Java Design Patterns

Adapter

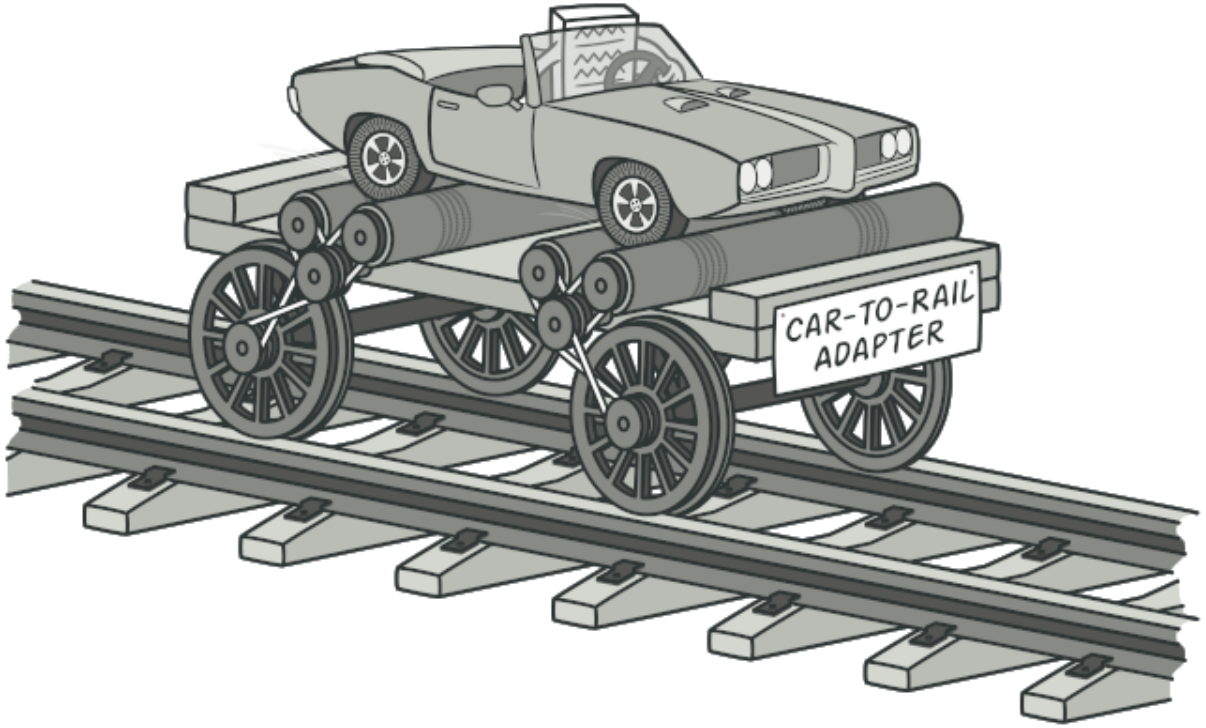
Тема

Adapter

Суть паттерна

Адаптер

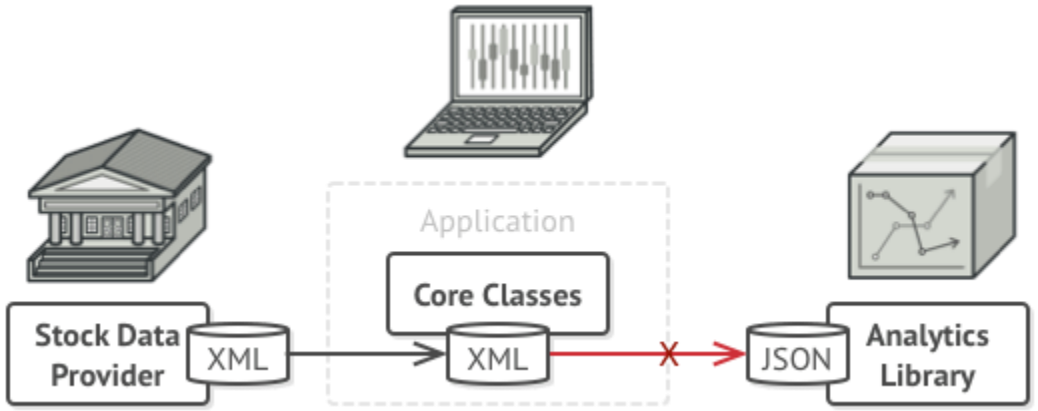
Адаптер — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.



