Seaport Project: Part 1-2

Nicholas Mills

CMSC 335: Object Oriented and Concurrent Programming

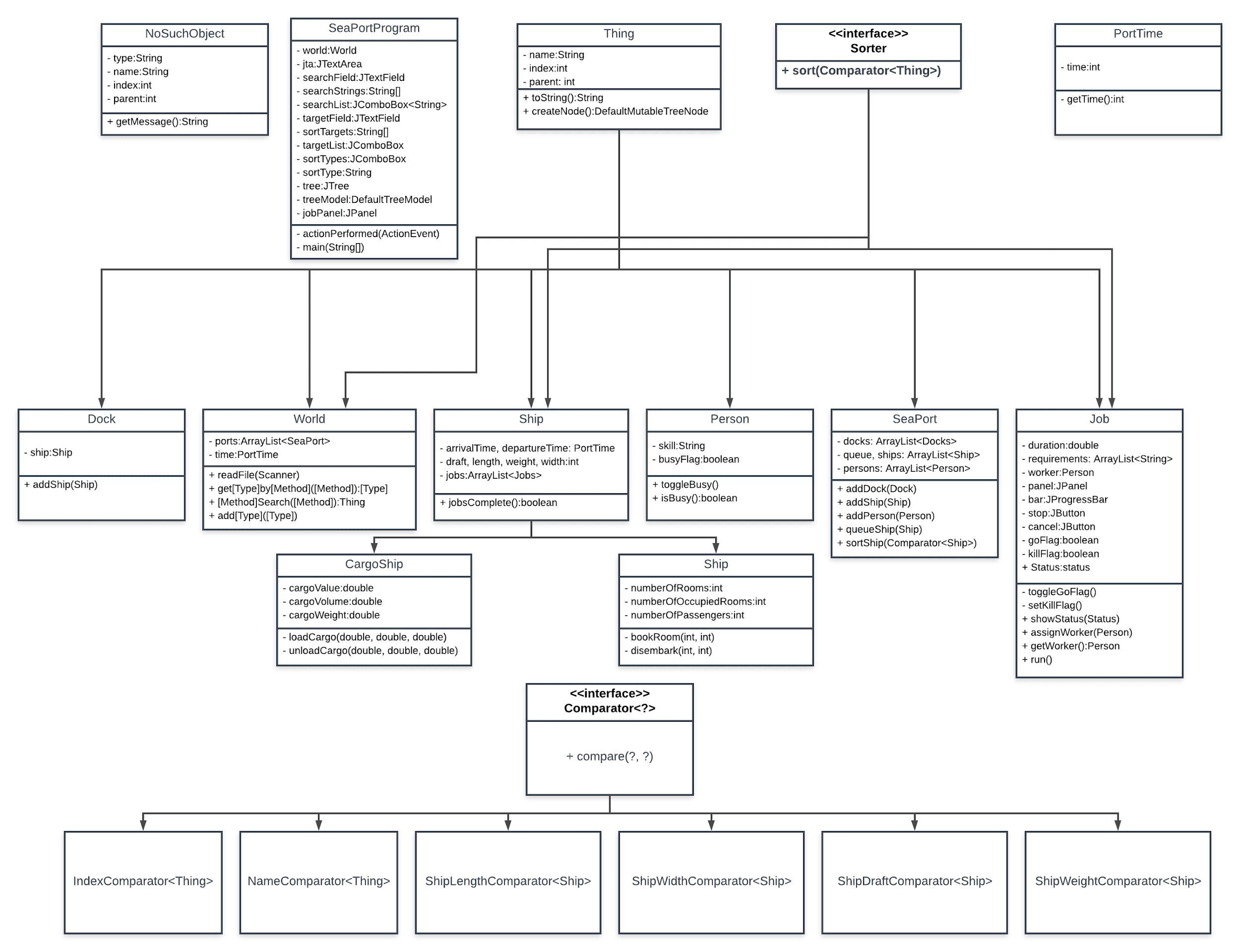
University of Maryland University College

Mihaela Dinsoreanu

11/3/2018

Design

**UML Diagram**

****

**Class Description**

Part #1

This portion of the SeaPort project led me to mainly focus around the SeaPortProgram and World classes due to the heavy focus on building the World and GUI, which is contained in SeaPortProgram. I also created a custom exception called NoSuchObject to handle inconsistencies within source files and unsuccessful searches.

