MALMÖ HÖGSKOLA
Teknik och samhälle

Malmö University
School of Technology

Programming Using Visual Basic

OOP and Encapsulation

Assignment 5 – Alt 1
Contact Registry

Mandatory only for Grade C or G
(If you aim at grades A, B or VG, do not do this assignment)

Farid Naisan
University Lecturer
Department of Computer Science

Contact 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 practice data-hiding to complete the process of encapsulation. This assignment covers the following concepts:

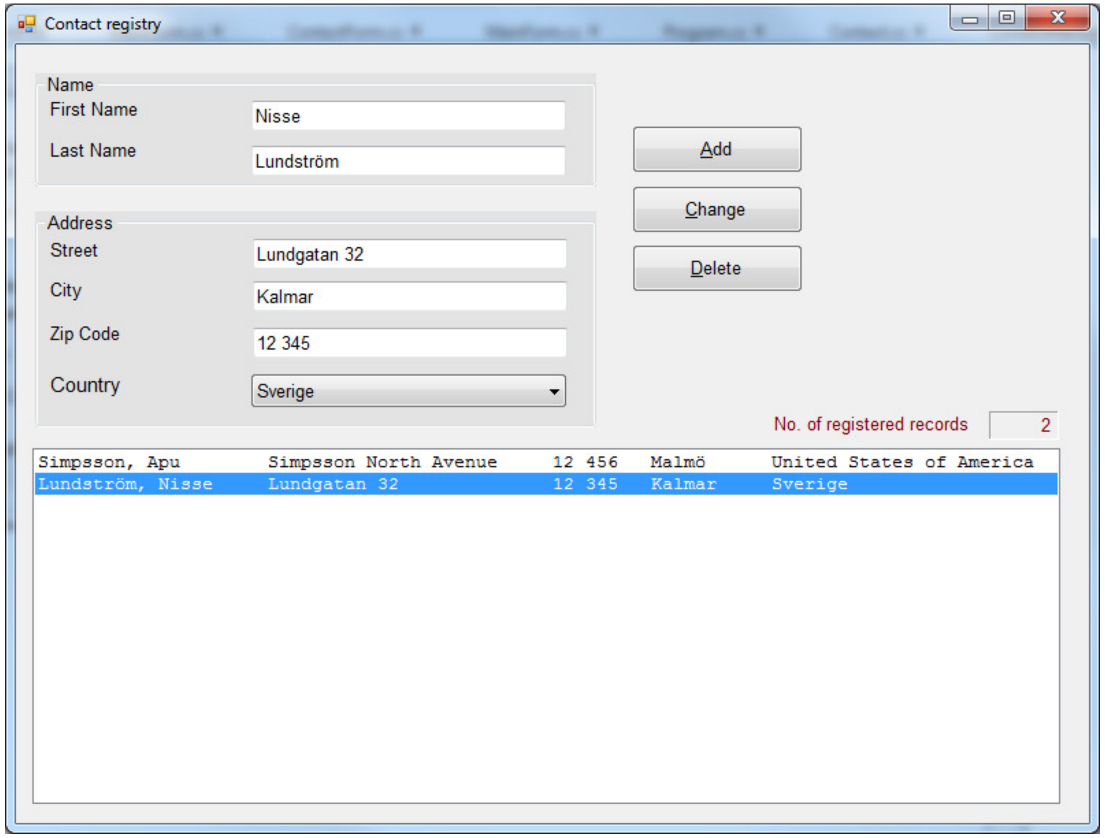
- Encapsulation
- Constructors
- "**Has a**" relation between objects
- Properties
- Collection of objects

2. Description

Write a Windows Form Application that saves a list of contacts. Every contact is a person that has the following data:

First name
Last name
Address including:
  Street address
  Zip code
  City
  Country

The GUI should allow the user to add a new contacts, change or delete an existing one. It should also present a list of all contacts saved in the registry. The list is to be updated after <u>every change</u> in the registry.



