

Re: Wrapper class

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2006-05/msg04867.html>

- *From:* "Greg Young" <druckdruckREMOVEgoose@xxxxxxxxxxx>
 - *Date:* Thu, 25 May 2006 13:36:00 -0400
-

this seems like encapsulation would be the best bet .. but in more general situations you should probably have a look at the proxy/decorator patterns (both instance and non-instance) which also support polymorphism.

example:

proxy ...

```
public class Foo {
public virtual void Bar() {
//some code
}
}
```

```
public class FooProxy : Foo{
private readonly Foo m_Foo;
public override void Bar() {
//add some code here
m_Foo.Bar();
//or add some code here
}
private FooProxy(Foo _Foo) {
m_Foo = Foo;
}
}
```

the instance proxy being ..

```
public class FooProxy : Foo{
public override void Bar() {
//add some code here
base.Bar();
//or add some code here
}
}
```

These patterns are generally used when you are changing the behaviors of methods (as opposed to defining helper methods as you added)

Re: Wrapper class

Cheers,

Greg Young

MVP – C#

"Marc Gravell" <marc.gravell@xxxxxxxx> wrote in message
news:%23o8Lh5AgGHA.1792@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

In this case B isn't an A; it just behaves like one,

One solution here would be an interface IA (i.e. "I" and your original class name) which defines the key properties etc; B then implements IA (B : IA), and forwards / specialises the methods to the instance of A that it holds.

Marc