Customer Registry

1. Objectives

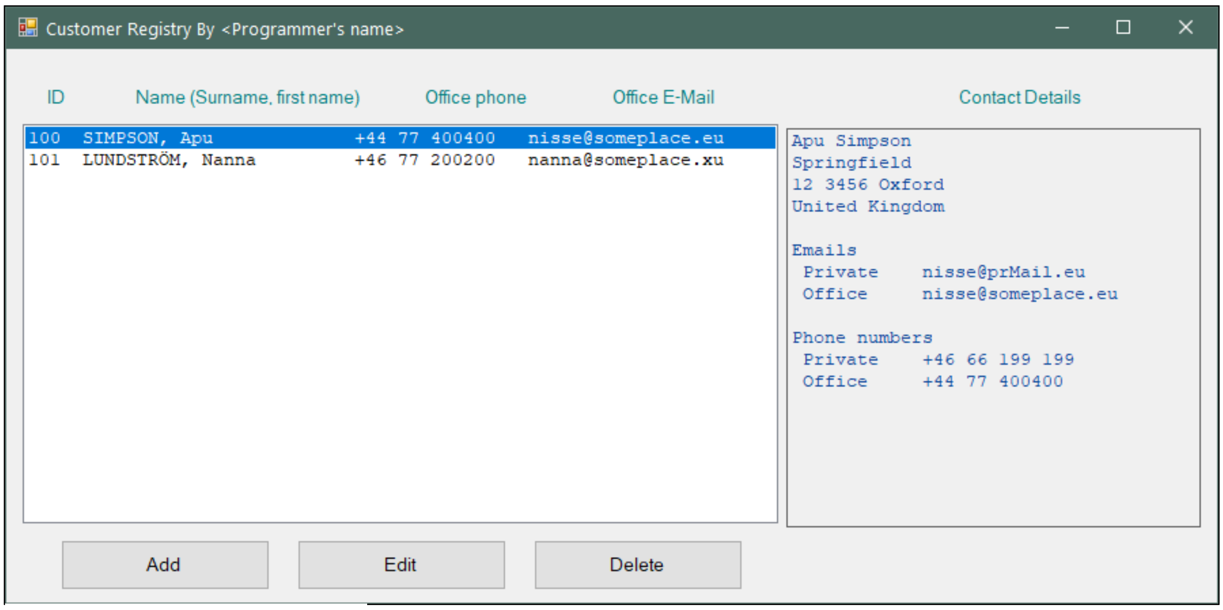
The main objectives of this assignment are to take a first step into the world of the object-oriented programming (OOP) by learning to work with one of the essential aspects of OOP, namely encapsulation. We will also practice data-hiding to complete the process of encapsulation. This assignment covers the following concepts:

- Encapsulation
- Constructors
- “Has a” relation between objects
- Properties
- Array of objects

In this assignment, it is possible to get a pass grade C as well as a higher grade B or A. For a grade C, the user should be able to add customers, i.e. the **Add** button must work satisfactorily. The **Edit** and **Delete** functionalities are required for grades A and B. The grades A or B will be decided by the instructor judging the code quality, structure, and documentation of code.

2. Description

Companies as well as organizations usually maintain information about their members. Members can be customers, students, or other type of individuals, which can collectively be considered as customers. The following is a run-time example of the application that you are going to develop in this assignment.


Customer Registry By <Programmer's name>
ID
Name (Surname, first name)
Office phone
Office E-Mail
Contact Details
Add
Edit
Delete

```
100  SIMPSON, Apu          +44 77 400400   nisse@someplace.eu
101  LUNDSTRÖM, Nanna      +46 77 200200   nanna@someplace.xu
```

```
Apu Simpson
Springfield
12 3456 Oxford
United Kingdom

Emails
 Private    nisse@prMail.eu
 Office     nisse@someplace.eu

Phone numbers
 Private    +46 66 199 199
 Office     +44 77 400400
```

Figure 1 – GUI example

Figure 1 shows an example of the main GUI. The box at the left displays the list of the registered customers with partial data (for example ID and names and some other data) that are currently registered. GUI is kept updated as data is added, changed or deleted.

