# **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**

The City corporation plans to offer discounts to the loyal property owners in the city who pay the property taxes on time. The discount is calculated based on the category of the property and the area of the property. Corporation wants to automate the process of taxation &calculation of discounted tax.

The property tax details of citizens of the city are stored in a .txt file (as a comma separated fields). This file contains the details/records of the properties/tax liabilities of both commercial and residential properties. But the discount is applicable only to the citizens who pay the taxes before the stipulated deadline.

The proposed system is supposed to pick up the citizens’ property details , whose actual payment dates are before the stipulated payment dates (which are pre-defined), and then the revised Tax Liability is calculated based on the city corporation administrative policy, as mentioned in the below requirement specification. After the Revised Tax is applied , the system needs to persist those records in the database.

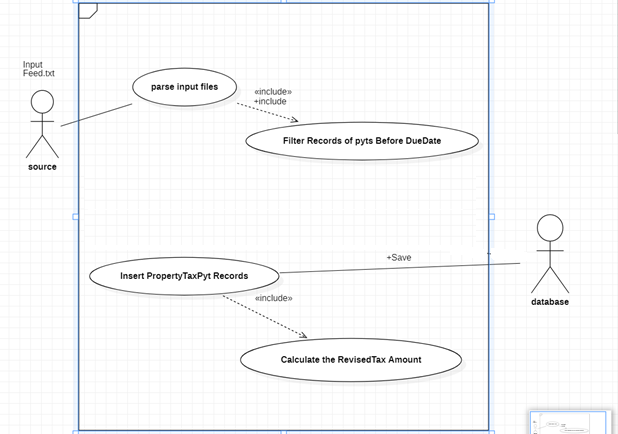
1. **Functional Requirement Specification:**

|  |  |  |
| --- | --- | --- |
| **Req. #** | **Req. Name** | **Req. Description** |
| **1** | Parse Input | The input file has to be parsed and Property records need to be filtered based on the payment dates |
| **2** | Update the RevisedTax and persist the data in the database | The property owner’s Property tax has to be revised based on the rules and then the modified records need to be saved in 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 the above functional requirements 1 & 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)*** | PropertyTaxationService.java, ApplicationUtil.java, PropertyTaxPyt.java, inputfeed.txt,PropertyTaxPytException.java |
| ***Responsibilities*** | Reads the input file, does validation to check if the record is with the dateOfPayment lesser than the dueDate, builds the PropertyTaxPytvalue 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:   <propertyId>,<ownersName>,<areaInSqFt>,<propertyType>,  <taxAmount>,<dateOfPyt>,<dueDate>   1. In the input feed, filter propertyTaxPyt records which have the dateOfPyt lesser than dueDate. You can identify the records by comparing the dates. 2. Assume that the taxAmount&revisedtaxAmts 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 values to java.util.date with format, yyyy-MM-dd before setting in PropertyTaxPytvalue 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 PropertyTaxationService.addPropTaxDetails method. File location/path must not be hardcoded |
| ***Process Flow*** | 1. The app will be invoked by calling the PropertyTaxationService.addPropTaxDetails with the inputfeed (.txt file) 2. addPropTaxDetails calls the readFile() method by passing the file to it as arg. 3. Read the file using File I/O or Java Streams in ApplicationUtil.readFile method 4. readFile returns List <String> of records, that were read from the file; It should filter only records where the payment date is less than due date and returns, these records.(with each records’s fields separated by comma) 5. Code the method PropertyTaxationService.buildPropertyTaxPytList. pass the output of the readFile method to this method 6. In the method PropertyTaxationService.buildPropertyTaxPytList read every line from the list returned by readFile method, split the records based on comma separator and return the ArrayList of records of PropertyTaxPyt. 7. Use the ApplicationUtil. convertStringToDate method to convert the date from String format to java.util.Date format (yyyy-MM-dd). 8. Build the PropertyTaxPytValue 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, PropertyTaxPytException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***ClassName*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| ApplicationUtil | readFile | String fileName | static List<String> |
| PropertyTaxationService | buildPropertyTaxList | ArrayList<String> records | ArrayList<PropertyTaxPyt> |
| PropertyTaxationService | addPropTaxDetails | String inputFeed | boolean |
| ApplicationUtil | stringToDateConverter | String input Date | Date |
| ApplicationUtil | checkIfDateOfPytIsLessThanDueDate | Date dateOfPyt,Date dueDate | boolean |

1. **A. Component Specification:**

|  |  |
| --- | --- |
| ***Requirement Name*** | 2. Persist Data into Database |
| ***Component Definition*** | Helps to calculate the revisedTaxAmt and add the changes made to database. |
| ***Files Included***  ***(refer Skeleton)*** | PropertyTaxationService.java, ApplicationUtil.java, PropertyTaxPyt.java, DBConnectionManager.java, PropertyTaxPytException |
| ***Responsibilities*** | Updates the propertyTax based on type of the property and area of the property. Persists all PropertyTaxPytdetails 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 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, PropertyTaxPytException 10. Calculate the taxDiscount based on the constraints in the table given below  |  |  |  | | --- | --- | --- | | **Property Type** | **AreaInSqFt** | **Discount** | | Commercial | <=1000 | 7% | | Commercial | 1001 to 10000 | 10% | | Commercial | >10000 | 13% | | Residential | <=1000 | 5% | | Residential | 1001 to 10000 | 9% | | Residential | >10000 | 11% | |
| ***Resources*** | database.properties – has connection details, used to establish database connection. |
| ***Process Flow*** | 1. Modify the PropertyTaxationService.buildPropertyTaxPytListmethod (refer the above section) then set revisedTaxAmt to propertyTaxPytobjects. 2. Use EmployeeAdminService .calculateTaxDiscount() method to calculate revisedtaxAmtto the taxAmountpassed as parameter, based on propertyType and AreaInSqFt. 3. The method EmployeeAdminService .buildPropertyTaxPytList must return the list of PorpertyTaxPytwith revisedTaxAmt calculated 4. After reading file, building records as List<PropertyTaxPyt>, call the PropertyTaxDAO. insertPropertyTaxPyt method to insert values to database. You may have to convert the java.util.date to java.sql.date before storing to database. If Insert has happened successfully, return true; false otherwise. 5. Use getPropertyTaxPyt() method to fetch the Updated records and display |
| ***Exceptional Conditions*** | While working with DAO methods, catch all exceptions and throw application specific exception, PropertyTaxPytException. |

1. **B. Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***ClassName*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| PropertyTaxationService | calculateTaxDiscount() | String propertyType,double areaInSqFt,double currentTaxAmt | double |
| PropertyTaxationService | buildPropertyTaxList | ArrayList<String> records | ArrayList<PropertyTaxPyt> |
| DBConnectionManager | DBConnectionManager() | NA | NA |
| DBConnectionManager | get Instance() | NA | DBConnectionManager |
| PropertyTaxDAO | insertPropertyTaxPyt() | ArrayList<PropertyTaxPyt>propertyTaxPyts | 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.