CS 3500 — Exam 1 Part B

Instructions:

- *This is a do-it-all-by-yourself exam. You are not allowed to communicate with any current and past students of the course in any way during the exam.*
- *You are allowed to use an IDE, or any other text editor on your computer. The only online resources you are allowed to use are the course web page, and the official Java documentation by Oracle.*
- *Please submit on time. The times will be enforced strictly by the server. If you try to submit in the final seconds and are unable to due to a slow server response, you will not be able to submit anything! You cannot use any late days, or otherwise submit later for partial credit.*
- *If you have a question, please email your instructor privately over Teams (quicker) or email.*

Good luck!

Flight (public interface)	An interface for describing airplane flights	p. **??**
BasicFlight (public class)	An implementation of Flight	p. **??**
Reservation (public class)	Information about a group seat reservation	p. **??**
Epsilon (public class)	An airline, that manipulates flights and reservations	p. **??**

Figure 1: Class and interface index

The questions on this exam are related to a set of classes and interfaces that implement the ticketing and reservations system for *Epsilon*, a new budget airline.

Please submit your answers in Part B for your section on the submission server.

1. (a) (15 points) Consider the test `testSomething`. Explain in one or two sentences what this test is doing, and why it is an important test. We are not looking for a line-by-line explanation of what the code does, but rather what the objective of the test is.

   (b) (10 points) The test above is correctly written and *ought* to pass but does not, because of a bug somewhere in the code. What is the bug? (Answer in one or two sentences.)

   (c) (10 points) How can it be fixed? (Explain, or show corrected code.)

2. (15 points) Design one or more complete JUnit test methods for `Epsilon#findFlights`, to confirm that it behaves according to its Javadoc specification.

3. (35 points) Epsilon Airlines wants to implement a feature found on many other airlines: the ability to overbook a flight and place passengers on *standby*. They've asked your help to design and implement this new feature. They intend to roll it out slowly to their various flights; for a while, not every flight will have a standby list.

   Epsilon Airlines wants to treat its customers fairly, so they want to ensure:

   - When reserving seats, either all seats in the request are confirmed, or the whole reservation is placed on standby.
   - As seats become available, reservations on the standby list are retried in the order they were received.
   - Passengers should be able to find out if their reservations are on standby, confirmed, or canceled.
   - Flights need to know the names of all the passengers on the standby list.

   (a) (10 points) Explain how you would extend the code you have been given to support this new feature. **You do not need to implement your extended design here.**

   Your answer should be a bulleted list. Each item in the list must begin with **Add**, **Delete**, **Replace** or **Edit**, followed by fields, methods, interfaces or classes, along with a one-sentence description of each change. For example: "Add a new interface XXXX that has methods to...", "Replace field XX in class YY with...", etc.

   (b) (20 points) From the feature description above: "As seats become available, reservations on the standby list are retried in the order they were received." Concretely, the following two scenarios ought to work properly:

   - A `Flight` has three seats remaining. A customer tries to reserve seats for four people. That `Reservation` is put on the standby list for that flight.
   - A `Flight` has three `Reservations` on the standby list: first a group of four people, and then two single-person reservations. A `Reservation` for three people is canceled. The four-person `Reservation` stays on the standby list, but the two single passengers both get confirmed seats.

   Describe which portions of the code you need to extend, replace, or otherwise modify in order to implement this functionality.

   Your answer must be in the form of code/code snippets, along with comments explaining what the change is (they do not have to be in Javadoc style). You do not have to include tests for this code.

   (c) (5 points) We would like the following claim to be an invariant, but it simply isn't true: "Either the `Flight` is full, or there is nobody on standby (or possibly both)." Fix this statement to be a correct invariant for this standby-list feature.