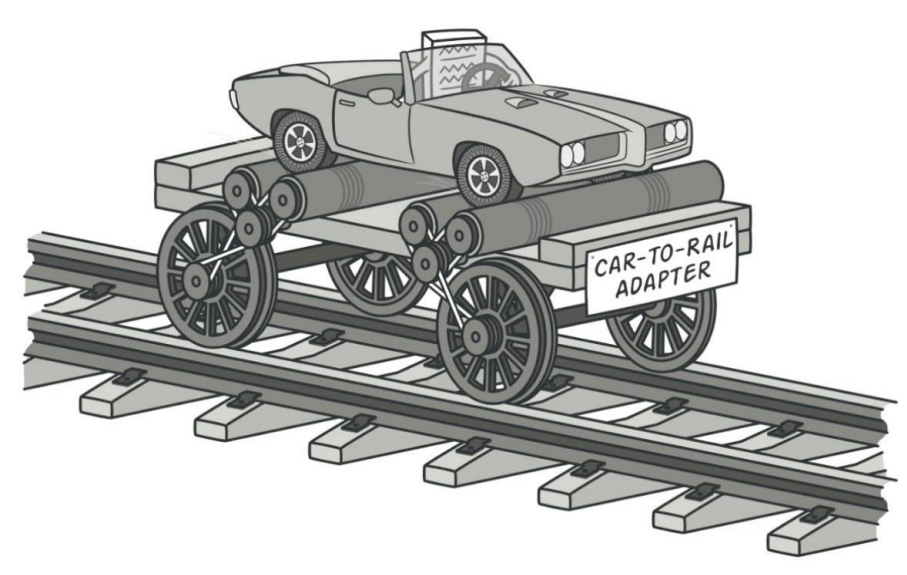
Паттерн “Адаптер”

