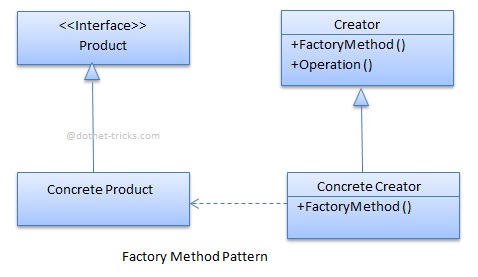
What is Factory Method Pattern?

In Factory pattern, we create the object without exposing the creation logic. In this pattern, an interface is used for creating an object, but let subclass decide which class to instantiate. The creation of object is done when it is required. The Factory method allows a class later instantiation to subclasses.

In short, factory method design pattern abstracts the process of object creation and allows the object to be created at run-time when it is required.

Factory Method Pattern - UML Diagram & Implementation

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



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

1. Product

This is an interface for creating the objects.

1. ConcreteProduct

This is a class which implements the Product interface.

1. Creator

This is an abstract class and declares the factory method, which returns an object of type Product.

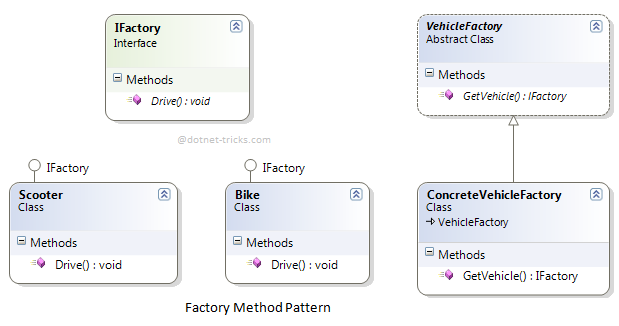
1. ConcreteCreator

This is a class which implements the Creator class and overrides the factory method to return an instance of a ConcreteProduct.

C# - Implementation Code

1. interface Product
2. {
4. }
6. class ConcreteProductA : Product
7. {
8. }
10. class ConcreteProductB : Product
11. {
12. }
14. abstract class Creator
15. {
16. public abstract Product FactoryMethod(string type);
17. }
19. class ConcreteCreator : Creator
20. {
21. public override Product FactoryMethod(string type)
22. {
23. switch (type)
24. {
25. case "A": return new ConcreteProductA();
26. case "B": return new ConcreteProductB();
27. default: throw new ArgumentException("Invalid type", "type");
28. }
29. }
30. }

Factory Method Pattern - Example



Who is what?

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

1. **IFactory** - Interface
2. **Scooter & Bike**- Concreate Product classes
3. **VehicleFactory**- Creator
4. **ConcreteVehicleFactory**- Concreate Creator

C# - Sample Code

1. using System;
2. namespace Factory
3. {
4. */// <summary>*
5. */// The 'Product' interface*
6. */// </summary>*
7. public interface IFactory
8. {
9. void Drive(int miles);
10. }
12. */// <summary>*
13. */// A 'ConcreteProduct' class*
14. */// </summary>*
15. public class Scooter : IFactory
16. {
17. public void Drive(int miles)
18. {
19. Console.WriteLine("Drive the Scooter : " + miles.ToString() + "km");
20. }
21. }
23. */// <summary>*
24. */// A 'ConcreteProduct' class*
25. */// </summary>*
26. public class Bike : IFactory
27. {
28. public void Drive(int miles)
29. {
30. Console.WriteLine("Drive the Bike : " + miles.ToString() + "km");
31. }
32. }
34. */// <summary>*
35. */// The Creator Abstract Class*
36. */// </summary>*
37. public abstract class VehicleFactory
38. {
39. public abstract IFactory GetVehicle(string Vehicle);
41. }
43. */// <summary>*
44. */// A 'ConcreteCreator' class*
45. */// </summary>*
46. public class ConcreteVehicleFactory : VehicleFactory
47. {
48. public override IFactory GetVehicle(string Vehicle)
49. {
50. switch (Vehicle)
51. {
52. case "Scooter":
53. return new Scooter();
54. case "Bike":
55. return new Bike();
56. default:
57. throw new ApplicationException(string.Format("Vehicle '{0}' cannot be created", Vehicle));
58. }
59. }
61. }
63. */// <summary>*
64. */// Factory Pattern Demo*
65. */// </summary>*
66. class Program
67. {
68. static void Main(string[] args)
69. {
70. VehicleFactory factory = new ConcreteVehicleFactory();
72. IFactory scooter = factory.GetVehicle("Scooter");
73. scooter.Drive(10);
75. IFactory bike = factory.GetVehicle("Bike");
76. bike.Drive(20);
78. Console.ReadKey();
80. }
81. }
82. }

Factory Pattern Demo - Output

https://dotnettricks.blob.core.windows.net/img/designpatterns/factory2.png

When to use it?

1. Subclasses figure out what objects should be created.
2. Parent class allows later instantiation to subclasses means the creation of object is done when it is required.
3. The process of objects creation is required to centralize within the application.
4. A class (creator) will not know what classes it will be required to create.