**Паттерн «Абстрактная фабрика»**

**Назначение и цель применения паттерна**

Паттерн «Абстрактная фабрика» применяется в том случае, когда необходимо оперировать взаимосвязанными объектами из разных семейств.

**UML-диаграмма**



Классы ProductA и ProductB представляют собой семейства объектов. От них наследуются объекты конкретного типа.

Интерфейс AbstractFactory содержит методы для создания объектов семейств ProductA и ProductB. Классы ConcreteFactory1 и ConcreteFactory2 реализуют этот интерфейс и порождают «конкретные» продукты.

**Области применения**

1. Интернет-магазин мебели: гарнитуры мебели из предметов одного типа, но разного стиля.
2. Компьютерные игры: для каждого персонажа свое сочетание оружие + магические способности.

Особенности паттерна

Клиентский код не зависит от конкретных класс фабрик и элементов интерфейса. Он общается с ними через абстрактные интерфейсы.

Текст программы

namespace AbstractFactory

{

//Интерфейс абстрактной фабрики

public interface IAbstractFactory

{

IProductA CreateProductA();

IProductB CreateProductB();

}

}

namespace AbstractFactory

{

//Интерфейс продукта семейства А

public interface IProductA

{

void WriteNameAndType();

}

}

namespace AbstractFactory

{

//Интерфейс продукта семейства В

public interface IProductB

{

void WriteNumber();

}

}

namespace AbstractFactory

{

public class ProductA1: IProductA

{

public void WriteNameAndType()

{

Console.WriteLine("Name: ProductA. Type: 1");

}

}

}

namespace AbstractFactory

{

public class ProductA2: IProductA

{

public void WriteNameAndType()

{

Console.WriteLine("Name: ProductA. Type: 2");

}

}

}

namespace AbstractFactory

{

public class ProductB1:IProductB

{

public void WriteNumber()

{

Console.WriteLine(1);

}

}

}

namespace AbstractFactory

{

public class ProductB2: IProductB

{

public void WriteNumber()

{

Console.WriteLine(2);

}

}

}

namespace AbstractFactory

{

public class Factory1: IAbstractFactory

{

public IProductA CreateProductA()

{

return new ProductA1();

}

public IProductB CreateProductB()

{

return new ProductB1();

}

}

}

namespace AbstractFactory

{

public class Factory2: IAbstractFactory

{

public IProductA CreateProductA()

{

return new ProductA2();

}

public IProductB CreateProductB()

{

return new ProductB2();

}

}

}

namespace AbstractFactory

{

public class ClientCode

{

private IProductA productA;

private IProductB productB;

public ClientCode(IAbstractFactory factory)

{

productA = factory.CreateProductA();

productB = factory.CreateProductB();

}

public void WriteNamaAndType()

{

productA.WriteNameAndType();

}

public void WriteNumber()

{

productB.WriteNumber();

}

}

}

namespace AbstractFactory

{

class Program

{

/\*

\* Вывод

\* Name: ProductA. Type:1

\* 1

\* 2

\*/

static void Main(string[] args)

{

ClientCode cc = new ClientCode(new Factory1());

cc.WriteNameAndType();

cc.WriteNumber();

ClientCode cc2 = new ClientCode(new Factory2());

cc2.WriteNumber();

}

}

}