Питон
Курс лекций
Лекция десятая

© Зайцев Вадим Валерьевич, 2002–2010,
spider.vz@gmail.com

6 Составные части объектного подхода

Объектная модель не появляется просто так. Существует большая теория, аргументирующая применения той или иной техники. Мы ограничимся беглым описанием основных составных частей объектно-ориентированного программирования и проектирования и их реализации в языке питон.

6.1 Абстрагирование

Абстрагирование является одним из основных подходов, используемых для решения сложных задач. Абстрагирование — это стратегический выбор точки зрения, это то, что опрееляет важные для нас свойства и качества моделируемого. Не имея возможность познать окружающий его мир полностью, человек выделяет из него то, что кажется ему важнее для поставленной задачи. Они из крупнейших теоретиков и практиков программирования прошлого века, основоположник структурного программирования, голландский математик Эдсгер Дейкстра писал, что «абстрагирование проявляется в нахождении сходств между определёнными объектами, ситуациями или процессами реального мира, и в принятии решений на основе этих свойств, отвлекаясь на время от имеющихся различий».

Не отрекаясь от уважения к великому человеку, предложим своё определение: *Абстракция определяет концептуальные границы некоторого объекта, опуская одни его характеристики и выделяя другие, отличающие его ото всех прочих видов объектов.*

Гради Буч, один из главных теоретиков OOD, ввёл в дополнение к определению так называемый **принцип наименьшего удивления**, согласно которому абстракция должна охватывать всё поведение объекта, не больше и не меньше, не привнося сюрпризов и побочных эффектов.

Правильный выбор набора абстракций для предметной области — главная задача OOD. Существуют три основных типа абстракций:

- **Абстракция сущности** — объект олицетворяет собой модель некой сущности, принадлежащей предметной области.
- **Абстракция поведения** — объект состоит из обобщённого множества операций для работы с элементами предметной области.
- **Абстракция виртуальной машины** — объект группирует операции, находящиеся на одном уровне абстракции.

В качестве примера абстракции сущности можно привести первый класс, созданный нами в питоне — «собаку». Если взять какое-либо приложение, вынужденное обмениваться сообщениями по сети, для которого, тем не менее, этот процесс не является основным (не лежит в предметной области), то объект «соединение» будет абстракцией поведения, потому что, не отвечая никакой сущности, он содержит методы «послать», «получить» и т.д.

Объект, реализующий всевозможные функции работы с определённым форматом файлов, представляет собой типичнейший пример абстракции виртуальной машины. Его реализация использует объект «файл», лежащий на более низком уровне абстракции, его интерфейс может использоваться объектом более высокого уровня, собственно работающим с этим форматом.

Соглашение о взаимодействии абстракций двух объектов называют контрактом. Контрактную модель программирования впервые рассмотрел Бертран Мейер — человек, сделавший для OOD не меньше Буча. Эта модель базируется на инвариантах (логических условиях, значение которых должно сохраняться). Для каждой операции объекта задаются предусловия (инварианты, предполагаемые операцией) и постусловия (инварианты, которым удовлетворяет операция). Изменение инварианта нарушает контракт, связанный с абстракцией. В нарушении предусловия нужно винить вызывающую сторону, постусловия — выполняющую. В современных языках программирования для надзора за контрактами применяются средства работы с исключительными ситуациями. Есть они и в питоне.

Исключительные ситуации (или исключения) — это ошибки, возникающие во время выполнения программы. Когда программы исполнялись неинтерактивно (без взаимодействия с пользователем), соответствующая реакция на исключительную ситуацию заключалась в том, чтобы напечатать сообщение об ошибке и завершить выполнение программы. Однако реакция на исключение в интерактивной среде не модет быть ограничена сообщением, а должна также включать восстановление. Это может быть продолжение вычислений, возврат к последнему узлу, в котором пользователь может выбрать другой вариант, либо выход из внутренней процедуры с последующим её блокированием. В системах, которые должны работать стабильно по своей природе (например, в программном комплексе системы управления запуском ракет), восстановление должно выполняться без участия человека!

Восстановление при ошибках не даётся даром — какие-то накладные расходы неизбежны. Но нужно следить за тем, чтобы:

- В случае отсутствия исключения издержки должны быть очень небольшими,
- Обработка исключения должна быть простой, быстрой и свободной от ошибок.

Действительно, мы рассчитываем на то, что исключительные ситуации, как правило, **не возникают**, поэтому они не должны слишком замедлять выполнение программы. И, конечно, программирование реакции на исключения не должно быть слишком утомительно по той же причине.

