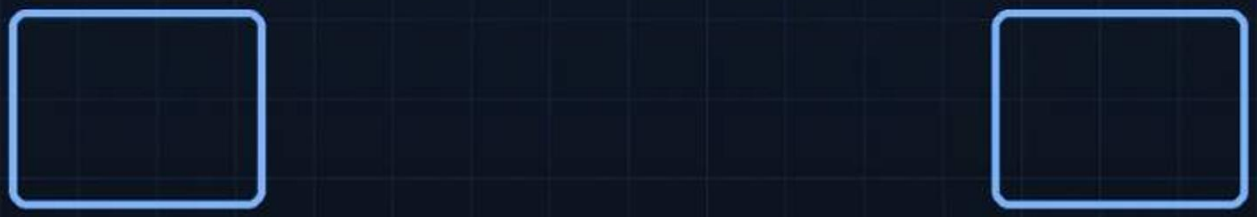
GoF · 07/23

СТРУКТУРНЫЙ

Python C++23 Rust Java





# Мост

Bridge

Алексей Примак

Алексей Примак

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

«Автор»
2026

**Примак А.**

Мост. Паттерн GoF на Python, C++23, Rust и Java / А. Примак — «Автор», 2026

Когда количество вариантов реализации начинает порождать взрыв подклассов, помогает разделить абстракцию и реализацию. «Мост» показывает этот подход на системе уведомлений с разными типами сообщений и каналами доставки. Паттерн GoF последовательно разобран и реализован на Python, C++23, Rust и Java. В книге есть классическая схема, практические модификации, гибридные варианты и самостоятельные работы.

© Примак А., 2026
© Автор, 2026

# Содержание

Введение 5
Как читать 6
Глава 7. Реализация на Python 7
7.1. Зачем нужен мост 7
7.2. Участники и схема 8
7.3. Классический пример: уведомления и каналы 9
Конец ознакомительного фрагмента. 12

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

## Введение

Эта книга посвящена одному паттерну проектирования — **Мост**(Bridge). 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.

