|  |  |
| --- | --- |
| sait_icon_wordmark_horiz_text_black | **CPRG 251**  **Assignment 2 (Modules 4-8)**  **Winter 2021** |

**Student:**

**Mark:**  / 45

## **D2L Submission Instructions**

1. One ZIP file needs to submit to D2L with the following naming convention

The file should have the following naming convention: **group\_number.zip**

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

* bin/ – Compiled Java files.
* src/ – Java source code files:
  + sait/frms/application/
  + sait/frms/gui/
  + sait/frms/exception/
  + sait/frms/manager/
  + sait/frms/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 can be empty.
* 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 **Readme.txt** file that contains:

* Project’s title.
* The date.
* The authors.
* What the program does.
* List of deficiencies if any.
* How to run the program.

1. A runnable JAR file in the root folder of the ZIP archive.
   1. Use the naming convention: **Assignment2.jar**.

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

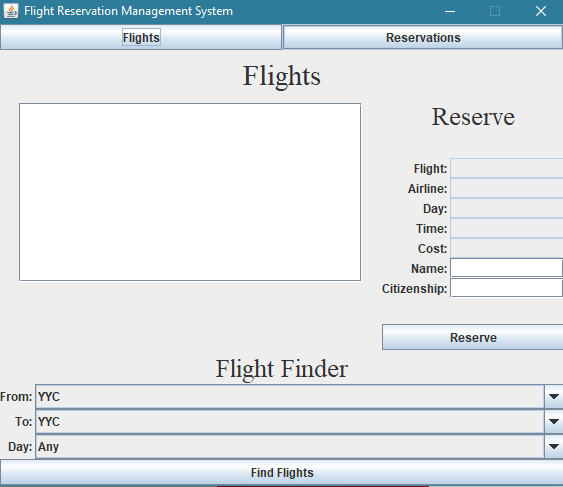
## **Assignment Instructions**

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

## **Problem**

A travel agency is looking to implement a flight reservation management system to improve its productivity and services. **This is a big assignment**. In order to tackle the big problem, you must break down the solution to three major parts:

Part 1 – Non-functional front-end GUI.





Part 2 – Functional back-end.

The supplied data files are to be placed in the *res/* directory.

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.

|  |  |
| --- | --- |
| *sait.frms.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 | |

|  |  |
| --- | --- |
| *sait.frms.problemdomain* |  |
| **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 | |

|  |  |
| --- | --- |
| *sait.fmrs.manager* |  |
| **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): ArrayList<Flight>  - populateFlights(): void  - populateAirports(): void | |

|  |  |
| --- | --- |
| *sait.frms.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 | |

### 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.

Part 3 – Functional front-end GUI.

Your task is to create a functional event driven program. It will allow the user to do the following:

1. 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.
2. 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.
3. 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.
4. 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.

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |
|  |
|  |

## Details

Upon the graphical user interface being launched the user can chose to either search flights and make a reservation or search for and modify a reservation.

### Find Flights

The findFlights method receives as its input arguments: the originating airport, the destination airport, and the day of week. The method returns an ArrayList of any matching *Flight* objects. If no matches are found, the list control will be empty.

### Make Reservation

When a travel agent selects a flight from the list, the text fields will be populated with the selected flight code, airline, day, time and cost. The travel agent will enter the traveler’s full name and citizenship. The flight code, airline, day, time and cost cannot be edited. An error message will be displayed if a reservation is to be made and no flight is selected, the name field is empty, or the citizenship field is empty.

The makeReservation method receives as its input arguments: a Flight object, the travelers name and citizenship. An exception is thrown if the flight is completely booked, or the flight is null, or the name is empty/null, or the citizenship is empty/null. If there are no exceptions thrown a Reservation object is created, saved to the binary file and returned by the method.

### Find Reservations

A travel agent can search for an existing reservation that contains the specified reservation code, or airline or traveller’s full name. The list will be populated with any reservations that are found. Each row in the list displays the code of the corresponding reservation record.

The findReservation method receives as its input arguments: reservation code, airline and traveler’s full name. The method returns an ArrayList of matching *Reservation* objects. If no matches are found, the list control will be empty.

### Update Reservation

When a reservation in the list is selected, the corresponding fields will be populated. These fields will display the:

* Reservation code
* Flight code
* Airline name
* Cost
* Name
* Citizenship
* Status (Active or inactive)

The only fields that can be edited are the name, citizenship, and status. None of the other fields can be modified in any way by the user. After the travel agent has made any changes to the reservation, the update button will be clicked. The mutator methods in the Reservation object will be called and an error maybe displayed if an exception occurs. The persist method in the *ReservationManager* class saves all Reservation objects to a binary file on the hard drive.

### Notes

* The *ReservationManager* class generates the reservation code.
* Each reservation is for one seat only.
* The name and citizenship do not need to follow any specific format; however, they cannot be empty.
* Each problem domain class overrides the *toString()* method.
* Flight codes use the following format: (L meaning Letter, D meaning Digit)
  + *LL-DDDD* (I.e.: *GA-1234*)
* Reservation codes use the following format: (L meaning Letter, D meaning Digit)
  + *LDDDD* (i.e.: *I1234*)
* Times are in 24-hour format:
  + *HH:MM*
* A reservation that is set to inactive is persisted and retained when the program opens again.

## **Marking Guide**

|  |  |  |  |
| --- | --- | --- | --- |
| **Follows submission guidelines** | | | |
| 1. | Correct files |  |  |
| 2. | File naming conventions |  |  |
| 3. | NO extra files |  |  |
| **Subtotal** | |  | **/3** |
| **Flight Reservation Management System** | | | |
| 4. | Compiles and runs |  |  |
| 5. | Runnable JAR file |  |  |
| 6. | Javadoc documentation |  |  |
| 7. | Clear instructions |  |  |
| 8. | User interface design and functionality |  |  |
| 9. | Populates airports and flights |  |  |
| 10. | Find flights |  |  |
| 11. | Generate a reservation code |  |  |
| 12. | Make a reservation |  |  |
| 13. | Find reservations |  |  |
| 14. | Store reservations in a random-access file |  |  |
| 15. | Modify and cancel a reservation |  |  |
| 16. | Populate lists and allows selection |  |  |
| 17. | Handle Invalid flight code exception |  |  |
| 18. | Handle invalid citizenship exception |  |  |
| 19. | Handle no more seats exception |  |  |
| 20. | Applicable fields not editable |  |  |
| **Subtotal** | |  | **/42** |
| **Total** | |  | **/45** |