Объектно-ориентированное программирование и делегирование: промежуточные объекты-оболочки

Наряду с наследованием и композицией программисты, занимающиеся ООП, часто говорят о *делегировании*, что обычно подразумевает объекты контроллеров с внедренными другими объектами, которым они передают запросы операций. Контроллеры могут заниматься административными действиями, такими как ведение журналов либо проверка достоверности доступа, добавляя дополнительные шаги к компонентам интерфейса или отслеживая активные экземпляры.

В известном смысле делегирование является особой формой композиции с единственным внедренным объектом, управляемым классом *оболочки* (иногда называемого *промежуточным* классом), который предохраняет большую часть или весь интерфейс внедренного объекта. Понятие промежуточных классов временами применяется и к другим механизмам, таким как вызовы функций; при делегировании нас интересуют промежуточные классы для *всех* линий поведения объекта, включая вызовы методов и прочие операции.

Такая концепция была введена через пример в главе 28, и в Python она часто реализуется с помощью метода `__getattr__`, рассмотренного в главе 30. Поскольку этот метод перегрузки операции перехватывает доступ к несуществующим атрибутам, класс оболочки может использовать `__getattr__` для маршрутизации произвольного доступа к внутреннему объекту. Из-за того, что метод `__getattr__` дает возможность маршрутизировать запросы атрибутов обобщенным образом, класс оболочки предохраняет интерфейс внутреннего объекта и сам может добавлять дополнительные операции.

В качестве обзора взгляните на содержимое файла `trace.py` (одинаково выполняющегося в Python 2.X и 3.X):

```
class Wrapper:
    def __init__(self, object):
        self.wrapped = object                          # Сохранить объект
    def __getattr__(self, attrname):
        print('Trace: ' + attrname)                    # Трассировать извлечение
        return getattr(self.wrapped, attrname)         # Делегировать извлечение
```

Вспомните из главы 30, что метод `__getattr__` получает имя атрибута в виде строки. В коде атрибут извлекается из внутреннего объекта по строковому имени посредством встроенной функции `getattr` – вызов `getattr(X,N)` похож на `X.N`, но `N` является выражением, вычисляемым в строку во время выполнения, а не переменной. На самом деле вызов `getattr(X,N)` подобен выражению `X.__dict__[N]`, но первый вариант также производит поиск в иерархии наследования, как и `X.N`, а второй – нет (за информацией об атрибуте `__dict__` обращайтесь в главу 22 первого тома и в главу 29).

Вы можете применять подход с классом оболочки, продемонстрированный в модуле, для управления доступом к любому объекту с атрибутами – спискам, словарям и даже классам и экземплярам. Здесь класс `Wrapper` просто выводит трассировочное сообщение при каждом доступе к атрибуту и делегирует запрос атрибута внутреннему объекту `wrapped`:

```
>>> from trace import Wrapper
>>> x = Wrapper([1, 2, 3])                  # Создать оболочку для списка
>>> x.append(4)                             # Делегировать списковому методу
Trace: append
```