Name
First Name: Nisse
Last Name: Lundström

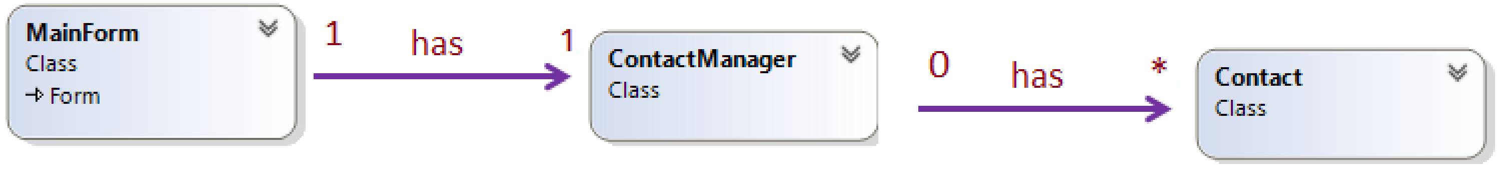
Address
Street: Lundgatan 32
City: Kalmar
Zip Code: 12 345
Country: Sverige

No. of registered records: 2

```
Simpsson, Apu       Simpsson North Avenue    12 456   Malmö     United States of America
Lundström, Nisse    Lundgatan 32             12 345   Kalmar    Sverige
```

3. Requirements for a Pass grade:

3.1 Write at least three classes as in the class diagram below:



3.2 Use a **List (Of T)** (or an **ArrayList**) for the registry.

3.3 The registry should be handled in a separate class (**ContactManager**).

3.4 Write a class **Address** for handling addresses. The **Address** class should have three constructors:

3.4.1 One default constructor.

3.4.2 One constructor with 3 parameters (for street, zip code and city).

3.4.3 One constructor with 4 parameters (street, zip code, city, and country).

3.4.4 **Chain calling**: The constructor with 3 parameters should call the one with 4 parameters (using the syntax **Me.New**). This way of methods calling each other (to avoid rewriting same code) is called "chain calling". We implement chain calling on our constructors, but remember that this requires a special syntax (see the Help section later in this document).

3.4.5 Override the **ToString** method so the method returns a string formatted with values saved in the address object.

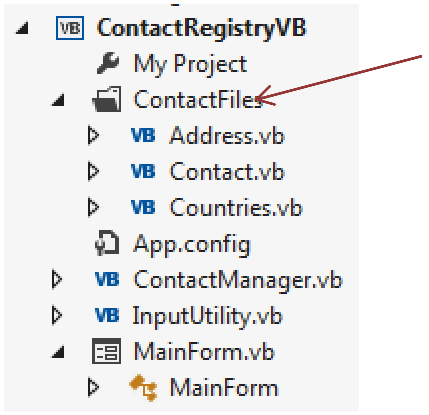
3.5 The Contact class should contain data about the first name and last name as well as the address of a person.

3.6 You shall also override the **ToString** method of the **Contact** class. The method should format a string with the first and last name and the address of the contact. For the address part, make use of the **ToString** of the Address class.

3.7 The **ContactManager** should be responsible for adding, deleting, changing and all other operation on the contact list. However, it should use the services (methods) of the **Address** class.

3.8 All instance variables in all classes are to be **Private** (as before in other assignments) and access to them should be handled via Properties. However, do not write a property that returns an array or a list (e.g. the contact list in the **ContactManager** class). Instead write a method that returns an element of the list in a given position.

3.9 Organize your contact files in a separate folder in VS (see the figure to the right).



Provide a brief documentation of every method and every class with XML-documentation comments (using ''').

