

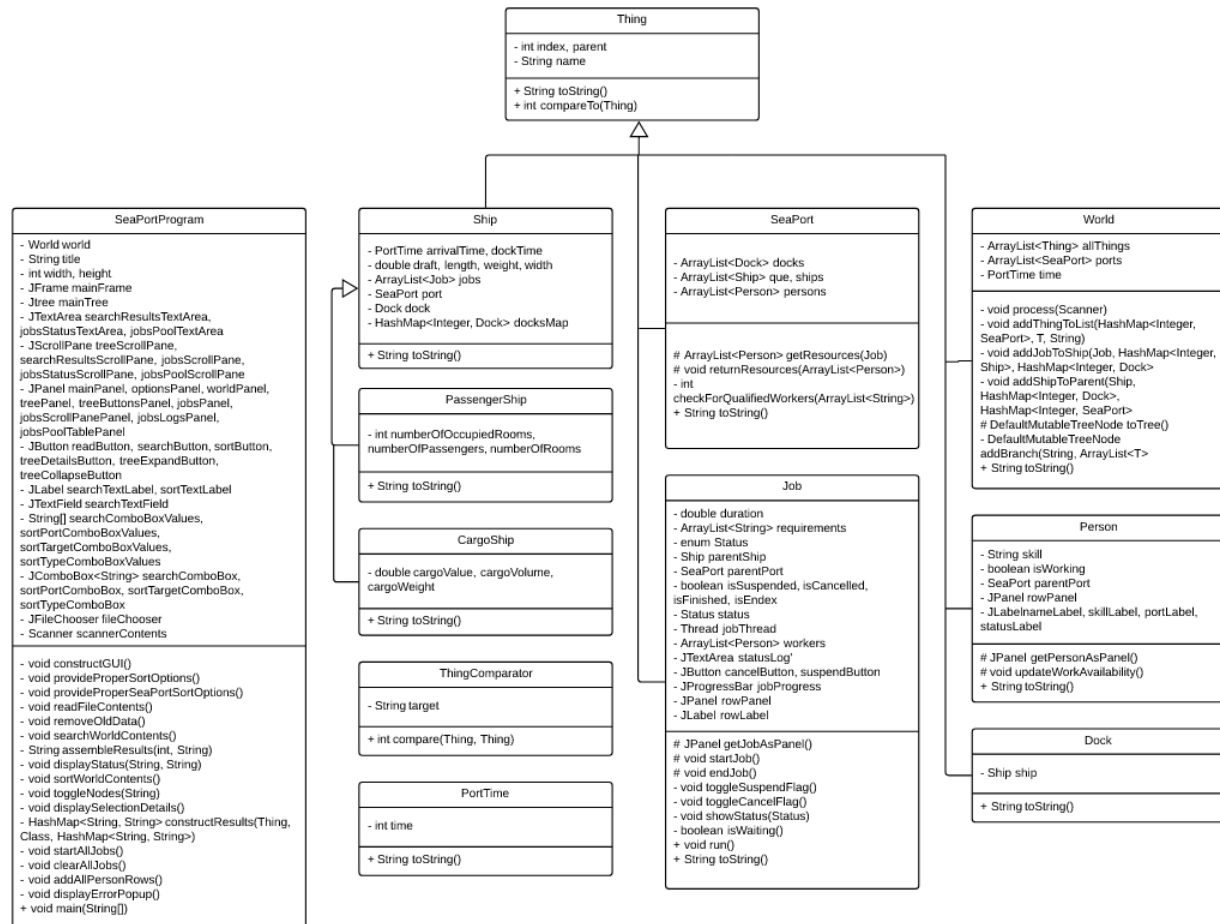
## Project 4 Documentation

02/27/18

Andrew Eissen

UMUC CMSC 335

## Part I: Design



(Author note: Getters/setters not included in UML diagram due to a lack of space and an inability to make the entire diagram fit in one frame to screen capture. Relatedly, all images, diagram and screenshots included, have been included in a zip file for convenient HD viewing.)

As per the first three projects, the only unused class was `PortTime` which was mentioned nowhere in the rubric and was simply included due its mention on the first page. The author is sad to see that this class was never used at all over the course of the SeaPort project progression, as it feels like a red herring the author spent too much theorizing about over the course of the last eight weeks as to its specific purpose and usage.

