Пермский филиал федерального государственного автономного
образовательного учреждения высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Рязанов Иван Дмитриевич

ОРГАНИЗАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ. ПОРОЖДАЮЩИЕ ПАТТЕРНЫ АБСТРАКТНАЯ ФАБРИКА И ОДИНОЧКА

Лабораторная работа

студента образовательной программы «Программная инженерия»
по направлению подготовки *09.03.04 Программная инженерия*

Руководитель
к.т.н., доцент кафедры Информационных технологий в бизнесе НИУ ВШЭ-Пермь

А.В. Кычкин

Пермь, 2020 год

Оглавление

Глава 1. Абстрактная фабрика

Название и классификация паттерна. Абстрактная фабрика - паттерн, порождающий объекты.

Назначение. Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, без указания их конкретных классов.

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.

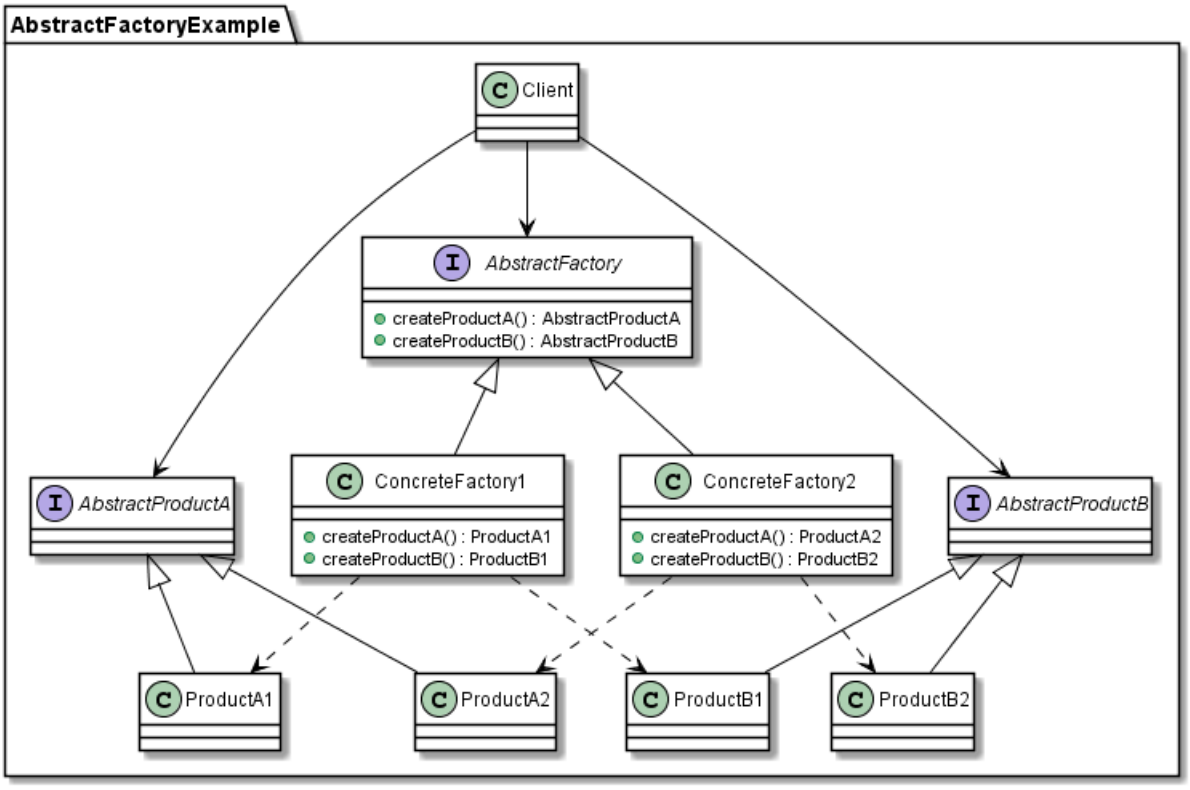

Рис. 1.1. Диаграмма классов паттерна «Абстрактная фабрика»

