CS/CE 2336
Spring 2023

Project 1 – Airline Reward System

**Project Background:** In December 2022, NorthEast Airlines cancelled most of its flights due to what they claimed was really bad weather. Upon further investigation, it turned out that the airline's flight scheduling system had failed, due to an `if` statement block with two lines of code but no curly braces to enclose them. This caused a `goto` statement to execute accidentally, and the entire system to fail. NorthEast Airlines, unable to bear the shame of having `goto` statements in their code, fired the programmer who wrote the code, and made up a story about bad weather. The fired programmer was subsequently hired by Apple.

Due to the mass cancellations, no one wants to fly on NorthEast Airlines anymore. In an attempt to win back passengers, NorthEast plans to roll out a new rewards program to incentivize traveling with them. Unlike other airlines, NorthEast will reward miles to passengers when their flights are cancelled. Also unlike other airlines, when a passenger earns enough miles to qualify for a tier, that passenger will belong to that tier immediately. You have been hired by NorthEast to implement this rewards program.

**Project Specification:** The rewards program is based on the miles flown within the span of a year, which itself is based on the number of cancelled flights. Flight miles and flight cancellations start to accumulate on January 1, and end on December 31. The following describes the reward tiers, based on miles earned / flights cancelled within a single year:

- *Gold* – 25 flight cancellations. Each cancellation is worth 1000 miles.
- *Platinum* – 50 flight cancellations. Each cancellation is worth 1000 miles.
    - *Platinum Pro* – A special sub-tier of Platinum, reserved for those passengers with the mileage multiplier. This will earn double the miles per cancelled flight (2000 miles) for passengers who did not complain about flight cancellations at all throughout the year.
- *Executive Platinum* – 100 flight cancellations. Each cancellation is worth 1000 miles.
    - *Super Executive Platinum* – A special sub-tier of Executive Platinum, reserved for those passengers with the mileage multiplier. This will earn double the miles per cancelled flight (2000 miles) for passengers who did not complain about flight cancellations at all throughout the year.

For each passenger, you will calculate the tier based on the span of an entire year (assume all flight data is for one year only). The tier that a passenger belongs to is updated in real-time as a flight is cancelled. So, a passenger initially starts without a tier, and once 25 flights are cancelled, that passenger will be upgraded to Gold. Once a year has passed, any passengers with Platinum or Executive Platinum tier who have not complained will be upgraded to Platinum Pro or Super Executive Platinum, respectively.

Create a class for each reward tier and use inheritance to create a class hierarchy, starting with a base class called `Tier`, which is either `abstract` or an `interface`. The list of tiers given above provides an outline of how to structure your classes into an inheritance hierarchy. If you feel that more classes

should be written in a hierarchy than what is given above, feel free to write more as you see fit in your design.

Every tier class should support the following methods. Consider this to be the minimum contract which all tier classes must implement or inherit (the method declarations can be in the base class or interface). This list represents at a minimum the methods which should be supported. You can add as many additional methods as you choose.

- `public int getMiles()` – Returns the miles earned so far.
- `public int getCancelledFlights()` – Returns the current number of flights the passenger was supposed to take but which were cancelled.
- `public int getFlights()` – Returns the current total number of flights the passenger took, including cancelled flights.
- `public void addFlight(boolean isCancelled)` – Adds a new flight. Assume each call to `addFlight` adds a single flight. The parameter, `isCancelled`, will be `true` if the flight was cancelled.

Write a class called `Passenger`, which will represent a NorthEast Airlines passenger. It will have a `private` field of type `Tier`, which will store a reference to the passenger's current reward tier. The Passenger class should implement the following methods as a minimum:

- `public String getTier()` – Returns (as a `String`) the name of the tier the passenger belongs to.
- `public int getMiles()` – Returns the miles earned over the year. This method will simply return the miles from the `Tier` object (the private field mentioned above), by calling its `getMiles` method. This is a form of composition (we are using both composition and inheritance in this project).
- `public int getCancelledFlights()` – Returns the number of flights the passenger was supposed to take but which were cancelled. As with `getMiles`, this method will simply return the number of cancelled flights from the `Tier` object.
- `public boolean hasMultiplier()` – Returns `true` if the passenger has the mileage multiplier. This will return false until the end of the year, at which point a determination can be made as to whether the passenger earned the mileage multiplier.
- `public void addFlight(boolean isCancelled)` – Adds a new flight. This method will call the same method in the `Tier` object. This method may also be a good place to determine if a passenger needs a tier upgrade.

Consider this specification to be the minimum implementation that your design requires. You may need to add additional methods, classes, etc. to handle additional functionality that may be required.

**File Input:** Your program will input passenger and flight data, in chronological order. This data will come from a file called `flight-data.txt`. While you will be inputting a file for simplicity, assume that the file is a real-time stream of data. This means that you *cannot* load the entire file and precalculate things such as total cancelled flights to determine tier, as this would be considered 'looking

into the future'. Write your program so you can update specific passengers with specific flight data as the data 'comes in' (ie, when you read a line in the file).

You may find that as you read the file you need to do various tasks, such as creating a new `Passenger` object, and updating flight information for the passenger's tier. If you feel that you need to create another method or class to handle a task, then please create the method or class.

The input file looks like this:

```
101 Y N
107 Y N
101 Y Y
112 N
103 Y Y
```

The file is formatted such that each line represents a flight for a single passenger. The values are separated by spaces, in which the first value represents the passenger id, the second value represents if the flight was cancelled (can be Y or N), and the third value represents if the passenger complained about the flight being cancelled (can be Y or N). Note that if a flight was *not* cancelled, the passenger would not be able to complain about it, thus if the second value is N, there is not a third value (assume it is also N).

Assume the input file is over the span of exactly one year, in which the first line is the first flight of the year, and the last line the last flight of the year, in chronological order. You are given a sample input file to test your program (feel free to modify it with additional flight data to test your program). When grading, another input file will be used.

**User Prompt / Output:** Once the input file is fully read (all lines read and processed), display a prompt asking the user to enter a passenger id. When a valid id is given, display the following for that passenger:

- Rewards tier the passenger belongs to
- Total miles accumulated over the entire year
- Total cancellations accumulated over the entire year
- Did the passenger earn the mileage multiplier?

After displaying the output, prompt the user to enter another passenger id. Utilize a sentinel value which the user can input to quit the program.

**Code Commenting / Annotations:** There needs to be *Javadoc* compatible comments above every class, field, and method you write. You can optionally add Javadoc annotations to your Javadoc comments. Additionally, method bodies should contain some regular code comments, given where you see fit.

When overriding a method, you need to add the *@Override* annotation.

**Submission information:** Please submit your code directly to eLearning under *Programming Projects* → *Project 1.* Use the *one-class, one-file* paradigm, in which each .java file only contains one public class, named the same as the file. Upload all files individually.

**Grading Criteria:** Your grade will be based on correct output from given user prompt input, and whether the specification was followed as given in this document. As Zylabs is not used for this assignment, formatting output to match test cases is not required. Your program will be run manually with a specific input file and specific user prompt input.