MET CS 520

Information Structures with Java

Assignment 2

BOSTON UNIVERSITY

Before You Begin

You must follow the instructions in the Assignment Instructions guide (found in the Assignments section in Blackboard), which provides important information about the correct process for cloning the GitHub assignment repository.

You must follow the instructions in the Assignment Instructions guide (found in the Assignments section in Blackboard), which provides important information about the correct process for cloning the GitHub assignment repository and committing your finished work.

Do not begin this assignment until you have read the entire Module 2 lecture in Blackboard. All of the information that you will need to complete this assignment is contained in the lecture.

Instructions

GitHub Classroom assignment invitation: **https://classroom.github.com/a/nd9VQe6D?7245**

Please note that the parts highlighted in yellow below represent the code that you will need to write. However, be sure to read all of the assignment carefully.

The scenario for this assignment is airline crew scheduling system. You will create classes to represent the various crew roles as well as a `Flight` class that incorporates several objects of different classes. In the cloned repository under the package named `cs520.assignment2`, there are several *.java* source files and one runnable file called `Module2FlightTest`.

Here are the components and requirements for this application:

- `AirlineEmployee` is a superclass with common properties and methods.
- `Pilot` and `FlightAttendant` are subclasses of `AirlineEmployee`.
- `Flight` is a class composed of instances of the other classes (in a *HAS-A* composition relationship) and other functionality.
- Each flight has a captain and first officer (of type `Pilot`) and two flight attendants (of type `FlightAttendant`).

- When a flight is ready, call its `fly()` method.
- The `fly()` method will first check whether the crew is valid according to these rules:
    - The pilot in the captain role must be designated as a captain and the pilot in the first officer role must *not* be designated as a captain.
    - Flight attendant #1 must be assigned to the first-class cabin, and flight attendant #2 must *not* be assigned to the first-class cabin.
    - The duration of the flight may not cause any crew member's total flight hours to exceed the maximum for their job: 40 hours for pilots and 50 hours for flight attendants.
- If a flight cannot take off because it would cause crew members to exceed their max hours, substitutes must replace them for the flight.

All of the class files (*.java*) are already there in the cloned repository. Do not create any other classes or packages. Also, please keep in mind that the steps are not necessarily the best way to implement a real flight staffing system, so do not use any concepts or techniques not yet taught in this course (such as data structures). Please do not add any other functionality that isn't specifically listed below.

AirlineEmployee.java

This is a superclass for the `Pilot` and `FlightAttendant` classes, so it will contain properties and methods common to both.

- Create the following `private` instance variables and do not assign them values:
    - `jobTitle` (`String`)
    - `name` (`String`)
    - `flightHours` (`int`)
    - `maxFlightHours` (`int`)

- Create `public` getter and setter methods for all of the instance variables. In Eclipse, you can autogenerate them by selecting *Source* > *Generate Getters and Setters* in the top menu.
    - Do *not* include a setter for `maxFlightHours`; this value is never changed from outside of the class (as you will see below), so no setter should be exposed.
- Create a `public void` method called `addFlightHours` that accepts a single `int` argument called `hours`. In the body of the method, add `hours` to the current value of the `flightHours` variable. Either of the following is acceptable:
    - `this.setFlightHours(hours);`
    - `this.flightHours = hours;`
- Create a `public boolean` method called `isAvailable` that accepts a single `int` argument called `duration`. This method must return a `boolean` (`true`/`false`) result of a logic test that determines whether the planned flight will cause the employee to exceed the maximum number of flight hours, i.e., test whether `this.flightHours + duration <= this.maxFlightHours`.

Pilot.java

`Pilot` is a subclass of `AirlineEmployee`, which means that it will automatically inherit the `jobTitle`, `name`, `flightHours` and `maxFlightHours` variables and the corresponding getters/setters. Add the following to this class:

- A `private boolean` variable called `captainRank` (do not assign a value)
- Getter and setter methods for `captainRank`. Important note: Your IDE may name the getter `isCaptainRank()` rather than `getCaptainRank()`; this is standard practice for `boolean` variable getter method names
- A single constructor
    - It should accept two arguments: `name` (`String`) and `captainRank` (`boolean`)
    - Within the constructor body:

- Call `this.setName()` and `this.setCaptainRank()` based on the arguments
- Call `this.setJobTitle()` with a value of "Pilot"
- Set `this.maxFlightHours` to 40 (there is no setter method for this variable because it must not be changed from outside the class).

FlightAttendant.java

`FlightAttendant` is another subclass of `AirlineEmployee`, which means that it will automatically inherit the `jobTitle`, `name`, `flightHours` and `maxFlightHours` variables and the corresponding getters/setters. Add the following to this class:

- A `private boolean` variable called `firstClass` (do not assign a value)
- Getter and setter methods for `firstClass`. Again note that your IDE may name the getter `isFirstClass()`
- A single constructor
    - It should accept two arguments: `name` (`String`) and `firstClass` (`boolean`)
    - Within the constructor body:
        - Call `this.setName()` and `this.setFirstClass()` based on the arguments
        - Call `this.setJobTitle()` with a value of "Flight Attendant"
        - Set `this.maxFlightHours` to 50 (there is no setter method for this variable because it must not be changed from outside the class).