Упражнение. Является ли алгоритм обработки исключительных ситуаций эквивалентным условному оператору? Почему? Какие у них общие черты? Какие различия?

Теперь поговорим об этом механизме в питоне. Исключительная ситуация (exception) *возникает* (raised) в том месте, где происходит ошибка. Она может быть *перехвачена* (handled) специальным аппаратом, о котором и идёт речь. Исключительная ситуация может возникнуть не только по вине, но и по желанию программиста:

```
raise <имя>
```

При этом имя должно означать имя стандартной исключительной ситуации, строку с именем новой или же экземпляр какого-либо класса. Исключительные ситуации сами по себе являются объектами соответствующих классов. Так, конкретная ошибка деления на ноль — это объект класса `ZeroDivisionError`. Если в качестве имени дать что-либо другое (например, число), исключительная ситуация возникнет, но другая — `TypeError`. Вторым, необязательным, параметром оператора raise может идти параметр ошибки.

Оператор попытки выполнения (а именно он занимается перехватом исключительных ситуаций) бывает двух видов. Вот так записывается первый:

```
try:
 <операторы>
except <Имя>:
 <операторы>
else:
 <операторы>
```

Этот вид оператора мы будем называть try/except. Операторы, помещаемые между try и первым except, называются *испытательными инструкциями* (термин в англоязычной литераторе — *try clause*). Они выполняются в любом случае, и, если ничего не происходит (не возникает ошибки), выполнение программы продолжается с оператора, следующего за всей конструкцией. Дальше идут обработчики ошибок, начинающиеся с ключевого слова `except` и названия ошибки, после чего идёт собственно

код обработчика. Таких обработчиков может быть несколько. Необязательная else-ветвь оператора try/except выполняется в том случае, если ни одна ошибка не произошла. Обычно в этом месте располагается код, являющийся продолжением кода из испытательной части, но ошибки в котором не должны обрабатываться (или должны обрабатываться по другому — не стоит забывать о том, что конструкции try могут быть вложенными).

Второй вид, называемый нами try/finally, имеет следующий вид:

```
try:
 <операторы>
finally:
 <операторы>
```

Финализирующие инструкции выполняются *в любом случае*: была ошибка или нет. В том случае, если ошибка произошла, её обработка задерживается. Сначала выполняются все операции, стоящие после `finally`, и только затем исключение возникает ещё раз. Эту конструкцию используют в основном для того, чтобы успеть что-то сделать до выхода из программы (выдать сообщение, закрыть файл, etc).

В питоне существуют двадцать семь стандартных исключения. С помощью определённых объектно-ориентированных приёмов можно написать любое количество своих исключений. Основные стандартные исключения таковы:

- **ArithmeticError** — арифметические ошибки (кроме деления на ноль),
- **AttributeError** — ошибка доступа к несуществующему свойству или методу,
- **EOFError** — ошибка конца файла,
- **EnvironmentError** — ошибка интерактивной среды,
- **FloatingPointError** — ошибка при вычислениях с плавающей точкой,
- **ImportError** — ошибка подключения внешних модулей,
- **IndentationError** — ошибка отступов (отступ без составного оператора, отсутствие отступа, etc),
- **IndexError** — попытка чтения несуществующего элемента последовательности,
- **KeyError** — попытка чтения несуществующего элемента словаря,
- **MemoryError** — ошибка выделения памяти,
- **NameError** — неверное имя переменной,

- **NotImplementedError** — возникает при вызове абстрактного метода объекта, то есть метода с описанием, но без тела,
- **OSError** — внешняя ошибка питона, предположительно относящаяся к операционной системе (у этого класса наследует класс **WindowsError** — класс ошибок, относящихся к операционной системе Microsoft Windows),
- **OverflowError** — ошибка переполнения,
- **RuntimeError** — ошибка времени выполнения (базовый класс, наследниками которого являются многие другие),
- **SyntaxError** — ошибка синтаксиса,
- **SystemError** — внутренняя системная ошибка питона,
- **TypeError** — ошибка приведения типов данных (например, сложение числа со строкой),
- **UnboundLocalError** — при попытке сослаться на локальную переменную, с которой не связано ни одно значение,
- **UnicodeError** — ошибка при кодировании или раскодировании уникодовых строк,
- **ValueError** — ошибка значения, очень часто возникает, имеет много наследников,
- **ZeroDivisionError** — ошибка деления на ноль.

Специально заучивать названия ошибок, конечно, не нужно. Те, что будут у вас возникать достаточно часто, запомнятся и без лишних усилий, а об остальных всегда может поведать стандартная документация.