Forum:Hack Forums

Thread: Field/Properties

abratar - *September 5, 2017, 11:06 am*

So is there something like a convention or a rule or something on how to and when to use one of these?
I never really understood it. I didn't know what exactly to google and if I found something, there were different answers...
Does it even matter what I use?

Code public int age;
Code public int Age { get; set; }
Code private int age;

```
public int Age
{
    get { return this.age; }
    set { this.age = value; }
}
```

And also whats the difference between tha last two?

severos - *September 5, 2017, 12:20 pm*

Far as I use them, fields are used to store data without any checks or what so ever, properties allow you to do checks on get/set. You can also use them as replacement for read only values that are derived from other data. It's pretty hard to draw a line there, but it's usually about how the team decides to work.

+Haze - *September 5, 2017, 1:01 pm*

1) You should never really make your variable publicly accessible like this
public int age;

2)
public int Age { get; set; }
is the preferred approach because it enforces encapsulation with get and set methods(which are hidden if you use an auto property, which the example is). The reason the second example is used is just to enforce rules of access to a variable. For example, you don't want a user to set an age of 1337 as that's not real. So you'd use the set method to enforce the rules of your logic. Also, in your get method, you can use it to transform the variable before displaying it to the user. For example,

Code public int Area

```
{
    get { return Width * Length; }
}
```

3)
Code private int age;

```
public int Age
{
    get { return this.age; }
```

```
    set { this.age = value; }
}
```
If you're not making use of the get and set routines then just leave it as an auto property(the second one), because that's what the second one boils down to behind the scenes.

Hawk HF - *September 5, 2017, 4:27 pm*

(Sep 5, 2017 01:01 PM)+Haze Wrote: [CITE] 1) You should never really make your variable publicly accessible like this

public int age;

Unless you're working with structs

abratar - *September 5, 2017, 10:03 pm*

So I use the third one if I want to make checks on the Property like not allowing age less than 0 and when I don't have or need any checks, then I use the second one? And the first one shouldn't be used, unless it's really really needed. Did I get it right? Thanks guys

Hawk HF - *September 6, 2017, 5:47 am*

(Sep 5, 2017 10:03 PM)abratar Wrote: [CITE] So I use the third one if I want to make checks on the Property like not allowing age less than 0 and when I don't have or need any checks, then I use the second one? And the first one shouldn't be used, unless it's really really needed. Did I get it right? Thanks guys

Correct ?

th3s3cr3t0n3 - *September 6, 2017, 5:52 am*

Example one is if you just want to store a flat value.

Example two is practically the same with a basic get/set. But obviously it allows you to simply define a get, IE readonly. Also achievable with a keyword in example 1.

Example three is like if you want to 'do something' when you get and set, the field is called a 'backing field' when you use it with a property to store a temporary value. I would use the first or second examples for basic storage IE 'ID' or 'RegistrationDate', these are things that are usually set once and then retrieved but not really changed. I would use example 3 to 'do something' for the data, like 'ProductCode', or 'Price', where you may need to reference other data to get the end result.

I think its a combination of preference and usage to decide on what to use. Don't get hung up on it though, if it works and makes sense to the devs then that's most important.

Imperishable - *September 6, 2017, 10:43 am*

Very well explained by Haze. Make it a point to never expose your field level variables publicly or even internally(keyword). Using properties ensures that validation and optimization can be done easily whenever you want. For example, you can do something like:

Code
```
public int Age
{
    get { return this.age; }
    set {
        if(value==age) return;
        this.age = value;
        }
}
```
The above technique is particularly useful for notifying property changed in viewmodel for a WPF app.

0xDEAD10CC - *September 6, 2017, 2:33 pm*

(Sep 5, 2017 04:27 PM)Hawk HF Wrote: *[CITE]* Unless you're working with structs

structs still aren't really an exception IMHO. Unless you're dealing with Marshalling and you require a very tight packed and aligned datastructure or something... There's no reason not to use properties.