Объектно-ориентированное программирование и композиция: отношения "имеет"

Понятие композиции было введено в главах 26 и 28. С точки зрения *программиста* композиция затрагивает внедрение других объектов в объект контейнера и их активизацию для реализации методов контейнера. С точки зрения *проектировщика* композиция является еще одним способом представления отношений в предметной области. Но вместо членства в наборе композиция имеет дело с компонентами — частями целого.

Композиция также отражает взаимосвязи между частями, называемые отношениями "имеет". В некоторых книгах по объектно-ориентированному проектированию на композицию ссылаются как на *агрегирование* или проводят различие между этими двумя терминами, используя агрегирование для описания более слабой зависимости между контейнером и его содержимым. Здесь под "композицией" понимается просто совокупность внедренных объектов. Составной класс обычно предоставляет собственный интерфейс и реализует его, направляя выполнение действий внедренным объектам.

После реализации классов сотрудников давайте поместим их в пиццерию и предоставим работу. Наша пиццерия является составным объектом: в ней есть духовой шкаф, а также сотрудники вроде официантов и шеф-поваров. Когда клиент входит и размещает заказ, компоненты пиццерии приступают к действиям — официант принимает заказ, шеф-повар готовит пиццу и т.д. Следующий пример (файл `pizzashop.py`) выполняется одинаково в Python 3.X и 2.X и эмулирует все объекты и отношения в описанном сценарии:

```
# Файл pizzashop.py (Python 2.X + 3.X)
from __future__ import print_function
from employees import PizzaRobot, Server

class Customer:
    def __init__(self, name):
        self.name = name
    def order(self, server):
        print(self.name, "orders from", server)        # заказы от
    def pay(self, server):
        print(self.name, "pays for item to", server)   # плата за единицу
class Oven:
    def bake(self):
        print("oven bakes")                # духовой шкаф выпекает
class PizzaShop:
    def __init__(self):
        self.server = Server('Pat')        # Внедрить другие объекты
        self.chef = PizzaRobot('Bob')      # Робот по имени bob
        self.oven = Oven()
    def order(self, name):
        customer = Customer(name)          # Активизировать другие объекты
        customer.order(self.server)        # Заказы клиента, принятые официантом
        self.chef.work()
        self.oven.bake()
        customer.pay(self.server)
if __name__ == "__main__":
    scene = PizzaShop()                    # Создать составной объект
    scene.order('Homer')                   # Эмулировать заказ клиента Homer
    print('...')
    scene.order('Shaggy')                  # Эмулировать заказ клиента Shaggy
```