Assignment 4: The Party Organizer

This alternative is a Pass version of Assignment 4, and the highest grade will be a C. If you would like to acquire a higher grade (A and B), you have to solve an alternative assignment that is intended for these grades, in which case you do not have to proceed with this one.

1. Objectives

The main objectives of this assignment are:

- To learn using one and two-dimensional arrays.
- To learn how to write and use properties to access values saved in the private fields of an object.

2. Description

In this assignment you will be working with two projects: (1) **Phone Book**, for which you can either use a Console Application or a Windows Forms application, and (2) **Party Organizer**, a Windows Forms application. The first project gives training in working with one and two-dimensional arrays. In the second project, you will get training in one-dimensional arrays as well as working with Windows Forms and controls. Both parts are mandatory for receiving a C grade.

3. Part 1: PhoneBook App

Using a console application in Part 1 is only a simplification, and if you would like to do this part as a Windows Forms App, you are welcome to do so. In this case, you can send the output to a ListBox, a Label or any other suitable control, and use buttons to start an action (such as displaying a list, sorting, etc.).

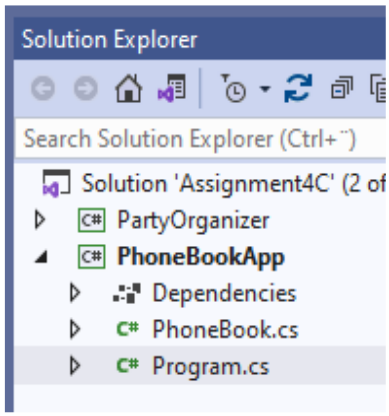
The description below is however assumes a solution using a Console application.

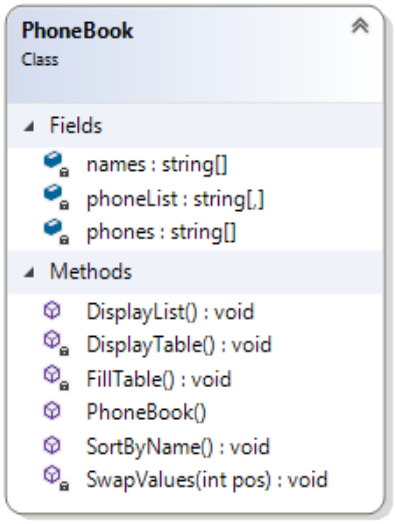
3.1 Create a Console Application, **PhoneBookApp** and a new solution which you can call Assignment4C.

3.2 Create then a class PhoneBook.cs. The purpose of the class is to store a list of names and their phone numbers. For these, we are going to use two arrays initialized with some values, as shown in the code snippet below. Change the values (Friend1, etc.) to names of your own friends, artists you like, or any names you wish. Change also phone numbers. There are five elements in each array, but you may change it to a minimum of four elements or larger to any max number.

```
class PhoneBook
{
    private string[] names = { "Friend1", "Friend2", "Adam", "Malcom", "Friend5" };
    private string[] phones = { "040-1231", "040-1232", "040-1233", "040-1234", "040-1235" };
```

Note: It is mandatory that you change the values of the arrays, `names` and `phones`. Follow all the steps that are described below.

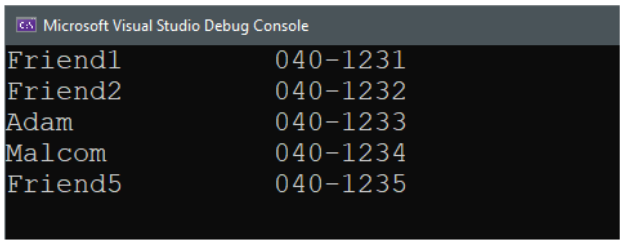




Display the contents of the arrays:

3.3 Write a method that prints out (**DisplayList**) each name and its corresponding phone number to the console window.

3.4 Write a constructor and call the above the above method and get an output showing names and phone numbers. The code for the constructor is provided at the end of this part.

