CS5010, Fall 2024

Assignment 1

Brian Cross & Richard Cobbe
Assignment Credit: Tamara Bonaci

Assignment objectives:

After completing this assignment, you should have:

- Refreshed your memory of Java syntax
- Refreshed your knowledge, and gained practice with object-oriented principles, such as abstraction, encapsulation, inheritance, information hiding and polymorphism
- Gained practice with unit testing
- Gained practice with UML class diagrams
- Gained practice with code documentation and Javadoc
- Gained practice with using the build tool, Gradle

Resources

You might find the following online resource useful while tackling this assignment:

- Java 17 Documentation: https://docs.oracle.com/en/java/javase/17/docs/api/
- JUnit 5 Getting Started Online Documentation: https://junit.org/junit5/docs/current/user-guide/#overview-getting-started
- Sample Java Project setup to build with Gradle: https://github.khoury.northeastern.edu/cs5010seaF24/Code_From_Lectures
- UML Diagram Online Documentation: https://www.smartdraw.com/uml-diagram/

Submission Requirements

Create a new Gradle project for this assignment, named **assignment1**. Please implement every problem within its own package, and name each package **assignmentN.problemM**, where N replaces the assignment number, and M the problem number, e.g., all your code for problem 1 for this assignment must be in a package named **assignment1.problem1.**

Each package should contain:

- One .java file per Java class
- One .java file per Java test class
- One pdf or image file for each UML Class Diagram that you create
- All public methods must have tests

    - Test quality is part of the grade. It is best to have short tests that test a single or small "idea" or "case". Giant test methods with multiple checks are problems because an early failure can hide further failures in the same test method.
- All non-test classes and non-test methods must have a valid Javadoc
- Additionally, your repo should have a class **.gitignore** file. You should not commit any:
    - Any .class files.
    - Any .html files.
    - Any IntelliJ specific files.

Lastly, a few more rules that you should follow in this assignment:

- Your classes should have a public modifier
- Instance fields should have a private modifier
- Use this to access instance methods and fields inside a class
- No "magic numbers" (Literals)
- Use correct Java coding style
    - You can consult Oracle's coding conventions for help: https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html
- Your code must build on grader's machine. Make sure you do not have hardcoded paths in tests (example, test file data, etc.)

To submit your homework, push your solutions to your designated repo within our course organization in a new branch and create a pull request. Lab2 will help with these concepts as well.

Problem 1

You are interning with a small software company, developing a new **frequent flyer management system** (similar to, for example, Alaska Airlines or Delta apps). Your team is developing part of a system that would allow a user to see their current miles balance and use their miles.

For now, your frequent flyer app consists of:

- **Frequent Flyer**. A frequent flyer is an individual with:
    - A unique account ID, which is a 12-character long `String`.
    - A name, consisting of first, middle and last name.
    - An email address, and
    - A miles balance.
- **Miles Balance** is an object consisting of:
    - An integer value, representing total miles available,
    - An integer value, representing miles earned this year,
    - An integer value, representing miles expiring by the end of this calendar year.

Your tasks:

1. Write code to implement your team's part of the frequent flyer system.

2. A frequent flyer app allows a flyer to transfer miles from their account to someone else's account. This functionality is implemented in method `transferMiles(Deposit deposit)`, that takes a `Deposit` as an input argument. `Deposit` consists of:
    a. Deposit amount that is in the range [1000 – 10000] miles.
    b. The information about the recipient's unique account ID and their name.

For security and privacy reasons, the method checks that the provided information about a recipient corresponds to one of the existing customers, and that the provided ID matches recipient's name.

When miles are transferred to someone's account, they count both towards miles earned this year, and miles expiring by the end of this calendar year.

Please implement the method `transferMiles(Deposit deposit)`.

3. Write tests for your class `FrequentFlyer`, and all dependent classes.
4. Provide your final UML diagram that includes method `transferMiles(Deposit deposit)`.

Problem 2

You are a part of vehicle valuation company, like Kelley Blue Book. Your company is developing the next generation of the vehicle management system. At this stage, however, the company just wants to build a simple system, to store the needed information about vehicles.

So, currently, your system distinguishes between two major kinds of **vehicles**:

- Cars
- Vessels

A car can be one of:

- Used car
- New car

The only vessel that your system currently recognizes is a Boat.

For every vehicle, your system keeps track of the following:

- **ID**, a unique identifier of a vehicle, represented as a `String`
- **Manufacturing year,** a year vehicle was manufactured, represented as an `Integer`
- **Make and model**, represented as a `MakeModel`, a custom class that you will have to develop, and that keeps track of:
    - **Make**, vehicle make, represented as a `String`
    - **Model**, vehicle model, represented as a `String`
- **MSRP**, Manufacturer Suggested Retail Price, represented as a `double`.

For every new car, the system additionally keeps track of the number of available vehicles within 50 miles, represented as an integer.

For every used car, the system keeps track of:

- **Mileage**, represented as an integer
- **Number of previous owners**, represented as an integer
- **Number of minor traffic accidents** the vehicle was involved in, represented as an integer

Lastly, for every boat, the system additionally keeps track of:

- **Length**, boat length, represented as a float
- **Number of passengers,** represented as an integer
- **Propulsion type**, represented as an `PropulsionType`, a custom enumeration that you will have to develop, with possible value: ***Sail Power, Inboard Engine, Outboard Engine, Jet Propulsion.***

Your tasks:

3. Design Java classes to capture the above requirements and information.
4. Please write appropriate Javadoc documentation for all your classes and methods.
5. Please write the corresponding test classes for all your classes.
6. Please provide a final UML Class Diagram for your design.

