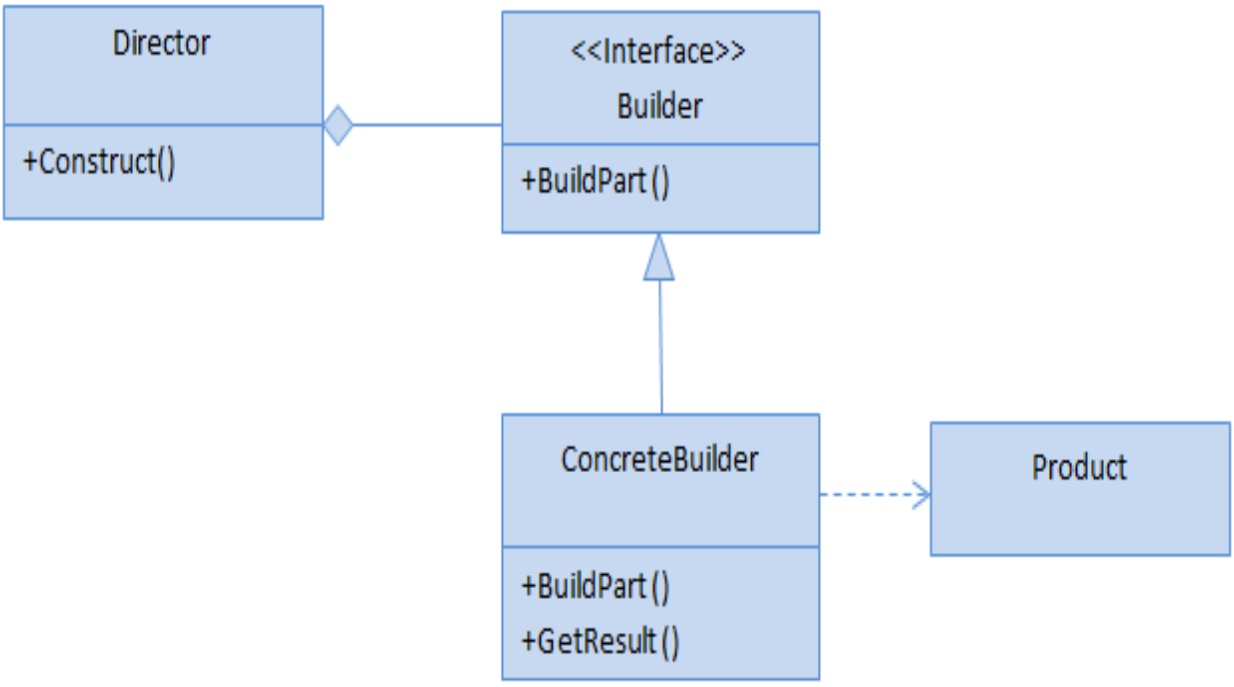
Builder Pattern Implementation Lab Task 1

Introduction & Concept

In this lab task we will learn how to implement the builder pattern using C#.Net

- Builder Pattern separate the construction of a complex object from its representation so that the same construction processes can create different representations.
- The Builder pattern is useful for creating complex objects that have multiple parts.
- The creation process of an object should be independent of these parts; in other words, the construction process does not care how these parts are assembled. In addition, you should be able to use the same construction process to create different representations of the objects.
- The following is the structure of builder pattern as given in the GoF book.



Builder Pattern

Implementation

1. Create a console application in visual studio.
2. Create a folder and name it BuilderImplementation
3. Create ***Product.CS*** file in this folder and write the following code in it:

```
/// <summary>
/// Product -
/// </summary>
public class Computer
{
    public string Type { get; set; }
    public string Processor { get; set; }
    public string Ram { get; set; }
    public string HDD { get; set; }

}
```

The above created computer class is used as a complex product that requires to be build part wise and will have different representations. To create computer objects of different specifications in this lab task, the builder pattern is best suited to these kinds of problem scenarios.

In the next step we are going to create builder pattern interface and concrete builder class that performs the actual role of creating products.

