# الگوی Abstract Factory در #C

## تعریف الگوی Abstract Factory

الگوی **Abstract Factory** یک الگوی طراحی **خلاقی** (Creational) پیشرفته است که امکان ایجاد خانواده‌ای از اشیا مرتبط را بدون مشخص کردن کلاس‌های concrete آنها فراهم می‌کند. این الگو یک سطح بالاتر از انتزاع نسبت به Factory Method ارائه می‌دهد.

## مشکل اصلی که Abstract Factory حل می‌کند

1. **ایجاد خانواده‌ای از اشیا مرتبط** بدون وابستگی به کلاس‌های concrete آنها
2. **حفظ سازگاری** بین اشیایی که باید با هم کار کنند
3. **جدا کردن** منطق ساخت اشیا از استفاده از آنها

## ساختار پایه Abstract Factory

// 1. رابط‌های محصولات (Abstract Products)

public interface IChair {

string SitOn();

}

public interface ISofa {

string LieOn();

}

public interface ICoffeeTable {

string PutCup();

}

// 2. پیاده‌سازی‌های مختلف محصولات (Concrete Products)

public class ModernChair : IChair {

public string SitOn() => "Sitting on a modern chair";

}

public class VictorianChair : IChair {

public string SitOn() => "Sitting on a victorian chair";

}

public class ModernSofa : ISofa {

public string LieOn() => "Lying on a modern sofa";

}

public class VictorianSofa : ISofa {

public string LieOn() => "Lying on a victorian sofa";

}

// 3. رابط Abstract Factory

public interface IFurnitureFactory {

IChair CreateChair();

ISofa CreateSofa();

ICoffeeTable CreateCoffeeTable();

}

// 4. پیاده‌سازی‌های Concrete Factory

public class ModernFurnitureFactory : IFurnitureFactory {

public IChair CreateChair() => new ModernChair();

public ISofa CreateSofa() => new ModernSofa();

public ICoffeeTable CreateCoffeeTable() => new ModernCoffeeTable();

}

public class VictorianFurnitureFactory : IFurnitureFactory {

public IChair CreateChair() => new VictorianChair();

public ISofa CreateSofa() => new VictorianSofa();

public ICoffeeTable CreateCoffeeTable() => new VictorianCoffeeTable();

}

## نحوه استفاده از الگو

class Program {

static void Main(string[] args) {

Console.WriteLine("Creating modern furniture:");

ClientCode(new ModernFurnitureFactory());

Console.WriteLine("\nCreating victorian furniture:");

ClientCode(new VictorianFurnitureFactory());

}

public static void ClientCode(IFurnitureFactory factory) {

var chair = factory.CreateChair();

var sofa = factory.CreateSofa();

Console.WriteLine(chair.SitOn());

Console.WriteLine(sofa.LieOn());

}

}

## مزایای Abstract Factory

1. **سازگاری محصولات**: تضمین می‌کند محصولات ایجاد شده با هم سازگار هستند
2. **جدا کردن کد کلاینت**: کد کلاینت فقط با رابط‌های abstract کار می‌کند
3. **تعویض آسان خانواده محصولات**: فقط با تغییر factory می‌توان کل خانواده محصولات را تغییر داد
4. **تست‌پذیری**: امکان استفاده از mock objects در تست‌ها
5. **پیروی از اصل Open/Closed**: اضافه کردن خانواده‌های جدید بدون تغییر کد موجود

## معایب Abstract Factory

1. **پیچیدگی**: نیاز به تعریف تعداد زیادی رابط و کلاس دارد
2. **انعطاف‌پذیری محدود**: اضافه کردن محصولات جدید به خانواده موجود نیاز به تغییر abstract factory دارد
3. **یادگیری دشوار**: درک و پیاده‌سازی آن نسبت به الگوهای ساده‌تر دشوارتر است

## مثال کاربردی: سیستم UI چندپلتفرمی

// Abstract Products

public interface IButton {

void Render();

}

public interface ITextBox {

void Display();

}

// Concrete Products for Windows

public class WindowsButton : IButton {

public void Render() => Console.WriteLine("Windows style button rendered");

}

public class WindowsTextBox : ITextBox {

public void Display() => Console.WriteLine("Windows style textbox displayed");

}

// Concrete Products for Mac

public class MacButton : IButton {

public void Render() => Console.WriteLine("Mac style button rendered");

}

public class MacTextBox : ITextBox {

public void Display() => Console.WriteLine("Mac style textbox displayed");

}

// Abstract Factory

public interface IGUIFactory {

IButton CreateButton();

ITextBox CreateTextBox();

}

// Concrete Factories

public class WindowsFactory : IGUIFactory {

public IButton CreateButton() => new WindowsButton();

public ITextBox CreateTextBox() => new WindowsTextBox();

}

public class MacFactory : IGUIFactory {

public IButton CreateButton() => new MacButton();

public ITextBox CreateTextBox() => new MacTextBox();

}

// Client Code

public class Application {

private readonly IButton \_button;

private readonly ITextBox \_textBox;

public Application(IGUIFactory factory) {

\_button = factory.CreateButton();

\_textBox = factory.CreateTextBox();

}

public void RenderUI() {

\_button.Render();

\_textBox.Display();

}

}

## تفاوت با Factory Method

* **Abstract Factory**: ایجاد خانواده‌ای از محصولات مرتبط (چندین متد factory)
* **Factory Method**: ایجاد یک محصول (یک متد factory)

## استفاده در #C و .NET

1. **ADO.NET DbProviderFactory**:

DbProviderFactory factory = DbProviderFactories.GetFactory("System.Data.SqlClient");

DbConnection connection = factory.CreateConnection();

DbCommand command = factory.CreateCommand();

1. **Dependency Injection در ASP.NET Core**:

services.AddScoped<IHttpClientFactory, HttpClientFactory>();

## بهترین روش‌های پیاده‌سازی

1. از **رابط‌ها (Interfaces)** برای محصولات و factoryها استفاده کنید
2. **پارامتری کردن** انتخاب factory (مثلاً با استفاده از config)
3. استفاده از **Dependency Injection** برای تزریق factory مناسب
4. پیاده‌سازی **Lazy Initialization** برای محصولات اگر سنگین هستند
5. استفاده از **متدهای اختیاری** در رابط factory با پیاده‌سازی پیش‌فرض:

public interface IFurnitureFactory

{

IChair CreateChair();

ISofa CreateSofa();

ICoffeeTable CreateCoffeeTable() => new DefaultCoffeeTable();

}

این الگو زمانی مناسب است که:

* سیستم باید مستقل از نحوه ایجاد، ترکیب و نمایش محصولاتش باشد
* سیستم باید با چندین خانواده محصولات پیکربندی شود
* می‌خواهید خانواده‌ای از محصولات مرتبط را با هم استفاده کنید

-------------------------------------------------------------------------------------------------------