COMP 2522
Object Oriented Programming
Lab 3

Yue Wang
yue_wang@bcit.ca

Due before noon on Sunday, January 29, 2023

1 Introduction

Welcome to your third COMP 2522 lab.

This week you will implement the Demeter Airlines system using the UML diagram you created for Lab 02. As you implement this system, you will make modifications to your UML. Good! I want you to submit your new UML when you are done.

This lab may seem challenging, but you have ALL the tools you need. You know everything about associations. And you've been thinking about this design for a week now. Start coding it up while it's fresh!

Take your time. Help each other. Ask me lots of questions and ask Akila lots of questions in lab!

Yes, you may use inheritance and polymorphism now that we have talked about it!

Good luck!

2 Submission requirements

This lab must be completed and pushed to GitHub Classroom at or before 11:59:59 AM on Sunday, January 30, 2023.

Don't wait until the last minute as you never know what might happen (computer issues, no Internet, power outage, ...). Submit often, and please make sure your comments are short and clear and specific.

You must not ignore Checkstyle warnings or the code style warnings produced by IntelliJ. **You must style your code so that it does not generate any warnings.**

When code is underlined in red, IntelliJ is telling us there is an error. Mouse over the error to (hopefully!) learn more.

When code is underlined in a different colour, we are being warned about something. We can click the warning triangle in the upper right hand corner of the editor window, or we can mouse over the warning to learn more. If the warning seems silly, tell me about it, and I will investigate and possibly ask the class to modify some settings to eliminate it once and for all!

3 Grading scheme

Your lab will be marked out of 5. On a high level, expect that marks will be assigned for:

Figure 1: This lab, like all labs, will be graded out of 5

1. (2 points) Meeting the functional requirements in this lab, i.e., does the code do what it is supposed to do?
2. (1 point) Meeting non-functional Java OOP design requirements such as encapsulation, information hiding, minimizing mutability, maximizing cohesion and minimizing coupling, making good choices about associations and thinking about Demeter, etc.
3. (1 point) Writing code that is self-documenting and well-styled that generates no warnings from IntelliJ or Checkstyle, with correct and complete Javadocs, using short and atomic methods correctly encapsulated in classes that are well defined.
4. (1 point) Submitting an updated UML diagram in PDF format that matches your finished implementation.

4 Style Requirements

Continuing with this lab, there are code style requirements that you must observe:

1. Your code must compile.
2. Your code must not generate any IntelliJ code warnings or problems.
3. Your code must execute without crashing.
4. Your code must not generate any Checkstyle complaints (unless they are complains I have specifically said you can ignore).
5. Don't squeeze your code together. Put a blank space on either side of your operands, for example. I will be assessing the readability and clarity of your code.
6. All of your program classes must be in package ca.bcit.comp2522.xxx (replace xxx as required by the assignment, lab, quiz, etc.). For example, today's work should go in the ca.bcit.comp2522.labs.lab0X package.
7. All classes require Javadoc comments including the @author tag AND the @version tag. Class comments go after the package and import statements. There should be no blank lines between a class comment and the class it describes.
8. Public constants require Javadoc comments, private constants do not. Constants should be the first thing in your class.
9. Constants should be static and final, are often public, and should be followed by instance variables.
10. Instance variables have private or protected visibility, never public or default visibility.
11. Public and protected methods require Javadoc comments including @param tag(s)the @return tag, and the @throws tag where needed (we won't worry about throws until we talk about exceptions in depth).
12. A method's comment must begin with verbs describing what the method does, i.e., Calculates, Returns, Sets, Prints, etc. Note that we use present tense in Java – Returns, not Return. Prints, not Print.
13. The @return and @param tags go AFTER the description.

14. Private methods require non-Javadoc comments (the comments that start with a slash and a single asterisk).

15. Do comment complicated logical blocks of code inside methods with sparse, clear inline comments.

16. Do not use magic numbers (you must use constants instead). Remember that a magic number is any numeric literal. A constant can be local in a method (use the final keyword with it) or class-level (make it static and ALL_CAPS).

17. All method parameters that are object references must be made final (so we don't forget parameters are passed by value):

    (a) Nice to prevent erroneous assignments, and necessary if parameter is referenced by inner class, but that is perhaps a little advanced for now.

    (b) References made final mean that the reference, once pointing to an object, cannot be changed to point at a different object.

18. Consider making your methods final:

    (a) Making a method final prevents subclasses (those that inherit the method) from changing its meaning.

    (b) Final methods are more efficient (the methods become inline, thus avoiding the stack and generating overhead).

19. Data and methods that work together must be encapsulated in the same class.

20. Code duplication must be minimized.

21. The values of local variables that are primitives are set when they are declared, and local variables are not declared until they are needed.

22. Every class that stores state needs an equals method, a hashCode method, and a toString method.

23. In general, we enforce a fairly hard maximum method length of 20 lines of code. Aim for 10 (excluding braces).

5 Demeter Airlines

1. **Using your UML class diagram from Lab 02 implement the following system**. You will modify this diagram as you implement the design. That's good! That's okay! Update it as you change it.

2. **Submit your updated UML diagram in a PDF called Updated_Demeter.PDF** by dragging the PDF into the project directly in IntelliJ and adding it to version control.

3. Here is a reminder of what you are doing:

    (a) You are designing a system for a charter holiday airline, Demeter Airlines, to track its flights.

    (b) A charter holiday airline is an airline that charters or 'rents' airplanes from the companies that own the planes.

    (c) Demeter Airlines must somehow manage information about all of its flights. You will design the core back end.

    (d) Demeter Airlines sells flights to desirable destinations on the planes that it rents from airline companies.

    (e) Demeter Airlines flights are piloted by a captain and a copilot.

    (f) Demeter Airlines flights are staffed by some flight attendants.

    (g) Demeter Airlines flights have a departure time and airport, and an arrival time and airport.

    (h) Demeter Airlines flights have a capacity which depends on the capacity of the plane booked for the flight.

(i) Demeter Airlines sells seats to customers, who have names and credit card numbers.

(j) At any given time, Demeter Airlines must have immediate and up-to-date access to the following information about each of its flights:

   i. the ID of the plane rented for the flight
   ii. the flight number
   iii. the total number of seats
   iv. the customers booked on the flight
   v. the number of available (unsold) seats

(k) The Demeter Airlines system must also readily provide the departure time, arrival time, and flight duration for every flight.

(l) Demeter Airlines also wants to maintain a list of employees who will be working on each flight.

(m) For simplicity, assume each Demeter Airlines flight has two pilots and some flight attendants.

(n) Pilots and flight attendants have names and IDs.

(o) Planes have an ID (unique for each plane) and capacity (number of seats) and can be booked for a Demeter Airlines flight when they are available.

(p) Flights have an ID composed of two letters followed by three numbers, like AC108.

(q) Airports have a location and a 3 letter ID.

(r) Seats have seat numbers composed of a row and a letter. Seats are available if they are unbooked, else they are unavailable and booked.

(s) Seats can be business class, economy plus, or cattle class.

(t) You do not need to include any information about money, cost, etc. Let's pretend all flights are free. I know. But let's pretend.

(u) Maximize coherence and minimize coupling.

(v) Minimize mutability and visibility.

Test your code and submit before the deadline. Good luck, and have fun!