1. **Introduction**

Graphical user interface, text, email

Description automatically generated

A Namespace can contain another namespace too.

1. **Reading and Writing to Console**

Graphical user interface, text, application

Description automatically generated

1. **Built in types**

Graphical user interface, application

Description automatically generated

1. **String Type**
2. **Inheritance**

Graphical user interface, application

Description automatically generated

Timeline

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application, website

Description automatically generated Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. **Polymorphism**

Graphical user interface, text, application, email

Description automatically generated

public class BaseClass

{

public void WriteNum()

{

Console.WriteLine(12);

}

public virtual void WriteStr()

{

Console.WriteLine("abc");

}

}

public class DerivedClass : BaseClass

{

public new void WriteNum()

{

Console.WriteLine(42);

}

public override void WriteStr()

{

Console.WriteLine("xyz");

}

}

/\* ... \*/

BaseClass isReallyBase = new BaseClass();

BaseClass isReallyDerived = new DerivedClass();

DerivedClass isClearlyDerived = new DerivedClass();

isReallyBase.WriteNum(); // writes 12

isReallyBase.WriteStr(); // writes abc

isReallyDerived.WriteNum(); // writes 12

isReallyDerived.WriteStr(); // writes xyz

isClearlyDerived.WriteNum(); // writes 42

isClearlyDerived.writeStr(); // writes xyz

Overriding is the classic OO way in which a derived class can have more specific behaviour than a base class (in some languages you've no choice but to do so). When a virtual method is called on an object, then the most derived version of the method is called. Hence even though we are dealing with isReallyDerived as a BaseClass then functionality defined in DerivedClass is used.

Hiding means that we have a completely different method. When we call WriteNum() on isReallyDerived then there's no way of knowing that there is a different WriteNum() on DerivedClass so it isn't called. It can only be called when we are dealing with the object **as** a DerivedClass.

Most of the time hiding is bad. Generally, either you should have a method as virtual if its likely to be changed in a derived class, and override it in the derived class. There are however two things it is useful for:

1. Forward compatibility. If DerivedClass had a DoStuff() method, and then later on BaseClass was changed to add a DoStuff() method, (remember that they may be written by different people and exist in different assemblies) then a ban on member hiding would have suddenly made DerivedClass buggy without it changing. Also, if the new DoStuff() on BaseClass was virtual, then automatically making that on DerivedClass an override of it could lead to the pre-existing method being called when it shouldn't. Hence it's good that hiding is the default (we use new to make it clear we definitely want to hide, but leaving it out hides and emits a warning on compilation).
2. Poor-man's covariance. Consider a Clone() method on BaseClass that returns a new BaseClass that's a copy of that created. In the override on DerivedClass this will create a DerivedClass but return it as a BaseClass, which isn't as useful. What we could do is to have a virtual protected CreateClone() that is overridden. In BaseClass we have a Clone() that returns the result of this - and all is well - in DerivedClass we hide this with a new Clone() that returns a DerivedClass. Calling Clone() on BaseClass will always return a BaseClass reference, which will be a BaseClass value or a DerivedClass value as appropriate. Calling Clone() on DerivedClass will return a DerivedClass value, which is what we'd want in that context. There are other variants of this principle, however it should be noted that they are all pretty rare.

An important thing to note with the second case, is that we've used hiding precisely to **remove** surprises to the calling code, as the person using DerivedClass might reasonably expect its Clone() to return a DerivedClass. The results of any of the ways it could be called are kept consistent with each other. Most cases of hiding risk introducing surprises, which is why they are generally frowned upon. This one is justified precisely because it solves the very problem that hiding often introduces.

In all, hiding is sometimes necessary, infrequently useful, but generally bad, so be very wary of it.

1. **Method Hiding vs Method overriding**

Graphical user interface, application

Description automatically generated

1. **Method overloading**

Graphical user interface, text, application

Description automatically generated