PCS4 Assignments – Week 1

Version: November 2014

NOTE: The assignments marked with an asterisk are the most important ones.
The others serve as extra practice.

WEEK 1

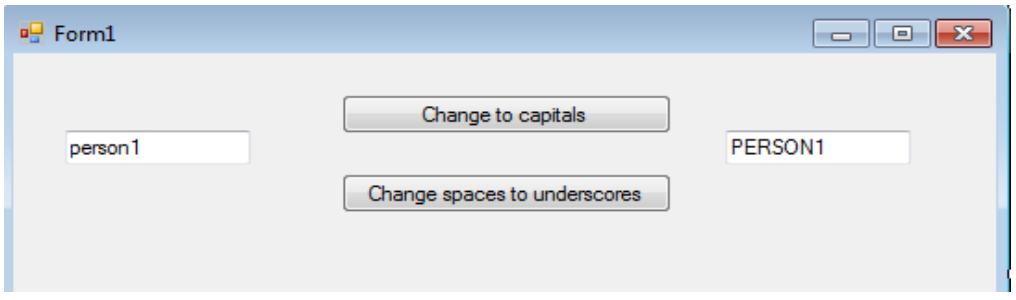
* Assignment 1.1: Using delegates to a simple string-manipulation application

We ask you to write a program that allows you to modify strings in all kinds of ways. For example there must be an option to change all the letters in the string to capital letters (so the string "Person1" will become "PERSON1"), or another option that should be provided is to replace all the space-characters in a string to underscore-characters (so the string "Hello World" becomes "Hello_World"). But other options will be added later on.
Although you can write the program without delegates we require from you to make use of delegates.

STEP 1:
First create a new windows application and add 2 textboxes to the form. One at the left side where the user can enter a string (named textBoxIn) and another one at the right side with the name textBoxOut, to show the result after changing the string.

STEP 2:
For every string-modification-variant that must be provided by the program you need to write a separate method. All these methods have the same signature:
- one parameter: a string (the original string that must be changed)
- the return value is also of type string (the new string after the modification)
So for the two examples mentioned above you must write 2 separate methods (named ChangeToCapital and ChangeSpaceToUnderscore). Later on we will add more methods for other kinds of modifications.
(To convert a string to an array of char or vice versa you can use the statements:
`char[] arr1 = str.ToCharArray();` or: `string s = new string (arr1);`
But you can also find useful information about how to modify strings on the internet, for example : http://msdn.microsoft.com/en-us/library/ms228599.aspx)

STEP 3:
You must also add for every string-modification-variant a button on the form. If the user enters a string in the left textbox and clicks one of these buttons the modified string must be shown in the textbox at the right side. Give each button a clear text, so that the user knows what should happen with the string when he clicks that button. At this moment 2 buttons are needed (one for the ChangeToCapital method and one for the ChangeSpaceToUnderScore method). Implement these button-clicks and test your methods.



STEP 4:
Now we are going to replace some code to make use of a delegate. As you already saw each string-modification-method has the same signature. So we can replace them by a delegate-variable that holds a reference to the right method. To realize this perform the following actions:
- First declare a delegate type StringChange, again with that same signature: one parameter (a string) and a string as return value.
- Next declare a variable stringChange of type StringChange in your Form class.
- Finally change the code in the button-clicks: instead of using one of the string-modification-methods, now use that variable stringChange (after you assigned the right method to that variable).

(No changes are necessary to the string-modification-methods you already made in the previous steps, only the way how they are used has been changed.)
Test your program again.

STEP 5:
Now we ask you to extend the program in the following way: change the 2 textBoxes to richTextBoxes and adjust the button-clicks so that instead of modifying just one string now all the strings contained in the richTextbox at the left side are changed and shown in the richTextbox at the right side, after first clearing that right richTextbox. (You don't need to use the delegate variable here.)

STEP 6:
As you can see, both button-click methods look almost the same. We don't like to write (almost) the same code several times. Because you already introduced the delegate StringChange, it is easy now to improve your code and prevent writing the same code twice:

- Write a new void method (named Change) with one parameter, this parameter is of the delegate type StringChange. Method Change must have the same implementation as each of the button-click methods, but instead of calling one of the string-modification-methods (ChangeToCapital or ChangeSpaceToUnderScore) this method makes use of the parameter, which can be considered as another string-modification-method.
- Once this method is written you can replace the whole content of the button-click method by just one call to that newly written method Change with the correct string-modification-method as a parameter.