## Part II: User's Guide

Users unfamiliar with the process of running Java programs may have some difficulty running Project 4 without some prior experience and a set of qualifications. Users must have Java installed on their computer and must have administrator privileges on their chosen machine. Pointing the Windows command prompt to the folder containing the files, users must first compile the classes by typing `javac SeaPortProgram.java`. Once compiled, users may enter `java SeaPortProgram` to run the program and begin interacting with the GUI. Alternatively, if users possess an IDE like NetBeans, they can open `SeaPortProgram.java` and run the program from within after creating a new package.

To prevent invalid input, the program will not open non-text files or ill-formatted 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. Case and spelling are both integral to retrieving the desired results, as the program will not find results that are improperly formatted as they appear in the text file.

## Part III: Test Plan

Upon starting the project, the author discovered he had inadvertently fulfilled most of the Project 4 rubric requirements in his Project 3 submission. As such, the vast majority of the author's Project 4 changes were focused on two specific overarching categories, namely, GUI improvements and outstanding bug fixes.

Regarding the aforementioned GUI updates, the author determined that Project 3's largely text-based approach of displaying workers engaged in certain jobs was not the most conducive way of displaying the worker pool. As such, the former workers' status log was merged with the more general job progression log, with the single `JTextArea` now displaying messages related to both ship mooring/unmooring and worker reservation. Though logs are now significantly lengthier, this combination ensures that all relevant messages related to the job progression operation are all visible in a single location.

Secondly, the newfound space where this previous log entry originally sat has been replaced by a new `JTable`-esque section, listing the names, locations, skills, and availability of all workers in the `World` instance. This section was modelled according to the design ethos of the `Job` listing section from Project 3, wherein each `Person` object possessed a protected method permitting an exterior, GUI class-located method to assemble a pseudo-row `JPanel` for display in a table-like listing. This decision was reached after the author unsuccessfully attempted to implement a similar idea with `JTables` and an `AbstractTableModel` implementation, an approach which technically worked but proved unsatisfactory in the long run. Initially, each `SeaPort` had a specific `JTable` and `AbstractTableModel` of its own in order to more readily demarcate the distinctions between each port's workers, though this was replaced by a global listing created by the GUI class following the removal of the `JTable` implementation.

As such, an additional utility/assembly method was added to the body of `SeaPortProgram` that assembles the individual rows and adds to them to a specific `JPanel`

location, thus maintaining some semblance of consistency with the MVC design paradigm by keeping GUI code in the specific GUI-oriented class. As workers are reserved and returned by `Ships` as their jobs are completed, the color and text of each `Person`'s status `JLabel` is changed to reflect the worker's current situation, whether available or employed. All `Persons` are organized alphabetically by the name of their respective `SeaPort` location.

Concerning bug fixes, the author took the time to look over a number of longstanding bugs that have not been patched, as focus has repeatedly shifted to more pressing matters over the course of the four projects and left little time for related fixes. One such bug that has persisted since the first project concerned instances wherein the user opened the `JFileChooser` modal but exited out of the window prior to file selection. In such cases, the program failed to check if any file had been selected and generally threw an exception related to the fact that a `World` instance was initialized anyway without any proper `Scanner` data. The author repaired this by initializing the file selection to `null` and checking for null cases after the modal's close, returning from such cases without creating a new `World` instance.

A bug related to the `Job` progression process that has been known to the author since Project 3 concerns `Job` instances requiring multiple instances of a certain skill (i.e. a job requires an electrician and two clerks). In such cases, the program lists such jobs as forever waiting, as they believe that enough workers with required skills exist in their port, while in actuality this may not be the case. While it is possible that a port may have an electrician and a clerk, for example, there may be only one clerk, not the two the job requires to process. This case would throw a false positive in `SeaPort.checkForQualifiedWorkers` and

`SeaPort.getResources` that would essentially indicate that all required workers exist (even if not available), while in actuality all that would be indicated would be that the port has workers matching all skills. The fix for this bug was “janky” and likely could be improved upon somewhat. It involves adding qualified workers to an `ArrayList` and checking to see in an `if` statement whether that worker is being listed as a matching candidate multiple times.