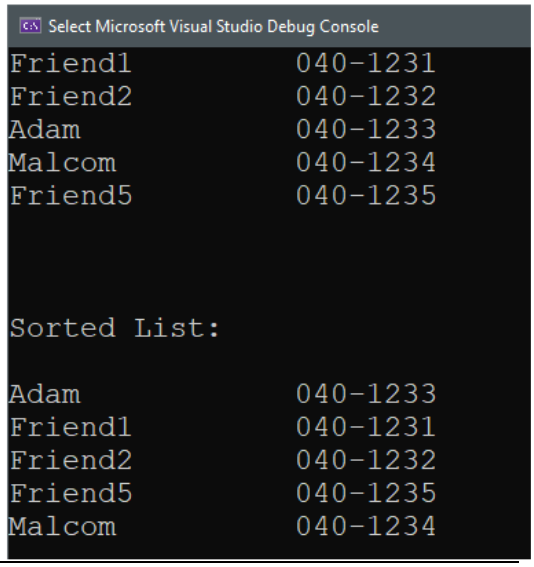


Sort arrays:

3.5 Write a method (**SortByName**) that sorts the **names** array in ascending (or descending) order using a bubble sort algorithm. Make sure that the phone numbers follow the sorting of the names, so the phone number is moved to the same position as the name it corresponds to. For instance, if a name is moved from position i to i+1, the phone number at position i must also be moved to position i+1.

3.6 Call this method from the constructor and then call the method **DisplayList** to display the contents of the arrays after sorting.

Note: When sorting values, you need to swap values. Swapping can be done inline or can be coded in a separate method. This method swaps values between two positions.



Assume we have array named `arr` and we would like to move the value from a next position `i+1` to current position, `i`:

```
temp = arr[i + 1];
arr[i + 1] = arr[i];
arr[i] = temp;
```

For more info on bubble sort, see:
https://www.tutorialspoint.com/data_structures_algorithms/bubble_sort_algorithm.htm).
You will also receive the code in the help document.

Two-dimensional array:

3.7 Instead of using two one-dimensional arrays, we can use an array of 5 x 2, i.e. a matrix (or a table) containing 5 rows and 2 columns. Each row represents a name and a phone number, where the name is saved in col 0 and phone number in col 1. Create a new instance variable in the class:

```
class PhoneBook
{
    private string[] names = { "Friend1", "Friend2", "Adam", "Malcom", "Friend5" };
    private string[] phones = { "040-1231", "040-1232", "040-1233", "040-1234", "040-1235" };

    private string[,] phoneList;
```

3.8 Write a method (**FillTable**) that fills the **phoneList** array with values taken from the **names** and **phones** array. The **names [row]** is to be saved in column 0 and the **phones [row]** should be saved in column 1. The variable row is the index to positions in the arrays. Run a loop for this action, with row as the counter.

```
phoneList[row, 0] = names[row];
phoneList[row, 1] = phones[row];
```

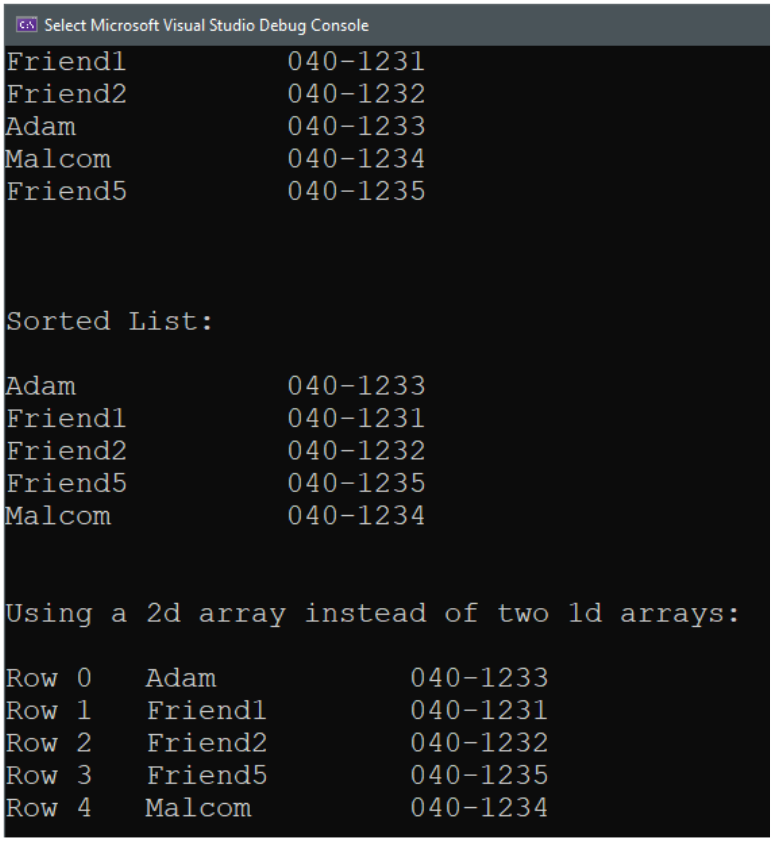
Call the method from the constructor.