2.1 Do not forget to write your name on top of each file!

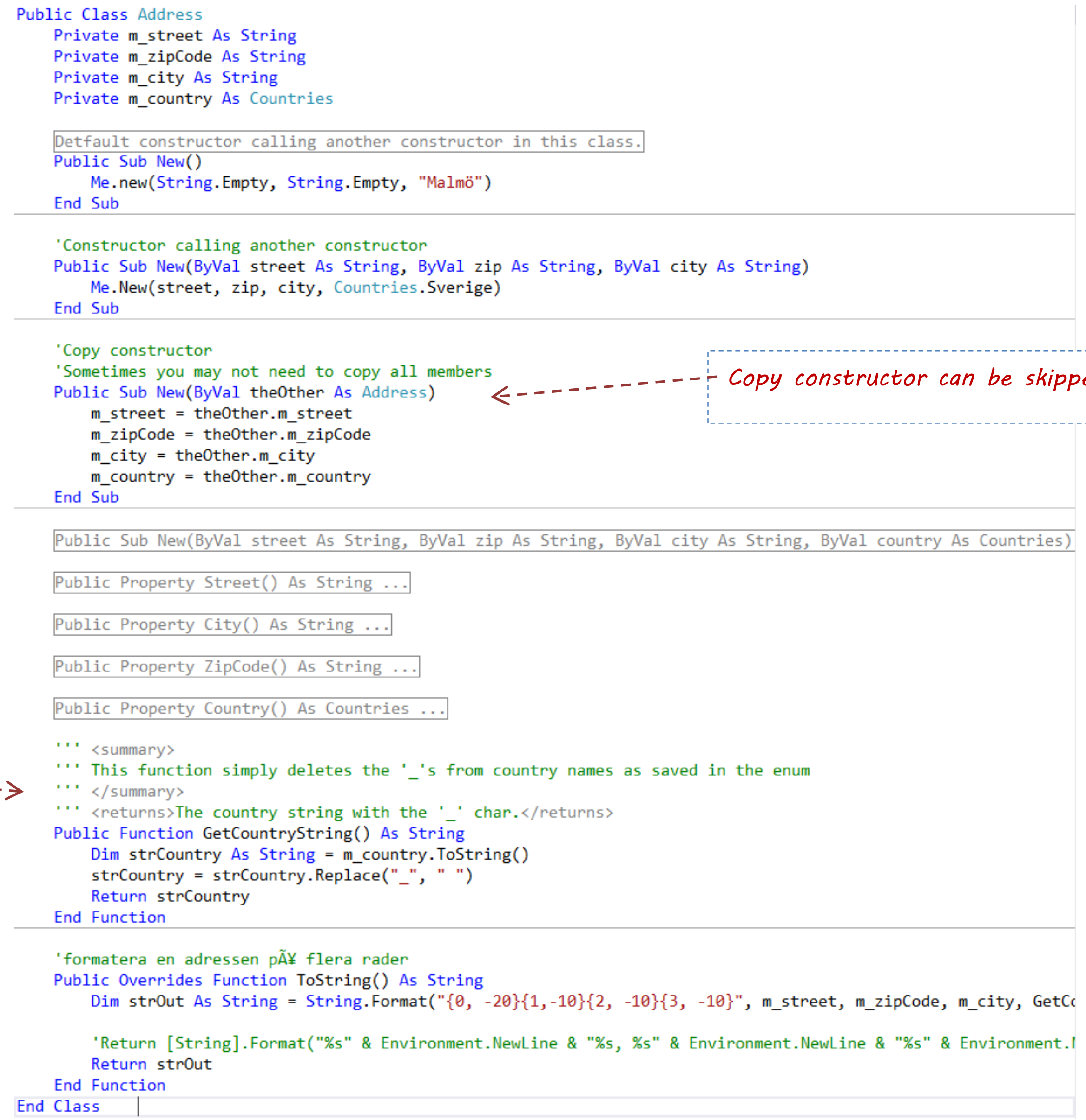
HELP AND GUIDANCE

In the sections that follow, you are given some code snippets as screen dump images. It is very important that you understand every step and not just imitate the code. Use the forum in the module to ask questions and discuss the things you are not sure about.

4. The Address Class

An overview of the Address class is shown here.

Note: the comments are deleted from the code images to save space. Don't forget to comment all methods of the classes as well as the classes themselves, using XML-compatible comments (with '''), to avoid resubmission.