**SeaPortProgram.** In this class, I allowed the user to select the file of their choice using a JFileChooser that starts off in the project directory, as requested by the project, as well as displaying the output of the file in a JTextArea in the center of the window. I also included error handling for if the source file wasn’t found and also detection for if there are some problems with the source file. I then created a subpanel for the search bars that used a 3x3 grid, each row containing a label for the type of search that is being done, a text field for inputting the search terms, and a button to search the world for those terms. The results of these searches are printed in the JTextArea in the middle. If there is no result, a dialog box will appear informing the user. I also included error handling for incorrect formats for the numeric fields. Below the search bars, there is a reset button to return the JTextArea to its original state of displaying the entire output of the source file.

**World.** This class contains logic for building and searching the world. The constructor takes a scanner of the source file and passes information from it one line at a time to the process method. This method then determines the correct type of Thing and attaches it to the appropriate parent Thing. Search methods are also included for each of the types of Thing, as well as a nonspecific search method which will check for all types of Thing that match the search term.

**NoSuchObject.** This exception is thrown whenever a search method comes up with nothing. It captures the type of search occurring and generates an appropriate error message through the getMessage method.

Part #2

In order to increase the quality of the previous section of the project, I replaced the fields in the GUI for JComboBoxes to make selection of sort and search options a bit more compact. I initially was thinking about radio buttons but there proved to be too many options for that to be a feasible solution. For this portion of the project, there was a heavy focus on using HashMaps in order to more efficiently construct the World and the use of Comparators in order to sort the different Lists in the World.

**Sorting.** I created a different comparator for each type of sort that the user is able to sort by, including the index, name, and various dimensions of ships. I then created a Sorter interface which is implemented by all classes which have Lists within them. This interface contains a single method, sort which takes a Thing Comparator. This method sorts the appropriate lists in the appropriate order based on the passed Comparator. I also created a second method within SeaPort specifically called sortShip which can take Ship Comparators for sorting based on Ship specific fields. I then created a number of dropdowns in the GUI for selecting the desired type of target and desired type of sort. I also created a field for the name of the target as well as a button to initiate the sort. The types of sorts available in the dropdown changes dynamically based on the type of target selected, since not all sorts work with all types of Thing. The sorted results for the target then display in the text field below.

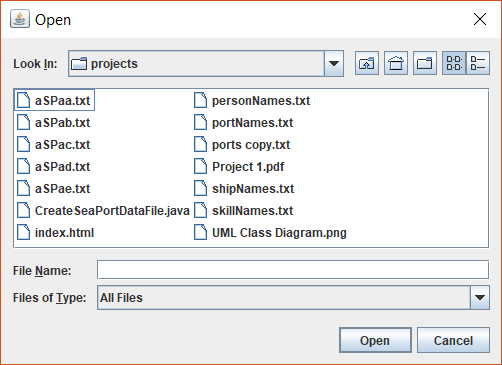
**HashMaps.** In order to effectively use HashMaps to increase the efficiency of the building of the world, I created separate HashMaps for each type of object within the readFile method. I then used these HashMaps in order to add the created objects to the appropriate lists within the other Things. This allowed for the circumvention of having to search for each index iteratively. Since the specification required the HashMaps to be local to the readFile method, I did not change the indexSearch method to search by index via HashMaps and it still currently uses the iterative search technique.

