MALMÖ HÖGSKOLA

Programming Using C#, Basic Course

Assignment 3

GUI and Methods with parameters

Cinema Booking System (CBS)

Version 1- GUI and Input

Mandatory

Farid Naisan
University Lecturer
Department of Computer Science
Malmö University, Sweden

Assignment 3: CBS Version 1, GUI and Input

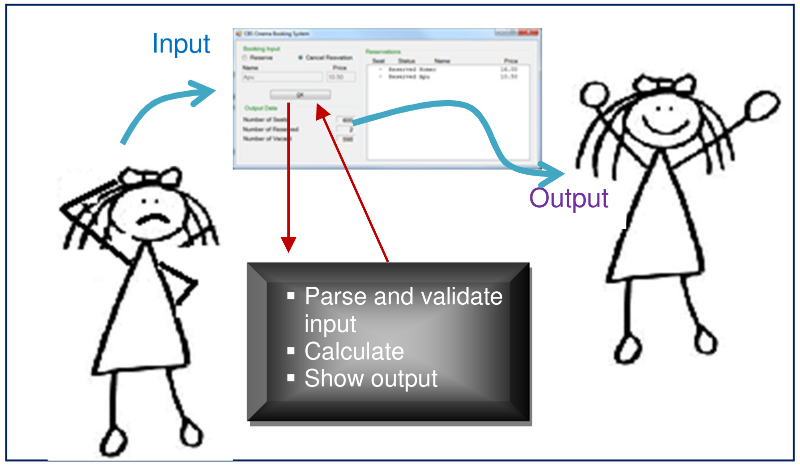
1. Objectives

So far, we have been working with Console applications to learn and test the basic syntax and structures of the C# language. It is now time to move over to more practical cases and create applications having graphical user interface (GUI). From now on, all our applications, examples, exercises and assignments will be GUI-based.

The main objectives of this assignment are:

- To begin writing Windows Forms based desktop applications with graphical user interface (GUI) using some of the most common Windows Form controls.
- To get input from different Windows Form controls and save them in variables.
- Perform input validation, using the **TryParse** methods.
- Use parameterized methods to establish communication between objects.

2. Introduction



Almost all computer programs need to receive input from the user. The input is to be saved in objects of classes for manipulation and producing results (output). However, the values given via controls usually have types that need to be converted to types that you would like to work with. As an example, we would like to work with a numeric value of type double in this assignment. We provide a TextBox control on the GUI for the user to write in a numeric value. The TextBox stores the input as a text in its Text property, for example **txtPrice.Text**. The

Text property has a string data type but we would like to save the data in a variable of type double. Therefore, we need to convert the text, for example “9.95” (string) to a corresponding numeric value, i.e. 9.95 (double value with no quotation marks). However, after processing data and doing calculations, we might have a numerical value as output that should be displayed on the screen using a control for example a Label. The Label control also has a property named Text than can contain data of the type string. Here, we must convert our numerical value to a corresponding string in order to be able to save in a Label. Thus, conversion to a type is necessary in both directions in most of the cases. To that comes also the question as to whether the converted data is within a valid range. This assignment is intended to give you practice in this type of problems.