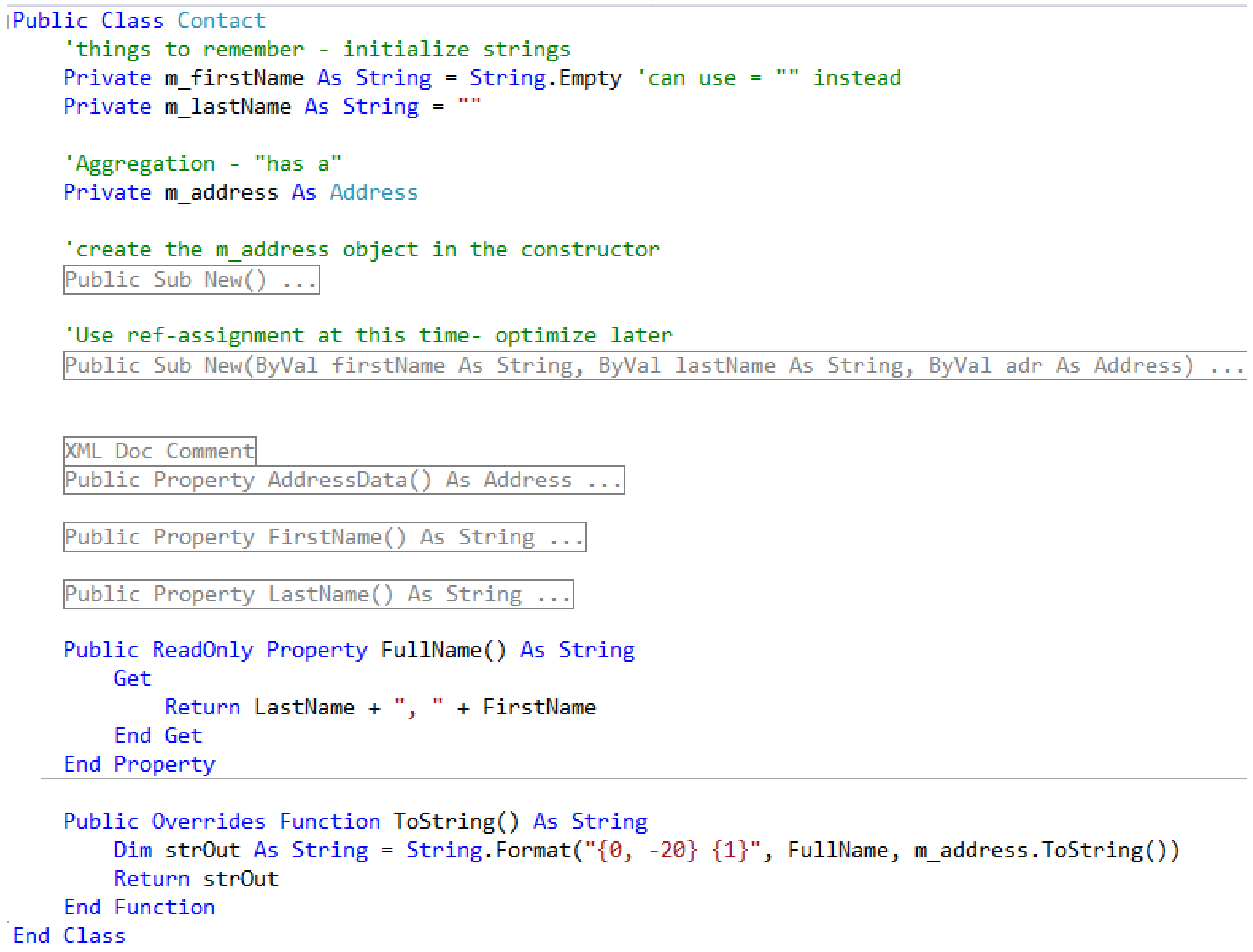
```
Public Class Address
    Private m_street As String
    Private m_zipCode As String
    Private m_city As String
    Private m_country As Countries

    Detfault constructor calling another constructor in this class.
    Public Sub New()
        Me.new(String.Empty, String.Empty, "Malmö")
    End Sub

    'Constructor calling another constructor
    Public Sub New(ByVal street As String, ByVal zip As String, ByVal city As String)
        Me.New(street, zip, city, Countries.Sverige)
    End Sub

    'Copy constructor
    'Sometimes you may not need to copy all members
    Public Sub New(ByVal theOther As Address)
        m_street = theOther.m_street
        m_zipCode = theOther.m_zipCode
        m_city = theOther.m_city
        m_country = theOther.m_country
    End Sub

    Public Sub New(ByVal street As String, ByVal zip As String, ByVal city As String, ByVal country As Countries)

    Public Property Street() As String ...

    Public Property City() As String ...

    Public Property ZipCode() As String ...

    Public Property Country() As Countries ...

    ''' <summary>
    ''' This function simply deletes the '_'s from country names as saved in the enum
    ''' </summary>
    ''' <returns>The country string with the '_' char.</returns>
    Public Function GetCountryString() As String
        Dim strCountry As String = m_country.ToString()
        strCountry = strCountry.Replace("_", " ")
        Return strCountry
    End Function

    'formatera en adressen pÃ¥ flera rader
    Public Overrides Function ToString() As String
        Dim strOut As String = String.Format("{0, -20}{1,-10}{2, -10}{3, -10}", m_street, m_zipCode, m_city, GetC

        'Return [String].Format("%s" & Environment.NewLine & "%s, %s" & Environment.NewLine & "%s" & Environment.
        Return strOut
    End Function
End Class
```

Copy constructor can be skipped.

5. The Contact class

Declare instance variables for storing first name, last name and address. For the address, declare an instance of the Address class and create the instance in the Contact's constructor.

Write two constructors and a **ToString** method.