The application maintains a collection of customers where every customer has contact data. This way we can divide the development work into two parts:

1. Handling the contact data for a customer (**Contact** class)
2. Handling a collection of customers (**Customer** and **CustomerManager** classes)

Functionalities and Features:

2.1 The application should provide the following functionalities:

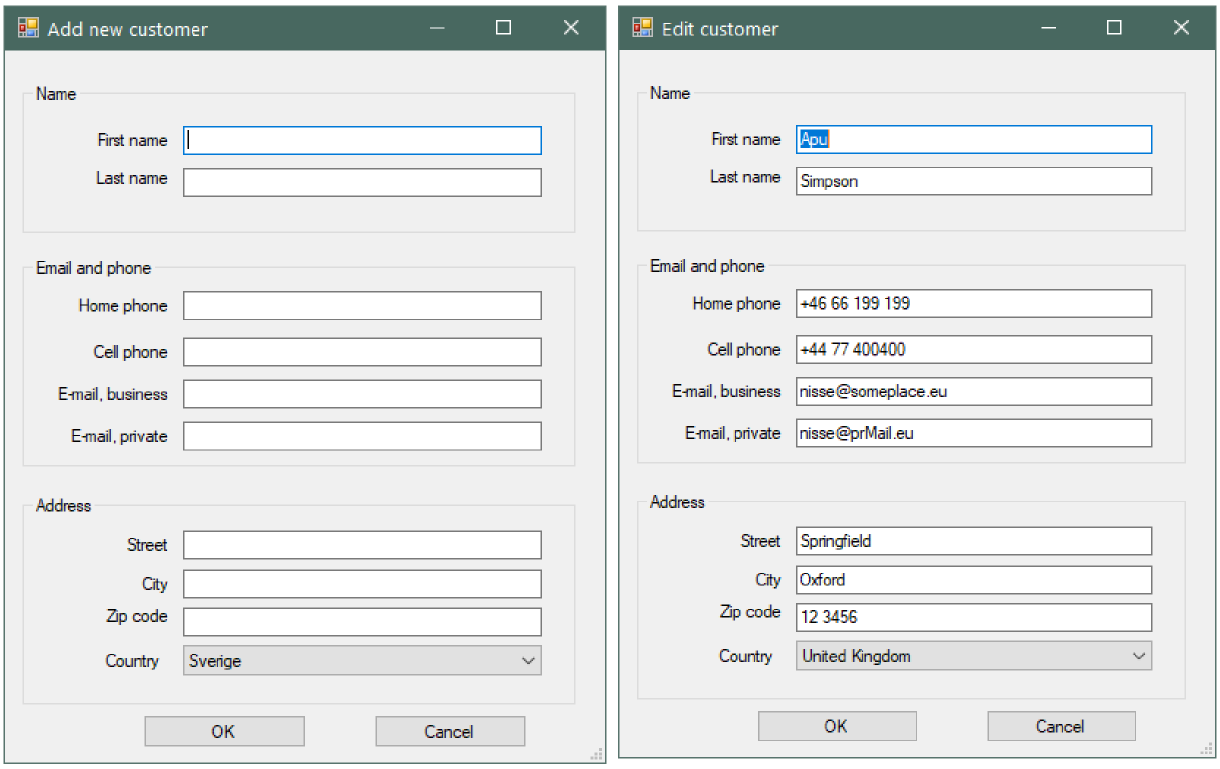
- Add a new customer
- Change data for an existing customer
- Remove an existing customer

2.2 Every customer has an ID and a contact data which includes:

- First name and last name
- An address, street, zip code, city and country.
- A number of phones, at least two, a private phone and an office phone.
- A number of emails, at least two, a private email and an office email.

2.3 The contact data is to be given through a separate user interface that allows the user to add input all the above data.

2.4 When the user clicks on the **Add** or **Edit** buttons on the main GUI (Figure 1), a form for reading input should be displayed as in the figures below. Both cases use the same form.



