Объектно-ориентированное программирование

В разработке большинства современных программ принимает участие множество специалистов. При этом возникает новая проблема – нужно разделить работу между ними так, чтобы каждый мог работать независимо от других, а потом готовую программу можно было бы собрать вместе из готовых блоков, как из кубиков.

Как отмечал известный нидерландский программист Эдсгер Дейкстра, человечество еще в древности придумало способ управления сложными системами: «разделяй и властвуй». Это означает, что исходную систему нужно разбить на подсистемы (выполнить декомпозицию) так, чтобы работу каждой из них можно было рассматривать и совершенствовать независимо от других.

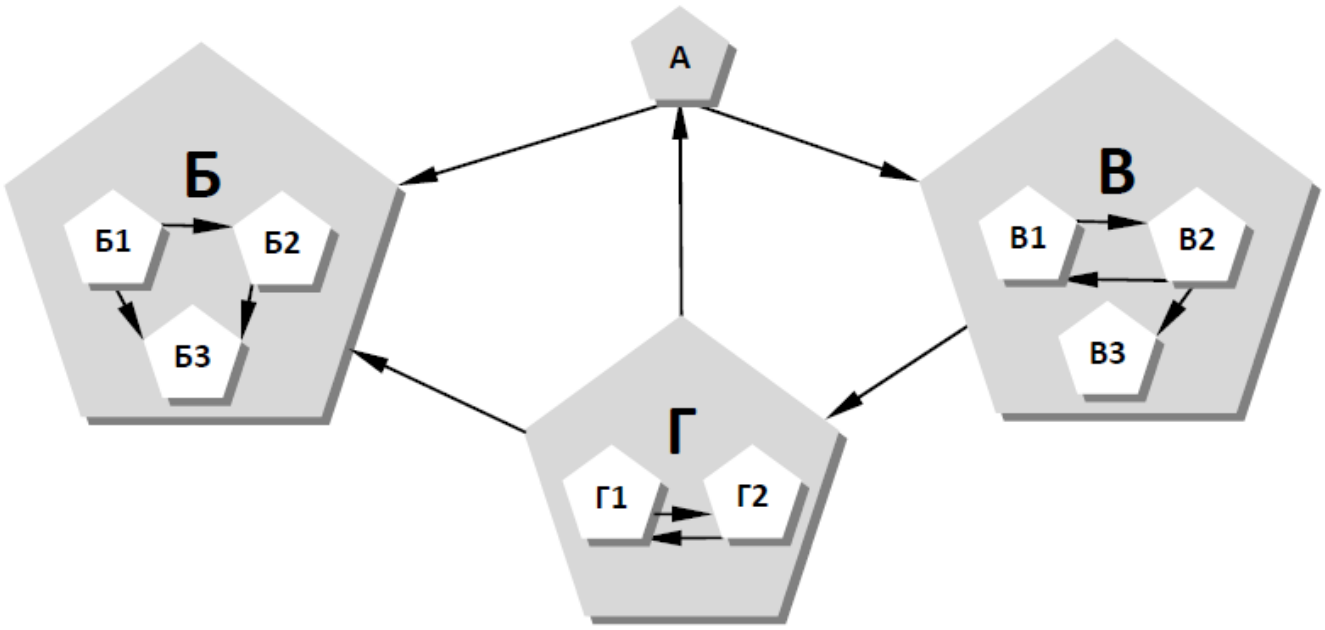
Для этого в классическом (процедурном)программировании используют метод проектирования *«сверху вниз»:* сложная задача разбивается на части (подзадачи и соответствующие им алгоритмы), которые затем снова разбиваются на более мелкие подзадачи и т.д. Однако при этом задачу «реального мира» приходится переформулировать, представляя все данные в виде переменных, массивов, списков и других структур данных. При моделировании больших систем объем этих данных увеличивается, они становятся плохо управляемыми, и это приводит к большому числу ошибок. Так как любой алгоритм может обратиться к любым глобальным (общедоступным) данным, повышается риск случайного недопустимого изменения каких-то значений.

