**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**

1. **BUSINESS SCENARIO**

A Prestigious Bank in the country wants to better their service to their customers and also in order to mobilize the funds, plans to reward the customers with discounts on their installments, who pay their loan instalments on time. Bank plans to add a module to their automated system for this purpose.

The Customer Loan Installment details of the Bank are stored in a .txt file (as a comma separated fields). This file contains the details/records of the customers Loans of all categories. The discount is applicable for only those customers who pay their installments at least 20 days in advance with respect to their due dates

The proposed system is supposed to pick up the customer loan  details, who pay their installments in advance and revised installment is calculated based on the business policies defined in the below specifications. After the discounted installment is calculated, the system needs to persist those records in the database.

**Functional Requirement Specification:**

|  |  |  |
| --- | --- | --- |
| **Req. #** | **Req. Name** | **Req. Description** |
| **1** | Parse Input | The input file has to be parsed and Customer’s Loan Installment Records need to be filtered based on the Date of Payment. |
| **2** | Update the Revised Installment and persist the data in the database | The Customer’s discounted Installment has to be calculated and then the modified records need to be saved in the database. |

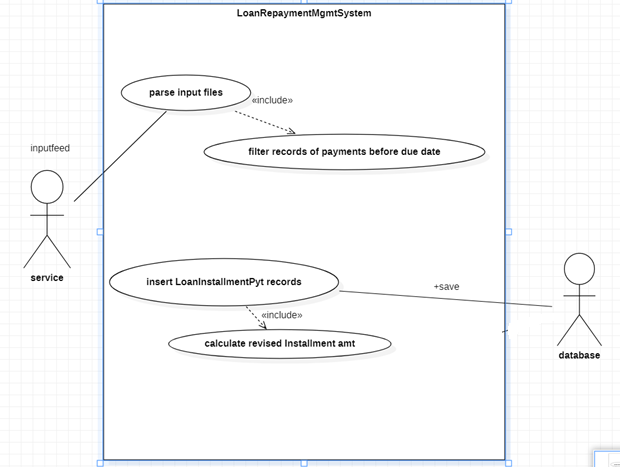
**Skeleton File for Development**

 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.

Skeleton code missing



**Use case Diagram**



**Technical Requirements**

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

1. **A. Component Specification:**

|  |  |
| --- | --- |
| ***Requirement Name*** | 1. **Parse Input** |
| ***Component Definition*** | Reads the input text file, and convert the data into objects |
| ***Files Included***  ***(refer Skeleton)*** | LoanRepaymentService.java, ApplicationUtil.java, LoanInstallmentPyt.java, inputfeed.txt,LoanRepaymentException.java |
| ***Responsibilities*** | Reads the input file, does validation to check if the record is with the actualPytDate lesser than the dueDate, builds the LoanInstallmentPytvalue 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:   <pytId>,<customerId>,<loanId>,<customerName>,<loanType>,<installmentAmtInRs>  <dueDate>,<actualPytDate>,<revisedInstallmentInRs>,<loanAmount>   1. In the input feed, filter LoanInstallmentPyt records which have the actualPytDate lesser than dueDate by 20 days and more. You can identify the records by comparing the dates. 2. Assume that currencies are in INR 3. Assume that the Date fields 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 date field’s value to java.util.date with format, yyyy-MM-dd before setting in LoanInstallmentPytvalue 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 theLoanRepaymentService.addLoanInstallmentPytDetails() method. File location/path must not be hardcoded |
| ***Process Flow*** | 1. The app will be invoked by calling theLoanRepaymentService.addLoanInstallmentPytDetails() with the input feed (.txt file). 2. Read the file using File I/O or Java Streams in ApplicationUtil 3. readFile returns List <String> of records, that were read from the file; It should filter  only records where the payment date is atleast 20  days less than due date and returns, these records.(with each records’ s fields separated by comma) 4. Code the method LoanRepaymentService.buildLoanInstallmentPytList() pass the output of the readFile method to this method as arg. 5. In the method LoanRepaymentService.buildLoanInstallmentPytList() read the list returned by readFile method, split the records based on comma separatorand return the ArrayList of records of LoanInstallmentPyt. 6. Use the ApplicationUtil. convertStringToDate method to convert the date from String Format to java.util.Date format (yyyy-MM-dd). 7. Build the LoanInstallmentPytValue 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, LoanRepaymentException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***ClassName*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| LoanRepaymentService | addLoanInstallmentPytDetails() | String inputFeed | boolean |
| ApplicationUtil | readFile() | String fileName | static List<String> |
| LoanRepaymentService | buildLoanInstallmentPytList() | List <String>loanInstPytRecords | ArrayList<LoanInstallmentPyt> |
| ApplicationUtil | convertStringToDate | String input Date | Date |

