Coding Exercise

General Instructions

1. Fully review the instructions for all exercises before beginning.
2. Select 2 of the 5 exercises to complete.
3. Please commit your solution no later than due date. The due date is Tuesday, September 17, 2019.
4. Be sure that you have JDK 1.7+ installed and Apache Ant 1.9+ installed on your system
5. Each exercise has some amount of JUnit tests already. More details on tests are provided in the exercise description.
6. There is an ant script that can be used to build and test the project.
7. The exercise can be downloaded using the following link https://drive.google.com/file/d/18yzzVe-mnh2SPOUBiqkHLlMSS4krtlZ1/view?usp=sharing

How to submit work

1. Create from GitHub repo
2. Create a branch
3. Commit your solution to your chosen exercises
4. Share the link of GitHub

Exercise 1 - User and Group Parsing

Package: com.ge.exercise1

For this exercise you will need to retrieve application, user, and group data from structured text. The text will contain the data for a single application and may contain many users and groups. The users and groups are unique based on their ID and may appear multiple times in the text (for example, same user in two different groups). Whitespace and newlines should be ignored.

The format of the text follows the convention

```
Application(
    id: 0,
    name: MyApp,
    users: [
        User(
            id: 2,
            name: Beth Jones
        ),
        ...
    ],
    groups: [
        Group(
            id: 1,
            name: SimpleGroup,
            users: [
                User(
                    id: 2,
                    name: Beth Jones
                ),
                ...
            ]
        ),
        ...
    ]
)
```

What is provided

1. A java interface for the parser and abstract classes for the Application, User, and Group
2. A JUnit test pre-loaded with test data (src/test/com/ge/exercise1/ParserTest.java)

What to do

1. Build a parser class named MyParser that conforms to the Parser interface
2. Build your own Application, User, and Group classes that extend their abstract counterparts
3. Your parser should output an Application object that contains all the appropriate users and groups

Other considerations

1. Do not modify the Parser interface or any of the abstract classes. Build your own classes that extend and implement these interfaces/abstract classes.

Exercise 2 - Array Quartering Utility

Package com.ge.exercise2

For this exercise you will need to build a utility that can perform various operations on a given two dimensional array. The operations are as follows

1. Return all of the values in a specific row in order of appearance (left to right)
2. Return all of the values in a specific column in order of appearance (top to bottom)
3. Return all of the values in a specific quadrant in order of appearance (left to right, top to bottom)

If you consider the following array of data

```
a b c d
e f g h
i j k l
m n o p
```

Operation #1 for row 0 should return `a b c d`
Operation #2 for column 0 should return `a e i m`
Operation #3 for quadrant 0,0 should return `a b e f`

What is provided

1. ArrayQuadrantUtil in src/main/com/ge/exercise2/ArrayQuadrantUtil.java
2. A JUnit test in src/test/com/ge/exercise2/ArrayQuadrantUtilTest.java

What to do

1. Implement the 3 operations in ArrayQuadrantUtil
2. Update the ArrayQuadrantUtil to work on any data type rather than just char
3. Update the ArrayQuadrantUtil to work for any array of valid size
4. Update the ArrayQuadrantUtil to work for an arbitrary partition size. For example, a 12x12 array could be partitioned into 3x3 or 4x4 partitions.
5. Write tests that confirm all functionality

Other considerations

1. Feel free to make any necessary changes

Exercise 3 - Bug Ridden Bank

Package com.ge.exercise3

For this exercise you will need to first identify and fix several bugs in our Bank and Account classes. Once that is done you will have to add several new features that are desperately needed.

What is provided

1. The Bank and Account classes in src/main/com/ge/exercise3
    - The Bank has many Accounts
    - The Account has a balance, a monthly interest rate, a monthly fee, and several other fields and methods
        - Checking accounts default to no interest and no fees
        - Savings accounts default to 1% interest and no fees
2. JUnit tests in src/test/com/ge/exercise3

What to do

1. Identify and fix as many bugs as possible in the Bank and Account classes
2. Get all of the existing unit tests passing for those classes
3. Add functionality in the Bank and Account class to do the following
    - Get a sum of current holdings
    - Project if the bank will produce a profit or loss in the next month based on fees collected on each account vs interest paid out
    - Prevent checking accounts from being overdrawn by more that $100
    - Prevent savings accounts from ever having a negative balance

Other considerations

1. Feel free to make any necessary changes.
2. Retain the existing tests as much as possible.

Exercise 4 - Jet Engines

Package com.ge.exercise4

For this exercise you will be working some classes that represent different GE aircraft engines. We need to make a number of updates and add new functionality. We also need to add a new class for an upcoming engine.

What is provided

1. Classes for the GE90, the GENx, and the GEPassport engines
    - Each class has variables to describe many attributes of the engine such as
        - how many flight hours the engine has
        - how many flight hours before the engine needs to be rebuilt
        - how many times the engine has been rebuilt
        - how many times the engine can be rebuilt before it reaches its end of life
        - its thrust
        - its weight
        - etc.
2. JUnit tests for each engine class in src/test/com/ge/exercise4

What to do

1. Update as necessary to accomplish the following
    - Update the thrustToWeightRatio method to use the dryWeight of the engine rather than the wetWeight
    - Add a method for the engines that calculates how many hours are left before a rebuild is needed
    - Add a method for the engines that calculates how much service life is left in the engine based on the current flight hours and number of rebuilds vs the maximum number of rebuilds number of flight hours between rebuilds
    - Add a new class for the upcoming GE9x engine - attributes below

- maxNumRebuilds = 5
- flightHoursBeforeRebuild = 30,000
- dryWeight = 15,505
- wetWeight = 15,900
- takeoffThrust = 100,000

Other considerations

1. Feel free to make any necessary changes.

Exercise 5 - Bad Warehouse

Package com.ge.exercise5

For this exercise you will need to update our warehouse inventory software to accommodate a new type of item.

The software currently works as follows:

1. Items have a type attribute, sellBy attribute, and a value attribute
    - sellBy indicates how many days we have left to sell the item
    - value indicates how valuable the item is
2. The warehouse can manage any number of items
3. Each day the warehouse runs the updateItems method and updates the status of each item lowering the sellBy and value attribute by 1

However, there are some special cases

1. Once the sell by date has passed, value goes down at twice the normal rate
2. The value of an item can never go below 0
3. Ageable items improve in value over time. Each day their value goes up by one. The rate doubles if past sellBy
4. The value of an item never goes above 50
5. Precious items never decrease in value and never need to be sold
6. Rare items increase in value over time like ageable items, however, within 14 days of the sell by the value improves by 2x the normal rate, and within 7 days the value improves by 3x the normal rate. After the sellBy the value drops to 0.

What is provided

1. Warehouse class
    - Has the updateItems logic
2. Item class
    - Describes the type of item, the sellBy, and the value
3. Extensive JUnit tests in src/test/com/ge/exercise5

What to do

1. Update the software as necessary to accomplish the following
    - Add a new type of item that is perishable.
    - Perishable items are to degrade in value twice as fast as normal items

Other considerations

1. Feel free to make any necessary changes.
2. Retain the existing tests as much as possible.