# **C# Abstract Factory**

Шаблон проектування Abstract Factory надає інтерфейс для створення сімейств пов'язаних або залежних об'єктів без вказівки їх конкретних класів.

**Учасники(Participants)**

Класи та об'єкти, що беруть участь у цьому паттерні, включають:

* **AbstractFactory (ContinentFactory) -** оголошує інтерфейс для операцій, які створюють абстрактні продукти
* **ConcreteFactory (AfricaFactory, AmericaFactory) -** реалізує операції для створення конкретних об'єктів продуктів.
* **AbstractProduct (Herbivore, Carnivore)** **-** оголошує інтерфейс для типу об'єкту продукту.
* **Product (Wildebeest, Lion, Bison, Wolf) -** визначає об'єкт продукту, який буде створений відповідною конкретною фабрикою та реалізує інтерфейс **AbstractProduct**.
* **Client (AnimalWorld) -** використовує інтерфейси, оголошені класами **AbstractFactory** та **AbstractProduct**.

**Структурний код на мові C#**

Цей структурний код демонструє патерн **Abstract Factory**, що створює паралельні ієрархії об'єктів. Створення об'єктів було абстраговано і немає необхідності в жорстко закодованих іменах класів у клієнтському коді.

using System;

namespace DoFactory.GangOfFour.Abstract.Structural

{

/// <summary>

/// MainApp startup class for Structural

/// Abstract Factory Design Pattern.

/// </summary>

class MainApp

{

/// <summary>

/// Entry point into console application.

/// </summary>

public static void Main()

{

// Abstract factory #1

AbstractFactory factory1 = new ConcreteFactory1();

Client client1 = new Client(factory1);

client1.Run();

// Abstract factory #2

AbstractFactory factory2 = new ConcreteFactory2();

Client client2 = new Client(factory2);

client2.Run();

// Wait for user input

Console.ReadKey();

}

}

/// <summary>

/// The 'AbstractFactory' abstract class

/// </summary>

abstract class AbstractFactory

{

public abstract AbstractProductA CreateProductA();

public abstract AbstractProductB CreateProductB();

}

/// <summary>

/// The 'ConcreteFactory1' class

/// </summary>

class ConcreteFactory1 : AbstractFactory

{

public override AbstractProductA CreateProductA()

{

return new ProductA1();

}

public override AbstractProductB CreateProductB()

{

return new ProductB1();

}

}

/// <summary>

/// The 'ConcreteFactory2' class

/// </summary>

class ConcreteFactory2 : AbstractFactory

{

public override AbstractProductA CreateProductA()

{

return new ProductA2();

}

public override AbstractProductB CreateProductB()

{

return new ProductB2();

}

}

/// <summary>

/// The 'AbstractProductA' abstract class

/// </summary>

abstract class AbstractProductA

{

}

/// <summary>

/// The 'AbstractProductB' abstract class

/// </summary>

abstract class AbstractProductB

{

public abstract void Interact(AbstractProductA a);

}

/// <summary>

/// The 'ProductA1' class

/// </summary>

class ProductA1 : AbstractProductA

{

}

/// <summary>

/// The 'ProductB1' class

/// </summary>

class ProductB1 : AbstractProductB

{

public override void Interact(AbstractProductA a)

{

Console.WriteLine(this.GetType().Name +

" interacts with " + a.GetType().Name);

}

}

/// <summary>

/// The 'ProductA2' class

/// </summary>

class ProductA2 : AbstractProductA

{

}

/// <summary>

/// The 'ProductB2' class

/// </summary>

class ProductB2 : AbstractProductB

{

public override void Interact(AbstractProductA a)

{

Console.WriteLine(this.GetType().Name +

" interacts with " + a.GetType().Name);

}

}

/// <summary>

/// The 'Client' class. Interaction environment for the products.

/// </summary>

class Client

{

private AbstractProductA \_abstractProductA;

private AbstractProductB \_abstractProductB;

// Constructor

public Client(AbstractFactory factory)

{

\_abstractProductB = factory.CreateProductB();

\_abstractProductA = factory.CreateProductA();

}

public void Run()

{

\_abstractProductB.Interact(\_abstractProductA);

}

}

}

##### **Output**

ProductB1 interacts with ProductA1

ProductB2 interacts with ProductA2