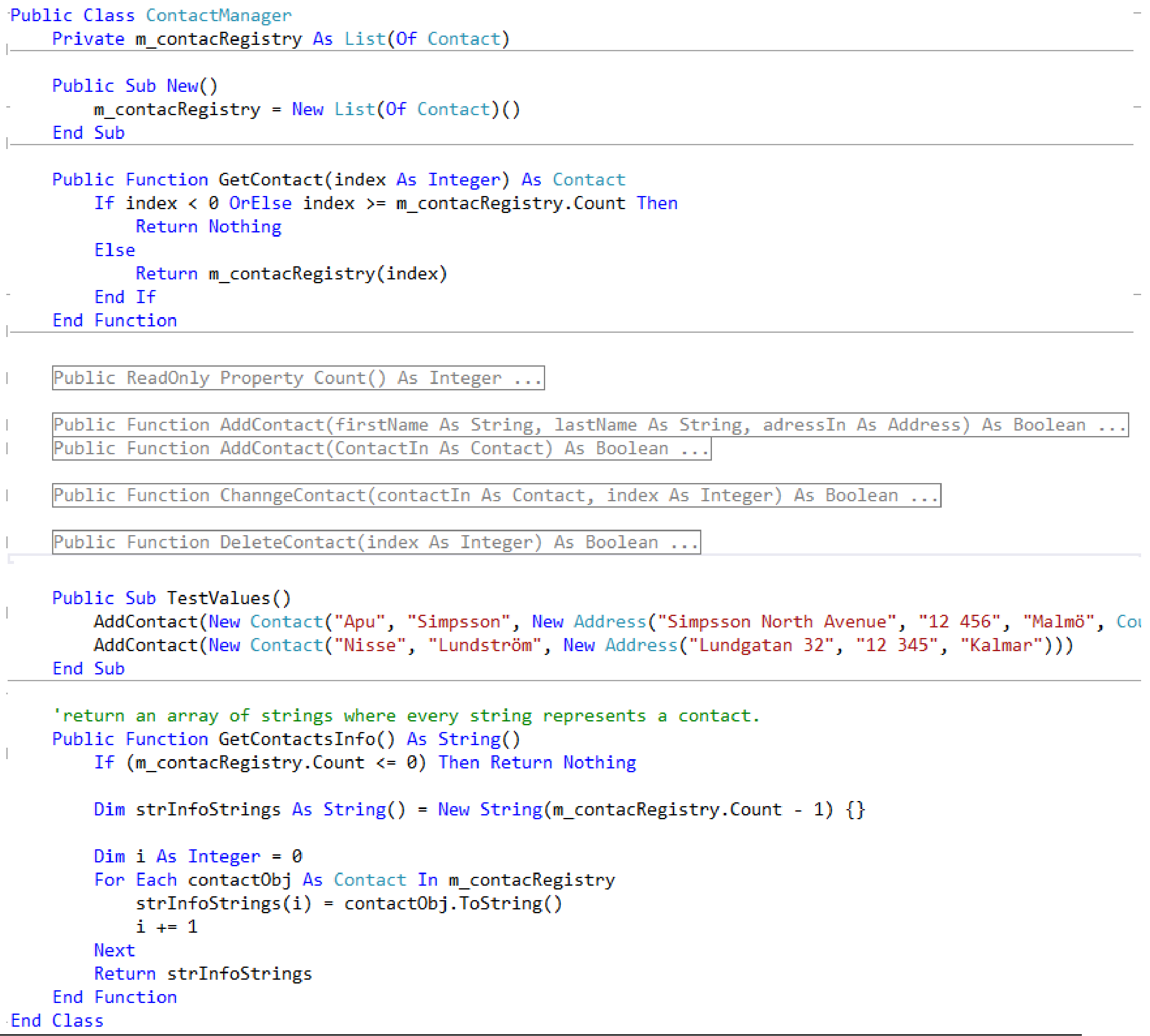


```
Public Class Contact
    'things to remember - initialize strings
    Private m_firstName As String = String.Empty 'can use = "" instead
    Private m_lastName As String = ""

    'Aggregation - "has a"
    Private m_address As Address

    'create the m_address object in the constructor
    Public Sub New() ...

    'Use ref-assignment at this time- optimize later
    Public Sub New(ByVal firstName As String, ByVal lastName As String, ByVal adr As Address) ...


    XML Doc Comment
    Public Property AddressData() As Address ...

    Public Property FirstName() As String ...

    Public Property LastName() As String ...

    Public ReadOnly Property FullName() As String
        Get
            Return LastName + ", " + FirstName
        End Get
    End Property

    Public Overrides Function ToString() As String
        Dim strOut As String = String.Format("{0, -20} {1}", FullName, m_address.ToString())
        Return strOut
    End Function
End Class
```

6. The ContactManager class

This class is a container class for **Contact** objects. It saves contact objects in a dynamic list of the type **List(Of T)**. Notice how the list is declared in the class and created in the constructor of the class.

