SEA PORT APPLICATION

Project 3

Abstract

Use the Swing class JTree effectively to display the contents of the data file. Implement a thread for each job representing a task that ship requires. Use the synchronize directive to avoid race conditions and insure that a dock is performing the jobs for only one ship at a time. Use a JProgressBar for each job to display the progress of that job.

Xavier DAVIS

Project 3

CMCS 335

February 18, 2018

Xavier Davis
CMSC 335
February 18, 2018

Release Notes:

- **Runnable Job Threads**
  - **Pause, Stop, Resume each thread**
- **Job workforce hiring**
  - **Compete for workers based on job requirements and available skilled persons (first come, first served)**
- **GUI progress display**
  - **Give a real time update of job completion status**
- **JTree**
  - **Information from data file structured into a JTree with appropriate nodes**

Class List

Class Name	Description
SeaPortProgram.java	Constructs and displays the GUI interface of the application. Also passes the search information to World.java which fetches a list of search results to display to user. GUI interface contains two JComboBoxes for search modifiers, a JTextField for search input and a submit button. Two JScroll panels included to display the data file content parsed by World.java and the other panel to display search results.
World.java	Parses the data file selected from the SeaPortProgram.java class, to display to the user on the GUI interface. As the data file is parsed the class creates the appropriate objects encompassed within the SeaPort world. During the creation of each object it is assigned to the proper parent object by use of the parent index value included in the data line and the HashMap object within the file parsing method. Class contains search functionality to fetch a results list of the user's search operation.
SeaPort.java	Object to represent a sea port that contains a list of assigned people, docks, and ship objects. Getter and setters are included to associate the objects to the SeaPort instance.
Thing.java	A parent class for the classes, Ship, CargoShip, PassengerShip, Dock and Person. Thing.java contains information on the object's name, index value and index of the parent object.

Xavier Davis
CMSC 335
February 18, 2018

Ship.java	A parent class for the classes CargoShip and PassengerShip. Class contains the basic information about the ship's dimensions, port times, and a list of jobs associated with the Ship object.
CargoShip.java	A child class of the Ship class. Extends the parent class to contain characteristics of a container ship (cargo weight, volume, and value).
PassengerShip.java	A child class of the Ship class. Extends the parent class to contain characteristics of a passenger ship (number of passengers and rooming information).
Dock.java	Class represents an individual dock located within a SeaPort. Contains a getter and setter for a ship assigned to the Dock object.
Person.java	Object for individuals located within a SeaPort. Each person extends Thing.java to contain a skill.
Job.java	Each Job contains a list of required skills and a total time duration for the Job's completion. Class extends Thing.java.
PortTime.java	Simple class to contain the information on a Ship's port time.

Class Variable List

Variable Type	**Variable Name**	**Variable Description**
SeaPortProgram.java		
Font	font	Establishes the font display of GUI components
World	world	Instance of the World.java class
JTable	resultsTable	Formatted JTable of the search results
World.java		
ArrayList<SeaPort>	ports	A list of SeaPort objects created from the data file.
PortTime	time	Instance of the PortTime.java class
Thing.java		
int	index	Unique index value of Thing object
int	parent	Unique index value of parent object
String	name	Name of Thing object

Xavier Davis
CMSC 335
February 18, 2018

Ship.java		
PortTime	arrivalTime	Ship arrival time
PortTime	dockTime	Ship dock time
double	draft	Total draft measurement of ship
double	length	Total length measurement of ship
double	weight	Total weight of ship
double	width	Total width of ship
ArrayList<Job>	jobs	A list of jobs assigned to ship
SeaPort.java		
ArrayList<Dock>	docks	List of docks within the SeaPort
ArrayList<Ship>	queue	List of ships awaiting to be docked
ArrayList<Ship>	ships	List of all ships in port
ArrayList<Person>	persons	List of all people assigned to port
PortTime.java		
int	time	Port time
Person.java		
String	skill	Skill of person
PassengerShip.java		
int	numberOfOccupiedRooms	Number of Rooms occupied on passenger ship
int	numberOfPassengers	Total number of people on passenger ship
int	numberOfRooms	Total number of rooms on passenger ship
Job.java		
double	duration	Total duration of a job
ArrayList<String>	requirement	List of required skills for a job
Int	NUM_REQUIREMENTS	Total number of job requirements
JProgressBar	progressBar	Returns progress bar
Boolean	isRecruitingComplete	Flag for recruiting
boolean	startedJob	Flag to starting job
boolean	lackSkill	Flag for unskilled work force
Thread	thread	Thread object to start runnable
boolean	isPaused	Flag for paused thread
Dock.java		
Ship	ship	Instance of the Ship.java class
CargoShip.java		
double	cargoValue	Total amount of cargo value aboard cargo ship
double	cargoVolume	Total cargo volume aboard cargo ship

double	cargoWeight	Total cargo weight aboard cargo ship

Application Method List

Return type	Method Signature	Description
SeaPortProgram.java		
void	showGUI ()	Constructs and displays the GUI interface
ArrayList<String>	search (String searchType, String searchAttribute, String searchKeyword)	Stores the returned search results from World.java into an ArrayList
File	selectFileMenu ()	Displays the file selector GUI to upload the data file for the application
World.java		
void	assignShip (Ship ship)	Assigns a Ship object to a Dock object
int	getNumPorts()	Returns total number of ports in World.java
void	readFile (File file)	Reads the data file selected from the selectFileMenu method
void	parseData (String line, HashMap<Integer, Thing>)	Creates the appropriate object based on the current line from data file.
ArrayList<? Extends Thing>	search (String subject, String attribute, String keyword)	Searches all possible subject objects within the SeaPort application and returns an ArrayList of those objects.
ArrayList<? Extends Thing>	scanSearchList (ArrayList<? Extends Thing> searchList, String attribute, String keyword)	Scans the list of subjects returned from the search method within World.java for matches based on the keyword parameter.
LinkedList<JTree>	getJTreeList()	Iterates through all world ports and calls the createTree method on each port. Once each port create a JTree it is added to a linked list and return to the GUI
Thing.java		
int	getIndex ()	Returns index value of Thing object

