SEA PORT APPLICATION

Project 1

Abstract

Read a data file, create the internal data structure, create a GUI to display the structure, and let the user search the structure.

Xavier DAVIS

Project 1

CMCS 335

January 21, 2018

Xavier Davis
CMSC 335
January 21, 2018

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. 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).

Xavier Davis
CMSC 335
January 21, 2018

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
ArrayList<String>	searchResults	ArrayList to store returned search results from application queries.
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

Xavier Davis
CMSC 335
January 21, 2018

ArrayList<Ship>	que	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

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		
Ship	getShipByIndex (int x)	Finds the ship by index number
Dock	getDockByIndex (int dockIndex)	Finds a Dock by index number
SeaPort	getSeaPortByIndex (int index)	Finds a SeaPort by index number

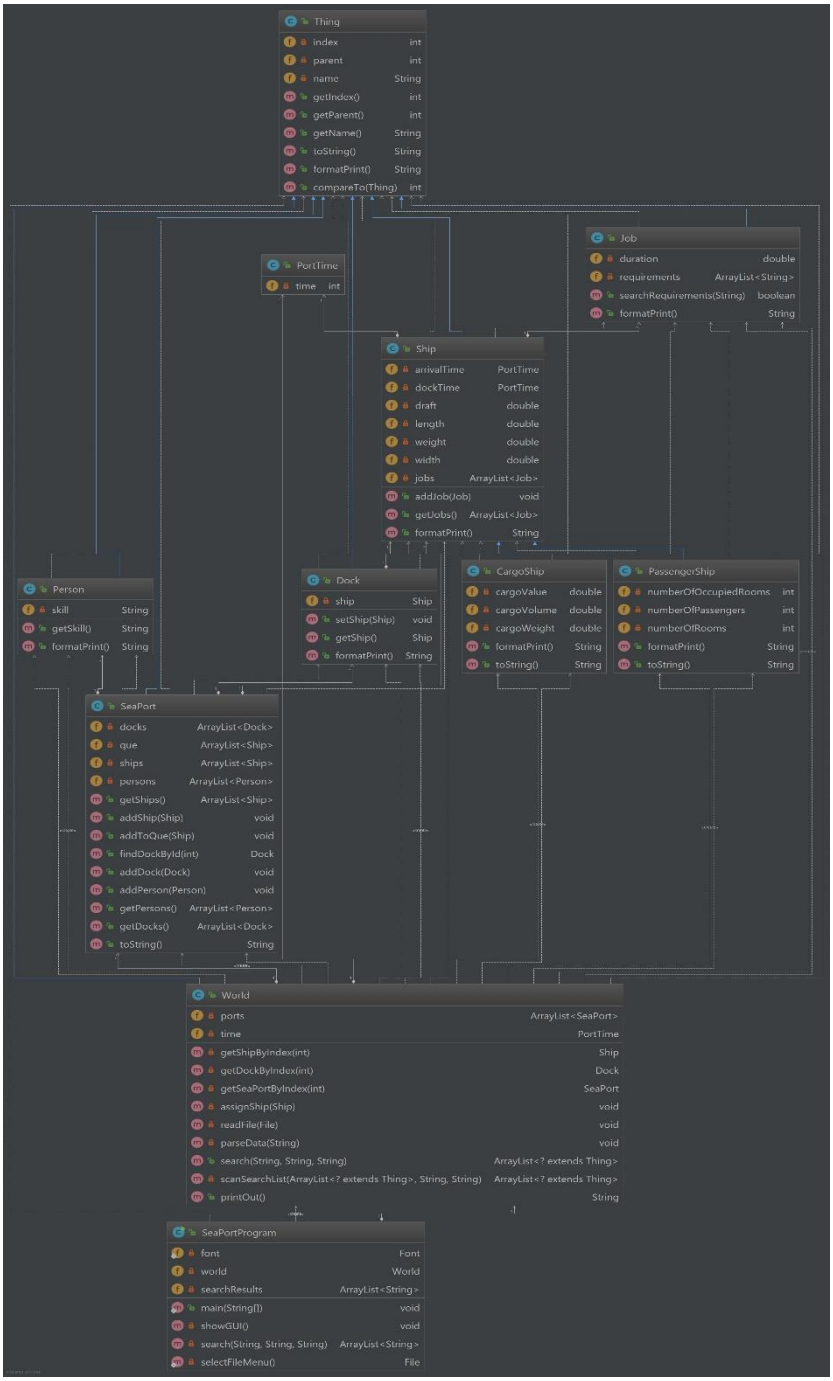
void	assignShip (Ship ship)	Assigns a Ship object to a Dock object
void	readFile (File file)	Reads the data file selected from the selectFileMenu method
void	parseData (String line)	Creates the appropriate object based on the current line from data file.
ArrayList<? Extends Thing>	search (String subject, 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
Ship.java		
void	addJob (Job job)	Adds a job to a Ship object
ArrayList<Job>	getJobs ()	Returns list of Job attached to Ship object
String	formatPrint ()	Returns formatted String of matching search results
SeaPort.java		
ArrayList<Ship>	getShips ()	Returns a list of Ships in a SeaPort
void	addShip (Ship ship)	Adds a Ship to a SeaPort
void	addToQue (Ship ship)	Adds a Ship to a SeaPort's que
Dock	findDockById (int id)	Finds a Dock by index id
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