Problem 3

You are a part of a company developing a new interactive tax return preparation system, similar to *TurboTax* or *QuickBooks*. Your system is intended to help tax filers fill in and submit their tax return forms, but for now, your team is focused on building a simple proof-of-concept **tax calculator**.

The tax calculator distinguishes between two kinds of **tax filers**:

- Individual filers
- Group filers

Individual filer can be one of:

- Employee

Group filer can be one of:

- Married, filling jointly
- Married, filling separately
- Head of the household

For ***every* tax filer**, your system keeps track of the following:

- **Tax ID**, a unique tax filer's identifier, represented as a String.
- **Contact info**, represented as a `ContactInfo`, a *custom class that you will have to develop*, and that keeps track of:
    - **Name**, a tax filer's first and last name, represented as a Name, *another custom class that you will have to develop*,
    - **Address**, a tax filer's address, represented as a `String`,
    - **Phone number**, the tax filer's phone number, represented as a `String`,
    - **Email address**, the tax filer's email address, represented as a `String`.
- **Last year's earnings**, represented as a `double`.
- **Total income tax already paid**, represented as a `double`.
- **Mortgage interest paid**, represented as a `double`.
- **Property taxes paid**, represented as a `double`.
- **Student loan and tuition paid**, represented as a `double`.
- **Contributions made to a retirement savings account**, represented as a `double`.
- **Contributions made to a health savings account**, represented as a `double`.
- **Charitable donations and contributions**, represented as a `double`.

For every **group filer**, the tax calculator keeps track of:

- **Number of dependents**, represented as an `int`.
- **Number of minor children**, represented as an `int`.
- **Childcare expenses**, represented as a `double`.
- **Dependent-care expenses**, represented as a `double`.

Your tasks:

1. Design and implement classes to represent the above tax filers.
2. Additionally, every tax filer, should have a method **`public double calculateTaxes()`**.

When the method `calculateTaxes()` is called on a specific tax filer, it calculates that filer's taxes according to the following rules[1]:

- **Basic taxable income:** for ***all tax filers***, their basic taxable income is calculated by subtracting the amount of income taxes already paid from their last year's earnings.
- **Retirement and health savings deduction**: for ***all tax filers***, their current taxable income is reduced by subtracting the retirement and health savings deduction, which is calculated as follows:
    - For **individual tax filers**, the health and retirement savings deduction is calculated by summing up their reported retirement and health savings contributions, and multiplying the sum by 0.7.

[1] Please note that this is a made-up tax code, and it does not correspond to a tax code of any country.

    - For **group tax filers**, the health and retirement deduction is calculated by also summing up their reported retirement and health savings contributions, but the sum is now multiplied by 0.65. Any result higher than $17500 is floored to $17500.
    - If the retirement and health savings deduction is higher than the current taxable income, then the difference between the taxable income and the deduction is set to be equal to 0.
- **Mortgage interest and property deduction: all tax filers** who have earned less than $250 000 last year, and who have reported more than $12500 in mortgage interests and property taxes expenses, can apply $2500 mortgage interest and property tax deduction.
- **Childcare deduction:** All **group tax filers** who have earned less than $200 000, and who have reported more than $5000 in childcare expenses, can apply $1250 childcare deduction.
- **Tax amount: the tax amount** is calculated by taking the resulting taxable income, and applying the following formula:
    - **For individual tax filers** with the resulting taxable income lower than $55 000, the tax amount is calculated by multiplying the resulting taxable income by 0.15.
    - **For individual tax filers** with the resulting taxable income higher than $55 000, the tax amount is calculated by multiplying the resulting taxable income by 0.19.
    - **For group tax filers** with the resulting taxable income lower than $90000, the tax amount is calculated by multiplying the resulting taxable income by 0.145.
    - **For group tax filers** with the resulting taxable income higher than $90000, the tax amount is calculated by multiplying the resulting taxable income by 0.185.

3. Please document your code, write tests, and generate an up-to-date UML diagram!

Submitting Your Assignment

Before beginning work on this assignment (if you are working on this before the git lab, you can work in a temporary folder on your machine and then do the following before you copy your project into your new repo), please confirm that you are starting from your main branch by running:

```
git checkout main
```

Ensure that you have the most up-to-date files by running

```
git pull
```

If this is the first time you're using branches in your repo, you won't see any changes. However, these two steps will become very important once you have multiple branches.

Create a new branch for this assignment by running the following command from your local repo:

```
git checkout -b assignment1
```

Check that you are in the correct branch by running `git branch` or `git status`.

Do this every time you start a work session or resume work after a break. You can switch branches by entering
`git checkout branch_name (no -b flag).`
Replace branch_name with the name of the branch you want to switch to.

Carry out all work for this assignment in your assignment1 branch. This includes adding, editing, and committing files. To push to your remote branch, enter the following:

```
git push -u origin assignment1
```

To submit your work, create a “pull request” to merge your topic branch into your main branch. Tag the TAs as reviewer.

1. Make sure all your work on your topic branch has been committed and all commits have been pushed to your remote repo.
2. Open your repo in your browser. You will see a yellow notification box with the name of the branch you just pushed and a green "compare & pull request" button.
3. Click the "compare & pull request" button. You will be taken to a new page called, "Open a pull request".
4. Click the settings icon in the "Reviewers" tab to the right of the text box. In the popup window, search for your TAs. Click on your reviewer’s username to send them a request.
5. Click "Create pull request" to complete the process. You should see a confirmation that a review has been requested.

After grading, your TA will approve your changes. At this point, you will be able to close the pull request and complete the merge by clicking on the “Merge pull request” button. Do not merge the branch until you have received a review from your TA.

You can read more about pull requests in the Github docs:
https://help.github.com/en/articles/merging-a-pull-request