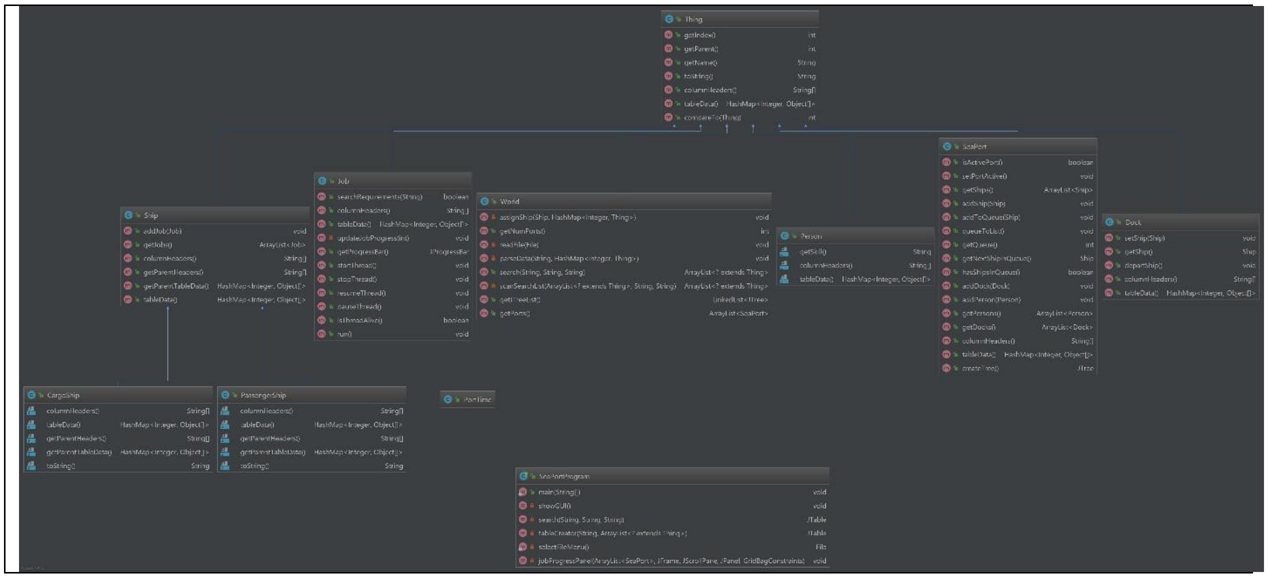
int	getParent ()	Returns parent index of Thing object
String	getName ()	Returns name of Thing object
int	compareTo(Thing o)	Compares the two Thing objects names.
String[]	columnHeaders()	Method to return column headers for Thing objects
HashMap<Integer, Object[]>	tableData()	Method to return row data for Thing objects.
Ship.java		
void	addJob (Job job)	Adds a job to a Ship object
ArrayList<Job>	getJobs ()	Returns list of Job attached to Ship object
HashMap<Integer, Object[]>	getParentTableData()	Method to return row data from parent class object. Used with instances of ships.
HashMap<Integer, Object[]>	getParentHeaders()	Method to return column headers from parent class object. Used with instances of ships.
Comparator<Ship>	WeightComparator	Comparator for two Ship weight values
Comparator<Ship>	LengthComparator	Comparator for two Ship length values
Comparator<Ship>	WidthComparator	Comparator for two Ship width values
Comparator<Ship>	DraftComparator	Comparator for two Ship draft values
Comparator<Double>	doubleComparator	Comparator for two double values—used in results JTable
SeaPort.java		
ArrayList<Ship>	getShips ()	Returns a list of Ships in a SeaPort
void	addShip (Ship ship)	Adds a Ship to a SeaPort
void	addToQueue (Ship ship)	Adds a Ship to a SeaPort's queue
void	queuetoList()	Converts the ArrayList queue into a LinkedList structure
int	getQueue()	Returns size of queue linked list
Ship	getNextShipInQueue()	Pops the next ship off the linked list queue so ship can be docked
boolean	hasShipsInQueue()	Returns if more ships remain in port queue
void	addDock (Dock dock)	Adds Dock to a SeaPort
void	addPerson (Person person)	Adds a passenger to a SeaPort

Xavier Davis
CMSC 335
February 18, 2018

ArrayList<Person>	getPerson ()	Get a list of people assigned to a SeaPort
ArrayList<Dock>	getDocks ()	Returns a list of Docks within a SeaPort
JTree	createTree ()	Creates JTree structure of port objects for GUI interface
Person.java		
String	getSkill ()	Returns a Person's skill
Dock.java		
void	setShip (Ship ship)	Sets the Ship object assigned to a Dock
Ship	getShip ()	Returns the Ship assigned to a Dock
Job.java		
boolean	searchRequirements(String skill)	Scans the job requirements to see if a requested skill is present in list
void	updateJobProgress(int remainingOpenings)	Updates the JProgress bar based on the number of job openings remaining
JProgressBar	getProgressBar()	Returns JProgressBar
void	startThread()	Starts Thread
void	stopThread()	Terminates thread -- disallows running by setting altering thread dependent loop condition variable.
void	resumeThread()	Resumes thread -- sends notification to object lock for continued use
void	pauseThread()	Pauses thread
boolean	isThreadAlive()	Returns if thread is running.
void	run()	Runs the thread to start recruiting for job.