Xavier Davis
CMSC 335
January 21, 2018

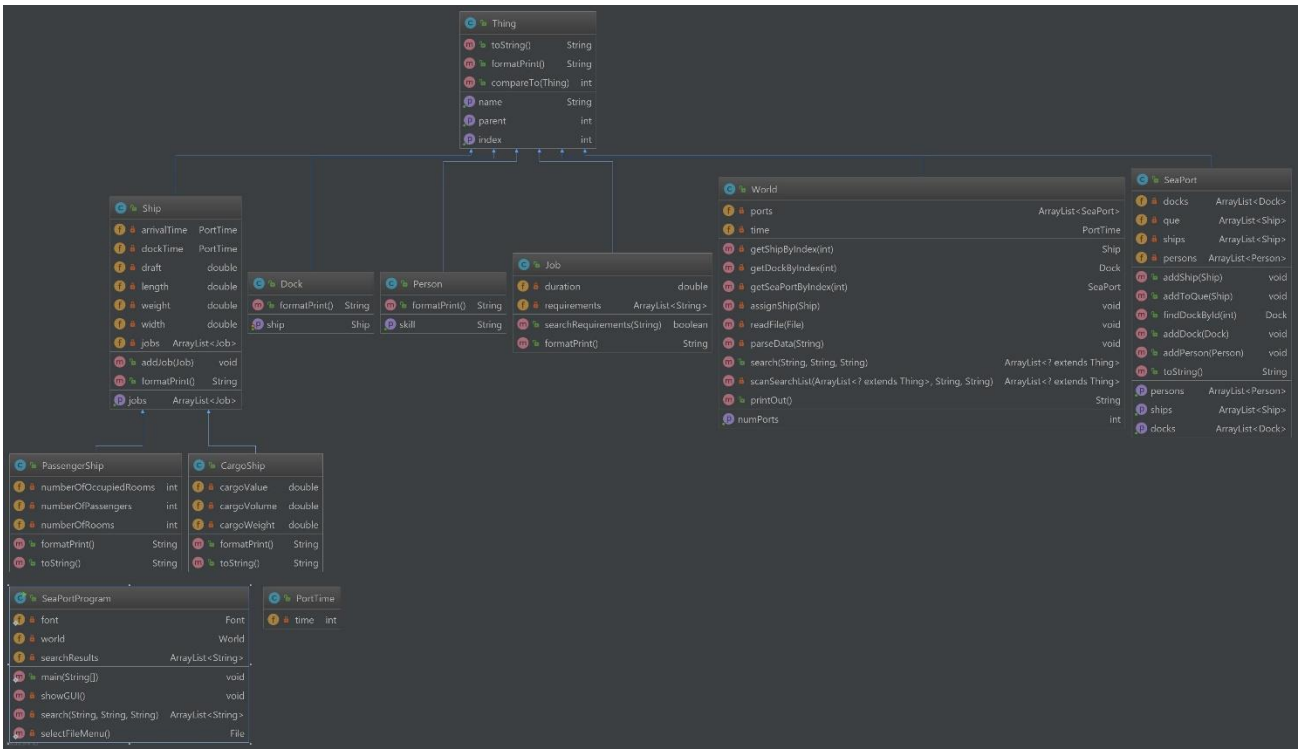
Dock.java		
void	setShip (Ship ship)	Sets the Ship object assigned to a Dock
Ship	getShip ()	Returns the Ship assigned to a Dock

