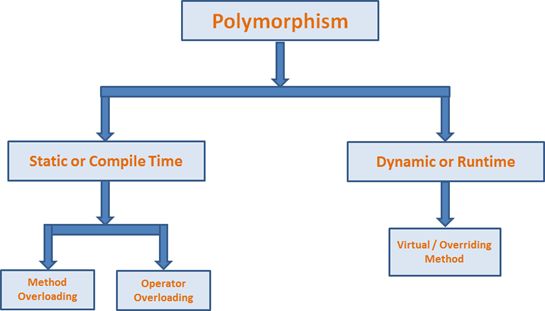
**პოლიმორფიზმი C#-ში**

პოლიმორფიზმი ბერძნული სიტყვაა და ნიშნავს ერთ სახელს ბევრ ფორმაში („პოლი“-ბევრი, „მორფ“-ფორმა). სხვა სიტყვებით რომ ვთქვათ, ერთ ობიექტს აქვს ბევრი ფორმა, ან ერთ სახელს აქვს რამდენიმე ფუნქციონალურობა. პოლიმორფიზმი წარმოადგენს შესაძლებლობას კლასისთვის, რომ მას ჰქონდეს რამდენიმე იმპლემენტაცია ერთი და იმავე სახელით. ის არის ობიექტზე ორიენტირებულ დაპროგრამების ერთ-ერთი მთავარი პრინციპი (დანარჩენი 2 არის ინკაფსულაცია და მემკვიდრეობითობა).

**პოლიმორფიზმის ტიპები**

C#-ში გვაქვს პოლიმორფიზმის 2 ტიპი:

* სტატიკური/პოლიმორფიზმი კომპილაციის დროს.
* დინამიური/ Runtime Polymorphism

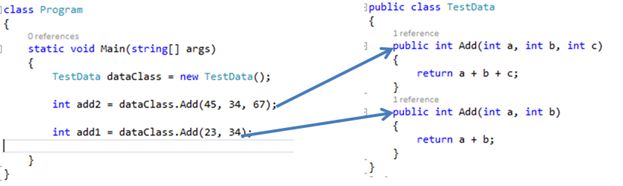


**სტატიკური პოლიმორფიზმი ანუ პოლიმორფიზმი კომპილაციის დროს**

ის ასევე, ცნობილია, როგორც ადრეული აკრძალვა( Early Binding). გადატვირთვა არის სტატიკური პოლიმორფიზმის მაგალითი. ამ დროს მეთოდს/ფუნქციას აქვს ერთნაირი სახლი თუმცა განსხვავებული აღწერა. მას ასევე ეძახიან პოლიმორფიზმს კომპილაციის დროს, რადგან გადაწყვეტილება, თუ რომელი მეთოდი უნდა გამოვიძახოთ მიიღება კომპილაციის დროს. გადატვირთვა არის ისეთი კონცეფცია, რომლის დროსაც მეთოდის სახელები არის ერთნაირი განსხვავებული პარამეტრებით მოცემული.

ჩვენს შემთხვევაში C# ის კომპილერი ამოწმებს გადაცემული პარამეტრების რაოდენობას და ამ პარამეტრების ტიპებს. ამის მიხედვით იღებს გადაწყვეტილებას, თუ რომელი მეთოდი უნდა გამოიძახოს ან ერორს აგდებს არათავსებადი მეთოდის გამოძახების შემთხვევაში.   
 მეტი თვალსაჩინოებისთვის განვიხილოთ მაგალითი.TestData კლასს აქვს 2 ერთნაირი მეთოდი „დამატება“ , მაგრამ თითოეულს გადაეცემა განსხვავებული პარამეტრები (პირველ მეთოდს აქვს 3 პარამეტრი, მეორეს - 2).

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. }



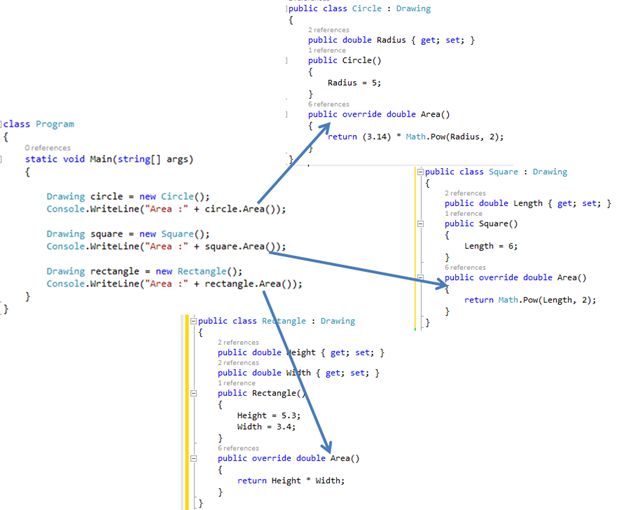
**დინამიური პოლიმორფიზმი ანუ პოლიმორფიზმი RUNTIME ის დროს**

დინამიუი პოლიმორფიზმი ასევე ცნობილია, როგორც გვიანი აკრძალვა( late binding).ამ შემთხვევაში, მეთოდის სახელი და მეთოდის აღწერა (პარამეტრების რაოდენობა და ტიპები )შესაძლოა იყოს ერთნაირი, თუმცა ჰქონდეთ განსხვავებული იმპლემენტაცია. დინამიური პოლიმორფიზმის მაგალითია მეთოდის გადაფარვა.

მეთოდის გადაფარვა შესაძლოა გამოვიწვიოთ მემკვიდრეობითობის გამოყენებით.   
გადაფარვის მეთოდის გამოყენებით შესაძლებელია, რომ საბაზისო და ნაწარმოებ კლასს ჰქონდეთ ერთნაირ სახელიანი მეთოდი და სხვა მსგავსი რამეებიც. კომპილერი არ ამოაგდებს შეცდომას კომპილაციის დროს, ის გადაწყვიტავს რომელი მეთოდი უნდა გამოიძახოს და თუ ვერ იპოვნის მეთოდს, ამ შემთხვევაში დაარტყამს შეცდომას. უკეთ გასაგებად მოვიყვანოთ მაგალითი:

**public** **class** Drawing

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

  
  
კომპილერი მოითხოვს Area () მეთოდს და კომპილირდება წარმატებულად, მაგრამ ამ მეთოდის სწორი ვერსია თავს იჩენს არა კომპილაციის, არამედ გაშვების დროს.

უნდა გავითვალისწინოთ, რომ გადაფარულ მეთოდს უნდა ჰქონდეს იგივე სახელი და იგივე აღწერილობა (პარამეტრების ტიპი და რაოდენობა) რაც ვირტუალურ ან აბსტრაქტულ მეთოდს აქვს საბაზისო კლასში მეთოდად აღებული, სწორედ ეს მეთოდი გადაიფარება წარმოებული კლასის მიერ.

მეთოდი ან ფუნქცია საბაზისო კლასში ხელმისაწვდომია შვილი კლასისთვისo ტერმინი „გადაფარვის“ (overriding) გამოყენების გარეშე. ამ დროს კომპილერი მალავს საბაზისო კლასის ფუნქცუას ან მეთოდს. ეს კონცეფცია ცნობილია როგორც დაჩრდილვა ან მეთოდოს დამალვა (shadowing or method hiding).

**ვირტუალური წევრების გადაფარვის შეზღუდვა მემკვიდრე კლასებისათვის**

ვირტუალური

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.

I hope this helps!