Задача: Универсальная Система Отчетов для Различных Источников Данных

Цель

Вам нужно создать систему, которая может генерировать унифицированные отчеты, получая данные из **различных внешних сервисов** или **библиотек**, которые имеют совершенно **разные API и структуры данных**. Вы будете использовать паттерн **Адаптер** для того, чтобы "обернуть" эти несовместимые источники и привести их к единому, ожидаемому вашей системой **интерфейсу отчета**.

Сценарий

Представьте, что вы разрабатываете внутренний инструмент для компании, который должен консолидировать данные из трех условных источников:

1. **Старый Сервис Учета (Legacy System):** Возвращает данные в формате **XML** с устаревшими именами полей.
2. **Современный Аналитический Сервис (Modern API):** Возвращает данные в формате **JSON** с другими именами полей.
3. **Локальная Библиотека Файлов (File Reader Library):** Возвращает данные в виде **списка объектов** (или словарей), но требует специфического метода инициализации.

Ваша **Система Отчетов** (Клиентский код) должна работать только с **одним унифицированным интерфейсом** (например, DataSource), не зная, откуда на самом деле приходят данные.

Требования к Коду и Этапы Разработки

1. Единый Целевой Интерфейс (Target)

Определите интерфейс, который ваша **Система Отчетов** ожидает от любого источника данных.

- Интерфейс DataSource должен иметь метод:

    - get_report_data(fields: list) -> list[dict]: Возвращает список словарей, где каждый словарь представляет собой унифицированную запись (например, с ключами: id, name, value, date).

2. Несовместимые Сторонние Сервисы (Adaptees)

Создайте три класса, имитирующих работу внешних, несовместимых источников:

- **LegacyXmlService**:
    - Имеет метод fetch_old_data(): возвращает строку-XML.
    - **Пример данных (внутри строки XML):** Поля могут называться record_id, client_name, amount_val, trans_date.
- **ModernJsonAPI**:
    - Имеет метод get_current_metrics(): возвращает строку-JSON.
    - **Пример данных (внутри строки JSON):** Поля могут называться entity_uuid, user_handle, metric_total, timestamp.
- **LocalFileReader**:
    - Имеет метод read_records(): возвращает список словарей.
    - **Пример данных:** Поля могут называться record_identifier, full_name, count, creation_time.

3. Классы Адаптеров (Adapters)

Создайте по одному классу Адаптера для каждого несовместимого источника. Каждый Адаптер должен:

- Реализовывать **целевой интерфейс** DataSource.
- **Содержать экземпляр** соответствующего **Несовместимого Сервиса** (Adaptee).
- В методе get_report_data() **вызывать специфический метод** Adaptee'a (например, fetch_old_data()).

- **Выполнять преобразование данных** (например, парсинг XML/JSON, переименование полей) из формата Adaptee'a в **унифицированный формат** (т.е. с ключами: id, name, value, date).

4. Клиентский Код (Reporting System)

Создайте класс или функцию, которая принимает объект, реализующий DataSource, и выводит отчет.

- ReportingSystem.generate_report(data_source: DataSource): Получает данные через унифицированный интерфейс и просто выводит их на экран, демонстрируя, что все источники выглядят одинаково для системы.