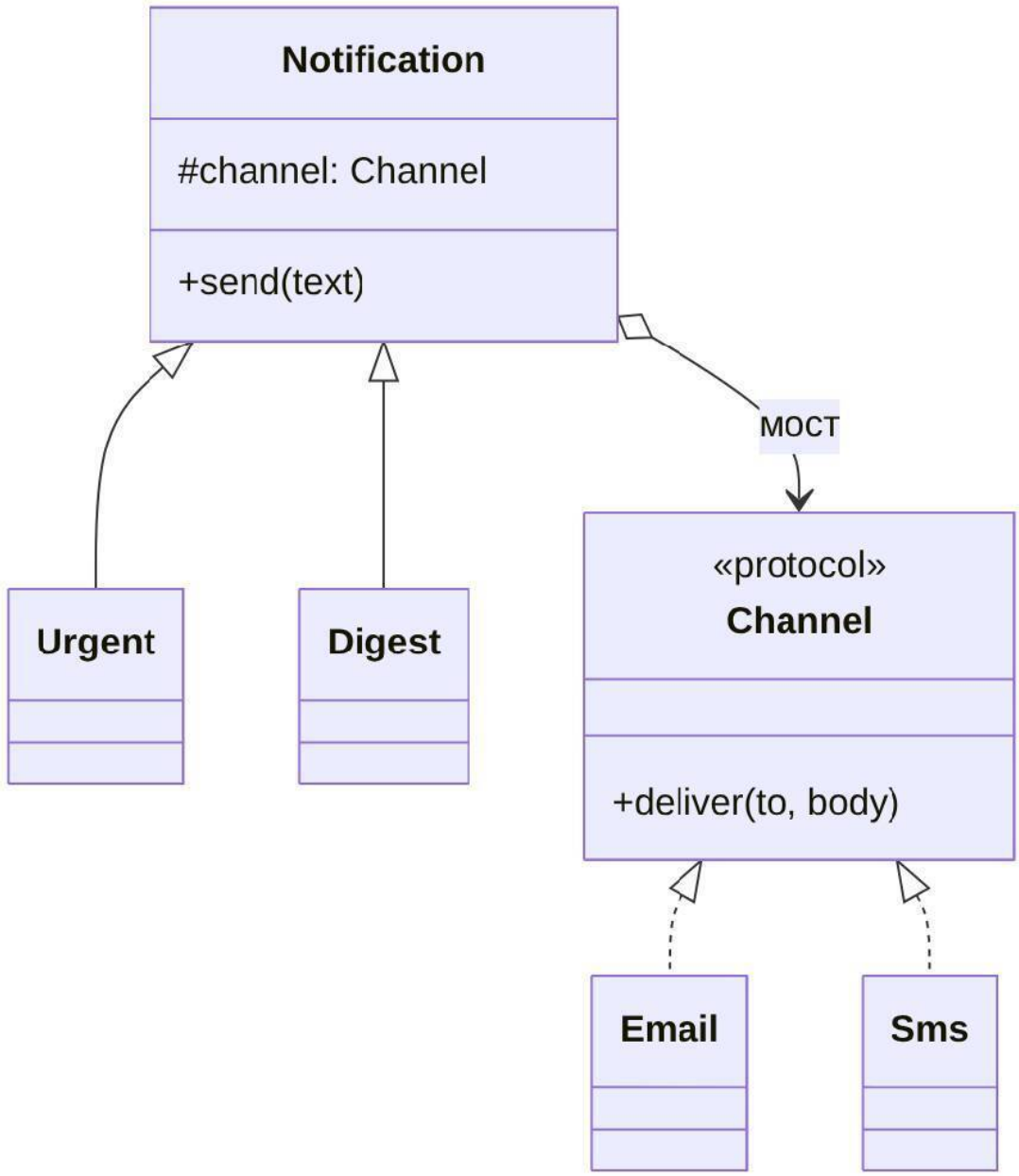
# Глава 7. Реализация на Python

## 7.1. Зачем нужен мост

Представьте систему уведомлений. Уведомления бывают обычные, срочные и сводные, а доставлять их можно по почте, в SMS и в мессенджер. Если делать отдельный класс на каждое сочетание, получится девять классов: UrgentEmail, UrgentSms, DigestTelegram… Новый канал добавит ещё три, новый вид уведомления — ещё три. Это «взрыв подклассов».

Мост разделяет две независимые оси изменений на две иерархии: **абстракцию**(что делаем — вид уведомления) и **реализацию**(как делаем — канал доставки). Абстракция хранит ссылку на реализацию и делегирует ей низкоуровневую работу. Теперь сочетаний по-прежнему девять, а классов — шесть, и оси растут независимо.

## 7.2. Участники и схема




**Abstraction—** высокоуровневый интерфейс, хранит ссылку на реализацию.
**Refined Abstraction—** разновидности абстракции.
**Implementor—** интерфейс реализации, обычно примитивнее абстракции.
**Concrete Implementor—** конкретные реализации.

Мост похож на адаптер устройством, но отличается целью: адаптер соединяет уже существующие несовместимые части, а мост проектируют заранее, чтобы две оси развивались независимо.

## 7.3. Классический пример: уведомления и каналы

**Файл 01_notifications.py:**

01_notifications.py — Python

```python
from typing import Protocol


class Channel(Protocol):
    name: str

    def deliver(self, to: str, body: str) -> None: ...


class Email:
    name = "email"

    def deliver(self, to: str, body: str) -> None:
        print(f"  [email → {to}@mail.ru] {body}")


class Sms:
    name = "sms"
    LIMIT = 40

    def deliver(self, to: str, body: str) -> None:
        if len(body) > self.LIMIT:
            body = body[: self.LIMIT - 1] + "…"
        print(f"  [sms → {to}] {body}")


class Notification:
    def __init__(self, channel: Channel) -> None:
        self.channel = channel

    def format(self, text: str) -> str:
        return text

    def send(self, to: str, text: str) -> None:
```

***Листинг 7.1*** *— Классический пример: уведомления и каналы*

01_notifications.py (продолжение) Python

```python
        self.channel.deliver(to, self.format(text))


class Urgent(Notification):
    def format(self, text: str) -> str:
        return f"СРОЧНО: {text.upper()}"

    def send(self, to: str, text: str) -> None:
        for attempt in (1, 2):  # срочное дублируем
            print(f"  попытка {attempt}:", end="")
            super().send(to, text)


class Digest(Notification):
    def __init__(self, channel: Channel) -> None:
        super().__init__(channel)
        self.items: list[str] = []

    def send(self, to: str, text: str) -> None:
        self.items.append(text)

    def flush(self, to: str) -> None:
        body = f"Сводка ({len(self.items)}): " + "; ".join(self.
            items)
        self.channel.deliver(to, body)
        self.items.clear()


if __name__ == "__main__":
    for channel in (Email(), Sms()):
        print(f"Канал {channel.name}:")
        Notification(channel).send("anna", "Ваш заказ отправлен"
            )
        Urgent(channel).send("anna", "сервер упал")
        digest = Digest(channel)
        for t in ("новый комментарий", "лайк", "подписчик"):
            digest.send("anna", t)
        digest.flush("anna")
```

```
$ python 01_notifications.py
Канал email:
  [email → anna@mail.ru] Ваш заказ отправлен
  попытка 1:  [email → anna@mail.ru] СРОЧНО: СЕРВЕР УПАЛ
  попытка 2:  [email → anna@mail.ru] СРОЧНО: СЕРВЕР УПАЛ
  [email → anna@mail.ru] Сводка (3): новый комментарий; лайк;
    подписчик
Канал sms:
  [sms → anna] Ваш заказ отправлен
  попытка 1:  [sms → anna] СРОЧНО: СЕРВЕР УПАЛ
  попытка 2:  [sms → anna] СРОЧНО: СЕРВЕР УПАЛ
  [sms → anna] Сводка (3): новый комментарий; лайк; по…
```

***Листинг 7.1 (продолжение)*** *— Классический пример: уведомления и каналы*

**Разбор.** Три вида уведомлений и два канала дают шесть сочетаний, но классов пять. Каждая сторона занимается своим: абстракция решает, *что* отправить и сколько раз (Urgent дублирует, Digest копит), реализация — *как* доставить (SMS обрезает длинный текст). Добавить Telegram — один новый класс, и он сразу работает со всеми видами уведомлений.

# 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.