**Important Instructions:**

1. **Please read the document thoroughly before you code.**
2. **Import the given skeleton code into your Eclipse.**
3. **Run the database script provided to set up your database.**
4. **You have to test the code and ensure there are no compilation errors before submission**
5. **Business Scenario:**

ABC AIRLINE wants to incorporate the process of adding the flight schedule for the upcoming months of their flights. If the flight is not scheduled, add the new schedule for the flight and if any changes on flight schedule, update the existing schedule for the flight.

The schedule details of all the flights will come as flat file from location wise, which is referred as **source** in the case study. The proposed system will validate the flight schedule and update in a database.

1. **Functional Requirement Specification:**

|  |  |  |
| --- | --- | --- |
| Req. # | Req. Name | Req. Description |
| 1 | Parse Input | The input feed has to be parsed and Flights has to be filtered based on the business rules. |
| 2 | Calculate the duration time, Persist Air Flight Schedule Details | If departure date is after current date and If arrival time is after departure time calculate the duration, add the duration to the flight schedule.  If any changes in the existing flight schedule for any date, then reschedule the flight.  Save/Update the flight schedule details into the database based on business rules. |

1. **Skeleton File for Development**

Import the below attached skeleton code into your eclipse project and implement the required functionalities. The skeleton also has .SQL file which can be used to set up your database.

1. **Use case Diagram**

­

Database

Source

Save

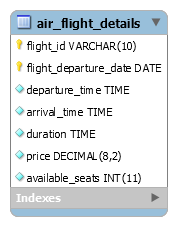
<<include>>

<<include>>

Save

** **

1. **Database**

****

1. **Technical Requirement**

For both the functional requirements 1 and 2, component specification and method specification are given below. Go in the same order to implement them using the code skeleton

1. **A. Component Specification:**

|  |  |
| --- | --- |
| *Requirement Name* | 1. Parse Input |
| *Component Definition* | Helps to parse Input file, and convert into value objects |
| *Files Included*  *(refer Skeleton)* | AirFlightAdminService.java, ApplicationUtil.java, AirFlightSchedule.java, inputFeed.txt, FlightScheduleAdminSystemException.java |
| *Responsibilities* | Reads the input file, does validation to check if the record is Air Flight Schedule, builds the Air Flight Schedule value object and returns it |
| *Design Constraints* | 1. Input file format is .txt and is comma separated (Sample rows are added. You can add any number of rows to test your service class, from main method. 2. Do not hard code the input file path inside any method – has to be used from the input argument only as per code skeleton. 3. File Structure is like below:   <flight id>,<flight departure date >,<departure time >,<arrival time>,<duration>,<price >,<available seats >.   1. In the inputFeed, filter flight schedule. You can identify the flight schedule by checking if the flight id of the row starts with ‘AF’. 2. Assume that the Flight Departure Date in the file will be in the format yyyy-MM-dd. 3. Assume that the Departure Time or Arrival Time or Duration in the file will be in the format HH:mm:ss. 4. Do not change the data types of the value object given in POJO. 5. Always convert the Flight Departure Date values to java.util.Date with format, yyyy-MM-dd before setting in AirFlightSchedule value object. 6. Always convert the Departure Time or Arrival Time or Duration values to java.util.Date with format, HH:mm:ss before setting in AirFlightSchedule value object. 7. Use ApplicationUtil.java for reading file, performing date operations, etc. |
| *Resources* | inputFeed.txt is the input file that must be parsed. The file, along with file location will be sent as an argument to the AirFlightAdminService method. File location/path must not be hardcoded |
| *Process Flow* | 1. The app will be invoked by calling the AirFlightAdminService. addAirFlightSchedules with the input feed (.txt file) 2. Read the file using File I/O or Java Streams in ApplicationUtil.readFile method 3. Return a list of Air Flight Schedule rows from input file, from the readFile method 4. Code the method AirFlightAdminService.buildAirFlightScheduleList. Call the readFile method from this method. Read every line from the list returned by readFile method, split the records based on comma separator 5. Use the ApplicationUtil. convertStringToDate method to convert the date from String Format to java.util.Date format (yyyy-MM-dd). 6. Use the ApplicationUtil. convertStringToTime method to convert the date from String Format to java.util.Date format (HH:mm:ss). 7. Build the Air Flight Schedule Object from the values obtained in every line (Check the Input file format in Design Constraints row) |
| *Exceptional Conditions* | While doing File I/O in the ApplicationUtil.readFile method, catch all exceptions and throw application specific exception, FlightScheduleAdminSystemException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class******Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| AirFlightAdminService | addAirFlightSchedules | String | Boolean |
| ApplicationUtil | readFile | String | static List<String> |
| AirFlightAdminService | buildAirFlightScheduleList | List<String> | List<AirFlightSchedule> |
| ApplicationUtil | convertStringToDate | String | Java.util.Date |
| ApplicationUtil | convertStringToTime | String | Java.util.Date |