Проблема

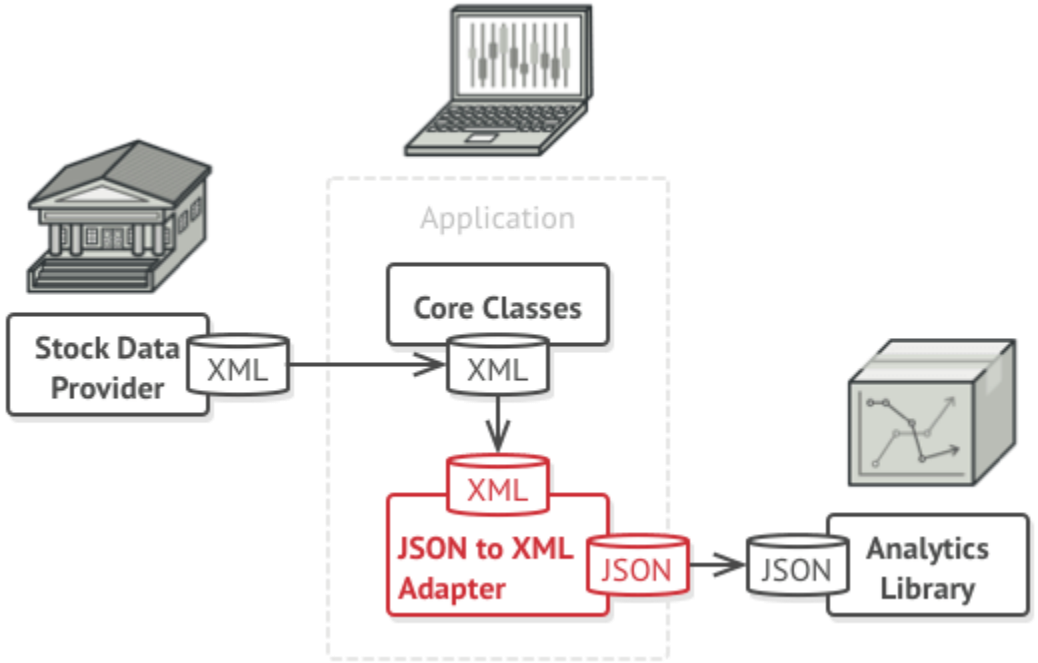
Постановка задачи

Представьте, что вы делаете приложение для торговли на бирже. Ваше приложение скачивает биржевые котировки из нескольких источников в XML, а затем рисует красивые графики.



Решение задачи

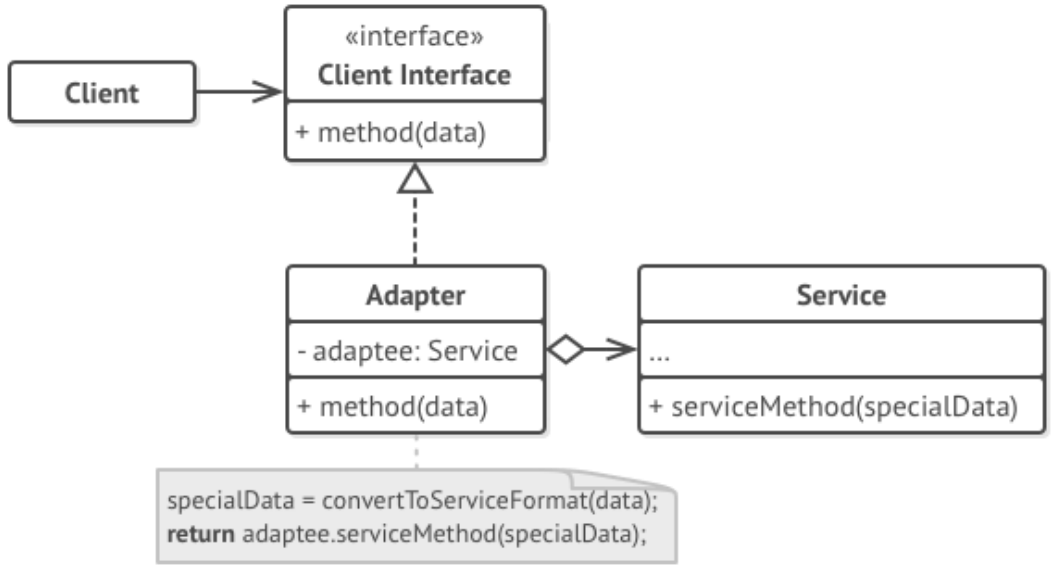
Вы можете создать *адаптер*. Это объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту. При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о его наличии. Например, вы можете обернуть объект, работающий в метрах, адаптером, который бы конвертировал данные в футы.



