Quarter Programming Project

Quarter Project: "Logistics Application"

Phase 1 Requirements

- Phase 1: Facilities, Transport Network & Item catalog (4 Weeks: 4/6 – 5/4) **Due on 5/4 by 5:45pm CST.**
  - Load the Facility Network and Inventory, Code the Shortest-Path Determination, and Code the Facility Output Generation
  - In this phase, you must fully implement the following:
    - Create the Facility Inventory XML File (offline process)
    - Load the Facility Inventory XML File and create internal representation of Facility and its inventory
    - Create the Transport Network File (offline process)
    - Load the Transport Network File and create internal representation of Transport Network
    - Develop/research and implement algorithm to determine the shortest path between 2 Facilities (in decimal days)
    - Create the Items File (offline process)
    - Load the Items File and create internal representation of the Items
    - Generate Facility Status Output
  - You must submit your project with a "Main" class with a "main" method that MUST produce the following 3 outputs, *formatted as shown below*.

Output 1:

Facility Status Output and Format ***for all 18 Facilities*** (format should match the below exactly).

Example:

```
--------------------------------------------------------------------------------
Chicago
-------
Rate per Day: 10
Cost per Day: $300
Direct Links:
Detroit, MI (0.7d); Fargo, ND (1.6d); New York City, NY (2.0d); St. Louis, MO (0.7d);

Active Inventory:
   Item ID     Quantity
   ABC123      60
   CT1928      20
   E241i       64
   RTF110      110
   XTP202      20

Depleted (Used-Up) Inventory: None

Schedule:
Day:           1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20
Available:     10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
--------------------------------------------------------------------------------
```

Output 2:

Item Catalog Content Output (format should match the below exactly).

Note: Multiple columns are not necessary. A single column is fine, multiple columns are better.

```
ABC123  : $550      CR2032  : $240      CT1928  : $910     E241i   : $10,400
JBL3100 : $180      MM35P   : $1,950    PL132-C : $440     PU238   : $2,200
RL123A  : $360      RTF110  : $715      RX100-3 : $642     SN-241-L: $620
SR71-D  : $1,600    XLK200B : $820      XTP202  : $345     ZTF109  : $1,100
```

Output 3:

Example Shortest Path output results (in miles and decimal days of travel time) between the following 10 site-pairs (a - j): *(format should match the below exactly).*

Shortest Path Tests:

a) Santa Fe, NM to Chicago, IL:
- Santa Fe, NM → St. Louis, MO → Chicago, IL = 1,329 mi
- 1,329 mi / (8 hours per day * 50 mph) = 3.32 days

b) Atlanta, GA to St. Louis, MO:
- Atlanta, GA → New Orleans, LA → Nashville, TN → St. Louis, MO = 1,310 mi
- 1,310 mi / (8 hours per day * 50 mph) = 3.28 days

c) Seattle, WA to Nashville, TN:
- Seattle, WA → Fargo, ND → Chicago, IL → St. Louis, MO → Nashville, TN = 2,675 mi
- 2,675 mi / (8 hours per day * 50 mph) = 6.69 days

d) New York City, NY to Phoenix, AZ:
- New York City, NY → Chicago, IL → St. Louis, MO → Santa Fe, NM → Phoenix, AZ = 2,599 mi
- 2,599 mi / (8 hours per day * 50 mph) = 6.50 days

e) Fargo, ND to Austin, TX:
- Fargo, ND → Chicago, IL → St. Louis, MO → Austin, TX = 1,765 mi
- 1,765 mi / (8 hours per day * 50 mph) = 4.41 days

f) Denver, CO to Miami, FL:
- Denver, CO → Santa Fe, NM → Austin, TX → New Orleans, LA → Miami, FL = 2,456 mi
- 2,456 mi / (8 hours per day * 50 mph) = 6.14 days

g) Austin, TX to Norfolk, VA:
- Austin, TX → New Orleans, LA, → Atlanta, GA → Norfolk, VA = 1,547 mi
- 1,547 mi / (8 hours per day * 50 mph) = 3.87 days

