EECS 293
Software Craftsmanship
2019 Fall Semester

Programming Assignment 2

Due at the beginning of your discussion session on

September 9-13, 2019

Reading

In addition to the following topics, the quiz syllabus includes any material covered in the lectures:

- Chapter 14 in Code Complete
- Items 28, 60, and 62 in Effective Java
- Java's BigInteger documentation (introduction only)
- Section 19.1 in Code Complete (excluding "Forming Boolean Expression Positively", "Guidelines for Comparison to 0", "In C-derived languages …", "In C++, consider ...")
- Section 15.1 ("Plain if-then Statements" only) in Code Complete
- Sections 17.1, 18.1 (excluding "Two Issues …"), and 19.4 in Code Complete ("Convert a nested if …" and "Factor deeply nested code …" only)

Programming

A major headache for airline companies is to reroute passengers who were stranded due to delays or other problems. In programming assignments 2 through 5, you will implement a software system to help airlines find an alternative travel arrangement for a passenger. In this assignment, you will start with setting up the various entities that are part of the rerouting system.

Package

You should organize your project in a package. The package name is up to you: it could range from the simple ('airtravel) to the detailed ('edu.cwru.<cwruid>.airtravel).

Airports

The
`public final class Airport implements Comparable<Airport>`
encodes relevant information for an airport. The `Airport` contains a `private final String code`, which is the universally recognized identifier for the airport, such as "CLE" for Cleveland Hopkins. The airport also contains a `private final Duration connectionTimeMin`, which represent the shortest length of time that a passenger needs to transfer planes or to walk from the reservation counter to the gates. The Airport has a *private* constructor

```
private Airport(
      String code,
      Duration connectionTimeMin)
```

that sets the values of the private variables. It also has a *build* method:

```
public static final Airport of(
      String code,
      Duration connectionTimeMin)
```

that returns an airport with the given code and connection time, or throws a `NullPointerException` if either parameter is null. The main reason for this build method is that it can throw a `NullPointerException` if either parameter is null, thereby relieving the constructor from throwing the exception.

The `code` and `connectionTimeMin` have getters. Airports overrides `equals` and `hashCode` according to the principle that two airports are equal if they share the same code. Airports overrides `compareTo` according to their `code`, which means that the natural order among airports is by their identifier. Airports overrides `toString` to return a short string containing the airport code.

Null Arguments

The programming assignments will ask you to implement various methods. These methods take arguments that are potentially null even though null parameters are not expected or meaningful. Whenever you implement such methods, you should make sure to throw a `NullPointerException` with an appropriate error message

if the argument(s) are null. An example is `Airport::of`. There are three exemptions to this rule:

- The assignments will specify build methods, such as `Airport::of`, to construct objects or throw the exception. Therefore, your constructors can avoid checking whether their arguments are null.
- More generally, private methods do not need to check if arguments are null if they can only be invoked from methods that have already checked.
- If a method is automatically generated by the IDE, you do not need to add manually a validation for null parameters.

Legs

A leg represents a non-stop route from a departure to a destination airport. Multiple legs will be pieced together in future assignments to implement arbitrary routes from departure to destination. The `public final class Leg` has `private final Airport origin` and `private final Airport destination` that represent the departure and destination airport and that have public getters, a private constructor that sets the values of these private variables, and a public final static build method:

```
public static final Leg of(
     Airport origin,
     Airport destination).
```

Flights

A flight represents the trip of an airplane that spans a leg following a timetable.

Flight Schedules

Create the `public final class FlightSchedule` that is supposed to keep track of the departure and arrival time of a single plane. The `FlightSchedule` has two `private final LocalTime` variables called `departureTime` and `arrivalTime` with getters, and a private constructor that sets the private variables. `FlightSchedule` has a public static builder that returns a new `FlightSchedule`, or throws an `IllegalArgumentException` with an appropriate error message if the arrival time precedes the departure time. All airplanes fly

during the daytime, so `LocalTime` does not have to be wrapped after midnight. The `FlightSchedule` has a

```
public final boolean isShort(Duration durationMax)
```

that returns whether the flight is shorter than or equal to the given duration.

