# Object-oriented programming (OOP) Lecture 4:Properties

## Properties

* Private fields in a class can’t be accessed outside the class.
* By using properties we can access private fields of a class
* “**A property** is a member that provides a flexible mechanism to **read**, **write**, or **compute** the value of a field in a class.
* Every property contains special methods called “**Accessors**”
* In each property, here are two accessors called *get* and *set.*
* The **get** accessor gets values from the field.
* The **set** accessor sets values to the field.
* There is also a special keyword for a property, named *value*. This represents the value of a field

## Types of Properties

* By using access modifiers, properties can have different access levels. A property can be **public**, **private**, **read only**, **open for read and write**, and **write only**.
* If only the **set** accessor is implemented, this means that the **only write** permission is given.
* If both set and get accessors are implemented, this means that both read and write permissions are open for that property.
* The following are the types of properties:

• **Read only property**.

• **Write only property**.

• **Auto implemented property**.

## Read only Properties:

Using a simple technique we can apply the read only property by just defining the get accessor in the property implementation. As shown in the example below (project name is Readonlyproperties).

using System;

namespace Readonlyproperties

{

public class Car

{

private int \_model = 2012;

private string \_maker = "mercedes";

public int Model\_ID

{

get

{

return \_model;

}

}

public string Maker\_Name

{

get

{

return \_maker;

}

}

}

class Program

{

static void Main(string[] args)

{

Car SportCar = new Car();

Console.WriteLine("Model: {0}", SportCar.Model\_ID);

Console.WriteLine("Maker: {0}", SportCar.Maker\_Name);

}

}

}

In the above example code:

* We created the Car class with two **read-only properties**, Model\_ID and Maker\_Name.
* You can say that each property is read-only because **they only have get accessors**.
* We assign the values for the \_model and \_maker at the time of defining the variables. In this case, the values are 2012 and "mercedes".
* The Main method of the Readonlyproperties class instantiates a new Car object named SportCar. The instantiation of SportCar uses the default constructor of the Car class.
* Since the Model\_ID and Maker\_Name properties of the Car class are read-only, if you insert SportCar.Model\_ID = 2013 into the example, the program will generate an error and not compile, because Model\_ID is read-only; the same goes for Maker\_Name.
* When the Model\_ID and Maker\_Name properties are used in Console.WriteLine, they work fine. This is because these are read operations that only invoke the get accessor of the Model\_ID and Maker\_Name properties.

## Write only Property

A write-only property only has a set accessor. The following example demonstrate this property

using System;

namespace WriteOnlyProperty

{

public class Car

{

private int \_model = -1;

public int model

{

set

{

\_model = value;

}

}

private string \_maker = string.Empty;

public string maker

{

set

{

\_maker = value;

}

}

public void DisplayCustomerData()

{

Console.WriteLine("Model: {0}", \_model);

Console.WriteLine("Maker: {0}", \_maker);

}

}

class Program

{

static void Main(string[] args)

{

Car SportsCar = new Car();

SportsCar.model = 2013;

SportsCar.maker = "lamborghini";

SportsCar.DisplayCustomerData();

}

}

}

In the above example code:

* In the example above we create the Car class with two write-only properties model and maker.
* You can say that each property is write-only because they only have set accessors. Using the set property we have assigned values to model and maker. In this case, the values are 2013 and "lamborghini".
* The Main method of the WriteOnlyProperty class instantiates a new Car object named SportCar. The instantiation of SportCar uses the default constructor of the Car class.
* Since the model and maker properties of the Car class are write-only, if you inserted Console.WriteLine (SportCar.model) into the example, the program will generate an error and not compile, because model is write-only; the same goes for maker.
* When the model and maker properties are used in SportCar.model =2012, they work fine. This is because these are write operations that only invoke the set accessor of the model and maker properties.

## Auto implemented Property

* C# provides a smart way of writing setter and getter methods. If you create a property in C#, you don't have to manually write setter and getter methods for a particular field. Consequently, the common practice in C# is to create properties in a class, rather than creating fields and setter and getter methods for those fields.
* Example below shows how the auto implemented properties are created

using System;

namespace AutoImplementedproperty

{

public class car

{

public int model { get; set; }

public string maker { get; set; }

}

class Program

{

static void Main(string[] args)

{

car SportCar = new car();

SportCar.model = 2014;

SportCar.maker = "ferrari";

Console.WriteLine("Model: {0}", SportCar.model);

Console.WriteLine("Maker: {0}", SportCar.maker);

}

}

}

* When you create a property in **camel case** (first letter of first word is small, then for the remaining words the first letter will be capital), a field with the same name is created **internally**, but in **Pascal case (**the first letter of each word including the first word is capital**)**.