Python C++23 Rust Java



Приспособленец

Flyweight

Алексей Примак

Алексей Примак

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

«Автор»
2026

Примак А.

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

Как экономить память, когда система содержит огромное количество похожих объектов? «Приспособленец» показывает, как отделить общие данные от индивидуального состояния и переиспользовать разделяемые объекты. На примерах текстового редактора и игровых объектов разобраны принципы паттерна GoF. Затем идут реализации на Python, C++23, Rust и Java, практические модификации, гибридные решения и самостоятельные задания.

© Примак А., 2026
© Автор, 2026

Содержание

Введение 5
Как читать 6
Глава 11. Реализация на Python 7
11.1. Зачем нужен приспособленец 7
11.2. Участники и схема 8
11.3. Классический пример: лес 9
Конец ознакомительного фрагмента. 11

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

Введение

Эта книга посвящена одному паттерну проектирования — **Приспособленец** (Flyweight). 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 11. Реализация на Python

11.1. Зачем нужен приспособленец

Текстовый редактор хранит миллион символов. Если у каждого символа есть собственный объект со шрифтом, размером, цветом и начертанием, память кончится быстро. Но разных сочетаний оформления в документе — от силы несколько десятков. В игре с лесом из ста тысяч деревьев у каждого дерева свои координаты, но модель, текстура и цвет общие для всех сосен.

Приспособленец разделяет состояние объекта на две части:

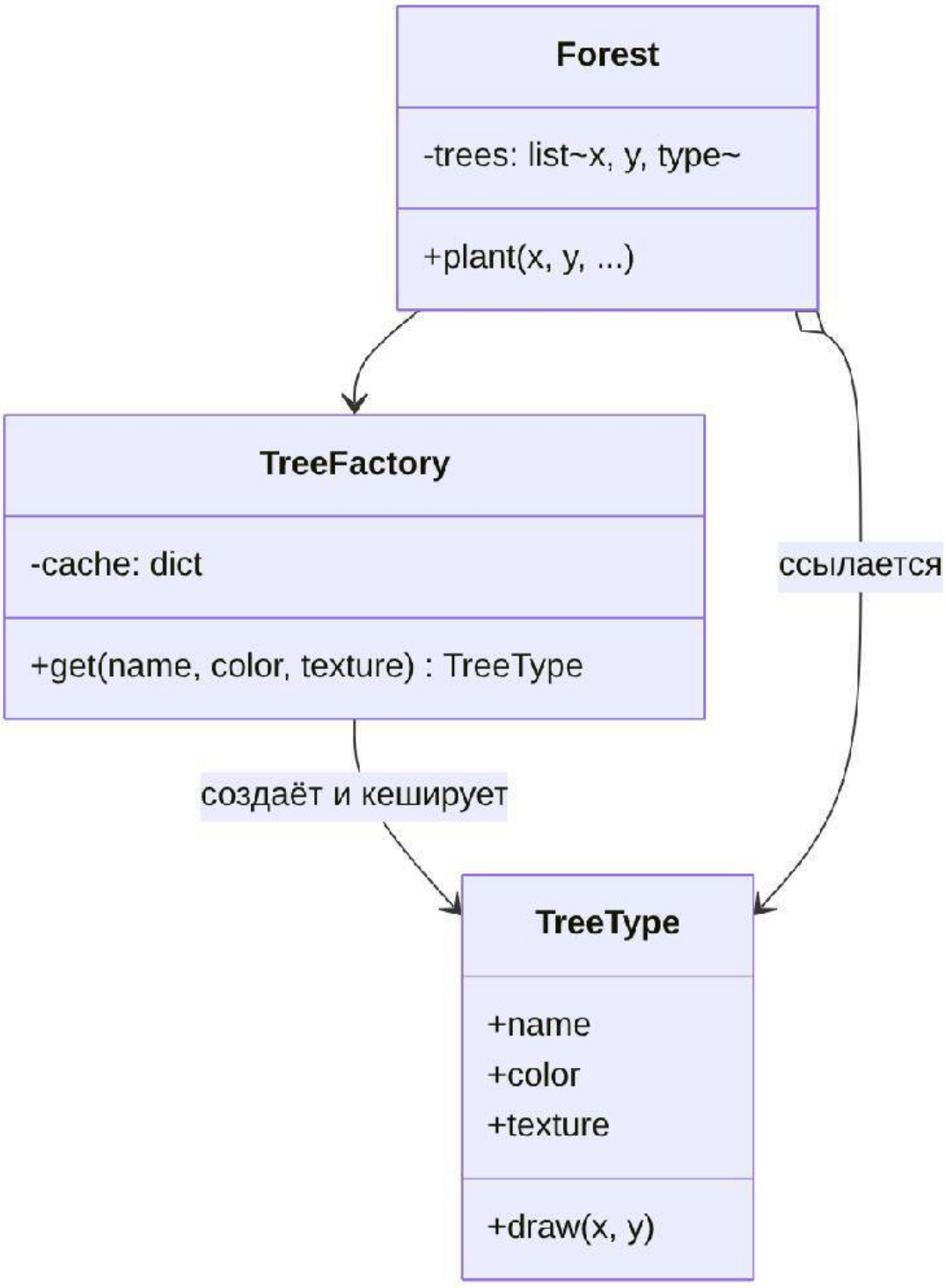
внутреннее(intrinsic) — общее, неизменяемое, хранится в разделяемом объекте: шрифт, текстура;

