Объектно-ориентированное программирование

Объявление классов

src/12. OOP/example_01/

Не рекомендуемый способ добавления переменных класса

src/12. OOP/example_02/

Не рекомендуемый доступ к внутренним переменным класса

src/12. OOP/example_03/

Переменные уровня класса (статические переменные)

src/12. OOP/example_04/
src/12. OOP/example_05/

Свойства класса

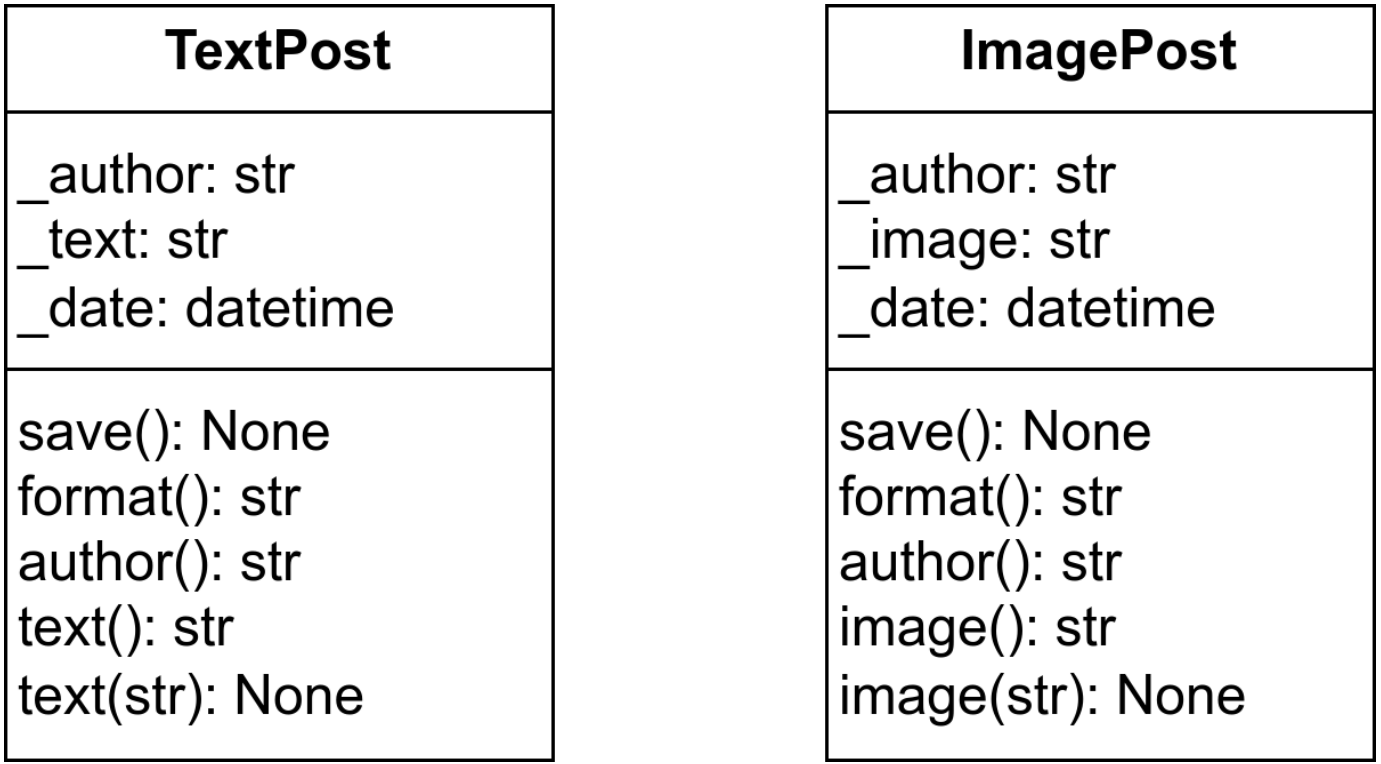
src/12. OOP/example_06/

Статические методы класса

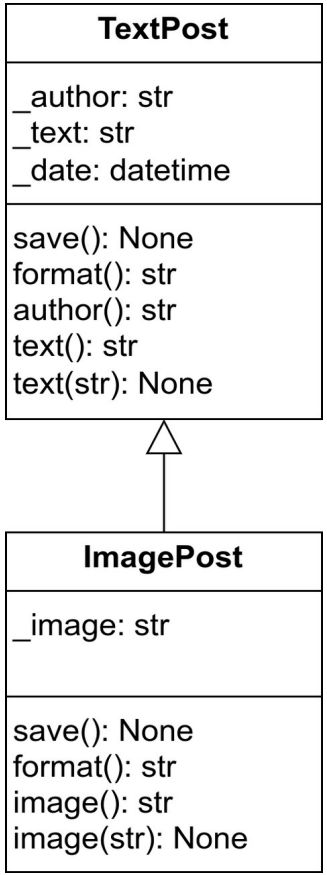
src/12. OOP/example_07/

Наследование

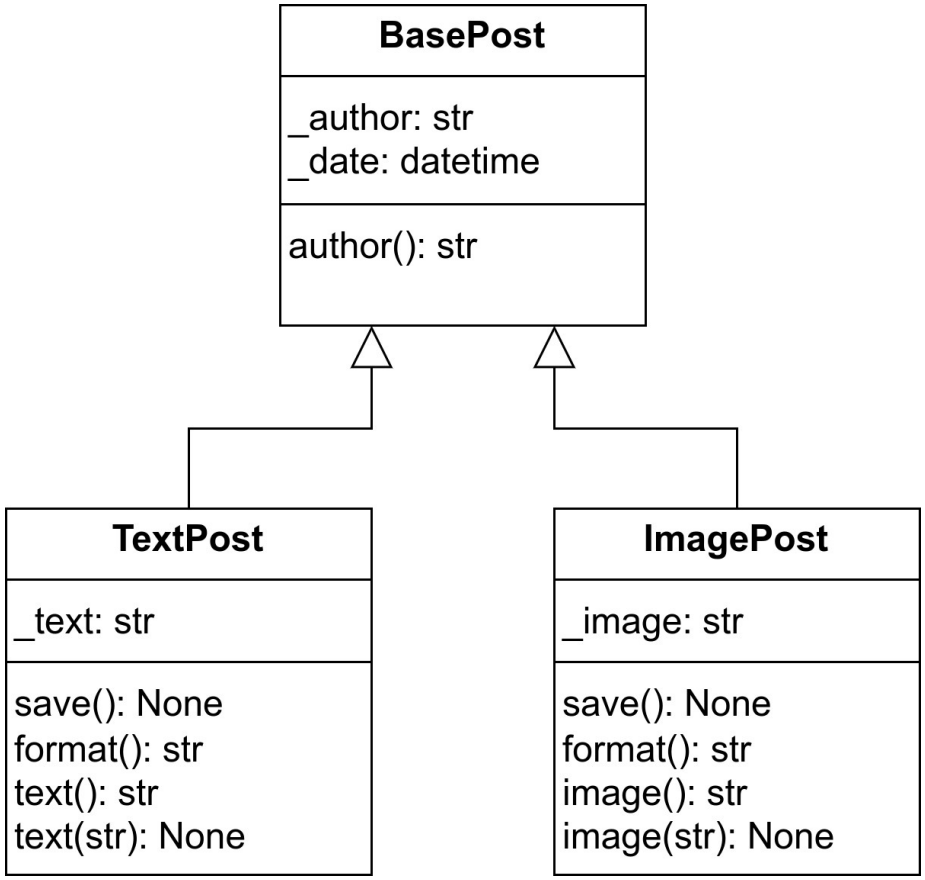
UML-диаграмма классов без использования наследования (src/10. OOP/example_08/)



