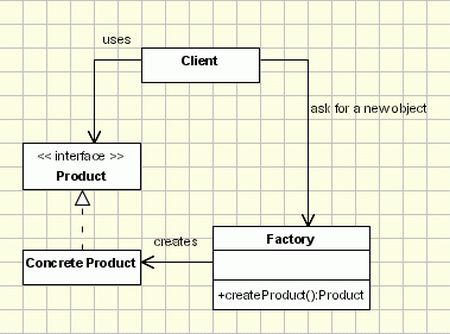
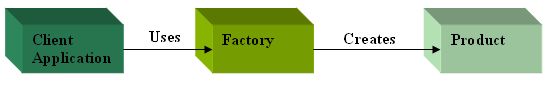
**<https://www.c-sharpcorner.com/UploadFile/97fc7a/factory-pattern-in-net-with-an-example/>**

**Factory Pattern Definition**  
  
GOF says: Define an interface for creating an object, but let subclasses decide which class to instantiate. The Factory Method lets a class defer instantiation to subclasses.  
  
The Factory Pattern is a Creational Pattern that simplifies object creation. You need not worry about the object creation; you just need to supply an appropriate parameter and factory to provide you a product as needed.

* Creating objects without exposing the instantiation logic to the client
* Referring to the newly created objects through a common interface

One of the most traditional designs of a Factory Pattern is depicted below:  
  


A typical image of a Factory pattern is depicted below to understand the fact.  
  
  
  
The images given above simply specifies the association of the Client app with the Factory class and the creation of the object Via the Factory class.  
  
We do have a Factory class named Factory Class. Factory class is responsible for creating an object and its initiation. The Factory class has one method named CreateMobileObject that creates an object on behalf of the parameter being passed by the user (client) and returns the mobile type.  
  
Code snippet for the Factory class:

public static class FactoryClass

{

    public static IMobile CreateMobileObject(MobileType mobileType)

    {

        IMobile objIMobile=null;

        switch (mobileType)

        {

            case MobileType.Samsung:

                objIMobile = new Samsung();

                return objIMobile;

            case MobileType.Apple:

                objIMobile = new Apple();

                return objIMobile;

            case MobileType.Nokia:

                objIMobile = new Nokia();

                return objIMobile;

            default:

                return null;

        }

    }

}  
  
There are a few classes that I've built for the mobile classes that implements the IMobile interface.

public class Samsung : IMobile

{

    public string ModelName()

    {

        return "Samsung Galaxy Grand";

    }

    public string OperatingSystem()

    { return "Samsung Uses Android OS For Galaxy Mobile series "; }

}

public class Apple : IMobile

{

    public string ModelName()

    {

        return "Apple IPhone 5";

    }

    public string OperatingSystem()

    { return "Apple Uses ios OS for Apple Mobiles "; }

}

public class Nokia : IMobile

{

    public string ModelName()

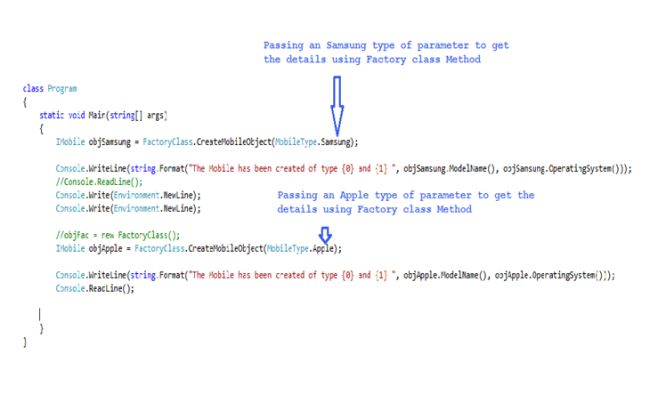
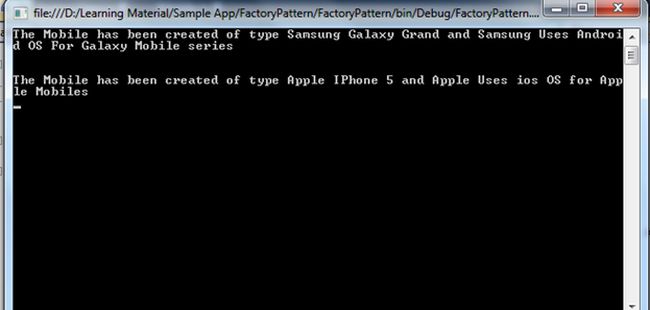
    {

        return "Nokia Lumia 960";

    }

    public string OperatingSystem()

    { return "Nokia Uses Symbion OS for Lumia Mobile series "; }

}  
  
Now proceeding further and passing a parameter to the static method CreateMobileObject() it accepts a parameter of the desired type.  
  
Code snippet for the Main is as follows:  
  
  
Press F5 and you will get results as in the following black window:  
  


**Advantages**

* Easy to implement
* Client application code doesn't need to change drastically
* Moreover, the tight coupling between client and mobile classes is overcome and turned into a coupling between the factory and mobile classes. Hence the client needs not understand the instantiation logic of the products.

**Disadvantages**

* If we add any new product (mobile) then we need a new case statement in the CreateMobileObject method of the Factory class. This violates open/closed design principle.
* We can avoid modifying the Factory class by using sub classing. But sub classing means replacing all the factory class references everywhere throughout the code.
* We have a tight coupling between the Factory class and products.

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.

### 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://dotnettricksweb.blob.core.windows.net/img/designpatterns/factory2.png