Федеральное агентство железнодорожного транспорта
Сибирский государственный университет путей сообщения
Кафедра «Системный анализ и управление проектами»

Фабричный метод

Выполнила студентка
гр.БПИ-411
Шестакова А.А
Проверила Доцент
Красникова К.В.

Новосибирск 2017

Определение

- Фабричный метод (Factory Method) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах.

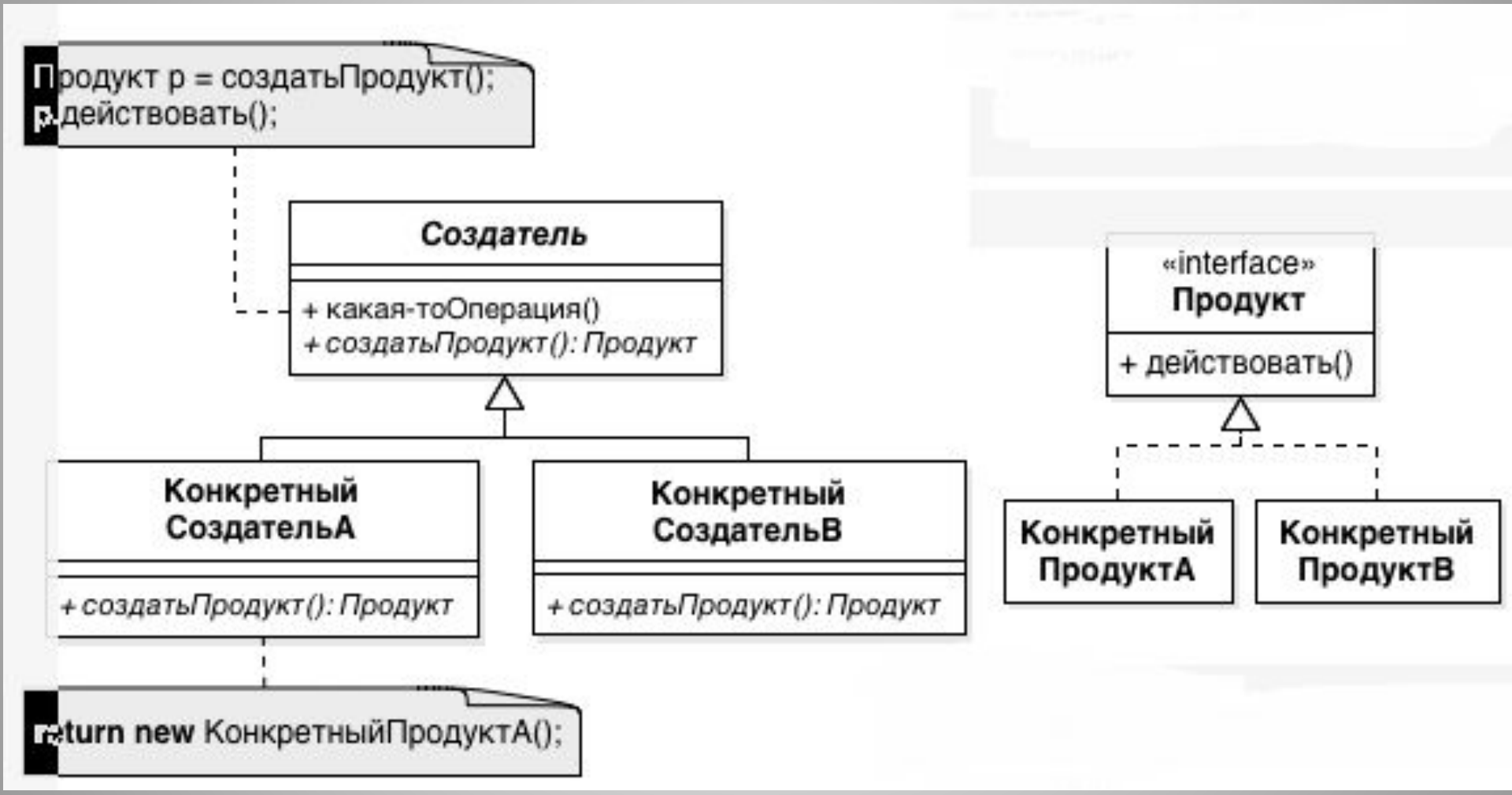
Задачи

- Система должна оставаться расширяемой путем добавления объектов новых типов. Непосредственное использование выражения new является нежелательным, так как в этом случае код создания объектов с указанием конкретных типов может получиться разбросанным по всему приложению. Тогда такие операции как добавление в систему объектов новых типов или замена объектов одного типа на другой будут затруднительными Паттерн Factory Method позволяет системе оставаться независимой как от самого процесса порождения объектов, так и от их типов.
- Заранее известно, когда нужно создавать объект, но неизвестен его тип.

Описание паттерна Factory Method

- Механизм полиморфизм (определение единого интерфейса);
- Создание объектов конкретных типов в специальном классе-фабрике:
  - ❖ Обобщенный конструктор;
  - ❖ Классический вариант фабричного метода.

Структура


Продукт p = создатьПродукт();
p.действовать();
Создатель
+ какая-тоОперация()
+ создатьПродукт(): Продукт
«interface»
Продукт
+ действовать()
Конкретный СоздательA
+ создатьПродукт(): Продукт
Конкретный СоздательB
+ создатьПродукт(): Продукт
Конкретный ПродуктA
Конкретный ПродуктB
return new КонкретныйПродуктA();

Когда надо применять паттерн

- Когда заранее неизвестно, объекты каких типов необходимо создавать
- Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать.
- Когда создание новых объектов необходимо делегировать из базового класса классам наследникам

Достоинства

- позволяет сделать код создания объектов более универсальным, не привязываясь к конкретным классам (ConcreteProduct), а оперируя лишь общим интерфейсом (Product);
