Построение и жизненный цикл: Строитель, Прототип, Одиночка

Автор: Мокобиа Джоан Чидиеберe

Группа: 24.Б83_мм

Курс: Программирование на Python

Дата: 25 Октябрь 2025

Порождающие паттерны управляют тем, как создаются объекты, делая код гибким и понятным.
Разберём три популярных паттерна: Builder (Строитель), Prototype (Прототип) и Singleton (Одиночка).

1. Что такое объект

Объект — это конкретный экземпляр класса.
Он объединяет данные (**атрибуты**) и поведение (**методы**).

Пример:

```
class Car:
    def __init__(self, model, color):
        self.model = model
        self.color = color

    def drive(self):
        print(f"{self.color} {self.model} drive!")

car = Car("Toyota Model 2", "Blue")
car.drive()
```

```
Blue Toyota Model 2 drive!
```

2. Строитель (Builder)

Идея:
Отделить процесс пошагового создания сложного объекта от его представления.
Один и тот же процесс можно использовать для разных версий продукта.

Когда применять:

- Объект имеет много частей или опций
- Не хочется писать громоздкий конструктор

- Нужно собирать разные варианты одного объекта

```
from dataclasses import dataclass, field

@dataclass
class PC:
    cpu: str = ""
    gpu: str = ""
    ram: int = 0
    storage: list = field(default_factory=list)

    def __str__(self):
        return f"PC(cpu={self.cpu}, gpu={self.gpu}, ram={self.ram}GB,
storage={self.storage})"


class GamingPCBuilder:
    def __init__(self):
        self.pc = PC()

    def set_parts(self):
        self.pc.cpu = "AMD Ryzen 7"
        self.pc.gpu = "RTX 4070"
        self.pc.ram = 32
        self.pc.storage = ["1TB NVMe", "2TB HDD"]
        return self

    def build(self):
        return self.pc


class OfficePCBuilder(GamingPCBuilder):
    def set_parts(self):
        self.pc.cpu = "Intel i5"
        self.pc.gpu = "Integrated"
        self.pc.ram = 16
        self.pc.storage = ["512GB SSD"]
        return self


gaming = GamingPCBuilder().set_parts().build()
office = OfficePCBuilder().set_parts().build()

print("Gaming PC:", gaming)
print("Office PC:", office)

Gaming PC: PC(cpu=AMD Ryzen 7, gpu=RTX 4070, ram=32GB, storage=['1TB
NVMe', '2TB HDD'])
Office PC: PC(cpu=Intel i5, gpu=Integrated, ram=16GB, storage=['512GB
SSD'])
```

Плюсы:

- Пошаговое создание сложного объекта
- Один процесс — разные продукты
- Код проще тестировать и расширять

3. Прототип (Prototype)

Идея:
Создавать новые объекты копированием существующих.
Полезно, когда создание “с нуля” дорого или сложно.

Разница копий:

- copy.copy() — поверхностная копия (вложенные объекты не дублируются)
- copy.deepcopy() — глубокая копия (всё копируется рекурсивно)

```
import copy
from dataclasses import dataclass, field

@dataclass
class Config:
    name: str
    settings: dict = field(default_factory=dict)

base = Config("Base", {"debug": False, "mode": "prod"})
clone = copy.deepcopy(base)
clone.settings["debug"] = True

print("Оригинал:", base)
print("Копия:", clone)
```

```
Оригинал: Config(name='Base', settings={'debug': False, 'mode':
'prod'})
Копия: Config(name='Base', settings={'debug': True, 'mode': 'prod'})
```

Плюсы:

- Быстрое создание новых объектов
- Меньше дублирования кода

4. Одиночка (Singleton)

Идея:
Обеспечить, чтобы существовал только один экземпляр класса.
Полезен для логгеров, настроек, кэшей и подключений к БД.

Когда применять:

- Нужен один глобальный объект (например, логгер)
- Все части программы должны обращаться к одной и той же сущности

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super().__new__(cls)
            cls._instance.value = 0
        return cls._instance

a = Singleton()
b = Singleton()
a.value = 42

print(a is b, b.value)  # True 42
```

```
True 42
```

Плюсы:

- Единая точка доступа к данным
- Централизованное управление ресурсами

Минусы:

- Скрытая глобальность
- Сложнее тестировать

5. Сравнение

Паттерн	Суть	Когда применять
Builder	Пошаговая сборка объекта	Сложные объекты с множеством параметров
Prototype	Копирование готового объекта	Когда дорого создавать с нуля
Singleton	Один экземпляр на всё приложение	Глобальные настройки, логгер, кэш

7. Краткое резюме

Builder:

«Помогает строить сложные объекты пошагово, отделяя процесс сборки от деталей.»

Prototype:

«Позволяет быстро копировать готовые объекты, экономя ресурсы.»

Singleton:

«Гарантирует, что в системе есть только один экземпляр класса.»

Финал:

«Builder — как строим, Prototype — как копируем, Singleton — сколько экземпляров существует.»