Object-oriented programming

Laboratory classes

Task 1: Contact Manager

The goal of this task is to create a simple contact manager application.

1. Run the Visual Studio IDE, and then create a new console application project in C#. Name it *Task1*.
2. Add to the project a new class representing a person. To do this, right-click the project name in *Solution Explorer* window and select either *Add->New Item->Class* or *Add->Class*. Name the class *Person* and confirm the selection (cs extension will be added automatically).
   ☞ Note that the *Person* class is automatically created in the *Person.cs* file. Although C# doesn't enforce the "one class per file" rule, it's a good convention that you should stick to for code manageability – each (public) class stored in a separate file with the same name.
3. Add the *firstName* (note: camelCase) field of type *string* to the *Person* class.
4. In the *Main()* function of the program, declare a variable of type *Person*, and then assign a new object of the same class to it. Try to assign your given name to the person you created, then build a project (Ctrl+Shift+B). Was the build successful?
5. Read the error message. What is the reason for this situation?
6. Add the appropriate access modifier to the *firstName* field. By the way, change the field name to start with a capital letter: *FirstName* (PascalCase).
   ☞ This is another important naming convention: use *camelCase* to name private members (fields, methods, properties), and *PascalCase* to name public members.
7. In the *Main()* function, try again to set the first name of the person you created.
8. Add in the *Main()* function the statement to output the first name on the screen and run the program (Ctrl + F5).
9. Add a field to the class to store the family name (*FamilyName*), then assign your family name to the person object you created and output fields on the screen in the following format:
   `My name is <FirstName> <FamilyName>.`
   ☞ Some experts recommend accessing class members using the keywords *this*, what makes easier to distinguish references to class instance members from references to local parameters and variables.
10. Add the *IntroduceYourself()* method in the *Person* class that returns the above text, and then use it in the *Main()* function of the program to output your person's data.
11. In the *Person* class, add the *SetData()* method, which takes the first name and the surname as parameters and sets the values of the corresponding fields. Use the method in the *Main()* function of the program.
    ☞ Next naming convention: use *camelCase* to name method parameters.
12. Replace the fields in the *Person* class with properties so that the names can only be set from inside the class but read in any context.
13. Add a new class *PostalAddress* to the project. The class will store street name, house number, apartment number, post code, city and country names. In addition, it should have a read-only property returning the address to be read in "postal" format, i.e.:
    ```
    11/1 StreetName
    City
    POSTCODE
    Country
    ```

14. In the *Person* class, add the *Address* property to store the person's address.
15. Adapt the *IntroduceYourself()* and *SetData()* methods to the new form of the class.
16. Add a new read-only property in the *Person* class returning the full postal address. Use the previously created property in the *PostalAddress* class.
17. Adapt the code in the *Main()* function to the new definition of the *Person* class.
18. Modify the console application to work as an address book manager. After starting the application, the user is first asked for the number of contacts to be entered, and then for remaining data of successive persons. After entering all the data, a list of all contacts is displayed in the format:
    `N. <FirstName> <FamilyName>`
    where *N* is an ordinal number. The user must enter the number corresponding to the given contact and confirm the selection with the *Enter* key to display detailed postal address of this contact. Next *Enter* key returns to the list of all contacts.