4. Create ***IComputerBuilder.cs*** file in this folder and write the following code in it:

```
/// <summary>
/// Builder Interface : Abstraction
/// </summary>
public interface IComputerBuilder
{
    IComputerBuilder AddType(string type);
    IComputerBuilder AddProcessor(string processor);
```


```
        IComputerBuilder AddRam(string ram);
        IComputerBuilder AddHDD(string hdd);
        Computer GetComputer();
    }
```

In this interface we define the behavior required in the concreted builder to be able to create part wise computer instances.

5. Create ***ComputerBuilder.cs*** file in this folder and write the following code in it:

```
/// <summary>
/// Concreate Builder : Implementation
/// </summary>
public class ComputerBuilder : IComputerBuilder
{
    private readonly Computer computer = new Computer();
    public IComputerBuilder AddHDD(string hdd)
    {
        computer.HDD = hdd;
        return this; //Required for method chaining
    }

    public IComputerBuilder AddType(string type)
    {
        computer.Type = type;
        return this;
    }
    public IComputerBuilder AddProcessor(string processor)
    {
        computer.Processor = processor;
        return this;
    }

    public IComputerBuilder AddRam(string ram)
    {
        computer.Ram = ram;
        return this;
    }
    public Computer GetComputer()
    {
        return computer;
```


```
        }
    }
```

Now we have implemented the builder interface and wrote the behaviors so that we can create build computer objects part wise with different specifications.

Next, we are going to create a director class that is responsible of creating instances with specific steps or sequences.

6. Create ***SystemDirector.cs*** file in this folder and write the following code in it:

```
/// <summary>
/// System Director
/// </summary>
public class SystemDirector
{
    IComputerBuilder Builder { get; set; }
    public SystemDirector(IComputerBuilder builder)
    {
        Builder = builder;
    }

    public IComputerBuilder BuildWithFullConfiguration()
    {
        //Step 1
        Builder.AddHDD("500 GB");
        //Step 2
        Builder.AddType("Desktop");
        //Step 3
        Builder.AddRam("8 GB");
        //Step 4
        Builder.AddProcessor("Core I7");
        return Builder;
    }

    public IComputerBuilder BuildWithMediumConfig()
    {
        //Step 1
        Builder.AddHDD("250 GB").
                AddRam("4 GB").
                AddProcessor("Core I5");
        return Builder;
```


```
        }


        public Computer GetComputer()
        {
            return Builder.GetComputer();
        }
    }
```

The above system director directs in creating two different representation with specific steps for each. The medium configuration has its own steps and the full configurations its own.

7. Create ***ExecuteBuilderDemo.cs*** file and write the following code in it. Here we create a concreate builder and pass it to he director constructor. Then get different representations by calling different methods on director instance.

```
    public class ExecuteBuilderDemo
    {
        public static void Main(string[] args)
        {
            var builder = new ComputerBuilder();
            SystemDirector director = new SystemDirector(builder);
            director.BuildWithFullConfiguration();

            var system = builder.GetComputer();
            Console.WriteLine($"Ram : {system.Ram}, HDD : {system.HD
D}, Processor : {system.Processor}");

            director.BuildWithMediumConfig();
            system = builder.GetComputer();
            Console.WriteLine($"Ram : {system.Ram}, HDD : {system.HD
D}, Processor : {system.Processor}");
            Console.ReadKey();
        }
    }
```

8 In visual studio press f5 to execute and watch the demo.

==========END TASK==========

Builder Pattern Implementation Lab Task 2

Now you have good understanding of builder design pattern, its concept and implementation as in the above task 1 we have a complete demonstration.

It's time to test your learning knowledge by giving you the following problem scenario for the builder pattern implementation.

A vehicle factory manufactures cars and motor bikes of different types. Both the car and motor bike have different integrated components like engine, body, tyre, head lights and many others. In other words, the car and bike are complex objects and require to be built in different steps, also it can have different configurations.

Try you self and if you feel any kind of problem, just visit the above computer example lab task.

Goodbye, wish you all the best and see you in next lab task!