STEP 7:
Finally add some more string-modification-methods to the program, together with a button every time and implement the button-click again by just calling the method Change with the correct method as parameter. Add in this way at least two new modifications:
- one that removes all the vowels in the strings (so the string "Person1" will become "Prsn1"),
- one modification chosen by yourself

* Assignment 1.2: Using events for monitoring machines

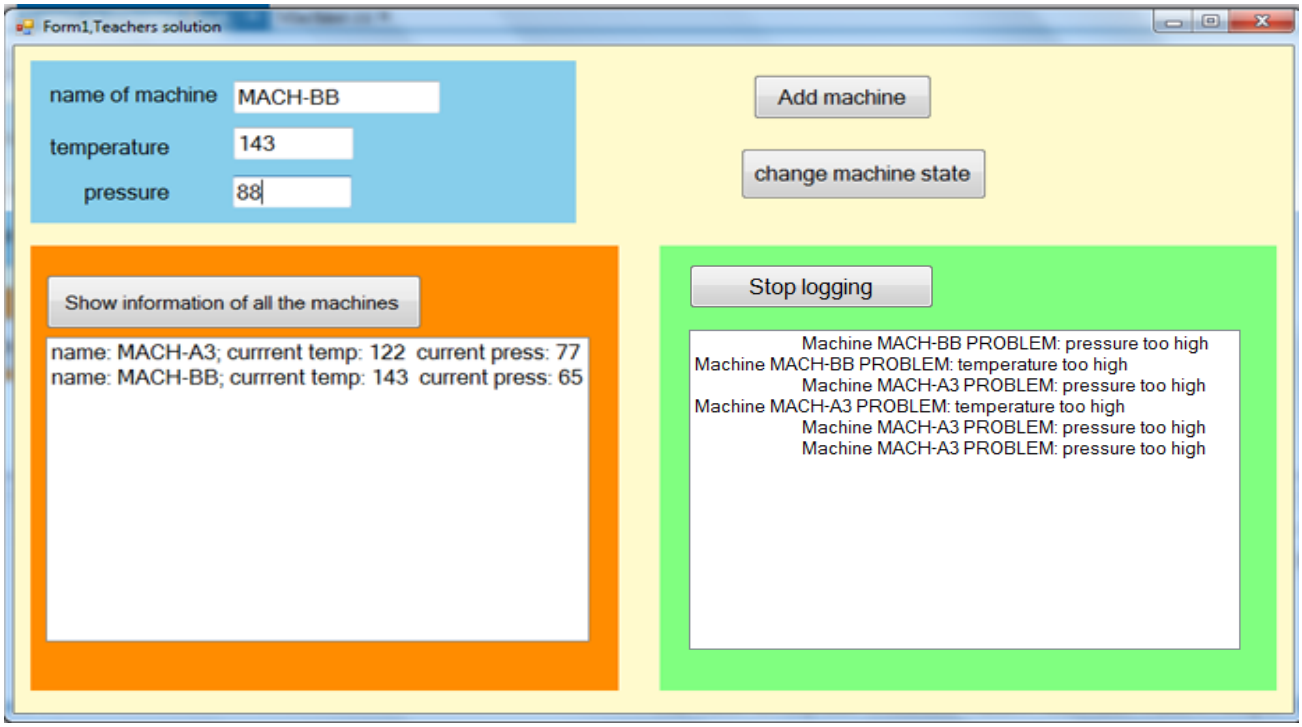
A company uses several machines. These machines have a certain working temperature and a certain working pressure. If this temperature becomes too high (more than 120 degrees) or too low (less than 80) the produced products might be of bad quality. That means that the working

temperature should lie between these values. Something similar holds for the pressure and the upper and lower limits for the working pressure is 80 resp. 60 for every machine.

Most of the time a machine can regulate its temperature and pressure, but sometimes it cannot regulate itself. If temperature or pressure is out of range, the system should display a message on the screen, so the problem can be fixed.

We are going to write an application to keep track of the state (both temperature and pressure) of the machines. In reality every machine sends periodically its temperature and pressure to an application. Our application gets the possibility to simulate such machines sending their temperature and pressure. And if temperature or pressure is out of range, our application will display a warning (by means of events).

The GUI of our program looks like this:



In this example there are 2 machine in use (look at the orange panel): The current temperature of the first machine, Mach-A3 is 122 degrees and its pressure is 77. In the top-left blue panel you see the machine-name MACH-BB. Pushing the button "change machine state" must result in a warning, since its new pressure will become 88 which is more than 80.

On Sharepoint you can find the project to start with. It only includes the controls you need on the form without any code.

STEP 1: the Machine class
Add a new class to your project, named Machine, according to the following class diagram.

