**Лабораторная работа №1**

**«Реализация одного из порождающих паттернов проектирования»**

**Цель работы:** научиться применять порождающие паттерны проектирования.

**Продолжительность работы** - 4 часа.

**Содержание**

[Порождающие паттерны 1](#_Toc99797138)

[Паттерн Абстрактная фабрика 1](#_Toc99797139)

# **Порождающие паттерны**

Порождающие паттерны проектирования абстрагируют процесс инстанцирования объектов. Они позволяют сделать код независимым от способа создания, композиции и представления используемых в его работе объектов.

Список порождающих паттернов (GoF):

* Фабричный метод (*Factory method*)
* Абстрактная фабрика (*Abstract Factory*)
* Строитель (*Builder*)
* Прототип (*Prototype*)
* Одиночка (*Singleton*)

# **Паттерн Абстрактная фабрика**

**Назначение**

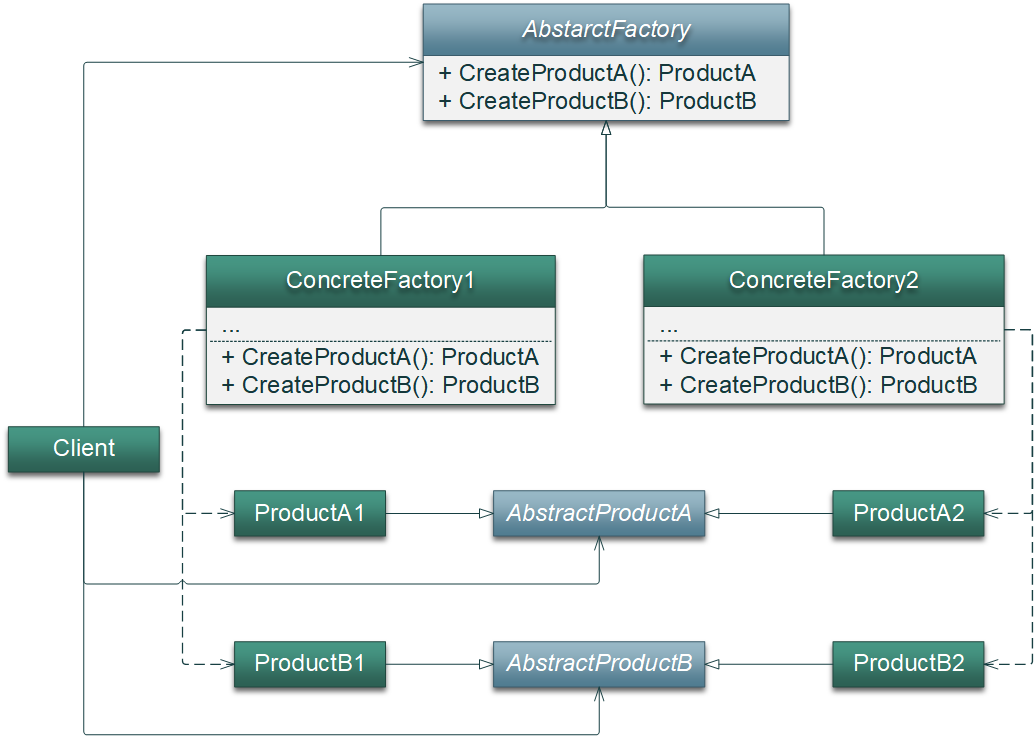
Абстрактная фабрика – паттерн, порождающий объекты. Этот паттерн позволяет работать с разными видами связанных друг с другом продуктов, не завися от конкретных продуктов. Он скрывает от клиентского кода подробности того, какие конкретные объекты будут созданы. Клиент будет работать со всеми типами конкретными продуктов через заранее определённый для них общих интерфейс.

**Применимость**

Использование паттерна Abstract Factory (абстрактная фабрика) целесообразно если:

* Система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты
* Входящие в семейство взаимосвязанные объекты должны использоваться вместе и вам необходимо обеспечить выполнение этого ограничения.
* Система должна конфигурироваться одним из семейств составляющих ее объектов, а вы хотите предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.
* Когда класс содержит много фабричных методов, чтобы он не потерял свою основную цель, рекомендуется перенести логику создания продуктов в отдельную иерархию классов. Один класс должен решать только какую-то одну задачу, такой принцип программирования называется *принципом единственной ответственности* (Single Responsibility Principle, SRP)

**Структура**



Участники

* AbstractFactory - абстрактная фабрика: объявляет интерфейс для операций, создающих абстрактные объекты-продукты
* ConcreteFactory (ConcreteFactory1, ConcreteFactory2) - конкретная фабрика: реализует операции, создающие конкретные объекты-продукты
* AbstractProduct (AbstractProductА, AbstractProductВ) - абстрактный продукт: объявляет интерфейс для типа объекта-продукта
* ConcreteProduct (ProductА, ProductВ) - конкретный продукт: определяет объектпродукт, создаваемый соответствующей конкретной - реализует интерфейс Abstract Product
* Client - клиент: пользуется исключительно интерфейсами, которые объявлены в классах AbstractFactory и AbstractProduct

**Примеры использования**

* Нужно выпускать машины с разным типом кузова, конкретными фабриками будут производители Toyota, Kia и Lada, а их конкретные продукты – седан, купе и универсал.
* Нужно выпускать одежду разного типа, конкретными фабриками будут производители ZARA, H&M и Bershka, а их конкретные продукты – верхняя одежда, нижняя и головные уборы.
* Когда делаете кроссплатформенное приложение, то для правильного отображения UI элементов на разных устройствах нужно использовать стиль и настройки, которые подходят именно этой платформе.

**Примеры использования**

* Нужно выпускать машины с разным типом кузова, конкретными фабриками будут производители Toyota, Kia и Lada, а их конкретные продукты – седан, купе и универсал.
* Нужно выпускать одежду разного типа, конкретными фабриками будут производители ZARA, H&M и Bershka, а их конкретные продукты – верхняя одежда, нижняя и головные уборы.
* Когда делаете кроссплатформенное приложение, то для правильного отображения UI элементов на разных устройствах нужно использовать стиль и настройки, которые подходят именно этой платформе.