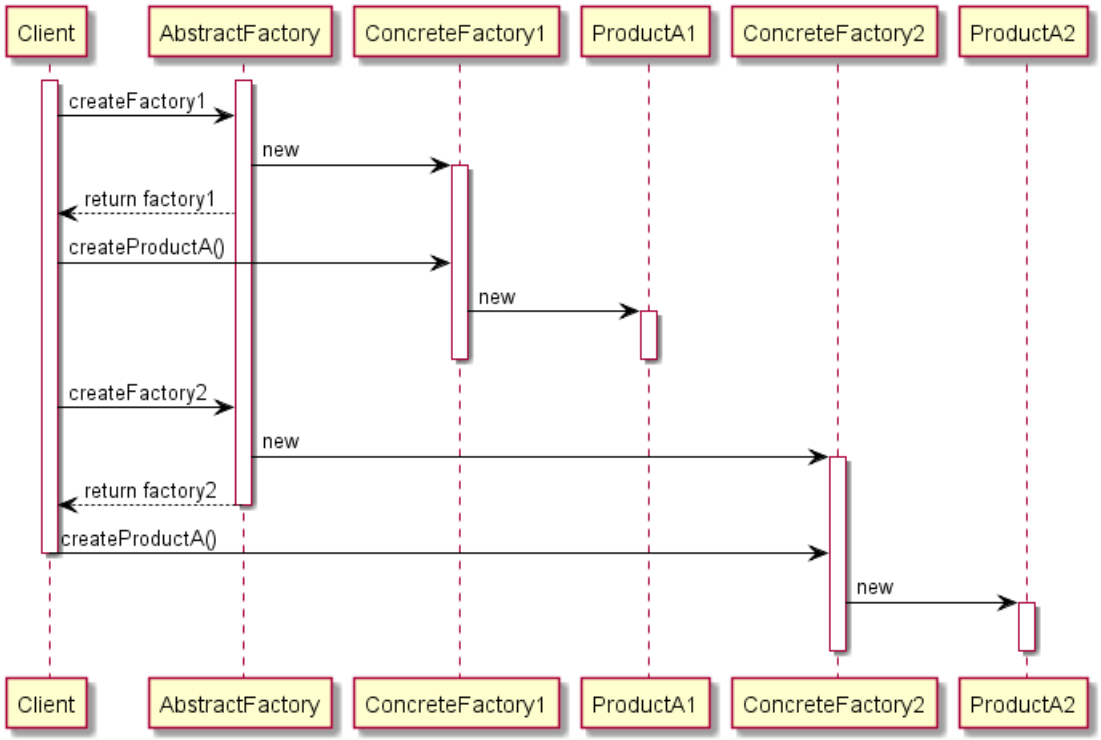

Рис. 1.2. Диаграмма последовательности паттерна «Абстрактная фабрика»

Участники

- AbstractFactory - абстрактная фабрика: объявляет интерфейс для операций, создающих абстрактные объекты-продукты;
- ConcreteFactory (ConcreteFactory1, ConcreteFactory2) - конкретная фабрика: реализует операции, создающие конкретные объекты-продукты;
- AbstractProduct (AbstractProductA, AbstractProductB) - абстрактный продукт: объявляет интерфейс для типа объекта-продукта;
- ConcreteProduct (ProductA, ProductB) - конкретный продукт: определяет объект-продукт, создаваемый соответствующей конкретной - реализует интерфейс Abstract Product;
- Client - клиент: пользуется исключительно интерфейсами, которые объявлены в классах AbstractFactory и AbstractProduct.

Отношения

- Обычно во время выполнения создается единственный экземпляр класса ConcreteFactory. Эта конкретная фабрика создает объекты-продукты, имеющие вполне определенную реализацию. Для создания других видов объектов клиент должен воспользоваться другой конкретной фабрикой;
- AbstractFactory передоверяет создание объектов-продуктов своему подклассу ConcreteFactory.

Плюсы и минусы.
Плюсы:

- *изолирует конкретные классы.* Помогает контролировать классы объектов, создаваемых приложением. Поскольку фабрика инкапсулирует ответственность за создание классов и сам процесс их создания, то она изолирует клиента от деталей реализации классов. Клиенты манипулируют экземплярами через их абстрактные интерфейсы. Имена изготавливаемых классов известны только конкретной фабрике, в коде клиента они не упоминаются;

- *упрощает замену семейств продуктов.* Класс конкретной фабрики появляется в приложении только один раз: при инстанцировании. Это облегчает замену используемой приложением конкретной фабрики.

- *гарантирует сочетаемость продуктов.* Если продукты некоторого семейства спроектированы для совместного использования, то важно, чтобы приложение в каждый момент времени работало только с продуктами единственного семейства. Класс AbstractFactory позволяет легко соблюсти это ограничение;

Минусы:

- *поддержать новый вид продуктов трудно.* Расширение абстрактной фабрики для изготовления новых видов продуктов - непростая задача. Интерфейс AbstractFactory фиксирует набор продуктов, которые можно создать. Для поддержки новых продуктов необходимо расширить интерфейс фабрики, то есть изменить класс AbstractFactory и все его подклассы.