Add new customer

Name

First name	
Last name	

Email and phone

Home phone	
Cell phone	
E-mail, business	
E-mail, private	

Address

Street	
City	
Zip code	
Country	Sverige

OK Cancel

Edit customer

Name

First name	Apu
Last name	Simpson

Email and phone

Home phone	+46 66 199 199
Cell phone	+44 77 400400
E-mail, business	nisse@someplace.eu
E-mail, private	nisse@prMail.eu

Address

Street	Springfield
City	Oxford
Zip code	12 3456
Country	United Kingdom

OK Cancel

Figure 2: Contact GUI

2.5 When the OK Button on the above form is clicked, the input is used by the **MainForm**.

2.6 Data that are inputted by the user, are used to create an object of a **Contact** class that we are going to create.

3. The Project

Figure 3 suggests the classes that can be identified for the problem, based on the earlier discussions.

The class diagram below shows the association between the classes.

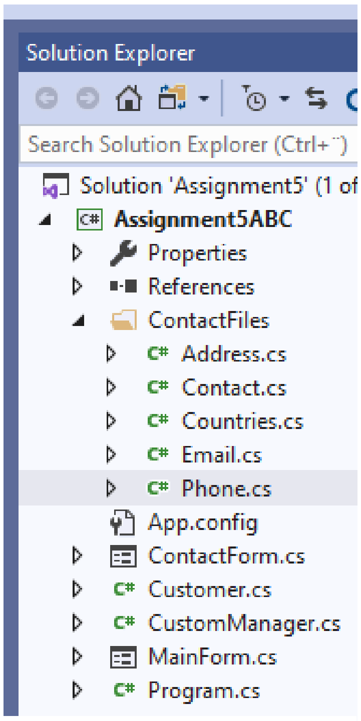
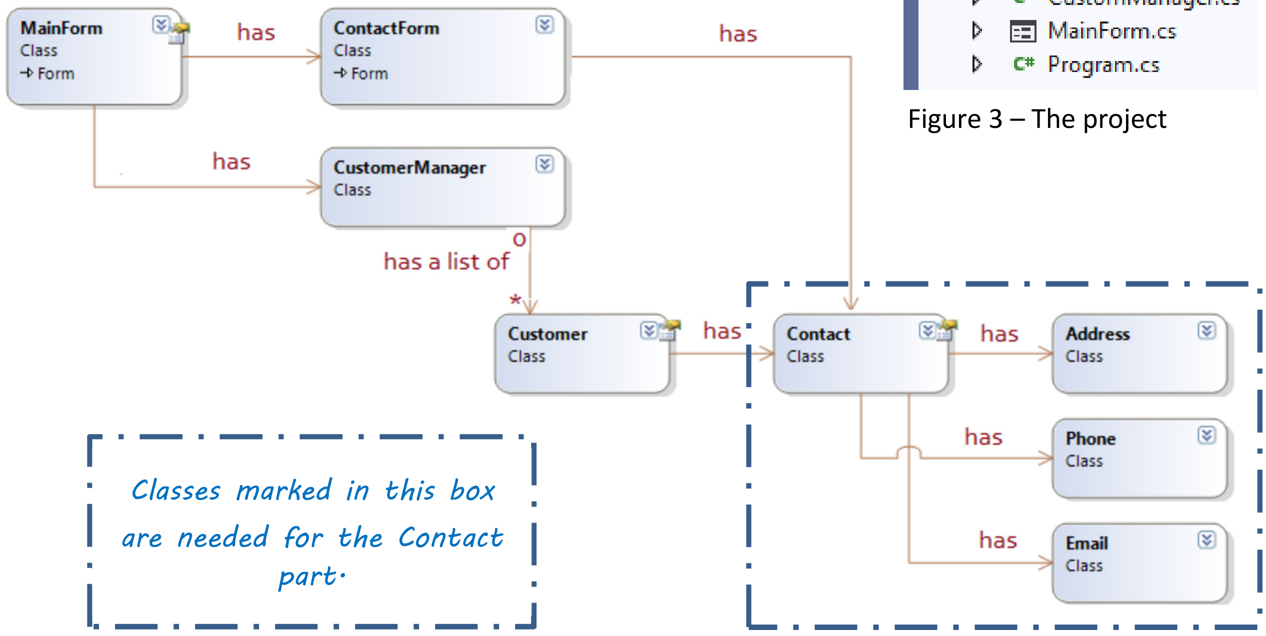


Figure 3 – The project



MainForm creates an instance of **CustomerManager** as a private field (instance variable).

CustomerManager has an array of **Customer** objects (ArrayList, or Lis<T>), as private field

Customer has an instance of **Contact,** as a private field.

