**Project Assessment**

**GENERAL INSTRUCTIONS:** Please carefully read the below instructions

The objective of this assessment is to check your ability to complete a project as per the provided "Project Design".

**You are expected to –**

1. Write the source code for the classes, methods and packages **EXACTLY** as mentioned in the "**Project Design**" section.

2. Ensure that the names of the packages, classes, methods and variables **EXACTLY MATCH** with the names specified in the "Project Design" section.

3. Understand the project requirements and ACCORDINGLY WRITE the code and logic in the classes and methods so as to meet all given requirements.

**Creating the project and testing it –**

1. You are expected to create your project locally using eclipse (or any other IDE) on your desktop.

2. Once you are ready with the code, you should upload the src folder of your project in .zip format, using the "Upload Zip File" button.

   IMPORTANT NOTE 1 : The extension of the zip file should be ONLY .zip (any other zip formats such as .7z will produce unexpected results)

   IMPORTANT NOTE 2 : The .zip file should contain zip of ONLY the src folder structure from your project. (If the zip file has anything other than the src folder structure, the result will be unexpected. Do not zip the entire project folder structure. Just do the zip of the src folder structure and upload it)

   IMPORTANT NOTE 3 : The name of the .zip file should be <your employee number>.zip For e.g., if your emp no. is 12345, the zip file should be named 12345.zip.

3. After uploading the zip file, you can click on "Compile & Test" button and the assessment engine will compile your source code and test it using its pre-defined test-cases.

4. If some of the test-cases fail, you can make the fixes in your source code locally on your desktop, and again repeat the above two steps.

5. Once you are finished with all the fixes, you can click on "Final Submission" button, which will show you the final result/score.

**NOTE that –**

6. The assessment engine will create objects and invoke methods as per the project design, and while doing so, it will use your packages, classes and methods. If your packages, classes and methods have a name mismatch or method prototype mismatch w.r.t the expected "Project Design", the tool will show it as an ERROR. If your packages, classes and methods match as per the names but do not perform the expected functionality, the tool will show it as a FAILURE.

7. Unless specified in the Project Design, DO NOT use **System.exit(0)** anywhere in your code. Using **System.exit(0)** in your project code will cause the CPC test engine to exit and it will not be able to run all test-cases.

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

# Electricity Bill Calculation

## Project Objective:

Create a console based Java application that would allow an electricity board clerk to compute the electricity bill amount that needs to be paid by the customer for a given type of electricity connection

## Details:

There are two types of electric connections available. One is Domestic Connection and the other one is Commercial Connection. The following are the formulas that are to be used for computation of Bill for each type.

**Domestic**

| Unit Slabs | Tariff Rate |
|---|---|
| First 50 units | 2.3 |
| Next 50 units | 4.2 |
| Remaining units | 5.5 |

Eg) If units consumed is 120, then amount payable is 435.

**Commercial**

| Unit Slabs | Tariff Rate |
|---|---|
| First 50 units | 5.2 |

| | |
|---|---|
| Next 50 units | 6.8 |
| Remaining units | 8.3 |

In Commercial Connection type, in addition to the bill amount there is an electricity duty that is applicable. The calculations for the electricity duty is as follows:

| BillAmount | Electricity Duty |
|---|---|
| Bill Amount >= 10000 | 0.09 |
| Bill Amount >=5000 | 0.06 |
| Bill Amount < 5000 | 0.02 |

For example, if Bill Amount>10000 then

**Electricity Duty=Bill Amount* 0.09**

So the **Final Amount Payable =Bill Amount + Electricity Duty**

Eg) If units consumed is 120, then amount payable is 781.32 (i.e. 766 + 15.32)

**Project Design:**

**A. System Design:**

| **Name of the package** | **Usage** |
|---|---|
| com.wipro.eb.entity | This package will contain the EB Connection related classes |
| com.wipro.eb.exception | This package will contain the user defined exception classes |
| com.wipro.eb.main | This package will contain the MainClass that is used to test the application |
| com.wipro.eb.service | This package will contain the class that is used to validate the data and invoke the respective EB Connection Classes to calculate the bill amount |

**Package: com.wipro.eb.exception**

| Class | Method and Variables | Description |
|---|---|---|
| **InvalidReadingException** | | A user defined exception class. Details about when this exception is given in the respective methods |
| | **public String toString()** | This function should return<br><br>"**Incorrect Reading**" |

