PCS4 Assignments – Week 1

Version: 2016, April

[](http://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRxqFQoTCLHmi7jDhckCFcJ4Dgod7EQDzQ&url=http://www.potentash.com/2014/07/09/delegation-and-shooting-yourself-in-the-foot/&psig=AFQjCNEfFZs7GWqflwcgb0RIYubbJdNPXg&ust=1447232883362100)

[\* Assignment 1.1: Using delegates for a string-manipulation application 2](#_Toc449122033)

[\* Assignment 1.2: Using events for monitoring machines 4](#_Toc449122034)

[Assignment 1.3: Monitoring machines, Serializable in combination with events 7](#_Toc449122035)

[Assignment 1.4: Use Events to make a wake-up application 8](#_Toc449122036)

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

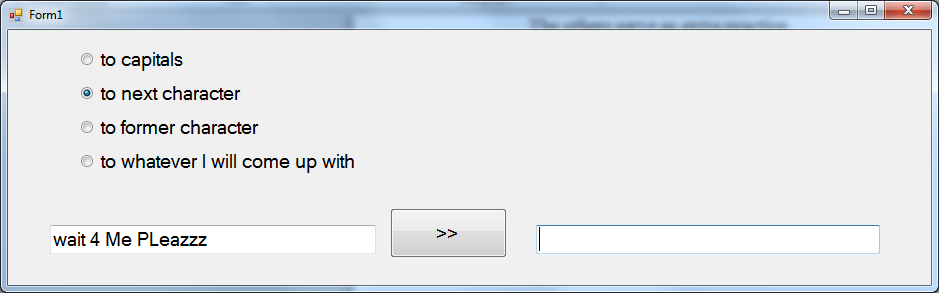
## **\* Assignment 1.1: Using delegates for a string-manipulation application**

We ask you to write a program that allows you to manipulate 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 “Carla13” will become “CARLA13”), 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”), or to replace every character with the next character in the alphabet. But other options might be added later on.

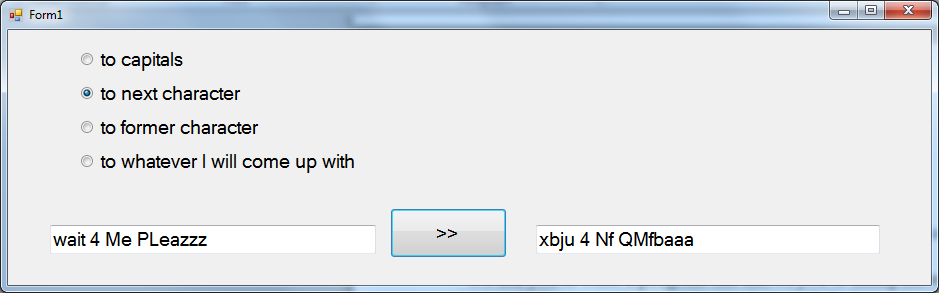
Although you can write the program without delegates we require from you to make use of delegates.

STEP 1:

First create a new windows application with some radiobuttons, 2 textboxes and a button. In one of the textboxes the user can enter a string. Clicking the button manipulates the string in a certain way and displays the result in the other textbox.  
The app looks like:



In the above picture the radiobutton with text "to next character" is checked. Now, if the user clicks the button, in the textbox on the right side appears the answer (as you notice, every alpha-character is replaced by the next character; the last alpha-character 'z' is replaced by the first alpha-character 'a'):



STEP 2:  
For every string-manipulation-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 )  
- the return value is also of type string (the string holding the answer)

So for the string-manipulations as indicated by the radiobuttons you must write separate methods to do the job for you.

For the last radiobutton: where do you come up with? Make your own choice!

To help you a bit, here is some code you may use for the option "to next character":

private String changeToNextCharacter(String s)

{

char[] chars = s.ToCharArray(); //now chars is an array containing all the characters of string s

for (int i = 0; i < chars.Length; i++)

{

char ch = chars[i];

if ((ch >= 'a' && ch < 'z') || (ch >= 'A' && ch < 'Z'))

{

chars[i]++;

}

else

{

if (ch == 'z') { chars[i] = 'a'; }

if (ch == 'Z') { chars[i] = 'A'; }

}

}

//now add the characters in the array chars one by one to the answer-string

String answer = "";

foreach (char ch in chars) { answer += ch; }

return answer;

}

Add methods for the other kinds of manipulations.