Contact has an instance of **Address**, **Email** and **Phone** as its fields.

MainForm uses a local instance of **ContactForm** (local variable inside the event-handler methods).

4. To Do

You can begin sketching your GUI forms or begin with writing the classes that are to store and process data. Draw the Contact form and use proper names for the controls and then begin coding.

Part 1: Contact

Assuming that you are ready for writing code, it is important to structure the solution by first thinking about the identifying classes. In the above GUI example, the user is required to input phone numbers, emails, address, in addition to first and last name of a customer. However, a Customer can have other attributes, ID, type of customer, and so on.

4.1 Create a class and save it as **Contact**.cs. It is possible to save all data into class but think first if the data can be broken down to separate classes. A user may have several phone numbers and different emails. If we write separate classes for them, we separate their handling and meanwhile we can add more items in the future. This way, we are using encapsulation for bundling related data and operations into separate entities.

4.2 Create a class **Contact** and use the following classes as its private fields (aggregation):

- **Phone** for handling of phone numbers
- **Email** for handling of e-mails.
- **Address** for handling of all about an address.

Although we could write a class also for names, but we let the **Contact** class have the names as its private fields.

Complete this part and test it a Contact object is created when running the form. You can use a **MessageBox** to test the contents of your object or run VS debugger. When this part works well, then go to the next step and work on customer and customer registry.

Part 2 – Customer and CustomerManager

4.3 Create a class **Customer** with Contact as its field. More details are given in a moment.

4.4 Create also a class **CustomerManager** to server as the registry for storing Customer objects.

Note: By saving or storing data it is meant that data is stored into variables in the program and the data exists temporarily in the computer's memory. The data will not be available to the program after the program terminates. On a later occasion when we talk about saving data to files, we will learn how to save the contents of the registry permanently into files or a database.

In this part of the assignment, all functionalities are to work well, which are:

4.5 Add a new **Customer**:

- Display the Contact Form.
- If the Contact Form is closed used the Add- button, create a new customer with the data from the form, and add to your Customer Registry (**CustomerManager**). For this purpose, the **CustomerManager** must have an Add method to call.

4.6 Edit an existing **Customer**:

- The user selects a customer on the ListBox, containing all customers. If no customer is highlighted, a message is given and no further processing is to be done.
- Using the index, acquire the object from the **CustomerManager.**
- Display the Contact Form, passing the contact object that is to be edited.
- Fill the form with data from the contact object.
- If the Contact Form is closed used the Add- button, save back the object changed data to the **CustomerManager**, calling its Change method.

4.7 Delete an existing **Customer**:

- The user selects a customer on the ListBox, containing all customers. If no customer is highlighted, a message is given and no further processing is to be done.
- Using the index, call the **CustomerManager**'s Remove (or Delete) method to remove the object at position index in the registry.

4.8 When the user selects a customer in the list, other data for the selected customer is to be listed in the box shown at right side on Figure 1.

5. Requirements for Part I

Write the classes **Address**, **Email**, **Phone** and **Contact**.

5.1 Connect these classes with each other with a “has a” relation, as illustrated in the class diagram and described above.

5.2 Design your GUI (ContactForm) to allow the user give input to the mentioned classes. The user should at least provide a first name, or a last name, city and country for a contact. Write a method in the Contact class (CheckData()) to perform this validation.

5.3 The Address class should have three constructors. These should chain-call each other; do not write same code in the constructors. Chain-calling implies that a constructor with less number of parameters call one that has more parameters. As an example a constructor with 2 parameters should call the constructor with 3 parameters passing the two parameters and a default value for the third one. If there no such constructor, call the next one with 4 parameters and so forth.

5.4 Other classes can have only a default constructor.

5.5 All class fields are to be declared private. Use of public instance variables is strictly forbidden and will cause resubmission.

5.6 To provide access to the private fields of a class, use properties.

5.7 All classes and methods of each class are to be documented using the xml-documentation comments. The comments should have appropriate and valuable information to add clarification to your code.

