**Important Instructions:**

1. **Please read the document thoroughly before you code.**
2. **Import the given skeleton code into your Eclipse.**
3. **Use Java 8 for solving the code challenge.**
4. **Run the database script provided to set up your database.**
5. **You have to test the code and ensure there are no compilation errors before submission**
6. **Business Scenario:**

A leading consumer bank wants to automate their process of adding late payment charges to bill amount of their Master Card Customers. If the due date is expired, add the late payment charges as per given business rules.

The bill details of ALL the customers will come as a flat file from region wise transactional systems, which is referred as “source” in the case study. The proposed system has to look up for Master Card holders and update the Bill Amount by adding late payment charges based on the bill due date as applicable. The Master Card Account details along with updated bill amount have to be persisted in a database.

1. **Functional Requirement Specification:**

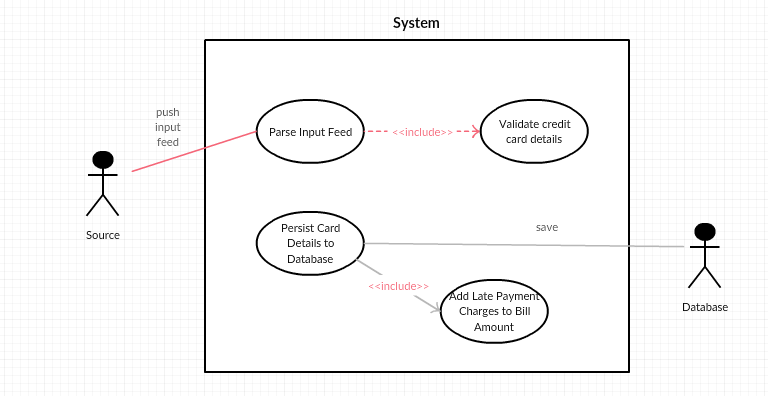
|  |  |  |
| --- | --- | --- |
| Req. # | Req. Name | Req. Description |
| 1 | Parse Input | The input feed has to be parsed and Master Credit Card holders have to be filtered |
| 2 | Add Late Payment Charges, Persist Master Card Account Details | If Due date is expired, add late payment charges to the bill amount as per business rules (explained in section 5) for the customers.  Save the master card account details with updated bill amount into the database. |

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**



1. **Technical Requirements**

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)* | CreditCardAdminService.java, ApplicationUtil.java, CreditCard.java, Inputfeed.txt, CreditCardAdminSystemException.java |
| *Responsibilities* | Reads the input file, does validation to check if the record is Master Card, builds the Credit Card 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:   <credit card number>,<customer name>,<customer email>,<customer phone>,<bill amount>,<due date>,<paid date>   1. In the input feed, filter customers who hold only Master card. You can identify the master card holder by checking if credit card number of the row starts with ‘5’. 2. Assume that the bill amount is in INR 3. Assume that the Due Date or Paid Date in the file will be in the format yyyy-MM-dd. 4. Do not change the data types of the value object given in POJO. 5. Always convert the due date and paid date values to java.util.date with format, yyyy-MM-dd before setting in CreditCard value object. 6. 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 CreditCardService method. File location/path must not be hardcoded |
| *Process Flow* | 1. The app will be invoked by calling the CreditCardAdminService. addCreditCardDetails 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 Master Card rows from input file, from the readFile method 4. Code the method CreditCardAdminService.buildMasterCreditCardList. 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. Build the Credit Card Value 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, CreditCardAdminSystemException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class******Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| CreditCardAdminService | addCreditCardDetails() | String inputFeed | Boolean |
| ApplicationUtil | readFile | String fileName | static List<String> |
| CreditCardAdminService | buildMasterCreditCardList | List<String> records | List<CreditCard> |
| ApplicationUtil | convertStringToDate | String inputDate | Date |

1. **A. Component Specification:**

|  |  |
| --- | --- |
| *Requirement Name* | 1. Persist Data into Database |
| *Component Definition* | Helps to calculate the updated bill amount and add card details to database. |
| *Files Included*  *(refer Skeleton)* | CreditCardAdminService.java, ApplicationUtil.java, CreditCard.java, DBConnectionManager.java, CreditCardAdminSystemException |
| *Responsibilities* | Updates bill amount if due date is expired. 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 if any SQL exception has occurred. Throw application specific exception, CreditCardAdminSystemException |
| *Resources* | database.properties – has connection details, used to establish database connection. |
| *Process Flow* | 1. Modify the CreditCardAdminService.buildMasterCreditCardList method check if the due date is expired (past date). If yes, add 500 INR to the bill amount, and set the updated bill amount to the Master Credit Card objects. 2. Use CreditCardAdminService .getBillAmountWithLatePaymentCharges method to add 500 to the bill amount passed as parameter, based on due date. 3. The method CreditCardAdminService .buildMasterCreditCardList must return the list of master cards with updated bill amount, after adding late payment charges as applicable. For records where the due date is not expired, let’s persist the records as it is. 4. After reading file, building records as List<CreditCard>, call the CreditCardDAO. addCreditCardDetails method to insert values to database. You may have to convert the java.util.date to java.sql.date before storing to database. 5. If Insert has happened successfully, return true; false otherwise. |
| *Exceptional Conditions* | While working with DAO methods, catch all exceptions and throw application specific exception, CreditCardAdminSystemException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class******Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| CreditCardAdminService | getBillAmountWithLatePaymentCharges | Double billAmount | Double |
| CreditCardAdminService | buildMasterCreditCardList | List<String> records | List<CreditCard> |
| CreditCardAdminService | buildMasterCreditCardList | List<String> records | List<CreditCard> |
| DBConnectionManager | DBConnectionManager() | NA | NA |
| DBConnectionManager | getInstance() | NA | DBConnectionManager |
| CreditCardDAO | addCreditCardDetails | List<CreditCard> | Boolean |

**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.