SEA PORT APPLICATION

Project 2

Abstract

Use the HashMap class to support efficient linking of the classes used in Project 1. Implement comparators to support sorting. Extend the GUI from Project 1 to allow the user to sort by comparators.

Xavier DAVIS

Project 2
CMCS 335
February 4, 2018

Xavier Davis
CMSC 335
February 4, 2018

Release Notes:

- **Added data structure of a HashMap in World.java**
- **Search results is now displayed in a JTable, accompanied by column headers.**
    - **Table data can be sorted by clicking on the desired column header.**
    - **Enabled user-defined sorting ability.**
- **Search by indices has been deprecated**
- **Added search subject for Sea Ports. Search Sea Ports by name, returning list of ships in queue.**

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.
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
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
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

Xavier Davis
CMSC 335
February 4, 2018

Application Method List

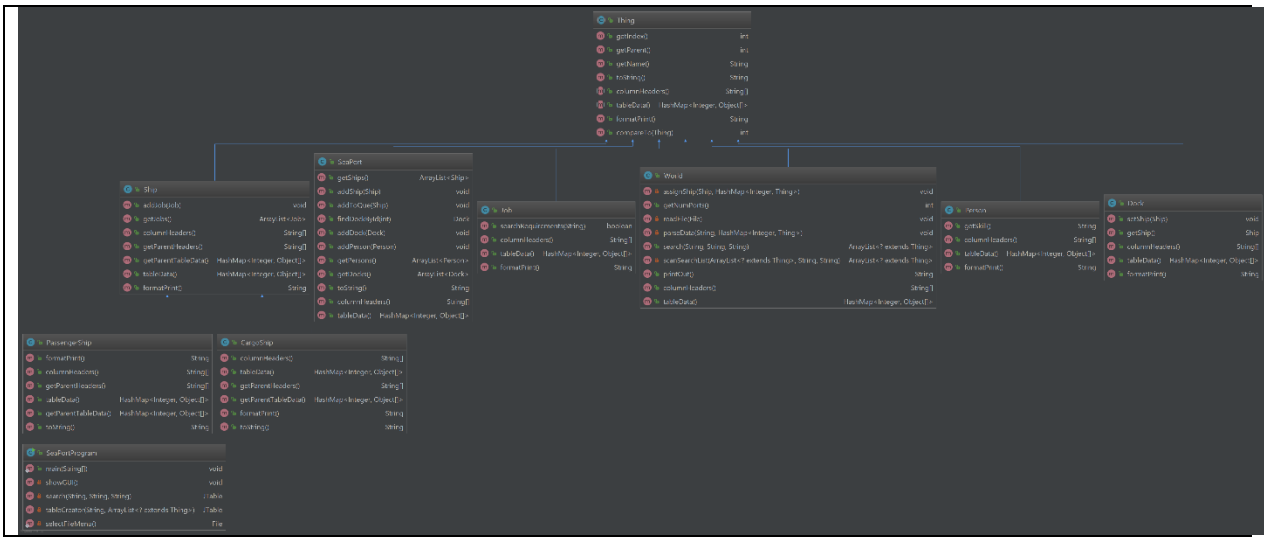
Return type	Method Signature	Description
SeaPortProgram.java		
void	showGUI ()	Constructs and displays the GUI interface
ArrayList<String>	search (String searchType, String searchAttribute, String searchKeyword)	Stores the returned search results from World.java into an ArrayList
File	selectFileMenu ()	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.
String	printout ()	Prints out the information parsed from the data file.
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		

Xavier Davis
CMSC 335
February 4, 2018

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	addDock (Dock dock)	Adds Dock to a SeaPort
void	addPerson (Person person)	Adds a passenger to a SeaPort
ArrayList<Person>	getPerson ()	Get a list of people assigned to a SeaPort
ArrayList<Dock>	getDocks ()	Returns a list of Docks within a SeaPort
String	toString ()	Overridden toString method
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