Flights

The `public interface Flight` represents flights of different types, and has the methods:

- `public String getCode()` that is meant to return the flight identifier, such as "AA12",
- `public Leg getLeg()` that returns the flight leg,
- `public Airport origin(), destination()` that return the endpoints of the flight leg, and
- `public FlightSchedule getFlightSchedule(), LocalTime departureTime(), arrivalTime(), boolean isShort()` that returns the flight scheduling information.

Additional methods will be added in this and future assignments.

Abstract and Simple Flights

Future assignments will specify different types of flights. However, most flights have some commonalities. The

`public abstract class AbstractFlight implements Flight` implements common flight features. Specifically, `AbstractFlight` delegates to the leg the `origin()` and `destination()` methods, and to the flight schedule the `departureTime(), arrivalTime()`, and `isShort()` methods.

The main flight type is the `public final class SimpleFlight`, which `extends the AbstractFlight`. The `SimpleFlight` has `private final String code, private final Leg leg`, and `private final FlightSchedule flightSchedule` with getters. The private constructor sets the value of the private variable. The build method creates and returns a new `SimpleFlight`.

In the next programming assignment, you will implement policies to find out whether there is a seat on a flight for a passenger.

Flight Groups

The `public final class FlightGroup` represents a set of flights that have the same origin airport and that is organized by departure time. It contains a `private final Airport origin` with a getter, a private constructor that sets the origin airport (`origin` field), and a public static final build method. Furthermore, `FlightGroup` contains

```
private final NavigableMap<LocalTime, Set<Flight>> flights
```

initialized to an empty self-balancing binary search tree.

The `FlightGroup` has a `public final boolean add(Flight flight)` that adds the given flight to the flight group, returns true if it did not already contain the given flight, and throws an `IllegalArgumentException` with an appropriate error message if the given flight did not originate from the correct airport.

The `FlightGroup` has a `public final boolean remove(Flight flight)` that similarly removes the given flight from the flight group, returns true if the `FlightGroup` contained the given flight, and throws an `IllegalArgumentException` with an appropriate error message if the given flight did not originate from the correct airport.

The `FlightGroup` has a

```
public final Set<Flight> flightsAtOrAfter(
     LocalTime departureTime)
```

that returns the flights leaving at or after the given departure time.

Add to `Airport` a `private final FlightGroup outFlights` with `public final boolean addFlight(Flight flight)`, `removeFlight(Flight flight)` that add and remove flights from the airport.

Edit the `SimpleFlight` builder so that it also adds the new flight to the departure airport.

Group Assignment

Your discussion leader will randomly group the section into two student teams. The team will be jointly responsible for Programming Assignments 2 through 5.

General Considerations

These classes may contain as many auxiliary private and package-private methods as you see fit, and additional package-private helper classes may be defined. However, any modification or addition to public classes and methods must be approved by the instructors at the discussion board.

You should write JUnit tests to make sure that your primary methods work as intended. However, we will revisit testing later on in the course, so extensive testing is not yet recommended. Similarly, your code should have a reasonable number of comments, but documentation is going to be the topic of a future assignment. As a general guideline at this stage of the course, comments and tests should be similar to those accepted in EECS 132. Additionally, comments should only be applied to the code sections that you feel are not self-documenting.

Canvas Resources

The module on Java Language Features contains a folder on useful Java features, such as enumerated types, regular expressions, and (under Java 8) the optional class. A discussion board can be used to ask questions and post answers on this programming assignment.

Discussion Guidelines

The class discussion will focus on:

- High-level code organization
- The design and documentation of the implementation
- Straight-line code, conditional code

Your grade will be affected by the topics covered in this and in previous weeks. Your discussion leader may introduce material that has not been covered yet in the lecture. Although future topics will not contribute to your current grade, you are expected to follow your leader's advice in revising the assignment.

Submission

Submit an electronic copy of your program to Canvas. In addition to your code, include a README file explaining how to compile and run the code. The code should be handed in a zip, tar.bz2, or tar.gz archive. Archives in 7z cannot be accepted. You can either bring a laptop to discussion to display your project on a projector, or present your project from the canvas submission.

Note on Academic Integrity

It is a violation of academic integrity not only to copy another group's work but also to provide your code for other groups to copy including but not limited to posts on public github projects or social media.