Builder Design Pattern in C#

Back to: Design Patterns in C# With Real-Time Examples

Builder Design Pattern in C# with Examples

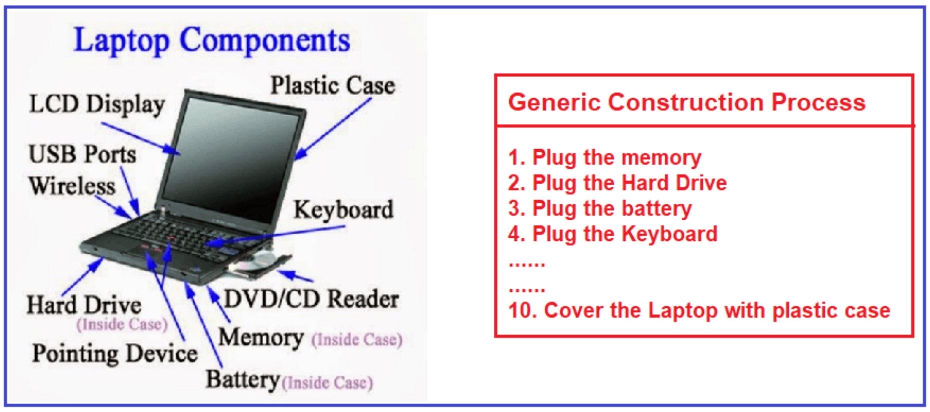
In this article, I am going to discuss the **Builder Design Pattern in C#** with examples. Please read our previous article where we discussed the **Abstract Factory Design Pattern in C#** with examples. The Builder Design Pattern falls under the category of the Creational Design Pattern. As part of this article, we are going to discuss the following pointers.

What is the Builder Design Pattern?

The Builder Design Pattern builds a complex object using many simple objects and using a step-by-step approach. The Process of constructing a complex object should be generic so that the same construction process can be used to create different representations of the same complex object.

So, the Builder Design Pattern is all about separating the construction process from its representation. When the construction process of your object is very complex then only you need to use to Builder Design Pattern. If this is not clear at the moment then don't worry we will try to understand this with an example.

Please have a look at the following diagram. Here, Laptop is a complex object. In order to build a laptop, we have to use many small objects like LCD Display, USB Ports, Wireless, Hard Drive, Pointing Device, Battery, Memory, DVD/CD Reader, Keyboard, Plastic Case, etc. So, we have to assemble these small objects to build laptop complex objects.

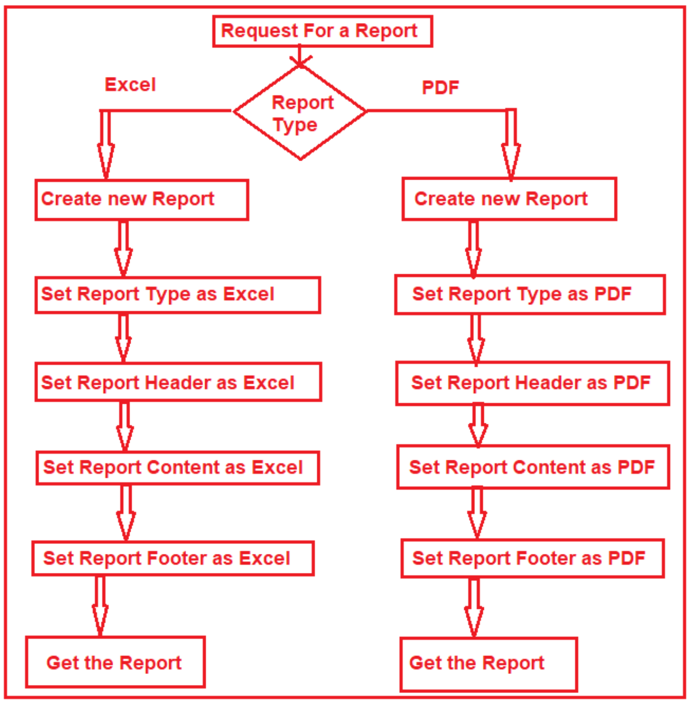


So, to build the complex object laptop we need to define some kind of generic process something like below.
1. Plug the memory
2. Plug the Hard Drive
3. Plug the battery
4. Plug the Keyboard
……
……
10. Cover the Laptop with a plastic case