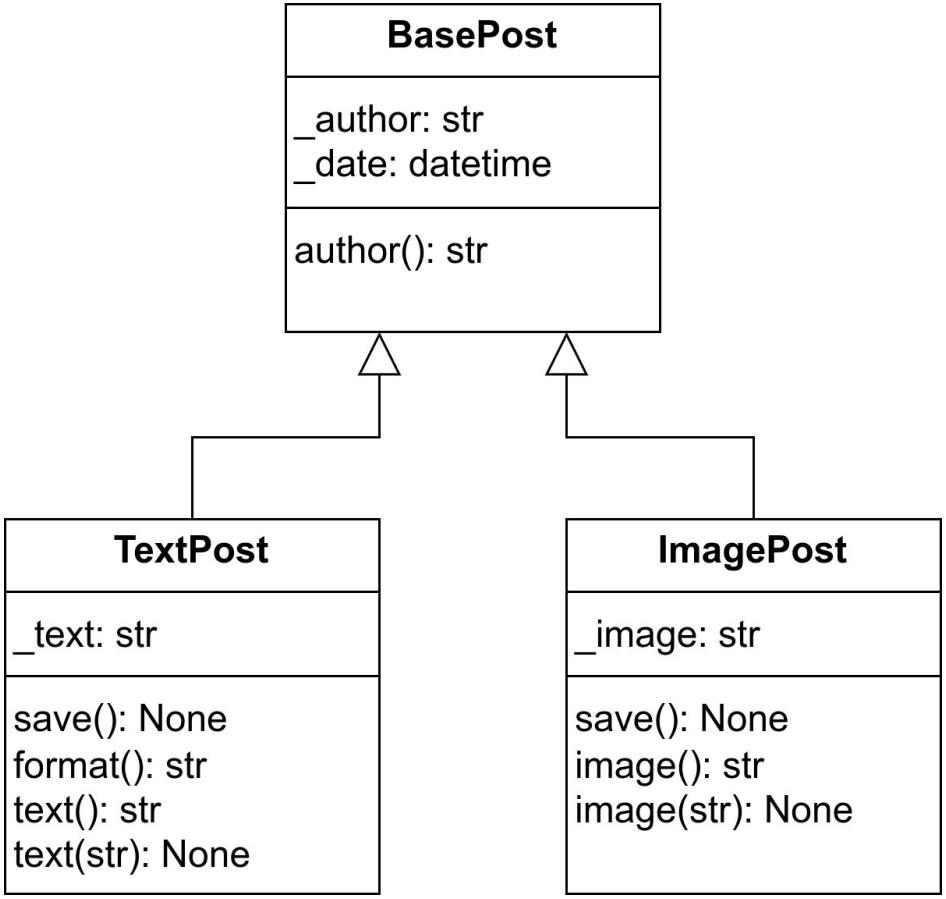
UML-диаграмма классов с использованием наследования (src/10. OOP/example_09/)



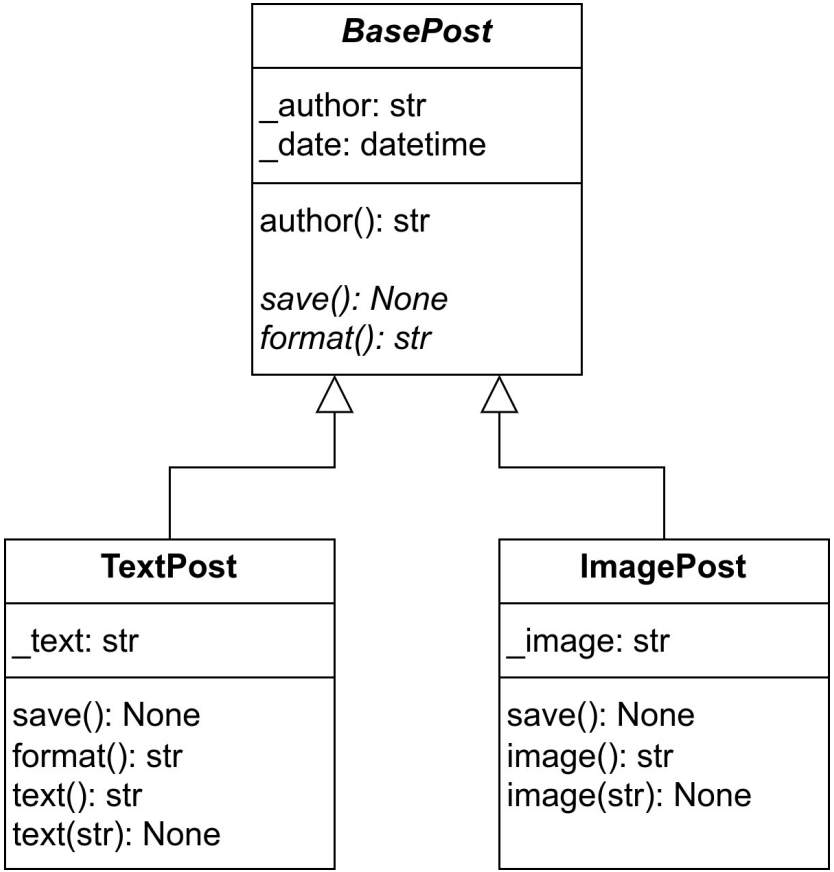
UML-диаграмма классов с базовым классом (src/10. OOP/example_10/)



