Project 4

Introduction - the SeaPort Project series

For this set of projects for the course, we wish to simulate some of the aspects of a number of Sea Ports.

Here are the classes and their instance variables we wish to define:

- SeaPortProgram extends JFrame
    - variables used by the GUI interface
    - world: World
- Thing implement Comparable <Thing>
    - index: int
    - name: String
    - parent: int
- World extends Thing
    - ports: ArrayList <SeaPort>
    - time: PortTime
- SeaPort extends Thing
    - docks: ArrayList <Dock>
    - que: ArrayList <Ship> // the list of ships waiting to dock
    - ships: ArrayList <Ship> // a list of all the ships at this port
    - persons: ArrayList <Person> // people with skills at this port
- Dock extends Thing
    - ship: Ship
- Ship extends Thing
    - arrivalTime, dockTime: PortTime
    - draft, length, weight, width: double
    - jobs: ArrayList <Job>
- PassengerShip extends Ship
    - numberOfOccupiedRooms: int
    - numberOfPassengers: int
    - numberOfRooms: int
- CargoShip extends Ship
    - cargoValue: double
    - cargoVolume: double
    - cargoWeight: double
- Person extends Thing
    - skill: String
- Job extends Thing - optional till Projects 3 and 4
    - duration: double
    - requirements: ArrayList <String>
      // should be some of the skills of the persons
- PortTime
    - time: int

Eventually, in Projects 3 and 4, you will be asked to show the progress of the jobs using JProgressBar's.

Here's a very quick overview of all projects:

1. Read a data file, create the internal data structure, create a GUI to display the structure, and let the user search the structure.
2. Sort the structure, use hash maps to create the structure more efficiently.
3. Create a thread for each job, cannot run until a ship has a dock, create a GUI to show the progress of each job.
4. Simulate competing for resources (persons with particular skills) for each job.

Project 4 General Objectives

Project 4 - Concurrency

- Resource pools
    - Threads competing for multiple resources
- Blocking threads
- Extending the GUI interface to visualize the resource pools and progress of the various threads.

Documentation Requirements:

You should start working on a documentation file before you do anything else with these projects, and fill in items as you go along. Leaving the documentation until the project is finished is not a good idea for any number of reasons.

The documentation should include the following (graded) elements:

- Cover page (including name, date, project, your class information)
- Design
    - including a UML class diagram
    - classes, variables and methods: what they mean and why they are there
    - tied to the requirements of the project
- User's Guide
    - how would a user start and run your project
    - any special features
    - effective screen shots are welcome, but don't overdo this
- Test Plan
    - do this BEFORE you code anything
    - what do you EXPECT the project to do
    - justification for various data files, for example
- Lessons Learned
    - express yourself here
    - a way to keep good memories of successes after hard work

Project 4 Specific Goals:

Extend project 3 to include making jobs wait until people with the resources required by the job are available at the port.

Elaboration:

1. Reading Job specifications from a data file and adding the required resources to each Job instance.
2. Resource pools - SeaPort.ArrayList <Person> list of persons with particular skills at each port, treated as resource pools, along with supporting assignment to ships and jobs.
3. Job threads - using the resource pools and supporting the concept of blocking until required resources are available before proceeding.
4. The Job threads should be efficient:
    1. If the ship is at a dock and all the people with required skills are available, the job should start.
    2. Otherwise, the Job should not hold any resources if it cannot progress.
    3. Use synchronization to avoid race conditions.
    4. Each Job thread should hold any required synchronization locks for a very short period.
    5. When a job is over, all the resources used by the job (the people) should be released back to the port.
    6. When all the jobs of a ship are done, the ship should depart the dock and if there are any ships in the port que, one of then should should be assigned to the free dock, and that ships jobs can now try to progress.
    7. NOTE: If a job can never progress because the port doesn't have enough skills among all the persons at the port, the program should report this and cancel the job.
5. GUI showing:
    - Resources in pools - how many people with skill are currently available
    - Thread progress, resources acquired, and resources requests still outstanding

Deliverables:

1. Java source code files
2. Data files used to test your program
3. Configuration files used
4. A well-written document including the following sections:
    a. Design: including a UML class diagram showing the type of the class relationships
    b. User's Guide: description of how to set up and run your application
    c. Test Plan: sample input and ***expected*** results, and including test data and results, with screen snapshots of some of your test cases
    d. Optionally, Comments: design strengths and limitations, and suggestions for future improvement and alternative approaches
    e. Lessons Learned
    f. Use one of the following formats: MS Word docx or PDF.

Your project is due by midnight, EST, on the day of the date posted in the class schedule. We do not recommend staying up all night working on your project - it is so very easy to really mess up a project at the last minute by working when one was overly tired.

Your instructor's policy on late projects applies to this project.

Submitted projects that show evidence of plagiarism will be handled in accordance with UMUC Policy 150.25 — Academic Dishonesty and Plagiarism.

Format:

The documentation describing and reflecting on your design and approach should be written using Microsoft Word or PDF, and should be of reasonable length. The font size should be 12 point. The page margins should be one inch. The paragraphs should be double spaced. All figures, tables, equations, and references should be properly labeled and formatted using APA style.

