**Polymorphism**

1. Polymorphism
2. Static or compile time polymorphism
   * Method overloading
3. Dynamic or runtime polymorphism
   * Method overriding
4. Virtual keyword and virtual method
5. Difference between method overriding and method hiding
6. Sealed method

## **What is Polymorphism?**

Polymorphism means one name many forms. Polymorphism means one object behaving as multiple forms. One function behaves in different forms. In other words, "Many forms of a single object is called Polymorphism."

Polymorphism means many forms (ability to take more than one form). In Polymorphism poly means “multiple” and morph means “forms” so polymorphism means many forms.

### **Real World Example of Polymorphism**

#### Example 1

* A Teacher behaves with student.
* A Teacher behaves with his/her seniors.

Here teacher is an object but the attitude is different in different situations.

#### Example 2

* Person behaves as a SON in house, at the same time that person behaves like an EMPLOYEE in the office.

#### Example 3

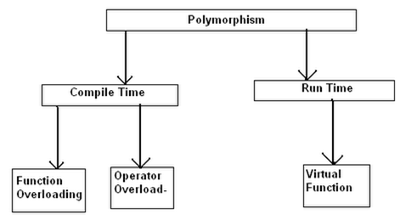
Your mobile phone, one name but many forms:

* As phone
* As camera
* As mp3 player
* As radio

With polymorphism, the same method or property can perform different actions depending on the run-time type of the instance that invokes it.

There are two types of polymorphism:

1. Static or compile time polymorphism
2. Dynamic or runtime polymorphism



### **Static or Compile Time Polymorphism**

In static polymorphism, the decision is made at compile time.

* Which method is to be called is decided at compile-time only.
* Method overloading is an example of this.
* Compile time polymorphism is method overloading, where the compiler knows which overloaded method it is going to call.

Method overloading is a concept where a class can have more than one method with the same name and different parameters.

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.

#### Example

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace MethodOverloadingByManishAgrahari

{

class Program

{

public class TestOverloading

{

public void Add(string a1, string a2)

{

Console.WriteLine("Adding Two String :" + a1 + a2);

}

public void Add(int a1, int a2)

{

Console.WriteLine("Adding Two Integer :" + a1 + a2);

}

}

static void Main(string[] args)

{

TestOverloading obj = new TestOverloading();

obj.Add("Manish " , "Agrahari");

obj.Add(5, 10);

Console.ReadLine();

}

}

}



### **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.

See the following example:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public virtual void Show()

{

Console.WriteLine("Show From Base Class.");

}

}

public class Derived : Base

{

public override void Show()

{

Console.WriteLine("Show From Derived Class.");

}

}

static void Main(string[] args)

{

Base objBase;

objBase = new Base();

objBase.Show();*// Output ----> Show From Base Class.*

objBase = new Derived();

objBase.Show();*//Output--> Show From Derived Class.*

Console.ReadLine();

}

}

}

Compiler demands virtual Show() method and it compiles successfully. The right version of Show() method cannot be determined until run-time since only at that time Base objBase is initialized as Derived.

### **Virtual Keyword**

According to MSDN, “The virtual keyword is used to modify a method, property, indexer or event declaration, and allow it to be overridden in a derived class.”

### **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.

Virtual method solves the following problem:

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.

In C#, polymorphism is explicit - you must have a virtual (or abstract) modifier on the base class method (member) and an override on the derived class method, which you probably already know.

If you don't put a modifier on a base class method, polymorphism can't ever happen. If you then add a non-modified method to the derived class with the same signature as the non-modified base class method, the compiler will generate a Warning message.

See the following example:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public void Show()

{

Console.WriteLine("Show From Base Class.");

}

}

public class Derived : Base

{

*//Following line will Give an Warning*

*/\**

*'PolymorphismByManishAgrahari.Program.Derived.Show()'*

*hides inherited member*

*'PolymorphismByManishAgrahari.Program.Base.Show()'.*

*Use the new keyword if hiding was intended.*

*\*/*

public void Show()

{

Console.WriteLine("Show From Derived Class.");

}

}

static void Main(string[] args)

{

Base objBase = new Base();

objBase.Show();*// Output ----> Show From Base Class.*

Derived objDerived = new Derived();

objDerived.Show();*//Output--> Show From Derived Class.*

Base objBaseRefToDerived = new Derived();

objBaseRefToDerived.Show();*//Output--> Show From Base Class.*

Console.ReadLine();

}

}

}

It means that you are hiding (re-defining) the base class method.

