|  |  |
| --- | --- |
| sait_icon_wordmark_horiz_text_black | **CPRG 251**  **Assignment 4 (Modules 5, 7 & 8)** |

**Student:**

**Mark:**  / 31

## **D2L Submission Instructions**

1. One ZIP file with the naming convention **CPRG251\_A4\_Firstname\_Lastname.zip** (using your first and last name).

If working in a group of two (2), only one team member needs to submit to D2L (both can if you so wish). Both members will receive the same feedback. The file should have the following naming convention: **CPRG251\_A4\_Lastname of member 1\_Lastname of member 2.zip**

1. The ZIP file must contain the following:   
   1. The following directory structure:

* bin/ – Compiled Java files.
* src/ – Java source code files:
  + sait/frs/application/
  + sait/frs/exception/
  + sait/frs/gui/
  + sait/frs/manager/
  + sait/frs/problemdomain/
* doc/ – Generated Javadoc files.
  + Ensure the *private* option is checked and everything is included in the generated documentation.
* lib/ – Any third-party libraries. This folder must contain the backend jar file provided.
* res/ – Any resource or data files. This folder must include airports.csv and flights.csv.
* test/ – Unit test cases. This folder can be empty.

1. A text file named **Readme.txt** in the root folder of the ZIP archive and contain:

* A project title.
* What the program does.
* The date.
* The author.
* How to run the program.

1. A runnable JAR file in the root folder of the ZIP archive.
   1. Use the naming convention: **FirstInitial.Lastname4.jar** (i.e.: J.Blow4.jar).

## It is to be built using only Eclipse IDE and JDK 1.8x.

## **Assignment Instructions**

1. You will use only Eclipse IDE.
2. The due date for this assignment is posted in D2L in the assignment submission area and in the provided calendar located in the *Course Information* area**.** Any assignment submitted after the due date will receive a mark of zero, but feedback may be given.
3. Submissions must be student’s original work. Refer to the Academic Misconduct (AC.3.4) policies and procedures.

## **Problem**

The travel agency is impressed with the look and feel of the travel management system you created in your previous assignment. However, some changes to the flight database had to be made causing the previous backend JAR file to become unusable. You are to implement a new flight reservation system that works with the graphical user interface you designed and developed previously. The new system is to replace the CPRG251A3Backend.jar.

The flight reservation management system will do the following:

1. Load the airports from the existing database.
2. Load the flights from the existing database.
3. Find flights
   * Travel agent can find a flight by providing the origin airport, the destination airport, and a day of the week the flight is departing.
4. Make a reservation
   * The travel agent can make a flight reservation for a traveler. A reservation code will be generated and assigned to the traveler’s name and citizenship.
5. Find reservations
   * A travel agent can find existing flight reservations using the reservation code, airline, and traveler name. The criteria can match any combination of the three fields.
6. Modify a reservation
   * An existing flight reservation can be modified. The travelers name and citizenship can be updated.
   * An existing flight reservation can be soft-deleted, marking it as inactive and freeing up a seat on the flight.

## Requirements

1. The graphical user interface will be in working condition.
2. You will use a random-access file to access and manipulate the reservation database.
3. The flight, and reservation records will be stored as objects in memory.
4. Appropriate exception handling techniques will be used to validate data.

## Details

Using the graphical user interface that was created for the previous assignment, provide your own functional backend. There will be some differences between your functionality and the functionality that was provided in the previous assignment. Therefore, your user interaction handling (controller) will be modified and will not work as it is.

The UML Class Diagram shown below outlines the design of each of the manager and problem domain classes. You will need to provide the exception classes and determine where they are to be thrown.

### Airports

The airports are included in the attached “airports.csv”. Each line in the file represents an airport with the following format:

*Code,Airport*i.e.: YYC,Calgary International Airport

Store airports in an ArrayList<String>. Do not store airports in a random-access file.

### Flights

The flights are included in the attached “flights.csv”. Each line in the file represents a flight record with the following format:

*Flight Code,Departing Airport Code,Arrival Airport Code,Weekday,Time,Seats,Cost Per Seat*  
i.e.: CA-8346,YYC,ATL,Thursday,20:15,174,501.00

Store flight objects in an ArrayList<Flight>. Do not store flights in a random-access file.

The flight code uses the format *LL-DDDD* (L meaning Letter, D meaning Digit) and this needs to be validated. This can be done using regular expressions or by testing each character using the *Character* wrapper class. A flight cannot exist without a valid flight code. In all valid flight codes, the first two letters are an abbreviation and are used to determine the full name of the airline. The following is a list of possible airlines:

* OA – Otto Airlines
* CA – Conned Air
* TB – Try a Bus Airways
* VA - Vertical Airways

### Reservations

The travel agent makes the reservations using the management system. There are no predefined reservations that need to be loaded. Each reservation will be saved to a random-access file. Each reservation record will have the following:

* Reservation code – The reservation code will be generated when a reservation is created.
* Flight code – The flight code the reservation is for.
* Airline – The airline who owns and operates the flight.
* Name – The name of the traveler.
* Citizenship – The citizenship of the traveler.
* Cost – The cost of the reservation.