Xavier Davis
CMSC 335
February 4, 2018

UML Diagrams:



Fulfilling Project Requirements:

Usage of HashMap

Within the parse file method I implemented a HashMap to store all the world objects as they were created from the reading of the file. Key for the map is assigned to the index value of the object and the value of the map is the Sea Port object. To be able to sort all types of Thing objects in the map I assigned the values of the map with the syntax "? extends Thing". This way only objects that extend from the Thing class can be added to the map. This removed the restriction of only adding Ship objects but not people or other similar type exclusive restrictions. By implementing the HashMap I was able to completely eliminate the need for "find[OBJECT]ByIndex" methods in the World class. Since the keys of the map were the indices, I only had to call upon the get method of the map data structure. Once the file has been completely read, the Hash Map is release to the Garbage collector and released from memory.

Implement Comparators

To be able to sort by multiple class fields I would have to extend from the Comparator interface as opposed to the Comparable interface. The comparator interface allows sorting of the ships weight, length, draft, and all other fields specified in the documented requirements.

User enabled data sort

Once the comparators are established the user will be able to benefit from them when navigating the data table of search results. By simply clicking on the table column header titled ship's weight, all data within the column can be sorted in both ascending and descending order.

Xavier Davis
CMSC 335
February 4, 2018

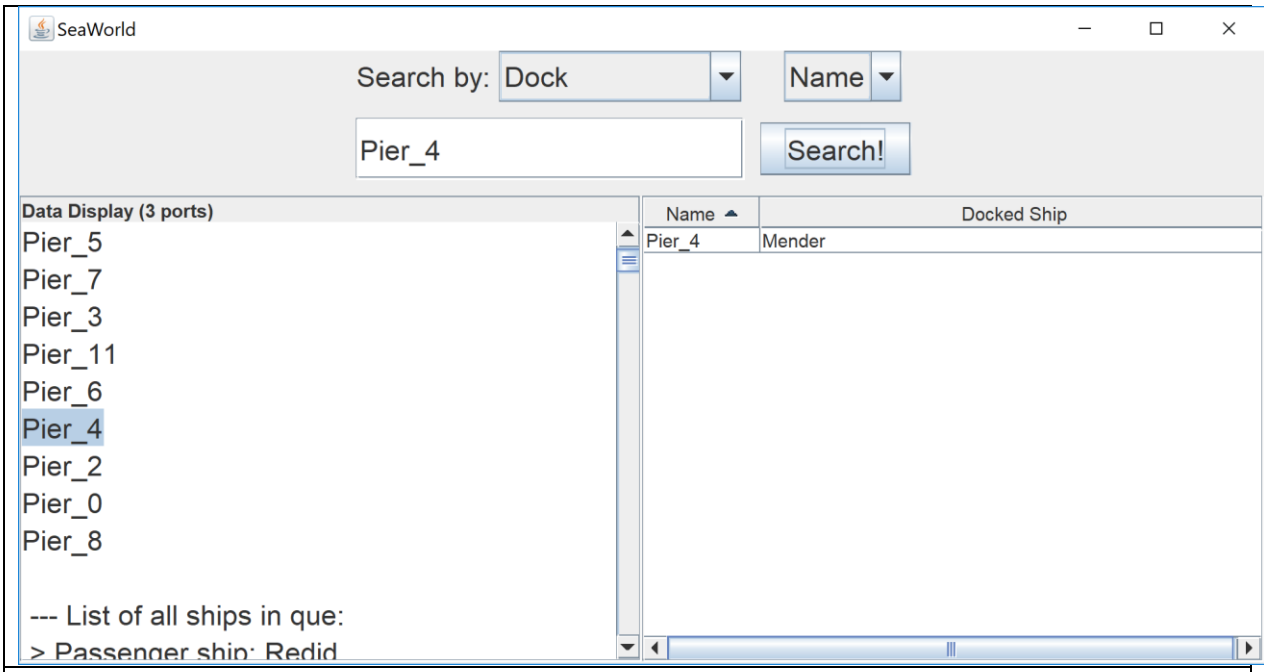
Test Plan:

Test Case	Input	Expected Output	Actual Output
Children assign to parent object without findByIndex methods	**Data File**	Identical data structures as project 1	**See Below**
Default sort objects by name	**Subject: Jobs** **Attribute: Skill** **Keyword: <Empty>**	All jobs without skill requirements displayed in ascending order.	**See Below**
Sort by ship object dimensions **Enable user sort operations**	**Subject: Ports** **Attribute: Name** **Keyword: Townsville**	A list of all ships in queue and each column is sortable by user click on column header.	**See Below**
Sort by Job duration	**Subject: Jobs** **Attribute: Skill** **Keyword: <Empty>**	All jobs without skill requirements displayed and can be sorted by duration by clicking on duration column.	**See Below**

Xavier Davis
CMSC 335
February 4, 2018

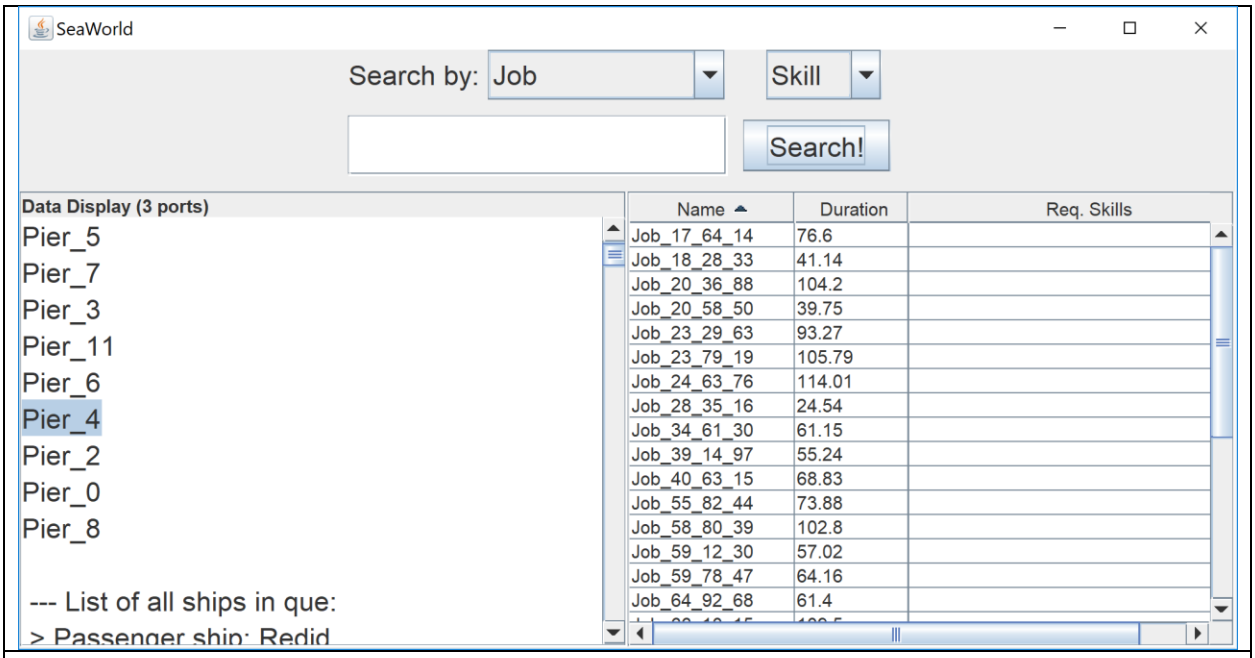
Test Case Screenshots:

Children assign to parent object without findByIndex methods



As shown the ship was still able to be linked to the dock object even without the findDockByID method.

