<https://www.dotnettricks.com/learn/designpatterns/builder-design-pattern-dotnet>

Builder Design Pattern - C#

Builder Design 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 builds a complex object by using a step by step approach. It provides an interface for creating parts of a product. In this article, I would like to share what is builder pattern and how is it work?

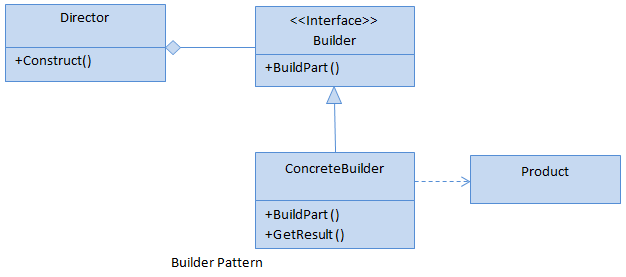
## **What is Builder Pattern?**

**Builder pattern** builds a complex object by using a step by step approach. Builder interface defines the steps to build the final object. This builder is independent of the objects creation process. A class that is known as Director, controls the object creation process.

Moreover, **builder pattern** describes a way to separate an object from its construction. The same construction method can create a different representation of the object.

## **Builder Pattern - UML Diagram & Implementation**

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



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

### **Builder**

This is an interface which is used to define all the steps to create a product

### **ConcreteBuilder**

This is a class which implements the Builder interface to create a complex product.

### **Product**

This is a class which defines the parts of the complex object which are to be generated by the builder pattern.

### **Director**

This is a class which is used to construct an object using the Builder interface.

### **C# - Implementation Code**

public interface IBuilder

{

void BuildPart1();

void BuildPart2();

void BuildPart3();

Product GetProduct();

}

public class ConcreteBuilder : IBuilder

{

private Product \_product = new Product();

public void BuildPart1()

{

\_product.Part1 = "Part 1";

}

public void BuildPart2()

{

\_product.Part2 = "Part 2";

}

public void BuildPart3()

{

\_product.Part3 = "Part 3";

}

public Product GetProduct()

{

return \_product;

}

}

public class Product

{

public string Part1 { get; set; }

public string Part2 { get; set; }

public string Part3 { get; set; }

}

public class Director

{

public void Construct(IBuilder IBuilder)

{

IBuilder.BuildPart1();

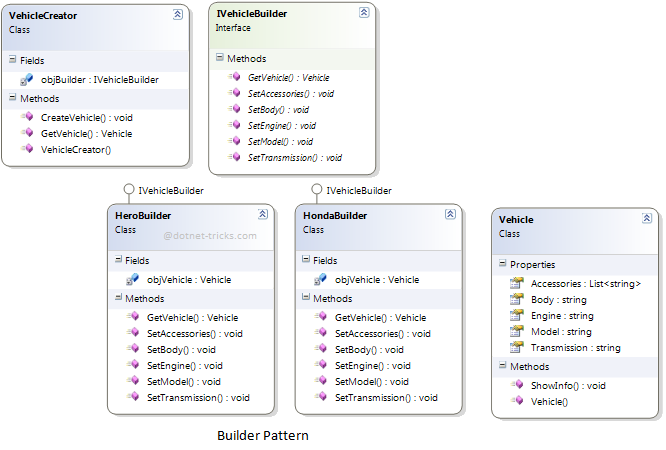
IBuilder.BuildPart2();

IBuilder.BuildPart3();

}

}

## **Builder Pattern - Example**



### **Who is what?**

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

1. **IVehicleBuilder** - Builder interface
2. **HeroBuilder & HondaBuilder**- Concrete Builder
3. **Vehicle**- Product
4. **Vehicle Creator**- Director

### **C# - Sample Code**

*/// <summary>*

*/// The 'Builder' interface*

*/// </summary>*

public interface IVehicleBuilder

{

void SetModel();

void SetEngine();

void SetTransmission();

void SetBody();

void SetAccessories();

Vehicle GetVehicle();

}

*/// <summary>*

*/// The 'ConcreteBuilder1' class*

*/// </summary>*

public class HeroBuilder : IVehicleBuilder

{

Vehicle objVehicle = new Vehicle();

public void SetModel()

{

objVehicle.Model = "Hero";

}

public void SetEngine()

{

objVehicle.Engine = "4 Stroke";

}

public void SetTransmission()

{

objVehicle.Transmission = "120 km/hr";

}

public void SetBody()

{

objVehicle.Body = "Plastic";

}

public void SetAccessories()

{

objVehicle.Accessories.Add("Seat Cover");

objVehicle.Accessories.Add("Rear Mirror");

}

public Vehicle GetVehicle()

{

return objVehicle;

}

}

*/// <summary>*

*/// The 'ConcreteBuilder2' class*

*/// </summary>*

public class HondaBuilder : IVehicleBuilder

{

Vehicle objVehicle = new Vehicle();

public void SetModel()

{

objVehicle.Model = "Honda";

}

public void SetEngine()

{

objVehicle.Engine = "4 Stroke";

}

public void SetTransmission()

{

objVehicle.Transmission = "125 Km/hr";

}

public void SetBody()

{

objVehicle.Body = "Plastic";

}

public void SetAccessories()

{

objVehicle.Accessories.Add("Seat Cover");

objVehicle.Accessories.Add("Rear Mirror");

objVehicle.Accessories.Add("Helmet");

}

public Vehicle GetVehicle()

{

return objVehicle;

}

}

*/// <summary>*

*/// The 'Product' class*

*/// </summary>*

public class Vehicle

{

public string Model { get; set; }

public string Engine { get; set; }

public string Transmission { get; set; }

public string Body { get; set; }

public List<string> Accessories { get; set; }

public Vehicle()

{

Accessories = new List<string>();

}

public void ShowInfo()

{

Console.WriteLine("Model: {0}", Model);

Console.WriteLine("Engine: {0}", Engine);

Console.WriteLine("Body: {0}", Body);

Console.WriteLine("Transmission: {0}", Transmission);

Console.WriteLine("Accessories:");

foreach (var accessory in Accessories)

{

Console.WriteLine("\t{0}", accessory);

}

}

}

*/// <summary>*

*/// The 'Director' class*

*/// </summary>*

public class VehicleCreator

{

private readonly IVehicleBuilder objBuilder;

public VehicleCreator(IVehicleBuilder builder)

{

objBuilder = builder;

}

public void CreateVehicle()

{

objBuilder.SetModel();

objBuilder.SetEngine();

objBuilder.SetBody();

objBuilder.SetTransmission();

objBuilder.SetAccessories();

}

public Vehicle GetVehicle()

{

return objBuilder.GetVehicle();

}

}

*/// <summary>*

*/// Builder Design Pattern Demo*

*/// </summary>*

class Program

{

static void Main(string[] args)

{

var vehicleCreator = new VehicleCreator(new HeroBuilder());

vehicleCreator.CreateVehicle();

var vehicle = vehicleCreator.GetVehicle();

vehicle.ShowInfo();

Console.WriteLine("---------------------------------------------");

vehicleCreator = new VehicleCreator(new HondaBuilder());

vehicleCreator.CreateVehicle();

vehicle = vehicleCreator.GetVehicle();

vehicle.ShowInfo();

Console.ReadKey();

}

}

### **Builder Pattern Demo - Output**