In other languages, take Java for instance, you have what is called "implicit" polymorphism where just putting the method in the derived class with the same signature as a base class method will enable polymorphism.

In Java, all methods of a class are virtual by default unless the developer decides to use the final keyword thus preventing subclasses from overriding that method. In contrast, C# adopts the strategy used by C++ where the developer has to use the virtual keyword for subclasses to override the method. Thus all methods in C# are non virtual by default.

The C# approach is more explicit for the purpose of making the code safer in versioning scenarios, i.e., you build your code based on a 3rd party library and use meaningful, but common, method names. The 3rd party library upgrades, using the same common method name. With implicit polymorphism the code would break, but with C#, you would receive a compiler warning so you can double check to see if polymorphism was something you wanted to do.

### **Difference between Method Overriding and Method Hiding**

Method overriding allows a subclass to provide a specific implementation of a method that is already provided by base class. The implementation in the subclass overrides (replaces) the implementation in the base class.

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.

When a virtual method is called on a reference, the actual type of the object to which the reference refers is used to determine which method implementation should be used. When a method of a base class is overridden in a derived class (subclass), the version defined in the derived class is used. This is so even should the calling application be unaware that the object is an instance of the derived class.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public virtual void Show()

{

Console.WriteLine("Show From Base Class.");

}

}

public class Derived : Base

{

*//the keyword "override" change the base class method.*

public override void Show()

{

Console.WriteLine("Show From Derived Class.");

}

}

static void Main(string[] args)

{

Base objBaseRefToDerived = new Derived();

objBaseRefToDerived .Show();*//Output--> Show From Derived Class.*

Console.ReadLine();

}

}

}

Output--> Show From Derived 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.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public virtual void Show()

{

Console.WriteLine("Show From Base Class.");

}

}

public class Derived : Base

{

public new void Show()

{

Console.WriteLine("Show From Derived Class.");

}

}

static void Main(string[] args)

{

Base objBaseRefToDerived = new Derived();

objBaseRefToDerived .Show();*//Output--> Show From Base Class.*

Console.ReadLine();

}

}

}

Output is:? Show From Base Class.

In the preceding example, Derived.Show will be called; because, it overrides Base.Show.

The C# language specification states that "You cannot override a non-virtual method." See the following example:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public void Show()

{

Console.WriteLine("This is Base Class.");

}

}

public class Derived : Base

{

*//Following Line will give error.*

*/\**

*Error:- 'PolymorphismByManishAgrahari.Program.Derived.Show()'*

*cannot override inherited member 'PolymorphismByManishAgrahari.Program.Base.Show()'*

*\* because it is not marked virtual, abstract, or override*

*\*/*

public override void Show()

{

Console.WriteLine("This is Derived Class.");

}

}

static void Main(string[] args)

{

Base objBase = new Base();

objBase.Show();*// Output ----> This is Base Class.*

Derived objDerived = new Derived();

objDerived.Show();*//Output--> This is Derived Class.*

Base objBaseRefToDerived = new Derived();

objBaseRefToDerived.Show();*//Output--> This is Base Class.*

Console.ReadLine();

}

}

}

Error: 'PolymorphismByManishAgrahari.Program.Derived.Show()' cannot override inherited member 'PolymorphismByManishAgrahari.Program.Base.Show()' because it is not marked virtual, abstract, or override.

### **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 sealedmethod. 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.

See the following example:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public sealed void Show()*//This Line will give an error - "cannot*

{ *//be sealed because it is not an override"*

Console.WriteLine("This is Base Class.");

}

}

public class Derived : Base

{

public void Show()

{

Console.WriteLine("This is Derived Class.");

}

}

static void Main(string[] args)

{

Base objBaseReference = new Derived();

objBaseReference.Show();*// Output ---------> This is Base Class.*

Console.ReadLine();

}

}

}

Error: 'PolymorphismByManishAgrahari.Program.Base.Show()' cannot be sealed because it is not an override.

To remove error from the above program, use the following:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

namespace PolymorphismByManishAgrahari

{

class Program

{

public class Base

{

public virtual void Show()

{

Console.WriteLine("This is Base Class.");

}

}

public class Derived : Base

{

public override sealed void Show()

{

Console.WriteLine("This is Derived Class.");

}

}

static void Main(string[] args)

{

Base objBaseReference = new Derived();

objBaseReference.Show();*// Output ---> This is Derived Class.*

Console.ReadLine();

}

}

}

Output ---> This is Derived Class.

## **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.