Эссе

212 Скорбилин Илья

28 сентября 2023 г.

1 GOF паттерн

1.1 Что такое паттерн GOF

Паттерн GOF (Gang of Four) — это набор основных шаблонов проектирования, описанных в книге «Design Patterns: Elements of Reusable Object-Oriented Software» («Приемы объектно-ориентированного проектирования. Паттерны проектирования») авторства Эриха Гаммы, Ричарда Хелма, Ральфа Джонсона и Джона Влиссидеса. Эта книга, изданная в 1994 году, стала одной из наиболее известных и влиятельных книг в области проектирования программного обеспечения.

Книга определяет 23 различных шаблона проектирования, которые объединены в три категории:

1.2 Шаблоны создания (Creational Patterns)

1. Фабричный метод (Factory Method): определяет интерфейс для создания объектов, позволяя подклассам выбирать класс для инстанцирования.

2. Абстрактная фабрика (Abstract Factory): предоставляет интерфейс для создания семейств взаимосвязанных объектов без указания их конкретных классов.

3. Строитель (Builder): предоставляет способ создания сложного объекта шаг за шагом, не раскрывая его внутреннего представления.

4. Прототип (Prototype): определяет протокол создания объекта путем копирования уже существующего объекта вместо создания нового экземпляра с нуля. Это позволяет создавать новые объекты, избегая сложной логики инициализации.

5. Одиночка (Singleton): гарантирует, что класс имеет только один экземпляр, и предоставляет глобальную точку доступа к этому экземпляру.

1.3 Шаблоны структуры (Structural Patterns)

1. Адаптер (Adapter): преобразует интерфейс одного класса в другой, чтобы классы с несовместимыми интерфейсами могли работать вместе.

2. Мост (Bridge): разделяет абстракцию от ее реализации, позволяя им меняться независимо друг от друга. Это позволяет легко добавлять новые варианты реализации без изменения абстракции.

3. Компоновщик (Composite): обрабатывает отдельные объекты и группы объектов единообразно. Он позволяет создавать иерархические древовидные структуры из объектов и работать с ними, как с единым объектом.

4. Декоратор (Decorator): динамически добавляет новую функциональность объекту путем оборачивания его в другой класс.

5. Фасад (Facade): предоставляет унифицированный интерфейс для набора интерфейсов в сложной системе. Он упрощает взаимодействие с системой, скрывая ее сложность и предоставляя удобный способ работы с ней.

6. Легковес (Flyweight): позволяет эффективно поддерживать множество мелких объектов, используя общие данные и сокращая использование памяти.

7. Заместитель (Proxy): представляет объект-заместитель, контролирующий доступ к другому объекту и предоставляющий дополнительную функциональность.

1.4 Шаблоны поведения (Behavioral Patterns)

1. Цепочка обязанностей (Chain of Responsibility): позволяет создавать цепочку объектов, которые последовательно обрабатывают запросы, передавая их по цепи, пока один из объектов не обработает запрос или он не достигнет конца цепи.

2. Команда (Command): инкапсулирует запрос в виде объекта, позволяя параметризовать клиентов с разными запросами, организовывать историю команд и поддерживать отмену операций.

3. Итератор (Iterator): предоставляет способ последовательного доступа к элементам коллекции без раскрытия ее внутренней структуры.

4. Медиатор (Mediator): определяет объект, который инкапсулирует способ взаимодействия между набором объектов, обеспечивая слабую связь между ними. Это позволяет уменьшить зависимости между объектами и сделать систему более гибкой.

5. Хранитель (Memento): позволяет зафиксировать и сохранить внутреннее состояние объекта так, чтобы его можно было восстановить позже, без раскрытия деталей реализации.

6. Наблюдатель (Observer): определяет зависимость «один-ко-многим» между объектами, чтобы при изменении состояния одного объекта все зависящие от него объекты автоматически обновлялись.

7. Состояние (State): позволяет объекту изменять свое поведение при изменении его внутреннего состояния.

8. Стратегия (Strategy): определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми.

9. Шаблонный метод (Template Method): определяет скелет алгоритма, перекладывая некоторые шаги на подклассы.

10. Посетитель (Visitor): позволяет добавлять новые операции к объектам без изменения их классов. Он достигается путем вынесения операций в отдельные классы посетителей, которые могут быть применены к объектам.

2 Паттерн Facade

2.1 Название и классификация паттерна

Фасад - паттерн, структурирующий объекты.

2.2 Назначение

Предоставляет унифицированный интерфейс вместо набора интерфейсов не- которой подсистемы. Фасад определяет интерфейс более высокого уровня, кото- рый упрощает использование подсистемы.

2.3 Мотивация

Разбиение на подсистемы облегчает проектирование сложной системы в целом. Общая цель всякого проектирования - свести к минимуму зависимость подсистем друг от друга и обмен информацией между ними. Один из способов решения этой задачи - введение объекта фасад, предоставляющий единый упрощенный интер- фейс к более сложным системным средствам.

Рассмотрим, например, среду программирования, которая дает приложениям доступ к подсистеме компиляции. В этой подсистеме имеются такие классы, как Scanner (лексический анализатор), Parser (синтаксический анализатор), ProgramNode (узел программы), BytecodeStream (поток байтовых кодов) и ProgramNodeBuilder (строитель узла программы). Все вместе они состав- ляют компилятор. Некоторым специализированным приложениям, возможно, понадобится прямой доступ к этим классам. Но для большинства клиентов ком- пилятора такие детали, как синтаксический разбор и генерация кода, обычно не нужны; им просто требуется откомпилировать некоторую программу. Для таких клиентов применение мощного, но низкоуровневого интерфейса подсистемы компиляции только усложняет задачу.