Xavier Davis
CMSC 335
February 18, 2018

UML Diagrams:


Thing
getIndex() int
getParent() int
getName() String
toString() String
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
compareTo(Thing) int
Ship
addJob(Job) void
getJobs() ArrayList<Job>
columnHeaders() String[]
getParentHeaders() String[]
getParentTableData() HashMap<Integer, Object[]>
tableData() HashMap<Integer, Object[]>
Job
searchRequirements(String) boolean
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
updateJobProgress(int) void
getProgressBar() JProgressBar
startThread() void
stopThread() void
resumeThread() void
pauseThread() void
isThreadAlive() boolean
run() void
World
assignShip(Ship, HashMap<Integer, Thing>) void
getNumPorts() int
readFile(File) void
parseData(String, HashMap<Integer, Thing>) void
search(String, String, String) ArrayList<? extends Thing>
scanSearchList(ArrayList<? extends Thing>, String, String) ArrayList<? extends Thing>
getTreeList() LinkedList<JTree>
getPorts() ArrayList<SeaPort>
Person
getSkill() String
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
SeaPort
isActivePort() boolean
setPortActive() void
getShips() ArrayList<Ship>
addShip(Ship) void
addToQueue(Ship) void
queueToList() void
getQueue() int
getNextShipInQueue() Ship
hasShipsInQueue() boolean
addDock(Dock) void
addPerson(Person) void
getPersons() ArrayList<Person>
getDocks() ArrayList<Dock>
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
createTree() JTree
Dock
setShip(Ship) void
getShip() Ship
departShip() void
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
CargoShip
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
getParentHeaders() String[]
getParentTableData() HashMap<Integer, Object[]>
toString() String
PassengerShip
columnHeaders() String[]
tableData() HashMap<Integer, Object[]>
getParentHeaders() String[]
getParentTableData() HashMap<Integer, Object[]>
toString() String
PortTime
SeaPortProgram
main(String[]) void
showGUI() void
search(String, String, String) JTable
tableCreator(String, ArrayList<? extends Thing>) JTable
selectFileMenu() File
jobProgressPanel(ArrayList<SeaPort>, JFrame, JScrollPane, JPanel, GridBagConstraints) void

Fulfilling Project Requirements:

JTree

The new addition of the JTree was a seamless process as I simply used the framework of the SeaPort toString structure to construct each JTree. The toString method used for loops to iterate through all the dock, ships, and people of the port. So instead of capturing this information and storing it as a String, I used each for loop object as a new node of the JTree using the name of the sea port as the parent node. When all SeaPort JTrees were created I added each JTree to a list which I would later iterate through to add to the main GUI interface.

Threads

To effectively run the multiple threads without deadlock or resource interference I had to establish a suitable object lock that was common among each thread. That lock was allocated as the skillMap object of World.java. Only one thread could acquire a lock of the object by utilizing the synchronize keyword on the object. Once a thread properly acquired the lock, the job recruiting process could be performed. The lock is release if all job requirements has been filled or if it was determined that the available workers within the skillMap did not provide the skills necessary for the job. The running of the thread or “recruiting” process continues to run as long as the job is not deemed completed.

Being able to pause, stop, and resume the thread was mainly accomplished by the checking of the pause variable at the start of each while loop iteration. If the thread was paused by the user, the object lock on skillMap would be released so that other threads are not held up by the halted process. There is also an ability to resume a thread once it has been paused by pressed the resume button. The resume thread operation cannot be executed once a thread has stopped.

Progress Bars

Added to the constructor of each Job object is the creation of a JProgressBar which is added to the GUI interface at the start of each job thread. As job opening are being filled in the recruiting process, the progress bars will increase in completion percentage until the recruiting process has been deemed completed.

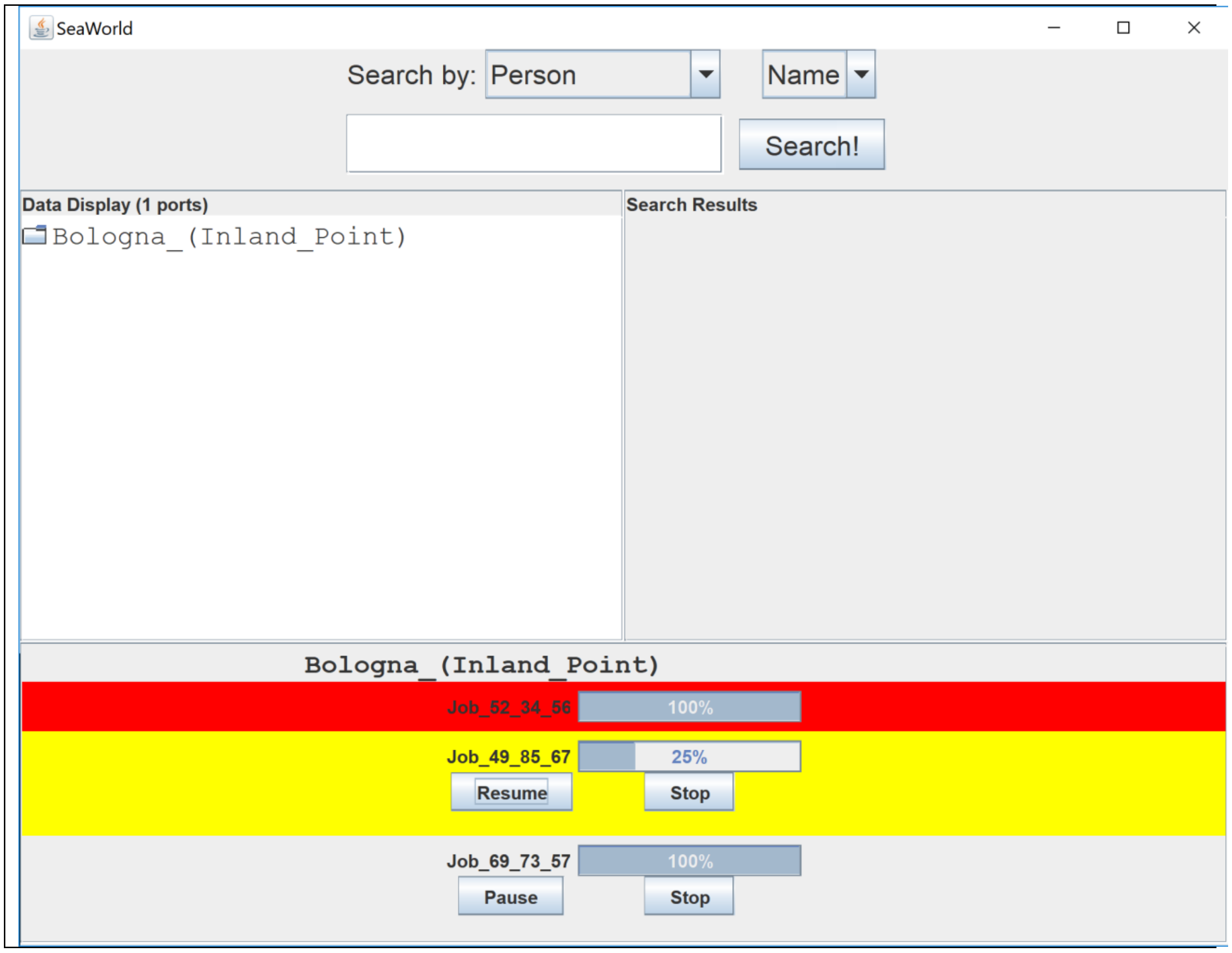
Test Plan:

Test Case	Input	Expected Output	Actual Output
Pausing Thread	Pressing of the pause button	The thread should suspend and give a GUI indication of the action	**See Below**
Resuming Thread	Pressing of the resume button	Should resume the suspended thread and give a GUI indication of the action	**See Below**
Stopping Thread	Pressing of the stop button	The thread should terminate and give a GUI indication of the action	**See Below**
Single Port World	Create and select the data file containing only one port in the world.	Loop continues to iterate through all the docks until all jobs are performed.	**See Below**
Multi-port World	Create and select the data file containing only multiple ports in the world.	Loop continues to iterate through all the docks until all jobs are performed.	**See Below**
Job w/ no requirements	Select a data file with a job listed with no requirements.	A random worker selection process is activated to fulfill the job listing.	**See Below**
Navigating JTree	Extend each node of JTree to ensure information population	All node should be populated with the names of all ports, ships, docks and people.	**See Below**

Xavier Davis
CMSC 335
February 18, 2018

Test Case Screenshots:

Stop, Pausing and Resuming Thread


SeaWorld

Search by: Person Name

Search!

Data Display (1 ports)

Bologna_(Inland_Point)

Search Results

Bologna_(Inland_Point)

Job_52_34_56 100%

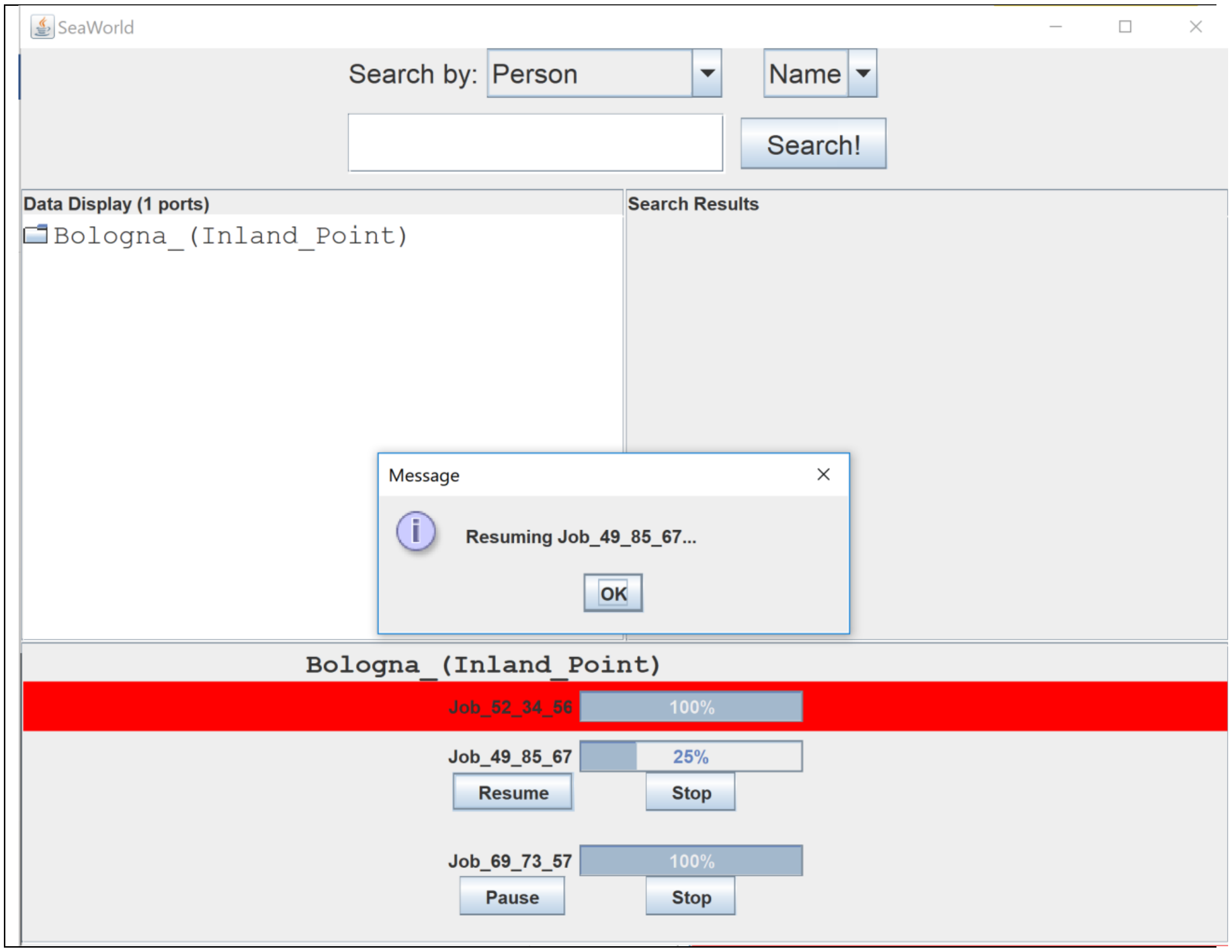
Job_49_85_67 25%

Resume Stop

Job_69_73_57 100%

Pause Stop

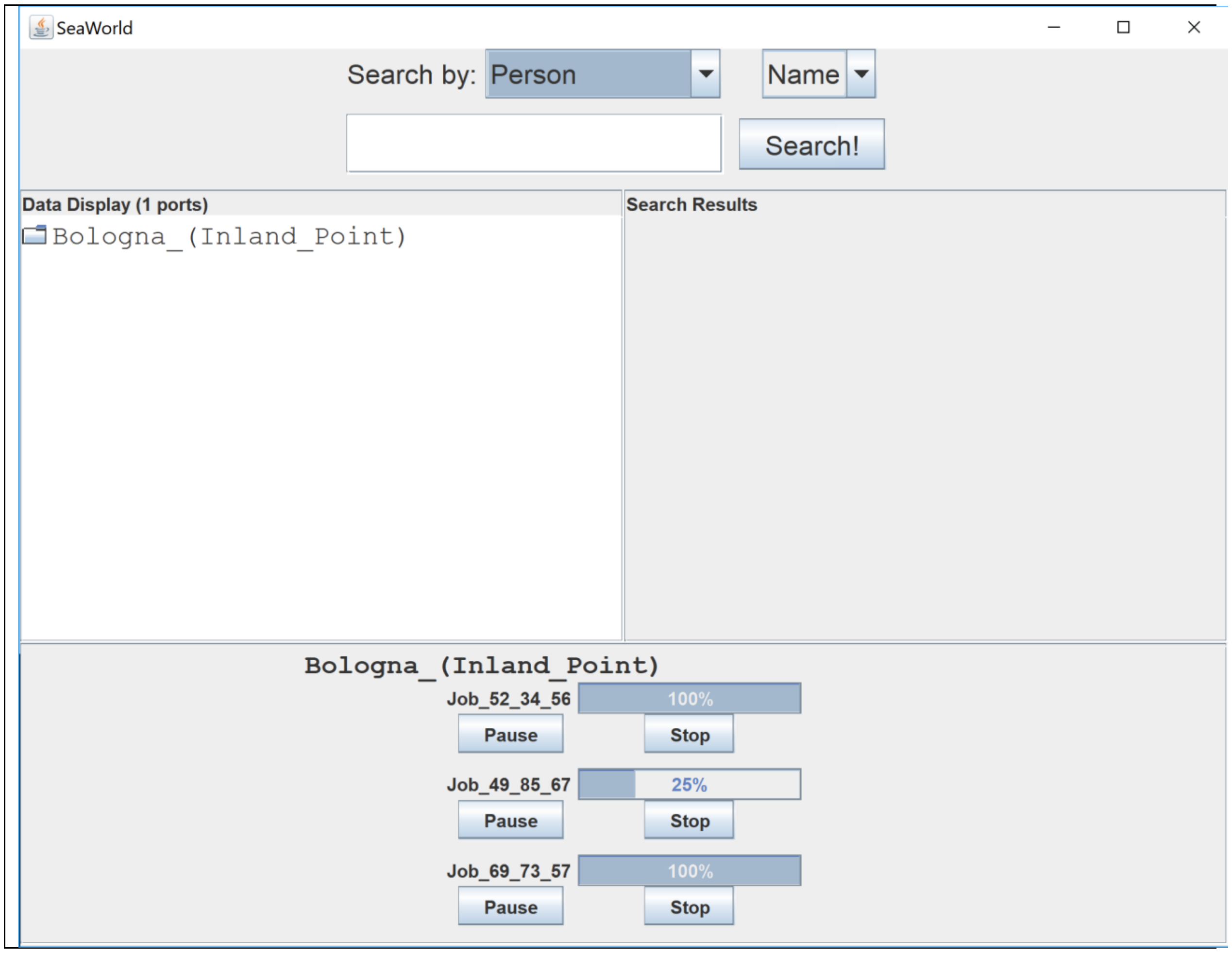
Xavier Davis
CMSC 335
February 18, 2018



Resuming Job_49_85_67...