В конце 60-х годов XX века появилась новая идея – применить в разработке программ тот подход, который использует человек в повседневной жизни. Люди воспринимают мир как множество объектов – предметов, животных, людей – это отмечал еще в XVII веке французский математик и философ Рене Декарт. Все объекты имеют внутреннее устройство и состояние, свойства (внешние характеристики) и поведение. Чтобы справиться со сложностью окружающего мира, люди часто игнорируют многие свойства объектов, ограничиваясь лишь теми, которые необходимы для решения их практических задач. Такой прием называется абстракцией.

Абстракция – это выделение существенных характеристик объекта, отличающих его от других объектов

Для разных задач существенные свойства могут быть совершенно разные. Например, услышав слово «кошка», многие подумают о пушистом усатом животном, которое мурлыкает, когда его гладят. В то же время ветеринарный врач представляет скелет, ткани и внутренние органы кошки, которую ему нужно лечить. В каждом из этих случаев применение абстракции дает разные модели одного и того же объекта, поскольку различны цели моделирования.

Как применить принцип абстракции в программировании? Поскольку формулировка задач, решаемых на компьютерах, все более приближается к формулировкам реальных жизненных задач, возникла такая идея: представить программу в виде множества объектов (моделей), каждый из которых обладает своими свойствами и поведением, но его внутреннее устройство скрыто от других объектов. Тогда решение задачи сводится к моделированию взаимодействия этих объектов. Построенная таким образом модель задачи называется объектной. Здесь тоже идет проектирование «сверху вниз», только не по алгоритмам (как в процедурном программировании), а по объектам. Если нарисовать схему такой декомпозиции, она представляет собой граф, так как каждый объект может обмениваться данными со всеми другими:



Здесь А, Б, В и Г – объекты «верхнего уровня»; Б1, Б2 и Б3 – подобъекты объекта Б и т.д.

Для решения задачи «на верхнем уровне» достаточно определить, что делает тот или иной объект, не заботясь о том, как именно он это делает. Таким образом, для преодоления сложности мы используем абстракцию, то есть сознательно отбрасываем второстепенные детали.

Если построена объектная модель задачи (выделены объекты и определены правила обмена данными между ними), можно поручить разработку каждого из объектов отдельному программисту (или группе), которые должны написать соответствующую часть программы, то есть определить, как именно объект выполняет свои функции. При этом конкретному разработчику необязательно держать в голове полную информацию обо всех объектах, нужно лишь строго соблюдать соглашения о способе обмена данными (интерфейсе) «своего» объекта с другими. Программирование, основанное на моделировании задачи реального мира как множества взаимодействующих объектов, принято называть объектно-ориентированным программированием (ООП). Более строгое определение мы дадим немного позже.

Объекты и классы

Для того, чтобы построить объектную модель, нужно:

- выделить взаимодействующие объекты, с помощью которых можно достаточно полно описать поведение моделируемой системы;
- определить их свойства, существенные в данной задаче;
- описать поведение (возможные действия) объектов, то есть команды, которые объекты могут выполнит

Этап разработки модели, на котором решаются перечисленные выше задачи, называется *объектно-ориентированным анализом (ООА).* Он выполняется до того, как программисты напишут самую первую строчку кода, и во многом определяет качество и надежность будущей программы.

Рассмотрим объектно-ориентированный анализ на примере простой задачи. Пусть нам необходимо изучить движение автомобилей на шоссе, например, для того, чтобы определить, достаточна ли его пропускная способность. Как построить объектную модель этой задачи? Прежде всего, нужно разобраться, что такое объект.

Объектом можно назвать то, что имеет четкие границы и обладает *состоянием* и *поведением*.

