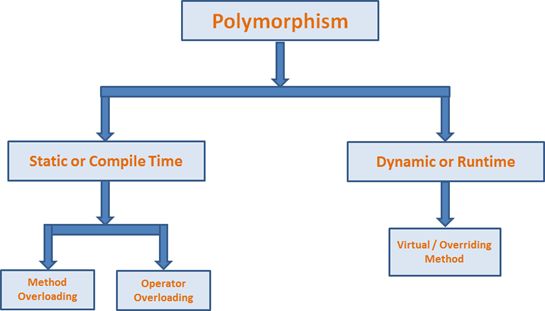
Polymorphism in C#

Polymorphism is a Greek word, meaning "one name many forms". In other words, one object has many forms or has one name with multiple functionalities. "Poly" means many and "morph" means forms. Polymorphism provides the ability to a class to have multiple implementations with the same name. It is one of the core principles of Object Oriented Programming after encapsulation and inheritance. In this article, you'll learn what polymorphism is, how it works, and how to implement polymorphism in C#.

Types of Polymorphism

There are two types of polymorphism in C#:

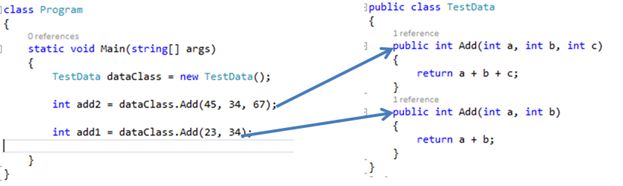
* Static / Compile Time Polymorphism.
* Dynamic / Runtime Polymorphism.



Static or Compile Time Polymorphism

It is also known as **Early Binding**. Method overloading is an example of Static Polymorphism. In overloading, the method / function has a same name but different signatures. It is also known as Compile Time Polymorphism because the decision of which method is to be called is made at compile time. Overloading is the concept in which method names are the same with a different set of parameters.  
  
Here C# compiler checks the number of parameters passed and the type of parameter and make the decision of which method to call and it throw an error if no matching method is found.  
  
In the following example, a class has two methods with the same name "Add" but with different input parameters (the first method has three parameters and the second method has two parameters).

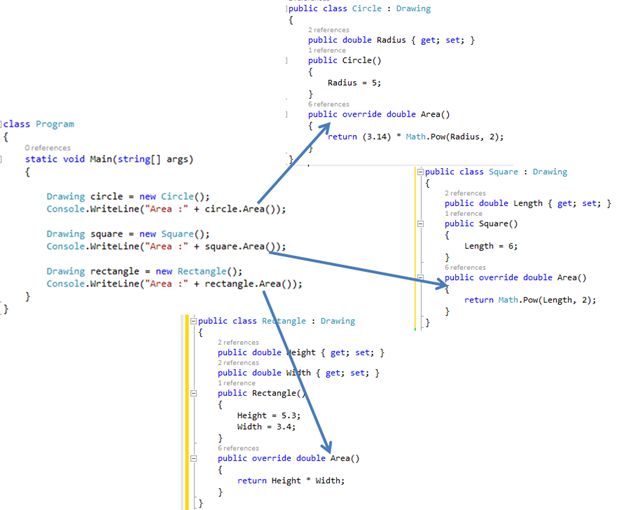
1. **public** **class** TestData
2. {
3. **public** **int** Add(**int** a, **int** b, **int** c)
4. {
5. **return** a + b + c;
6. }
7. **public** **int** Add(**int** a, **int** b)
8. {
9. **return** a + b;
10. }
11. }
12. **class** Program
13. {
14. **static** **void** Main(**string**[] args)
15. {
16. TestData dataClass = **new** TestData();
17. **int** add2 = dataClass.Add(45, 34, 67);
18. **int** add1 = dataClass.Add(23, 34);
19. }
20. }



Dynamic / Runtime Polymorphism

Dynamic / runtime polymorphism is also known as **late binding**. Here, the method name and the method signature (number of parameters and parameter type must be the same and may have a different implementation). Method overriding is an example of dynamic polymorphism.  
  