Области применения

1. Создание кроссплатформенных интерфейсов.
2. Класс для работы с разными СУБД.

Глава 2. Одиночка

Название и классификация паттерна. Одиночка - паттерн, порождающий объекты.

Назначение. Гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Применимость. Использование паттерна Singleton (одиночка) целесообразно если:

- должен быть ровно один экземпляр некоторого класса, легко доступный всем клиентам;
- единственный экземпляр должен расширяться путем порождения подклассов, и клиентам нужно иметь возможность работать с расширенным экземпляром без модификации своего кода.

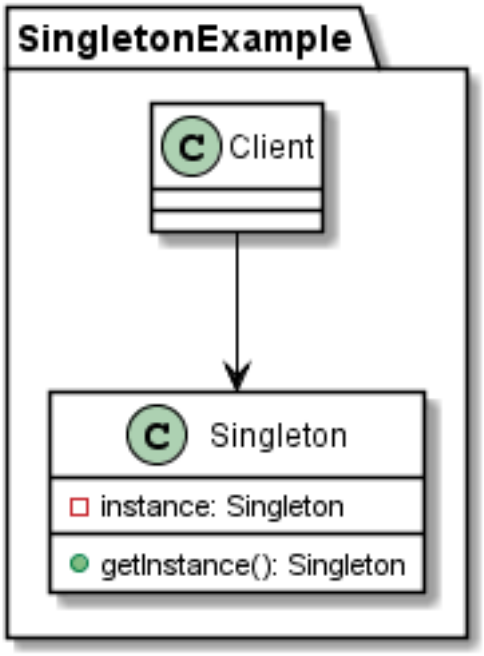

Рис. 2.1. Диаграмма классов паттерна «Одиночка»

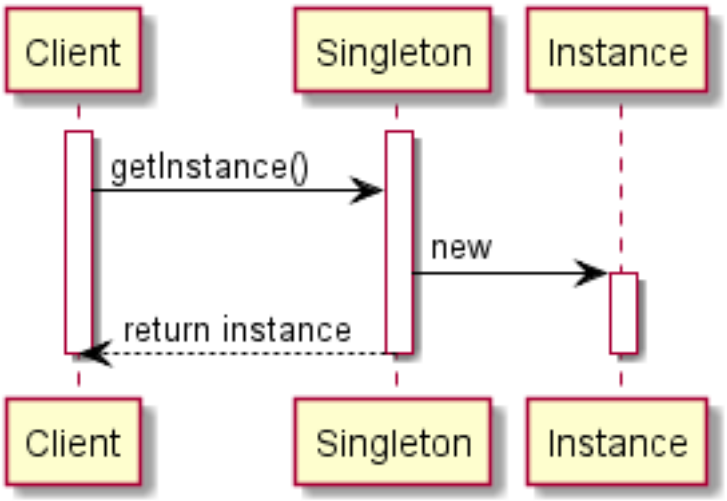

Рис. 2.2. Диаграмма последовательности паттерна «Одиночка»

Участники

Singleton - одиночка:

- определяет операцию getInstance, которая позволяет клиентам получать доступ к единственному экземпляру;
- может нести ответственность за создание собственного уникального экземпляра.

Отношения

Клиенты получают доступ к экземпляру класса Singleton только через его операцию getInstance.

Плюсы и минусы.

Плюсы:

- *контролируемый доступ к единственному экземпляру.* Поскольку класс Singleton инкапсулирует свой единственный экземпляр, он полностью контролирует то, как и когда клиенты получают доступ к нему;
- *уменьшение числа имен.* Паттерн одиночка - шаг вперед по сравнению с глобальными переменными. Он позволяет избежать засорения пространства имен глобальными переменными, в которых хранятся уникальные экземпляры;
- *допускает уточнение операций и представления.* От класса Singleton можно порождать подклассы, а приложение легко сконфигурировать экземпляром расширенного класса. Можно конкретизировать приложение экземпляром того класса, который необходим во время выполнения;
- *допускает переменное число экземпляров.* Паттерн позволяет вам легко изменить свое решение и разрешить появление более одного экземпляра класса Singleton. Вы можете применять один и тот же подход для управления числом экземпляров, используемых в приложении. Изменить нужно будет лишь операцию, дающую доступ к экземпляру класса Singleton;

Минусы:

- глобальные объекты могут быть вредны для объектного программирования, в некоторых случаях приводят к созданию немасштабируемого проекта;

- усложняет написание модульных тестов и следование TDD;
- усложняется контроль за межпоточными гонками и задержками.