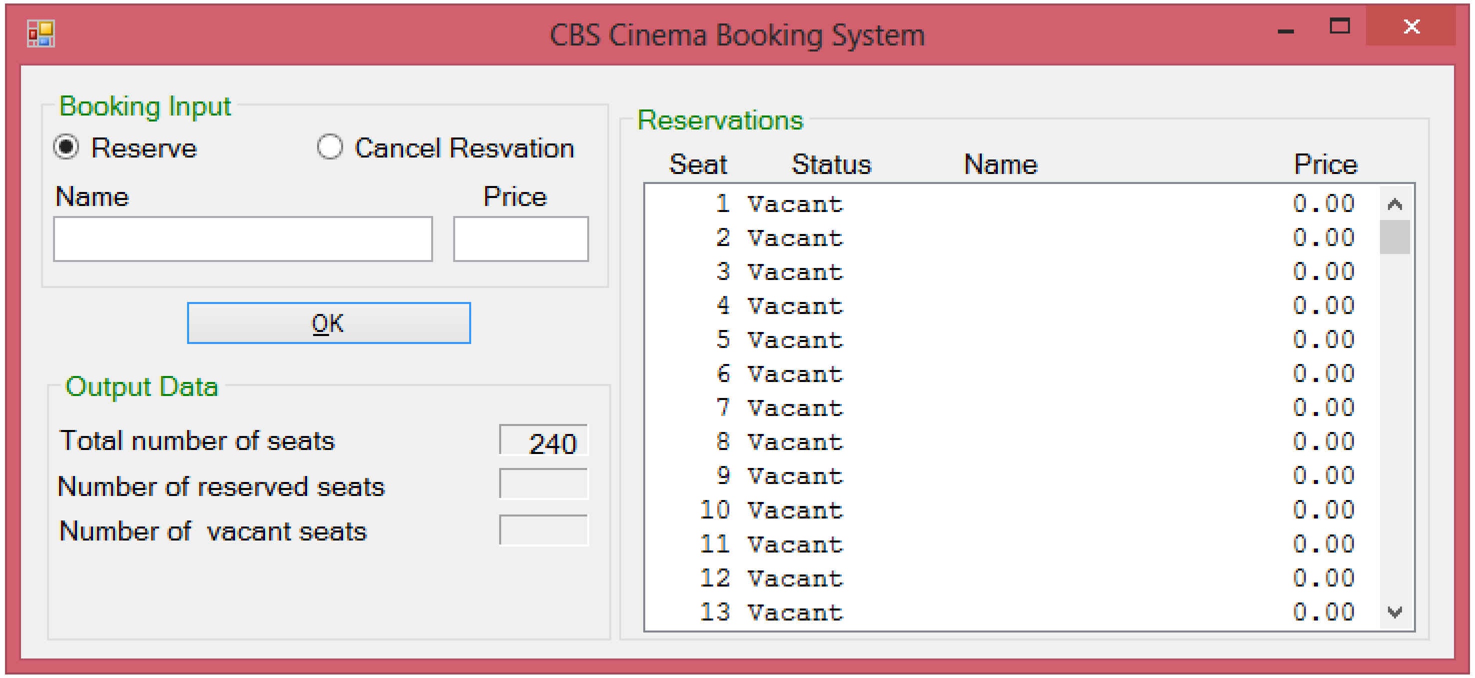
In this assignment we will work with the above two issues, i.e. **parsing** and **validating** the user input given through **TextBoxes**. We use **Labels** and a **ListBox** for the presentation of results. In addition, we handle user input through mouse clicking on buttons.

Note: In this assignment we work with the Form and Controls to get values from the user. We test the GUI so the input handling works well. We put more logics in the program in the next assignment. So make sure to save and back up your project files to continue in the assignment..

3. Description

A new movie theater has opened in your town and the owner needs a system that facilitates the reservation of seats in the cinema’s auditorium. Your assignment is to write a GUI-based application that facilitates the reservation of tickets for this movie theater. The user of this application is a cinema staff, for ex the Cashier. When a customer wishes to reserve a seat in the movie, the Cashier registers the name of the customer and the price for the seat using your program.

