Prototype pattern falls under Creational Pattern of [Gang of Four (GOF) Design Patterns in .Net](http://www.dotnettricks.com/learn/designpatterns/gang-of-four-gof-design-patterns-in-net). It is used to used to create a duplicate object or clone of the current object. It provides an interface for creating parts of a product. In this article, I would like share what is Prototype pattern and how is it work?

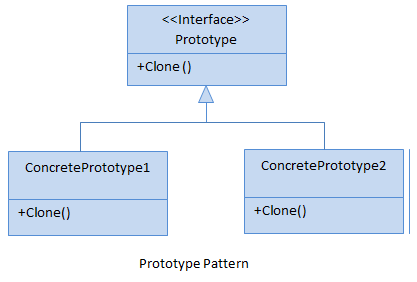
What is Prototype Pattern?

Prototype pattern is used to create a duplicate object or clone of the current object to enhance performance. This pattern is used when creation of object is costly or complex.

**For Example:** An object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

Prototype Pattern - UML Diagram & Implementation

The UML class diagram for the implementation of the Prototype design pattern is given below:



The classes, interfaces and objects in the above UML class diagram are as follows:

1. Prototype

This is an interface which is used for the types of object that can be cloned itself.

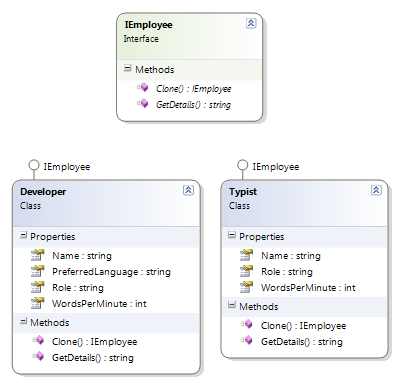
1. ConcretePrototype

This is a class which implements the Prototype interface for cloning itself.

C# - Implementation Code

1. public interface Prototype
2. {
3. Prototype Clone();
4. }
6. public class ConcretePrototypeA : Prototype
7. {
8. public Prototype Clone()
9. {
10. *// Shallow Copy: only top-level objects are duplicated*
11. return (Prototype)MemberwiseClone();
13. *// Deep Copy: all objects are duplicated*
14. *//return (Prototype)this.Clone();*
15. }
16. }
18. public class ConcretePrototypeB : Prototype
19. {
20. public Prototype Clone()
21. {
22. *// Shallow Copy: only top-level objects are duplicated*
23. return (Prototype)MemberwiseClone();
25. *// Deep Copy: all objects are duplicated*
26. *//return (Prototype)this.Clone();*
27. }
28. }

Prototype Pattern - Example



Who is what?

The classes, interfaces and objects in the above class diagram can be identified as follows:

1. **IEmployee** - Prototype interface
2. **Developer & Typist**- Concrete Prototype

C# - Sample Code

1. */// <summary>*
2. */// The 'Prototype' interface*
3. */// </summary>*
4. public interface IEmployee
5. {
6. IEmployee Clone();
7. string GetDetails();
8. }
10. */// <summary>*
11. */// A 'ConcretePrototype' class*
12. */// </summary>*
13. public class Developer : IEmployee
14. {
15. public int WordsPerMinute { get; set; }
16. public string Name { get; set; }
17. public string Role { get; set; }
18. public string PreferredLanguage { get; set; }
20. public IEmployee Clone()
21. {
22. *// Shallow Copy: only top-level objects are duplicated*
23. return (IEmployee)MemberwiseClone();
25. *// Deep Copy: all objects are duplicated*
26. *//return (IEmployee)this.Clone();*
27. }
29. public string GetDetails()
30. {
31. return string.Format("{0} - {1} - {2}", Name, Role, PreferredLanguage);
32. }
33. }
35. */// <summary>*
36. */// A 'ConcretePrototype' class*
37. */// </summary>*
38. public class Typist : IEmployee
39. {
40. public int WordsPerMinute { get; set; }
41. public string Name { get; set; }
42. public string Role { get; set; }
44. public IEmployee Clone()
45. {
46. *// Shallow Copy: only top-level objects are duplicated*
47. return (IEmployee)MemberwiseClone();
49. *// Deep Copy: all objects are duplicated*
50. *//return (IEmployee)this.Clone();*
51. }
53. public string GetDetails()
54. {
55. return string.Format("{0} - {1} - {2}wpm", Name, Role, WordsPerMinute);
56. }
57. }
59. */// <summary>*
60. */// Prototype Pattern Demo*
61. */// </summary>*
63. class Program
64. {
65. static void Main(string[] args)
66. {
67. Developer dev = new Developer();
68. dev.Name = "Rahul";
69. dev.Role = "Team Leader";
70. dev.PreferredLanguage = "C#";
72. Developer devCopy = (Developer)dev.Clone();
73. devCopy.Name = "Arif"; *//Not mention Role and PreferredLanguage, it will copy above*
75. Console.WriteLine(dev.GetDetails());
76. Console.WriteLine(devCopy.GetDetails());
78. Typist typist = new Typist();
79. typist.Name = "Monu";
80. typist.Role = "Typist";
81. typist.WordsPerMinute = 120;
83. Typist typistCopy = (Typist)typist.Clone();
84. typistCopy.Name = "Sahil";
85. typistCopy.WordsPerMinute = 115;*//Not mention Role, it will copy above*
87. Console.WriteLine(typist.GetDetails());
88. Console.WriteLine(typistCopy.GetDetails());
90. Console.ReadKey();
92. }
93. }

Prototype Pattern Demo - Output

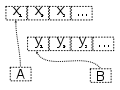
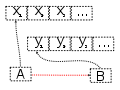
https://dotnettricks.blob.core.windows.net/img/designpatterns/prototype2.png

When to use it?

1. The creation of each object is costly or complex.
2. A limited number of state combinations exist in an object.

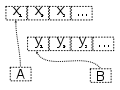
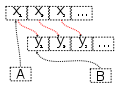
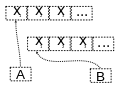
Breadth vs Depth; think in terms of a tree of references with your object as the root node.

Shallow:

The variables A and B refer to different areas of memory, when B is assigned to A the two variables refer to the same area of memory. Later modifications to the contents of either are instantly reflected in the contents of other, as they share contents.

Deep:

The variables A and B refer to different areas of memory, when B is assigned to A the values in the memory area which A points to are copied into the memory area to which B points. Later modifications to the contents of either remain unique to A or B; the contents are not shared.