3.9 Write a method (**DisplayTable**) that displays the contents of the **phoneList** showing the names and the columns. Call also this method from the constructor. The figure that follows show the results of a running all methods called in the constructor.

The constructor is also given on the next page.

3.10 Test the application by creating an object of the **PhoneBook** class.

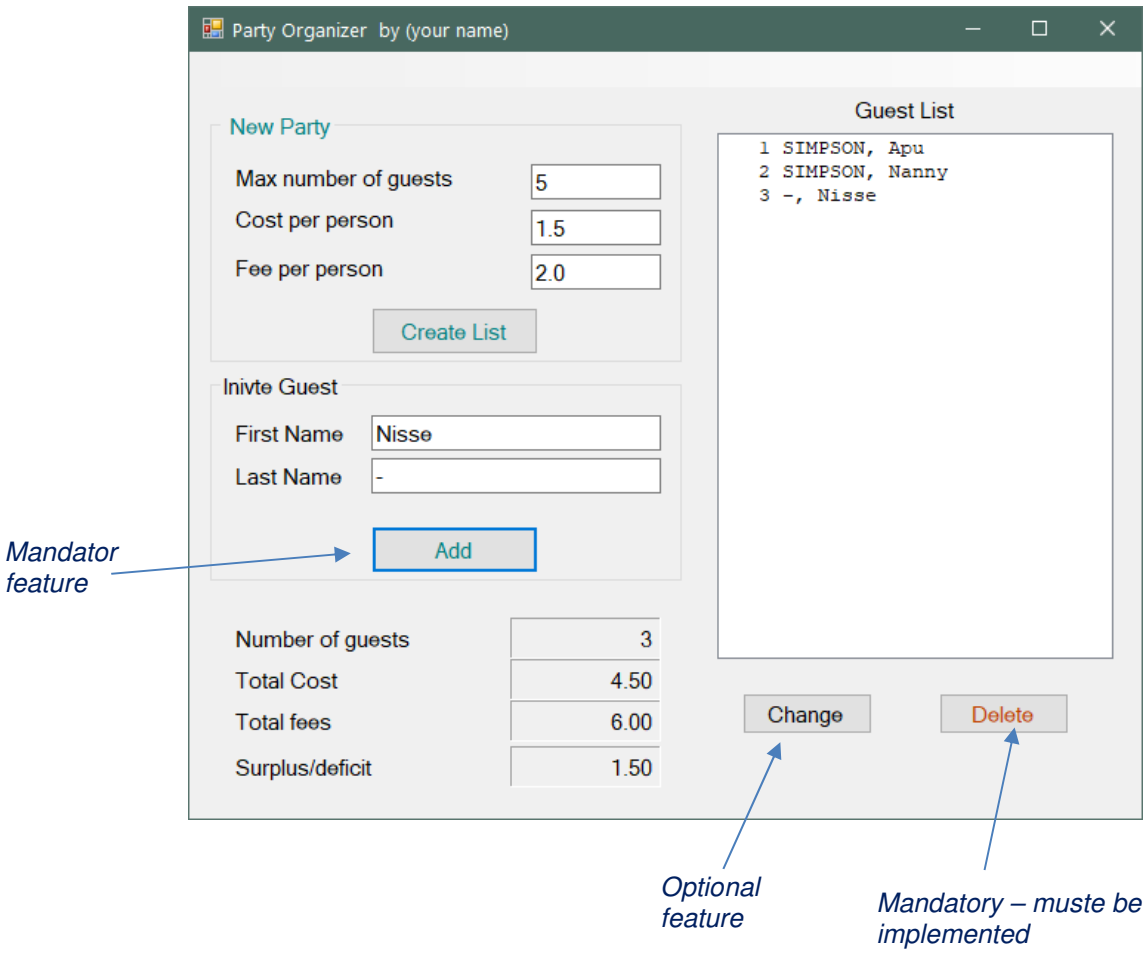
```
class Program
{
    0 references
    static void Main(string[] args)
    {
        //Create an object of the PhoneBook
        //The keyword "new" calls the constructor and
        //all code in the constructor gets executed.
        PhoneBook phones = new PhoneBook();
    }
}
```



