Use the **static** modifier to declare a static member, which belongs to the type itself rather than to a specific object. The **static** modifier can be used with classes, fields, methods, properties, operators, events, and constructors, but it cannot be used with indexers, destructors, or types other than classes

A [static](https://msdn.microsoft.com/en-us/library/98f28cdx.aspx) class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated. In other words, you cannot use the [new](https://msdn.microsoft.com/en-us/library/51y09td4.aspx) keyword to create a variable of the class type. Because there is no instance variable, you access the members of a static class by using the class name itself. For example, if you have a static class that is named UtilityClass that has a public method named MethodA, you call the method as shown in the following example:

If a local variable is declared with the Static keyword, its lifetime is longer than the execution time of the procedure in which it is declared. If the procedure is inside a module, the static variable survives as long as your application continues running.

The **virtual** keyword is used to modify a method, property, indexer, or event declaration and allow for it to be overridden in a derived class. For example, this method can be overridden by any class that inherits it:

**Using Statement**

Provides a convenient syntax that ensures the correct use of [IDisposable](https://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) objects.

# IDisposable Interface

Provides a mechanism for releasing unmanaged resources.

### Polymorphism means one object behaving as multiple forms.

### Static or Compile Time Polymorphism

In static polymorphism, the decision is made at compile time by using Method Overloading...

Compiler checks the type and number of parameters passed on to the method and decides which method to call at compile time and it will give an error if there are no methods that match the method signature of the method that is called at compile time.

### Dynamic or Runtime Polymorphism

Run-time polymorphism is achieved by method overriding.

* Method overriding allows us to have methods in the base and derived classes with the same name and the same parameters.
* By runtime polymorphism, we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.
* Through the reference variable of a base class, the determination of the method to be called is based on the object being referred to by reference variable.

Compiler would not be aware whether the method is available for overriding the functionality or not. So compiler would not give any error at compile time. At runtime, it will be decided which method to call and if there is no method at runtime, it will give an error.

### Virtual Method

Virtual method is a method whose behavior can be overridden in derived class. Virtual method allows declare a method in base class that can be redefined in each derived class.

When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.

* By default, methods are non-virtual. You cannot override a non-virtual method.
* You cannot use the virtual modifier with the static, abstract, private or override modifiers.
* Virtual properties behave like abstract methods, except for the differences in declaration and invocation syntax.
* It is an error to use the virtual modifier on a static property.
* A virtual inherited property can be overridden in a derived class by including a property declaration that uses the override modifier.

In OOP, when a derived class inherits from a base class, an object of the derived class may be referred to (or cast) as either being the base class type or the derived class type. If there are base class methods overridden by the derived class, the method call behavior is ambiguous.

The important thing to remember about overriding is that the method that is doing the overriding is related to the method in the base class.

Method hiding does not have a relationship between the methods in the base class and derived class. The method in the derived class hides the method in the base class.

### Sealed Keyword

Sealed keyword can be used to stop method overriding in a derived classes.

By default, all methods are sealed, which means you can't override them, so that "sealed" keyword is redundant in this case and compiler will show you an error when you'll try to make sealed already sealed method. But if your method was marked as virtual in a base class, by overriding and marking this method with "sealed" will prevent method overriding in derived classes.

## Summary

1. It is not compulsory to mark the derived/child class function with override keyword while base/parent class contains a virtual method.
2. Virtual methods allow subclasses to provide their own implementation of that method using the override keyword.
3. Virtual methods can't be declared as private.
4. You are not required to declare a method as virtual. But, if you don't, and you derive from the class, and your derived class has a method by the same name and signature, you'll get a warning that you are hiding a parent's method.
5. A virtual property or method has an implementation in the base class, and can be overridden in the derived classes.
6. We will get a warning if we won't use Virtual/New keyword.
7. Instead of Virtual, we can use New keyword.