- позволяет установить связь между параллельными иерархиями классов.

Недостатки

- необходимость создавать наследника Creator для каждого нового типа продукта (ConcreteProduct).

Формальное определение паттерна на языке C#

```
1  abstract class Product
2  {}
3
4  class ConcreteProductA : Product
5  {}
6
7  class ConcreteProductB : Product
8  {}
9
10 abstract class Creator
11 {
12     public abstract Product FactoryMethod();
13 }
14
15 class ConcreteCreatorA : Creator
16 {
17     public override Product FactoryMethod() { return new ConcreteProductA(); }
18 }
19
20 class ConcreteCreatorB : Creator
21 {
22     public override Product FactoryMethod() { return new ConcreteProductB(); }
23 }
```

Пример

```
class Program
{
    static void Main(string[] args)
    {
        Developer dev = new PanelDeveloper("ООО КирпичСтрой");
        House house2 = dev.Create();

        dev = new WoodDeveloper("Частный застройщик");
        House house = dev.Create();

        Console.ReadLine();
    }
}
// абстрактный класс строительной компании
abstract class Developer
{
    public string Name { get; set; }

    public Developer (string n)
    {
        Name = n;
    }
    // фабричный метод
    abstract public House Create();
}
// строит панельные дома
class PanelDeveloper : Developer
{
    public PanelDeveloper(string n) : base(n)
    { }

    public override House Create()
    {
        return new PanelHouse();
    }
}
// строит деревянные дома
class WoodDeveloper : Developer
{
    public WoodDeveloper(string n) : base(n)
    { }

    public override House Create()
    {
        return new WoodHouse();
    }
}

abstract class House
{ }

class PanelHouse : House
{
    public PanelHouse()
    {
        Console.WriteLine("Панельный дом построен");
    }
}
class WoodHouse : House
{
    public WoodHouse()
    {
        Console.WriteLine("Деревянный дом построен");
    }
}
```