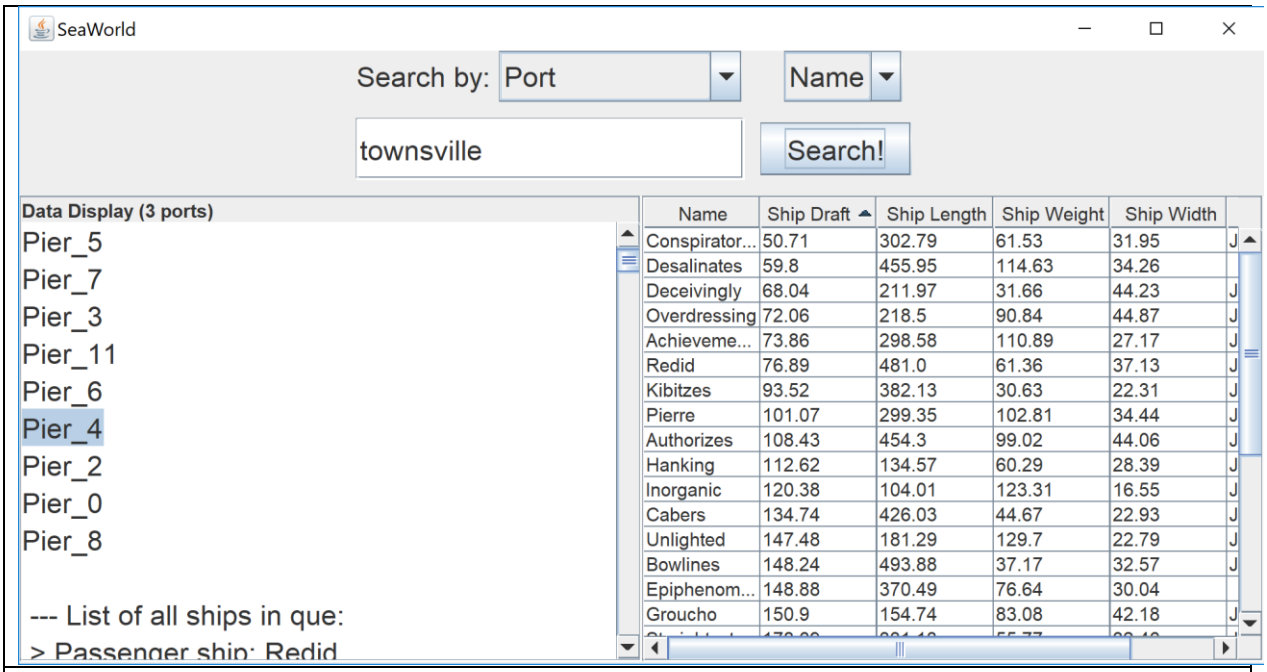
Default sort objects by name



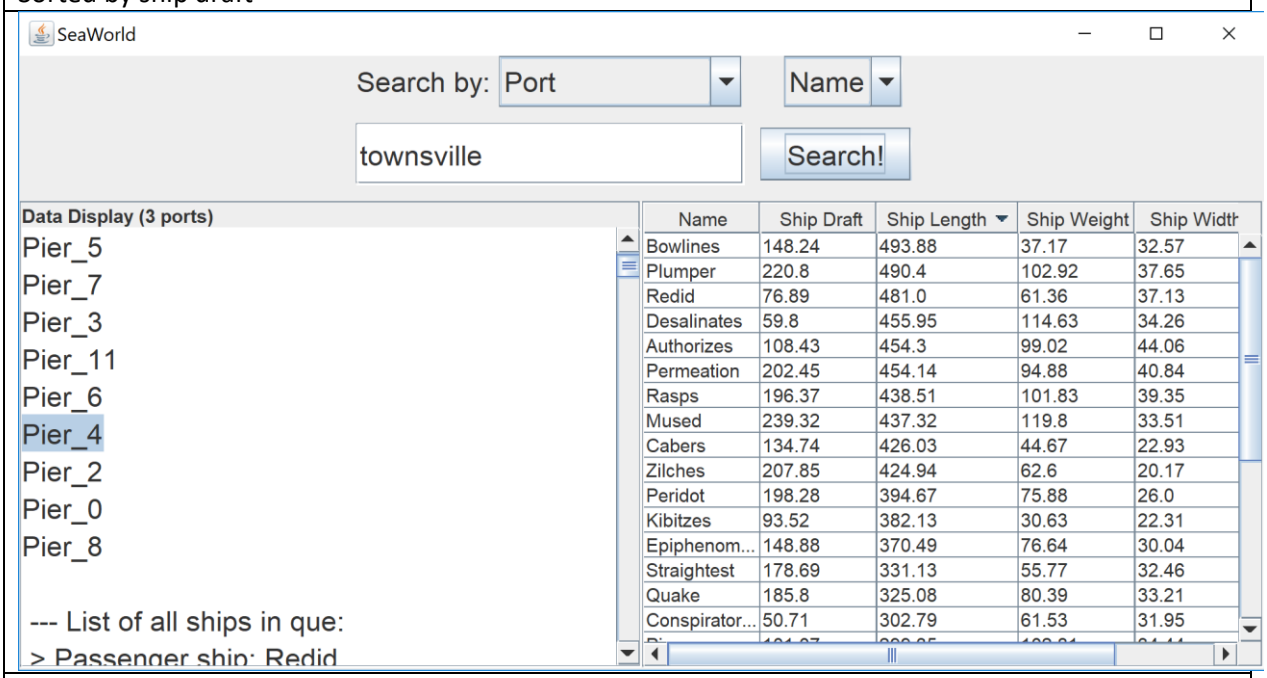
All job names are sorted by default in ascending order.