Another bug that the author discovered in the course of writing Project 3 is related to cases wherein the port in which the job’s ship exists has no workers, available or otherwise, possessing the necessary skills to process the ship’s job to completion. Previously in Project 3, the author saw fit to simply pass an empty `ArrayList` bereft of worker candidates back from the `SeaPort.getResources` method to ensure that the ship is not kept waiting indefinitely in `Job.isWaiting`, forbidden from progressing. As this method seemed a bit hacky and long-winded in waiting for said thread to die, the author simplified this by adding a few lines to `SeaPort.getResources` that automatically call `Job.endJob` on the thread at hand, naturally ending the thread and marking the job as 100% completed and “Done” in the job progression `JPanel` pseudo-table. This behavior was achieved by adding an invocation of the same method used to cancel a thread/job via the cancelation `JButton`, namely, `Job.toggleCancelFlag`, to the body of `Job.endJob`.

### **Part III b: Test Cases**

The test files employed by the author to test the output of the program were mostly unchanged variants of the Project 3 text files, namely `aSPaa.txt` and `aSPae.txt`.

However, for the purposes of testing the assorted bug fixes implemented over the course of this project, the author added a few new jobs and persons to the latter file. This was done to test the two specific `Job`-based bug fixes, namely cases wherein jobs require skills not possessed by any workers in the port, and cases wherein jobs require multiple instances of the same skill to finish. As such, a new job was added to the *SS Drones* that requires a clown and a basketweaver, skills not possessed by any workers in the Bergen port, and a new worker, cook Clint, was added to help complete a second new job attached to the *SS Absolution* in Baltimore that requires the services of an inspector and two cooks.

Once again, for the aforementioned reasons listed in the Project 3 documentation file, the author has elected to not include `aSPad.txt` from the Project 2 submission due to its sheer size and inherent complexity. However, the few tests of that file the author undertook ended successfully, with all specific job instances completing as expected.

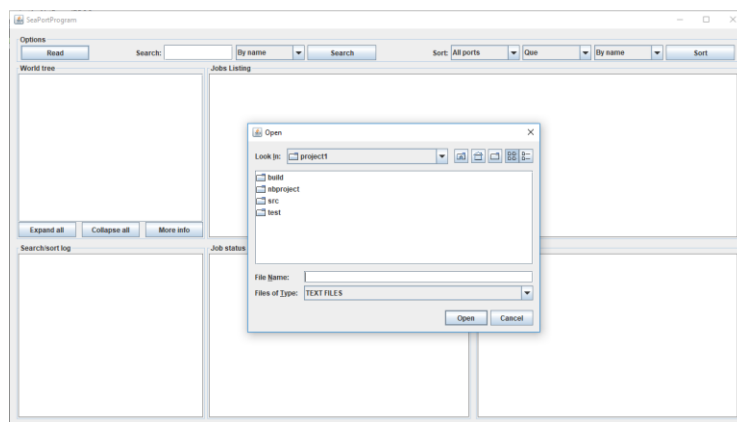
As was the case with the first three project documentation files, the author will not include tests of previously tested functionality, such as the search/sort functions, as no major changes have been made to the underlying methods. Only tests of the `Job` progression-related features and functionality will be included. For details regarding the other functionality, the author recommends a look back at the specific cases included in the Projects 1, 2, and 3 documentation files.

Lastly, due to the inherent complexity of displaying images of processes that occur in real time and would be better suited to `.gif` files, the author has included a bit of verbal

discussion charting the specific operations and actions of the various progressions that may be encountered for each of the text files.

Text File	Input	Expected Output	Output
N/a	Closing JFileChooser w/o selecting file	No error message shown or exception thrown	No error message shown or exception thrown
aSPaa.txt	"Cancel" button on Shoetrees	Ship's progression is halted, button color/text is changed.	Ship's progression is halted, button color/text is changed.
aSPae.txt	"Suspend" button on all running jobs	Progression is paused, button color/text is changed.	Progression is paused, button color/text is changed.
aSPaa.txt	No input	Differs depending on which job gets SeaPort access first; see associated discussion	Differs depending on which job gets SeaPort access first; see associated discussion
aSPae.txt	No input	Differs depending on which job gets SeaPort access first; see associated discussion	Differs depending on which job gets SeaPort access first; see associated discussion