Xavier Davis
CMSC 335
January 21, 2018

UML Diagrams:



Xavier Davis
CMSC 335
January 21, 2018



Fulfilling Project Requirements:

Define and implement appropriate classes:

Using the class information provided in the project details I constructed a new java class for each of the objects need per the requirements. Processing down the list as provide in the documentation I created the parent class Thing then proceeded to create all the child classes to eliminate an error messages from the IDE. All constructors for the class accepts a single parameter of a Scanner object to continue to parse through the data file line to extract the needed information for object instance creation. For security all class variables are created as private variables. Each object overrides the toString method to return the approperate class information, formatted with new-line and tab escape characters. The toString method is called in the World.java class and passed to the GUI in seaPortProgram.java. After all the object classes were created I came back to the top of the list to create the seaPortProgram class for the GUI construction.

Read data from a text file:

Within World.java I constructed a parseData method that is responsible for the reading of the current file line and determining the correct class instance to create using the first word of the line and a switch statement. The first line of the data file dictates what sub-class of Things needs to be created. As each instance is created I utilize some of the provided methods from the project explanation file to properly link the object to its parent element (i.e. ship to dock, person to sea port, etc.).

Xavier Davis
CMSC 335
January 21, 2018

Create a simple GUI:

The seaPortProgram class is responsible for the construction of the GUI interface. The class is a subclass of the JFrame java class as required by the project documentation. The interface is simple but includes the required elements of a search field and the search modifiers to alter the search subject. An action listener is added to the search JButton, once the button is clicked the search modifiers and search query is gathered and passed to the search method for system query. A layout of the GUI is attached in the User Guide documentation.

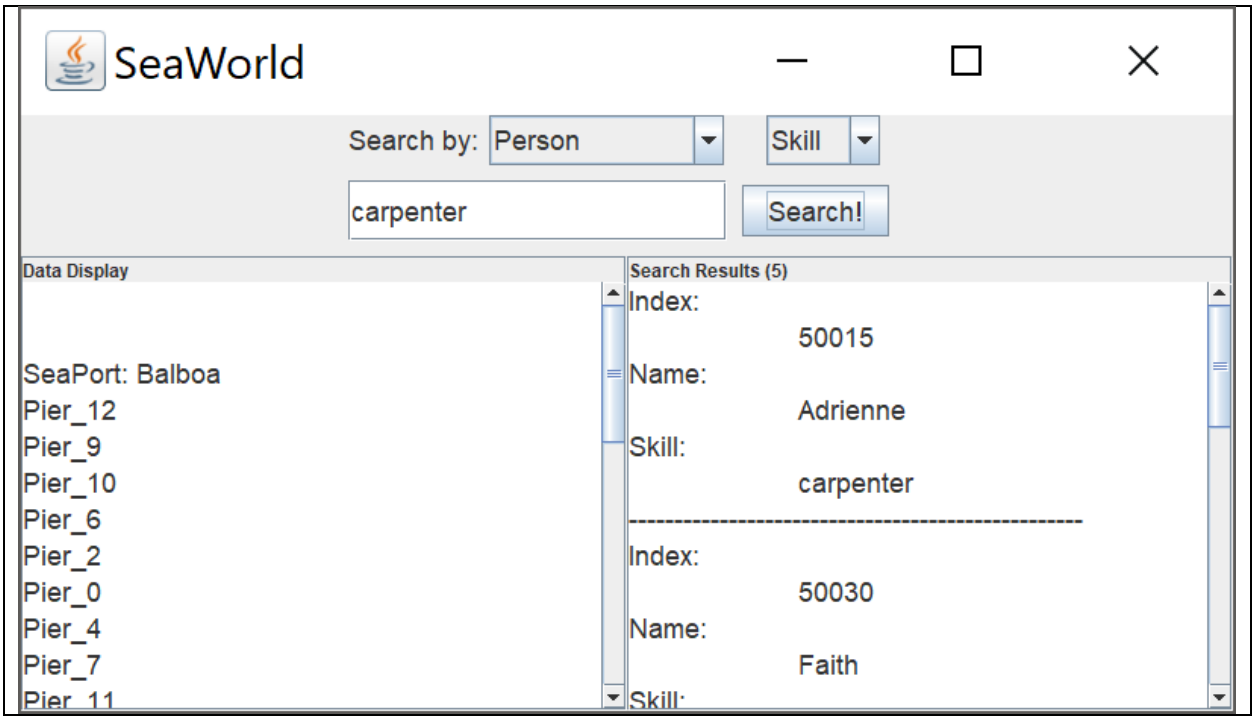
Test Plan:

