GoF · 22/23

ПОВЕДЕНЧЕСКИЙ

Python C++23 Rust Java



Шаблонный метод

Template Method

Алексей Примак

Алексей Примак

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

https://litres.ru/74810683
SelfPub; 2026

Аннотация

Если несколько алгоритмов имеют один и тот же общий порядок действий, не обязательно копировать этот порядок в каждом классе. «Шаблонный метод» показывает, как задать каркас алгоритма и оставить отдельные шаги подклассам. На примере импорта данных из CSV, JSON и Excel разобраны преимущества такого подхода. Паттерн GoF реализован на Python, C++23, Rust и Java, с модификациями и самостоятельными работами.

Содержание

Введение 4
Как читать 5
Глава 22. Реализация на Python 6
22.1. Зачем нужен шаблонный метод 6
22.2. Участники и схема 7
22.3. Классический пример: импорт данных 9
Конец ознакомительного фрагмента. 13

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

Введение

Эта книга посвящена одному паттерну проектирования — **Шаблонный метод**(Template Method). 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 22. Реализация на Python

22.1. Зачем нужен шаблонный метод

Импорт данных из CSV, JSON и Excel устроен одинаково: открыть источник, прочитать записи, проверить их, сохранить, сформировать отчёт. Различаются только чтение и, может быть, проверка. Если писать три независимых импортёра, общий порядок шагов будет скопирован трижды, и исправление в нём придётся вносить везде.

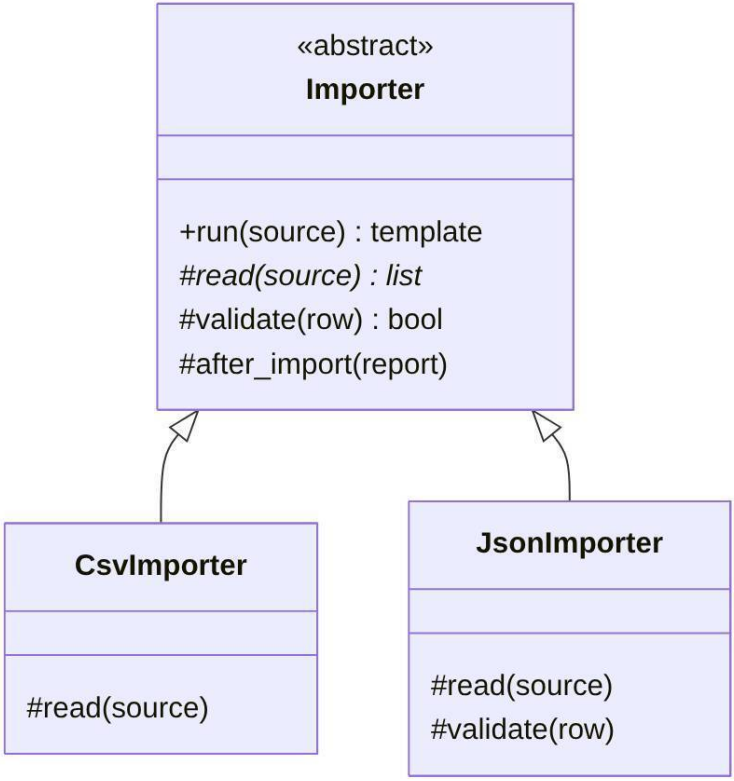
Шаблонный метод задаёт скелет алгоритма в базовом классе, а отдельные шаги оставляет подклассам. Порядок шагов зафиксирован в одном месте, а подклассы переопределяют только то, что отличается. Шаги бывают:

абстрактные — подкласс обязан их реализовать;

хуки — имеют реализацию по умолчанию (часто пустую), подкласс может переопределить их по желанию.

Шаблонный метод основан на принципе «не звоните нам, мы позвоним вам»: базовый класс управляет ходом работы и сам вызывает методы подкласса. Так устроены unittest.TestCase (setUp → тест → tearDown), threading.Thread.run, socketserver.BaseRequestHandler и многие фреймворки.

22.2. Участники и схема



Abstract Class— содержит шаблонный метод и объявляет шаги.

Concrete Class— реализует шаги.

22.3. Классический пример: импорт данных

Файл 01_importers.py:

01_importers.py Python

```python
import csv
import io
import json
from abc import ABC, abstractmethod


class Importer(ABC):
    def run(self, source: str) -> str:
        """Шаблонный метод: порядок шагов не меняется."""
        rows = self.read(source)
        good = [self.normalize(r) for r in rows if self.validate(r)]
        saved = self.save(good)
        report = f"{type(self).__name__}: прочитано {len(rows)}, сохранено {saved}"
        self.after_import(report)
        return report

    @abstractmethod
    def read(self, source: str) -> list[dict]: ...

    def validate(self, row: dict) -> bool:  # хук с поведением по умолчанию
        return bool(row.get("email"))

    def normalize(self, row: dict) -> dict:
        return {"email": row["email"].strip().lower(), "name": row.get("name", "").strip()}

    def save(self, rows: list[dict]) -> int:
        self.saved = rows
        return len(rows)

    def after_import(self, report: str) -> None:  # пустой хук
```

Листинг 22.1 *— Классический пример: импорт данных*

01_importers.py (продолжение) Python

```python
        pass


class CsvImporter(Importer):
    def read(self, source):
        return list(csv.DictReader(io.StringIO(source)))


class JsonImporter(Importer):
    def read(self, source):
        return json.loads(source)["users"]

    def validate(self, row):
        return super().validate(row) and row.get("active", True)

    def after_import(self, report):
        print("  [уведомление в Slack]", report)


if __name__ == "__main__":
    csv_data = "name,email\nАнна, Anna@Mail.ru \nБорис,\n
        Вера,vera@ya.ru\n"
    json_data = '{"users": [{"name": "Глеб", "email": "gleb@x.ru
        "}, {"name": "Дина", "email": "d@x.ru", "active"
        : false}]}'
    print(CsvImporter().run(csv_data))
    imp = JsonImporter()
    print(imp.run(json_data))
    print("Первый сохранённый:", imp.saved[0])
```

```
$ python 01_importers.py
CsvImporter: прочитано 3, сохранено 2
  [уведомление в Slack] JsonImporter: прочитано 2, сохранено 1
JsonImporter: прочитано 2, сохранено 1
Первый сохранённый: {'email': 'gleb@x.ru', 'name': 'Глеб'}
```

Листинг 22.1 (продолжение) *— Классический пример: импорт данных*

Разбор.run описывает алгоритм целиком: чтение, проверка, нормализация, сохранение, отчёт, хук. Подкласс CSV реализует только обязательное чтение. Подкласс JSON дополнительно уточняет проверку (через super(), сохраняя общее правило) и переопределяет хук уведомления. @abstractmethod не даст создать импортёр, забывший реализовать read.

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.