The class has methods for adding a new contact, deleting and changing an existing contact. Have a close look at the method **GetContactInfo** at the end of the code. This method prepares an array of strings, where every string is made up of information about the contact object. This method can be called from the **MainForm** to update the **ListBox**.



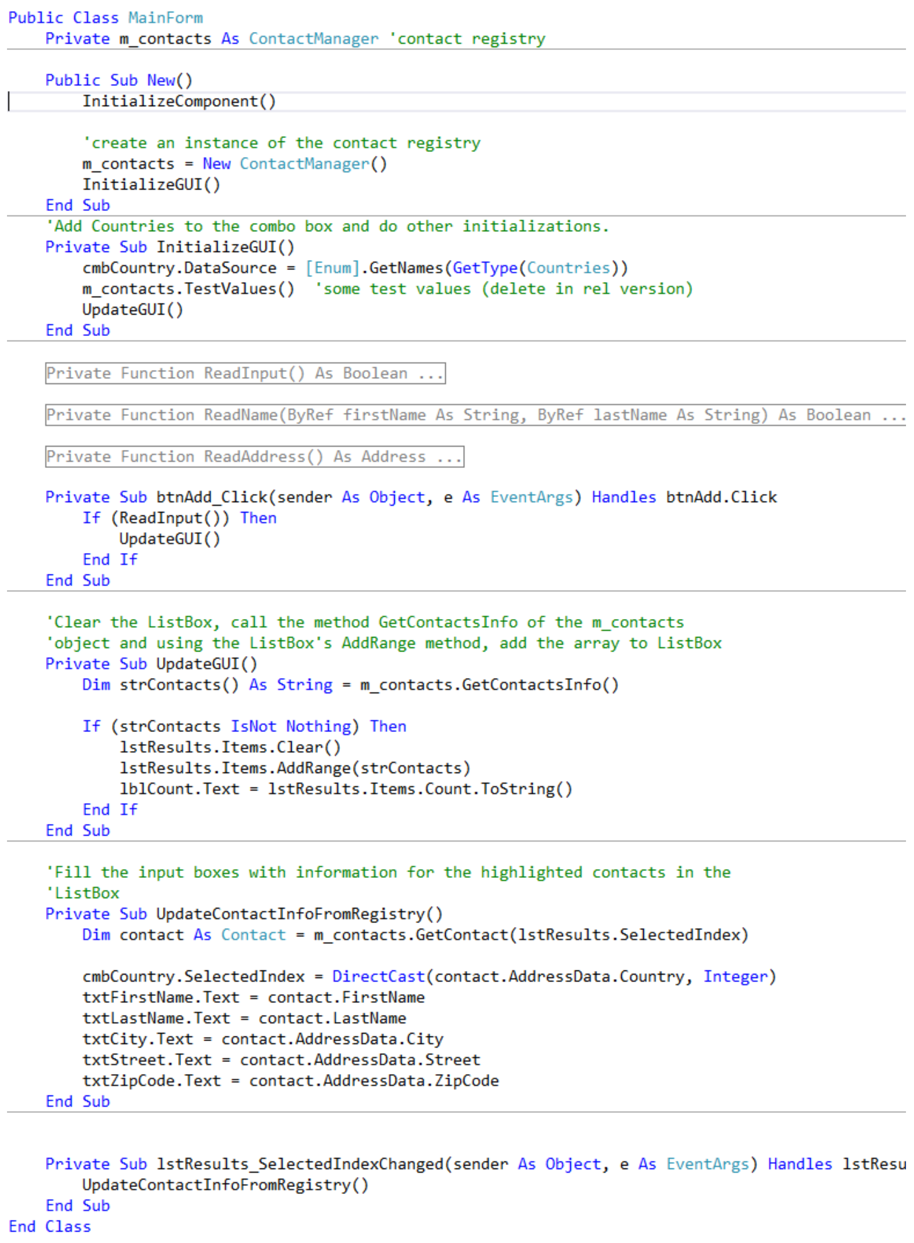
```
Public Class ContactManager
    Private m_contacRegistry As List(Of Contact)

    Public Sub New()
        m_contacRegistry = New List(Of Contact)()
    End Sub

    Public Function GetContact(index As Integer) As Contact
        If index < 0 OrElse index >= m_contacRegistry.Count Then
            Return Nothing
        Else
            Return m_contacRegistry(index)
        End If
    End Function

    Public ReadOnly Property Count() As Integer ...

    Public Function AddContact(firstName As String, lastName As String, adressIn As Address) As Boolean ...
    Public Function AddContact(ContactIn As Contact) As Boolean ...

    Public Function ChanngeContact(contactIn As Contact, index As Integer) As Boolean ...

    Public Function DeleteContact(index As Integer) As Boolean ...

    Public Sub TestValues()
        AddContact(New Contact("Apu", "Simpsson", New Address("Simpsson North Avenue", "12 456", "Malmö", Co
        AddContact(New Contact("Nisse", "Lundström", New Address("Lundgatan 32", "12 345", "Kalmar")))
    End Sub

    'return an array of strings where every string represents a contact.
    Public Function GetContactsInfo() As String()
        If (m_contacRegistry.Count <= 0) Then Return Nothing

        Dim strInfoStrings As String() = New String(m_contacRegistry.Count - 1) {}

        Dim i As Integer = 0
        For Each contactObj As Contact In m_contacRegistry
            strInfoStrings(i) = contactObj.ToString()
            i += 1
        Next
        Return strInfoStrings
    End Function
End Class
```