5.8 Organize your project files in folders as the example given here.

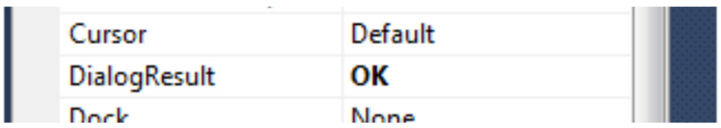
5.9 The ContactForm should also have a Cancel button. When the user clicks the Cancel button **the** program should ask the user for a confirmation, as in the run example here:

5.10 If the user clicks the Cancel button, a message box should be shown to let the user confirm. If the user selects the No button, the Contact form should not close in order for the user to continue inputting, but if she uses the Yes button, the form should be closed without saving the data.

5.11 **Note**: For the OK and Cancel Buttons to work well, you need to change their property DialogResult to the corresponding function (OK or Cancel) in VS, Properties Window.

Rremember that the OK, Yes, No and other such buttons on a form has a property named **DialogResult** For a OK button to work properly, set it to OK, using the Properties window in VS. Otherwise it will not work as an OK.

5.12 Do the same for the Cancel button but set the **DialogResult** to Cancel. Otherwise, they will not work



Cursor	Default
DialogResult	**OK**
Dock	None

Optional

5.13 When displaying the names of countries, replace the '_' char in countries like United_States_Of_America to a blank space so it can be read as “United States of America”.

6. Requirements for Part I

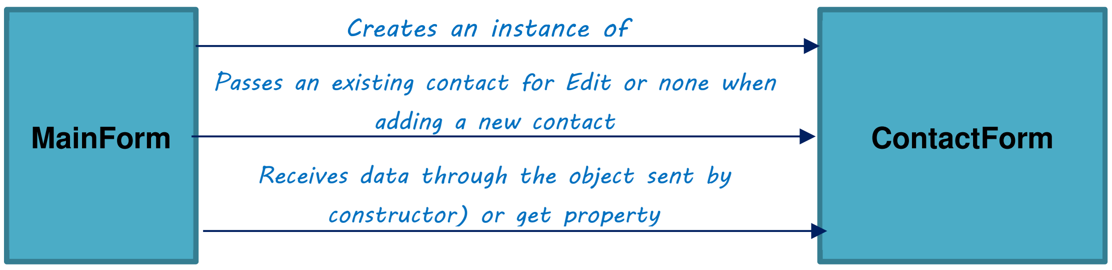
6.1 Write the classes **Customer** and **CustomerManager** classes.

6.2 For storing customer data, we are going to use a dynamic array. Dynamic arrays are known as collections. Collections are a big subject and as it is both too early and too much to talk all about them at this time, we will only concentrate on the very useful and perhaps the most used collection called the **List<T>.** This collection is generic which means it works for all kinds of objects. The collection is recommended to be uses instead of its non-generic counterpart **ArrayList** that was used a lot before (and it still can be used).

6.3 Each customer has contact data as in the previous part and a customer ID that can be generated internally in the application. The ID may be as simple as a string or more complicated as a GUID.

6.4 The Add, Edit and Delete features should work properly according to the descriptions provided earlier.

6.1. The **ContactForm** should not interact with any GUI element in the MainForm. Likewise, the MainForm should not interact with any of the GUI elements on the **ContactForm**. **MainForm** should only send input to ContactForm and receive output through the properties provided in the **ContactForm**


MainForm
Creates an instance of
Passes an existing contact for Edit or none when adding a new contact
Receives data through the object sent by constructor) or get property
ContactForm

6.2. MainForm can use a local variable of the Contact form, i.e. it can create an instance for the ContctForm form every time the user presses the Add button on the main GUI (Figure 1).