You will notice that the completed `Pilot` and `FlightAttendant` classes are very similar. They only differ in the `boolean` variable name (`captainRank` vs. `firstClass`), corresponding getter and setter method names, `jobTitle` and `maxFlightHours`.

Flight.java

The `Flight` class brings together `Pilot` and `FlightAttendant` objects and adds some methods (commented out) to ensure that the flight crew is valid and to add to each member's flight hours. Above the existing methods, add the following:

- The following `private` instance variables, with the types shown in parentheses. Do not set values for them.
    - `flightNumber` (`int`)
    - `destination` (`String`)
    - `duration` (`int`)
    - `captain` (`Pilot`)
    - `firstOfficer` (`Pilot`)
    - `flightAttendant1` (`FlightAttendant`)
    - `flightAttendant2` (`FlightAttendant`)
- Getter and setter methods corresponding to all of the above variables.
- A single constructor that accepts the following arguments and calls their corresponding setters; for example, `this.setFlightNumber(flightNumber)`
    - `flightNumber` (`int`)
    - `destination` (`String`)
    - `duration` (`int`)
- A public void `fly()` method that does not accept any arguments. Within `fly()`, call the existing `this.isCrewValid()` method and use an `if` statement based on the `boolean` result:
    - If `this.isCrewValid()` returns `true`:
        - Add the proposed flight's duration (`this.duration`) to each of the crew members' `flightHours` by calling their `addFlightHours(int hours)` methods.

        - Print a message to the console that confirms that the flight may proceed. For example:

          ```
          Flight 456 for New York is now departing. It will arrive in 12 hours.
          ```

          The `System.out.printf()` statement would look like this:

          ```
          System.out.printf("Flight %d for %s is now departing. It will arrive in %d hours.%n%n", this.flightNumber, this.destination, this.duration);
          ```

    - If `this.isCrewValid()` returns `false`, simply print a failure message to the console such as this:

      ```
      Can't take off because of invalid crew.
      ```

Do not change or remove the following existing methods in `Flight` but study them to understand how they work and what they are trying to accomplish:

- `void setCrew()`: This is a method for convenience that allows the four crew members to be set in one call rather than in four separate calls to the setter methods. It sets the members using the setters and then calls `printCrewDetails()`.
- `void printCrewDetails()`: Displays information about each crew member, including `name`, `flightHours` and `maxFlightHours`
- `boolean isCrewValid()`: Determines whether the current crew is valid for the proposed flight. There are two checks that it performs:
    - Confirm roles
        - Is the `Pilot` assigned to the `captain` instance variable actually a captain (i.e., has its `captainRank` property set to `true`)?
        - Is the `Pilot` assigned to the `firstOfficer` instance variable **not** a captain (i.e., has its `captainRank` property set to `false`)?
        - Is the `FlightAttendant` assigned to the `flightAttendant1` instance variable assigned to first class (i.e., has its `firstClass` property set to `true`)?

        - Is the `FlightAttendant` assigned to the `flightAttendant2` instance variable **not** assigned to first class? (That is, has its `firstClass` property set to `false`?)
    - Check flight hours
        - For each of the four employees, will the proposed flight's `duration` cause them to exceed their `maxFlightHours`? To do this, the `determineAvailability()` method is called for each.

- `boolean determineAvailability(AirlineEmployee employee)`: Returns a `boolean` value determined by adding the employee's current `flightHours` (total hours flown so far) to the proposed flight's `duration` and checking whether this will exceed the employee's `maxFlightHours`. If this is the case, the crew will be invalid and the flight cannot proceed.

Module2FlightTest.java

This is the class containing the `main()` method, which is where the program will start. Within `main()`:

- Create an instance of `Pilot` in a variable called `captain`. Its constructor requires `name` (`String`) and `captainRank` (`boolean`). Use any `name` you wish, but pass `true` for `captainRank`.
- Create an instance of `Pilot` in a variable called `firstOfficer`. Its constructor requires `name` (`String`) and `captainRank` (`boolean`). Use any `name` you wish, but pass `false` for `captainRank`.
- Create an instance of `FlightAttendant` in a variable called `flightAttendant1`. Its constructor requires `name` (`String`) and `firstClass` (`boolean`). Use any `name` you wish, but pass `true` for `firstClass`.

- Create another instance of `FlightAttendant` in a variable called `flightAttendant2`. Its constructor requires `name` (`String`) and `firstClass` (`boolean`). Use any `name` you wish, but pass `false` for `firstClass`.
- With the crew in place, we can now try some flights!