Part #3

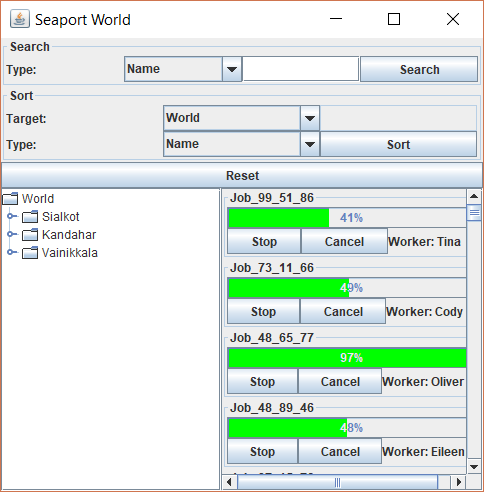
**JTree.** I rearranged the UI using GridBagLayout and a second JScrollPane with a BoxLayout for the job progress bars. I removed search by index since it wasn’t immediately relevant anymore and I changed search by parent so that if you type in the name of a Thing, it will refresh the JTree with only the World root and parent subnodes of that Thing. I eliminated index and parent numbers in the tree since the existence of the tree made those somewhat unhelpful. Sorts work as before, except implemented on the JTree. Lastly, after each of these actions, the tree is reloaded.

**Job Threading.** I assigned Threads to each of the Jobs. I mostly used the code from the Cave example, but I made a few tweaks. Instead of changing the color of the buttons, I changed the color of and text over the JProgressBars, since I thought that looked a little more natural. I also changed the color of complete from red to blue since that also seemed more intuitive. I also added a cancelled status which disabled the buttons and turned the color of the bar red in order to better show that the thread was cancelled. Since I still needed a method of assigning workers, I made the SeaPorts runnable as well. The SeaPorts assign workers to the different jobs and assign new ships to the docks when they are finished with their jobs. I added a jobsComplete method to Ship to assist with that.

**User’s Guide**

When SeaPortProgram is run, a dialog box will appear. Select a source file from the list contained in the ./src/projects subfolder. Valid files start with an a and are .txt files. 

You can also make your own source files using CreateSeaPortDataFile.java. Press open. This will display the following window:



A tree output of the world can be found in the text box in the lower half of the window. You can use the collapsable items and scroll bar to look through the data. From here, you can search the world by name or parent. Type what you wish to search into the search field and select the desired search type from the dropdown. The results will appear in the text box, replacing the existing information.

To sort the information, select the desired target type from the dropdown. If the type is not the whole World, type the name of the desired target into the text box. From here, select the appropriate type of sort desired. Lastly press the sort button. The sorted data will appear in the text box below. Either way, the reset button will display the information for the entire World.

One can see the Jobs of all the ships off to the right hand side. This can be navigated with the scroll wheel. One can pause a task by hitting Stop. One can continue said task by hitting Resume, which Stop will transform into. The cancel button irrevocably cancels the task, so be sure only to press that if you really want to cancel the job.

Test Cases

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Purpose: | Input: | Expected Output: | Actual Output: | P/F |
| Base Test | aSPaa.txt | Window with formatted output |  | P |
| Index Search | aSPaa.txt  20004 | Pier\_4 toString() |  | P |
| Name Search | aSPaa.txt  Shoetrees | Shoetrees toString() |  | P |
| Parent Search | aSPaa.txt  10000 | Children of Lanshan toString() |  | P |
| Reset | aSPaa.txt | Same as Test Case #1 |  | P |
| Bigger File | aSPab.txt | Window with formatted output |  | P |
| NoSuchObject index | aSPab.txt  10 | The Object 10 was not found. |  | P |
| NoSuchObject name | aSPab.txt  blargh | The Object with the name of blargh was not found. |  | P |
| NoSuchObject parent | aSPab.txt  10 | The Object with the parent of 10 was not found. |  | P |
| Sort World by Name | aSPab.txt  World  Name | World toString() with all items sorted by name within their lists |  | P |
| Sort World by Index | aSPab.txt  World  Index | World toString() with all items sorted by index within their lists |  | P |
| Sort Port by Name | aSPab.txt  Port  Wuchun  Name | Wuchun toString() with all items sorted by name within their lists. |  | P |
| Sea Port by Index | aSPab.txt  Port  Wuchun  Index | Wuchun toString() with all items sorted by index within their lists. |  | P |
| Sea Port by Length | aSPab.txt  Port  Wuchun  Length | Wuchun toString() with Ships in Queue sorted by length. |  | P |
| Sea Port by Width | aSPab.txt  Port  Wuchun  Width | Wuchun toString() with Ships in Queue sorted by width. |  | P |
| Sea Port by Draft | aSPab.txt  Port  Wuchun  Draft | Wuchun toString() with Ships in Queue sorted by draft. |  | P |
| Sea Port by Weight | aSPab.txt  Port  Wuchun  Weight | Wuchun toString() with Ships in Queue sorted by weight. |  | P |
| Invalid Target | aSPab.txt  Port  name | Target not found. |  | P |
| Normal Job Progression | aSPad.txt | Allocates workers to docked ships and rotates ships as their jobs complete |  | P |
| Stop and Resume | aSPad.txt  Stop  Resume | Pauses bar, turns it yellow  Unpauses bar, turns it back |  | P |
| Cancel | aSPad.txt  Cancel | Terminates job, turns bar red, adds CANCELLED to bar |  | P |