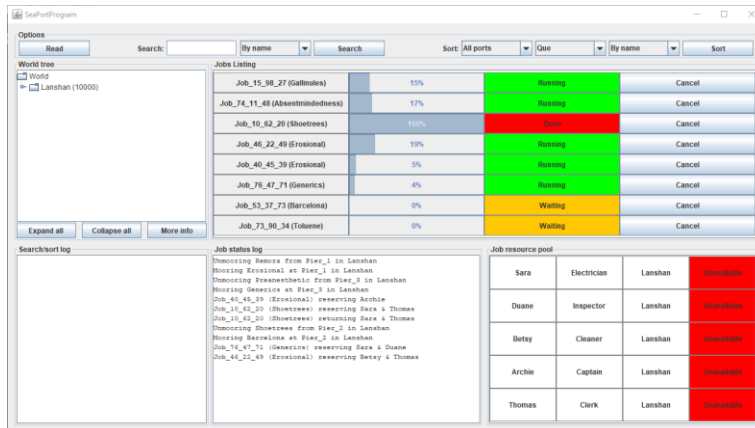
### Test Case 1



In the first test case, the author tested a fix for a longstanding issue, namely the opening of a `JFileChooser` modal and exit prior to file selection. This fix results in a simple return from the method without any erroneous error message displayed or `World` constructed.

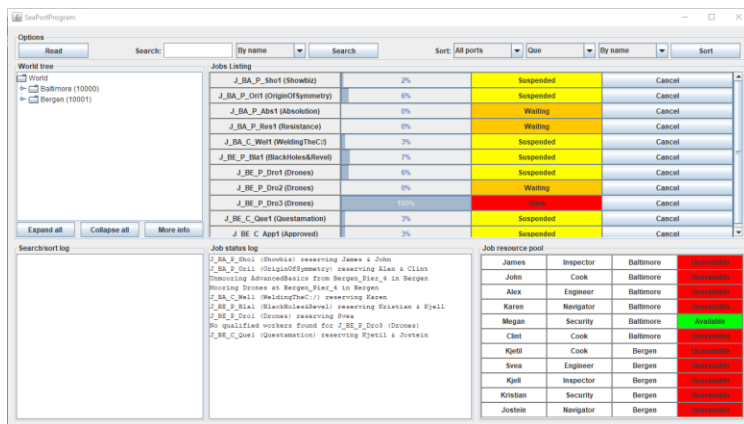


## Test Case 2



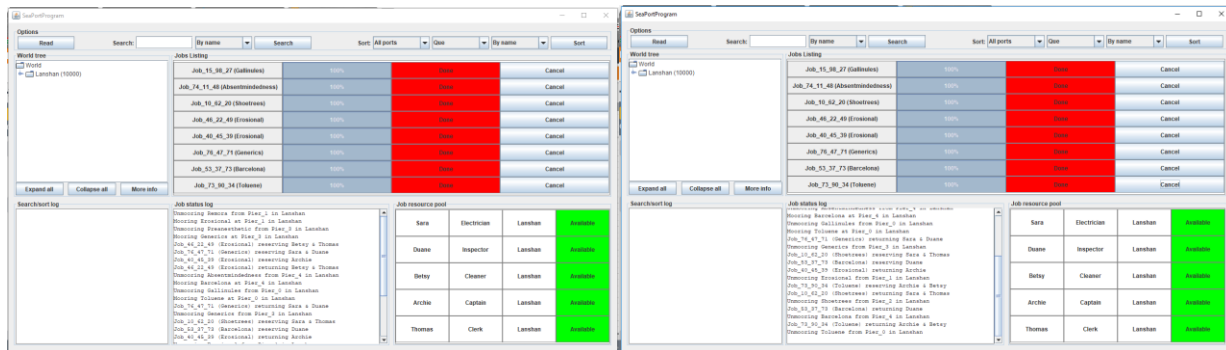
This test case, like that following it, is somewhat of a duplicate of a set of tests run in the Project 3 documentation. It tests the standard function of the “Cancel” button, ending the job attached to the SS *Shoetrees* and displaying it as “Done.”