Test Case	Input	Expected Output	Actual Output
Find person by skill	**Search: Person** **Attribute: skill** **Keyword: "carpenter"**	Result list of all people with skill Sorted by name ascending order	**See Below**
Blank search entry	**Search: Passenger Ship** **Attribute: Name** **Keyword: <blank>**	Flag an error for no results	**See Below**
Non-numeric index	**Search: Cargo Ship** **Attribute: Index** **Keyword: "word"**	Flag an error input string instead number	**See Below**
Search for jobs without requirements	**Search: Job** **Attribute: Skill** **Keyword: {<blank> || "none"}**	List of all jobs with no job requirements	**See Below**
Find dock by index	**Search: Dock** **Attribute: Index** **Keyword: {correct index #}**	Return the dock information for inputted index	**See Below**
Find job by name	**Search: Job** **Attribute: Name** **Keyword: {correct job title}**	Return job information matching name	**See Below**
Displaying skill attribute based on search subject	**Search: Cargo Ship**	Attribute list: {Index, Name}	**See Below**

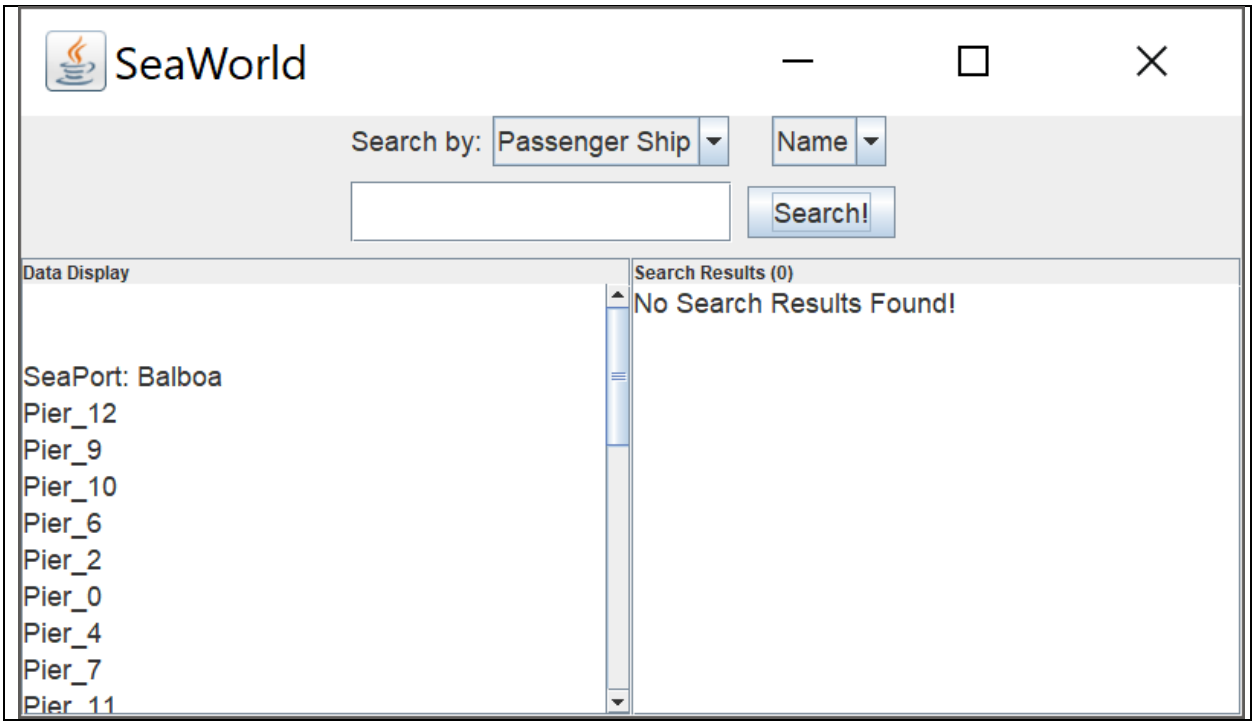
Xavier Davis
CMSC 335
January 21, 2018

Test Case Screenshots:

Find person by skill

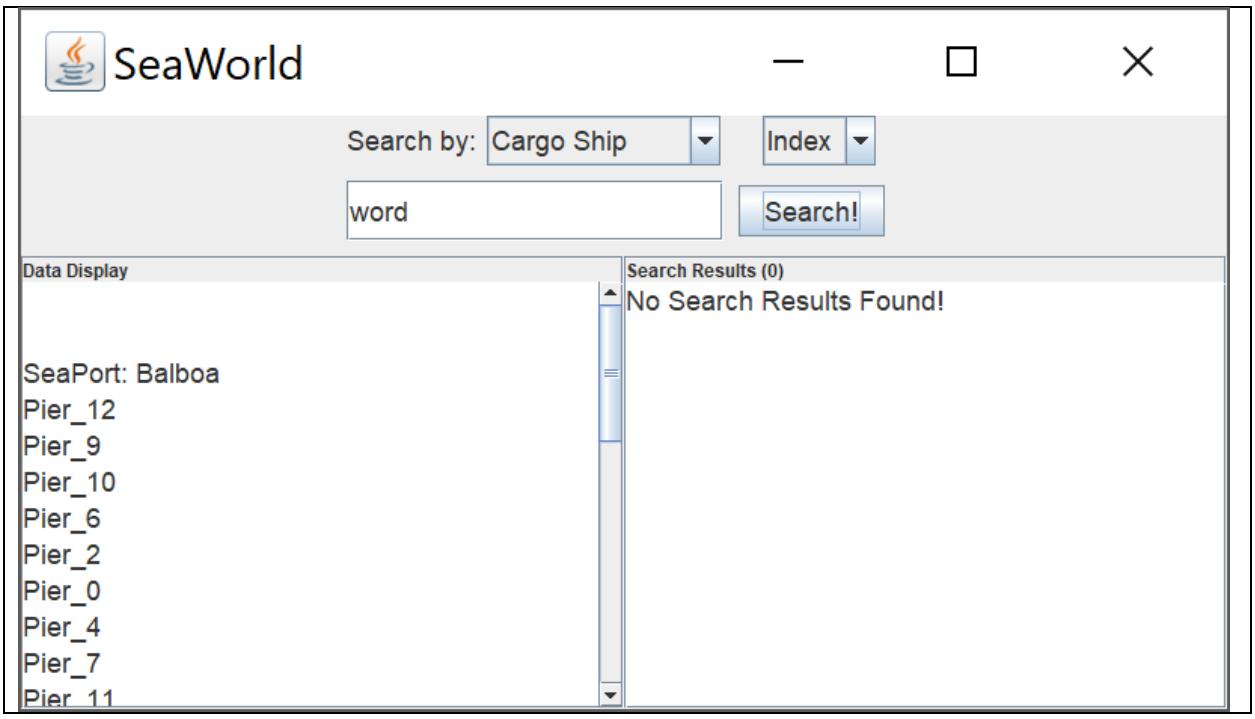


Blank search entry



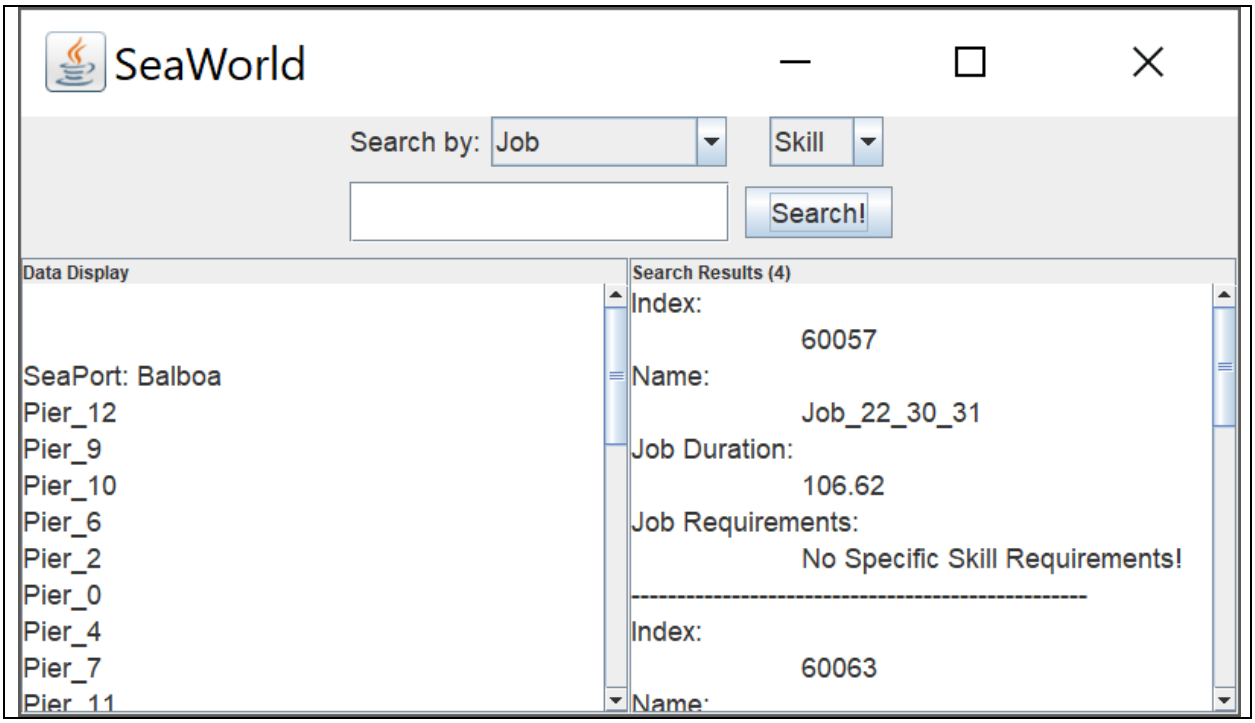
Xavier Davis
CMSC 335
January 21, 2018

Non-numeric index

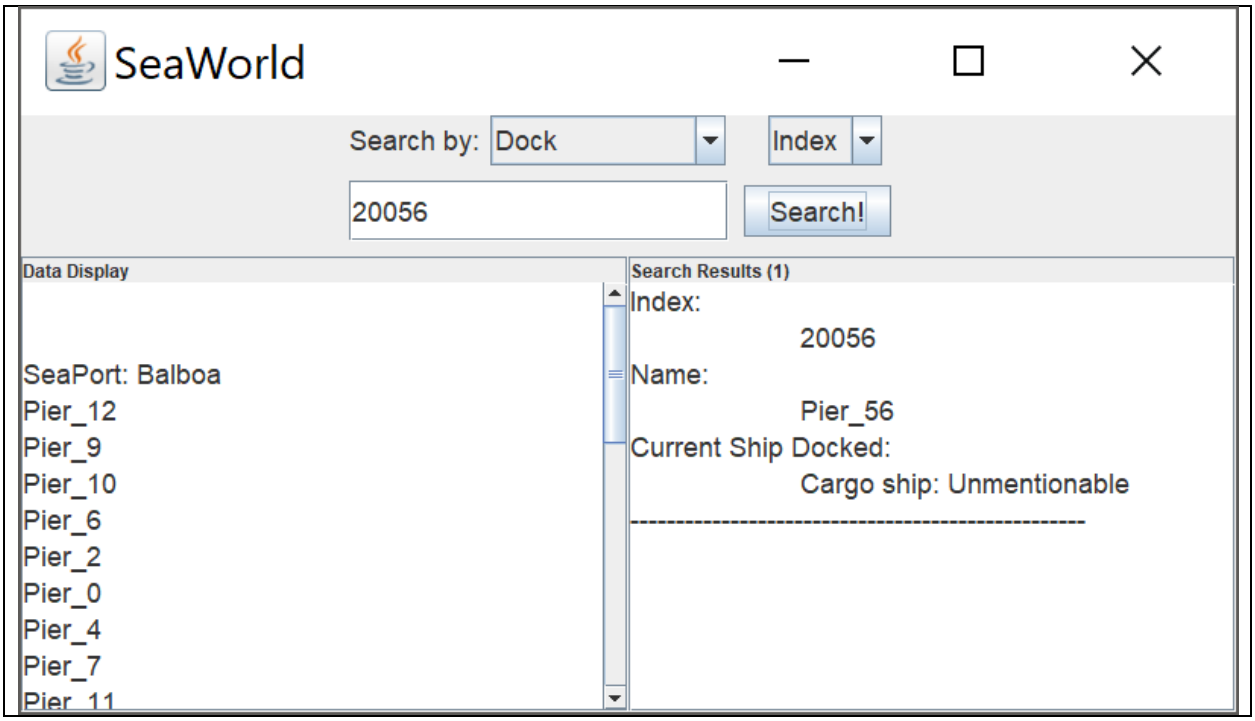


Xavier Davis
CMSC 335
January 21, 2018

Search for jobs without requirements

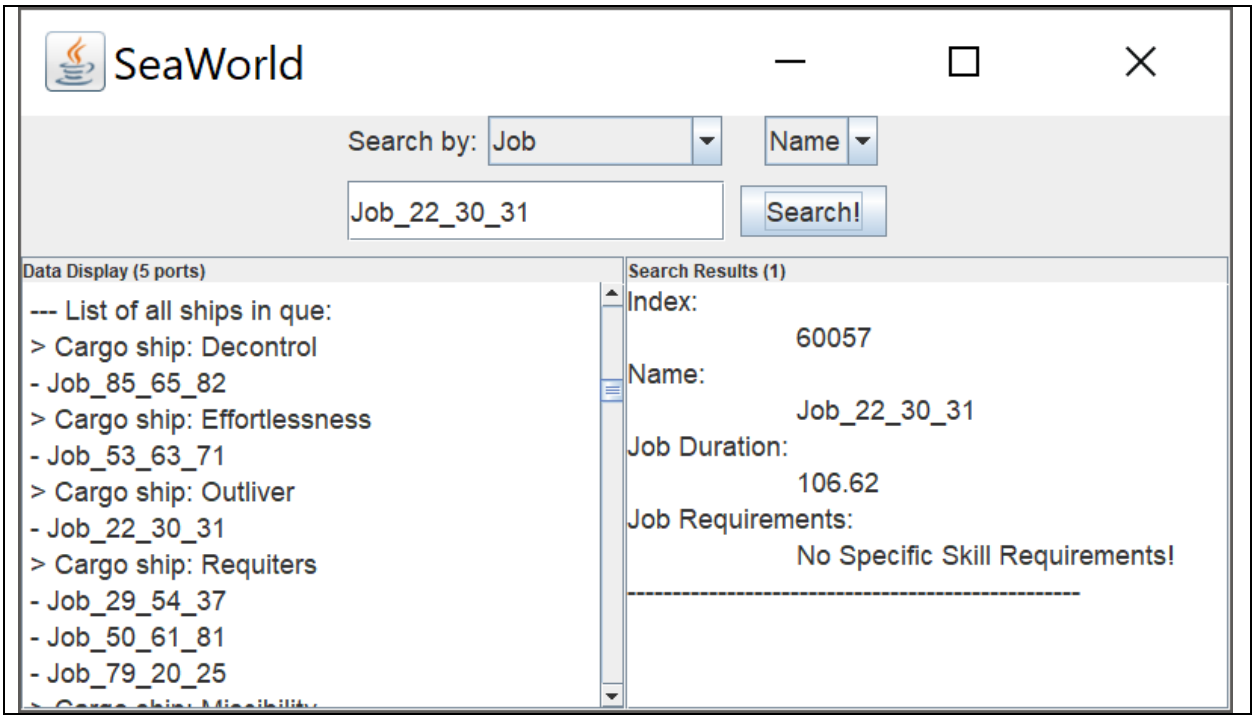


Find dock by index



Xavier Davis
CMSC 335
January 21, 2018

Find job by name