Xavier Davis
CMSC 335
February 18, 2018

Data File with 1 Port

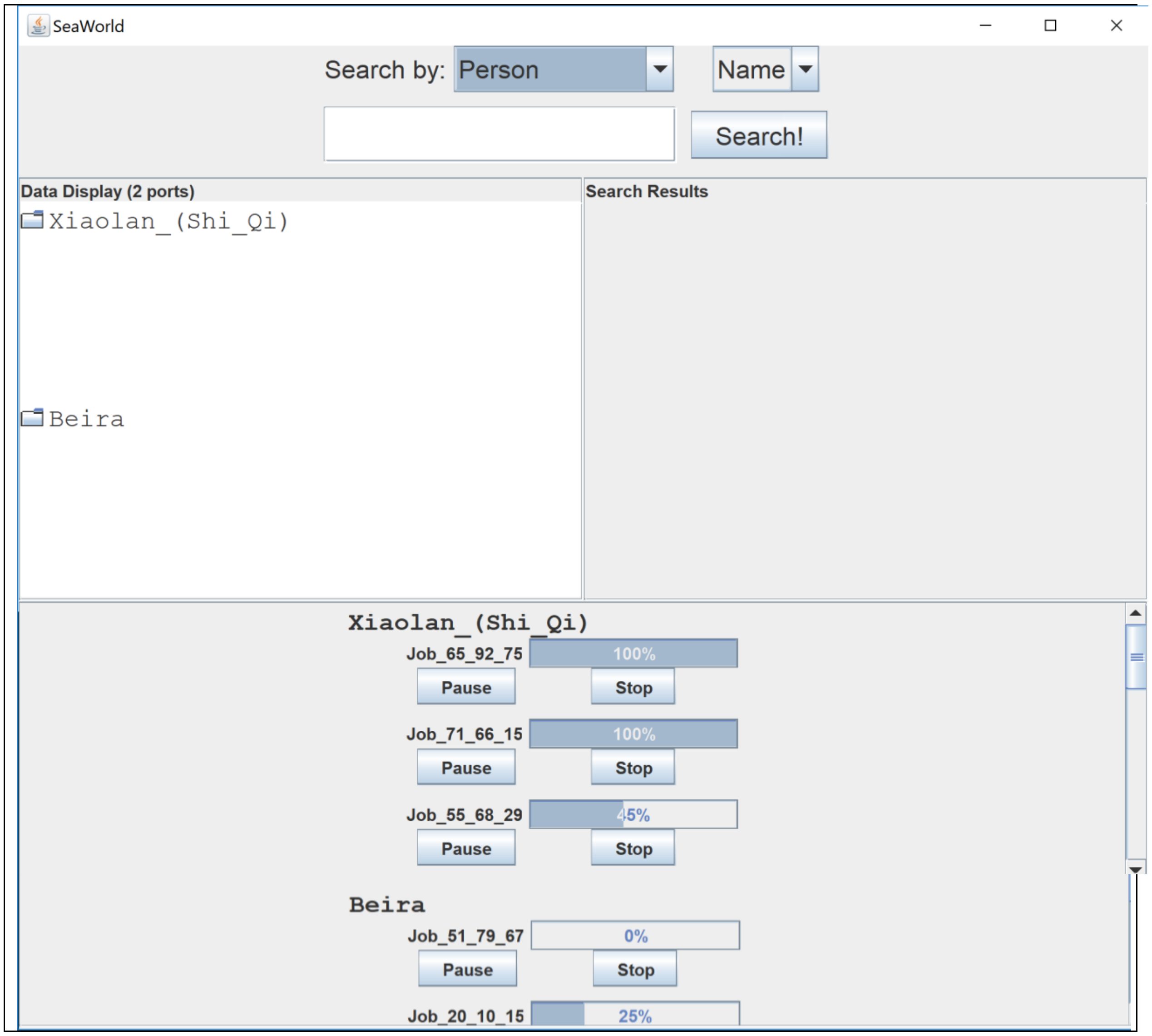


Bologna_(Inland_Point)

