CS603 Programming Assignment 4

Due by 4:30 PM on Tuesday, 11/07

1. Getting Started

This assignment focuses on classes, instance methods, and static methods. You will be writing two **classes whose name must exactly match** those specified here: **Flight.java** and **Travel.java**. In addition, all **method signatures (i.e., names, parameter order and type) and return types** must match those specified below. When completed, submit both of your Java files via the submit link for this assignment on Blackboard.

2. Programming Assignment (30 points total)

The Flight class (21 points)

Define a class called Flight for representing an airline flight with the following **private instance variables**:

1. **`flightCode`** – a string representing a flight designator
2. **`departure`** – a string representing the departure city
3. **`destination`** – a string representing the arrival city
4. **`duration`** – an integer representing the duration of the flight in minutes

There should be **no other** instance variables in this class.

Define the following **public methods with these exact method names** in the Flight class:

1. **`isValidCode`**: **static** method with one parameter of **type String** that returns a boolean value of **true** if the value of the string parameter represents a valid flight code or **false** otherwise. A valid flight code begins with **two letters** (representing the airline) followed by a **dash** followed by between **3 and 5 digits**. "JB 467", "aa-01", "AA-BBBB", and "x-123456" are all examples of invalid flight codes, while "JB-467", "aa-0123", "AA-1234", "xy-12345" are all valid. Hint: the Character class defines methods for determining if characters are letters or digits.

2. **`setFlightCode`**: **instance** method with one parameter of **type String** that does not return a value. It invokes the above static method, isValidCode(), using the string passed to it as an argument. If the value return by that method call is **true**, then the flight code instance variable is set to the **uppercase** value of the string parameter. If it is **false**, then the code is set to "ZZ-000".

3. **`setDeparture`**: **instance** method with one parameter of **type String** that does not return a value. It sets the corresponding instance variable to the **uppercase value** of the string passed to it.

4. **`setDestination`**: **instance** method with one parameter of **type String** that does not return a value. It sets the corresponding instance variable to the **uppercase value** of the string passed to it.

5. **`getFlightCode`**: **instance** method with no parameters. It returns the flight code instance variable of the invoking Flight object.

6. **Four argument** (4-arg) **constructor**: Sets all instance variables to the values passed as arguments. The parameters for this constructor **must be in the following order**: flight code, departure city, destination city, and flight duration. The constructor must invoke the **set** methods described above for setting the corresponding instance variables. The duration value can be assigned directly, or you can define a set method with to provide this functionality (not required).

7. **`hasMeal`**: **instance** method with no parameters. It returns a boolean value of true if the duration of the invoking Flight object is **at least 5 hours**, in which case a meal is served, or false otherwise.

8. **`toString`**: **instance** method with no parameters that returns a string description of the invoking object, including information on all of the instance variables as well as whether or not the flight includes a meal. For example:

   Flight code: AA-12345 Departure: BOSTON Destination: NEW YORK Duration: 77 minutes No Meal

   Flight code: DL-113 Departure: NEWARK Destination: MIAMI Duration: 300 minutes Meal Included

   For full credit, make use of the **hasMeal()** method to determine whether or not a meal is served on the flight.

9. **`connecting`**: **instance** method with one parameter of **type Flight**. This method returns a boolean value of true if the **destination city** of the invoking Flight object is the **same as the departure city** for the Flight object passed as an argument and if the **first two letters in the flight code** of each flight are the **same**. Otherwise, a value of false is returned.

The Travel class (7 points)

This class will have a single **main method** that provides the following functionality:

1. Prompt the user for the information required to create a Flight object in the following **exact** order:
   - flight code, departure city, destination city, and the duration of the flight in minutes.

