CSCB07 Introduction to Software Design
Project Phase III

Overview

In Phase III of the project you will complete the implementation of the Android application for searching air travel itineraries.

Learning Goals

By the end of this phase, you should have:

- practised dealing with software requirements that evolve over the course of a software development project,
- worked closely with your teammates to re-evaluate and possibly update your design of a software system,
- produced a working Android application that implements your software design and corresponds to user requirements.

Phase IIIa Interview

The purpose of the interview is to demonstrate to your TA that you have a working (although very limited!) Android application. The application must be able to do the following:

1. Launch and start the main activity.
2. Take some input from the user and use it (in a `.java` class).
3. Transition from one `Activity` to another `Activity` and carry information between the two `Activity`s in an `Intent`.
4. Use at least one class from your back-end (implemented in Phase II of the project) intelligently.

All team members must be present for the interview and should be prepared to answer questions about the implementation of the Android application. Your TA will contact you to schedule the meeting.

Feature List for this Phase

Here are the features that you will implement for this Phase of the project. A lot of this functionality you should have already implemented in the previous phase of the project.

- A User (Client or Admin) can launch the flight booking application and login using a username and password; this loads saved data, if it exists. In our unrealistic implementation, we will simply store usernames and passwords (and any other information you may want to store) in a file on the device. It is entirely up to your design how to store all data, including passwords.
- A User (Client or Admin) can search available flights by entering a departure date, and travel origin and destination. Each flight should be reported with: (1) flight number (alphanumeric), (2) departure date and time, (3) arrival date and time, (4) airline, (5) origin, (6) destination, (7) cost, and (8) travel time.
  The flight numbers will be unique (although they are not restricted to numeric format). We will not deal with time zones, leap years, or any calendar or timing differences, in this project.
- A User can search available itineraries by entering a departure date, and travel origin and destination. An itinerary should include, per flight:
  (1) flight number (alphanumeric), (2) departure date and time, (3) arrival date and time, (4) airline, (5) origin, and (6) destination, plus an overall itinerary cost and travel time.
  A valid itinerary contains no cycles, i.e. it does not visit the same place more than once.
- A User (Client or Admin) can display search results sorted by total travel time or by total cost, in non-decreasing order (at least).
- A Client can view personal and billing information stored for that Client.
- An Admin can view personal and billing information stored for any Client.
- An Admin can upload (in a text file — see format below) personal and billing client information into the system. Each upload adds new records to the existing stored information. If the file contains information for an existing client (i.e., client with the same ID), then that client's information is updated.
- A Client can edit personal and billing information for that Client.
- An Admin can edit personal and billing information for any Client.
- An Admin can upload (in a text file — see format below) flight information into the system. Each upload adds new records to the existing stored information. If the file contains information for an existing flight (i.e., flight with the same ID), then that flight's information is updated.
- An Admin can edit information for a flight.
- A User (Client or Admin) can select an itinerary from the displayed list for booking.
- A Client can book an itinerary for that Client.
- An Admin can book an itinerary for any Client.
- A Client can view booked itineraries for that Client.
- An Admin can view booked itineraries for any Client.
- All information stored in the system should persist (be available in the next launch) when the application is not running.

The Admin will use a file to upload personal and billing information about clients. The file format is as follows:

```
LastName;FirstNames;Email;Address;CreditCardNumber;ExpiryDate
```

where the expiry date is stored in the format `yyyy-MM-dd`.

Flight information will also be uploaded using a file. The file format is as follows:

```
Number;DepartureDateTime;ArrivalDateTime;Airline;Origin;Destination;Price;NumSeats
```

where the date and time is stored in the format `yyyy-MM-dd HH:mm`.

Our airlines are not very client friendly — they do not assign seats at the time of booking. The `NumSeats` value in the flight's data file specifies the total number of available seats that are available for sale on that flight.

We have also created a project structure for you, so that we can auto-test your programs. Do not change this structure, or the auto-tester may assign you a grade of 0. The only changes you will make is:

- add your packages to the `src` folder, and
- modify the class `Driver` so that it can be used to test your code: implement all required public methods and add all the necessary `import` statements.
- You may use private variables in `Driver`, but make sure that `Driver` is required for testing purposes only! It should not contain any algorithms or do any interesting work. It should merely call appropriate methods in the classes that you designed and implemented. In particular, the Android front-end and your back-end should function entirely **without** the `Driver` class.

The software development process

Your team should meet regularly while working on the project. You are required to have two types of meetings — planning meetings and status meetings.

