Java Design Patterns

Visitor

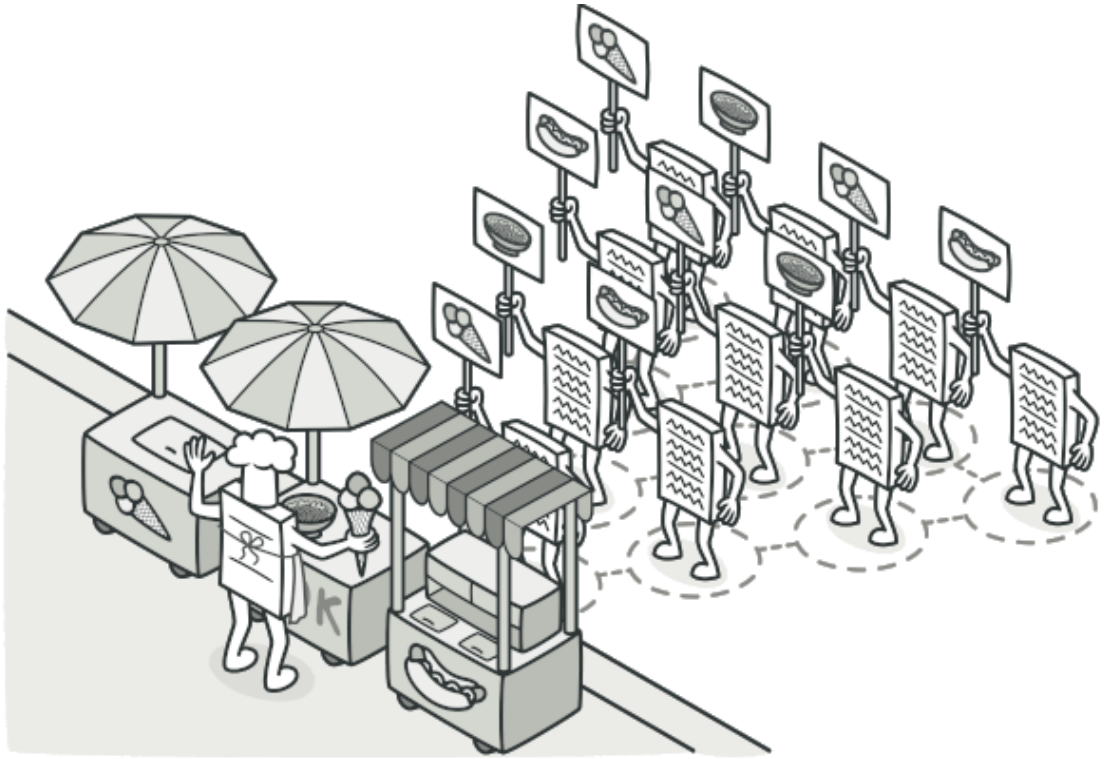
Тема

Visitor

Суть паттерна

Посетитель

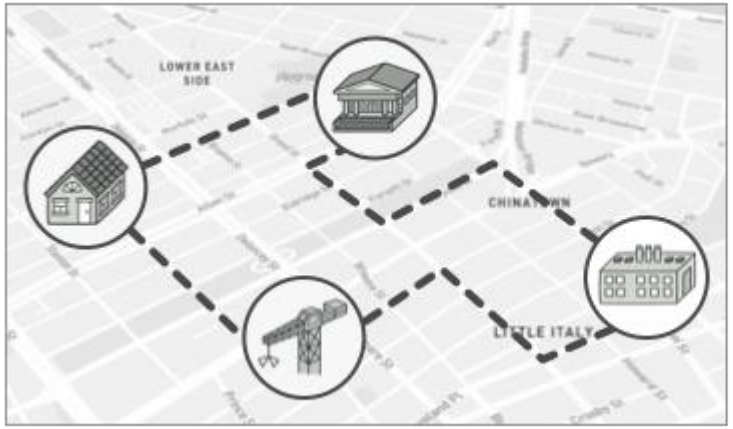
Посетитель — это поведенческий паттерн проектирования, который позволяет создавать новые операции, не меняя классы объектов, над которыми эти операции могут выполняться.