   Note that the flight code and cities may include more than one word, so you will need to use the **nextLine()** method in the Scanner class in reading in these input values. Recall that nextLine() treats the linefeed character as valid input so causes unexpected behavior when followed by any other method in the Scanner class used for reading in input. See the notes from Week 1 on how to correct for this.
2. Invoke the 4-arg constructor to **create** a Flight object with the values input by the user.
3. If the flight code of the newly created Flight object is invalid – i.e., it has been set to "ZZ-000" - then continue to prompt until a valid code has been entered. **Set** the flight code for the flight once you have a valid value.
4. P**rint** the Flight object (printing will automatically make use of the **toString()** method without your having to explicitly invoke it).
5. Repeat the above steps to create a second Flight object. Note that storing the two Flight objects to an array of type Flight could help in making your code less repetitive but is not required.
6. Print if the first Flight object connects to the second Flight object, making use of the **connecting()** method.

Following are sample interactions, with user input shown in **boldface**.

Sample 1:

```
Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: DL-123
Enter the departure city: St. Louis
Enter the destination city: Boston
Enter the flight duration in minutes: 300

Flight code: DL-123 Departure: ST. LOUIS Destination: BOSTON Duration: 0 minutes Meal
included

Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: DL-34567
Enter the departure city: Newark
Enter the destination city: Toronto
Enter the flight duration in minutes: 250

Flight code: DL-34567 Departure: NEWARK Destination: TORONTO Duration: 0 minutes No
meal

The two flights do not connect.
```

Sample 2:

```
Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: aa-1234
Enter the departure city: new york
Enter the destination city: boston
Enter the flight duration in minutes: 55

Flight code: AA-1234 Departure: NEW YORK Destination: BOSTON Duration: 55 minutes No
meal

Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: AA-345
Enter the departure city: Boston
Enter the destination city: Cleveland
Enter the flight duration in minutes: 120

Flight code: AA-345 Departure: BOSTON Destination: CLEVELAND Duration: 120 minutes No
meal

The two flights connect.
```

Sample 3:

```
Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: 123
Enter the departure city: Los Angeles
Enter the destination city: dallas
Enter the flight duration in minutes: 320

The flight code for the newly created flight is invalid. Please re-enter: 12-abc
The flight code for the newly created flight is invalid. Please re-enter: aa 123
The flight code for the newly created flight is invalid. Please re-enter: aa123
The flight code for the newly created flight is invalid. Please re-enter: 12345
The flight code for the newly created flight is invalid. Please re-enter: 12-345
The flight code for the newly created flight is invalid. Please re-enter: AB-12345

Flight code: AB-12345 Departure: LOS ANGELES Destination: DALLAS Duration: 320
minutes Meal included

Please enter a flight code that begins with two letters followed by a dash and 3 to 5
digits: A
Enter the departure city: Dallas
Enter the destination city: West Palm Beach
Enter the flight duration in minutes: 250

The flight code for the newly created flight is invalid. Please re-enter: A-1123
The flight code for the newly created flight is invalid. Please re-enter: AA-1123

Flight code: AA-1123 Departure: DALLAS Destination: WEST PALM BEACH Duration: 250
minutes No meal

The two flights do not connect.
```

3. Grading

Your programs must compile and run to receive any credit. Points will be allocated as follows:

21 points for correct Flight class, including:

- 1 point for exactly 4 instance variables, as specified in assignment
- 5 points for set and get methods
- 5 points for **static** isValidCode() method
- 2 points for 4-arg constructor – must make use of set methods
- 2 points for hasMeal() instance method
- 2 points for toString() instance method - must use hasMeal() method for full credit
- 4 points for connecting() instance method

7 points for correct Travel class

- 2 points for prompting for inputs, creating a Flight object, printing its description, and repeating for a second flight
- 3 points for re-prompting for flight codes until a valid code is entered
- 2 points for printing connecting flight info – must use connecting() method for full credit

2 points for style

4. Submitting Your Assignment

When you are satisfied that your program meets all requirements, submit your **two java** files, **Flight.java** and **Travel.java**, to the Blackboard page for Assignment 4.