**Accessibility of accessors and Anonymous types**

**Accessibility of Accessors:**

* Properties can be used as if they are public data members, but they are actually special methods called **accessors**.
* This enables data to be accessed easily and help to promote the flexibility and safety of methods.

**Accessors:**

* The block of “set” and “get” is known as “Accessors”.
* It is very essential to restrict the accessibility of property.
* There are two type of accessors i.e. **get accessors**and **set accessors**.
* There are different types of properties based on the “get” and set accessors:
* **Read and Write Properties:** When property contains both get and set methods.
* **Read-Only Properties:** When property contains only get method.
* **Write Only Properties:** When property contains only set method.
* **Auto Implemented Properties:** When there is no additional logic in the property accessors and it introduce in C# 3.0.
* Where, <access\_modifier> can be public, private, protected or internal.
* <return\_type> can be any valid C# type.
* <property\_name> can be user-defined.Properties can be different access modifiers like as public, private, protected, internal.
* Access modifiers define how users of the class can access the property.
* The get and set accessors for the same property may have different access modifiers.

**Accessor Accessibility**

* we can’t use accessor modifiers on an interface or an explicit interface member implementation.
* we can use accessor modifiers only if the property has both set and get accessors.
* If the property is an override modifier, the accessor modifier must match the accessor of the overridden accessor.
* The accessibility level on the accessor must be more restrictive than the accessibility level on the property.

**Anonymous Type**

* Anonymous type, as the name suggests, is a type that doesn't have any name.
* C# allows you to create an object with the *new* keyword without defining its class.
* The [implicitly typed variable- var](https://www.tutorialsteacher.com/csharp/csharp-var-implicit-typed-local-variable) is used to hold the reference of anonymous types.

**Example:**

var myAnonymousType = new { firstProperty = "First",

secondProperty = 2,

thirdProperty = true

};

* In the above example, myAnonymousType is an object of anonymous type created using the new keyword and [object initializer syntax](https://www.tutorialsteacher.com/csharp/csharp-object-initializer)
* It includes three properties of different data types.
* An anonymous type is a temporary data type that is inferred based on the data that you include in an object initializer.
* Properties of anonymous types will be **read-only** properties so you cannot change their values.
* Notice that the compiler applies the appropriate type to each property based on the value expression.
* For example, firstProperty is a string type, secondProperty is an int type and thirdProperty is a bool.
* Internally, the compiler automatically generates the new type for anonymous types.
* You can check the type of an anonymous type as shown below.

static void Main(string[] args)

{

var myAnonymousType = new { firstProperty = "First",

secondProperty = 2,

thirdProperty = true

};

Console.WriteLine(myAnonymousType.GetType().ToString());

}