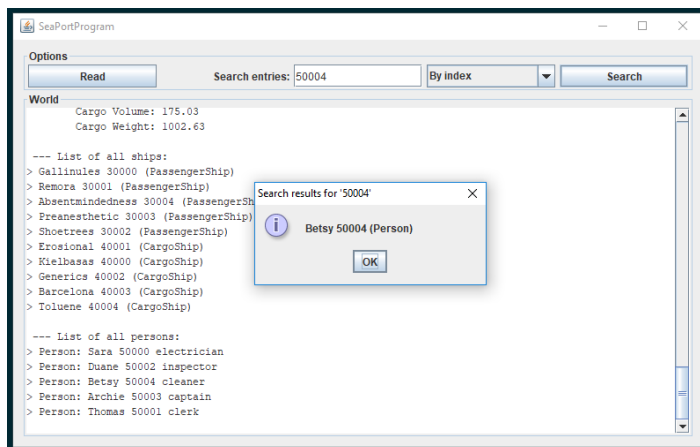
The first test case concerns the assembly of a legitimate file and proper search input, namely “Remora.” The program displays a popup window with information pertaining to this ship as expected.

Test Case 2



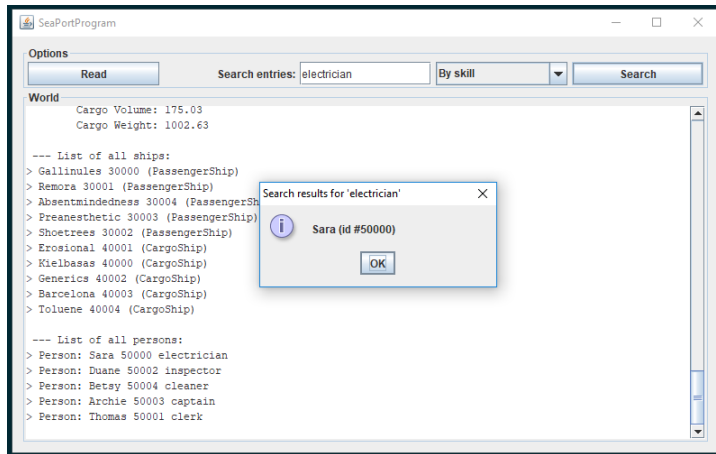
The second test handles cases wherein a proper file has been read and assembled, but one where the user's chosen search term does not exist in the database. As a result, the program displays a warning popup to notify the user of this fact.

Test Case 3



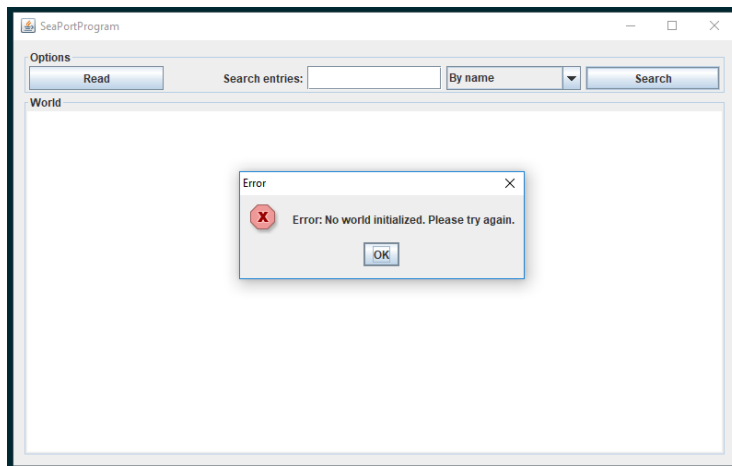
The third case tests the program's response to searches by index rather than by name, as was the case with the previous searches. In this case, the program displays the chosen Thing with the selected index as expected.

Test Case 4

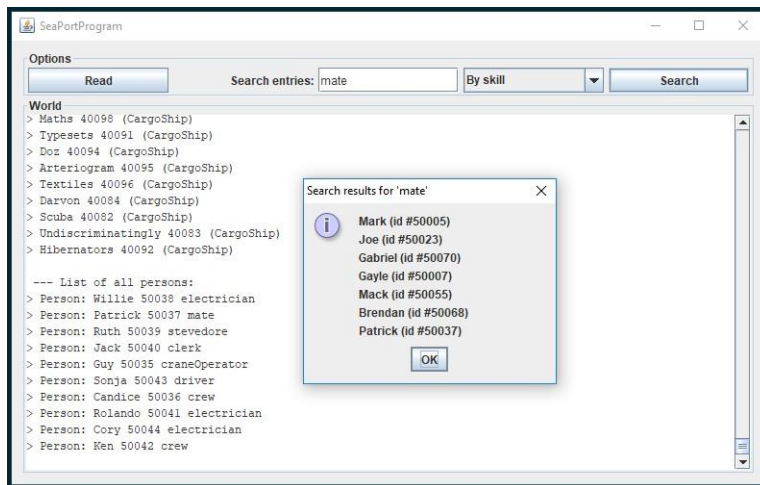


This test case handles retrieving all instances of `Persons` with the inputted skill/profession. In the case of `aSPaa.txt`, there is only one `Person` with the skill of electrician, so that user is displayed in the popup modal.

Test Case 5



The fifth test case handles cases wherein the user may have accidentally failed to select a file and build a world before pressing the “Search” button. Since there is no data, the program displays an error window notifying the user of this fact.

Test Case 6

As an illustration of the program's ability to display multiple instances of shared input data, the text file `aSPad.txt` was used to display a list of `Person` instances with the skill of `mate`. As expected the popup window displays each in a list.

Part IV: Comments

As far as improvements over existing functionality go, the author may wish to revamp and improve upon the structure of the generic methods contained within the body of `World`. As they were integrated late in the game and expanded as the author learned more about generics and reflection, they appear somewhat messy in comparison to some of the starting methods that did not change as the project came together. Some more research into generics and reflection is required by the author to more naturally employ such powerful tools in future.

Part V: Lessons Learned

The author learned a number of things related to the implementation of her design plan during the course of this assignment. Most significant was probably the use of reflection to pass

the name of a method to be used by a generic method to properly invoke the desired

`ArrayList` getter. The ability to pass method names to be invoked within a single generic method decreased the amount of copy/pasted code within the body of the `World` class and allowed the author to consolidate such near-identical functions into a single set of methods.

Similarly related to this was the use of generics in the program. Though the author has used generic methods and classes in her programs in the past, she has never really fully grasped the power of generics in Java prior to the assignment. Though she still struggles with some of the aspects of generics and casting (namely visualization of the process actually happening), she was able to successfully reduce the number of near-identical methods through the application of such practices.