Fluent Interface Design Pattern in C#

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

Fluent Interface Design Pattern in C# with Examples

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

1. **What is the Fluent Interface Design Pattern?**
2. **Understanding Method Chaining in C#.**
3. **Implementing Fluent Interface Design Pattern in C#.**
4. **When do we need to use the Fluent Interface Design Pattern?**

Note: The Fluent Interfaces and Method chaining are related to each other. Or we can say that one is a concept and the other one is its implementation. Here in this article, first, we will discuss fluent interfaces and then we will move towards method chaining.

What is the Fluent Interface Design Pattern?

The main objective of the Fluent Interface Design Pattern is that we can apply multiple properties (or methods) to an object by connecting them with dots (.) without having to re-specify the object name each time.

How to Implement Fluent Interface Design Pattern in C#?

Let us understand How to Implement the Fluent Interface Design Pattern in C# with an example. Let say, we have the following Employee class.

```
public class Employee
{
    public string FullName { get; set; }
    public DateTime DateOfBirth { get; set; }
    public string Department { get; set; }
    public string Address { get; set; }
}
```

If we want to consume the above Employee class, then we generally, create an instance of the Employee and set the respective properties as shown below.

```
Employee employee = new Employee();
employee.FullName = "Anurag Mohanty";
employee.DateOfBirth = Convert.ToDateTime("10/10/1992");
employee.Department = "IT";
employee.Address = "Mumbai-India";
```

The Fluent interfaces simplify your object consumption code by making your code more simple, readable, and discoverable. Is not it nice to be set the object properties as shown below?

```
Employee employee = new Employee();

employee.NameOfTheEmployee("Anurag Mohanty")
        .Born("10/10/1992")
        .WorkingOn("IT")
        .StaysAt("Mumbai-India");
```

If we create such kinds of interfaces, then it is like speaking a sentence that would really make the class consumption code more simple and more readable. Now the next thing is how to achieve this. To achieve this, we have something called method chaining.

What is Method Chaining?

Method chaining is a common technique where each method returns an object and all these methods can be chained together to form a single statement. In order to achieve this, first, we need to create a wrapper class around the Employee class as shown below.

```
public class FluentEmployee
{
    private Employee employee = new Employee();

    public FluentEmployee NameOfTheEmployee(string FullName)
    {
        employee.FullName = FullName;
        return this;
    }

    public FluentEmployee Born(string DateOfBirth)
    {
        employee.DateOfBirth = Convert.ToDateTime(DateOfBirth);
        return this;
    }

    public FluentEmployee WorkingOn(string Department)
    {
        employee.Department = Department;
        return this;
    }

    public FluentEmployee StaysAt(string Address)
    {
        employee.Address = Address;
        return this;
    }
}
```

As you can see, here we have created methods for each property. Also, notice the return of the method is set to the FluentEmployee. Now the above fluent interface is going to be consumed by the client. So, with the above FluentEmployee class in place, now the client code should looks as shown below.

```
FluentEmployee obj = new FluentEmployee();

obj.NameOfTheEmployee("Anurag Mohanty")
   .Born("10/10/1992")
   .WorkingOn("IT")
   .StaysAt("Mumbai-India");
```

Example: Fluent Interface Implementation using C#

The following code example shows the Fluent Interface Design Pattern Implementation in C#.

```
using System;

namespace FluentInterfaceDesignPattern
{
    class Program
    {
        static void Main(string[] args)
```

```
        {
            FluentEmployee obj = new FluentEmployee();

            obj.NameOfTheEmployee("Anurag Mohanty")
                    .Born("10/10/1992")
                    .WorkingOn("IT")
                    .StaysAt("Mumbai-India");

            Console.Read();
        }
    }

    public class Employee
    {
        public string FullName { get; set; }
        public DateTime DateOfBirth { get; set; }
        public string Department { get; set; }
        public string Address { get; set; }
    }

    public class FluentEmployee
    {
        private Employee employee = new Employee();

        public FluentEmployee NameOfTheEmployee(string FullName)
        {
            employee.FullName = FullName;
            return this;
        }

        public FluentEmployee Born(string DateOfBirth)
        {
            employee.DateOfBirth = Convert.ToDateTime(DateOfBirth);
            return this;
        }

        public FluentEmployee WorkingOn(string Department)
        {
            employee.Department = Department;
            return this;
        }

        public FluentEmployee StaysAt(string Address)
        {
            employee.Address = Address;
            return this;
        }
    }
}
```

When do we need to use the Fluent Interface Design Pattern in C#?

1. During UNIT testing when the developers are not full-fledged programmers.
2. When you want your code to be readable by non-programmers so that they can understand if the code is satisfied with their business logic or not.
3. If you are a component seller and you want to stand out in the market as compared to the others by making your interface simpler.

I have seen fluent interfaces are used extensively in LINQ Queries: Searching, Sorting, pagination, grouping with a blend of LINQ are some of the real-world usages of the fluent interface in combination with the builder design pattern.

In the next article, I am going to discuss the **Prototype Design Pattern in C#** with Examples. Here, in this article, I try to explain the **Fluent Interface Design Pattern in C#** with Examples. I hope you understood the need and use of the Fluent Interface Design Pattern.