The GUI below is to be designed using Windows Forms and Controls. For those who are already familiar with Windows Forms and wish to try something new, creating a Windows Presentation Foundation (WPF) project is recommended.



```
   1 Vacant                    0.00
   2 Vacant                    0.00
   3 Vacant                    0.00
   4 Vacant                    0.00
   5 Vacant                    0.00
   6 Vacant                    0.00
   7 Vacant                    0.00
   8 Vacant                    0.00
   9 Vacant                    0.00
  10 Vacant                    0.00
  11 Vacant                    0.00
  12 Vacant                    0.00
  13 Vacant                    0.00
```

How is the program functioning? Here is a possible scenario:

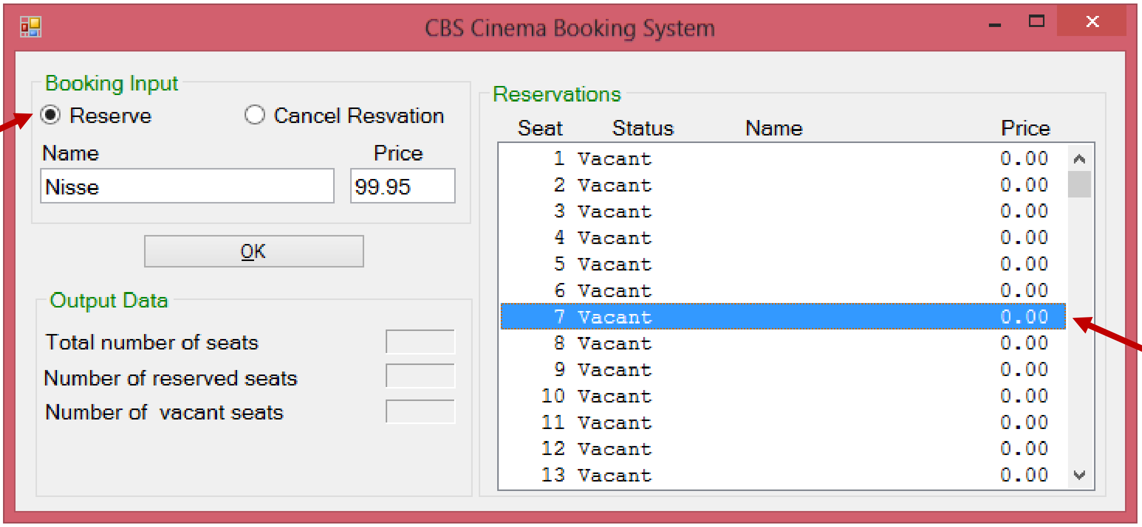
A customer will make a reservation. The user (Cashier) uses the GUI to give input. The program does nothing.

3.1 The user writes a customer's name in the TextBox object (**txtName**) at left and the price to be paid by the customer in the TextBox object (**txtPrice**) at right. The user may also click on the option buttons **Reserve** or **Cancel Reservation**.

3.2 The Cashier clicks on an item in the ListBox object (**lstReservations**) at right so an item is highlighted.

3.3 The program does not do anything until the user clicks the OK button.

How to detect in the code, if the button selected or not? The btnReserve.Checked has a true value if selected by the user. Other buttons get a false value automatically.



Name	Nisse
Price	99.95
Total number of seats	
Number of reserved seats	
Number of vacant seats	

```
Seat  Status    Name                 Price
   1  Vacant                          0.00
   2  Vacant                          0.00
   3  Vacant                          0.00
   4  Vacant                          0.00
   5  Vacant                          0.00
   6  Vacant                          0.00
   7  Vacant                          0.00
   8  Vacant                          0.00
   9  Vacant                          0.00
  10  Vacant                          0.00
  11  Vacant                          0.00
  12  Vacant                          0.00
  13  Vacant                          0.00
```

Item no 6 (counted from 0) is highlighted. The information can be obtained in the code:from

lstReservations.SelectedItem

The user clicks the OK button to make a reservation or cancel a reservation. The program (your code) should act now:

3.4 **Input Validation:** When the user clicks the OK button, the following tasks should take place in the your code:

3.4.1 The customer name, the price are parsed from the textboxes and saved into variables in the application. The validity of the two values are controlled so that:

3.4.1.1 Name is not allowed to be an empty string. Call a method that you will be writing in the **InputUtility** class to check this.

3.4.1.2 The price should be a numerical value >= 0.0. Call a method that you will be writing in the **InputUtility** class to convert the user input to a double.