Method overriding can be done using inheritance. With method overriding it is possible for the base class and derived class to have the same method name and same something. The compiler would not be aware of the method available for overriding the functionality, so the compiler does not throw an error at compile time. The compiler will decide which method to call at runtime and if no method is found then it throws an error.

1. **public** **class** Drawing
2. {
3. **public** **virtual** **double** Area()
4. {
5. **return** 0;
6. }
7. }
9. **public** **class** Circle : Drawing
10. {
11. **public** **double** Radius { **get**; **set**; }
12. **public** Circle()
13. {
14. Radius = 5;
15. }
16. **public** **override** **double** Area()
17. {
18. **return** (3.14) \* Math.Pow(Radius, 2);
19. }
20. }
22. **public** **class** Square : Drawing
23. {
24. **public** **double** Length { **get**; **set**; }
25. **public** Square()
26. {
27. Length = 6;
28. }
29. **public** **override** **double** Area()
30. {
31. **return** Math.Pow(Length, 2);
32. }
33. }
35. **public** **class** Rectangle : Drawing
36. {
37. **public** **double** Height { **get**; **set**; }
38. **public** **double** Width { **get**; **set**; }
39. **public** Rectangle()
40. {
41. Height = 5.3;
42. Width = 3.4;
43. }
44. **public** **override** **double** Area()
45. {
46. **return** Height \* Width;
47. }
48. }
50. **class** Program
51. {
52. **static** **void** Main(**string**[] args)
53. {
55. Drawing circle = **new** Circle();
56. Console.WriteLine("Area :" + circle.Area());
58. Drawing square = **new** Square();
59. Console.WriteLine("Area :" + square.Area());
61. Drawing rectangle = **new** Rectangle();
62. Console.WriteLine("Area :" + rectangle.Area());
63. }
64. }

  
  
The compiler requires an Area() method and it compiles successfully but the right version of the Area() method is not being determined at compile time but determined at runtime. Finally the overriding methods must have the same name and signature (number of parameters and type), as the virtual or abstract method defined in the base class method and that it is overriding in the derived class.  
  
A method or function of the base class is available to the child (derived) class without the use of the "overriding" keyword. The compiler hides the function or method of the base class. This concept is known as shadowing or method hiding. You may find the difference between overriding and shadowing [here](http://www.c-sharpcorner.com/UploadFile/ff2f08/overriding-vs-shadowing-in-C-Sharp/).

Preventing Derived class from overriding virtual members

Virtual members remain “virtual” indefinitely. In other words, virtual members remain “virtual” regardless of how many classes have been between virtual members and the class that originally declared it. For example, if class X has the virtual method "A" and the class Y is derived from X and the class Z is derived from Y, class Z inherits the virtual method "A" and override it.

1. **public** **class** X
2. {
3. **public** **virtual** **void** A()
4. {
5. }
6. }
7. **public** **class** Y : X
8. {
9. **public** **override** **void** A()
10. {
11. }
12. }

A derived class is able to stop virtual inheritance by declaring an override member as "sealed".

1. **public** **class** Y : X
2. {
3. **public** **sealed** **override** **void** A()
4. {
5. }
6. }

Accessing Base class virtual member

Using the "base" keyword, the derived class is able to access the method.

1. **public** **class** X
2. {
3. **public** **virtual** **void** A()
4. {
5. }
6. }
7. **public** **class** Y : X
8. {
9. **public** **override** **void** A()
10. {
11. **base**.A();
12. }
13. }

Summary

* The meaning of Polymorphism is one name having multiple forms.
* The following are the two types of Polymorphism:  
  + Static or compile-time polymorphism (for example, method overloading and operator overloading).
  + Dynamic or runtime polymorphism (for example, overriding).
* Method Overriding differs from shadowing.
* Using the "new" keyword, we can hide the base class member.
* We can prevent a derived class from overriding virtual members.
* We can access a base class virtual member from the derived class.