## Test Case 3



Like the above case, this test ensures that the “Suspend” button operates as intended, pausing all running threads until pressed again. To denote this change of status to the user, the text is replaced with “Suspend” and the background color is changed accordingly.

### Test Case 4: Extended discussion



The first extended test case concerns the default file formerly employed in the Project 3 test series, namely `asPaa.txt`. A default run-through of the program's most basic functions, the job progression in question begins first with the unmooring of two ships bereft of extant jobs, namely *SS Remora* and *SS Preatnestic*, from their Docks, replacing them with the *SS Generics* and *SS Erosional*. Immediately upon docking, these two new vessels reserve the entirety of the worker pool at the Lanshan port, blocking other already-moored vessels from beginning their own jobs. The first of *Erosional*'s two jobs is completed first, releasing cleaner Betsy and clerk Thomas back into the worker pool.

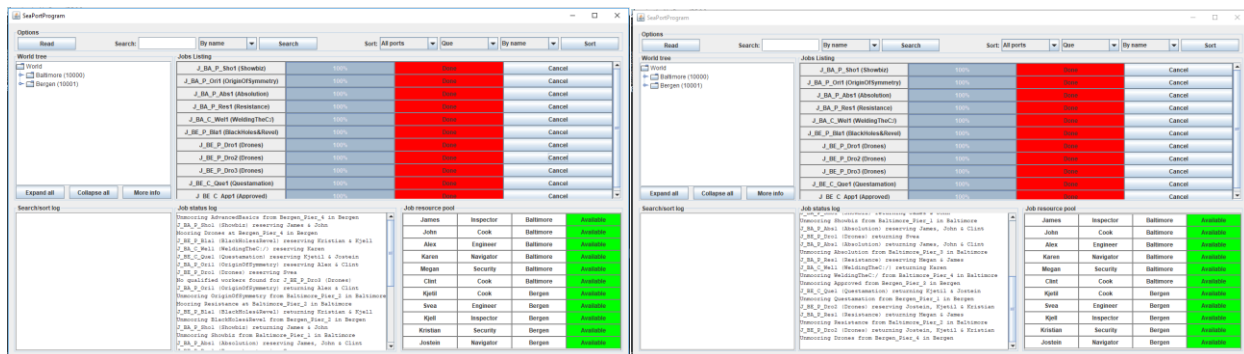
Meanwhile, the *SS Absentmindedness*, whose job requires no workers to complete, finishes its job and unmoors from its dock, making room for the *SS Barcelona*. Likewise, the *SS Gallinules*, also possessing a job requiring no workers, completes its job and unmoors to make room for the *SS Toluene*. Both new vessels remain waiting for their required workers to become available before commencing their jobs.

*Generics* finally completes its job and unmoors, returning electrician Sara and inspector Duane to the job pool. With its workers finally made available, the *SS Shoetrees*, one of the

initially-moored vessels, reserves Sara and Thomas and begins its sole job. Meanwhile, the *Barcelona* reserves its only required worker, Duane, and begins its only job. Eventually, the *Erosional* completes the second of its two required jobs and unmoors, freeing up Captain Archie and Betsy for use by the *Toluene*. One by one, the *Shoetrees*, *Barcelona*, and *Toluene* complete their jobs and unmoor, leaving all workers available and all docks devoid of ships.

Alternate arrangements are also possible, depending on which job thread is granted access to its port's resources first. The author encourages the reader to examine the associated screenshots closer for confirmation of this arrangement, and suggests a perusal of the Project 3 documentation for a number of alternate arrangements using this file.

### Test Case 5: Extended discussion



The second extended test case makes use of the author-created test file `aSPae.txt`, the contents of which have been modified since Project 3 to test issues related to the aforementioned bugs that previously affected certain types of jobs, such as jobs requiring multiple instances of a skill or jobs that can never be completed due to a lack of qualified workers. For ease of readability, job progression will be discussed by individual port.

Starting first in Baltimore, all ships initially moored at their respective piers have jobs in need of processing. The SS *Showbiz* is first to successfully acquire its needed dock workers, removing inspector James and cook John from the resource pool. The SS *WeldingTheC:/* reserves navigator Karen, while the SS *OriginOfSymmetry* reserves the second cook Clint and engineer Alex, leaving only security guard Megan in the worker pool.