**Package: com.wipro.eb.exception**

| Class | | Description |
|---|---|---|
| **InvalidConnectionException** | | A user defined exception class. Details about when this exception is given in the respective methods |
| | **public String toString()** | This function should return<br><br>"**Invalid ConnectionType**" |

**Package: com.wipro.eb.entity**

| Class | Method and Variables | Description |
|---|---|---|
| **Connection** | | **Abstract Class** |
| | int previousReading; | Previous month meter reading |
| | int currentReading; | Current month meter reading |
| | float[] slabs; | Used to store different slab rate |
| | public Connection(int currentReading, int previousReading,float slabs[]) | A constructor used to initialize the member variables |
| | public abstract float computeBill() | An abstract method to compute the bill for a particular month |
| | Getter and Setter methods for all the member variables | Getter and Setter methods for all member variables |

**Package: com.wipro.eb.entity**

| Class | Method and Variables | Description |
|---|---|---|
| **Domestic** | | **Inherits Connection Class** |
| | public Domestic(int currentReading, int previousReading,float slabs[]) | **A parameterized constructor** |
| | public float computeBill() | Should compute the amount to be paid by the customer for the given reading. Use the formula that is given at the beginning of the case study |

**Package: com.wipro.eb.entity**

| Class | Method and Variables | Description |
|---|---|---|
| **Commercial** | | Inherits Connection Class |
| | public Commercial(int currentReading, int previousReading,float slabs[]) | **A parameterized constructor** |
| | public float computeBill() | Should compute the amount to be paid by the customer for the given reading. Use the formula that is given at the beginning of the case study |

**Package : com.wipro.eb.service**

| Class | Method and Variables | Description |
|---|---|---|
| **ConnectionService** | | **Class** |
| | public boolean validate(int currentReading, int previousReading, String type) throws InvalidReadingException, InvalidConnectionException | If the **currentReading is less than previousReading** or if **any of the readings are negative** then the function should throw **InvalidReadingException**<br><br>If the type is **anything other than Domestic or Commercial** the function should throw **InvalidConnectionException**<br><br>If all the 3 data are valid the |

| | | |
|---|---|---|
| | | function should return true |
| | public float calculateBillAmt(int currentReading, int previousReading, String type) | This method will invoke the validate method to check whether all the 3 inputs received are valid<br><br>If the validate method throws InvalidReadingException, this function should handle it and return -1<br><br>If the validate method throws InvalidConnectionException, this function should handle it and return -2<br><br>If the validate method returns **true** this function should create appropriate Connection type object and invoke the computeBill method and return the computed billamount.<br><br>[If type is "Domestic", then the Domestic object needs to be created If type is "Commercial", then the Commercial object needs to be created.] |
| | public String generateBill(int currentReading, int previousReading, String type) | This method should invoke the calculateBillAmt method.<br><br>If the return value of calculateBillAmt is -1, this method should return **"Incorrect Reading"**<br><br>If the return value of calculateBillAmt is -2, this method should return **"Invalid ConnectionType"**<br><br>Else this method should return a String in the following format. Suppose the return value of calculateBillAmt is 256, then this function should return **"Amount to be paid:256"** |

**Package : com.wipro.eb.main**

| Class | Method and Variables | Description |
|---|---|---|
| **EBMain** | | **Main Class** |
| | public static void main(String[] args) | Get the following input from the user<br><br>Get the previous month reading<br><br>Get the current month reading<br><br>Get the Connection Type<br><br>After receiving all this data, invoke the generateBill method of ConnectionService class present in com.wipro.eb.service package and test your program |

**The main method of the EBMain Class may look like this:**

```
public static void main(String a[])

{

System.out.println(new ConnectionService().generateBill(130,100,"Commercial"));

}
```

Test your program with different inputs.

**TestCases:-**

1. Test calculateBillAmt() with all valid values for Domestic type connection where the consumed units are below or equal to 50

2. Test calculateBillAmt() with all valid values for Domestic type connection where the consumed units are between 50 and 100

3. Test calculateBillAmt() with all valid values for Domestic type connection where the consumed units are equal or above 100

4. Test calculateBillAmt() with all valid values for Commercial type connection where the consumed units are below or equal to 50

5. Test calculateBillAmt() with all valid values for Commercial type connection where the consumed units are between 50 and 100

6. Test calculateBillAmt() with all valid values for Commercial type connection where the consumed units are equal or above 100

7. Test validate() method with all valid values

8. Test validate() method for Invalid Reading Exception

9. Test validate() method for Invalid Connection Type Exception

10. Test generateBill() method with all valid values and invalid values