Области применения

1. Ведение отладочного файла для приложения.
2. Класс для подключения к СУБД.

Глава 3. Проектирование и реализация

3.1. Проектирование

Для реализации был выбран 4 вариант:

«Приложение с поддержкой графического интерфейса пользователя рассчитано на использование на различных платформах мобильных устройств, при этом внешний вид этого интерфейса должен соответствовать принятому стилю для той или иной платформы. Например, если это приложение установлено на iOS, то его кнопки, меню, полосы прокрутки должны отображаться в стиле, принятом для приложений iOS. Группой взаимосвязанных объектов в этом случае будут элементы графического интерфейса для конкретной платформы.»

Перед началом работы построим диаграмму классов.

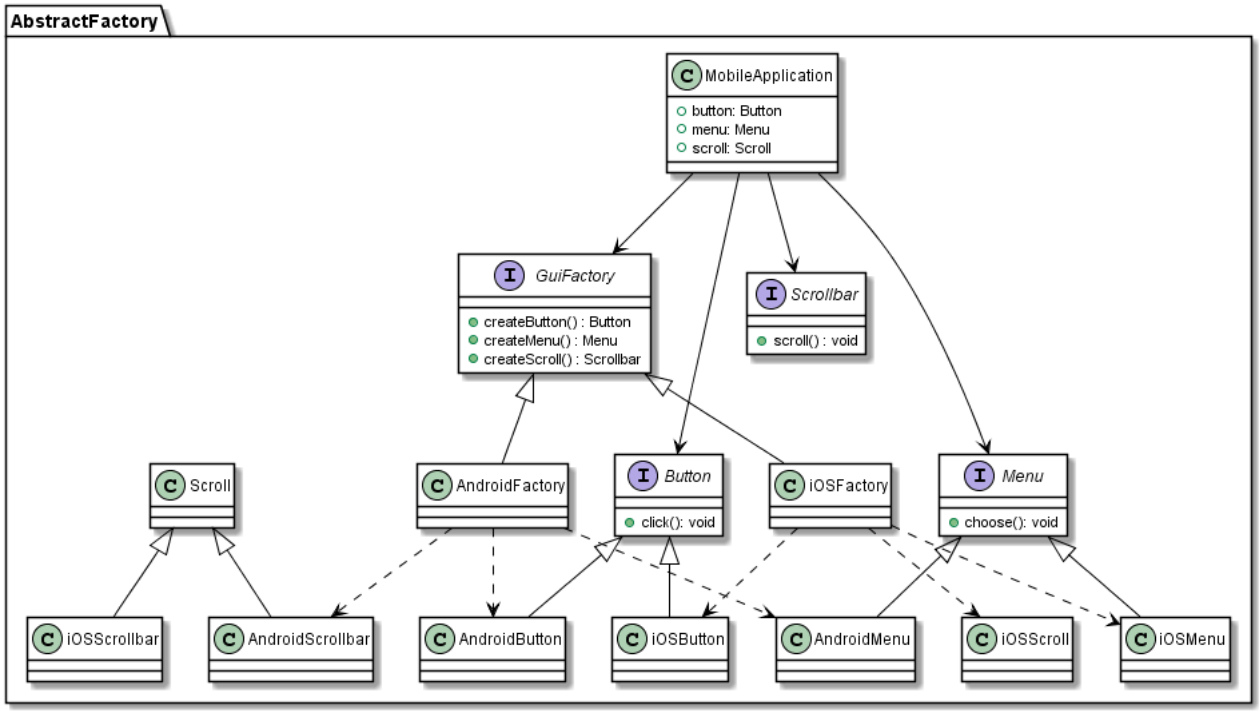

Рис. 3.1. Диаграмма классов

Участники.

- GuiFactory - абстрактная фабрика: объявляет интерфейс для операций, создающих абстрактные кнопки, меню и скроллбары;

- iOSFactory, AndroidFactory - фабрики для систем iOS и Android: реализует операции, создающие конкретные элементы интерфейса;
- Button, Menu, Scrollbar - абстрактный элементы интерфейса: объявляют интерфейс для типа объекта-продукта;
- iOSButton, iOSMenu, iOSScrollbar, AndroidButton, AndroidMenu, AndroidScrollbar - конкретные реализации элементов интерфейса: определяют объекты-продукты, создаваемые соответствующей конкретной фабрикой - реализуют интерфейсы Button, Menu, Scrollbar;
- MobileApplication - клиент: пользуется исключительно интерфейсами, которые объявлены в классах GuiFactory, Button, Menu, Scrollbar.

3.2. Реализация

Код программы приведен в репозитории по адресу: github.com