**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 business requirement.*
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**

The “Survivor” hospital has decided to give bonus for all permanent employees who are working extra time in hospital besides their normal shift in this pandemic situation. The hospital management find it very difficult to deal with the total salary calculation based on their levels and extra working hours. So, they decides to outsource the salary calculation process to their software consultant. You being their software consultant automate the above task.

The Hospital Management has the following business processes that must be automated.

1. Parse data and calculate total salary for all the permanent employees
2. Store the employee details of each permanent employee
3. **FUNCTIONAL REQUIREMENT SPECIFICATION**

|  |  |  |
| --- | --- | --- |
| Sl No. | Business Requirement | Business Description |
| 1 | Parse Input | The input file has to be parsed and the permanent Employee’s details need to be filtered |
| 2 | Insert the validated records into the database | The valid employee records are inserted into the database |

1. **SKELETON FILE FOR DEVELOPMENT**

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

1. **TECHNICAL REQUIREMENTS**

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

**D.1.1 Component Specification:**

|  |  |
| --- | --- |
| ***Requirement Name*** | **Parse Input** |
| ***Component Definition*** | Reads the input text file, and converts the data into objects |
| ***Files Included***  ***(refer Skeleton)*** | HospitalManagement.java, ApplicationUtil.java, EmployeeDetails.java, inputfeed.txt, InvalidEmployeeNumberException.java |
| ***Responsibilities*** | Reads the input file, filters the records after validation of employee number and then builds the EmployeeDetails object and returns it. |
| ***Design Constraints*** | * The 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 the main method) * Do not hard code the input file path inside any method – has to be used from the input argument only as per the code skeleton.   File Structure is similar to what is mentioned below:  <Employee Number>, <Employee Name>, <Level >, <Extra Working Hours> |
| ***Resources*** | inputfeed.txt is the input file that must be parsed. The file, along with the file location will be sent as an argument to the addEmployeeList() method of Hospital Management class. File location/path must not be hardcoded. |
| ***Process Flow*** | * The application will be invoked by calling the addEmployeeList() with the input feed (.txt file). * addEmployeeList() calls the readFile() * Read the file using File I/O or Java Streams in ApplicationUtil.readFile() and returns List <String> of records, that was read from the file; It should filter based on the valid Employee number. * ApplicationUtil.validate() is used to validate the employee number.   **Validation:**  The employeeNumber should start with PR followed by 5 digits and it should be 7 in length. If the employeeNumber is valid then parse the data and calculate the salary else throw a user defined Exception “InvalidEmployeeNumberException” with a message "Invalid Employee Number". The file can contain details of temporary employees also which are not considered and treated as invalid.   * Code the method HospitalManagement.buildEmployeeList() pass the output of the readFile method to this method as an argument. * In the method HospitalManagement.buildEmployeeList() read the list returned by the readFile method, split the records based on a comma separator, calculate total salary from calculateTotalSalary() and then return the ArrayList of records of EmployeeDetails. * Build the EmployeeDetails 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  including application specific InvalidEmployeeNumberException . |

**D.1.2 Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| HospitalManagement | addEmployeeList () | String inputFeed | boolean |
| ApplicationUtil | readFile() | String filePath | static List<String> |
| ApplicationUtil | validate() | String employeeNumber | static boolean |
| HospitalManagement | buildEmployeeList () | List <String> employeeRecords | static ArrayList<EmployeeDetails> |

**D.2.1 Component Specification:**

|  |  |
| --- | --- |
| ***Requirement Name*** | 1. ***Persist Data into Database*** |
| ***Component Definition*** | ***Helps to drop the records to the database*** |
| ***Files Included***  ***(refer Skeleton)*** | HospitalManagement.java, ApplicationUtil.java, EmployeeDetails.java, inputfeed.txt, InvalidEmployeeNumberException.java,DBConnectionManager.java, DetailsDAO.java |
| ***Responsibilities*** | Updates EmployeeDetails records based on the validation of the employee number. Inserts all the employee records validated to the database. |
| ***Design Constraints*** | * The database.properties has connection details required to connect to the backend * 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 the local settings. * Use only JDBC to establish a Database connection * Assume that the location of the property file will be always as given in the skeleton. * Don’t hardcode the connection string to establish a database connection. Read it from the property files. * Use a Prepared Statement to insert records * Close all the resources after use * Catch all the database related exception and throw an Application specific exception only from the DAO or from the DBConnectionManager class. There has to be a private constructor in the DBConnectionManager class, to load the database property file and to establish a database connection using JDBC * Rollback the Insert if any SQL exception has occurred. Throw an application specific exception, InvalidEmployeeNumberException.   **Calculation of total salary:**   |  |  | | --- | --- | | **Levels** | **Basic Salary** | | **level1** | 75000 | | **level2** | 50000 | | **level3** | 35000 | | **level4** | 25000 |   **Note: Total salary can be calculated by adding basic salary and 1000 rs per extra working hour for their respective levels**  **For example:** If the extra working hours is 13 for a permanent employee of level3 , then the total salary is calculated by adding basic salary (here for level3 35000) and 1000 per extra working hour( 13\*1000)  **Total salary = 35000+(13\*1000)**  **=35000+13000**  **=48000** |
| ***Resources*** | database.properties – has connection details, used to establish database connection. |
| ***Process Flow*** | * Modify the HospitalManagement.buildEmployeeList() method set the EmployeeDetails objects after calculating the total salary using HospitalManagement.calculateTotalSalary(). * After reading the file, building records as ArrayList<EmployeeDetails>, call the DetailsDAO.insertEmployeeList method to insert values to the database. The database will contain only validated employee records. * If the insert has happened successfully, return true; otherwise, false. |
| ***Exceptional Conditions*** | While working with DAO methods catch all exceptions and throw an application specific exception, InvalidEmployeeNumberException. |

**D.2.2 Method Specification:**

|  |  |  |  |
| --- | --- | --- | --- |
| ***Class Name*** | ***Method Name*** | ***Input Parameters*** | ***Output Parameters*** |
| HospitalManagement | buildEmployeeList | List <String> employeeRecords | ArrayList<EmployeeDetails> |
| DBConnectionManager | DBConnectionManager() | NA | NA |
| DBConnectionManager | getInstance() | NA | static DBConnectionManager |
| HospitalManagement | calculateTotalSalary() | String level, int extraWorkingHours | static double |
| DetailsDAO | insertEmployeeList () | List <EmployeeDetails> eList | boolean |

**Note:** You are allowed to modify the 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 the PMD rules in Eclipse or any other IDE that you use.

1. **SAMPLE INPUT (inputfeed.txt):**

PR47856,Steven,level1,24

TR87965,Sujatha,level1,12

PR78965,Seetha,level3,12

PR7895,Vaishnav,level2,6

PR78995,Yamuna,level2,25

PR78999,John,level4,13

TR88888,Wilsy,level4,16

PR12356,Nelson,level1,27

PR85203,Mithun,level3,11

1. **SAMPLE OUTPUT(IN DB)**

PR47856,Steven,level1,24,99000

PR78965,Seetha,level3,12,47000

PR78995,Yamuna,level2,25,75000

PR78999,John,level4,13,38000

PR12356,Nelson,level1,27,102000

PR85203,Mithun,level3,11,46000