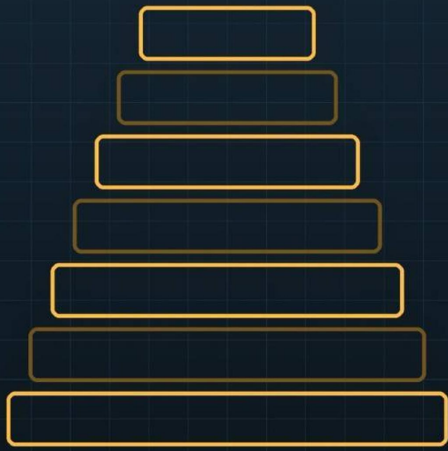
GoF · 02/23 ПОРОЖДАЮЩИЙ

Python C++23 Rust Java



Строитель

Builder

Алексей Примак

Алексей Примак

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

https://litres.ru/74810754
SelfPub; 2026

Аннотация

Объекты со множеством параметров легко превращаются в неудобные конструкторы и наборы условностей. «Строитель» показывает, как отделить процесс пошаговой сборки от готового объекта. Паттерн GoF разобран на практических примерах, а затем реализован на Python, C++23, Rust и Java. В книге есть классический вариант, альтернативы, модификации, гибридные решения и самостоятельные задания.

Содержание

Введение 4
Как читать 5
Глава 2. Реализация на Python 6
2.1. Зачем нужен строитель 6
2.2. Участники и схема 7
2.3. Когда применять и какие последствия 10
2.4. Классическая реализация: директор и два представления 11
Конец ознакомительного фрагмента. 16

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

Введение

Эта книга посвящена одному паттерну проектирования — **Строитель**(Builder). 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.

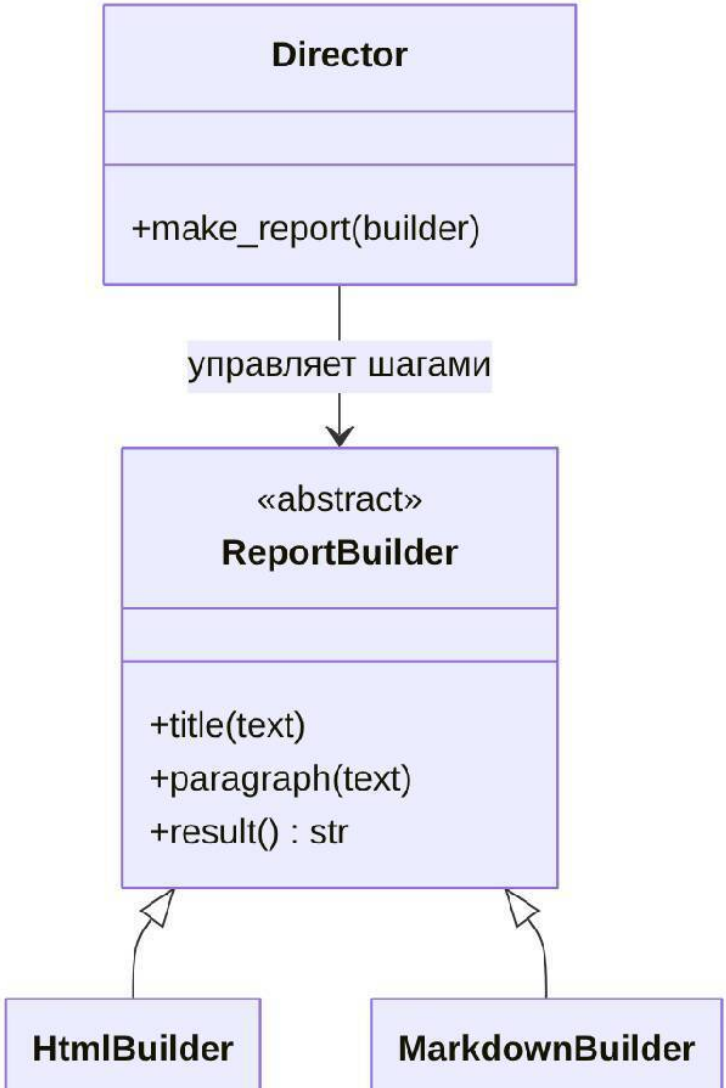
Глава 2. Реализация на Python

2.1. Зачем нужен строитель

HTTP-запрос — это метод, адрес, заголовки, тело, таймаут, число повторов. Большая часть параметров необязательна. В Python эту задачу частично решают именованные аргументы и значения по умолчанию: Request(«GET», url, timeout=5) читается хорошо. Но у них есть пределы. Если объект собирается по шагам (сначала заголовки, потом тело, потом условия), если шаги зависят от условий, если результат нужно проверить целиком или получить в разных представлениях — нужен строитель.