**2   A. Component Specification:**

|  |  |
| --- | --- |
| ***Requirement Name*** | 1. **Persist Data into Database** |
| ***Component Definition*** | Helps to calculate the discounted installment and add the changes made to database. |
| ***Files Included***  ***(refer Skeleton)*** | LoanRepaymentService.java, ApplicationUtil.java, LoanInstallmentPyt.java,LoanRepaymentException.java,DBConnectionManager.java, |
| ***Responsibilities*** | Updates LoanInstallmentPyt record’s revisedInstallment amount, based on LoanType and Loan Amount. Persists all Employee 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, LoanRepaymentException. 10. Revise the InstallmentAmt based on the constraints in the table given below      |  |  |  | | --- | --- | --- | | **Loan Type** | **Loan Amount in Rs** | **Discount** | | Housing Loan | 1000000 - 2500000 | 10% | | Housing Loan | 2500001 - 5000000 | 12% | | Housing Loan | >5000000 | 14% | | Personal Loan | 50000 - 100000 | 8% | | Personal Loan | 100001 - 500000 | 10% | | Personal Loan | 500001 - 1000000 | 12% | | Vehicle Loan | 100000 - 500000 | 8% | | Vehicle Loan | 500001 - 1000000 | 10% | | Vehicle Loan | >1000000 | 12% | | Educational Loan | 100000 - 500000 | 11% | | Educational Loan | 500001 - 1000000 | 12% | | Educational Loan | >1000000 | 13% | |
| ***Resources*** | database.properties – has connection details, used to establish database connection. |
| ***Process Flow*** | 1. Modify the LoanRepaymentService.buildLoanInstallmentPytList() method (refer the above section) then set revisedInstallmentAmt to LoanInstallmentPytobjects. 2. Use LoanRepaymentService.calculateDiscountedInstallment() method to calculate revisedInstallmentAmt to the currentInstallmentAmt passed as parameter, based on loanType and loanAmount. 3. The method LoanRepaymentService.buildLoanInstallmentPytList()   must return the list of LoanInstallmentPyt objects with revisedInstallmentAmt calculated 4. After reading file, building records as List<LoanInstallmentPyt>, call the LoanInstallmtDAO. insertLoanInstallmentPytmethod to insert values to database. You may have to convert the java.util.date to java.sql.date before storing to database.      1. If Insert has happened successfully, return true; false otherwise. |
| ***Exceptional Conditions*** | While working with DAO methods, catch all exceptions and throw application specific exception, LoanRepaymentException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***ClassName*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| LoanRepaymentService | calculateDiscountedInstallment() | String loanType,doubleloanAmount,doublecurrentInstallmentAmt | double |
| LoanRepaymentService | buildLoanInstallmentPytList | List <String>loanInstPytRecords | ArrayList<LoanInstallmentPyt> |
| DBConnectionManager | DBConnectionManager() | NA | NA |
| DBConnectionManager | get Instance() | NA | DBConnectionManager |
| LoanInstallmtDAO | insertLoanInstallmentPyt | ArrayList<LoanInstallmentPyt>loanInstlmtPyts | 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.

**Sample Input Data:**

P001,C001,L001,KeerthiKumar,HousingLoan,20345,2020-01-25,2020-01-04,,2000000

P002,C002,L002,KumarMartha,HousingLoan,33350,2020-02-28,2020-02-07,,3000000

P003,C003,L003,MaheshKumar,VehicleLoan,13000,2020-01-30,2020-01-05,,1200000

P004,C004,L004,RajeshKumar,EducationalLoan,7500,2020-02-20,2020-01-25,,800000

P005,C005,L005,Rakesh,HousingLoan,25800,2020-03-25,2020-03-04,,2500000

P006,C006,L006,Sudarshan,PersonalLoan,9400,2020-01-25,2020-01-01,,800000

**Sample Output Data:**

P001 C001 L001 KeerthiKumar HousingLoan 20345 2020-01-25 2020-01-04 18311 2000000

P002 C002 L002 KumarMartha HousingLoan 33350 2020-02-28 2020-02-07 29348 3000000

P003 C003 L003 MaheshKumar VehicleLoan 13000 2020-01-30 2020-01-05 11440 1200000

P004 C004 L004 RajeshKumar EducationalLoan 7500 2020-02-2020 20-01-25 6600 800000

P005 C005 L005 Rakesh HousingLoan 25800 2020-03-25 2020-03-04 23220 2500000

P006 C006 L006 Sudarshan PersonalLoan 9400 2020-01-25 2020-01-01 8272 800000