Классы являются объектами: обобщенные фабрики объектов

Временами проектные решения на основе классов требуют создания объектов в ответ на условия, которые невозможно предсказать на стадии написания кода программы. Паттерн проектирования "Фабрика" делает возможным такой отложенный подход. Во многом благодаря гибкости Python фабрики могут принимать многочисленные формы, ряд которых вообще не кажутся особыми.

Из-за того, что классы также являются объектами "первого класса", их легко передавать в рамках программы, сохранять в структурах данных и т.д. Вы также можете передавать классы функциями, которые генерируют произвольные виды объектов; в кругах объектно-ориентированного проектирования такие функции иногда называются *фабриками*. Фабрики могут оказаться крупным делом в строго типизированных языках вроде C++, но довольно просты в реализации на Python.

Скажем, синтаксис вызова, представленный в главе 18 первого тома, позволяет вызывать любой класс с любым количеством позиционных или ключевых аргументов конструктора за один шаг для создания экземпляра любого вида [2]:

```
def factory(aClass, *pargs, **kargs): # Кортеж или словарь с переменным
                                      # количеством аргументов
    return aClass(*pargs, **kargs)  # Вызывает aClass (или apply в Python 2.X)

class Spam:
    def doit(self, message):
        print(message)

class Person:
    def __init__(self, name, job=None):
        self.name = name
        self.job = job

object1 = factory(Spam)                          # Создать объект Spam
object2 = factory(Person, "Arthur", "King")      # Создать объект Person
object3 = factory(Person, name='Brian')      # То же самое, с ключевым аргу-
                                             # ментом и стандартным значением
```

В коде мы определяем функцию генератора объектов по имени `factory`. Она ожидает передачи объекта класса (подойдет любой класс) наряду с одним и более аргументов для конструктора класса. Для вызова функции и возвращения экземпляра применяется специальный синтаксис с переменным количеством аргументов.

В оставшемся коде примера просто определяются два класса и генерируются экземпляры обоих классов за счет их передачи функции `factory`. И это единственная фабричная функция, которую вам когда-либо придется писать на Python; она работает для любого класса и любых аргументов конструкторов. Ниже функция `factory` демонстрируется в действии (файл `factory.py`):

[2] В действительности этот синтаксис дает возможность вызывать любой вызываемый объект, включая функции, классы и методы. Следовательно, функция `factory` здесь может также запускать любой вызываемый объект, а не только класс (несмотря на имя аргумента). Кроме того, как было показано в главе 18 первого тома, в Python 2.X имеется альтернатива `aClass(*pargs, **kargs)`: встроенный вызов `apply(aClass, pargs, kargs)`, который был удален в Python 3.X из-за своей избыточности и ограничений.