3.4.2 If the control fails, the user should be given a message and the program should not do any more, return back and let the user fix the problems.

3.5 If the values are OK, the program should then check if the user has selected an item in the ListBox. If no selection is made, i.e. no item is highlighted, inform the user with a proper message and the program comes back to show the GUI and let the user click on an item in the ListBox. How do you if an item is highlighted? If the **lstReserverations.SelectedIndex** >= 0, it is ok. The item selected is the value saved in SelectedIndex.

```
int index = lstReservations.SelectedIndex;
```

3.6 The RadioButtons usually do not need any control as after the first click, there will always be one of buttons checked. How do you know if a button is checked? **rbtnReserved.Checked** is then true!

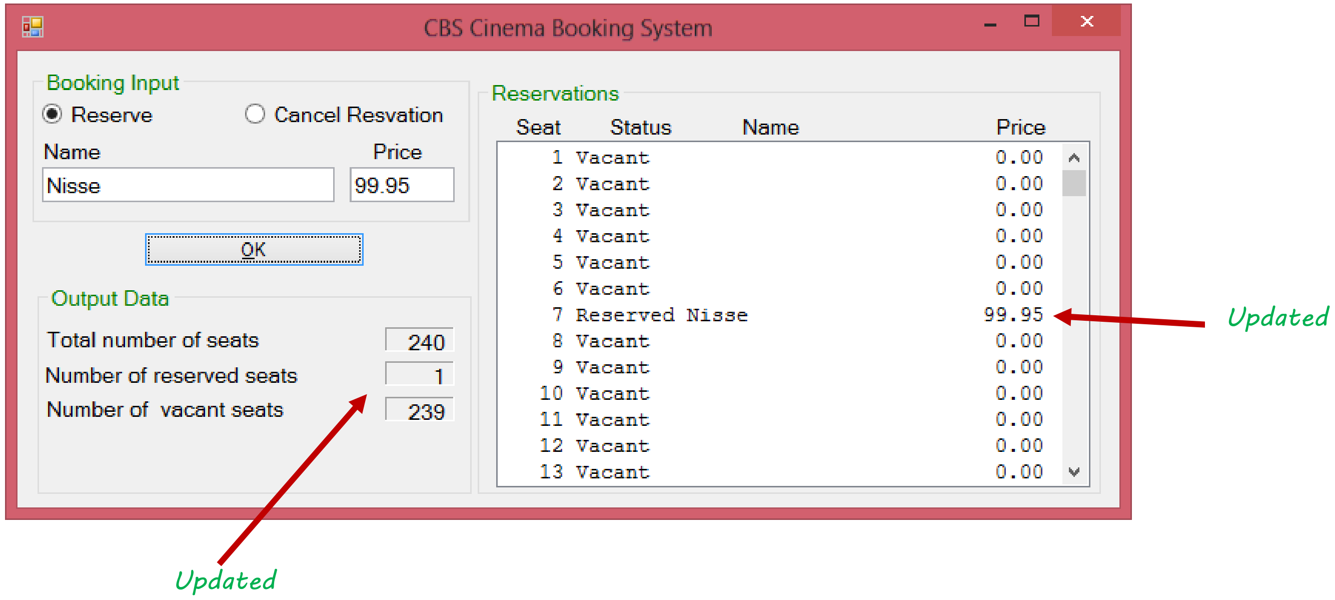
3.7 **Update output**: When the input is validated. Depending on whether the **Reserve** or **Cancel Reservation** button is checked, the following actions take place:

3.8 If the Reserve button is checked (see also the figure below):

3.8.1 The number of reserved seats (use a variable to save this data) is incremented by one (the value is 0 at the beginning).