Some remarks:

* It is not possible to change the characters in a string. A trick is to convert a string to an array of characters (with s.ToCharArray() ) . You can change the characters in an array.
* The String-class has many useful methods, like for instance the method ToUpper(). You can find useful information about methods of the String-class in the MSDN-help on the internet (for example : <http://msdn.microsoft.com/en-us/library/ms228599.aspx> )

STEP 4:  
Now we are going to use a delegate. As you already saw each string-manipulation-method has the same signature. So we want a delegate-variable that holds a reference to the right method. If the name of this delegate-variable is stringManipulator, the code for clicking the button will be like:

{

this.textBox2.Text = stringManipulator(this.textBox1.Text);

}  
- First declare a delegate type StringManipulationHandler, with that right signature (one parameter, a string and the return-type is also a string).  
- Next declare a variable stringManipulator of that delegate-type.   
- Add the above code in the button-click.

- Make the radiobuttons do their work: selecting a radiobutton should result in assigning the right actual method to the delegate-variable stringManipulator.

Test if your app works.

## **\* 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 80 and 120.

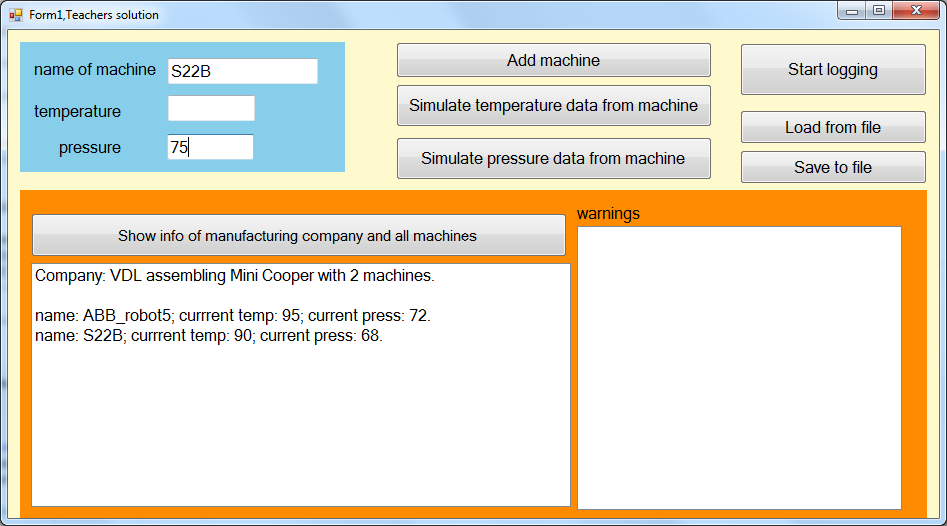
Something similar holds for the pressure. The pressure should be between 60 and 80 for every machine.

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 by a maintenance engineer.   
In reality every machine sends periodically its state to an application. In this assignment the state consists of the current temperature and pressure of the machine, but in the real world there might by more than 100 values describing the current state of a machine. Normally, the maintenance engineer does not want to see all information at all times, but only wants to be warned in case a machine is not functioning correctly.

Imagine you own a company with 40 machines. Every second every machine sends you 150 values. How many bytes do you get in one hour? Do you really want to read all this information?

We are going to write an app to keep track of the state (both temperature and pressure) of the machines. Our application gets the possibility to simulate such machines sending their temperature and pressure.

The GUI of our program looks like this:



In this example above there are 7 buttons.

The name of the company is "VDL assembling Mini Cooper" and there are 2 machines in use (look at the orange panel): The current temperature of the first machine, ABB\_robot5 is 95 degrees and its pressure is 72. By clicking the button in the orange part you will get this information in the listbox on the left side (already implemented).

In the blue region the user can type information about a new machine. Clicking the button "Add machine" should add a new machine with such data to the company.

In the top-left blue panel you see the machine-name S22B and the value 75 for its pressure. Clicking the button "simulate pressure data from machine" should find the machine named S22B (every machine has a unique name) and register its pressure to be 75. In case the pressure was exceeding the limits, a warning should appear in the listbox on the right side, but that is for later.

Something similar for changing the temperature.

The buttons on the right side will be explained later.

On Sharepoint you can find the project to start with. It has some code in it yet. As you can see there is a class Machine to register a Machine-object.

Machine

- name: String

- temperature: int  
- pressure: int

+ 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.

There are also some property's in this class.

The class ManufacturingCompany has a field to store its name and a list to store all its machines. There are some nice members in it, but feel free to add more, if you need.

