**ОТЧЁТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**Patterns**

**(Вариант 4)**

*Выполнил студент 3 курса МОиАИС*

*Гвозденко Демид*

**Паттерн проектирования “Фабрика”**

**Введение**

**Паттерн "Фабрика" (Factory)** - это один из наиболее распространенных **порождающих** паттернов проектирования в объектно-ориентированном программировании. Он предоставляет способ создания объектов, не открывая реализацию, а вместо этого использует общий интерфейс.

В паттерне "Фабрика" родительский класс отвечает за определение публичного интерфейса производных объектов, а подкласс — за создание конкретных "продуктовых" объектов. Суть в том, что реализация специфических свойств продукта делегируется подклассу фабрики. Другим словами, именно в подклассе фабрики определяется, какой продукт создается.

**Преимущества использования:**

* + 1. *Инкапсуляция процесса создания объектов:*
       - Фабрика скрывает детали реализации создания объектов от клиентского кода. Это упрощает использование и тестирование клиентского кода.
    2. *Гибкость и масштабируемость:*
       - Легко добавлять новые типы фабрик и продуктов без изменения клиентского кода. Это позволяет расширять систему, не нарушая ее существующую структуру.
    3. *Упрощение тестирования:*
       - Фабрики можно заменять на "заглушки" во время тестирования. Это облегчает изолированное тестирование компонентов системы.
    4. *Удобство использования:*
       - Клиентский код не знает о конкретных реализациях, что упрощает его. Клиент работает только с абстрактными интерфейсами, делая код более понятным и гибким.

**Основные роли паттерна “Фабрика”:**

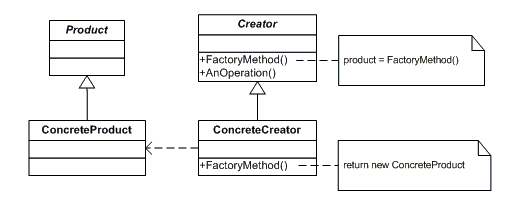
1. *Абстрактная фабрика:*
   1. Это интерфейс или абстрактный класс, который определяет методы для создания семейства связанных объектов.
   2. Например, интерфейс GUIFactory может определять методы createButton(), createTextField(), createWindow().
2. *Конкретные фабрики:*
   1. Это реализации абстрактной фабрики, которые отвечают за создание конкретных типов объектов.
   2. Например, WindowsGUIFactory и MacGUIFactory могут быть конкретными реализациями GUIFactory, создающими элементы GUI, соответствующие разным платформам.
3. *Абстрактные продукты:*
   1. Это интерфейсы или абстрактные классы, определяющие общий контракт для семейства связанных объектов.
   2. Например, интерфейсы Button, TextField, Window могут быть абстрактными продуктами.
4. *Конкретные продукты:*
   1. Это реализации абстрактных продуктов, создаваемые конкретными фабриками.
   2. Например, WindowsButton, MacButton, WindowsTextField, MacTextField, WindowsWindow, MacWindow могут быть конкретными реализациями абстрактных продуктов.

**Взаимодействие между этими компонентами выглядит следующим образом:**

1. Клиент создает экземпляр конкретной фабрики (например, WindowsGUIFactory или MacGUIFactory).
2. Клиент использует методы абстрактной фабрики (например, createButton(), createTextField(), createWindow()) для создания объектов.
3. Конкретная фабрика возвращает клиенту экземпляры соответствующих конкретных продуктов (например, WindowsButton, MacButton, WindowsTextField, MacTextField, WindowsWindow, MacWindow).

**UML-диаграмма**

UML— язык графического описания для объектного моделирования в области разработки программного обеспечения, который также можно применять для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.



Рассмотрим его по частям:

* **Creator**, обычно абстрактный базовый класс, который определяет и использует абстрактный фабричный метод в своей базовой функциональности:
* **ConcreteCreator** - реализация (наследник) абстрактного создателя. ConcreteCreator должен переопределить абстрактный фабричный метод, чтобы вернуть ConcreteProduct для Creator. Затем абстрактный создатель использует этот экземпляр для реализации базовой функциональности. Поскольку возвращаемым типом фабричного метода является abstractProduct, абстрактный создатель никогда не знает конкретный тип ConcreteProduct, а значит абстрактный создатель использует результат фабричного метода анонимно.
* **Product**, представляет собой абстрактный базовый класс для типов, которые абстрактный создатель использует для реализации базовой функциональности.
* **ConcreteProduct**, реализация (наследник) абстрактного продукта и фактический (анонимный) экземпляр, с которым работает абстрактный создатель.

**Пример**

*Рассмотрим “работу” паттерна “Фабрика” на примере RPG игры:*

Допустим, у нас есть игра, в которой игрок борется с врагами. Враги существуют двух типов: обычные и элитные.

У нас будет существовать абстрактная фабрика EnemyFactory, которая будет родительским классом для двух конкретных фабрик по созданию монстров разного типа: OrdinaryEnemyFactory (для создания обычных врагов) и EliteEnemyFactory (для создания элитных врагов).

В качестве Product будет объект Enemy, определяющий общий интерфейс для всех типов врагов. А ConcreteProduct будут являться OrdinaryEnemy и EliteEnemy, представляющие разные типы врагов в игре.

**Вывод**

В целом, паттерн "Фабрика" является одним из ключевых паттернов проектирования, который помогает решать задачи, связанные с созданием и конфигурированием объектов в программных системах.