3.8.2 A formatted string with the customer name and price is prepared. Send the string to the ListBox to replace the selected item in the ListBox with the updated info as shown below. How do you do this, one possible way is to remove the item and insert the new string at the same index. You can use the lstReservations.Items.**RemoveAt** och lstReservations.Items.**Insert** in correct order**.**

3.8.3 The program also should update the values grouped in the Output Data box.



Booking Input: ● Reserve ○ Cancel Resvation

Name	Price
Nisse	99.95

Output Data

Total number of seats	240
Number of reserved seats	1
Number of vacant seats	239

Reservations

```
Seat  Status    Name          Price
   1  Vacant                   0.00
   2  Vacant                   0.00
   3  Vacant                   0.00
   4  Vacant                   0.00
   5  Vacant                   0.00
   6  Vacant                   0.00
   7  Reserved  Nisse         99.95
   8  Vacant                   0.00
   9  Vacant                   0.00
  10  Vacant                   0.00
  11  Vacant                   0.00
  12  Vacant                   0.00
  13  Vacant                   0.00
```

3.9 If the **Cancel Reservation** button is checked, the following should be accomplished by your code:

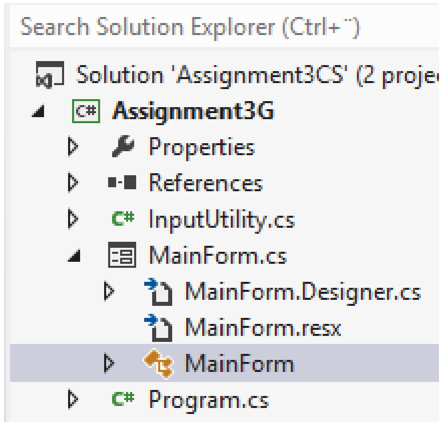
3.9.1 The number of reserved seats is decremented by one.

3.9.2 Both of the textboxes are to be disabled.

3.9.3 The selected item in the ListBox is replace by a string showing a vacant seat (as other vacant seats with name = string.Empty and price = 0.0).

3.9.4 The number of vacant and reserved seats are updated accordingly.

4. To Do


Search Solution Explorer (Ctrl+¨)

Solution 'Assignment3CS' (2 proje

Assignment3G

Properties

References

InputUtility.cs

MainForm.cs

MainForm.Designer.cs

MainForm.resx

MainForm

Program.cs

To begin with, open Visual Studio and this time create a Visual C# Windows Form application instead of a Console Application. Give a good name to your solution and project (name of your program) and click OK. VS creates and prepares the project and makes its initializations. You will be then getting a start form (Form1) to begin drawing your user-interface on. Remember, you do not need to worry about the method **Main** when working with Windows Forms in VS. You begin directly with Form1. Right-click on Form1 in the project window (a window at right side in VS) and rename **Form1**.cs to **MainForm**.cs. VS will ask you for permission to rename even the Form1 class accordingly. Accept the offer and you can now proceed as explained (very roughly) below:

4.1 In this assignment, only two classes are needed (but you may of course create more if you get an idea for a better solution). The classes are:

- **MainForm** (renamed Form1) for designing GUI and handling interaction with the user.
- **InputUtility** – a utility class for parsing values and converting them to a datatype of your choice, for instance parsing a double value from a textual representation. The methods should also perform validation of the converted values.

4.2 Begin with designing the user interface (GUI) using suitable controls for input and output. Controls are to be given suitable names. Label1, TextBox1, etc. have to be changed to proper names so that you can easily remember them in your code. It is recommended that you begin the control names with a three or four letter prefix, e.g. lbl for Label, txt for TextBox, lst for ListBox, btn for Button, grp for GroupBox, rbtn for RadioButton and so on. After shaping up the GUI, you can leave the MainForm and finish the classes with logics– in this assignment we have only the InputUtility class.

4.3 Write an InputUtility class as instructed later in this document. When you are done, run and test your project so you do not have any compilation errors.