Состояние объекта определяет его возможное поведение. Например, лежачий человек не может прыгнуть, а незаряженное ружье не выстрелит.

В нашей задаче объекты – это дорога и двигающиеся по ней машины. Машин может быть несколько, причем все они с точки зрения нашей задачи имеют общие свойства. Поэтому нет смысла описывать отдельно каждую машину: достаточно один раз определить их общие черты, а потом просто сказать, что все машины ими обладают. В ООП для этой цели вводится специальный термин – *класс*.

Класс – это множество объектов, имеющих общую структуру и общее поведение.

Например, в рассматриваемой задаче можно ввести два класса – *Дорога* и *Машина*. По условию дорога одна, а машин может быть много.

Будем рассматривать прямой отрезок дороги, в этом случае объект «дорога» имеет два свойства, важных для нашей задачи: длину и число полос движения. Эти свойства определяют состояние дороги. «*Поведение*» дороги может заключаться в том, что число полос уменьшается, например, из-за ремонта покрытия, но в нашей простейшей модели объект «дорога» не будет изменяться.

Дорога
длина ширина

Схематично класс Дорога можно изобразить в виде прямоугольника с тремя секциями: в верхней записывают название *класса*, во второй части – свойства, а в третьей – возможные действия, которые называют *методами*. В нашей модели дороги два свойства и ни одного метода.

Теперь рассмотрим объекты класса Машина. Их важнейшие свойства – координаты и скорость движения. Для упрощения будем считать, что

• все машины одинаковы;

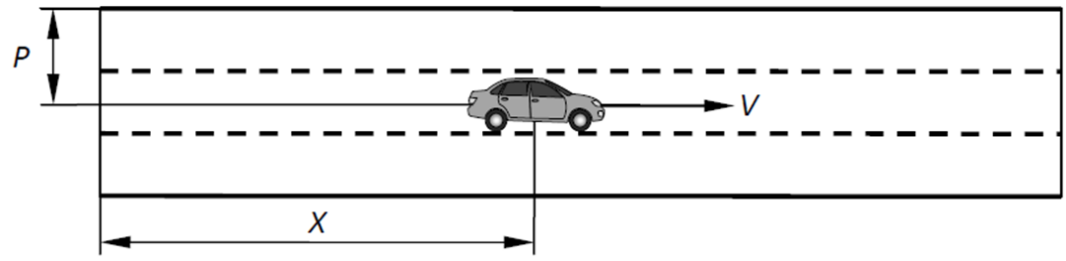
• все машины движутся по дороге слева направо с постоянной скоростью (скорости разных машин могут быть различны);

• по каждой полосе движения едет только одна машина, так что можно не учитывать обгон и переход на другую полосу;

• если машина выходит за правую границу дороги, вместо нее слева на той же полосе появляется новая машина.

Не все эти допущения выглядят естественно, но такая простая модель позволит понять основные принципы метода.

За координаты машины можно принять расстояние **X** от левого края рассматриваемого участка шоссе и номер полосы **P** (натуральное число). Скорость автомобиля **V** в нашей модели – неотрицательная величина.



Машина
X (координата) P(полоса) V (скорость)
двигаться

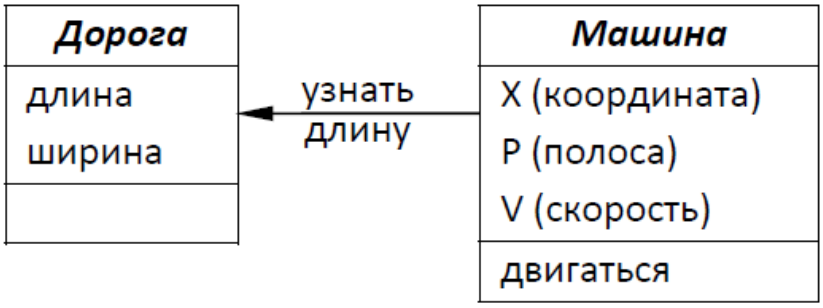
Теперь рассмотрим поведение машины. В данной модели она может выполнять всего одну команду – ехать в заданном направлении (назовём её «двигаться»). Говорят, что объекты класса Машина имеют метод «двигаться».

