Глава 20. Дескрипторы атрибутов

> Изучение дескрипторов не только расширяет доступный инструментарий, но позволяет глубже понять, как работает Python, и оценить элегантность его дизайна[1].
>
> — Раймонд Хэттигер, один из разработчиков Python и гуру

Дескрипторы — это способ повторного использования одной и той же логики доступа в нескольких атрибутах. Например, типы полей в ОО отображениях типа Django ORM и SQL Alchemy — дескрипторы, управляющие потоком данных от полей в записи БД к атрибутам Python-объекта и обратно.

Десериптор — это класс, который реализует протокол, содержащий методы `__get__`, `__set__` и `__delete__`. Класс `property` реализует весь протокол дескриптора. Как обычно, разрешается реализовывать протокол частично. На самом деле, большинство дескрипторов, встречающихся в реальных программах, реализуют только методы `__get__` и `__set__`, а многие — и вовсе лишь один из них.

Пример дескриптора: проверка значений атрибутов

Фабрика свойств — это функция высшего порядка, которая создает параметризованный набор функций-акцессоров. Она строит из них экземпляры конкретных свойств, настройки которых, например `storage_name`, хранятся в замыканиях. ОО способ решения той же задачи — дескрипторный класс.

Мы вернёмся к примеру класса `LineItem` с того места, где остановились, и переделаем фабрику свойств `quantity` в дескрипторный класс `Quantity`.

`LineItem` попытка №3: простой дескриптор

Класс, в котором реализован хотя бы один из методов `__get__`, `__set__` или `__delete__`, является дескриптором. Для использования дескриптора мы объявляем его экземпляром атрибута какого-то другого класса.

Мы создадим дескриптор `Quantity` и включим в класс `LineItem` два экземпляра `Quantity`: для управления атрибутами `wight` и `price`. Всё это изображено на диаграме классов.

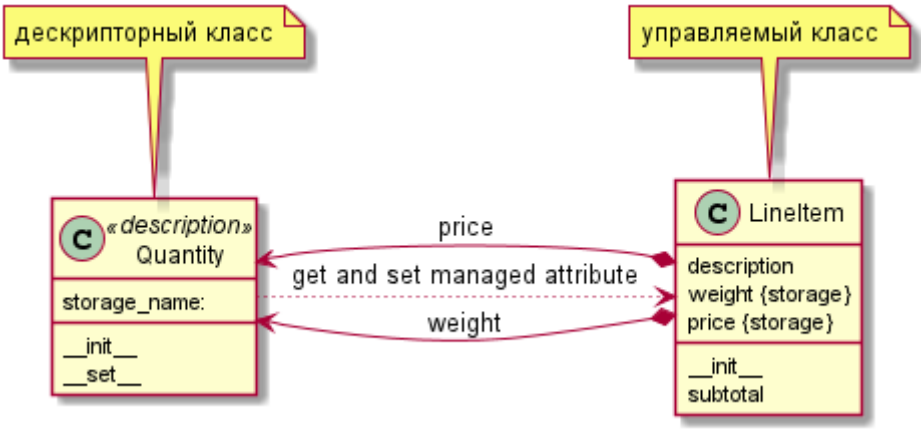

Figure 1. UML-диаграмма класса `LineItem` и используемого в нём дескрипторного класса `Quantity`

Дескрипторный класс

Класс, реализующий протокол дескриптора. Это класс `Quantity`.

Управляемый класс

Класс, в котором атрибуты класса, являющиеся экземпляром дескриптора. Это класс `LineItem`.

Экземпляр дескриптора

Любой экземпляр дескрипторного класса, объявленный атрибутом класса в управляемом классе.

Управляемый экземпляр

Один экземпляр управляемого класса.

Атрибуты хранения

Атрибуты управляемого экземпляра, в котором хранится значение управляемого атрибута для данного экземпляра.

Управляемый атрибут

Открытый атрибут управляемого класса, который обрабатывается экземпляром дескриптора, а значение которого хранится в одном из атрибутов хранения. Другими словами, экземпляр дескриптора и атрибут хранения в совокупности образуют инфраструктуру для управляемого атрибута.

bulkfood_v3.py: *дескриптор* Quantity *управляют атрибутами* LineItem.

```
class Quantity:
    """
    Дескриптор основан на протоколе,
    для его реализации не требуется
    наследование.
    """
    def __init__(self, storage_name):
        """
        :param storage_name: имя атрибута, в котором
        хранится значение управляемого экземпляра.
        """
        self.storage_name = storage_name

    def __set__(self, instance, value):
        """
        Вызывается при любой попытке присвоить значение управляемому атрибуту.
        :param instance: управляемый экземпляр.
        :param value: присваиваемое значение.
        :return:
        """
        if value > 0:
            """
            Здесь мы должны работать с атрибутом __dict__ управляемого экземпляра
напрямую;
            попытка воспользоваться встроенной функцией setattr привела бы к
повторному
            вызову метода __set__ и, стало быть, к бесконечной рекурсии.
            """
            instance.__dict__[self.storage_name] = value
        else:
            raise ValueError('value must be > 0')


class LineItem:
    weight = Quantity('weight')
    price = Quantity('price')

    def __init__(self, description, weight, price):
        self.description = description
        self.weight = weight
        self.price = price

    def subtotal(self):
        return self.weight * self.price
```

Кодируя метод `__set__`, не забывайте, что означают аргументы `self` и `instance`: `self` — это экземпляр дескриптора, а `instance` — управляемый экземпляр. Дескрипторы, управляющие атрибутами экземпляра, должны хранить значения в управляемых экземплярах. Поэтому-то Python и передаёт аргумент `instance` методам дескриптора.

Может возникнуть соблазн хранить значения всех управляемых атрибутов в экземпляре самого дескриптора, т.е. в методе `__set__` вместо кода

```
instance.__dict__[self.storage_name] = value
```

написать:

```
self.__dict__[self.storage_name] = value
```

Но это совершенно неправильно! Одновременно в памяти могут находиться тысячи экземпляров `LineItem`, но экземплярова дескриптора будет только 2: `LineItem.weight`, `LineItem.price`.

В примере есть недостаток: повторяющиеся имена атрибутов. Хорошо было бы использовать `LineItem` как-то так:

```
class LineItem:
    weight = Quantity()
    price = Quantity()
...
```

Проблема в том, что правая часть присваивания вычисляется ещё до того, как начинает существовать переменная.

`LineItem` попытка №4: автоматическая генерация имен атрибутов хранения

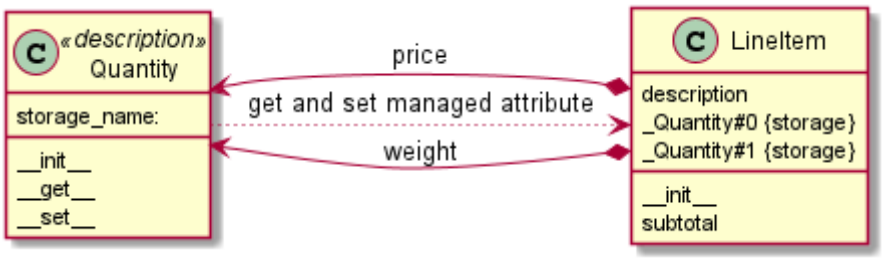

Figure 2. Теперь в классе `Quantity` есть оба метода `get` и `set`, а в экземплярах `LineItem` — атрибуты хранения со сгенерированными именами: `_Quantity#0` и `_Quantity#1`

Для генерации `storage`

[1] Raymond Hettinger "Descriptor HowTo Guide"