4.4 Go back to MainForm and write code to make the GUI work as specified in the scenario.

All these steps are controlled by the scenario described earlier as well as the requirements that are listed below. **However, you are allowed to bring changes to the GUI, and you do not have to follow the instructions step-by-step as long as you meet the requirements and that you come up with a solution that is better.** To produce a program that works well but is poorly programed is not what we are expecting. You must apply well-structured and documented style. Think always object-oriented!

5. Requirement for a Pass grade (ECT C, Swe G,):

THE HELPER CLASS – InputUtility

5.1 Create a utility class with methods to perform the following tasks:

5.1.1 A method, **GetInteger**, that converts a given string to an integer (**int** or System.**Int32**) and returns **true** if successful, **false** otherwise.

```
public static bool GetInteger(string stringToConvert, out int intOutValue)
```

This method can then be used to parse an integer value from the contents of a TextBox provided that the TextBox contains text that can represent a valid integer. This method is going to be useful in the future projects.

5.1.2 A method **GetDouble** that converts a given string to a double and returns **true** if successful, **false** otherwise.

```
public static bool GetDouble(string stringToConvert, out double dblOutValue)
```

This method can then be used by the MainForm to parse the price of a seat from the contents of **txtPrice.Text** provided the **txtPrice** contains text that can represent a valid double ("9", "95.99" or "95,99" if your Windows uses a comma as the decimal sign).

5.1.3 Write a method that validates so a given (`double` or `int`) number is not negative. This can be embedded into the above methods but can also be one or two separate methods.

5.1.4 A method which controls that a given string is neither **null** nor empty and should at least have one character other than a blank space (or escape sequences). **HINT**: The method: `string`.**IsNullOrEmpty**(strArg) is helpful for this purpose.

5.2 When converting from a text to a number, the string can be the content of a TextBox (e.g. txtPrice.Text) or any other textual representation of a numerical value.

5.3 In all conversions, you must use the respective data type's **TryParse** (double.TryParse and int.TryParse) to validate if a given text can represent a valid floating point number or an integer.

Remember: This is a utility class that you can use in all your future projects. In this assignment, we might be using only a couple of the methods that you write in this class.

THE CLASS – MainForm

5.4 Appropriate names are to be used for the Solution, the Project, the Form and all the input/output controls. However, the controls that contain static data, i.e. data that does not change during the program execution and which are not referred to in your code can maintain their default names. As an example, the label containing the heading "Total number of seats" may be named Label5 or similarly.

5.5 TextBoxes are to be used only for editable text, i.e. input of data.

5.6 For non-editable text (read-only text) such as headings, result of calculations (output), Labels are to be used. The boxes at the right side of the Output Data in the figures given above are all labels. The heading above the ListBox are also Labels. **Do not use textboxes even if you disable editing and change the colors to make them function and look like a label!**

5.7 You may use instance variables in the **MainForm** to store the values:

- Customer name
- Pric
- Current number of reserved seats
- Total number of seats (capacity) = 240 or any other constant number.

No instance variable for output data is needed and should not be used.

5.8 Write a method **InitializeGUI** in which you clear all output controls (textboxes, ListBox) from design-time text (Textbox1, etc.). Call this method from the MainForm's constructor. Write code to fill the ListBox showing all items as vacant (see the figure given earlier).

```
public MainForm()
{
    // This call is required by the designer.
    InitializeComponent();

    //Initialize all input/output controls
    InitializeGUI();
}
```

5.9 Write a method **UpdateGUI** that updates the GUI after each change in the results. This method can be called when the button OK is pressed (after reading input and performing necessary manipulations).

5.10 Call the method **ReadAndValidateInput** (see below) to fetch the price and name of the current customer given in the TextBoxes.

