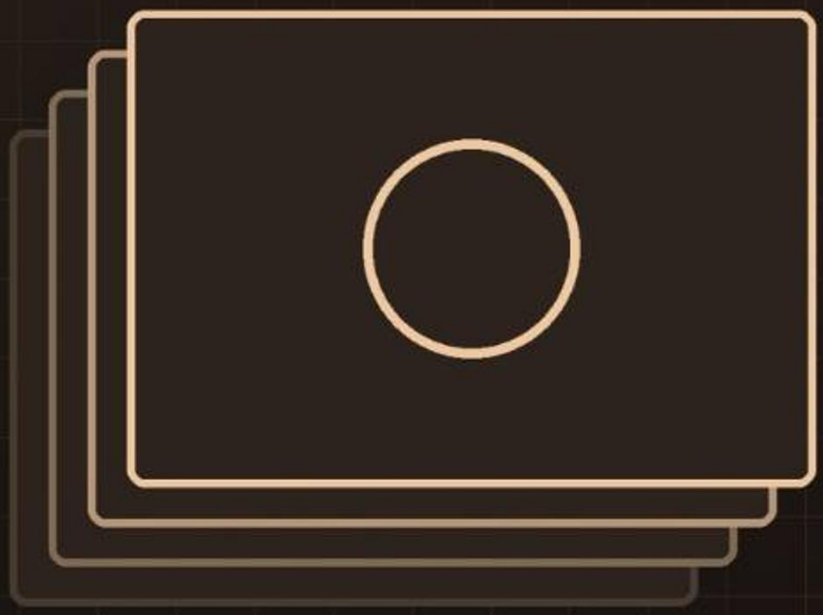
GoF · 18/23

ПОВЕДЕНЧЕСКИЙ

Python C++23 Rust Java



Снимок

Memento

Алексей Примак

Алексей Примак

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

«Автор»
2026

Примак А.

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

Как сохранить состояние объекта, чтобы позже восстановить его без раскрытия внутреннего устройства? «Снимок» разбирает этот вопрос на примерах сохранений игры, истории отмены, черновиков и отката операций. Паттерн GoF реализован на Python, C++23, Rust и Java. В книге рассматриваются структура снимка, управление историей, вопросы восстановления состояния, практические модификации и самостоятельные задания.

© Примак А., 2026
© Автор, 2026

Содержание

Введение 5
Как читать 6
Глава 18. Реализация на Python 7
18.1. Зачем нужен снимок 7
18.2. Участники и схема 8
18.3. Классический пример: сохранения в игре 9
Конец ознакомительного фрагмента. 11

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

Введение

Эта книга посвящена одному паттерну проектирования — **Снимок**(Memento). 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 18. Реализация на Python

18.1. Зачем нужен снимок

Игра сохраняется перед боссом, редактор графики хранит историю для отмены, форма помнит черновик, транзакция откатывается при ошибке. Во всех случаях нужно сохранить состояние объекта и позже его восстановить.

Сделать это «снаружи», копируя поля объекта, плохо: придётся открыть внутреннее устройство, и любое изменение класса сломает код сохранения. Снимок решает задачу иначе: объект сам создаёт снимок своего состояния и сам умеет из него восстановиться. Хранитель истории держит снимки, но не заглядывает внутрь.

Участники:

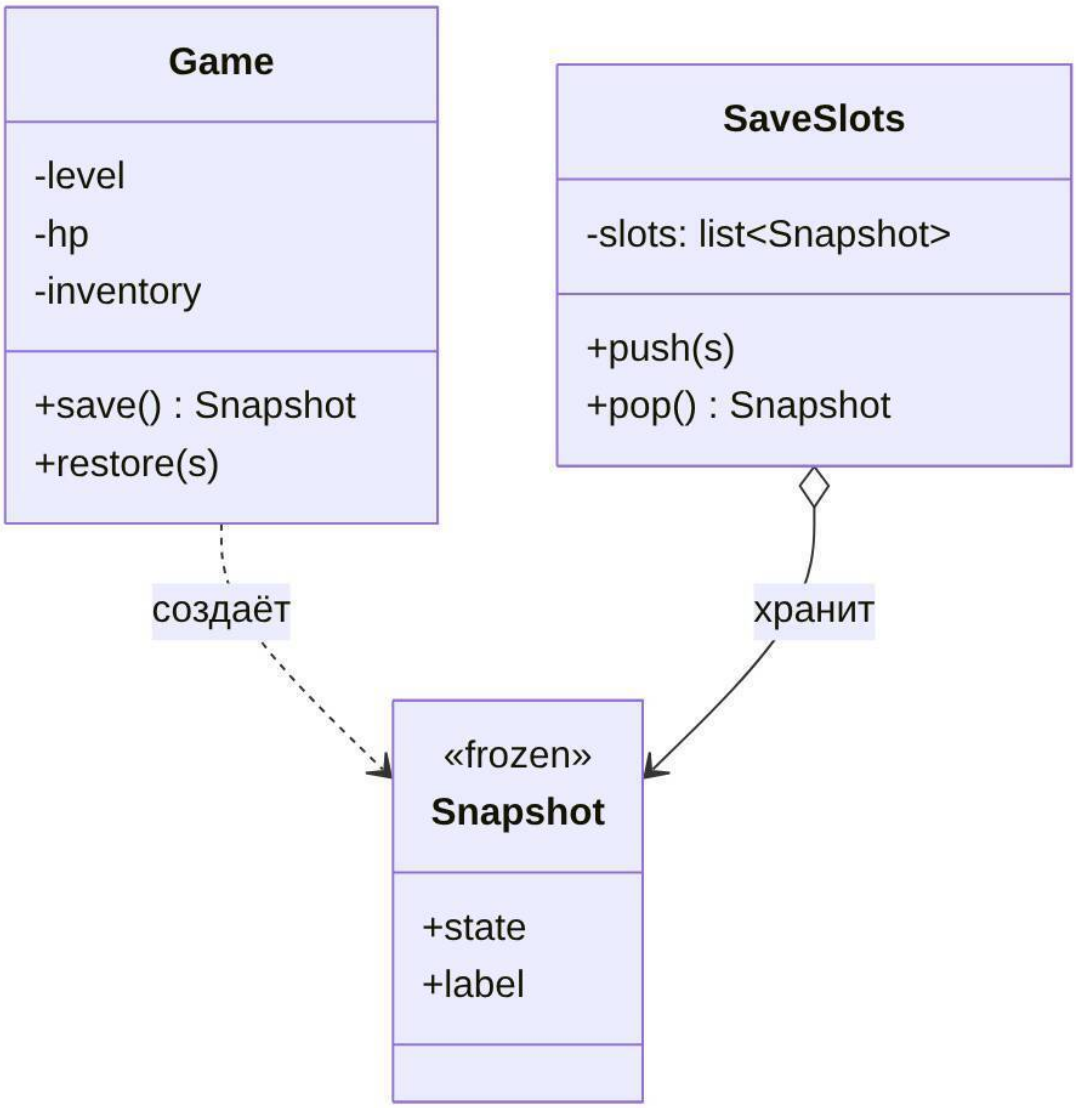
Originator(создатель) — объект, состояние которого сохраняется. Создаёт снимки и восстанавливается из них.

Memento(снимок) — неизменяемый объект с состоянием.

Caretaker(опекун) — хранит снимки и решает, когда сохранять и восстанавливать.

В Python снимок легко получить через неизменяемые структуры (dataclass(frozen=True), кортежи), модуль copy и сериализацию (pickle, json).

18.2. Участники и схема



18.3. Классический пример: сохранения в игре

Файл 01_game_saves.py:

01_game_saves.py — Python

```python
from dataclasses import dataclass, field


@dataclass(frozen=True)
class Snapshot:
    label: str
    _level: int
    _hp: int
    _inventory: tuple[str, ...]


@dataclass
class Game:
    level: int = 1
    hp: int = 100
    inventory: list[str] = field(default_factory=list)

    def save(self, label: str) -> Snapshot:
        return Snapshot(label, self.level, self.hp, tuple(self.
            inventory))

    def restore(self, s: Snapshot) -> None:
        self.level, self.hp, self.inventory = s._level, s._hp,
            list(s._inventory)

    def __str__(self) -> str:
        return f"уровень {self.level}, HP {self.hp}, вещи {self.
            inventory}"


class SaveSlots:
    def __init__(self, limit: int) -> None:
        self.limit = limit
        self.slots: list[Snapshot] = []
```

Листинг 18.1 *— Классический пример: сохранения в игре*

01_game_saves.py (продолжение) Python

```python

    def push(self, s: Snapshot) -> None:
        self.slots.append(s)
        if len(self.slots) > self.limit:
            dropped = self.slots.pop(0)
            print(f"  (старое сохранение «{dropped.label}
                » удалено)")

    def find(self, label: str) -> Snapshot:
        return next(s for s in self.slots if s.label == label)


if __name__ == "__main__":
    game, slots = Game(), SaveSlots(limit=2)
    game.inventory.append("меч")
    slots.push(game.save("начало"))
    game.level, game.hp = 5, 80
    game.inventory.append("зелье")
    slots.push(game.save("перед боссом"))
    game.hp = 0
    game.inventory.clear()
    print("После боя:   ", game)
    game.restore(slots.find("перед боссом"))
    print("Загрузка:    ", game)
    game.inventory.append("ключ")
    slots.push(game.save("после босса"))
    print("Сохранения:", [s.label for s in slots.slots])
    print("Снимок не изменился:", slots.find("перед боссом").
        _inventory)
```

```
$ python 01_game_saves.py
После боя:    уровень 5, HP 0, вещи []
Загрузка:     уровень 5, HP 80, вещи ['меч', 'зелье']
  (старое сохранение «начало» удалено)
Сохранения: ['перед боссом', 'после босса']
Снимок не изменился: ('меч', 'зелье')
```

Листинг 18.1 (продолжение) *— Классический пример: сохранения в игре*

Разбор.Снимок неизменяем: frozen=True запрещает присваивание полей, а инвентарь хранится кортежем. Это важно — если бы снимок хранил тот же список, что и игра, game.inventory.clear() уничтожил бы и сохранение. При восстановлении создаётся новый список, и последующие изменения игры не затрагивают снимок. Поля с подчёркиванием сообщают опекуну: «не трогай, это внутреннее».

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.