Machine
- temperature: string - pressure: string + Name: string
+ Machine(String name, int temperature, int pressure) + SetTemperature(int newTemperature):void + SetPressure(int newPressure):void + String ToString():string

The method SetTemperature enables the user to change the machine's temperature (into the value passed as an argument). The same hold for the method SetPressure.
The method ToString returns a string with the machine's name together with its current temperature and pressure.

STEP 2: The user interface.

- At startup an (empty) list of Machines must be created in the Form class. Program it.
- In the blue region you can type the name of a machine, its current temperature and its current pressure. Clicking the "Add machine"-button creates a new machine and adds it to the list (if the name is unique). Program it.
- Program the button with text "Show information of all the machines", which shows this information (about all the machines in the list) in the left listbox , after first clearing the old content of this listbox (lbInfo).
- The button "change machine state" also works with the information in the blue region. This button changes the current temperature and the current pressure of the machine with the given name. Now, the name of the machine must already exist in the list otherwise an error- message must be displayed.

STEP 3: Delegates and events.
Add two events to the Machine-class:

- an event (called CriticalTemp) for the temperature becoming out of range
- an event (called CriticalPressure) for the pressure becoming out of range.

Both events must be based on the same delegate, which must also be declared inside the Machine-class. This delegate (with the name OutOfRangeHandler) has return-type void and has two parameters:

(1) Machine sender, the machine that causes the problems
(2) String reason, to inform about the kind of problem

Now make sure that each of these events get raised at the right moment (every time when the temperature or the pressure of a machine gets out of range).

Add an event-handler-method to the Form1-class to show a messageBox . This method should fit to the delegate as used for the events, of course. That means this method also has the 2 parameters as mentioned above. Both arguments must be mentioned in the message shown in the messageBox (for example "Machine <machinename> : temperature is too high" or "Machine with name < machinename>: pressure is too low" etc.)

Finally attach this event-handler to both events of each machine-object.

STEP 4: Test the program

Assignment 1.3: Extension of assignment 1.2: Logging the errors

Sometimes we want to log all the error-messages in a listBox, but not always. From the moment somebody clicks the "Start logging"-button, the system should start logging . Every time a warning occurs, the message should be added to the right listBox (lsWarnings), but the messageBoxes also have to be shown like before.
To make a clear distinction in the listbox between the 2 kinds of warnings (about temperature or about pressure), messages about pressure should be indented several spaces. (Look at the screenshot above.)

When you click the "Start logging"-button, the text of the button changes in "Stop logging". If you click the button again, it stops logging and the text changes in "Start logging".

Assignment 1.4: Use Events to make a wake-up application

Write an application that can be used to send an alarm to anyone who has subscribed for receiving an alarm at a certain time.

To realize this, first create 2 simple classes:
- one class Clock, with 2 private fields to keep track of the time (both integers): one for the hour, one for the minutes. Add for each of these two fields a read-only property.
  Add two methods, one to increment the hours by 1 and the other to increment the minutes by one. (Of course you can use a timer to change the time every minute, but for testing we need a faster way to increase the time.) And also add a constructor to the class.
- one simple class Person with only one private field: a name (string) . No property is necessary only a constructor. (Or , if you prefer, you can use a class Person already written in previous assignments instead of writing a new class again.)

Create an object of each class in your form-class.
Add 2 textboxes on your form to display the time of your Clock-object (one for the minutes and the other one for the hours) and also add 2 buttons. Each button enables the user to go forwards in time: one to increment the hours by one, the other one to increment the minutes by one. (Otherwise you must wait a whole day before the alarm time occurs.) Make sure that the textboxes always display the correct values (so they must be updated each time a button is pressed).

This was all introduction, now we need to implement the event(s):
- In the Clock-class: add the following pieces of code:
  - Declare a delegate type AlarmHandler with return type void and with a string as an argument.
  - Add in this same Clock class an event (with the name AlarmEvent8) of the type AlarmHandler.
  - Make sure that this event get raised at the right moment (whenever the time becomes equal to 8 o'clock). The string passed to the event must be something like: "Wake_up, it is 8 o'clock".

- In the Person-class a method must be added that can be used as an event-handler (so it must have the same signature as the AlarmHandler delegate). This method displays the

passed string in a messagebox together with the name of the person. So you get something like: “John: Wake_up, it is 8 o'clock”.

- Finally attach in the constructor of the form-class this event-handler (which is a member of Person) to the event from class Clock.

Test your program: click several times on the buttons until the time becomes 8 o’clock. The messagebox must pop up.

EXTENSION:
Add more persons, some with the same wakeup-time, others with a wake-up time of 9 o’clock. Add code where necessary, to get the correct messages at the correct time(event).