Лабораторная работа №7

Исследование способов применения порождающих паттернов проектирования при рефакторинге ПО

1.Цель работы

Исследовать возможность использования порождающих паттернов проектирования. Получить практические навыки применения порождающих паттернов при объектно-ориентированном проектировании и рефакторинге ПО.

2. Основные положения

2.1. Порождающие паттерны

Порождающие паттерны проектирования абстрагируют процесс инстанцирования. Они помогут сделать систему независимой от способа создания, композиции и представления объектов. Паттерн, порождающий классы, использует наследование, чтобы варьировать инстанцируемый класс, а паттерн, порождающий объекты, делегирует инстанцирование другому объекту.

Эти паттерны оказываются важны, когда система больше зависит от композиции объектов, чем от наследования классов. Получается так, что основной упор делается не на жестком кодировании фиксированного набора поведений, а на определении небольшого набора фундаментальных поведений, с помощью композиции которых можно получать любое число более сложных. Таким образом, для создания объектов с конкретным поведением требуется нечто большее, чем простое инстанцирование класса.

Для порождающих паттернов актуальны две темы. Во-первых, эти паттерны инкапсулируют знания о конкретных классах, которые применяются в системе.

Во-вторых, скрывают детали того, как эти классы создаются и стыкуются. Единственная информация об объектах, известная системе, – это их интерфейсы, определенные с помощью абстрактных классов. Следовательно, порождающие паттерны обеспечивают большую гибкость при решении вопроса о том, что создается, кто это создает, как и когда. Можно собрать систему из «готовых» объектов с самой различной структурой и функциональностью статически (на этапе компиляции) или динамически (во время выполнения).

2.2. Паттерн «Абстрактная фабрика»

Назначение

Предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Мотивация

Рассмотрим инструментальную программу для создания пользовательского интерфейса, поддерживающего разные стандарты внешнего облика, например Motif и Presentation Manager. Внешний облик определяет визуальное представление и поведение элементов пользовательского интерфейса («виджетов») – полос прокрутки, окон и кнопок. Чтобы приложение можно было перенести на другой стандарт, в нем не должен быть жестко закодирован внешний облик виджетов.

Если инстанцирование классов для конкретного внешнего облика разбросано по всему приложению, то изменить облик впоследствии будет нелегко.

Можно решить эту проблему, определив абстрактный класс `WidgetFactory` (рисунок 2.1), в котором объявлен интерфейс для создания всех основных видов виджетов. Есть также абстрактные классы для каждого отдельного вида и конкретные подклассы, реализующие виджеты с определенным внешним обликом. В интерфейсе `WidgetFactory` имеется операция, возвращающая новый объект-виджет для каждого абстрактного класса виджетов. Клиенты вызывают эти операции для получения экземпляров виджетов, но при этом ничего не знают о том, какие именно классы используют. Стало быть, клиенты остаются независимыми от выбранного стандарта внешнего облика.

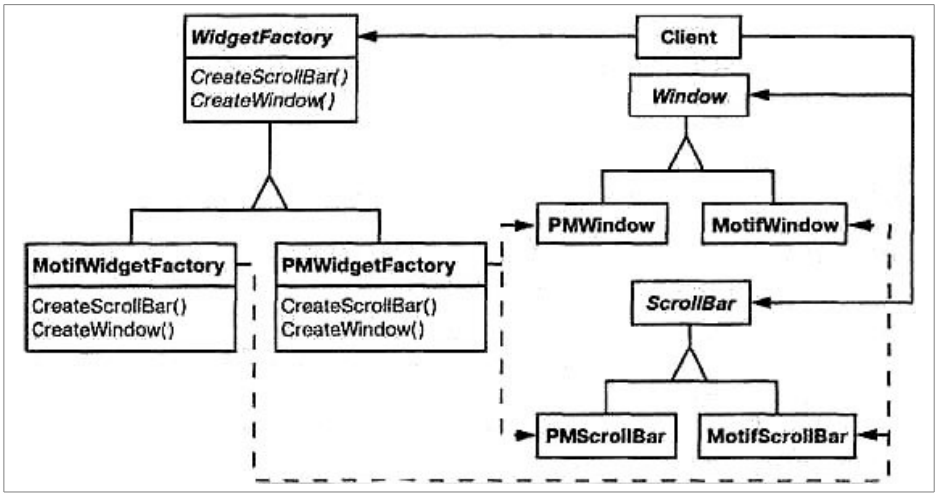

Рисунок 2.1 – Диаграмма классов

Для каждого стандарта внешнего облика существует определенный подкласс `WidgetFactory`. Каждый такой подкласс реализует операции, необходимые для создания соответствующего стандарту виджета. Например, операция `CreateScrollBar` в классе `MotifWidgetFactory` инстанцирует и возвращает полосу прокрутки в стандарте Motif, тогда как соответствующая операция в классе `PMWidgetFactory` возвращает полосу прокрутки в стандарте Presentation Manager. Клиенты создают виджеты, пользуясь исключительно интерфейсом `WidgetFactory`, и им ничего не известно о классах, реализующих виджеты для конкретного стандарта.

Другими словами, клиенты должны лишь придерживаться интерфейса, определенного абстрактным, а не конкретным классом.

Класс `WidgetFactory` также устанавливает зависимости между конкретными классами виджетов. Полоса прокрутки для Motif должна использоваться с кнопкой и текстовым полем Motif, и это ограничение поддерживается автоматически, как следствие использования класса `MotifWidgetFactory`.