внешнее(extrinsic) — уникальное для каждого использования, передаётся снаружи или хранится компактно: позиция, координаты.

Фабрика выдаёт приспособленцев и следит, чтобы для одинакового внутреннего состояния существовал ровно один объект.

Python сам использует этот приём: малые целые числа от −5 до 256 и многие строки-идентификаторы интернируются, а sys.intern позволяет делать это явно.

11.2. Участники и схема



Flyweight— хранит внутреннее состояние, принимает внешнее параметрами.
Flyweight Factory— создаёт и переиспользует приспособленцев.
Client— хранит внешнее состояние и ссылки на приспособленцев.

11.3. Классический пример: лес

Файл 01_forest.py:

01_forest.py — Python

```python
import sys
from dataclasses import dataclass


@dataclass(frozen=True, slots=True)
class TreeType:
    name: str
    color: str
    texture: bytes  # «тяжёлые» данные

    def describe(self, x: int, y: int) -> str:
        return f"{self.name} ({self.color}) в точке ({x}, {y})"


class TreeFactory:
    def __init__(self) -> None:
        self._types: dict[tuple[str, str], TreeType] = {}

    def get(self, name: str, color: str) -> TreeType:
        key = (name, color)
        if key not in self._types:
            self._types[key] = TreeType(name, color, texture=
                bytes(10_000))
        return self._types[key]

    def __len__(self) -> int:
        return len(self._types)


class Forest:
    def __init__(self, factory: TreeFactory) -> None:
        self.factory = factory
        self.trees: list[tuple[int, int, TreeType]] = []

```

Листинг 11.1 *— Классический пример: лес*

01_forest.py (продолжение) Python

```python
    def plant(self, x: int, y: int, name: str, color: str) ->
        None:
        self.trees.append((x, y, self.factory.get(name, color)))


if __name__ == "__main__":
    kinds = [("сосна", "тёмно-зелёный"), ("берёза", "
        светло-зелёный"), ("клён", "оранжевый")]
    forest = Forest(TreeFactory())
    for i in range(30_000):
        name, color = kinds[i % 3]
        forest.plant(i * 7 % 1000, i * 13 % 1000, name, color)

    print("Деревьев:", len(forest.trees))
    print("Типов (объектов с текстурой):", len(forest.factory))
    shared = sum(len(t.texture) for t in {id(t[2]): t[2] for t
        in forest.trees}.values())
    naive = len(forest.trees) * 10_000
    print(f"Текстуры: {shared:,} Б вместо {naive:,} Б".replace("
        ,", " "))
    print(forest.trees[4][2].describe(*forest.trees[4][:2]))
    print("Один и тот же объект:", forest.trees[0][2] is forest.
        trees[3][2])
    print("Интернирование строк:", sys.intern("сосна".upper())
        is sys.intern("СОСНА"))
```

```
$ python 01_forest.py
Деревьев: 30000
Типов (объектов с текстурой): 3
Текстуры: 30 000 Б вместо 300 000 000 Б
берёза (светло-зелёный) в точке (28, 52)
Один и тот же объект: True
Интернирование строк: True
```

Листинг 11.1 (продолжение) *— Классический пример: лес*

Разбор. 30 тысяч деревьев ссылаются на три объекта TreeType. Тяжёлая текстура хранится трижды, а не 30 тысяч раз — экономия в 10 000 раз. Внешнее состояние (координаты) хранится в компактных кортежах. Приспособленец обязан быть неизменяемым — frozen=True: если бы кто-то перекрасил одну сосну, перекрасились бы все. sys.intern — встроенный приспособленец для строк: две одинаковые строки, полученные разными путями, становятся одним объектом.

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.