Xavier Davis
CMSC 335
February 4, 2018

Sort by ship object dimensions



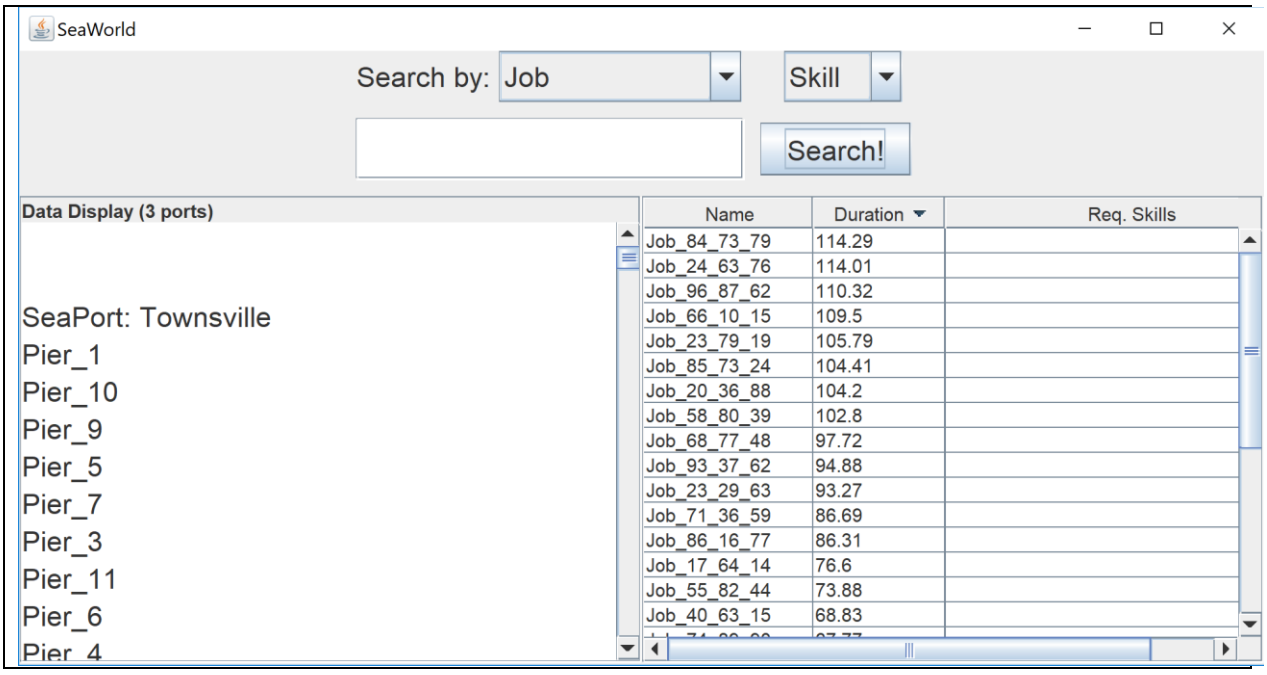
Sorted by ship draft



Sorted by ship's length

Xavier Davis
CMSC 335
February 4, 2018

Sort by Job Duration



Lessons Learned:

By having an understanding of the various data structures available to a programmer in the Java language, it can make varies search or sorting tasks easier. A HashMap object can be searched with better proficiency than compared to an Array or some list objects. By properly assigning meaningful key-value pairs, I was able to completely eliminate