7. The MainForm class

Sketch your GUI as suggested by the sample run example presented at the beginning of the document.

Double-click on the **ListBox** in VS at design time. VS will prepare the method for the **SelectedIndexChanged** event of the **ListBox**. All code your write in this method will be automatically called when the user highlights an entry in the ListBox (clicks on a row in the ListBox). What you need to do here is to get the information related to the entry from the registry (**ContactManager** object) and fill the input boxes with the data to make it easier for the user to edit.

In the code example here, the method **UpdateContactInfoFromRegistry**() is called to accomplish the task. Make sure that you understand the code written in the method.



```
Public Class MainForm
    Private m_contacts As ContactManager 'contact registry

    Public Sub New()
        InitializeComponent()

        'create an instance of the contact registry
        m_contacts = New ContactManager()
        InitializeGUI()
    End Sub
    'Add Countries to the combo box and do other initializations.
    Private Sub InitializeGUI()
        cmbCountry.DataSource = [Enum].GetNames(GetType(Countries))
        m_contacts.TestValues()  'some test values (delete in rel version)
        UpdateGUI()
    End Sub

    Private Function ReadInput() As Boolean ...

    Private Function ReadName(ByRef firstName As String, ByRef lastName As String) As Boolean ...

    Private Function ReadAddress() As Address ...

    Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
        If (ReadInput()) Then
            UpdateGUI()
        End If
    End Sub

    'Clear the ListBox, call the method GetContactsInfo of the m_contacts
    'object and using the ListBox's AddRange method, add the array to ListBox
    Private Sub UpdateGUI()
        Dim strContacts() As String = m_contacts.GetContactsInfo()

        If (strContacts IsNot Nothing) Then
            lstResults.Items.Clear()
            lstResults.Items.AddRange(strContacts)
            lblCount.Text = lstResults.Items.Count.ToString()
        End If
    End Sub

    'Fill the input boxes with information for the highlighted contacts in the
    'ListBox
    Private Sub UpdateContactInfoFromRegistry()
        Dim contact As Contact = m_contacts.GetContact(lstResults.SelectedIndex)

        cmbCountry.SelectedIndex = DirectCast(contact.AddressData.Country, Integer)
        txtFirstName.Text = contact.FirstName
        txtLastName.Text = contact.LastName
        txtCity.Text = contact.AddressData.City
        txtStreet.Text = contact.AddressData.Street
        txtZipCode.Text = contact.AddressData.ZipCode
    End Sub


    Private Sub lstResults_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstResu
        UpdateContactInfoFromRegistry()
    End Sub
End Class
```

8. ArrayList and List (Of T)

ArrayList and **List(Of T)** are two data structures (two collections) that are defined in the .NET Framework. Both are objects and are created in the same way as other objects:

```
myArrayList As ArrayList = New ArrayList();
myList As List (Of T) = New List(Of T)();
```

where **T** in the case of using a **List** can be any type of object, a primitive type such as **Integer**, **Double**, **Boolean** or classes such **Car**, **BankAccount**, **RealEstate**, etc.

