

I don't understand inheritance!

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-12/0742.html>

From: Francois (*francois_at_bettinghouses.com_NOSPAM*)

Date: 12/03/04

Date: Fri, 3 Dec 2004 12:05:26 -0000

Hi,

I am trying to extend a .NET class for overriding some methods.

The class that I try to override is : System.Web.UI.WebControls.CheckBox.

Its full definition is : public class CheckBox : WebControl, IPostBackDataHandler (notice the interface that is implemented)

I want to override 3 methods belonging to that class:

1. protected virtual void OnCheckedChanged(EventArgs e) //Inherited from the class itself
2. IPostBackDataHandler.LoadPostData(string postDataKey, NameValueCollection postCollection) // defined in the IPostBackDataHandler interface
3. IPostBackDataHandler.RaisePostDataChangedEvent() // defined in the IPostBackDataHandler interface

Methods 2 and 3 are not defined by the class CheckBox class itself but as the class implements the interface IPostBackDataHandler then it must implements those 2 methods.

Then a first question arise is: Why those 2 methods do not appear in the class reference of the CheckBox? Should the reference not display all methods that are implemented by the interface? Or can a class implements an interface and make the methods defined in the interface private?

Anyway I try to extends the checkBox class with the following code but it does not compile:

```
using System;
using System.Collections;
using System.Collections.Specialized;
using System.Web.UI;

namespace Bos.CustomControls
{
    public class CheckBoxExtension : System.Web.UI.WebControls.CheckBox
    // i also tried like this but it does not work neither:
```

microsoft.public.dotnet.languages.csharp: I don't understand inheritance!

```
// public class CheckBoxExtension : System.Web.UI.WebControls.CheckBox,
IPostBackDataHandler
{
    public CheckBoxExtension()
    {
    }

    protected override void OnCheckedChanged(EventArgs e)
    {
        //this method compiles
        base.OnCheckedChanged(e);
    }

    override bool IPostBackDataHandler.LoadPostData(string postDataKey,
NameValueCollection postCollection)
    {
        //this does not compile, compiler says: The modifier 'override' is not
valid for this item
        return true;
    }

    override void IPostBackDataHandler.RaisePostDataChangedEvent()
    {
        //this does not compile, compiler says: The modifier 'override' is not
valid for this item
    }
}
}
```

To make the code compile and work, I have to do like the following:

```
public class CheckBoxValue : System.Web.UI.WebControls.CheckBox,
IPostBackDataHandler
{

protected override void OnCheckedChanged(EventArgs e)
{
    // here i can override a method inherited from the base class
}

bool IPostBackDataHandler.LoadPostData(string postDataKey,
NameValueCollection postCollection)
{
    /* here i cannot override this method that the base class implements (as
the base class also implement the interface IPostBackDataHandler. I have a
to re implement the interface like if it was never implemented by the base
class. I do not use the override keyword
*/
}
}
```

I don't understand inheritance!

microsoft.public.dotnet.languages.csharp: I don't understand inheritance!

```
void IPostBackDataHandler.RaisePostDataChangedEvent()
{
    /* here i cannot override this method that the base class implements (as
    the base class also implement the interface IPostBackDataHandler. I have a
    to re implement the interface like if it was never implemented by the base
    class. I do not use the override keyword
    */
}
}
```

For me, this is a problem for 2 reasons:

- First, I cannot call the method `base.LoadPostData` neither `base.RaisePostDataChangedEvent`. What about the case I would like to keep the default behavior of those methods but just adding some extra functionality that I need for my app? When pple override a method, it can very likely be for adding a new functionality, not to replace the functionality.
- Second, as those methods must exist (as the base class implements the interface) why can i not override them? And what I am doing in the second code that compiles seems to me more like bypassing the implementation of the interface of the base class and re–implement it again int the derived class. Like if the interface was never implemented at all at first! This seems a little bit dangerous as people can reimplement interfaces inherited by their base class without knowing it! And without having to use the "override" keyword. Thus it may bring buggy code! Also it just does not make sense to me, if a base class implements this or that interface, how comes I can reimplement it again in a derived class like if it was never implemented at the first place?

I would really appreciate if someone could enlight me on this.

Tx a lot in advance,
Best regards,

Francois.