Coding Hints:

- Code format: (See Google Java Style guide for specifics (https://google.github.io/styleguide/javaguide.html))
    - header comment block, including the following information in each source code file:
    - file name
    - date
    - author
    - purpose
    - appropriate comments within the code
    - appropriate variable and function names
    - correct indentation
- Errors:
    - code submitted should have no compilation or run-time errors
- Warnings:
    - Your program should have no warnings
    - Use the following compiler flag to show all warnings:
      javac -Xlint *.java
    - More about setting up IDE's to show warnings
    - Generics - your code should use generic declarations appropriately, and to eliminate all warnings
- Elegance:
    - just the right amount of code
    - effective use of existing classes in the JDK
    - effective use of the class hierarchy, including features related to polymorphism.
- GUI notes:
    - GUI should resize nicely
    - DO NOT use the GUI editor/generators in an IDE (integrated development environment, such as Netbeans and Eclipse)
    - Do use JPanel, JFrame, JTextArea, JTextField, JButton, JLabel, JScrollPane
        - panels on panels gives even more control of the display during resizing
        - JTable and/or JTree for Projects 2, 3 and 4

        - Font using the following gives a nicer display for this program, setting for the JTextArea jta:

          ```
          jta.setFont (new java.awt.Font ("Monospaced", 0, 12));
          ```
    - GridLayout and BorderLayout - FlowLayout rarely resizes nicely
        - GridBagLayout for extreme control over the displays
        - you may wish to explore other layout managers
    - ActionListener, ActionEvent - responding to JButton events
        - Starting with JDK 8, lambda expression make defining listeners MUCH simpler. See the example below, with jbr (read), jbd (display) and jbs (search) three different JButtons.
          jcb is a JComboBox <String> and jtf is a JTextField.

          ```
             jbr.addActionListener (e -> readFile());
             jbd.addActionListener (e -> displayCave ());
             jbs.addActionListener (e -> search ((String)(jcb.getSelectedItem()),
          jtf.getText()));
          ```
    - JFileChooser - select data file at run time
    - JSplitPane - optional, but gives user even more control over display panels

Grading Rubric:

Attribute	**Meets**	**Does not meet**
Design	**20 points** Contains just the right amount of code. Uses existing classes in the JDK effectively. Effectively uses of the class hierarchy, including features related to polymorphism. GUI elements should be distinct from the other classes in the program.	**0 points** Does not contain just the right amount of code. Does not use existing classes in the JDK effectively. Does not effectively use of the class hierarchy, including features related to polymorphism. GUI elements are not distinct from the other classes in the program.
Functionality	**40 points** Contains no coding errors. Contains no compile warnings. Builds from previous projects. Includes reading Job specifications from a data file and adding the	**0 points** Contains coding errors. Contains compile warnings. Does not build from previous projects. Does not include reading Job specifications from a data file and

	required resources to each Job instance. Includes resource pools - SeaPort.ArrayList <Person> list of persons with particular skills at each port, treated as resource pools, along with supporting assignment to ships and jobs. Includes job threads - using the resource pools and supporting the concept of blocking until required resources are available before proceeding. The Job threads should be efficient. GUI shows resources in pools - how many people with skill are currently available and thread progress, resources acquired, and resources requests still outstanding.	adding the required resources to each Job instance. Does not include resource pools - SeaPort.ArrayList <Person> list of persons with particular skills at each port, treated as resource pools, along with supporting assignment to ships and jobs. Does not include job threads - using the resource pools and supporting the concept of blocking until required resources are available before proceeding. The Job threads are not efficient. GUI does not show resources in pools - how many people with skill are currently available and thread progress, resources acquired, and resources requests still outstanding.
Test Data	**20 points** Tests the application using multiple and varied test cases.	**0 points** Does not test the application using multiple and varied test cases.
Documentation and submission	**15 points** Source code files include header comment block, including file name, date, author, purpose, appropriate comments within the code, appropriate variable and function names, correct indentation. Submission includes Java source code files, Data files used to test your program, Configuration files used. Documentation includes a UML class diagram showing the type of the class relationships.	**0 points** Source code files do not include header comment block, or include file name, date, author, purpose, appropriate comments within the code, appropriate variable and function names, correct indentation. Submission does not include Java source code files, Data files used to test your program, Configuration files used. Documentation does not include a UML class diagram showing the type of the class relationships. Documentation does not include a user's Guide describing of how to set up and run your application.

	Documentation includes a user's Guide describing of how to set up and run your application. Documentation includes a test plan with sample input and ***expected*** results, test data and results and screen snapshots of some of your test cases. Documentation includes Lessons learned. Documentation is in an acceptable format.	Documentation does not include a test plan with sample input and ***expected*** results, test data and results and screen snapshots of some of your test cases. Documentation does not include Lessons learned. Documentation is not in an acceptable format.
Documentation form, grammar and spelling	**5 points** Document is well-organized. The font size should be 12 point. The page margins should be one inch. The paragraphs should be double spaced. All figures, tables, equations, and references should be properly labeled and formatted using APA style. The document should contain minimal spelling and grammatical errors.	**0 points** Document is not well-organized. The font size is not 12 point. The page margins are not one inch. The paragraphs are not double spaced. All figures, tables, equations, and references are not properly labeled or formatted using APA style. The document should contains many spelling and grammatical errors.