

Project 1: Sea Port Program

Justin VanWinkle^{1,2}

¹ University of Maryland University College

² Wycliffe Associates, Inc.

Author Note

Correspondence concerning this article should be addressed to Justin VanWinkle, 281
4th St, Geneva, FL. E-mail: justin@swissarmydev.com

Abstract

This paper is dedicated to a project developed in a course on object-oriented programming and concurrency. It explores the decisions made while designing the program, the way a user should go about making use of the program, standard practices for testing the program, lessons learned while

Keywords: java,object-oriented,

Word count: X

Project 1: Sea Port Program

Design

From a high level, this program has been built with a class that contains the UI separate from any workflow logic. This paves the way for several simplistic approaches when concurrency is included into the program in a future iteration. One such example would be creating the world using a thread that is separate from that of the Event Dispatch Thread that would be used by default when calling a function from an event listener that is attached to a swing component. Such a separation becomes easily manageable when business logic and view logic remain isolated.

Decisions

- It was determined that all classes that are representative of objects in the input file would not instantiate a scanner so as to prevent repetitive code across classes and also to prevent each class from needing to know how its definition is formed in the input file. This loosens the coupling in such a way that eases changes as a result of changes in the structure of the input file.

UML Diagram

Diagram

Meanings

The various classes, packages, variables, and methods in this program are organized in a logical manner that show good practices in object-oriented design. Each of the items has been placed in such a way that it has some sort of direct relationship with its containing parent or its contained child.

Classes. All classes assume no knowledge of their calling class whether by passing context or interface. This is likely to change in a future iteration, however.

The Thing Package. Each class in the *thing* package as well as classes in the sub packages of *thing* represent real world objects that reflect their name.

Other Classes. The PortTime class, in a similar manner represents the time of a port. Due to ambiguous requirements, however, the developer was unable to determine the intended use of this class. Therefore, it is assumed that this class will be better integrated in a future iteration.

Variables. In each class, there are sets of instance variables that represent a *has-a* relationship between the class and the item that variable holds. For instance, a SeaPort has a (or many) Dock(s).

Methods. The SeaPortProgram class contains the heart of creating the world including all of the logic for parsing out the input file, creating the objects, and modeling a world that represents what is defined in the input file. While the requirement documentation *suggests* that a scanner should be used, a string tokenizer was used instead. While this adds some additional coding overhead,

Project Requirements

User's Guide

How to start the program

How to create a world

1. Click "Select File".
2. Navigate to and select a valid input file.
3. Select "Open".
4. World is automatically modeled. Notice the output that matches the data in the chosen input file.

Search

Searches using regex allow for more control over the matching pattern. As a result, searches are made to iterate over the entire set of objects that exist in the world.

How to search.

1. Input a valid java regex engine pattern in the search box. (Valid regex patterns for the java regex engine can be obtained [Here](#).
2. Click “Search”. Searching returns a list of all *Things* that have a Name or Index that match the criteria. Some additional cases exist such as searching the skills of a Person or the duration of a Job.

Examples.

- Search Pattern: “ar”
- Matches “Sara”
- Does not Match “Archie”
- Search Pattern: “[A|a]r”
- Matches “Sara” and “Archie” For more extended documentation on regex patterns, [go here](#)

Special Features

Test Plan

Lessons Learned

About This Documentation

Since discovering Pandoc, I have written all of my professional documentation and papers in markdown and used Pandoc to convert them to whatever end format I require. Markdown allows me to focus on content rather than formatting and thus saves great amounts of time. One particular hurdle in writing this documentation was finding a way to use L^AT_EX

New Classes Used

While I purposefully avoid Java's built in UI libraries, it came as no surprise when I found myself using the JScrollPane class for the first time in this program. I did find JScrollPane to be comparatively easy to work with for the sake of just getting a scroll view set up.

Future Iterations