GoF · 06/23

СТРУКТУРНЫЙ

Python C++23 Rust Java



Адаптер

Adapter

Алексей Примак

Алексей Примак

Адаптер. Паттерн GoF на Python, C++23, Rust и Java

https://litres.ru/74810753
SelfPub; 2026

Аннотация

Как соединить два несовместимых интерфейса, не переписывая существующую систему? «Адаптер» показывает решение на примере интеграции платёжного модуля и сторонней библиотеки. Паттерн GoF разобран от общей схемы до практических реализаций на Python, C++23, Rust и Java. В книге рассматриваются разные варианты адаптации, модификации, гибридные решения и самостоятельные задания.

Содержание

Введение 4
Как читать 5
Глава 6. Реализация на Python 6
6.1. Зачем нужен адаптер 6
6.2. Участники и схема 7
6.3. Классический пример: устаревшая платёжная система 10
Конец ознакомительного фрагмента. 15

Адаптер. Паттерн GoF на Python, C++23, Rust и Java

Введение

Эта книга посвящена одному паттерну проектирования — **Адаптер**(Adapter). Он относится к каталогу «Банды четырёх» (Gang of Four, GoF). Здесь паттерн разобран сразу в четырёх реализациях: на Python, C++23, Rust и Java. Каждая реализация — отдельная часть книги: сначала задача и общая схема, затем код, модификации, гибридные варианты и самостоятельные работы.

Как читать

Основы каждого языка здесь не повторяются: они собраны в книгах серий по соответствующему языку. Разделы каждой части нумеруются заново, поэтому ссылка «раздел 1.2» указывает на раздел той же части, где она сделана.

Глава 6. Реализация на Python

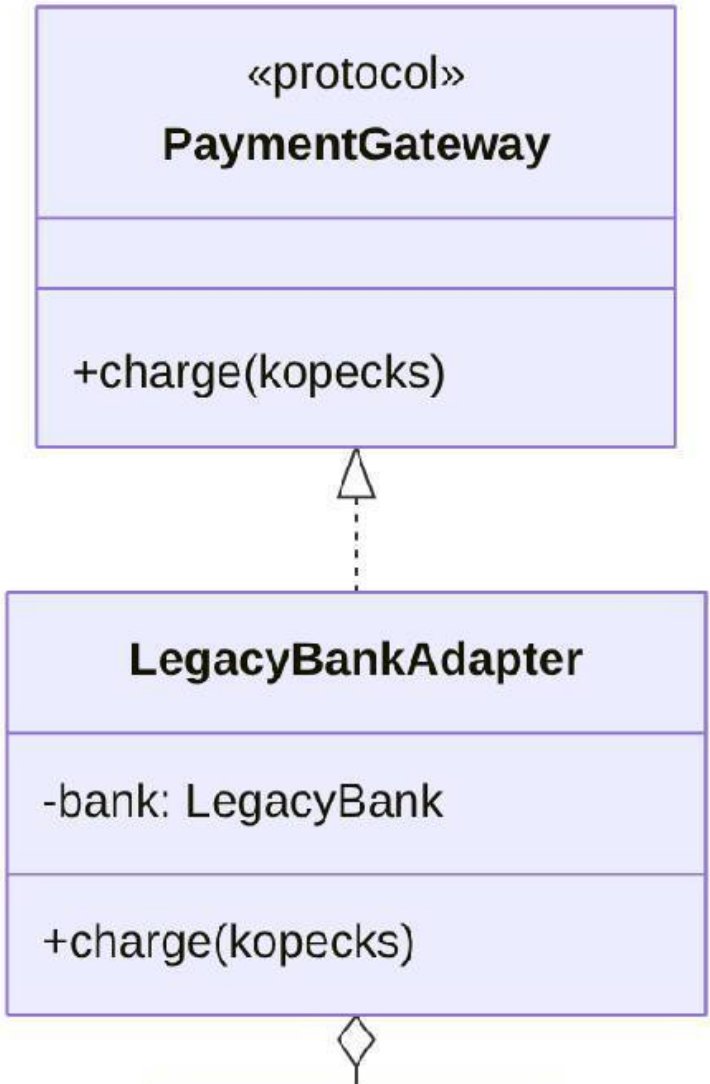
6.1. Зачем нужен адаптер

Платёжный модуль приложения работает с интерфейсом charge(amount_kopecks) и исключением PaymentError. Новая платёжная библиотека принимает сумму в рублях с плавающей точкой и возвращает числовой код ошибки. Менять библиотеку нельзя, а переделывать под неё всё приложение — плохо.

