Project 1

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 1 General Objectives

Project 1 - classes, text data file, GUI, searching

- Define and implement appropriate classes, including:
    - instance and class variables,
    - constructors,
    - toString methods, and
    - other appropriate methods.
- Read data from a text file:
    - specified at run time,
        - JFileChooser jfc = new JFileChooser (".");
          // start at dot, the current directory
    - using that data to create instances of the classes,
    - creating a multi-tree (class instances related in hierarchical, has-some, relationships), and
    - organizing those instances in existing JDK structures which can be sorted, such as ArrayList's.
- Create a simple GUI:
    - presenting the data in the structures with with some buttons and
    - text fields supporting SEARCHING on the various fields of each class.

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 1 Specific Goals:

1. Create a GUI
2. Let the user select a data file, using JFileChooser
3. Read the data file, creating the specified internal data structure (see the Introduction for the classes and variables of the structure).
4. Display the internal data structure in a nice format in the GUI
    1. use JScrollPane and JTextArea
5. Display the results of a Search specified by the user
    1. JTextField to specify the search target
    2. Searching targets: name, index, skill would be a minimum
       you are encouraged to provide other options
    3. Note that a search may return more than one item
    4. DO NOT create new data structures (beyond the specified internal data structure) to search
       you may create a structure of found items as a return value

Data file format:

- Each item in the simulation will appear on a single line of the data file.
- The data file may have comment lines, which will start with a //.
- There may be blank lines in the data file, which your program should accept and ignore.
- The data lines will start with one of the following flag values, indicating which item is being specified, its name, its index, and the index of its parent - which is used to specify the connections used to create the internal data structure (ie, assign an item to it parent or parent ArrayList).
- For most items there will be additional fields appropriate to the class of that item.
- The fields on a line are space delimited (perhaps more than one space)
    - This works well with Scanner methods, such as next(), nextInt() and nextDouble().

Here are the details of valid lines, with an example of each line.

```
// port   name index parent(null)
//    port   <string> <int> <int>
port Kandahar 10002 0
```

```
// dock   name index parent(port)
//    dock   <string> <int> <int>
  dock Pier_5 20005 10001 30005

// ship   name index parent(dock/port) weight length width draft
//    ship   <string> <int> <int> <double> <double> <double> <double>
    ship             Reason 40003 10000 165.91 447.15 85.83 27.07

// cship  name index parent(dock/port) weight length width draft cargoWeight cargoVolume cargoValue
//    cship  <string> <int> <int> <double> <double> <double> <double> <double> <double> <double>
    cship             Suites 40003 10000 165.91 447.15 85.83 27.07 125.09 176.80 857.43

// pship  name index parent(dock/port) weight length width draft numPassengers numRooms
numOccupied
//    pship  <string> <int> <int> <double> <double> <double> <double> <int> <int> <int>
    pship     "ZZZ_Hysterics" 30002 20002 103.71 327.92 56.43 30.23 3212 917 917

// person name index parent skill
//    person <string> <int> <int> <string>
    person           Alberto 50013 10001 cleaner

// job    name index parent duration [skill]* (zero or more, matches skill in person, may repeat)
//    job    <string> <int> <int> <double> [<string>]* (ie, zero or more)
    job       Job_10_94_27 60020 30007 77.78 carpenter cleaner clerk
```

You may assume that the data file is correctly formatted and that the parent links exist and are encountered in the data file as item indices before they are referenced as parent links.

There is a Java program (CreateSeaPortDataFile.java) provided with this package that will generate data files with various characteristics with the correct format. You should be using the program to generate your own data files to test various aspects of your project programs.

Sample input file:

```
// File: aSPaa.txt
// Data file for SeaPort projects
// Date: Sat Jul 09 22:51:16 EDT 2016
// parameters: 1 1 5 5 1 5
//   ports, docks, pships, cships, jobs, persons

// port   name index parent(null)
//    port   <string> <int> <int>
port Lanshan 10000 0

// dock   name index parent(port)
//    dock   <string> <int> <int>
  dock Pier_4 20004 10000 30004
  dock Pier_0 20000 10000 30000
  dock Pier_1 20001 10000 30001
  dock Pier_3 20003 10000 30003
  dock Pier_2 20002 10000 30002

// pship  name index parent(dock/port) weight length width draft numPassengers numRooms
numOccupied
//    pship  <string> <int> <int> <double> <double> <double> <double> <int> <int> <int>
    pship          Gallinules 30000 20000 125.99 234.70 60.67 37.14 746 246 246
    pship              Remora 30001 20001 126.38 358.27 74.12 31.54 3768 979 979
    pship    Absentmindedness 30004 20004 86.74 450.43 33.13 41.67 2143 920 920
    pship       Preanesthetic 30003 20003 149.85 483.92 125.71 31.21 166 409 83
    pship           Shoetrees 30002 20002 134.41 156.96 120.31 35.20 1673 633 633

// cship  name index parent(dock/port) weight length width draft cargoWeight cargoVolume cargoValue
//    cship  <string> <int> <int> <double> <double> <double> <double> <double> <double> <double>
    cship           Erosional 40001 10000 200.80 242.33 38.31 23.49 172.73 188.54 235.57
    cship           Kielbasas 40000 10000 120.85 362.55 96.82 19.09 33.08 188.31 261.57
    cship            Generics 40002 10000 79.90 234.26 73.18 15.71 125.27 179.00 729.95
    cship           Barcelona 40003 10000 219.92 443.54 104.44 34.16 86.69 139.89 813.72
    cship             Toluene 40004 10000 189.12 448.99 73.97 37.67 88.90 175.03 1002.63

// person name index parent skill
//    person <string> <int> <int> <string>
    person               Sara 50000 10000 electrician
    person              Duane 50002 10000 inspector
    person              Betsy 50004 10000 cleaner
    person             Archie 50003 10000 captain
    person             Thomas 50001 10000 clerk
```