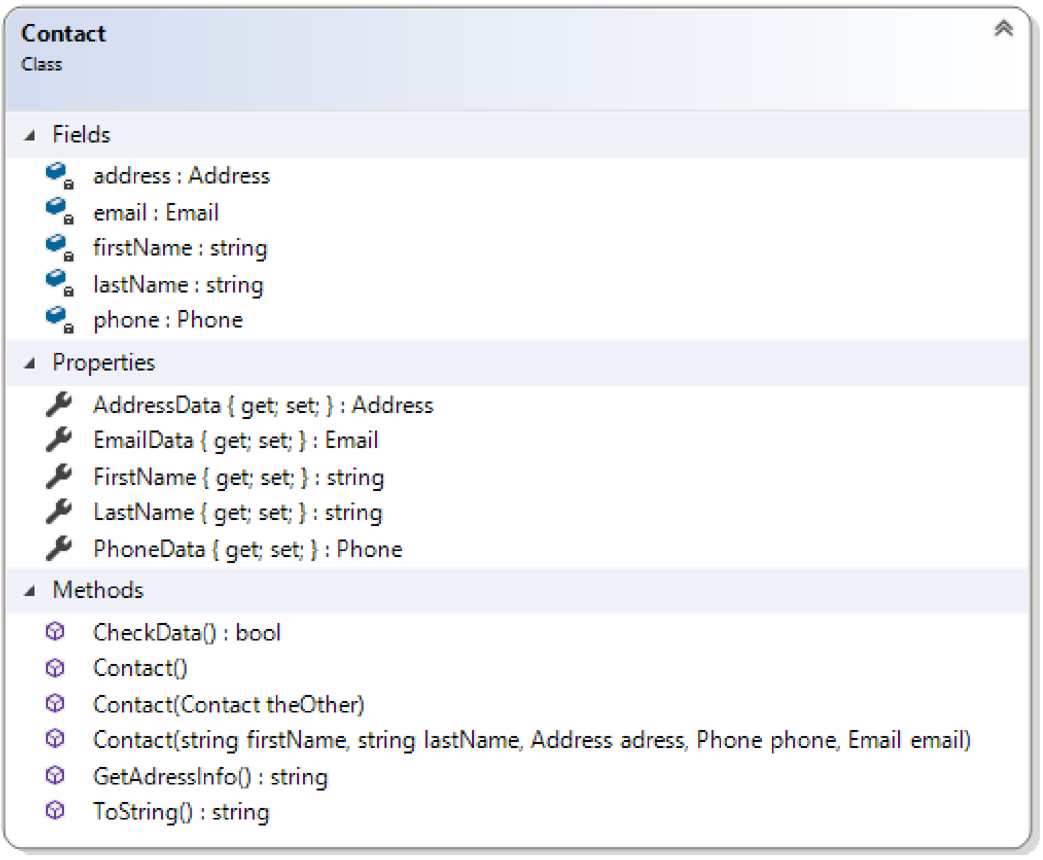
6.3. An existing contact can be passed to the Contact form either via constructor of the ContactForm (you can change it so it can receive a Contact object as a parameter) or using set property. This is needed when editing an exist

6.4. Control must be done in your code so that the list (registry) is not indexed out of range (make sure the index, when trying to access an element in the list is >= 0 and < the length of the list.

6.5. Use a List<Customer> to save a customer. Put this class in a container class (CustomerManager).

7. Class Details

The following diagrams are provided to give you an idea of the structure of the classes. You don't have to write all the methods or methods whose purpose is not clear for you.