Bologna_(Inland_Point)
Job_52_34_56 100%
Job_49_85_67 25%
Job_69_73_57 100%

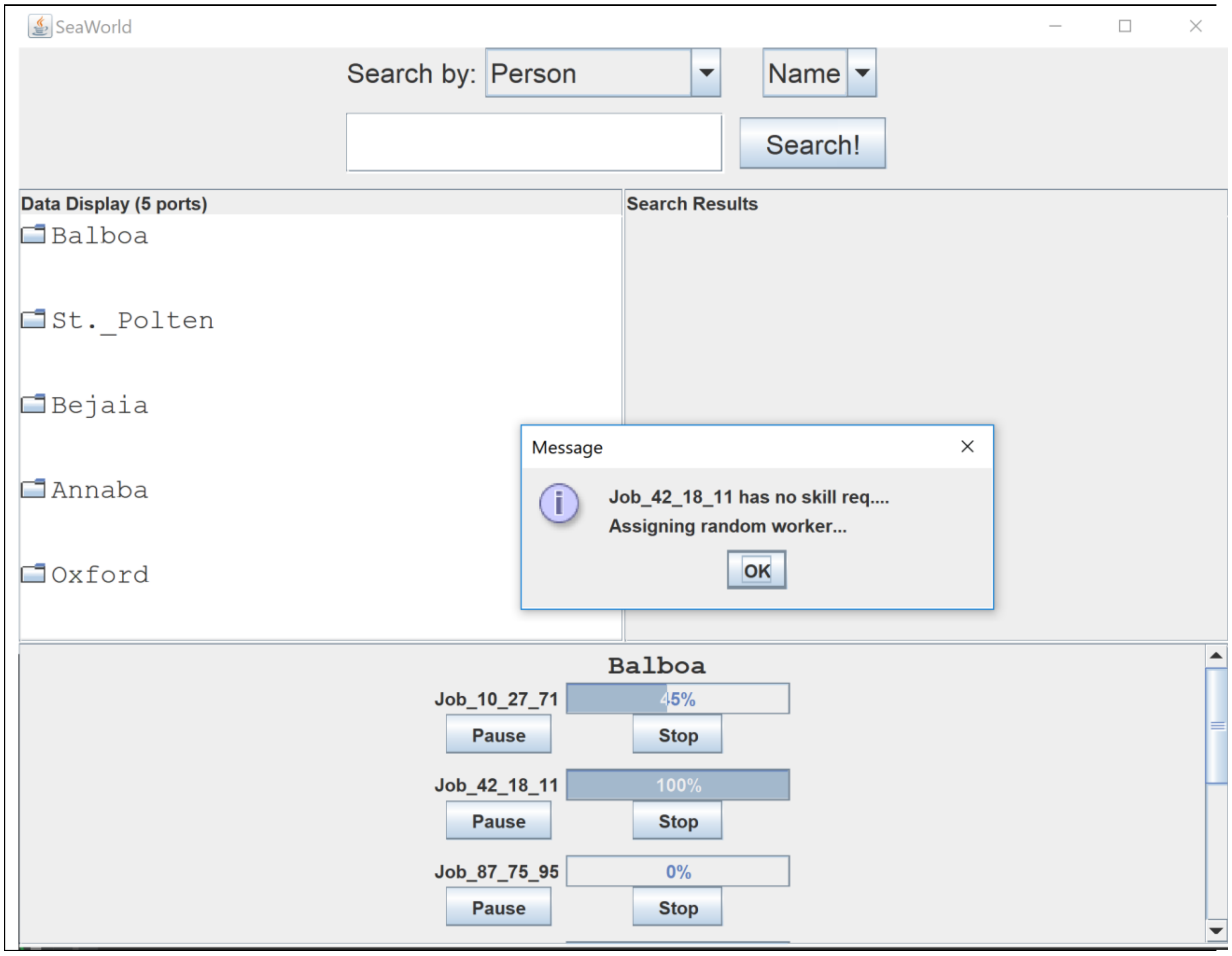
Xavier Davis
CMSC 335
February 18, 2018

Data File with Multiple Ports



Xavier Davis
CMSC 335
February 18, 2018

Data file Containing Job with No Requirements

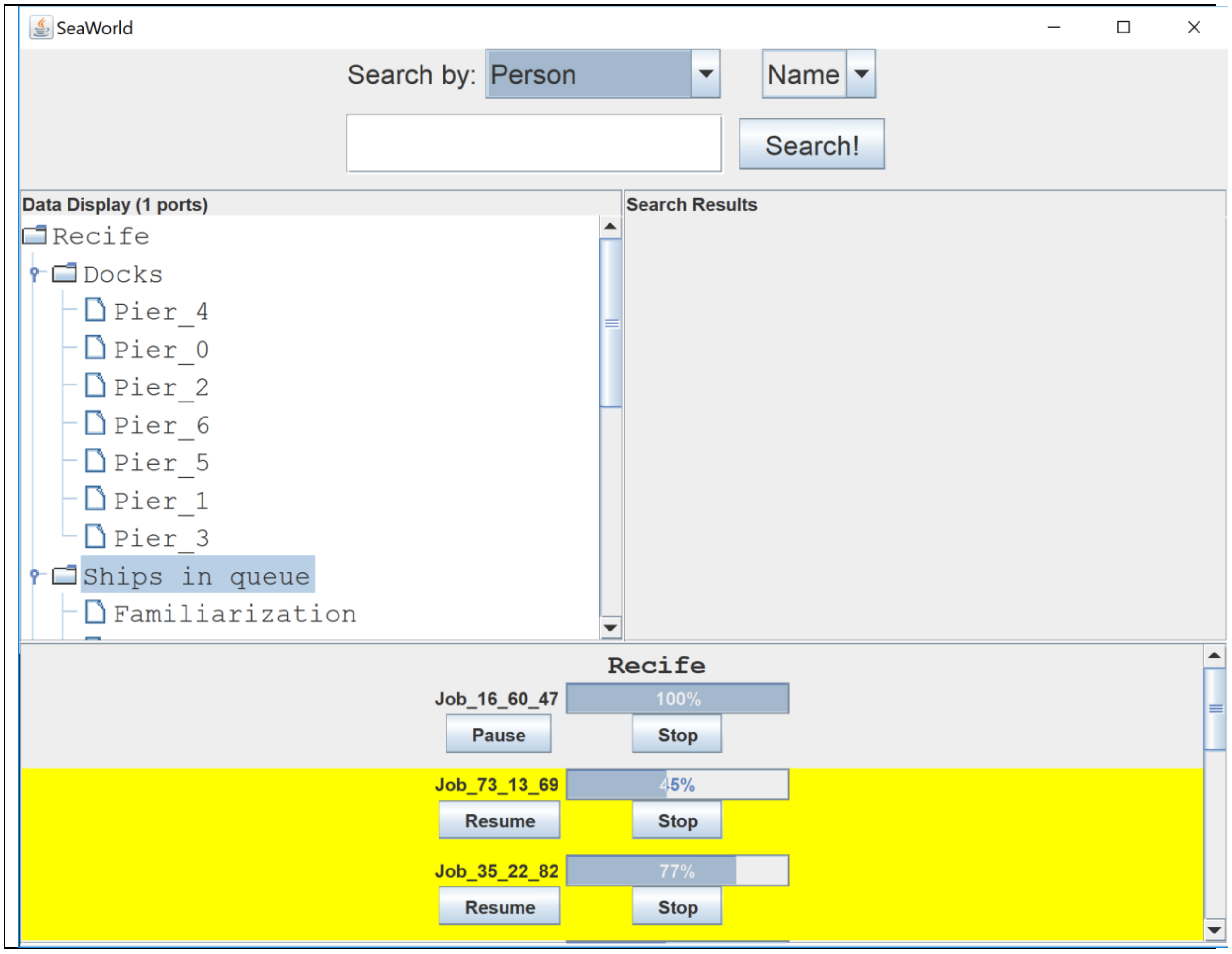


Message

Job_42_18_11 has no skill req....
Assigning random worker...

Xavier Davis
CMSC 335
February 18, 2018

Navigating JTree



Lessons Learned:

Learning how to properly obtain monitor locks through concurrent programming has been the first time in a while