Sample output as plain text - which should be displayed in a JTextArea on a JScrollPane in the BorderLayout.CENTER area of a JFrame:

```
>>>>> The world:


SeaPort: Lanshan 10000

  Dock: Pier_4 20004
    Ship: Passenger ship: Absentmindedness 30004

  Dock: Pier_0 20000
    Ship: Passenger ship: Gallinules 30000

  Dock: Pier_1 20001
    Ship: Passenger ship: Remora 30001

  Dock: Pier_3 20003
```

```
   Ship: Passenger ship: Preanesthetic 30003

  Dock: Pier_2 20002
   Ship: Passenger ship: Shoetrees 30002

 --- List of all ships in que:
   > Cargo Ship: Erosional 40001
   > Cargo Ship: Kielbasas 40000
   > Cargo Ship: Generics 40002
   > Cargo Ship: Barcelona 40003
   > Cargo Ship: Toluene 40004

 --- List of all ships:
   > Passenger ship: Gallinules 30000
   > Passenger ship: Remora 30001
   > Passenger ship: Absentmindedness 30004
   > Passenger ship: Preanesthetic 30003
   > Passenger ship: Shoetrees 30002
   > Cargo Ship: Erosional 40001
   > Cargo Ship: Kielbasas 40000
   > Cargo Ship: Generics 40002
   > Cargo Ship: Barcelona 40003
   > Cargo Ship: Toluene 40004

 --- List of all persons:
   > Person: Sara 50000 electrician
   > Person: Duane 50002 inspector
   > Person: Betsy 50004 cleaner
   > Person: Archie 50003 captain
   > Person: Thomas 50001 clerk
```

Suggestions:

Methods that should be implemented.

Each class should have an appropriate toString method. Here is an example of two such methods:

- In SeaPort - showing all the data structures:

```
  public String toString () {
    String st = "\n\nSeaPort: " + super.toString();
    for (Dock md: docks) st += "\n" + md;
    st += "\n\n --- List of all ships in que:";
    for (Ship ms: que ) st += "\n   > " + ms;
    st += "\n\n --- List of all ships:";
    for (Ship ms: ships) st += "\n   > " + ms;
    st += "\n\n --- List of all persons:";
    for (Person mp: persons) st += "\n   > " + mp;
    return st;
  } // end method toString
```

- In PassengerShip, using parent toString effectively:

```
  public String toString () {
    String st = "Passenger ship: " + super.toString();
    if (jobs.size() == 0)
      return st;
    for (Job mj: jobs) st += "\n       - " + mj;
    return st;
  } // end method toString
```

Each class should have an appropriate Scanner constructor, allowing the class to take advantage of super constructors, and any particular constructor focusing only on the addition elements of interest to that particular class. As an example, here's one way to implement the PassengerShip constructor:

- PassengerShip Scanner constructor, the earlier fields are handled by Thing (fields: name, index, parent) and Ship (fields: weight, length, width, draft) constructors.

```
  public PassengerShip (Scanner sc) {
    super (sc);
    if (sc.hasNextInt()) numberOfPassengers = sc.nextInt();
    if (sc.hasNextInt()) numberOfRooms = sc.nextInt();
    if (sc.hasNextInt()) numberOfOccupiedRooms = sc.nextInt();
  } // end end Scanner constructor
```

In the World class, we want to read the text file line by line. Here are some useful methods types and code **fragments** that you should find helpful:

- Handling a line from the file:

```
  void process (String st) {
  //      System.out.println ("Processing >" + st + "<");
    Scanner sc = new Scanner (st);
    if (!sc.hasNext())
      return;
    switch (sc.next()) {
      case "port"   : addPort      (sc);
        break;
```

- Finding a ship by index - finding the parent of a job, for example:

```
  Ship getShipByIndex (int x) {
    for (SeaPort msp: ports)
      for (Ship ms: msp.ships)
        if (ms.index == x)
          return ms;
    return null;
  } // end getDockByIndex
```

- Linking a ship to its parent:

```
  void assignShip (Ship ms) {
    Dock md = getDockByIndex (ms.parent);
    if (md == null) {
      getSeaPortByIndex (ms.parent).ships.add (ms);
      getSeaPortByIndex (ms.parent).que.add (ms);
      return;
    }
    md.ship = ms;
    getSeaPortByIndex (md.parent).ships.add (ms);
  } // end method assignShip
```

You will probably find the comments in the following helpful, they are mostly about similar projects and general issues in Java relevant to our programs:

- CMSC 335 Information
- Cave Strategy - getting started

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.	**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.
Functionality	**40 points** Contains no coding errors. Contains no compile warnings. Creates a GUI. Lets the user select a data file, using JFileChooser. Reads the data file, creating the specified internal data structure . Displays the internal data structure in a nice format in the GUI. Displays the results of a Search specified by the user.	**0 points** Contains coding errors. Contains compile warnings. Does not create a GUI. Does not let the user select a data file, using JFileChooser. Does not read the data file, creating the specified internal data structure. Does not display the internal data structure in a nice format in the GUI. Does not display the results of a Search specified by the user.
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	**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.

	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. 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.	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 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.