1. **A. Component Specification:**

|  |  |
| --- | --- |
| *Requirement Name* | 1. Persist Data into Database |
| *Component Definition* | Helps to calculate the duration time and add / modify air flight schedule details to database. |
| *Files Included*  *(refer Skeleton)* | AirFlightAdminService.java, AirFlightScheduleDao.java, ApplicationUtil.java, AirFlightSchedule.java, FlightScheduleAdminSystemException.java |
| *Responsibilities* | Calculate the duration time if arrival time is after departure time. Persists all Master card details to database. |
| *Design Constraints* | 1. The database.properties has connection details required to connect to the backend 2. Do not change the keys of the property files; you can update the values based on the local database settings. For example, do not change the key, db.username. Rather you can have any value as user name based on local settings. 3. Use only JDBC to establish Database connection 4. Assume the location of the property file will be always as given in the skeleton. 5. Don’t Hardcode the connection string to establish database connection. Read it from property files. 6. Use Prepared Statement to insert records 7. Close all the resources after use 8. Catch all database related exception and throw Application specific exception only from DAO or from DBConnectionManager class. There has to be a private constructor in DBConnectionManager class, to load the database property file and to establish a database connection using JDBC 9. Rollback the Insert or update if any SQL exception has occurred. Throw application specific exception, FlightScheduleAdminSystemException. |
| *Resources* | database.properties – has connection details, used to establish database connection. |
| *Process Flow* | 1. Modify the AirFlightAdminService. buildAirFlightScheduleList check the flight departure date is after current date and arrival time is after departure time. If yes, calculate the duration time as [arrival time – departure time] assuming that the flight departure and the arrival in same day, and set calculated duration to the Air Flight Schedule Objects. 2. Use ApplicationUtil.timeDifference method to calculate the duration time based on due departure time and arrival time. 3. The method AirFlightAdminService.buildAirFlightScheduleList must return the list of Air Flight Schedule with calculated duration time. For records where the flight departure date is current date or before current date and arrival time is before departure time don’t persist the records. 4. After reading file, building records as List<AirFlightSchedule>, call the AirFlightScheduleDao. insertAirFlightSchedules method if the flight is not scheduled for the given date and AirFlightScheduleDao.updateAirFlightSchedules method if the flight is already scheduled for the given date. You may have to convert the java.util.Date to java.sql.Date or java.sql.Time before storing to database. 5. If Insert/update has happened successfully, return true; false otherwise. |
| *Exceptional Conditions* | While working with DAO methods catch all exceptions and throw application specific exception, FlightScheduleAdminSystemException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class******Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| AirFlightAdminService | addAirFlightSchedules() | String | Boolean |
| DBConnectionManager | DBConnectionManager() | NA | NA |
| DBConnectionManager | getInstance() | NA | DBConnectionManager |
| AirFlightScheduleDao | findAirFlightSchedules() | AirFlightSchedule | Boolean |
| AirFlightScheduleDao | updateAirFlightSchedules() | List<AirFlightSchedule> | Boolean |
| AirFlightScheduleDao | insertAirFlightSchedules() | List<AirFlightSchedule> | Boolean |
| ApplicationUtil | timeDifference() | java.util.Date, java.util.Date | java.util.Date |

**Note:** You are allowed to modify input file text to incorporate more test data for various test scenarios / boundary conditions. Test your application by invoking the service methods from the main class, main() method. Follow Java Naming Conventions; test the code quality by running PMD rules in Eclipse or any other IDE that you use.