Comments

**Part #1**

I realized after I had finished that the search options would be better suited for radio buttons. I will change that in the next iteration of the project. Additionally, I will have to rework the NoSuchObject exception message generation if many more search options are added. There may also be a way to condense the code for the search methods but that will require more analysis.

**Part #2**

Removing the index was not required by the prompt, but probably makes sense as an eventuality so that would be something to look into in the future. Modifying the Ship toString so that it includes information about the dimensions of the ship would also be helpful, seeing as it currently simply contains the index and it was difficult to verify that the dimensional sorts were actually working. That should be included by the next project. Also adding more search options and reworking NoSuchObject are still on the agenda, but are relatively low priority.

**Part #3**

Adding in a dynamically updating JTree as the queue updates would be a useful feature so that one can see how the jobs are processing in real time better. Also, at this current moment, searching appears to work but if one tries to sort at the same time as jobs are processing, the program breaks. I spent a while trying to figure out how to fix this, but it seems to be occurring while iterating over the ArrayList and I wasn’t exactly sure what I was supposed to synchronize for that. I will attempt to have this fixed by the next project.

Lessons Learned

**Part #1**

Early on in the project, I was having trouble with a NullPointerException. It took me a bit to figure out where it was coming from. I realized eventually that I was trying to add something to an uninitialized list. This simply reiterates the importance of initializing variables. I also learned how to use JFileChoosers, since I had not done that previously. Lastly, I also learned how to use for-each loops, as I also have not used those in the past.

**Part #2**

In this portion of the project, I learned how to use Comparators and learned how to more effectively use HashMaps. I also, separately, learned how to use JComboBoxes as I have not previously used dropdowns in Java GUIs. The most complicated portion of this project had to do with the passing of Comparators. Initially, I intended to use a single sort method within each Sorter for all Comparator types. I mistakenly believed that I could pass a Comparator<Ship> to a method which accepted a Comparator<Thing>, since Ship is a subtype of Thing. I discovered that this does not work this way and spent a great deal of time trying to figure out how to force it to work. I ultimately decided to simply make a second method for dealing with Ships. In addition to becoming more familiar with how generics work, I also learned the lesson of not spending too much time attached to a specific solution since once I decided to give up on the single sort method the problem was solved within a matter of minutes, while I had spent a few hours trying to force the previous solution.  
**Part #3**

This portion of the project was my first exposure to thread based programs. I found that debugging a multithreaded program is quite a bit more challenging than a single-threaded program, since I am unaware of a way to step through all of them at the same time, and timing is an important aspect of it. For example, when I was stepping through one thread, the process for that SeaPort worked fine, but when I ran the program, nothing happened. I discovered that this was because the SeaPorts were trying to allocate the workers before all of the objects had been created. Adding a sleep at the beginning of the SeaPort threads fixed that problem, but it was a bit more difficult to pin that down due to the nature of the issue.

I also learned how to use GridBagLayouts in order to arrange my UI better and had not used JTree yet, so those are extra tools that I acquired during this portion of the project.