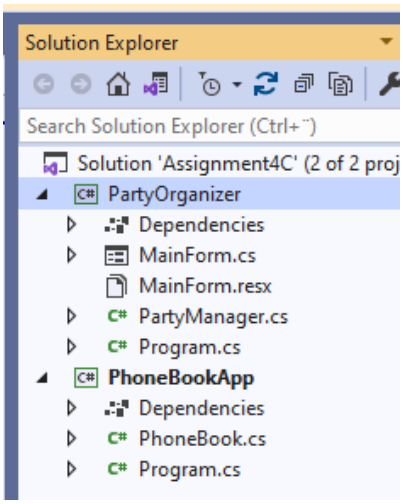

```
//Default Constructor (constructor with no parameter)
1 reference
public PhoneBook()
{
    Console.Clear();
    //1
    DisplayList();
    Console.WriteLine();

    //2
    SortByName();
    Console.WriteLine("\n\nSorted List:\n");

    DisplayList();

    //3 Two-dimensional array
    int count = names.Length;
    phoneList = new string[count, 2];  //2--> name and tel
    FillTable();

    Console.WriteLine("\n\nUsing a 2d array instead of two 1d arrays:\n");
    DisplayTable();
}
```

4. Part 2: Party Organizer

This part must be done as Windows Forms application with a graphical user interface (GUI). In case you have prior experience of Windows Forms, you may try working with Windows Presentation Foundation (WPF). You may also design the GUI and solve the problem in your own way, as long as you follow the main requirements specified here in this document and maintain a good code quality level.



Create a new project in the solution you created in Part 1, but it is also allowed to create a new solution for this project.



You mission in this assignment is to develop a simple application that can be used to organize parties, conferences and other such events. The application should register the names of the participants and get an idea of the total cost based on an estimated cost per guest. Some events have even a fee to be paid by the participants. The application calculates the total revenue collectable from fees and displays the difference (surplus or deficit).

We are going to use a form object (**MainForm**) for the GUI which will be responsible for all interactions with the user, and the application. We create another class, **PartyManager** to contain logics to handle the list of participants and do other calculations and processing of the data.

For storing the names of guests, use a one dimensional array of strings in the **PartyManager** class. **MainForm** should then create and use an object of thePartyManager. We hereafter refer to this object as **partyManager** (with a small p).

5. To Do

5.1 Start Visual Studio (VS), create a new project, Windows Forms Desktop Application to solve this assignment. Design your GUI as in the above figure (or after your own taste). Rename Form1 to **MainForm**.

Important: Write your own name as a part of the Form's title text. In the above GUI image, the title text contains the text "**by (your name)**" which you should replace by your name.

5.2 Use TextBoxes **only** for **input** and Labels **only** for **output**. Use appropriate names for all controls, e.g. txtFirstName, lblNumOfGuests, lstAllGuests.

5.3 **MainForm** should handle all the interactions (input/output) between the user and the application, and it should not contain logics that is (or can be) a part of the **PartyManager** class' responsibility. Every operation concerning the event, except processing of input and display of output, should be coded in the **PartyManager** class.

5.4 The **PartyManager** class should in no way interact with the user. No **MessageBoxes** in this class! It should only serve the **MainForm** and other classes that you may create in the future.