Применимость

Паттерн ***абстрактная фабрика*** следует использовать, когда:

- система не должна зависеть от того, как создаются, компонуются и представляются входящие в нее объекты;
- входящие в семейство взаимосвязанные объекты должны использоваться вместе и необходимо обеспечить выполнение этого ограничения;
- система должна конфигурироваться одним из семейств составляющих ее объектов;
- необходимо предоставить библиотеку объектов, раскрывая только их интерфейсы, но не реализацию.

Структура

Диаграмма классов для паттерна ***Абстрактная фабрика*** показана на рисунке 2.2.

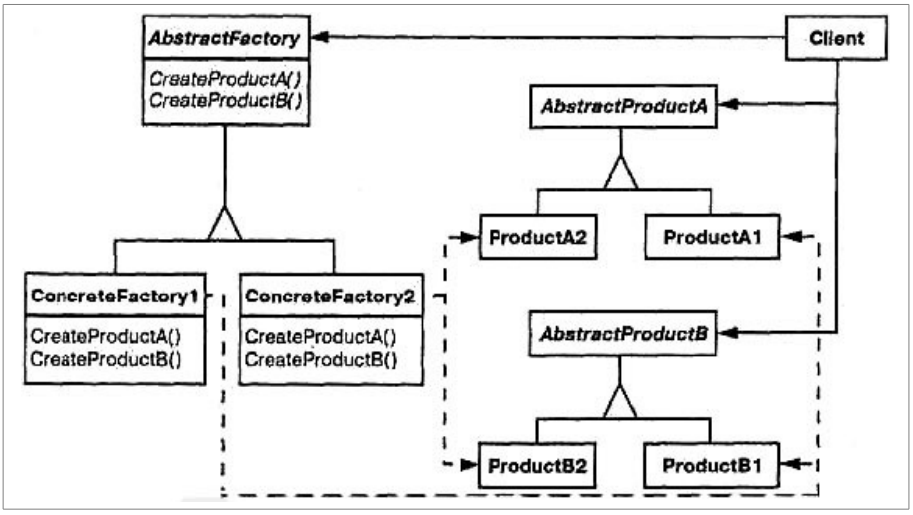

Рисунок 2.2 – Паттерн Абстрактная фабрика

Участники

- `AbstractFactory` (`WidgetFactory`) – абстрактная фабрика. Объявляет интерфейс для операций, создающих абстрактные объекты-продукты;
- `ConcreteFactory` (`MotifWidgetFactory`, `PMWidgetFactory`) – конкретная фабрика. Реализует операции, создающие конкретные объекты-продукты;
- `AbstractProduct` (`Window`, `ScrollBar`) – абстрактный продукт. Объявляет интерфейс для типа объекта-продукта;

- `ConcreteProduct` (`MotifWindow`, `MotifScrollBar`) – конкретный продукт. Определяет объект-продукт, создаваемый соответствующей конкретной фабрикой; реализует интерфейс `AbstractProduct`;
- `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`.

Результаты

Паттерн ***абстрактная фабрика*** обладает следующими плюсами и минусами:

- 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. Приложение может изменить конфигурацию продуктов, просто подставив новую конкретную фабрику. Поскольку абстрактная фабрика создает все семейство продуктов, то и заменяется сразу все семейство. В нашем примере пользовательского интерфейса перейти от виджетов Motif к виджетам Presentation Manager можно, просто переключившись на продукты соответствующей фабрики и заново создав интерфейс;
- 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;
- 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. Решение этой проблемы мы обсудим в разделе «Реализация».

3. Порядок выполнения работы

3.1. Изучить назначение и структуру паттерна ***Абстрактная фабрика*** (выполнить в ходе самостоятельной подготовки).

3.2. Применительно к программному продукту, выбранному для рефакторинга, проанализировать возможность использования паттерна ***Абстрактная фабрика***. Для этого построить диаграмму классов, на диаграмме классов выделить семейства взаимосвязанных и совместно используемых классов, которые должны инстанцироваться совместно и при этом инстанцирующий их клиент не должен быть привязан к конкретным именам классов (пример приведен в разделе 2.2.).

3.3. Выполнить перепроектирование системы, использовав паттерн ***Абстрактная фабрика***, изменения отобразить на диаграмме классов.

3.4. Сравнить полученные диаграммы классов, сделать выводы и целесообразности использования паттернов проектирования для данной системы.

3.5. На основе полученной UML-диаграммы модифицировать программный код, скомпилировать программу, выполнить ее тестирование и продемонстрировать ее работоспособность.

4. Содержание отчета

4.1. Цель работы.
4.2. Постановка задачи с описанием программного продукта, для которого проводится рефакторинг.
4.3. Словесное описание мотивации применения паттерна ***Абстрактная фабрика*** при проектировании данной системы.
4.4. UML-диаграмма классов с комментариями.
4.5. Текст программы.
4.6. Выводы по работе.

5. Контрольные вопросы

5.1. Для чего предназначены поведенческие паттерны проектирования?
5.5. Какие задачи решает паттерн «Абстрактная фабрика»?
5.6. Какие классы входят в состав паттерна «Абстрактная фабрика», каковы их обязанности?

Библиографический список

1. Фаулер М. Рефакторинг: улучшение существующего кода / М. Фаулер.– Пер. с англ.– СПб: Символ-Плюс, 2003.– 432 с.

2. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес.– СПб.: «Питер», 2009.– 366 с.

3. Рамбо Дж.UML 2.0. Объектно-ориентированное моделирование и разработка / Дж. Рамбо, М. Блаха.– М. и др. : «Питер», 2007.– 544 с.