Properties

This lesson discusses properties, how to declare them, initialize them and use them in C#

WE'LL COVER THE FOLLOWING

Introduction

C# properties are *class members* that expose functionality of *methods* using the syntax of *fields*. They simplify the *syntax* of calling traditional `get` and `set` methods (a.k.a. *accessor methods*). Like *methods*, they can be `static` or `instance`.

Let's take a look at `public` `get` and `set` *accessors* first.

Public Get

Getters are used to expose values from *classes*.

```
private name;
public string Name
{
 get { return this.name; }
}
```

Public Set

Setters are used to assign values to *properties*.

```
private name;
public string Name
{
set { this.name = value; }
}
```

Declaring Properties

Properties are defined in the following way:

```
public class PropertiesExample
{
    private bool _isValid;

    public bool IsValid
    {
        get
        {
            return _isValid;  // get returns the field you specify when this property is assi
        }
        set
        {
          //The C# keyword value contains the value assigned to the property.
            _isValid = value; // set assigns the value assigned to the property of the field
        }
    }
}
```

The shorter equivalent of the above code for **getter/setter** *accessors* is the following:

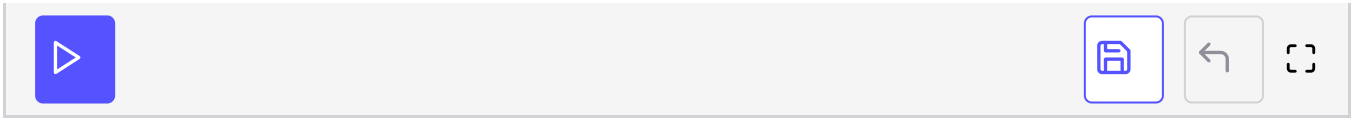
```
public bool IsValid { get; set; }
```

Example

Let's take a look at an example implementing this.

```
using System;

class Person
{
    public int age { get; set; } //get/set accessors for age
    public string name { get; set; }  //get/set accessors for name
}

public class Sample
{
    public static void Main()
    {
        Person person1 = new Person();
```

```
        person1.age = 20;  // ==> equivalent to person1.SetAge("Italian");
        int Age = person1.age; // ==> equivalent to ... = person1.GetAge();

        Console.WriteLine("person1's age is: {0}", Age);

        person1.name = "marissa"; // ==> equivalent to person1.SetName("Italian");

        string Name = person1.name; // ==> equivalent to ... = person1.GetName();

        Console.WriteLine("person1's name is: {0}", Name);
    }

}
```



After a **property** is *defined* it can be used like a **variable**. If you were to write some additional code in the `get` and `set` portions of the **property** it would work like a *method* and allow you to manipulate the data before it is *read* or *written* to the variable.

Properties cannot have any logic in their accessors, for example:

```
// Invalid code
public bool IsValid { get; set { PropertyChanged("IsValid"); } }
```

Properties can, however, have different *access modifiers* for its *accessors*:

```
public bool IsValid { get; private set; }
```

You can set *default* values as well in the following way:

```
public class Name
{
 public string First { get; set; } = "James";
 public string Last { get; set; } = "Smith";
}
```

That was all regarding **properties**. In the next lesson we will discuss *structs* in C#.