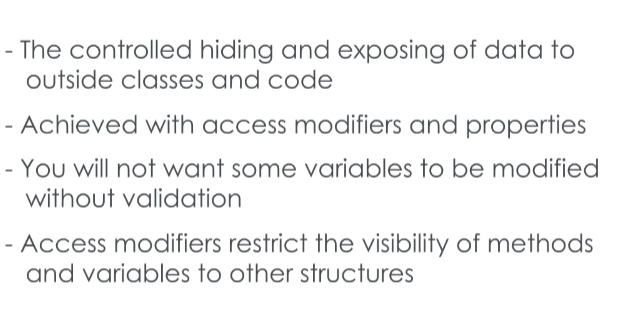
Encapsulation is the hiding and exposing of data within one object to the “outside world” – that is, other objects - by setting the visibility of class members like variables and methods.

The two ways of achieving this in code are with access modifiers and properties.

**Access modifiers** are keywords you can use to directly restrict the ability of other objects to access variables and methods inside another.

C# has four access modifiers public, private, protected, and internal, which is indicated by not specifying another modifier.

Access modifiers control how access to data is shared between a class and other classes in the same assembly, other assemblies, and child or subclasses.

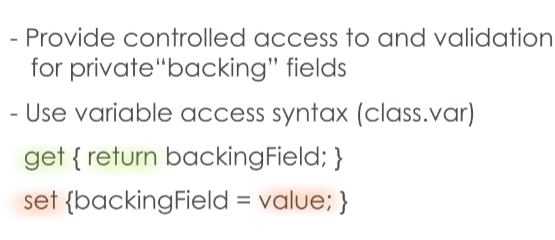
**Public** data can be accessed or modified by any other class in the application.

**Protected** data can be used within the same class, or any subclasses – even if they are in another assembly.

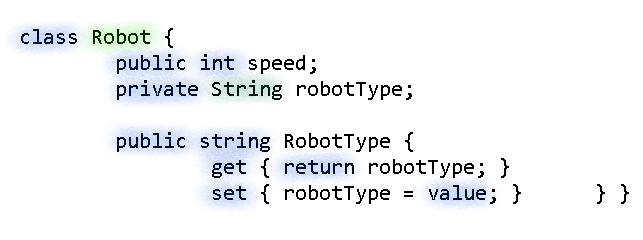
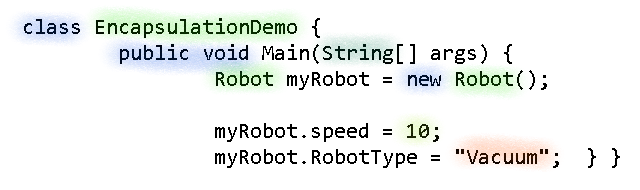
Data with no access modifier specified has **“internal”** access privileges. It's accessible by any class or subclass in the same assembly, but is invisible to code outside of the assembly

**Private** data is only accessible within the same class it is declared. It can't even be used by subclasses.

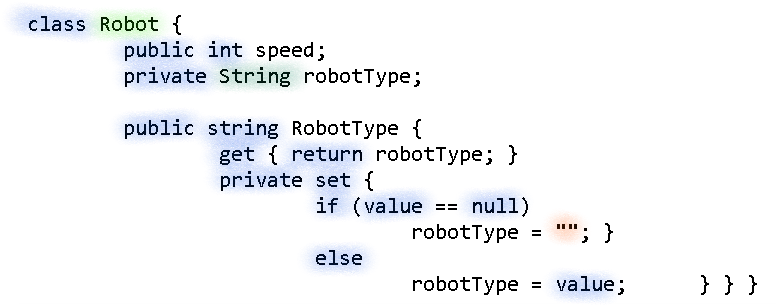
Let’s discuss a cool feature of C#, **Properties**.

**Properties** are commonly used with a private field. These hidden fields, \*referred as a **backing fields\***, can be accessed outside a class with a public Property of the same type.

Now, you can interact with a Property as if you were assigning a value to a variable. Yet internally, they are written like methods.

To return the value of a backing field, Properties have a special mechanism called **“get”** or a “getter.” Similarly, the **“set”**, or “setter”, is used assign a value to backing field using the keyword “value”.

Here we’ve created a property for the private variable robotType, which is now a backing field. In the EncapsulationDemo class, we can access this Property directly.



We can add some special logic to validate the setter for robotType too. This way, we can protect the robotType from being set to a null value.

Another use of Property is to provide data to another class, but preventing changing the data value, and there are two ways to implement this.

You can either declare the “set” with the private access modifier, making “set” only accessible within the class, or you can delete the “set” from the Property altogether.