three class methods from World.java. This made my code for concise and readable with is two of the main goals for a developer.

Sea port application

User Application Guide

Xavier DAVIS

Project 2

CMSC 335

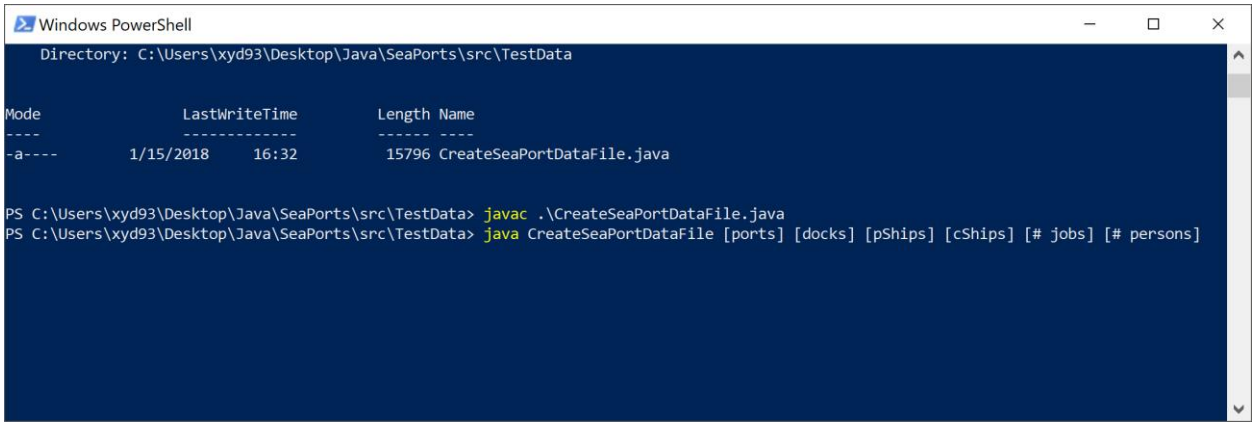
February 4, 2018

Xavier Davis
CMSC 335
February 4, 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.



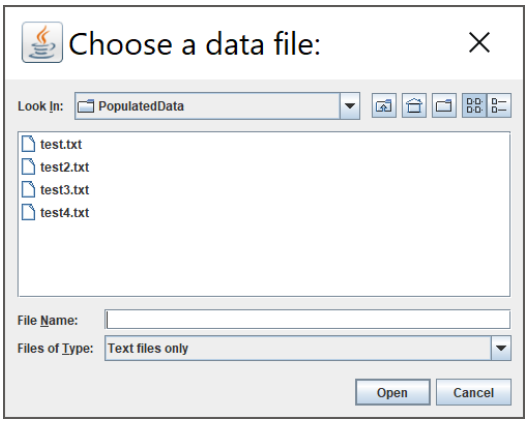
Launch application

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 4, 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, 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 4, 2018