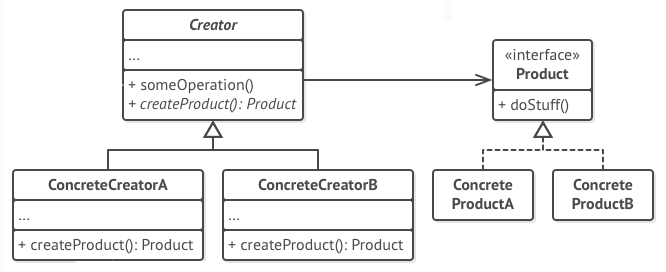
**Кукаркин И.А.**

**ПИ-15-1**

**Фабричный метод**

*Цель применения паттерна* - создания объектов различных типов одним интерфейсом.

*UML-диаграмма:*



Product — продукт. Определяет интерфейс объектов, создаваемых абстрактным методом;

ConcreteProduct — конкретный продукт. Реализует интерфейс Product;

Creator — создатель. Объявляет фабричный метод, который возвращает объект типа Product. Может также содержать реализацию этого метода «по умолчанию»; может вызывать фабричный метод для создания объекта типа Product;

ConcreteCreator — конкретный создатель. Переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объект класса ConcreteProduct.

*Используется, когда:*

* классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
* класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.
* класс делегирует свои обязанности одному из нескольких вспомогательных подклассов, и планируется локализовать знание о том, какой класс принимает эти обязанности на себя

*Особенности паттерна:*

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).

*Пример реализации на c#:*

**using** **System**;

**using** **System.Collections.Generic**;

**namespace** **Factory**

{

**abstract** **class** **Product**

{

**public** **abstract** string GetType();

}

**class** **ConcreteProductA** : Product

{

**public** **override** string GetType() { **return** "ConcreteProductA"; }

}

**class** **ConcreteProductB** : Product

{

**public** **override** string GetType() { **return** "ConcreteProductB"; }

}

**abstract** **class** **Creator**

{

**public** **abstract** Product FactoryMethod();

}

**class** **ConcreteCreatorA** : Creator

{

**public** **override** Product FactoryMethod() { **return** **new** ConcreteProductA(); }

}

**class** **ConcreteCreatorB** : Creator

{

**public** **override** Product FactoryMethod() { **return** **new** ConcreteProductB(); }

}

**public** **class** **MainApp**

{

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

{

*// an array of creators*

Creator[] creators = { **new** ConcreteCreatorA(), **new** ConcreteCreatorB() };

*// iterate over creators and create products*

**foreach** (Creator creator **in** creators)

{

Product product = creator.FactoryMethod();

Console.WriteLine("Created {0}", product.GetType());

}

*// Wait for user*

Console.Read();

}

}

}