4.18 Шаблоны проектирования, их применение. Классификация шаблонов проектирования. Примеры шаблонов проектирования

Шаблоны проектирования (паттерн, pattern) — это эффективные способы решения характерных задач проектирования, в частности проектирования компьютерных программ. Паттерн не является законченным образцом проекта, который может быть прямо преобразован в код, скорее это описание или образец для того, как решить задачу, таким образом, чтобы это можно было использовать в различных ситуациях. Объектно-ориентированные шаблоны зачастую показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться. Алгоритмы не рассматриваются как шаблоны, так как они решают задачи вычисления, а не проектирования.

Классификация паттернов

- *Основные шаблоны (Fundamental)*
- Порождающие шаблоны проектирования
- Структурные шаблоны (Structural)
- Поведенческие шаблоны (Behavioral)

Гамма и компания выделяют только последние три группы паттернов. Первая группа содержит общие идеи, применяемые в разных конкретных паттернах.

Основные шаблоны (Fundamental)

Шаблон делегирования

> В разработке ПО, **шаблон делегирования** (англ. *delegation pattern*) — это способ, которым объект внешне выражает некоторое поведение, но в реальности передаёт ответственность за выполнение этого поведения связанному объекту. Шаблон делегирования является фундаментальной абстракцией которая поддерживает композицию (также называемую агрегацией), примеси (mixins) и аспекты (aspects).

Шаблон функционального дизайна

> **Шаблон функционального дизайна** (англ. *Functional design*) — это шаблон проектирования, использующийся для упрощения проектирования ПО. Функциональный дизайн гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет её с минимумом

побочных эффектов на другие части программы. Функционально разработанные модули имеют очень маленькое сцепление.

Неизменяемый (Immutable)

Неизменяемый объект (англ. *Immutable object*) — в объектно-ориентированном программировании объект, который не может быть изменен после своего создания.

Объект может быть неизменяемым как полностью, так и частично. Например, применение директивы *const* к какому-либо члену класса в C++ делает объект частично неизменяемым. В некоторых случаях объект считается неизменяемым с точки зрения пользователя класса, даже если изменяются его внутренние поля. Как правило, неизменяемый объект получает все внутренние значения во время инициализации, либо значения устанавливаются в несколько этапов, но до того, как объект будет использован.

Неизменяемые объекты часто используются для устранения дорогих операций копирования и сравнения, для упрощения кода и увеличения скорости исполнения. Однако неуместно делать объект неизменяемым, если в нем есть большое количество изменяемых данных.

Порождающие шаблоны проектирования

Abstract Factory/Абстрактная фабрика

Абстрактная фабрика, **Abstract factory** — шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемые объекты, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса, он может создавать окна и кнопки). Затем пишутся наследующиеся от него классы, реализующие этот интерфейс по-своему.

Фабричный метод

Цель: определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанциировать. Фабричный метод позволяет классу делегировать создание подклассам. Используется, когда классу заранее неизвестно, объекты каких подклассов ему нужно создавать или класс спроектирован так, чтобы объекты, которые он создаёт, специфицировались подклассами.

4.18

Ленивая инициализация (Proxy)

Ленивая (отложенная) инициализация - приём в программировании, когда некоторая ресурсоемкая операция (создание объекта, вычисление значения) выполняется непосредственно перед тем, как будет использован ее результат. Таким образом, инициализация выполняется "по требованию", а не заблаговременно.

Структурные шаблоны (Structural)

Адаптер

Адаптер, **Adapter** — структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

Шаблон Адаптер позволяет в процессе проектирования не принимать во внимание возможные различия в интерфейсах уже существующих классов. Если есть класс, обладающий требуемыми методами и свойствами (по крайней мере, концептуально), то при необходимости всегда можно воспользоваться шаблоном Адаптер для приведения его интерфейса к нужному виду.

Мост

Bridge, **Мост** — шаблон проектирования, используемый в проектировании программного обеспечения, означает «разделять абстракцию и реализацию так, чтобы они могли изменяться независимо». Шаблон bridge (от англ. — мост) использует инкапсуляцию, агрегирование и может использовать наследование для того, чтобы разделить ответственность между классами. Пример: строим абстрактную иерархию оконной библиотеки, строим иерархию классов реализации под конкретную платформу (в ней реализуем все платформо-зависимые методы). После этого мост связывает корни иерархии так, что для создания объектов необходимо знать только о корневом классе иерархии реализации (для создания объектов – фабричный метод).

Декоратор

Декоратор, **Decorator** — структурный шаблон проектирования, предназначенный для динамического подключения дополнительных обязательств к объекту. Шаблон Декоратор предоставляет гибкую альтернативу практике определения подклассов с целью расширения функциональности. Декоратор агрегирует объект и расширяет его функциональность.

Добавляемая функциональность реализуется в небольших объектах. Преимущество состоит в возможности динамически добавлять эту функциональность до или после основной функциональности декорируемого объекта.

Поведенческие шаблоны (Behavioral)

Iterator/Итератор

Представляет собой объект, позволяющий последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящий в состав агрегации (т.е. без необходимости знания о конкретных классах этих объектов).

Observer/Наблюдатель, Dependents, Publish-Subscribe, Listener

Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии.

Область применения: шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующими свойствами: существует, как минимум, один объект, рассылающий сообщения, имеется не менее одного получателя сообщений, причём их количество и состав могут изменяться во время работы приложения.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают с предоставленной им информацией получатели.

Strategy/Стратегия

Стратегия, **Strategy** — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

Шаблон Strategy определяет семейство алгоритмов. Это позволяет отказаться от использования переключателей и/или условных операторов. Вызов всех алгоритмов должен осуществляться стандартным образом (все они должны иметь одинаковый интерфейс).