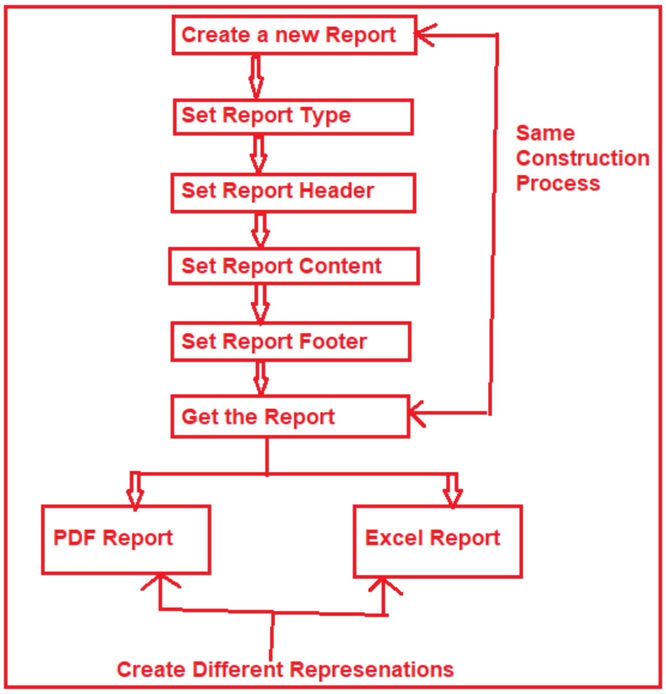
So, using the above process we can create different types of laptops such as a laptop with a 14inch screen or a 17inch screen. Laptop with 4GB RAM or 8GB RAM. So, like this, we can create different kinds of laptops. So, all the laptop creations will follow the same generic process. So, now if you read the definition, then definitely you will understand the definition of Builder Design Pattern.

Understanding the Builder Design Pattern with one real-time example:

Let us understand the builder design pattern with one real-time example. Suppose we want to develop an application for displaying the reports. The reports we need to display either in Excel or in PDF format. That means, we have two types of representation of my reports. In order to understand this better, please have a look at the following diagram.

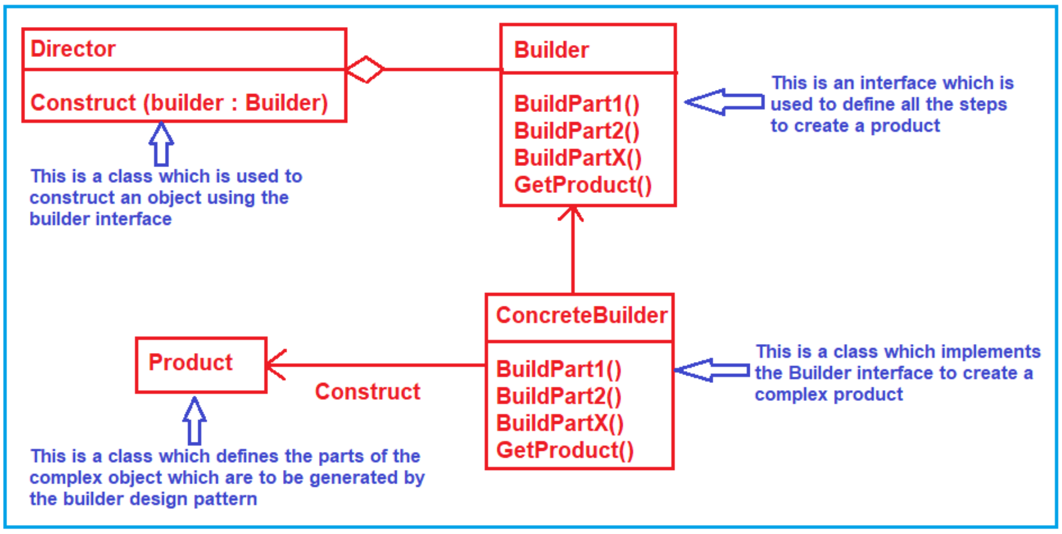


As you can see, in the above image, we are generating the report either in Excel and PDF. Here, the construction process involves several steps such as Create a new report, setting report type, header, content, and footer. If you look at the final output we have one PDF representation and one Excel representation. Please have a look at the following diagram to understand the construction process and its representation.