In the gui-class (Form1) there is already some code.

STEP 1: We need an event.

Add a delegate CriticalStateHandler and an event CriticalState to the Machine-class.

The delegate should describe void-methods with 2 parameters :

(1) Machine sender, the machine that causes the problems   
 (2) String reason, to inform about the kind of problem (for instance: "pressure too high")

We need the event CriticalState to be raised when the state of the machine is critical (if temperature or pressure is out of the correct range).

Make sure you raise the event when the temperature is changed into a critical temperature, and also if the pressure is changed into a critical pressure (Hint: consider the methods SetTemperature and SetPressure in the Machine-class. Use appropriate reasons, like "temperature too high", "temperature too low", etc.).

STEP 2: The gui-class Form1 (handling the event).

Add a method to Form1 that fits to the delegate CriticalStateHandler. This method should write some information in the listbox on the right side about which machine is critical (the name of the machine) and the reason why it is critical.

STEP 3:

When you add a new machine to the company, please do not forget to assign the method from step 2 to the event of that machine.

Now about the 2 buttons in the middle (about changing temperature or pressure of a certain machine). They must simulate that a certain machine sends its current temperature or pressure to our app. They should work as follows: it checks if there is a machine with name as specified in the blue region. If so it will update its current temperature or pressure (as specified in the blue region). If the value is out of range, a warning should be displayed in the listbox on the right, so the maintenance engineer can run to the machine to fix it.

STEP 4: The gui-class Form1: the "start logging"-button.

Sometimes for a certain machine many things go wrong and the maintenance engineer is not able to fix it. Then you want to contact the company where you bought the machine. But how can that company help you if you do not supply them with information about the dis-functioning of the machine? They want extra information. Some company's want this information in a text file, while others want you to insert certain information in their database. In this assignment we will register information in a text-file.

How should this button work? First of all the text on the button is "Start logging".

Suppose the user wants to log information about the dis-functioning of the machine ABB\_robot5. The user types the name of this machine in the blue region. Clicking this button makes sure that from now the dis-functioning of the machine ABB\_robot5 will be logged in a textfile. That means that every time its event CriticalState is raised, a string is added to the log-file. This string contains information about when it happened (DateTime.Now), which machine (its name) and a reason.

Write a method in the Form1-class to append such a string to a text-file (Hint: open the file in the mode Append). Of course, this method fulfils the signature of the delegate CriticalStateHandler.

Clicking the "Start logging"-button should register this method to the event of this particular machine. You should also change the text of the button in "Stop logging".

When the text of the button is "Stop logging" and the user clicks the button, you should stop logging, which is simple: you just remove the "logging"-event handler-method from the event CriticalState for that particular machine.

Test if it works!

STEP 5: Load/Save: see next assignment.

Assignment 1.3: Monitoring machines, Serializable in combination with events  
This assignment is about the "save"- and "load"-buttons.

As you remember from PCS3 it is very easy to serialize an object to a binary file. If an object contains other objects, like your manufacturing company has a list of machines, you should be sure all used classes must be serializable, otherwise serialization does not work.

STEP 1: Saving and loading, first attempt.

Copy from PCS3 the serialization-code to save the variable mc to a binary file. Do not forget to make the ManufacturingCompany-class and the Machine-class serializable.

Test if it works (most probably not: see next step)

STEP 2: Why it does not work?

A machine-object has an event and there might be actual methods attached to this event. Most of the cases such an actual method is a method in a Form-class. In C# the visual components are not serializable. That might be the reason you cannot serialize the variable mc to a binary file.

Before you serialize mc to a binary file, you should first unhook all actual methods. With the notation ". . . += . . ." you can add 1 actual method to an event and with ". . . -= . . ." you remove 1 actual method from an event. You might expect that there is a method to remove all event-handlers from an event, but there isn't such a method in C#. Such a remove-all-method would be nice, since you do not know what actual methods are attached to an event.

There is a trick. An event is a mysterious object, and every object could have the value null. The trick is to assign the value null to the event. This can only be done in the class the event is declared, so in our case in the Machine-class. But the saving is done in the Form1-class.

So we need a public method in the Machine-class to give the event the value null:

public void removeAllCriticalStateHandlers()

{

this.CriticalState = null;

}

Add this method to the Machine-class.

In the "Save to file"-button-click, first remove all event handlers for all machines, and then you can serialize the variable mc to a binary file.

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).