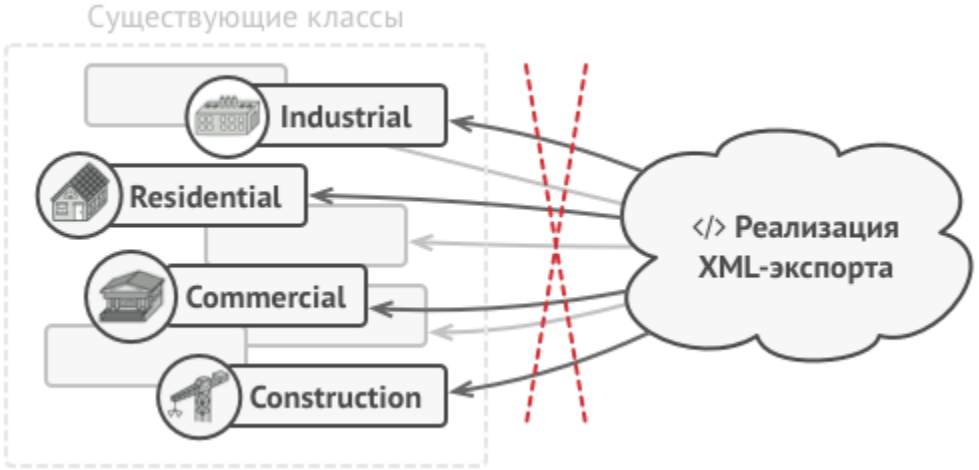


Проблема

Постановка задачи

Ваша команда разрабатывает приложение, работающее с геоданными в виде графа. Узлами графа могут быть как города, так и другие локации, будь то достопримечательности, большие предприятия и так далее. Каждый узел имеет ссылки на другие, ближайшие к нему узлы. Для каждого типа узла имеется свой класс, а каждый узел представлен отдельным объектом.





