Requirement Doc

Problem 1

You are interning with a small software company, developing a new **frequent flyer management system** (like, for example, Alaska Airlines or Delta apps). Your team is developing a part of 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 a 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, similar to Kelley Blue Book. Your company is developing a 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 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 company developing a new interactive tax return preparation system, similar to *TurboTax* or *QuickBooks*. Your system is intended to help tax filers fill 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 `Integer`.
- **Number of minor children**, represented as an `Integer`.
- **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.
  - 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:

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

- **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 a UML diagram.