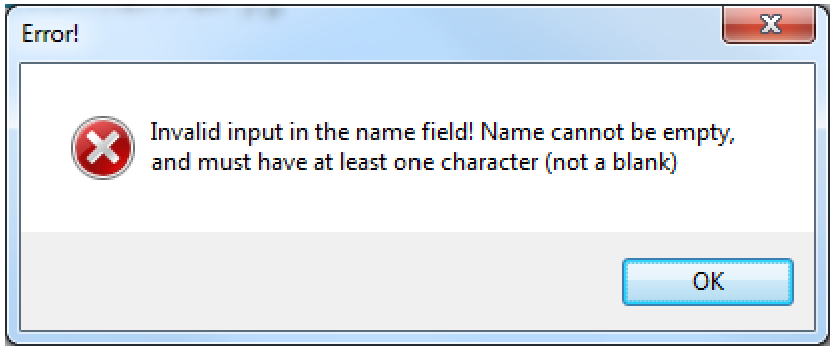
5.11 If this method returns **true**, proceed with the next step:

5.11.1.1 Call the method **UpdateGUI** to update output, the number of reserved/vacant seats and add a string to the ListBox).

5.11.1.2 Otherwise, proceed as follows:

5.11.1.3 Give a friendly message to the user of invalid input.

5.11.1.4 Exit the method.



Invalid input in the name field! Name cannot be empty, and must have at least one character (not a blank)

5.12 Read and validate input

5.12.1.1 Write a method that starts the input processing in the MainForm class:

```
private bool ReadAndValidateInput()
```

5.12.2 This methods should in turn call two other methods in the **MainForm** class:

```
private bool ReadAndValidateName()

private bool ReadAndValidatePrice()
```

5.13 Each of the above should validate the out parameter, **name** and **price** respectively, and return true if validation is successful and false if the value is invalid. If the conversion and validation is successful, the parameters **customerName** and **price** will have valid values.

5.14 **MessageBoxes** used to give error messages to the user, should have a caption and an icon, as those given earlier in this document. As an exercise, look up the documentation about **MessageBox** class (.NET 3.5 or above) at MSDN (msdn.microsoft.com) and learn how the buttons and icons are combined for different cases.

5.15 All methods must be commented using the XML-compatible format "///<". You must know the benefits of using this type of comments to document your classes and their members. Discuss this in the forum if you need to have more clarity about this type of comments.

6. Requirement for a higher grade (ECT A and B, Swe VG,):

All of the requirements listed above for a Pass grade is to be met with the following exceptions or additions. If you are good with a Pass grade and do not intend to aim at a higher grade, you may skip this section.

6.1 When reading and handling user input**, do not** declare any instance variable for storing the name of the customer or the price of the ticket in the **MainForm**. Use methods with out-parameters and let the methods communicate through passing values as parameters. (You may declare other constants and instance variables if needed).

6.2 Overload the **GetDouble** and the **GetInteger** so they can be called with the **minLimit** and **maxLimit** parameters. This method can be used to check if a given string can be converted to numerical value and also if the value is within the range minLimit and maxLimit (inclusively)

```
public static bool GetInteger(string stringToConvert, out int intOutValue, int minLimit, int
maxLimit)
```

6.3 Complete the above method and write a similar one for **GetDouble**.

6.4 Overload the **GetDouble** and the **GetInteger** methods so they can be used when a string is to be converted to a numerical value with only the **minLimit** condition. For example when you wish to get a value >= 0 for the price – no max limit is required. This method should call the above method, passing **Integer.MaxValue** for the last parameter.

Alternataive to out: If you do not like the **out**-parameter, there is a good alternative to that. You can write methods that return a **struct** with members to hold the converted value, a Boolean to flag success or failure for the conversion, etc. However, I think working with **out**-parameter is necessary as it is a part of the C# language and you should know how it works. It is used by the objects in the .NET Framework (TryParse for example).

7. Help and Submission

7.1 General help and guidance for this assignment will be available in a separate document in the module. Test your application well before submission. **Projects that cannot be compiled or are poorly tested will be returned immediately for resubmission.**

7.2 Compress all your files and folders (particularly the folder Properties) 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! - *Programming is fun. Never give up!*

Farid Naisan,
Course Responsible and Instructor