Understanding the Class Diagram of Builder Design Pattern in C#

Let us understand the class diagram and the different components involved in the Builder Design Pattern. In order to understand this please have a look at the following diagram.


Director
Construct (builder : Builder)
This is a class which is used to construct an object using the builder interface
Builder
BuildPart1()
BuildPart2()
BuildPartX()
GetProduct()
This is an interface which is used to define all the steps to create a product
ConcreteBuilder
BuildPart1()
BuildPart2()
BuildPartX()
GetProduct()
This is a class which implements the Builder interface to create a complex product
Product
Construct
This is a class which defines the parts of the complex object which are to be generated by the builder design pattern

In order to separate the construction process from its representation, the builder design pattern Involve four components. They are as follows.

1. **Builder:** The Builder is an interface that defines all the steps which are used to make the concrete product.
2. **Concrete Builder**: The ConcreteBuilder class implements the Builder interface and provides implementation to all the abstract methods. The Concrete Builder is responsible for constructs and assembles the individual parts of the product by implementing the Builder interface. It also defines and tracks the representation it creates.
3. **Director:** The Director takes those individual processes from the Builder and defines the sequence to build the product.
4. **Product:** The Product is a class and we want to create this product object using the builder design pattern. This class defines different parts that will make the product.

Implementation of Builder Design Pattern in C#:

Let us implement the Report example that we discussed using the Builder Design Pattern in C# step by step.

Step1: Creating the Product

Create a class file with the name **Report.cs** and then copy and paste the following code in it. This is our product class and within this class, we define the attributes (such as ReportHeader, ReportFooter, ReportFooter, and ReportContent) which are common to create a report. We also define one method i.e. DisplayReport to display the report details in the console.

```
using System;
namespace BuilderDesignPattern
{
    public class Report
    {
        public string ReportType { get; set; }
        public string ReportHeader { get; set; }
        public string ReportFooter { get; set; }
        public string ReportContent { get; set; }

        public void DisplayReport()
        {
            Console.WriteLine("Report Type :" + ReportType);
            Console.WriteLine("Header :" + ReportHeader);
            Console.WriteLine("Content :" + ReportContent);
            Console.WriteLine("Footer :" + ReportFooter);
        }
    }
}
```

Once we know the definition of the Product we are building, now we need to create the Builder.

Step2: Creating the Abstract Builder class.

Create a class file with the name **ReportBuilder.cs** and then copy and paste the following in it. This is going to be an abstract class and this class provides the blueprint to create different types of beverages. That means the subclasses are going to implement this **ReportBuilder** abstract class.

```
namespace BuilderDesignPattern
{
    public abstract class ReportBuilder
    {
        protected Report reportObject;

        public abstract void SetReportType();
        public abstract void SetReportHeader();
        public abstract void SetReportContent();
        public abstract void SetReportFooter();

        public void CreateNewReport()
        {
            reportObject = new Report();
        }

        public Report GetReport()
        {
            return reportObject;
        }
    }
}
```

Notice, here we have four abstract methods. So, each subclass of ReportBuilder will need to implement those methods in order to properly build a report. Now, we need to create a few concrete builder classes by implementing the above ReportBuilder interface.

Step3: Creating Concrete Builder classes.

In our example, we are dealing with two types of reports i.e. Excel and PDF. So, we need to create two concrete builder classes by implementing the ReportBuilder abstract class and providing implementation to the ReportBuilder abstract methods.