5.5 Compile, run and test your application before turning in.

6. Requirements for Part 2

6.1 The Add and Delete features must be implemented and work properly.

6.2 The Change button is optional and is not required for a Pass grade.

6.3 The **PartyManager** class should only define instance variables for storing **input**, as in the figure below (*costPerPerson* and *guestList*). Do not use any other **instance** variables, but you may (should) use as many local variables (variables inside a method) as you may need.

6.4 Write also a constructor with one parameter in the **PartyManager** class, (see the code snippet below).

6.5 All **output** values should be provided to the caller through the return statement of the method which you write for calculation of a value. For small calculations, a read-only Property, i.e. a Property with only get-accessor, can be used instead of a method.

6.6 The **guestList** array should be sized with a capacity given by the user on the GUI. It should store names of every guest in the format "firstName all caps" + "," and "lastName", e.g. "SIMPSON, Homer".

6.7 **MainForm** should only have one instance variable, the **partyManager**

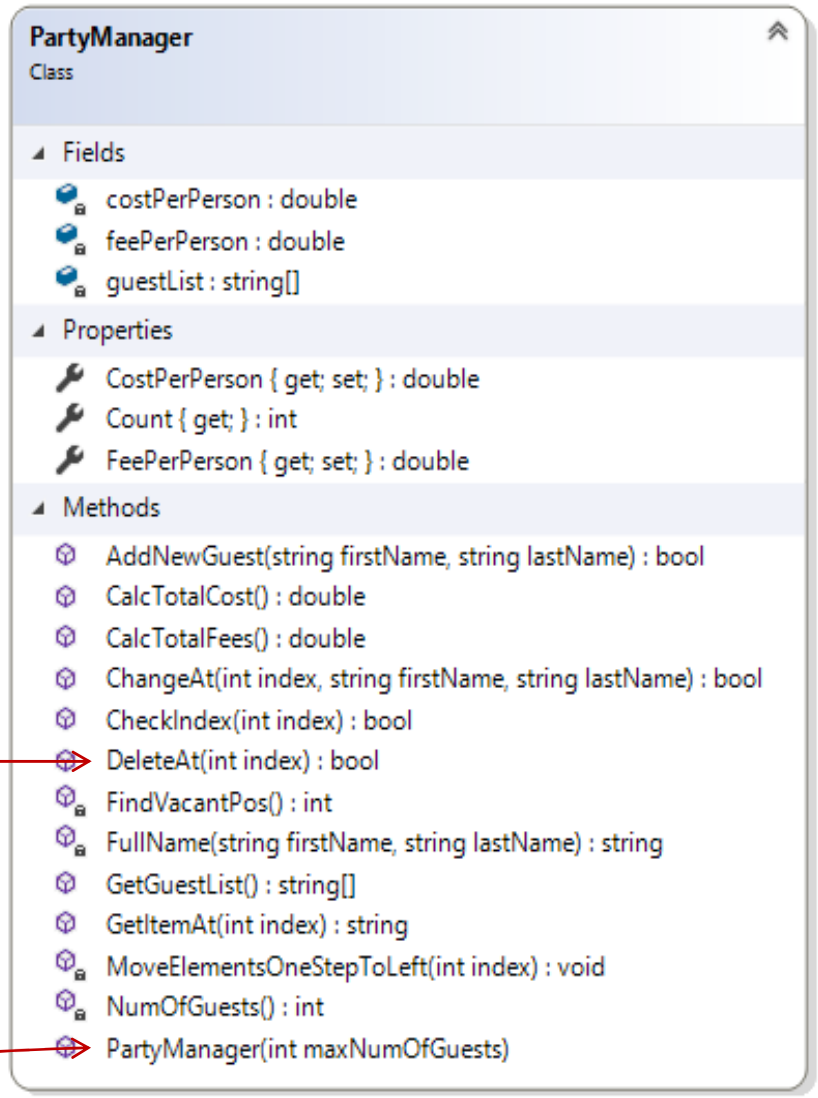
The PartyManager class

6.8 The size of the array is set by the caller method which in this case is the **MainForm**. The constructor is called automatically when an object the **PartyManager** is created in the **MainForm** by using the keyword new. Because the constructor has a parameter of int type, **MainForm** must supply the value (the size of the array) when calling the constructor.

```
class PartyManager
{
    private double costPerPerson;
    private double feePerPerson;
    private string[] guestList;

    //Constructor - expect maxNumOfGuests from the
    // caller (MainForm) to set the size of the array
    public PartyManager (int maxNumOfGuests)
    {
        //Create the array here in the constructor
        guestList = new string[maxNumOfGuests];
    }
```

6.9 Write properties (both get and set) for the instance variables, **costPerPerson** and **feePerPerson** but NOT for the **guestList**. Arrays are objects and passing of objects takes place by reference. As such, giving access to the array **guestList** is providing the address of the variable to the caller. Any changes in the caller method will also affect the **guestList**.