Строитель отделяет **процесс сборки**от **готового объекта**. Вспомогательный объект накапливает параметры, а метод build() проверяет их и возвращает готовый, обычно неизменяемый результат. В стандартной библиотеке Python так устроены, например, argparse.ArgumentParser (по шагам добавляются аргументы, затем разбор), email.message.EmailMessage и построители SQL-запросов в популярных библиотеках.

2.2. Участники и схема


Director
+make_report(builder)
управляет шагами
«abstract»
ReportBuilder
+title(text)
+paragraph(text)
+result() : str
HtmlBuilder
MarkdownBuilder

Builder— интерфейс шагов сборки.

ConcreteBuilder— реализация шагов для одного представления продукта.

Director— необязательный участник, знающий порядок шагов.

Product— готовый объект.

2.3. Когда применять и какие последствия

Строитель уместен, когда объект собирается по шагам, шаги зависят от условий, итог нужно проверить целиком, а готовый объект должен быть неизменяемым — или когда один процесс сборки даёт разные представления (HTML, Markdown, PDF).

Если параметров немного и все известны сразу, строитель не нужен: в Python для этого есть именованные аргументы и dataclass.

2.4. Классическая реализация: директор и два представления

Файл 01_director.py:

01_director.py

```python
from abc import ABC, abstractmethod


class ReportBuilder(ABC):
    @abstractmethod
    def title(self, text: str) -> None: ...

    @abstractmethod
    def paragraph(self, text: str) -> None: ...

    @abstractmethod
    def items(self, values: list[str]) -> None: ...

    @abstractmethod
    def result(self) -> str: ...


class HtmlBuilder(ReportBuilder):
    def __init__(self) -> None:
        self._parts: list[str] = []

    def title(self, text: str) -> None:
        self._parts.append(f"<h1>{text}</h1>")

    def paragraph(self, text: str) -> None:
        self._parts.append(f"<p>{text}</p>")

    def items(self, values: list[str]) -> None:
        self._parts.append("<ul>" + "".join(f"<li>{v}</li>" for
            v in values) + "</ul>")

    def result(self) -> str:
        return "\n".join(self._parts)

```

***Листинг 2.1** — Классическая реализация: директор и два*

представления

01_director.py (продолжение) Python

```python

class MarkdownBuilder(ReportBuilder):
    def __init__(self) -> None:
        self._parts: list[str] = []

    def title(self, text: str) -> None:
        self._parts.append(f"# {text}\n")

    def paragraph(self, text: str) -> None:
        self._parts.append(f"{text}\n")

    def items(self, values: list[str]) -> None:
        self._parts.extend(f"- {v}" for v in values)

    def result(self) -> str:
        return "\n".join(self._parts)


def monthly_report(b: ReportBuilder) -> str:
    """Директор: знает порядок шагов, но не формат."""
    b.title("Отчёт за май")
    b.paragraph("Выручка выросла на 12%.")
    b.items(["Москва", "Казань", "Пермь"])
    return b.result()


if __name__ == "__main__":
    print(monthly_report(HtmlBuilder()))
    print()
    print(monthly_report(MarkdownBuilder()))
```

```
$ python 01_director.py
<h1>Отчёт за май</h1>
<p>Выручка выросла на 12%.</p>
<ul><li>Москва</li><li>Казань</li><li>Пермь</li></ul>

# Отчёт за май

Выручка выросла на 12%.

- Москва
```

Листинг 2.1 (продолжение) *— Классическая реализация: директор и два представления*

Разбор. Директор monthly_report описывает структуру отчёта один раз; строители превращают её в конкретный формат. Чтобы добавить формат PDF, нужен один новый строитель, а директор не меняется. Внутреннее состояние строителя — список _parts; подчёркивание в имени говорит, что это деталь реализации.

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.