were I have been taken out of my comfort area with Java. Through this project I read countless documents and reviewed multiple source code examples to gain an understanding of how the JVM monitors work and the syntax structure of synchronized code blocks. One challenge that I would point out is the updating of the JProgress bars across multiple threads. I can honestly say I rewrote this source code 5 to 6 times to find the correct functionality.

Sea port application

User Application Guide

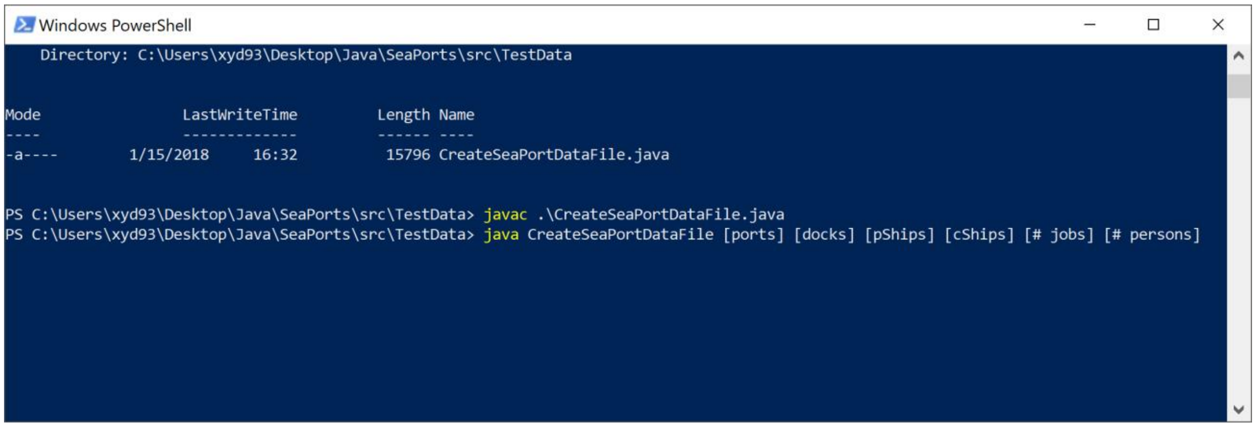
Xavier DAVIS
Project 3
CMSC 335
February 4, 2018