You need to have **two planning meetings**: one in the beginning of the project and one mid-way through the project phase. During a planning meeting, the team will (a) recap on the current state of the project (if mid-way meeting), (b) decide on a set of tasks the team will accomplish before the next planning meeting, and (c) decide who will perform which tasks.

In addition to the planning meetings, the team will meet for **weekly status meetings**. (You are advised, although not required, to have more frequent status meetings.) During status meetings, each member will report on (a) what (s)he has accomplished since the last meeting, (b) what (s)he plans to accomplish before the next meeting, and (c) if there are any problems or obstacles that prevent him/her from making progress.

To demonstrate the software development process the team followed, you need to **maintain a plain text file** called `meetings.txt`, where the team will record all meeting minutes. (See the lecture slides for some example meeting minutes.) *On the day of each meeting*, commit this file into the directory for this phase in your team repository. The contents of this file must match the state of the rest of your repository!

If a member repeatedly does not meet deadlines or misses meetings, contact your instructor immediately.

The search algorithm

The strategy for finding all flight itineraries should be familiar: think of the algorithms for traversing a directed Graph.

A correct algorithm will return all valid itineraries in a "reasonable" time. A valid itinerary does not contain any cycles. In addition, in our system in a valid itinerary:

- the time difference between one flight's arrival and the connecting flight's departure is at least `Driver.MIN_LAYOVER`, and
- the time difference between one flight's arrival and the connecting flight's departure is at most `Driver.MAX_LAYOVER`.

The end of this project phase

At the end of this project phase, your team should have a working version of the application that implements every feature on the above feature list. You should, of course, have Javadoc comments for all your code, and your code must pass the `checkstyle` tool.

Team member and self evaluations

Any student who does not submit their evaluations on time will receive a mark of 0 on this phase of the project.

You will be filling out and submitting a peer evaluation activity on CATME. This form will rate all team members, including yourself, on contributing to the team's work (contributing a sufficient amount of work, contributing work of good quality, being on time, helping teammates) and interacting with teammates (showing interest in teammates' ideas and contributions, asking teammates for feedback and using their suggestions to improve, making sure teammates stay informed and understand each other, providing encouragement and enthusiasm to the team).

These are meant to be private: each team member will submit these separately, and you are not required to show each other your forms. In the case of serious disagreement, or if you request it, we will hold a team meeting to discuss the results, but we will never reveal individual ratings.

Although your formal evaluations are private, that doesn't mean you shouldn't talk with your team members about how the team is functioning. It's much better to be open about this. In our experience, teams who ignore problems and just carry on with the work have the worst results.

Marking

All of these items will affect your grade:

- UML Model
  - The modularity of the design, and the degree to which it is reusable and extensible.
  - The degree to which the design meets the requirements.
  - The use of OO concepts, such as encapsulation and inheritance.
- The appropriate use of files and data structures:
  - Have you made reasonable choices from among the many possibilities?
  - The quality of the written description of data structures and files used by the system.
- Functionality and usability of the application:
  - All functions from the feature list implemented.
  - Stability of application (e.g., it shouldn't crash on invalid input).
- The implementation and the design:
  - The implementation directly follows the UML model.
- Javadoc:
  - The quality of the documentation of the code.
- Coding Style:
  - Code passes the style checker.
  - Simple clear code, no spaghetti code, clear logic.
- Quality of the software development process:
  - The file `meetings.txt` must be committed according to the schedule.
  - The contents of the repository and the state of the code must match the contents of the file `meetings.txt`.
- Subversion commit history:
  - Participation by all team members.
  - Frequent commits over an extended period of time.
  - Appropriate commit logs.
- Peer evaluation
  - To view the evaluation criteria, see the CATME online evaluation form (`www.catme.org`).

Bonus Marks for enhancements:

- A significant enhancement, such as a great user interface, a clever search algorithm, or use of a database, will be awarded bonus marks (up to 5% of the total project mark). Neither of these features constitute a requirement for the project. **The bonus marks will only be awarded if all of the requirements have been fulfilled.** In order to receive a grade, you must indicate, in the README file, what (if any) enhancement your team feels it has made.

Checklist

Have you...

- used your new team repository and submitted your work to directory `p3b`?
- committed the updated `model.pdf` and `data.txt` files?
- committed **all** of the necessary Android files?
- committed `Driver.java` in a package named `driver` in your project?
- committed `meetings.txt`?
- verified that your changes were committed using `svn list` and `svn status`?
- before the separate deadline for team evaluations: submitted your team evaluation forms using CATME?