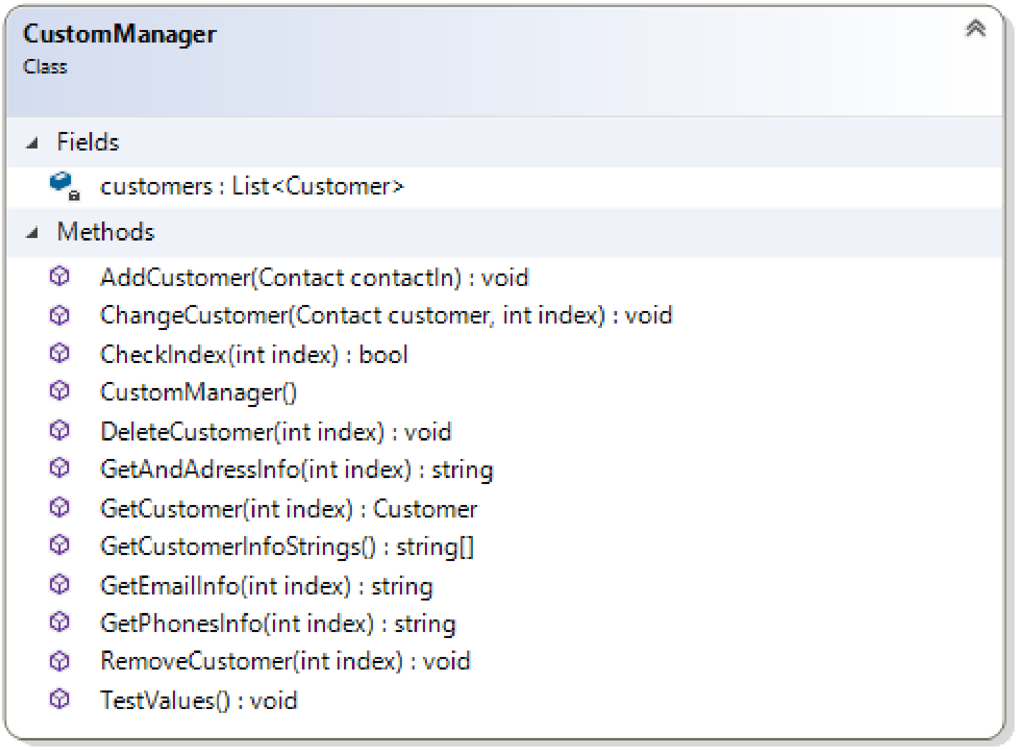


Contact
Class

Fields
- address : Address
- email : Email
- firstName : string
- lastName : string
- phone : Phone

Properties
- AddressData { get; set; } : Address
- EmailData { get; set; } : Email
- FirstName { get; set; } : string
- LastName { get; set; } : string
- PhoneData { get; set; } : Phone

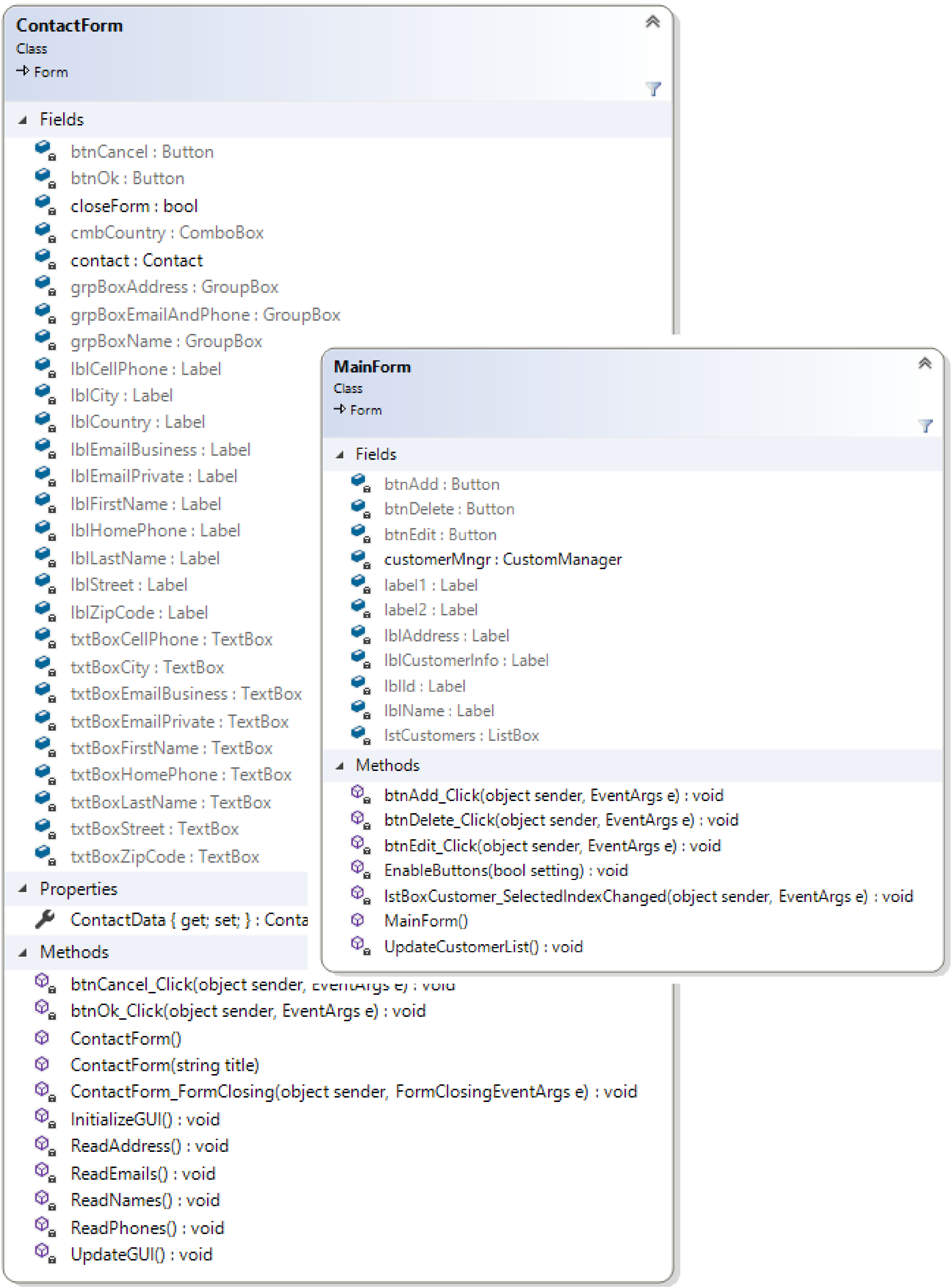
Methods
- CheckData() : bool
- Contact()
- Contact(Contact theOther)
- Contact(string firstName, string lastName, Address adress, Phone phone, Email email)
- GetAdressInfo() : string
- ToString() : string