Выполнила: Тарасова Даша, 2 курс, 1 группа

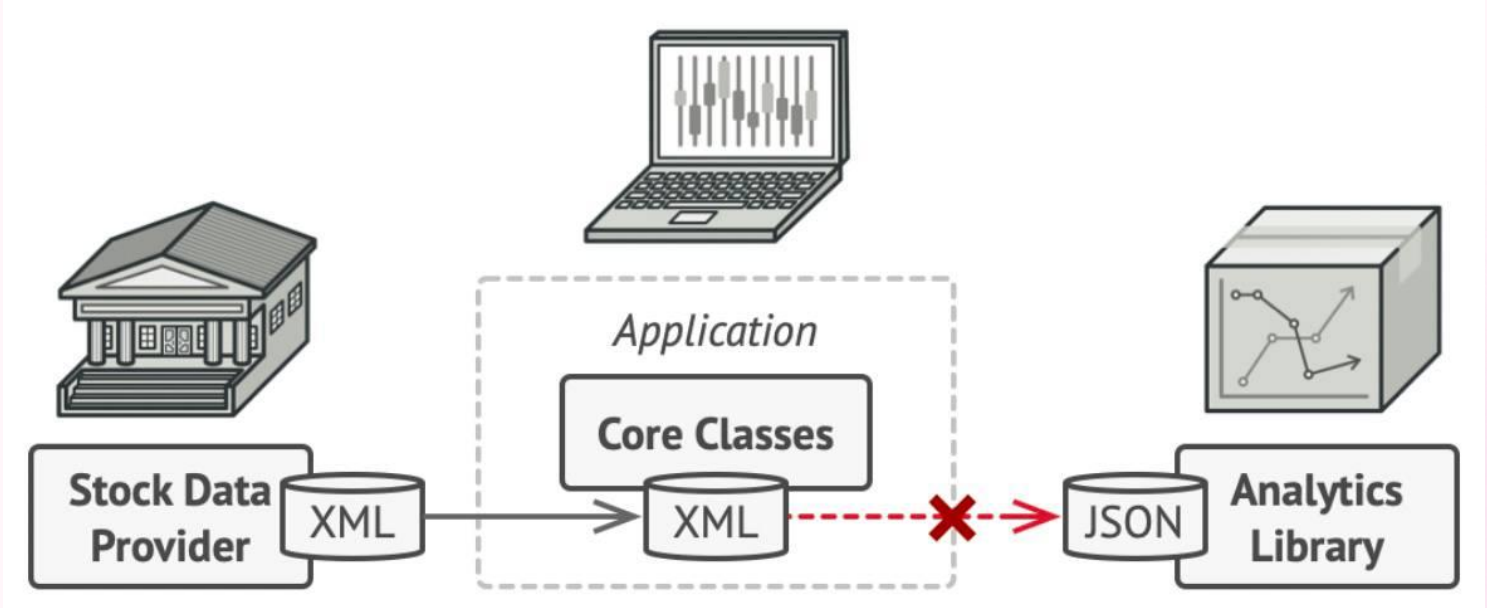
Определение

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

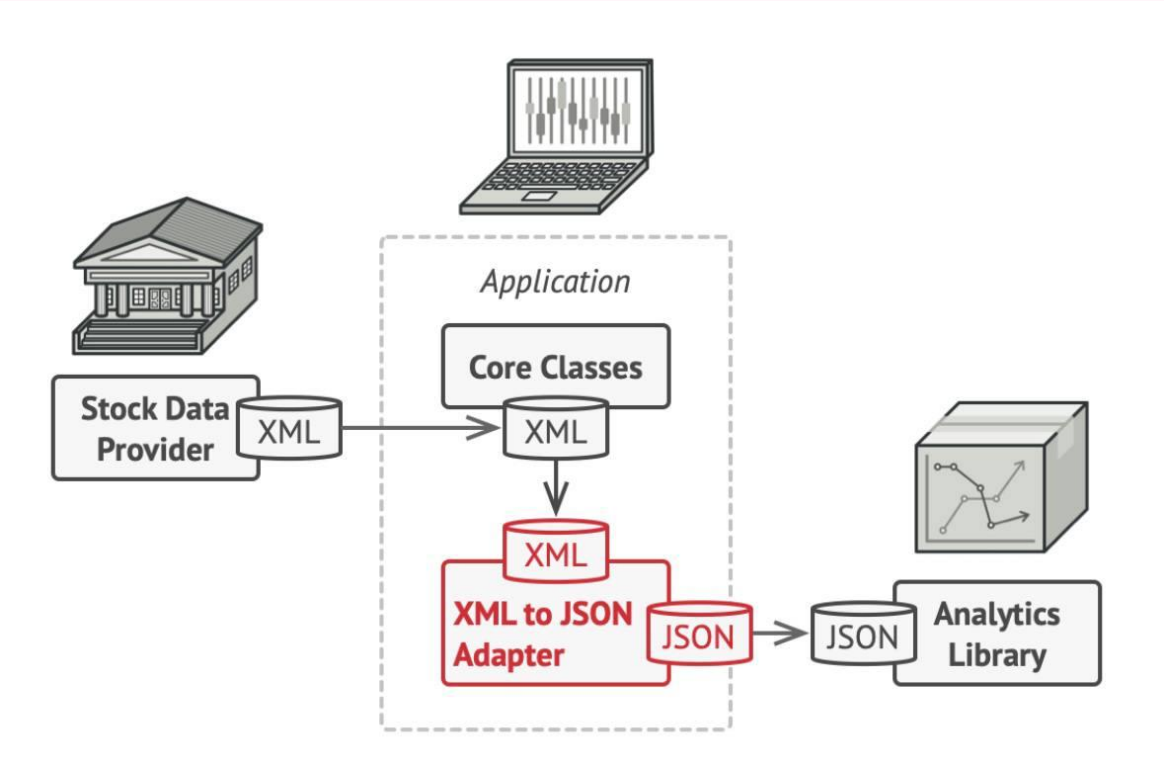


Аналогия из жизни

Пример


Application
Core Classes
Stock Data Provider
XML
XML
JSON
Analytics Library

Пример


Application
Core Classes
Stock Data Provider
XML
XML
XML
XML to JSON Adapter
JSON
JSON
Analytics Library

Листинг кода без паттерна

```python
import xml.etree.ElementTree as ET
import json

class XmlDataProvider:  1 usage
    def get_data(self):  2 usages (2 dynamic)
        return "<stock><symbol>AAPL</symbol><price>170</price></stock>"

class JsonAnalyzer:  1 usage
    def process_data(self, json_data):  2 usages (2 dynamic)
        print(f"Analyzing: {json_data}")

class TradingApp:  1 usage
    def __init__(self, data_provider, analyzer):
        self.data_provider = data_provider
        self.analyzer = analyzer

    def run(self):  1 usage
        xml_data = self.data_provider.get_data()
        try:
            root = ET.fromstring(xml_data)
            data = {"symbol": root.find("symbol").text, "price": root.find("price").text}
            json_data = json.dumps(data)
            self.analyzer.process_data(json_data)
        except Exception as e:
            print(f"Error: {e}")

# Использование
provider = XmlDataProvider()
analyzer = JsonAnalyzer()
app = TradingApp(provider, analyzer)
app.run()
```

Листинг кода с паттерном

```python
import xml.etree.ElementTree as ET
import json

class XmlDataProvider:  1 usage
    def get_data(self):  2 usages (2 dynamic)
        return "<stock><symbol>AAPL</symbol><price>170</price></stock>"

class JsonAnalyzer:  1 usage
    def process_data(self, json_data):  2 usages (2 dynamic)
        print(f"Analyzing: {json_data}")

class DataAdapterInterface:  2 usages
    def get_json_data(self):  1 usage
        pass

class XmlToJsonAdapter(DataAdapterInterface):  1 usage
    def __init__(self, xml_provider):
        self.xml_provider = xml_provider

    def get_json_data(self):
        xml_data = self.xml_provider.get_data()
        try:
            root = ET.fromstring(xml_data)
            data = {"symbol": root.find("symbol").text, "price": root.find("price").text}
            return json.dumps(data)
        except Exception as e:
            print(f"Error: {e}")
            return None
```

```python
class TradingApp:  1 usage
    def __init__(self, data_provider: DataAdapterInterface, analyzer):
        self.data_provider = data_provider
        self.analyzer = analyzer

    def run(self):  1 usage
        json_data = self.data_provider.get_json_data()
        if json_data:
            self.analyzer.process_data(json_data)

# Использование
provider = XmlDataProvider()
adapter = XmlToJsonAdapter(provider)
analyzer = JsonAnalyzer()
app = TradingApp(adapter, analyzer)
app.run()
```

Преимущества

- Разделение ответственности
- Гибкость
- Независимость
- Упрощение кода
- Повторное использование кода
- Тестируемость

Источники

Refactoring.Guru. Паттерн "Мост" [Электронный ресурс] // Refactoring.Guru. – URL: https://refactoring.guru/ru/design-patterns/bridge

Спасибо за внимание!