Структура

Структура паттерна

1. Клиент — это основной код программы.
2. Клиентский интерфейс — это интерфейс, через который клиент может работать с другими классами.
3. Сервис – это какой-то полезный класс, обычно сторонний. Клиент не может использовать этот класс напрямую, так как сервис он не следует существующему интерфейсу.
4. Адаптер — это класс, который реализует существующий интерфейс и содержит ссылку на объект сервиса.
5. Работая с адаптером через интерфейс, клиент не привязываться к конкретному классу адаптера.



Применимость

Применение паттерна

1. Когда вы хотите использовать сторонний класс, но его интерфейс не соответствует остальному коду приложения.
2. Когда вам нужно использовать несколько существующих подклассов, но в них не хватает какой-то общей функциональности. Причём расширять суперкласс вы не можете.

Шаги реализации

Алгоритм реализации паттерна

1. Убедитесь, что у вас есть два класса с неудобными интерфейсами:
   - полезный *сервис* — служебный класс, который вы не можете изменять (он либо сторонний, либо от него зависит другой код);
   - один или несколько *клиентов* — классов приложения, несовместимых с сервисом из-за неудобного или несовпадающего интерфейса.
2. Опишите клиентский интерфейс, через который классы приложения смогли бы использовать сторонний класс.
3. Создайте класс адаптера, реализовав этот интерфейс.
4. Поместите в адаптер поле-ссылку на объект-сервис. В большинстве случаев, это поле заполняется объектом, переданным в конструктор адаптера. В случае простой адаптации этот объект можно передавать через параметры методов адаптера.
5. Реализуйте все методы клиентского интерфейса в адаптере. Адаптер должен делегировать основную работу сервису.
6. Приложение должно использовать адаптер только через клиентский интерфейс. Это позволит легко изменять и добавлять адаптеры в будущем.

Преимущества и недостатки

Плюсы и недостатки

Плюсы:

- Отделяет и скрывает от клиента подробности преобразования различных интерфейсов.

Минусы:

- Усложняет код программы за счёт дополнительных классов.

Отношение с другими паттернами

- Мост проектируют чтобы развивать большие части приложения отдельно друг от друга. Адаптер применяется постфактум, чтобы заставить несовместимые классы работать вместе.
- Адаптер меняет интерфейс существующего объекта. Декоратор улучшает другой объект без изменения его интерфейса. Причём *Декоратор* поддерживает рекурсивную вложенность, чего не скажешь об *Адаптере*.
- Адаптер предоставляет классу альтернативный интерфейс. Декоратор предоставляет расширенный интерфейс. Заместитель предоставляет тот же интерфейс.
- Фасад задаёт новый интерфейс, тогда как Адаптер повторно использует старый. *Адаптер* оборачивает только один класс, а *Фасад* оборачивает целую подсистему. Кроме того, *Адаптер* позволяет двум существующим интерфейсам работать сообща, вместо того, чтобы задать полностью новый.
- Мост, Стратегия и Состояние (а также слегка и Адаптер) имеют схожие структуры классов — все они построены на принципе «композиции», то есть делегирования работы другим объектам. Тем не менее, они отличаются тем, что решают разные проблемы. Помните, что паттерны — это не только рецепт построения кода определённым образом, но и описание проблем, которые привели к данному решению.

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