**The Abstact Factory pattern**

Мотивиране

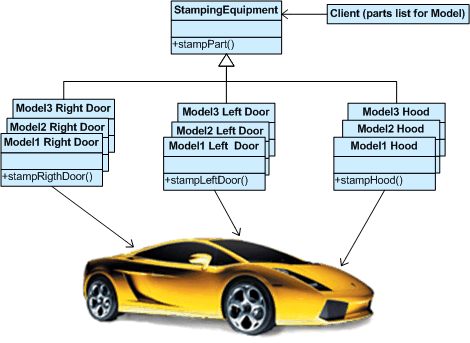
Създаване на интерфейс за изграждане на групи от зависими или свързани инстанции, които имплементират набор от абстрактни типове. Abstact Factoryкоординира инстанцииране на комплекти от обекти, които имат различни имплементации по такъв начин, че само легитимни комбинации от инстанции са възможни, и скрива тези инстанции зад набор от абстракции.

Намерение

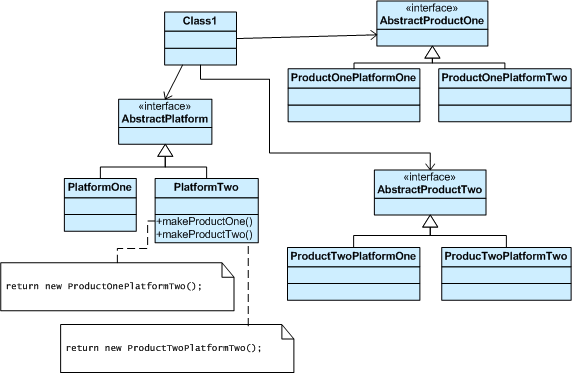
Доста често срещана необходимост от нещо като Abstract Factory възниква при поддържането на няколко операционни системи в едно приложение. За да постигнем това, ще е нужно да се избере най-подходящия набор от поведения: драйвер за диска, драйвер за мишка, графичен драйвер и т.н., за операционната система, на която в момента е инсталирано приложението.

Пример

Този модел може да се открие в оборудването за щамповане на метални листове, използвани при производството на японски автомобили. Оборудването за щамповане е Abstract Factory, която създава авточасти. Същата техника се използва за правене на щампи върху десните врати, левите врати, предния десен калник, предния ляв калник, капаците и т.н. за различни модели автомобили. Чрез използването на ролки за промяна на формите за щамповане, конкретните класове създавани от машината, може да се променят в рамките на три минути.



Структура



„Factory“ обекта има отговорността за предоставяне на услуги за създаване за цялата платформена фамилия. Клиентите никога създават платформени обекти директно, те карат фабриката да направи това за тях.

Този механизъм прави обмена на продуктови семейства лесно, защото специфичния клас на обекта на „фабриката“ се появява само веднъж в приложението - когато е инстанцииран. Приложението може едновременно да замени цялата фамилия продукти просто чрез инстанцииране на различна конкретна инстанция на абстрактната фабрика.

C# Example

using System;

**class** MainApp

{

**public** **static** void Main()

{

// Abstract factory #1

AbstractFactory factory1 = **new** ConcreteFactory1();

Client c1 = **new** Client(factory1);

c1.Run();

// Abstract factory #2

AbstractFactory factory2 = **new** ConcreteFactory2();

Client c2 = **new** Client(factory2);

c2.Run();

// Wait for user input

Console.Read();

}

}

// "AbstractFactory"

**abstract** **class** AbstractFactory

{

**public** **abstract** AbstractProductA CreateProductA();

**public** **abstract** AbstractProductB CreateProductB();

}

// "ConcreteFactory1"

**class** ConcreteFactory1 : AbstractFactory

{

**public** override AbstractProductA CreateProductA()

{

**return** **new** ProductA1();

}

**public** override AbstractProductB CreateProductB()

{

**return** **new** ProductB1();

}

}

// "ConcreteFactory2"

**class** ConcreteFactory2 : AbstractFactory

{

**public** override AbstractProductA CreateProductA()

{

**return** **new** ProductA2();

}

**public** override AbstractProductB CreateProductB()

{

**return** **new** ProductB2();

}

}

// "AbstractProductA"

**abstract** **class** AbstractProductA

{

}

// "AbstractProductB"

**abstract** **class** AbstractProductB

{

**public** **abstract** void Interact(AbstractProductA a);

}

// "ProductA1"

**class** ProductA1 : AbstractProductA

{

}

// "ProductB1"

**class** ProductB1 : AbstractProductB

{

**public** override void Interact(AbstractProductA a)

{

Console.WriteLine(**this**.GetType().Name + " interacts with " + a.GetType().Name);

}

}

// "ProductA2"

**class** ProductA2 : AbstractProductA

{

}

// "ProductB2"

**class** ProductB2 : AbstractProductB

{

**public** override void Interact(AbstractProductA a)

{

Console.WriteLine(**this**.GetType().Name + " interacts with " + a.GetType().Name);

}

}

// "Client" - the interaction environment of the products

**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);

}

}