CustomManager
Class

Fields
- customers : List<Customer>

Methods
- AddCustomer(Contact contactIn) : void
- ChangeCustomer(Contact customer, int index) : void
- CheckIndex(int index) : bool
- CustomManager()
- DeleteCustomer(int index) : void
- GetAndAdressInfo(int index) : string
- GetCustomer(int index) : Customer
- GetCustomerInfoStrings() : string[]
- GetEmailInfo(int index) : string
- GetPhonesInfo(int index) : string
- RemoveCustomer(int index) : void
- TestValues() : void



ContactForm
Class
→ Form

Fields
- btnCancel : Button
- btnOk : Button
- closeForm : bool
- cmbCountry : ComboBox
- contact : Contact
- grpBoxAddress : GroupBox
- grpBoxEmailAndPhone : GroupBox
- grpBoxName : GroupBox
- lblCellPhone : Label
- lblCity : Label
- lblCountry : Label
- lblEmailBusiness : Label
- lblEmailPrivate : Label
- lblFirstName : Label
- lblHomePhone : Label
- lblLastName : Label
- lblStreet : Label
- lblZipCode : Label
- txtBoxCellPhone : TextBox
- txtBoxCity : TextBox
- txtBoxEmailBusiness : TextBox
- txtBoxEmailPrivate : TextBox
- txtBoxFirstName : TextBox
- txtBoxHomePhone : TextBox
- txtBoxLastName : TextBox
- txtBoxStreet : TextBox
- txtBoxZipCode : TextBox

Properties
- ContactData { get; set; } : Conta

Methods
- btnCancel_Click(object sender, EventArgs e) : void
- btnOk_Click(object sender, EventArgs e) : void
- ContactForm()
- ContactForm(string title)
- ContactForm_FormClosing(object sender, FormClosingEventArgs e) : void
- InitializeGUI() : void
- ReadAddress() : void
- ReadEmails() : void
- ReadNames() : void
- ReadPhones() : void
- UpdateGUI() : void

MainForm
Class
→ Form

Fields
- btnAdd : Button
- btnDelete : Button
- btnEdit : Button
- customerMngr : CustomManager
- label1 : Label
- label2 : Label
- lblAddress : Label
- lblCustomerInfo : Label
- lblId : Label
- lblName : Label
- lstCustomers : ListBox

Methods
- btnAdd_Click(object sender, EventArgs e) : void
- btnDelete_Click(object sender, EventArgs e) : void
- btnEdit_Click(object sender, EventArgs e) : void
- EnableButtons(bool setting) : void
- lstBoxCustomer_SelectedIndexChanged(object sender, EventArgs e) : void
- MainForm()
- UpdateCustomerList() : void

Submission, help and guidance

Compress all files and sub-folders, from both Part I and this part, into a Zip or Rar file and upload it as before. A detailed description of how to program different sections of this part of the assignment is available as a separate document on the Module. The document is only to provide some type of a help. You may of course complete the assignment without the help document.

Good Luck!

Farid Naisan,
Course Responsible and Instructor