1. **Flight #1:** Flight 123 from New York to Tokyo (14 hours)
    - Instantiate `Flight` as a variable called `flight1`. The `Flight` constructor expects `flightNumber`, `destination` and `duration` arguments. Set them to 123, "Tokyo" and 14. For example:
      `Flight flight1 = new Flight(123, "Tokyo", 14);`
    - Call `flight1`'s `setCrew()` method, passing `captain`, `firstOfficer`, `flightAttendant1` and `flightAttendant2` as arguments.
    - Call flight1's `fly()` method. The flight will take off as long as the crew members are in valid roles. All of their `flightHours` are zero (default value), so that won't be an issue for this flight.
2. **Flight #2:** Flight 456 from Tokyo to Paris (16 hours)
    - Just below the code for flight #1, instantiate a new `Flight` object as a variable called `flight2`. Set the arguments to 456, "Paris" and 16.
    - Call `flight2`'s `setCrew()` method, passing `captain`, `firstOfficer`, `flightAttendant1` and `flightAttendant2` as arguments.
    - Call `flight2`'s `fly()` method. The flight will take off as long as the crew members are in valid roles. All of their `flightHours` are 20 following the first flight, but the duration of this flight won't cause them to exceed their `maxFlightHours`.
3. **Flight #3:** Flight 789 from Paris to Los Angeles (12 hours)
    - Just below the code for flight #2, Instantiate one final `Flight` object as a variable called `flight3`. Set the arguments to 789, "Los Angeles" and 12.

- Call `flight3`'s `setCrew()` method, passing `captain`, `firstOfficer`, `flightAttendant1` and `flightAttendant2` as arguments.
- Call `flight3`'s `fly()` method. The flight will not be allowed to proceed because it would cause the pilots to exceed their `maxFlightHours`. (30 + 12 > 40). The flight attendants are fine because their `maxFlightHours` is set to 50.
- However, the flight must go on as scheduled! Let's relieve the overworked pilots with substitutes. Create two new `Pilot` instances with different names than the first two and be sure to pass `true` for the first new one's `captainRank` and `false` for the second. Call these new objects `substituteCaptain` and `substituteFirstOfficer`.
- Now call `flight3`'s `setCrew()` method again, this time passing `substituteCaptain`, `substituteFirstOfficer`, `flightAttendant1` and `flightAttendant2` as arguments.
- Call `flight3`'s `fly()` method and now the flight should be able to take off with our fresh pilots and tired flight attendants.

Program Output

The console output of your program should look something like this:

```
Here is the crew for flight 123 to Tokyo:
Captain Charles Lindbergh has flown 0/40 hours
First Officer Amelia Earhart has flown 0/40 hours
Flight Attendant 1 A. Smith has flown 0/50 hours
Flight Attendant 2 B. Jones has flown 0/50 hours
Pilot Charles Lindbergh is available
Pilot Amelia Earhart is available
Flight Attendant A. Smith is available
Flight Attendant B. Jones is available
Crew members verified.
```

```
Flight 123 for Tokyo is now departing. It will arrive in 14 hours.

Here is the crew for flight 456 to Paris:
Captain Charles Lindbergh has flown 14/40 hours
First Officer Amelia Earhart has flown 14/40 hours
Flight Attendant 1 A. Smith has flown 14/50 hours
Flight Attendant 2 B. Jones has flown 14/50 hours
Pilot Charles Lindbergh is available
Pilot Amelia Earhart is available
Flight Attendant A. Smith is available
Flight Attendant B. Jones is available
Crew members verified.
Flight 456 for Paris is now departing. It will arrive in 16 hours.

Here is the crew for flight 789 to Los Angeles:
Captain Charles Lindbergh has flown 30/40 hours
First Officer Amelia Earhart has flown 30/40 hours
Flight Attendant 1 A. Smith has flown 30/50 hours
Flight Attendant 2 B. Jones has flown 30/50 hours
Pilot Charles Lindbergh is NOT available
One or more crew members are not available for this flight.
Can't take off because of invalid crew.

Here is the crew for flight 789 to Los Angeles:
Captain Chesley Sullenberger has flown 0/40 hours
First Officer Orville Wright has flown 0/40 hours
Flight Attendant 1 A. Smith has flown 30/50 hours
Flight Attendant 2 B. Jones has flown 30/50 hours
Pilot Chesley Sullenberger is available
Pilot Orville Wright is available
Flight Attendant A. Smith is available
Flight Attendant B. Jones is available
Crew members verified.
Flight 789 for Los Angeles is now departing. It will arrive in 12 hours.
```

Submitting Your Assignment

Do not submit anything in Blackboard. All of your code must be committed/pushed to the appropriate assignment repository in GitHub before the deadline. It is your responsibility to confirm that your code has been successfully pushed to your online GitHub repository. You may access your online GitHub repository in a web browser by going to:

> https://github.com/MET-CS/assignment-1-**ID**
>
> (Replace **ID** with your GitHub ID. For example: https://github.com/MET-CS/assignment-1-*johnsmith*)

Pushing code to the repository after the 6:00 a.m. deadline may subject you to a lateness penalty, but you may repeatedly commit/push and overwrite your files before the deadline. The teaching team will not look at your code until after the deadline, so it's fine to have broken/incomplete code in your repository before the finished product is pushed.

As with all submitted work in this course, your grade will appear in My Grades in Blackboard.