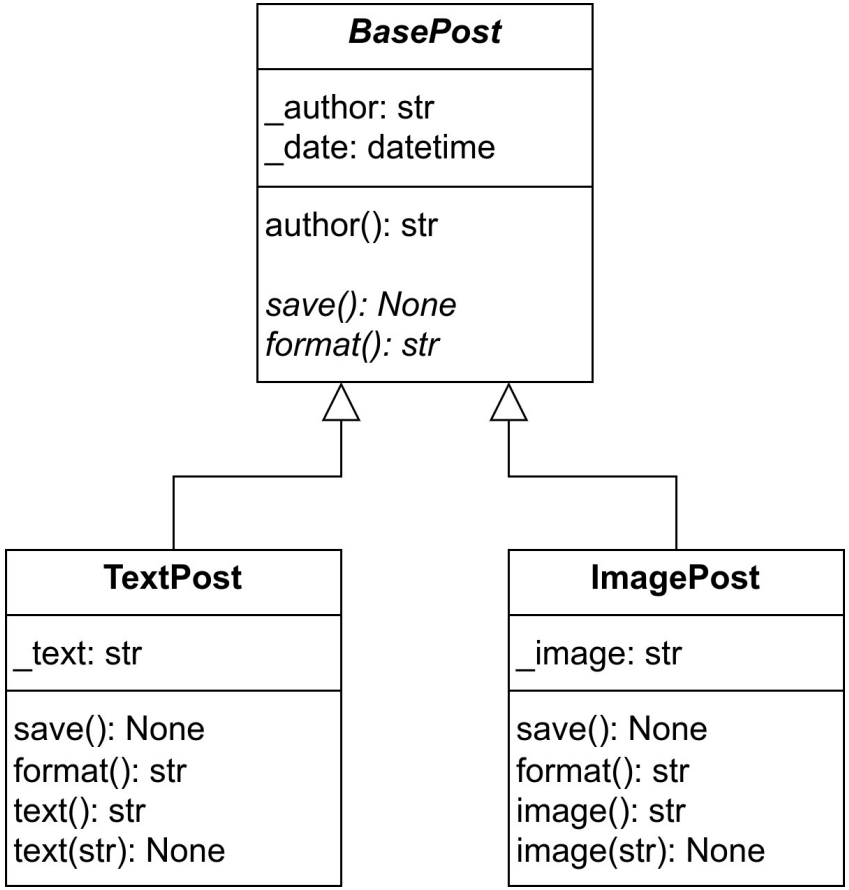
Пример с ошибкой отсутствия ожидаемого метода (src/10. OOP/example_11/)



Пример с ошибкой отсутствия ожидаемого метода и абстрактным базовым классом (src/10. OOP/example_12/)



UML-диаграмма классов с абстрактным базовым классом (src/10. OOP/example_13/)



Специальные (dunder) методы класса

* dunder — **d**ouble **under**line

```python
def add(a, b):
    return a + b

def mul(a, b):
    return a * b

def spam(a, b, action):
    return action(a, b) * 2 + 1

foo = spam(2, 3, add)
bar = spam(2, 3, mul)

print(dir(add))
print()

print(f"{foo=}")
print(f"{bar=}")
```

```
['__annotations__', '__builtins__', '__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delattr__',
'__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__getstate__',
'__globals__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__', '__lt__', '__module__',
'__name__', '__ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__']

foo=11
bar=13
```

```python
import math

class MathAction:
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y

    def __call__(self, a, b) -> float:
        print("MathAction.__call__()")
        return a * math.sin(self.x) + b * math.cos(self.y)


def add(a, b):
    return a + b

def spam(a, b, action):
    return action(a, b) * 2 + 1

foo = spam(2, 3, add)
bar = spam(2, 3, MathAction(math.pi / 3, math.pi / 4))

print(f"{foo=}")
print(f"{bar=}")
```

src/12. OOP/example_15/

UML-диаграмма классов с перегрузкой функции __len__() (src/10. OOP/example_16/)