Homework 06 – Travel Plans

Authors: Chelsea, Allan, Yash, Julia

Topics: polymorphism, exceptions, file I/O

Problem Description

Please make sure to read the document fully before starting!

You and your friends get bored and spontaneously decide to go traveling. As you look for tickets, you get overwhelmed by the number of options you have. Therefore, to help you decide, you create a database that will hold information about the different trips you can book.

Solution Description

Create files:

- `Transportation.java`
- `Flight.java`
- `Bus.java`
- `InvalidBookingException.java`
- `Booking.java`

You will be creating several fields and methods for each file. Based on the description given for each variable and method, you will have to decide whether the variables/method should be static, and what visibility modifier applies. To make these decisions, you should carefully follow the guidelines on these keywords as taught in lecture. In some cases, your program will still function with an incorrect keyword. Private helper methods are allowed and encouraged.

Transportation.java

This class represents a mode of Transportation, and it is an abstract class.

Variables:

All variables should only be visible to the subclasses of Transportation:

- `String company` – the name of the transportation company
- `int id` – the transportation vehicle number identifier
- `String departDate` – departure date and will be represented as "MM-DD-YYYY" (month-day-year) example: 10-29-2021
- `String departTime` – departure time and will be represented as "HHMM"
  example: 1200
- `String arrivalTime` – arrival time and will be represented as "HHMM"

Constructor(s):

- A constructor that takes in `company`, `id`, `departDate`, `departTime`, and `arrivalTime`.
- If the passed in value for `company`, `departDate`, `departTime`, or `arrivalTime` is an empty string or null, then throw an `IllegalArgumentException`.
- If the passed in value for id is not a 5-digit number or negative, throw an `IllegalArgumentException`.

Methods:

All methods in this class should be public:

- `toString()`
  - Should properly override `Object`'s `toString()` method and return `"{company},{id},{departDate},{departTime},{arrivalTime}"`
- `equals()`
  - Should properly override `Object`'s `equals()` method
  - Two `Transportation` objects are equal if they have the same company, id, departDate, departTime, and arrivalTime
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)

Flight.java

This class describes a flight, and it is a concrete implementation of Transportation.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared:

- `int connectors` – the number of connecting flights

Constructor(s):

- A constructor that takes in `company`, `id`, `departDate`, `departTime`, `arrivalTime`, and `connectors`
- If the value passed for `connectors` is negative, throw an `IllegalArgumentException`.
- Reuse any code if possible

Methods:

All methods in this class should be public:

- `toString()`: Should properly override `Transportation`'s `toString()` method and return:
  `"Flight,{company},{id},{departDate},{departTime},{arrivalTime},{connectors}"`

- `equals()`
    - Should properly override `Transportation`'s `equals()` method
    - Two Flight objects are equal if they have the same company, id, departDate, departTime, arrivalTime, and connectors
- Reuse any code if you can.
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)

Bus.java

This class describes a bus, and it is a concrete implementation of Transportation.

Variables:

All variables must not be allowed to be directly modified outside the class in which they are declared:

- `int stops` – the number of stops the bus trip has

Constructor(s):

- A constructor that takes in `company`, `id`, `departDate`, `departTime`, `arrivalTime`, and `stops`
- If the value passed for `stops` is negative, throw an `IllegalArgumentException`.
- Reuse any code if possible

Methods:

All methods in this class should be public:

- `toString()`: Should properly override `Transportation`'s `toString()` method and return:
  `"Bus,{company},{id},{departDate},{departTime},{arrivalTime},{stops}"`

- `equals()`
    - Should properly override `Transportation`'s `equals()` method
    - Two Bus objects are equal if they have the same company, id, departDate, departTime, arrivalTime, and stops
- Reuse any code if you can.
- Any **necessary** getter and setter methods (emphasis on **necessary** – don't write any that have no use)

InvalidBookingException.java

This class describes a checked exception.

Constructor(s):

- A constructor that takes in String representing the exception's message
- A no-args constructor that has default message "Invalid booking type"

Booking.java

This class will hold various public static methods that will let you read and write to the database.

Methods:

All methods must have proper visibility:

- `outputBookings()`
  - Takes in a String object representing the file name to read from
  - Throws a `FileNotFoundException` if the passed in file is null or nonexistent
  - Throws an `InvalidBookingException`
  - Returns an ArrayList of `Transportation` objects
  - Each line of the file will contain information about different types of transportation methods
    - File line tokens:
      `Flight/Bus,company,id,departDate,departTime, arrivalTime,connectors/stops`
    - Iterate through the file and create a `Transportation` object for each line
    - Add each `Transportation` object to an ArrayList
    - If the type of transportation method is not a `Flight` or `Bus`, throw an `InvalidBookingException`
  - **Note:** See example file "Booking.csv" for example input
- `writeBookings()`
  - Takes in a String object representing the file name to write to
  - Takes in an ArrayList of `Transportation` objects
  - Returns a boolean value based on if the write is successful (true) or not (false)
  - Iterate through the ArrayList and write each `Transportation` objects to its own line
  - **Note:** If there is already information contained in the given file, make sure to not overwrite it. Add to the existing file instead.
- `readBookings()`
  - Takes in a String object representing the file name to read from
  - Takes in a `Transportation` object
  - Throws a `FileNotFoundException` if the passed in file is null or nonexistent
  - Throws an `InvalidBookingException`
  - Returns an ArrayList of `Integer` objects
  - Iterate through the file
    - search for the inputted `Transportation` object
    - when you find the inputted `Transportation` object, add the line number it is on to the `ArrayList`.
  - If the inputted `Transportation` object is not found, throw an `InvalidBookingException`.
  - **Example:** If the `Transportation` object is found on lines 1, 3, and 9, return an ArrayList containing Integers 1, 3, and 9.

- `main()`
    - Create two flight and bus objects
    - Write the objects into a file called "TestBooking.csv"
    - Read the csv file using `outputBookings()` and print each object to a new line
    - Create another flight object and add it to "TestBooking.csv"
    - **Note:** This is to help you test your code, and it is not comprehensive. It is suggested you create more test cases.

Reuse your code when possible. Certain methods can be reused using certain keywords. These test on on Gradescope are by no means comprehensive, so be sure to create your own!

Checkstyle

You must run checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle cap for this assignment is 35 points.** This means there is a maximum point deduction of 35. If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Transportation.java`
- `Flight.java`

- `Bus.java`
- `InvalidBookingException.java`
- `Booking.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via Piazza for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Allowed Imports

- `java.io.File`
- `java.io.FileNotFoundException`
- `java.io.IOException`
- `java.io.PrintWriter`
- `java.util.ArrayList`
- `java.util.Scanner`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)

- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.