6.10 Write other methods which you find necessary to add a new participant, change or remove an existing guest and so on. A class diagram is given below, but you do not have to write all the methods listed there. Write methods that you need to use in your solution.



Move elements at right of the element being deleted one step to the left to avoid empty slots (explained later)

The constructor, with one parameter used to determine the size of the array be created.

The MainForm Class

6.11 Write a method **InitializeGUI** for preparing the GUI. In this method, you can change title of the Form, clear output components from design-time text and do other initializations. The groupbox component for writing names of a guest (see GUI below) should be disabled at start because the array must be created first.

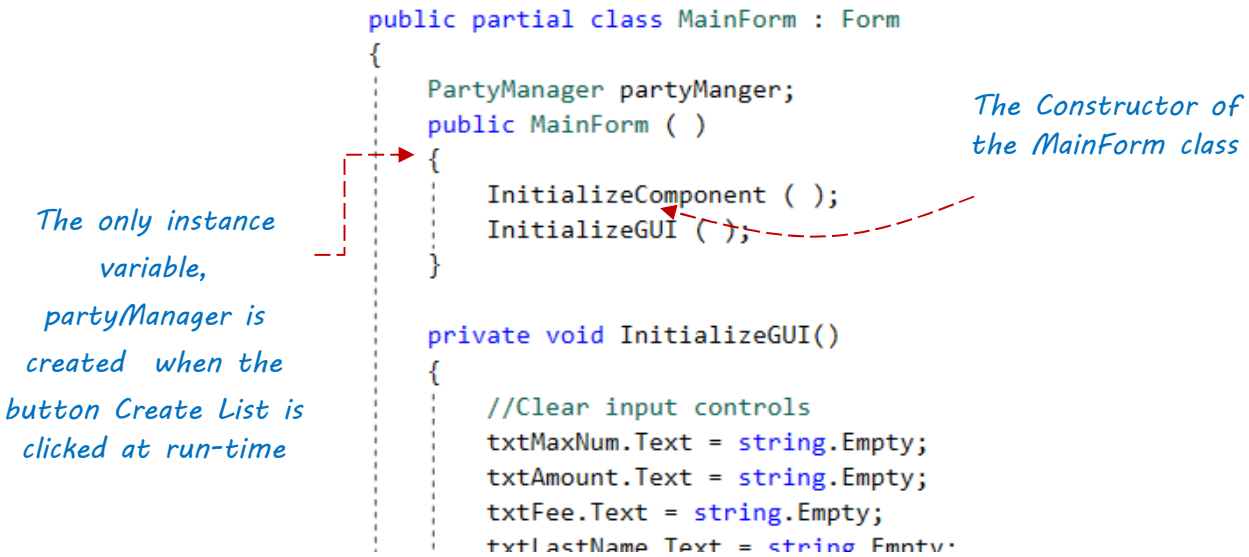
When the application is started:

6.11.1 The textboxes are to be empty.

6.11.2 The GroupBox, **Invite Guest**, is to be disabled.

6.11.3 The user can use the textboxes to specify the total number of guests, which is then used to size the array.

6.11.4 The user can give the cost and fee per guest.



Start a new party:

6.12 Every time a new **PartyManager** is to be planned, the user gives the total number of guests, the cost per person as well as a fee (revenue) per person. These data are given in the first three textboxes, as in the figure.

When the user clicks on the **Create List** button, code in the **MainForm** to perform the following:

6.13 Get the values from the textboxes and validates them (using int.TryParse and double.TryParse). If the control is successful, proceed as follows, otherwise, give an error message to the user and wait for new input.