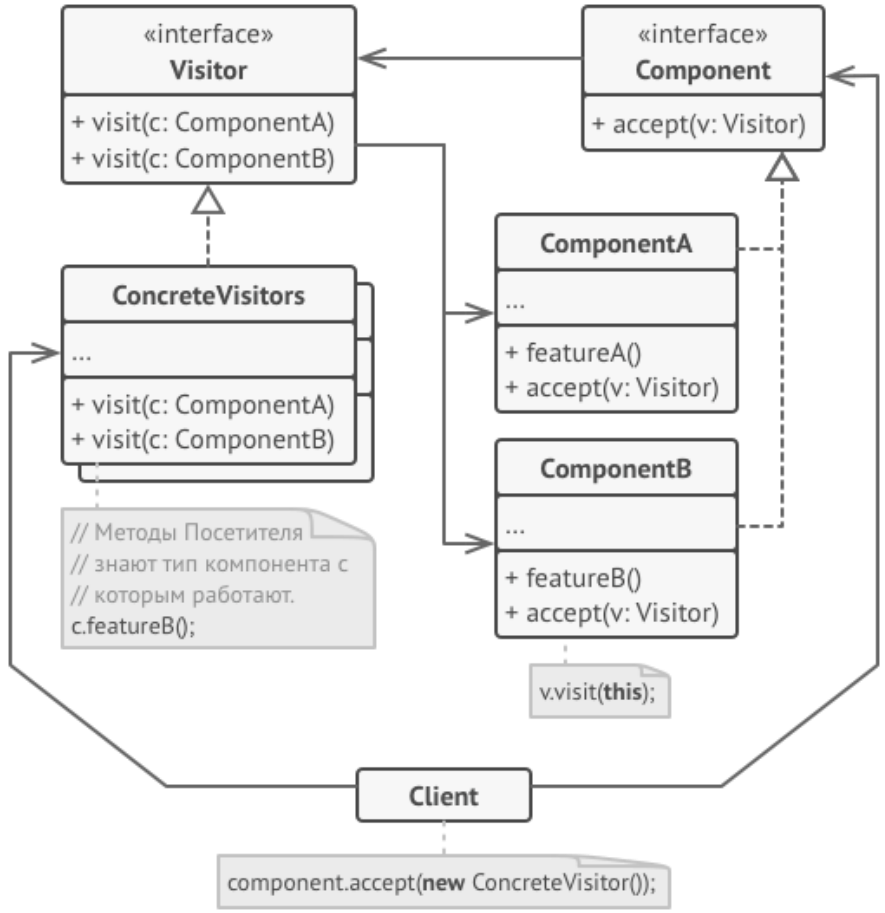
Решение задачи

Паттерн Посетитель предлагает разместить новое поведение в отдельном классе, вместо того, чтобы множить его сразу в нескольких классах. Объекты, с которыми должно было быть связано поведение, не будут выполнять его самостоятельно. Вместо этого, вы будете передавать эти объекты в методы посетителя.

Структура

Структура паттерна

1. Посетитель описывает общий интерфейс для всех типов посетителей.
2. Конкретные посетители реализуют какое-то особенное поведение для всех типов компонентов, которые можно подать через методы интерфейс посетителя.
3. Компонент описывает метод *принятия* посетителя. Этот метод должен иметь единственный параметр, объявленный с типом интерфейса посетителя.
4. Конкретные компоненты реализуют методы *принятия* посетителя. Цель этого метода — вызвать тот метод посещения, который соответствует типу этого компонента. Так посетитель узнает, с каким именно компонентом он работает.
5. Клиентом зачастую выступает коллекция или сложный составной объект (например, дерево Компоновщика). Клиент не знает конкретные классы своих компонентов.



Применимость

Применение паттерна

1. Когда вам нужно выполнить операцию над всеми элементами сложной структуры объектов (например, деревом).
2. Когда над объектами сложной структуры объектов надо выполнять некоторые, не связанные между собой операций, но вы не хотите «засорять» классы такими операциями.
3. Когда новое поведение имеет смысл только для некоторых классов из существующей иерархии.

Шаги реализации

Алгоритм реализации паттерна

1. Создайте интерфейс посетителя и объявите в нём методы «посещения» для каждого класса компонента, который существует в программе.
2. Опишите интерфейс компонентов. Если вы работаете с уже существующими классами, то объявите абстрактный метод принятия посетителей в базовом классе иерархии компонентов.
3. Реализуйте методы принятия во всех конкретных компонентах. Они должны переадресовывать вызовы тому методу посетителя, в котором класс параметра совпадает с текущим классом компонента.
4. Иерархия компонентов должна знать только о базовом интерфейсе посетителей. С другой стороны, посетители будут знать обо всех классах компонентов.
5. Для каждого нового поведения создайте свой конкретный класс. Приспособьте это поведение для всех посещаемых компонентов, реализовав все методы интерфейса посетителей.
6. Клиент будет создавать объекты посетителей, а затем передавать их компонентам, используя метод принятия.

Преимущества и недостатки

Плюсы и недостатки

Плюсы:

- Упрощает добавление новых операций над всей связанной структурой объектов.
- Объединяет родственные операции в одном классе.
- Посетитель может накоплять состояние при обходе структуры компонентов.

Минусы:

- Паттерн не оправдан, если иерархия компонентов часто меняется.
- Может привести к нарушению инкапсуляции компонентов.

Отношение с другими паттернами

- Посетитель можно рассматривать как расширенный аналог Команды, который способен работать сразу с несколькими видами получателей.
- Вы можете выполнить какое-то действие над всем деревом Компоновщика при помощи Посетителя.
- Посетитель можно использовать совместно с Итератором. *Итератор* будет отвечать за обход структуры данных, а *Посетитель* — за выполнение действий над каждым её компонентом.

Информационный видеосервис для разработчиков программного обеспечения

ITVDN

IT VIDEO DEVELOPERS NETWORK

TestProvider PROMETRIC TEST CENTER CyberBionic systematics Microsoft Partner Silver Learning Windows Azure Cloud Partner EBA European Business Association