Xavier Davis
CMSC 335
February 18, 2018

Starting the application

Create the data files

Before the user can run the Sea Port application, you must create the data files to population the information about the sea port. This can be done with the command line or a java IDE. Since settings and IDE interfaces differ, this guide will focus on the procedures with the command prompt. Exported in the zip file is a java file titled CreateSeaPortDataFile. For job practice compile the file using the command '**javac CreateSeaPortDataFile.java**' as shown below. Next run the java file with the command arguments for the number of ports, docks, ships, jobs, and people within the application world. The screenshot below shows the order in which these values should be entered. Once the command is executed a data file will be created in the current directory. Remember the file location of the data file, as it will be needed to later select the file when the Sea Port application runs.



```
    Directory: C:\Users\xyd93\Desktop\Java\SeaPorts\src\TestData


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----        1/15/2018     16:32          15796 CreateSeaPortDataFile.java


PS C:\Users\xyd93\Desktop\Java\SeaPorts\src\TestData> javac .\CreateSeaPortDataFile.java
PS C:\Users\xyd93\Desktop\Java\SeaPorts\src\TestData> java CreateSeaPortDataFile [ports] [docks] [pShips] [cShips] [# jobs] [# persons]
```

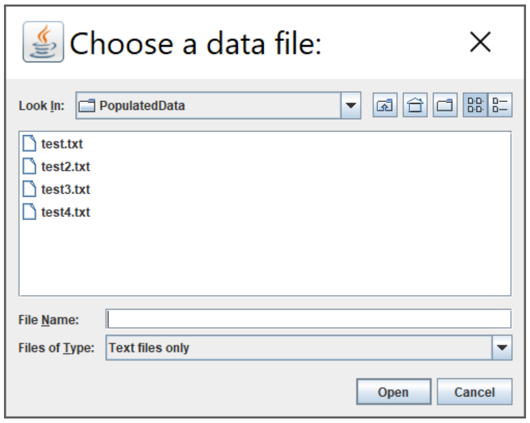
Launch application

The process of compiling and running the SeaPort application is identical to how the process was executed for the creation of the data files. Only difference is to be sure to navigate to the file directory the SeaPortProgram file is located or use the absolute file location when complying and running outside of the applications home directory. When running the main application there are no additional arguments that need to be passed. After compiling the simple command java SeaPortProgram will launch the GUI (assuming the command was executed within the application home directory).

Selecting a Data File

Once the application is loaded the first screen presented to the user is the data file selector interface. On this screen simply select the data file you want to utilize for the application and click open at the bottom of the window to proceed.

Xavier Davis
CMSC 335
February 18, 2018

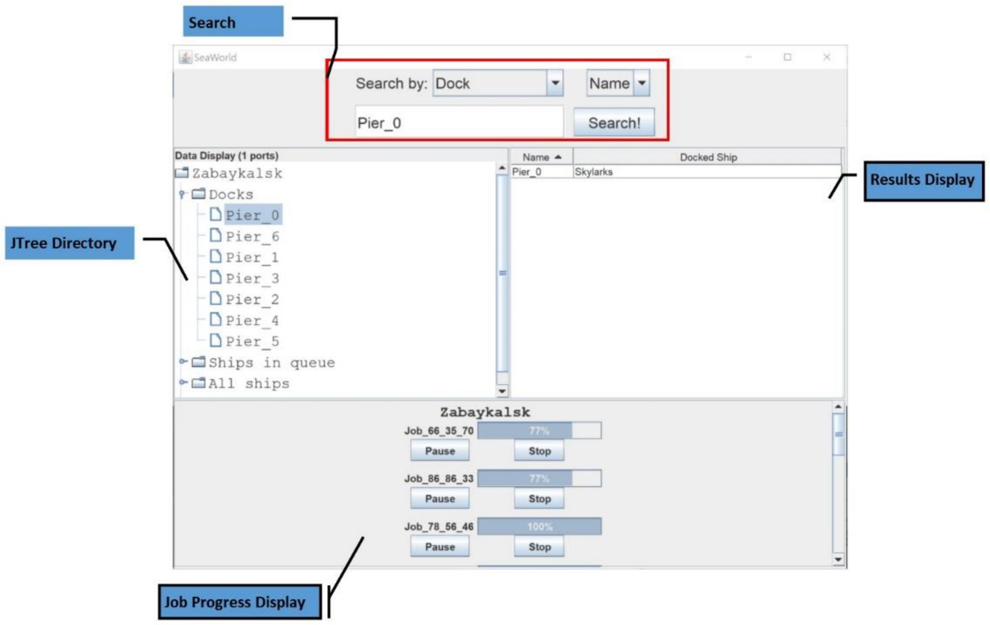


Search Capabilities

The main GUI display has a simple layout. On the screen are two dropdown box selectors to modify the search. Users can search by Person, Ships, Docks, Ports, and Jobs. Once the search subject is selected, users can select a subject's attribute to search for. These include name, and skills. (**Note**: The skill attribute is only searchable for subjects Person and Job as Ship and Dock objects do not contain skill information.) If the subjects of Ship or Dock is selected the user will only be presented with the search attributes of index and name for the reason previously stated. Once all search options are selected, the user can utilize the search text field to input the search keyword and execute the search by clicking the search button.

Below the search components are two scroll panes. The pane on the left side is the presentation display of all the information extracted from the data file selected by the user. On the right side of the screen is the display of any search results queried from the user's search. The printed results will be formatted in a table accompanied by table column headers. Each column header is sortable by clicking on the header. Sorting can be performed in both ascending and descending order. If the user's search fails to gather any results, then they would be presented with a "No results found" message in place of search results.

Xavier Davis
CMSC 335
February 18, 2018



Search by: Dock   Name

Pier_0

Name	Docked Ship
Pier_0	Skylarks

```
Job_66_35_70  77%
Job_86_86_33  77%
Job_78_56_46  100%
```