CPRG 200 Lab Assignment 2

You may discuss your approach to solving this problem with other students, or ask for advice if stuck, but ultimately you should develop all the code on your own. You may use any code that you have previously developed or that was posted in this course.

Problem Description

You were asked to expand the application for the city power company to store in a file calculated charge amounts for multiple customers. In addition to the data that you collect from each customer you will need to also collect a name and account number. You will be storing for each customer the following data:

- AccountNo – a positive integer value
- CustomerName – a non-empty string (there is no restriction on what characters a name can be made of; some companies have names that include numbers or even punctuation characters)
- CustomerType – a single character string that can be "R", "C", or "I" (capital letters)
- ChargeAmount (a numeric value, not a string)

<u>You do NOT need to store kWH values for the customers</u>.
As the new customers' data is added, or before the application closes, the current collection of customer data needs to be written to the file. When the application starts next time, data is read from this file to allow more adding more customers.
Your application should display at all times the current collection of customer's data as listed above. In addition, you were asked to calculate and display the following statistics:

- the total number of customers,
- the sum of charges for each customer type (three values), and
- the sum of all charges.

Step 1: Define class Customer

Add Class Library project named *CustomerData* to your Lab 1 solution. Make this project in your solution. In CustomerData project, define a class *Customer* that represents a customer. At the minimum, the class should have:

- public properties for the data listed above,
- a constructor or constructors,
- method CalculateCharge that calculates the charge amount for this customer according to the rules, and
- ToString() method that returns a display string.

Optional challenge: You may create three additional classes derive from class Customer: ResidentialCustomer, CommercialCustomer, and IndustrialCustomer. The Inherited CalculateCharge method will need to be redefined in each class to apply specific business rules for each type of customer.

Step 2: Test class Customer

Add a Unit Test project for the Customer Class. Include in it a minimum of 6 test methods that test CalculateCharge: 2 for each customer type. Make sure all tests pass before you proceed with integrating the class library with your Lab 1 project.

Step 3: Integrate Class Library with your Lab 1 project

Make your Lab 1 project a start project in the solution. Add in it a reference to CustomerData class library. Create a list of Customer objects on the form-level.

Augment the Form from Lab 1:

- Add controls to allow the user to enter for each customer the account number and the name, as requested.
- Include Add button that will trigger adding another customer's data.
- Add a list box or another control were data of all customers will be displayed, one customer per line.
- Add controls to display the statistics listed above.

No more buttons should be added. In particular reading data from the file and saving data to the file should happen automatically without the user having to click extra buttons. The file should be named *Customers.txt* and placed in the *bin/Debug* folder of the project. Data is read when the form loads (watch for the situation when you are running the application the very first time when the file does not exist – your application must not crash). Saving of data to the file may happen as the user clicks Add button or just before the form closes – it's up to you. Make sure that you include the Customers.txt file when you submit.

Adjust the code of your project so that it stores added customers in the list of customers and calculates the required statistics. Note that for each customer you will store the calculated charge, but not any of the kWh usage data.

General requirements

Present for marking only the resulting application after all work has been completed.

Like in Lab 1, the look of the form is up to you, but professionalism and clarity/easiness to use will be a factor in marking.

Required comments are the same as for Lab 1:

- Top block comments on each code file that explains purpose of the module or class, date when created (at least month and year), and the author's name
- Each method has a comment that explains its purpose
- Each variable has a comment that explains its meaning
- Each group of statements that perform a single task has a comment that explains the task

Zip the entire folder with your application and submit to the appropriate Dropbox. Make sure that the name of the folder includes your name and *CPRG200_Lab2*, e.g. *Bob_Smith_CPRG200_Lab2*. The assignment must be submitted before the due date. Unless an important reason can be documented, late penalty deductions of 20% per day will be applied.

Marking Scheme

Marking Component	**Out of**
Solution has three project: Class Library project, Unit Test project, and Windows Forms project	2
Customer class is defined and has public properties, constructor(s), CalculateCharge method, and ToString() method	3
Unit test project has a minimum of 6 test methods, and all tests pass	6
Form has all necessary controls, looks professional, and is easy to use	2
Customer data is read correctly from the text file Customers.txt that is placed in the bin/Debug folder of the project	3
Application does not crash when it runs the first time when the file does not exist	1
Customers' data is stored in the application in a list where each element is an object of class Customer	2
As new customers are added, or before the application closes, the current collection of customers' data is saved to the file	3
Application displays an up to date list of customers data	2
Application maintains and displays the current number of customers and total sum of charges	2
Application maintains and displays the sum of charges for each customer type (three values)	2
Code is clear, well-structured (uses methods, minimizes repetition of code), uses good naming practices, and has comments as required	2
Total:	**30**