Creational Design Patterns In C#

Bardia Mostafavi Nov 08, 2021 16.6k 0 6

What is a Design Pattern?

In simple words, a Design Pattern is a standard solution for handling a specific situation. Thus you can use it to save your time so do not have to reinvent the wheel, and most important of that you use it because this is the standard and optimum solution and approved by all other real programmers. Just what you need is to check the situation and pick the best and relevant design pattern and adjust it for use in your code. But keep in mind do not use them everywhere, sometimes simple code do the work better and no need for overusing Design Patterns.

Basically, there are three main categories in Design Patterns :

1. *Creational*
2. *Structural*

3. *Behavioral*

Creational Design Patterns are mainly focused on the creation of objects. So for example they can make creation of big object easy and reusable. There are five different patterns that exist in Creational category with different usage :

1. *Abstract Factory*
2. *Factory Method*
3. *Builder*
4. *Prototype*
5. *Singleton*

Abstract Factory

This pattern is used for creating of families of related objects so we have interface or abstract class as factory which have multiple methods that usually return related type.

First, make some related type.

```
1  public interface ITechCompany
2  {
3      void PrintDetail();
4  }
5  public interface ICarCompany
6  {
7      void PrintDetail();
8  }
9  //family #1
10 public class Asus : ITechCompany
11 {
12     public void PrintDetail()
13     {
14         Console.WriteLine("i am detail of asus company");
15     }
16 }
17 public class Dell : ITechCompany
18 {
```

```
19      public void PrintDetail()
20      {
21          Console.WriteLine("i am detail of dell company");
22      }
23  }
24  //family #2
25  public class Benz : ICarCompany
26  {
27      public void PrintDetail()
28      {
29          Console.WriteLine("i am detail of Benz company");
30      }
31  }
32  public class Audi : ICarCompany
33  {
34      public void PrintDetail()
35      {
36          Console.WriteLine("i am detail of Audi company");
37      }
38  }
```

T

```
1   public interface ICompanyFactory
2   {
3       ITechCompany CreateTechCompany();
4       ICarCompany CreateCarCompany();
5   }
6   public class CompanyFactoryA : ICompanyFactory
7   {
8       public ITechCompany CreateTechCompany()
9       {
10          return new Asus();
11      }
12      public ICarCompany CreateCarCompany()
13
```

```
    {
        return new Benz();
    }
}
public class CompanyFactoryB : ICompanyFactory
{
    public ITechCompany CreateTechCompany()
    {
        return new Dell();
    }
    public ICarCompany CreateCarCompany()
    {
        return new Audi();
    }
}
public static class AbstractFactoryExample
{
    public static void Test()
    {
        var factoryA = new CompanyFactoryA();
        var carA = factoryA.CreateCarCompany();
        var techA = factoryA.CreateTechCompany();
        carA.PrintDetail();
        techA.PrintDetail();

        var factoryB = new CompanyFactoryB();
        var carB = factoryB.CreateCarCompany();
        var techB = factoryB.CreateTechCompany();
        carB.PrintDetail();
        techB.PrintDetail();
    }
    //result :
    //i am detail of Benz company
    //i am detail of asus company
    //i am detail of Audi company
```

```
        //i am detail of dell company
    }
```

F

T
f

I

```
public interface ITechCompanyCreator
{
    ITechCompany CreateTechCompany();
}
public class TechCompanyCreatorA : ITechCompanyCreator
{
    public ITechCompany CreateTechCompany()
    {
        return new Asus();
    }
}
public class TechCompanyCreatorB : ITechCompanyCreator
{
    public ITechCompany CreateTechCompany()
    {
        return new Dell();
    }
}
public static class FactoryMethodExample
{
    public static void Test()
    {
        var creatorA = new TechCompanyCreatorA();
        var techA = creatorA.CreateTechCompany();
        techA.PrintDetail();

```

```
        var creatorB = new TechCompanyCreatorB();
        var techB = creatorB.CreateTechCompany();
        techB.PrintDetail();
    }
    //result :
    //i am detail of Asus company
    //i am detail of Dell company
}
```

E

T

```
public interface IBuilder
{
    Contact Build();
}
public class ContactBuilder : IBuilder
{
    private readonly Contact _contact = new Contact();
    public ContactBuilder WithName(string name)
    {
        _contact.Name = name;
        return this;
    }
    public ContactBuilder WithFamily(string family)
    {
        _contact.Family = family;
        return this;
    }
    public ContactBuilder WithAge(int age)
    {
        _contact.Age = age;
        return this;
    }
    public Contact Build()
```

```
        {
            return _contact;
        }
    }
    public class Contact
    {
        public string Name { get; set; }
        public string Family { get; set; }
        public int Age { get; set; }
    }
    public static class BuilderExample
    {
        public static void Test()
        {
            var contact = new ContactBuilder()
                          .WithName("name")
                          .WithFamily("family")
                          .WithAge(10)
                          .Build();
            Console.WriteLine(contact.ToJson());
        }
        //result :
        //{"Name":"name","Family":"family","Age":10}
    }
```

F

T
o

```
    public class Person
    {
        public string Name { get; set; }
        public Family Family { get; set; }
        public int Age { get; set; }
```

```
        //Pattern
        public Person ShallowCopy()
        {
            return (Person) this.MemberwiseClone();
        }
        public Person DeepCopy()
        {
            var clone =  (Person) this.MemberwiseClone();
            clone.Name = new string(Name);
            clone.Family = new Family
            {
                MiddleName = new string(Family.MiddleName),
                LastName = new string(Family.LastName)
            };
            return clone;
        }
    }
    public class Family
    {
        public string MiddleName { get; set; }
        public string LastName { get; set; }
    }

    public static class PrototypeExample
    {

        public static void Test()
        {
            var person = new Person
            {
                Name = "David",
                Family = new Family
                {
                    MiddleName = "Junior",
                    LastName = "Nolan"
```

```
            },
            Age = 20
        };

        var shallowCopied = person.ShallowCopy();
        var deepCopied = person.DeepCopy();

        person.Age = 55;
        person.Name = "rookie";
        person.Family.LastName = "bishop";

        Console.WriteLine(person.ToJson());
        Console.WriteLine(shallowCopied.ToJson());
        Console.WriteLine(deepCopied.ToJson());
    }

    //result:
    //{"Name":"rookie","Family":{"MiddleName":"Junior","LastName":"bi
    //{"Name":"David","Family":{"MiddleName":"Junior","LastName":"bis
    //{"Name":"David","Family":{"MiddleName":"Junior","LastName":"Nol
}
```

Singleton Pattern

With this pattern we ensure that we have one instance of object in runtime with one general global access.

```
public class Singleton
{
    private Singleton()
    {
    }

    private static Singleton _instance;

    public static Singleton GetInstance()
```

```
    {
        if (_instance == null)
            _instance = new Singleton();
        return _instance;
    }
}
public static class SingletonExample
{
    public static void Test()
    {
        var instance1 = Singleton.GetInstance();
        var instance2 = Singleton.GetInstance();

        if (instance1.Equals(instance2))
            Console.WriteLine("Yes");
        else
            Console.WriteLine("No");
    }
    //result:
    //Yes
}
```

C

c

Builder Design Pattern

Singleton Design Pattern Evolution and implementation C#

Design Patterns Part - I

Programming C# for Beginners

Download Now!

Bardia Mostafavi

NA 59k

0

Type your comment here and press Enter Key (Minimum 10 characters)