Адаптер — промежуточный объект, который реализует ожидаемый интерфейс и переводит вызовы в формат адаптируемого компонента. Клиент работает с привычным интерфейсом, компонент не меняется, а вся «перекодировка» сосредоточена в одном месте.

В Python адаптеры повсюду: io.TextIOWrapper адаптирует поток байтов к потоку строк, csv.DictReader — строки CSV к словарям, functools.partial — функцию с одним набором аргументов к другому. Утиная типизация делает паттерн особенно лёгким: адаптеру не нужно наследоваться от интерфейса, достаточно иметь нужные методы.

6.2. Участники и схема


«protocol»
PaymentGateway
+charge(kopecks)
LegacyBankAdapter
-bank: LegacyBank
+charge(kopecks)

Target— интерфейс, который ожидает клиент.

Adaptee— существующий компонент с несовместимым интерфейсом.

Adapter— реализует Target и обращается к Adaptee.

GoF различает **объектный**адаптер (хранит адаптируемый объект) и **классовый**(наследуется от него). В Python возможны оба, но объектный обычно предпочтительнее: он не смешивает интерфейсы двух классов.

6.3. Классический пример: устаревшая платёжная система

Файл 01_object_adapter.py:

01_object_adapter.py Python

```python
from typing import Protocol


# --- Чужая библиотека, менять нельзя ---
class LegacyBank:
    def __init__(self, balance_rub: float) -> None:
        self.balance_rub = balance_rub

    def pay_rub(self, amount: float) -> int:
        """Код: 0 — успех, 1 — мало денег, 2 — неверная
            сумма."""
        if amount <= 0:
            return 2
        if amount > self.balance_rub:
            return 1
        self.balance_rub -= amount
        return 0


# --- Наше приложение ---
class PaymentError(Exception):
    pass


class InsufficientFunds(PaymentError):
    pass


class PaymentGateway(Protocol):
    def charge(self, kopecks: int) -> None: ...


class LegacyBankAdapter:
    _errors = {1: InsufficientFunds("недостаточно средств"), 2:
```

Листинг 6.1 *— Классический пример: устаревшая пла-*

тёжная система

01_object_adapter.py (продолжение) Python

```
        PaymentError("неверная сумма")}

    def __init__(self, bank: LegacyBank) -> None:
        self._bank = bank

    def charge(self, kopecks: int) -> None:
        code = self._bank.pay_rub(kopecks / 100)
        if code != 0:
            raise self._errors.get(code, PaymentError(f"
                неизвестный код {code}"))


def checkout(gateway: PaymentGateway, items: list[tuple[str, int
    ]]) -> None:
    for name, price in items:
        try:
            gateway.charge(price)
            print(f"{name}: оплачено {price // 100}.{price % 100
                :02} ₽")
        except InsufficientFunds:
            print(f"{name}: отказ — не хватает денег")
        except PaymentError as e:
            print(f"{name}: отказ — {e}")


if __name__ == "__main__":
    bank = LegacyBank(500)
    checkout(LegacyBankAdapter(bank), [("книга", 34_950), ("
        ручка", 0), ("ноутбук", 9_999_900), ("кофе", 15_000)])
    print(f"Остаток: {bank.balance_rub:.2f} ₽")
```

```
$ python 01_object_adapter.py
книга: оплачено 349.50 ₽
ручка: отказ — неверная сумма
ноутбук: отказ — не хватает денег
кофе: оплачено 150.00 ₽
Остаток: 0.50 ₽
```

Листинг 6.1 (продолжение) *— Классический пример: устаревшая платёжная система*

Разбор. Адаптер переводит три вещи: единицы (копейки в рубли), типы (int в float) и способ сообщать об ошибках (числовые коды в исключения). Иерархия исключений позволяет клиенту ловить как конкретную ошибку (InsufficientFunds), так и любую платёжную (PaymentError). Неизвестные коды тоже превращаются в исключение — будущие версии библиотеки не сломают приложение молча.

checkout ничего не знает о библиотеке и её кодах. Если банк сменится, понадобится новый адаптер, а checkout останется прежним.

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.