Чтобы предоставить интерфейс более высокого уровня, изолирующий клиен- та от этих классов, в подсистему компиляции включен также класс Compiler (компилятор). Он определяет унифицированный интерфейс ко всем возможнос- тям компилятора. Класс Compiler выступает в роли фасада: предлагает простой интерфейс к более сложной подсистеме. Он «склеивает» классы, реализующие функциональность компилятора, но не скрывает их полностью. Благодаря фаса- ду компилятора работа большинства программистов облегчается. При этом те, кому нужен доступ к средствам низкого уровня, не лишаются его.

2.4 Применимость

Используйте паттерн фасад, когда:

хотите предоставить простой интерфейс к сложной подсистеме. Часто подсис- темы усложняются по мере развития. Применение большин-

ства паттернов приводит к появлению меньших классов, но в большем количестве. Такую подсистему проще повторно использовать и настраивать под конкретные нужды, но вместе с тем применять подсистему без настройки становится труднее. Фасад предлагает некоторый вид системы по умолчанию, устраи- вающий большинство клиентов. И лишь те объекты, которым нужны более широкие возможности настройки, могут обратиться напрямую к тому, что находится за фасадом;

между клиентами и классами реализации абстракции существует много за- висимостей. Фасад позволит отделить подсистему как от клиентов, так и от других подсистем, что, в свою очередь, способствует повышению степе- ни независимости и переносимости;

вы хотите разложить подсистему на отдельные слои. Используйте фасад для определения точки входа на каждый уровень подсистемы. Если подсисте- мы зависят друг от друга, то зависимость можно упростить, разрешив под- системам обмениваться информацией только через фасады.

2.5 Структура

2.5.1 Участники

1. Facade (Compiler) - фасад: - «знает», каким классам подсистемы адресовать запрос; - делегирует запросы клиентов подходящим объектам внутри подсистемы;

2. Классы подсистемы (Scanner, Parser, ProgramNode и т.д.): - реализуют функциональность подсистемы; - выполняют работу, порученную объектом Facade; - ничего не «знают» о существовании фасада, то есть не хранят ссылок на него.

2.5.2 Отношения

Клиенты общаются с подсистемой, посылая запросы фасаду. Он переадресует их подходящим объектам внутри подсистемы. Хотя основную работу выполня- ют именно объекты подсистемы, фасаду, возможно, придется преобразовать свой интерфейс в интерфейсы подсистемы. Клиенты, пользующиеся фасадом, не имеют прямого доступа к объектам под- системы.

2.6 Результаты

У паттерна фасад есть следующие преимущества:

изолирует клиентов от компонентов подсистемы, уменьшая тем самым чис- ло объектов, с которыми клиентам приходится иметь дело, и упрощая рабо- ту с подсистемой;

позволяет ослабить связанность между подсистемой и ее клиентами. Зачас- тую компоненты подсистемы сильно связаны. Слабая связанность позволя- ет видоизменять компоненты, не затрагивая при этом клиентов. Фасадь: помогают разложить систему на слои и структурировать зависимости между объектами, а также избежать сложных и циклических зависимостей. Это мо- жет оказаться важным, если клиент и подсистема реализуются независимо Уменьшение числа зависимостей на стадии компиляции чрезвычайно важ- но в больших системах. Хочется, конечно, чтобы время, уходящее на пере- компиляцию после изменения классов подсистемы, было минимальным Сокращение числа зависимостей за счет фасадов может уменьшить количе- ство нуждающихся в повторной компиляции файлов после небольшой моди- фикации какой-нибудь важной подсистемы. Фасад может также упростить процесс переноса системы на другие платформы, поскольку уменьшается ве- роятность того, что в результате изменения одной подсистемы понадобится изменять и все остальные;

фасад не препятствует приложениям напрямую обращаться к классам под- системы, если это необходимо. Таким образом, у вас есть выбор между прос- тотой и общностью.

3 Подход BCE (Boundary – Control – Entity)

Подход BCE (Boundary-Control-Entity – граница-управление-сущность) представляет собой подход к объектному моделированию, основанный на трехфакторном представлении классов.

В правильно спроектированной иерархии пакетов актер может взаимодействовать только с пограничными объектами из пакета BoundaryPackage, объекты-сущности из пакета EntityPackage могут взаимодействовать только с управляющими объектами из ControlPackage и управляющие объекты из ControlPackage могут взаимодействовать с объектами любого типа.

Основным преимуществом подхода BCE является группирование классов в виде иерархических уровней. Это способствует лучшему пониманию модели и уменьшает ее сложность.

В языке UML для классов определены 3 основных стереотипа:

Boundary - Пограничные классы, классы, которые представляют интерфейс между субъектом и системой.

Control - управляющие классы, описывают объект, который перехва-

тывает входные события инициированные пользователем и контролирует выполнение бизнес-процессов.

Entity - сущности, которые описывают семантику сущностей. Для каждого класса-сущности создают таблицу в БД, каждый атрибут становится полем БД.

4 Выбор STL контейнера

Я проанализировал варианты шаблонов, которые предоставляет STL, и самым удобным из всех посчитал std::vector. Работая с ним, не приходится думать о правильном выделении памяти и прочих технических трудностях. К тому же синтаксис работы с векторами очень похож на синтаксис массивов, что опять же упрощает жизнь.