Lessons Learned:

Through this process I learned the value of compartmentalizing an application's source code. Through keeping source code separated it makes the application easier to enhance and modify in the future. Also, the debugging process is a bit less stressful as each established class has a set purpose and functionality, no need to search through lines of code within one class to find an issue. The importance of error handling is very apparent in this application since the software replies heavily on user input. A great deal of testing needs to be accomplished to cover all possible input values the application by encounter from a user. Programmers should not expect the user to abide by the syntax rules of the application.

Sea port application

User Application Guide

Xavier DAVIS

Project 1

CMSC 335

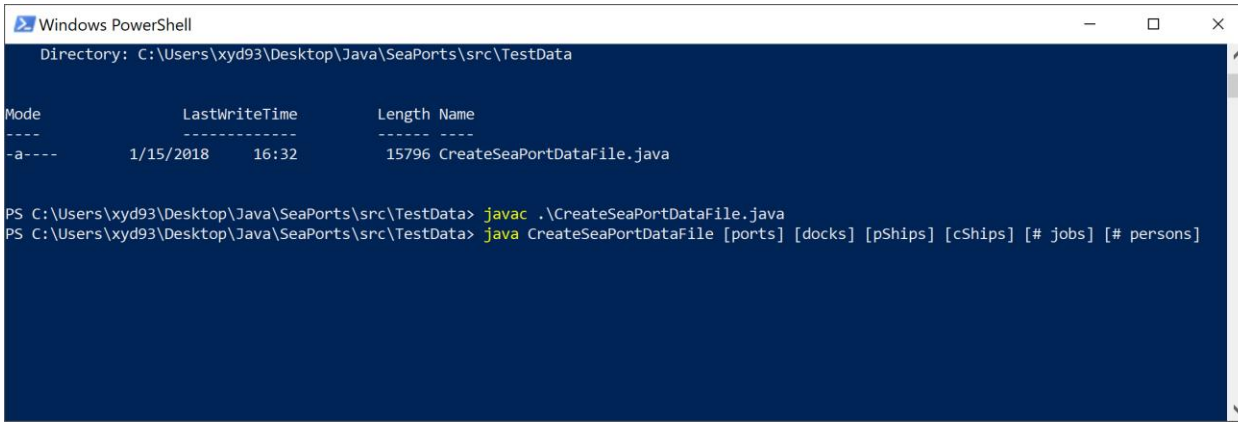
January 21, 2018

Xavier Davis
CMSC 335
January 21, 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.



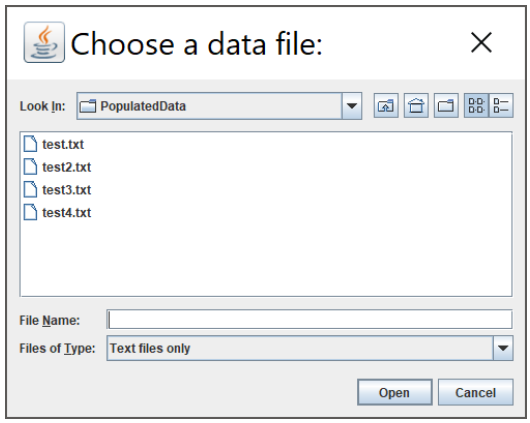
Launch application

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
January 21, 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 index, 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. 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.