Which one to use?

List and **ArrayList** share similar syntax and both are easy to work with. An **ArrayList** can have objects of different types while a **List (Of T)** cannot. Hence, List should be preferred when working with objects of the same type. List(Of T) is faster and easier to use. When declaring and creating an object of **List(Of T)** you provide the type of the object it is going to contain by replacing the T with the class name.

```
Private m_contacRegistry As List(Of Contact)
```

An **ArrayList** is declared as follows:

```
Private m_contacRegistry As ArrayList 'type of elements not known!
```

An List(Of T)object is created as follows:

```
m_contacRegistry = New List (Of Contact)()
```

Question: what are the ( ) at the end of the statement supposed to denote?

An **ArrayList** object is created as follows:

```
m_contacRegistry = New ArrayList()
```

Both of the list types have the **Add, Remove, RemoveAt, Insert, Count** and several methods and properties that are used exactly in the same way. However, when fetching an element from an ArrayList object, you must use casting so that the compiler knows what type of object is saved in the element. The following code will work with a List(Of T) without any problem but it might cause a compiler error when used with an ArrayList.

```
Return m_contacRegistry(index)
```

When using an **ArrayList** object, you must write as follows:

```
Return DirectCast(Contacts(index), Contact)
```

The table below outlines some of the most useful methods that are available in both of them.

Add a new object to the list (at the end of the list)	Add()
Remove an object from the list	Remove()
Insert an element at a certain position	Insert()
Remove all elements	Clear()
Info on the number of elements in the list	Count
Access an element using	( )

Remember that every time an element is to be saved in the list, it must have been created (somewhere). To avoid abnormal program termination, make sure to always check the validity of an object as in the example below:

```
If contactIn IsNot Nothing Then
   'code
End If
```

9. Accessing Array elements (an advanced discussion – can skip it)

Other classes like **MainForm** would of course need to access data stored in the **m_contacRegistry** of its **ContactManager** object. Because **m_contacRegistry** is (and definitely should be) declared as **Private**, you need to somehow make the data in the array available to other objects. What would you do? Would you write a Property that makes the array reference accessible as below?

```
Return m_contacRegistry 'Bad programming - (not an accepted way in solving this assignment)
```

Never do that!
Another way is to write a method that returns the reference to an element in a position in the array:

```
Return m_contacRegistry(index) 'Better (accepted in solving this assignment)
```

This method is better but still not very safe because of the fact that we are passing the address of our object (a **Contact** object saved in the position index) in the memory. However, if the object's data is protected by declaring them `private` and access to the data is provided through `set` and `get` methods (as in our case), there is a degree of safety. A better and safer way is to return a copy of an object.

- Create a new object of the **Contact**type
- Copy all fields from the original object (**m_contacRegistry(index)**) to this copy object

- Return the copy object.

```
Public Function GetContactCopy(index As Integer) As Contact
    Dim origObj As Contact = m_contacRegistry(index)

    'Create a copy of the contact element saved in position = index
    Dim copyObj As Contact = New Contact(origObj.FirstName, origObj.LastName, origObj.AddressData)

    'return the copy
    Return copyObj
End Function
```

To make the problem more complicated but make life easier, copying an object can be effectively done by using a so called **copy constructor.** A copy constructor can be used to initialize an object with values of another object of the same type. Although you don't need to do that to solve this assignment, it is undoubted going to pay back once you have learned how to use it. It is good to make it a habit to write a copy constructor to every new class that you write. The code below shows how a copy constructor may look like.

```
'Copy constructor
Public Sub New(ByVal theOther As Address)
   m_street = theOther.m_street
   m_zipCode = theOther.m_zipCode
   m_city = theOther.m_city
   m_country = theOther.m_country
End Sub
```

The above constructor can be called as follows: `Dim address2 As New Address(address1)`

All field members (as programmed in the copy constructor) of the object address1 are copied to the corresponding members of address2.

A copy constructor in the Contact class may be written is shown here:

Note: The Address class' copy constructor is called in the last statement.

```
Public Sub New(ByVal theOther As Contact)
    Me.m_firstName = theOther.m_firstName
    Me.m_lastName = theOther.m_lastName
    Me.AddressData = New Address(theOther.m_address)
End Sub
```

10. Submission

Upload your project to Its L as in earlier assignments.

Good Luck!

Farid Naisan, Course Responsible and Instructor