ExcelReport.cs

Create a class file with the name **ExcelReport.cs** and then copy and paste the following code in it. This ExcelReport class implements the ReportBuilder abstract class which is the blueprint for creating the report objects

```
namespace BuilderDesignPattern
{
    class ExcelReport : ReportBuilder
    {
        public override void SetReportContent()
        {
            reportObject.ReportContent = "Excel Content Section";
        }

        public override void SetReportFooter()
        {
            reportObject.ReportFooter = "Excel Footer";
        }

        public override void SetReportHeader()
        {
            reportObject.ReportHeader = "Excel Header";
        }

        public override void SetReportType()
        {
            reportObject.ReportType = "Excel";
        }
    }
}
```

PDFReport.cs

Create a class file with the name **PDFReport.cs** and then copy and paste the following code in it. This class also implements the ReportBuilder abstract class and provides implementation to all the abstract methods. The following PDFReport class is used to create the Report in PDF format.

```
namespace BuilderDesignPattern
{
    public class PDFReport : ReportBuilder
    {
        public override void SetReportContent()
        {
            reportObject.ReportContent = "PDF Content Section";
        }

        public override void SetReportFooter()
        {
            reportObject.ReportFooter = "PDF Footer";
        }

        public override void SetReportHeader()
        {
            reportObject.ReportHeader = "PDF Header";
        }

        public override void SetReportType()
        {
            reportObject.ReportType = "PDF";
        }
    }
}
```

Once you have the required concrete builder classes, now we need to create the director. The director will execute the required steps to create a particular report.

Step4: Creating the Director

Please create a class file with the name **ReportDirector.cs** and then copy and paste the following code in it. The following class is having one generic method i.e. MakeReport which will take the ReportBuilder instance as an input parameter and then create and return a particular report object.

```
namespace BuilderDesignPattern
{
    public class ReportDirector
    {
        public Report MakeReport(ReportBuilder reportBuilder)
        {
            reportBuilder.CreateNewReport();
            reportBuilder.SetReportType();
            reportBuilder.SetReportHeader();
            reportBuilder.SetReportContent();
            reportBuilder.SetReportFooter();

            return reportBuilder.GetReport();
        }
    }
}
```

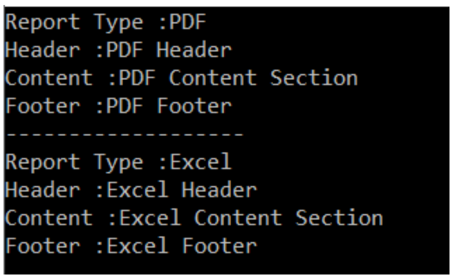
Note: This MakeReport is so generic that it can create and return different types of report objects. Once we have the Director and Concrete Builder, now we can use them in the Main method to create different types of Reports.

Step5: Client code.

Please modify the Main method as shown below. Here, first, we will create an instance of the ReportDirector class and then create an instance of PDFReport class. Once we have ReportDirector instance and PDFReport instance, then we call the MakeReport method on the ReportDirector instance bypassing the PDFReport instance as an argument which will create and return the report in PDF format. The same process is also for Excel report.

```
namespace BuilderDesignPattern
{
    class Program
    {
        static void Main(string[] args)
        {
            // Client Code
            Report report;
            ReportDirector reportDirector = new ReportDirector();

            // Construct and display Reports
            PDFReport pdfReport = new PDFReport();
            report = reportDirector.MakeReport(pdfReport);
            report.DisplayReport();

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

            ExcelReport excelReport = new ExcelReport();
            report = reportDirector.MakeReport(excelReport);
            report.DisplayReport();

            Console.ReadKey();
        }
    }
}
```

Now run the application and you should get the output as shown below.



```
Report Type :PDF
Header :PDF Header
Content :PDF Content Section
Footer :PDF Footer
-------------------
Report Type :Excel
Header :Excel Header
Content :Excel Content Section
Footer :Excel Footer
```

When to use the Builder Design Pattern in real-time applications?

We need to use the Builder Design Pattern in real-time applications in the following scenarios.

1. When you want to make a complex object by specifying only its type and content. The built object is constructed from the details of its construction.
2. When you decouple the process of building a complex object from the parts that make up the object.
3. When you want to isolate the code for construction and representation.

In the next article, I am going to discuss the **Builder Design Pattern Real-time** Example using C#. Here, in this article, I try to explain the **Builder Design Pattern in C#** with Examples. I hope you understood the need and use of the Builder Design Pattern in C#.