Eventually, the *OriginOfSymmetry* and *Showbiz* complete their sole jobs, releasing their workers and unmooring from their piers. The SS *Resistance* takes the *OriginOfSymmetry*'s place at its dock, but is unable to proceed with its job requiring an inspector and security guard until James becomes available. However, inspector James, along with cooks John and Clint, are first reserved by the SS *Absolution*, which blocks the remaining ships from continuing with their jobs until its conclusion. This vessel's job in particular was included to test formerly buggy cases involving jobs requiring more than one of a certain skill, in this case, cooks.

With the *Absolution*'s unmooring, the *Resistance* is finally able to begin its own job. With the return of its workers and the conclusion of the long-running *WeldingTheC:/* job, all workers are returned to the Baltimore job pool and execution of all Baltimore-bound ship threads draws to a close.

Meanwhile in Bergen, Norway, the job progression begins with the unmooring of the SS *AdvancedBasics*, a vessel initially moored at a Dock but bereft of any jobs to complete. Its place is taken by the SS *Drones*, which begins the first of its three jobs alongside the SS *BlackHoles&Revel* and SS *Questamation*. Between the three vessels, the entire Bergen worker pool is drained, though the SS *Approved*, requiring no workers to complete its job, continues to

progress with its job. As these jobs are begun, the third of the *Drones'* jobs, requiring a clown and a basketweaver, is automatically cancelled due to a distinct lack of workers possessing such skills in the worker pool. This was included to test the behavior required by the Project 4 rubric to automatically end a job if the job's requirements could never be met.

The remaining *Drones* job requiring Jostein, Kjetil, and Kristian does not begin until the first three aforementioned jobs on the *Drones*, *BlackHoles&Revel*, and *Questamation* are completed. With the completion of this remaining job, the vessel unmoors, returning all workers to the pool and leaving the piers devoid of vessels.

Alternate arrangements are also possible, depending on which job thread is granted access to its port's resources first. The author encourages the reader to examine the associated screenshots closer for confirmation of this arrangement, and suggests a perusal of the Project 3 documentation for a number of alternate arrangements using this file.

#### **Part IV: Comments**

Overall, the author is generally pleased with the manner in which the SeaPort project series has turned out. A few areas of interest for expansion or revamp are listed herein simply for the purposes of conceptual theorization; in all likelihood, the author will probably never touch this program again after submission.

As far as the GUI design goes, the author is generally satisfied with its overall appearance and layout, though the application of a more advanced layout manager from the

start may have been a better option than the overlapping mass of `GridLayout` and `FlowLayout` arrangements currently present in the program's GUI class. The author does wish that more functionality had been added to the `JTree` section, perhaps updating its contents in real time depending on the location of each specific ship (i.e. rebuilding or refreshing the `JTree` every time a ship is docked, processed, and moved out of the queue, etc.).

As far as back-end code is concerned, the author would likely have expanded upon the sorting function added in Project 2 to include more parameters and variability for users to employ. Relatedly, both the sort and search functions should probably have included more information pertaining to each desired entry, such as location of the `Thing` in question, its skills/requirements, etc. to prove more useful to users. At present, only the name and the associated index ID are displayed for users, which offers little actual information about the item in question and makes searching rather worthless.

## **Part V: Lessons Learned**

As per the third project, the author once again learned a fair bit about the specifics of `JTables` and their specific uses and disadvantages in such cases as the listing of `Persons`. Though the author was successfully able to assemble and display a set of `JTables` pertaining to the individual workers in each `SeaPort`, the display was less than optimal and subsequently replaced by a pseudo-row implementation similar in design to that seen in the `Job` progression panel. Though the research and experimentation undertaken by the author to attempt this

method in Project 4 was ultimately not employed herein, the author nonetheless managed to solidify his understanding of `JTables`' usefulness in displaying data in an organized, coherent fashion.

Furthermore, use of the NetBeans JConsole plugin was helpful in the course of testing the program's reaction to hard-stop thread cancellation orders, as it provided the author with a more concrete means of monitoring thread operation in real time. The independent research undertaken to better understand this useful tool provided invaluable in this case, and the author wishes he had had access to JConsole for the previous project, the writing of which was largely done blind.