h) Miami, FL to Seattle, WA:
- Miami, FL → New Orleans, LA → Nashville, TN → St. Louis, MO → Chicago, IL → Fargo, ND → Seattle, WA = 4,071 mi
- 4,071 mi / (8 hours per day * 50 mph) = 10.18 days

i) Los Angeles, CA to Chicago, IL:
- Los Angeles, CA → Phoenix, AZ → Santa Fe, NM → St. Louis, MO → Chicago, IL = 2,181 mi
- 2,181 mi / (8 hours per day * 50 mph) = 5.45 days

j) Detroit, MI to Nashville, TN:
- Detroit, MI → Chicago, IL → St. Louis, MO → Nashville, TN = 888 mi
- 888 mi / (8 hours per day * 50 mph) = 2.22 days

All outputs must be generated using the data and behaviors of your java objects. No output can be hard-coded to match the expected results. Failure to comply will result in significant grading penalties.

All programs should be implemented as non-interactive, meaning they must execute without any interaction needed by the person running them. Failure to comply will result in grading penalties.

Implementation Details

The purpose of the course project is to give the student hands-on experience implementing the object-oriented concepts covered in this course. As such, all implementations should reflect the programming practices, object-oriented principles, and design patterns covered throughout the quarter. These concepts include (but are not limited to):

- Abstraction
- Separation
- Encapsulation
- Information Hiding
- Inheritance
- Polymorphism
- Composition
- Interface
- Interface Polymorphism
- Delegation
- Good class design
- Strategy Design Pattern
- Factory Design Pattern
- Singleton Design Pattern
- Exception handling
- Single Responsibility Principle
- Open-Closed Principle
- Liskov Substitution Principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Avoiding Anti-Patterns

- **Implementations much consist of original code, written by you only.**
- **No 3rd party code (classes, jar files, libraries, etc.) is to be used in your project. Your project must consist of Java SDK classes and code you have written only. No exceptions.**

Project Assistance

If you are stuck on some design or code related problem that you have exhaustively researched and/or debugged yourself, you can email me a ZIP file of your entire project *(with compiled .class & JAR files removed before ZIP'ing)* so that I can examine the problem.

All emailed assistance requests must include a detailed description of the problem, and the details of what steps you have already taken in trying to determine the source of the problem.

Exceptions & Exception Handling

It is important to note that any method that accepts parameters should validate the incoming parameters before using them. If bad/unusable or invalid (by application standards) data is detected, an appropriate exception should be thrown. You should create your own exception classes to handle all such cases, *do not use "Exception" or pre-existing java exception classes*.

Remember to only "catch" an exception where the problem encountered can be "fixed". This means that in most cases you will not catch the exception, you will propagate it (declare that the method throws the exception).

Catching an exception:

```
try {
 …
}
catch (SomeException ex) {
 …
}
```

Propagating an exception:

```
public void someMethod (int param)
                    throws SomeException {
  …
}
```

Likewise, methods that access system resources (i.e., files) should manage any related exceptions in an orderly manner.

Keep in mind that if you decide to propagate any exceptions from methods that are declared in an interface, the interface declaration of that method must also indicate that the same exceptions are thrown.

Submissions & Grading

- All submissions must include all code and data files necessary to compile and run your application.
- Submissions should reflect the concepts and practices we cover in the source.
- Submitted code must be in the original package-folder form. *Improperly formatted submissions will be penalized.*
- Submissions should consist of your ZIP'd project folder *with compiled .class & JAR files removed before ZIP'ing*. *Submissions made WITHOUT removing compiled .class & JAR files will be penalized.*
- Late submissions will be penalized by 10% per week late. (i.e., from one second late to 1 week late: 10% penalty, from one week plus one second late to 2 weeks late: 20% penalty, etc.). *Note that no late submissions are allowed for the final project phase*.
- The following are the key points that will be examined in Project when graded:
  - Good Object Oriented Design & Implementation
  - Good Programming Practices
  - Good Programming Style & Appearance
  - Proper Application Execution & Output
  - Accurate and Properly Formatted Output

If you do not understand anything in this handout, please ask.
Otherwise the assumption is that you understand the content.

Unsure? Ask!