You are responsible for determining the data types and record size of each reservation in the random-access file. Save the random-access file in the “res” folder.

The reservation code must use the format *LDDDD*, where *L* is either D for Domestic or I for International and *DDDD* is a random number between 1000 and 9999. A domestic flight is a flight going from one Canadian airport to another Canadian airport. Only the Canadian airports start with the letter Y. You are not required to check if the reservation code is unique.

The name and citizenship of the traveler do not need to meet any specific format. However, they cannot be null or an empty string. Each flight has a limited number of seats and a check is needed to determine if a flight is full or not. Use proper exception handling techniques when creating reservations.

Once a reservation is made, the only information that can be changed is the travelers name and citizenship. These must be validated to ensure not null and not empty. The reservation record must be updated in the random-access file.

To cancel an existing reservation a soft or hard delete can be used. If you are using a soft delete, the cancelled reservation will not be included in the number of seats used on a flight. You may also use a hybrid delete, meaning a new reservation can overwrite an inactive reservation.

## UML Class Diagram

### Package: *sait.frs.manager*

|  |
| --- |
| **ReservationManager** |
| - reservations: ArrayList<Reservation> |
| + ReservationManager()  + makeReservation(flight: Flight, name: String, citizenship: String): Reservation  + findReservations(code: String, airline: String, name: String): ArrayList<Reservation>  + findReservationByCode(String code): Reservation  + persist(): void  - getAvailableSeats(flight: Flight): int  - generateReservationCode(flight: Flight): String  - populateFromBinary(): void |

|  |
| --- |
| **FlightManager** |
| + WEEKDAY\_ANY: String  + WEEKDAY\_SUNDAY: String  + WEEKDAY\_MONDAY: String  + WEEKDAY\_TUESDAY: String  + WEEKDAY\_WEDNESDAY: String  + WEEKDAY\_THURSDAY: String  + WEEKDAY\_FRIDAY: String  + WEEKDAY\_SATURDAY: String  - flights: ArrayList<Flight>  - airports: ArrayList<String> |
| + FlightManager()  + getAirports(): ArrayList<String>  + getFlights(): ArrayList<Flight>  + findAirportByCode(code: String): String  + findFlightByCode(code: String): Flight  + findFlights(from: String, to: String, weekday: String)  - populateFlights(): void  - populateAirports(): void |

### Package: *sait.frs.problemdomain*

|  |
| --- |
| **Flight** |
| - code: String  - airlineName: String  - from: String  - to: String  - weekday: String  - time: String  - seats: int  - costPerSeat: double |
| + Constructors  + getCode(): String  + getAirline(): String  + getFrom(): String  + getTo(): String  + getWeekday(): String  + getTime(): String  + getSeats(): int  + getCostPerSeat(): double  + isDomestic(): boolean  - parseCode(code: String): void  + toString(): String |

|  |
| --- |
| **Reservation** |
| - code: String  - flightCode: String  - airline: String  - name: String  - citizenship: String  - cost: double  - active: boolean |
| + Constructors  + getCode(): String  + getFlightCode(): String  + getAirline(): String  + getName(): String  + getCitizenship(): String  + getCost(): double  + isActive(): boolean  + setName(name: String): void  + setCitizenship(citizenship: String): void  + setActive(active: boolean): void  + toString(): String |

## **Marking Guide**

|  |  |  |  |
| --- | --- | --- | --- |
| **Follows submission guidelines** | | | |
| 1. | Correct files |  |  |
| 2. | File naming conventions followed |  |  |
| 3. | NO extra files |  |  |
| **Subtotal** | |  | **/3** |
| **Flight Reservation Management System** | | | |
| 4. | Compiles and runs |  |  |
| 5. | Runnable JAR file |  |  |
| 6. | Clear instructions |  |  |
| 7. | Javadoc documentation |  |  |
| 8. | Populates airports and flights |  |  |
| 9. | Find flights |  |  |
| 10. | Store reservations in random-access file |  |  |
| 11. | Make reservation |  |  |
| 12. | Generate reservation code |  |  |
| 13. | Handle Invalid flight code exception |  |  |
| 14. | Handle invalid citizenship exception |  |  |
| 14. | Handle no more seats exception |  |  |
| 15. | Applicable fields not editable |  |  |
| **Subtotal** | |  | **/28** |
| **Total** | |  | **/31** |