6.14 **MainForm** creates the object **partyManager** using the keyword new. Every time the partyManager is created (re-created), the old values will be lost. The **PartyManager** class should have a constructor with one parameter for the size of the array which is the same as the total number of guests. Use new and pass the size as a parameter.

6.15 **MainForm** also saves the cost per guest in the **partyManager** object so it can compute the total costs as guests are added. Use the **partyManager** object's related Property to save the cost per guest. The same procedure is to be used with the fee value.

6.16 Enable the **GroupBox**, **Invite Guest**, in order for the user to begin adding guests.

Add a new guest

6.17 The user can now give a first name and a last name for each guest to be invited. When the user clicks on the **Add** button, the following happens:

6.17.1 **MainForm** reads the contents of the two boxes (the first and the last names), and checks so they are not empty strings in which case an error message is given to the user and no more processing is done

6.17.2 The names are validated, then **MainForm** calls a method of the **partyManager** object to add the guest with these two as parameters.

```
bool ok = partyManger.AddNewGuest(txtFirstName.Text, txtLastName.Text );
```

6.17.3 The **partyManager** object combines the first name and the last name to **one** string (LAST NAME, first name) and saves the string in the first vacant position in the array.

6.17.4 The **partyManager** object also should have a method that calculates and returns the total cost for the number of guests registered so far.

6.18 When the user clicks the **Delete** button, the **MainForm** should use the index to the item selected in the ListBox. **MainForm** should then call the method written in **PartyManager** class that removes (marks the position as vacant). The user must have selected an item in the list box.

6.19 Every time a person is added or removed, the listbox with the current guests should be updated. This can be easily done by first clearing the contents of the list box and then filling the listbox with all registered guests provided by **partyManger**.

7. Keep array elements arranged from index =0 without empty slots.

When an element is deleted from a position in the array, it provides an empty slot. Having empty (unused) element in different positions in the array provides problems in keeping track of filled and empty locations. The indexes will then not match the index of a list component on the GUI where empty elements are not be displayed.

arrayManager

Apu	Bert		Nani	Dan
0	1	2	3	4

GUI (ListBox)

0	Apu
1	Bert
2	Nani
3	Dan
4	

arrayManager has an empty slot (index 2).
Nani is stored at index = 3 in the arrayManager but it should be displayed at the position 2 on the GUI component.

To fix this problem, you can use the following solution:

Declare an instance variable in the PartyManager class:

```
private int numOfElems = 0; //num of elems with values
```

When adding a new element, add the new element at the index = numOfElems and then increment the variable:

```
if (numOfElems < guestList.Length)
{
    guestList[numOfElems] = FullName(firstName, lastName);
    numOfElems++;
}
```

Every time an element is removed (deleted), decrement the variable: numOfElems--; and meanwhile move all element which are at the right of the index being removed one step to the left.

Use the code below in the **RemoveAt** method (or **DeleteAt**)

```
guestList[index] = string.Empty;
numOfElems--;
MoveElementsOneStepToLeft ( index );
```

```
private void MoveElementsOneStepToLeft(int index)
{
    for (int i = index+1; i < guestList.Length; i++)
    {
        guestList[i-1] = guestList[i];  //move 1 step to left
        guestList[i] = string.Empty;  //empty its place
    }
}
```

This solution can be used with array of other types and objects as well. With array of objects, use null instead of string.Empty in above. Actually, null can be used with array of string too as string is also an object in C#. With array of numeric elements, the default value (empty value) should be used based on the problem.

8. Optional but recommended

In addition to above requirements, the following features should also be implemented.

8.1 The **Change** button should work. When the user clicks an item in the **ListBox**, the corresponding item in the array of the partyManager object should get the values given in the name-textboxes.

8.2 When the user selects an item in the **ListBox**, get the item from the array of the **partyManager object**, split it into a first name and last name and display them in the textboxes. This feature makes it easier for the user to bring changes.

Hint: Use the Split method of the related string variable.

9. Submission

Compress all your files and folders (particularly the folder Properties if you are using .NET Framework) that are part of your project into a ZIP or RAR file. Upload the compressed file via the same page where you downloaded the assignment.

Good Luck!

Farid Naisan,
Course Responsible and Instructor