Метод – это процедура или функция, принадлежащая классу объектов.

Другими словами, метод – это некоторое действие, которое могут выполнять все объекты класса.

Пока мы построили только модели отдельных объектов (точнее, классов). Чтобы моделировать всю систему, нужно разобраться, как эти объекты взаимодействуют. Объект-машина должен уметь «определить», что закончился рассматриваемый участок

дороги. Для этого машина должна обращаться к объекту «дорога», запрашивая длину дороги (см. стрелку на схеме)



Такая схема определяет

• свойства объектов;

• операции, которые они могут выполнять;

• связи (обмен данными) между объектами.

В то же время мы пока ничего не говорили о том, *как* устроены объекты и *как* именно они будут выполнять эти операции. Согласно принципам ООП, ни один объект не должен зависеть от внутреннего устройства и алгоритмов работы других объектов. Поэтому, построив такую схему, можно поручить разработку двух классов объектов двум программистам, каждый из которых может решать свою задачу независимо от других. Важно только, чтобы все они четко соблюдали *интерфейс* – правила, описывающие взаимодействие «своих» объектов с остальными.

Создание объектов в программе

Класс Дорога

Объектно-ориентированная программа начинается с описания классов объектов. Класс в программе – это новый тип данных. Как и структура, класс – это составной тип данных, который может объединять переменные различного типа в единый блок. Кроме того, класс обычно содержит не только данные, но и методы работы с ними (процедуры и функции).

В нашей программе самый простой класс – это *Дорога* (англ. *road*). Объекты этого класса имеют два свойства (в Python они называются *атрибутами*): длину (англ. *length*), которая может быть вещественным числом, и ширину (англ. *width*) – количество полос, целое число. Для хранения значений свойств используются переменные, принадлежащие объекту, которые называются *полями*.

Поле – это переменная, принадлежащая объекту.

Значения полей описывают состояние объекта (а методы – его *поведение*). Простейшее описание класса *Дорога* в программе на Python выглядит так:

```
class TRoad:
  pass
```

Эти строчки вводят новый тип данных – класс **TRoad**, то есть сообщают компилятору, что в программе, возможно, будут использоваться объекты этого типа. При этом в памяти не создается ни одного объекта. Это описание – как чертёж, по которому в нужный момент можно построить сколько угодно таких объектов.

Чтобы создать объект класса **TRoad**, нужно вызвать специальную функцию без параметров, имя которой совпадает с именем класса:

```
road = TRoad()
```

Созданный объект относится к классу **TRoad**, поэтому его называют экземпляром класса **TRoad**.

При описании класса мы ничего не говорили о функции, которую только что вызвали. Она добавляется транслятором ко всем классам автоматически и называется *конструктором*.

Конструктор – это метод класса, который вызывается для создания объекта этого класса.

У каждого объекта класс **TRoad** должно быть два поля (длина и ширина), которые мы сразу заполним нулями. Для этого нужно определить свой конструктор – метод класса с именем **__init__** (от англ. *initialization* – начальные установки).

```
class TRoad:
  def __init__( self ):       # конструктор
    self.length = 0
    self.width = 0
```

Первый параметр конструктора (как и любого метода класса) в Python – это ссылка на сам объект, который создаётся. По традиции он всегда называется **self** (от англ. *self* – «сам»). С помощью этой ссылки мы обращаемся к полям объекта, например, **self.length** означает «поле **length** текущего объекта». Если вместо этого написать просто **length**, транслятор будет считать, что речь идет о локальной или глобальной переменной, а не о поле объекта. В этом конструкторе создаются и заполняются нулями два поля (*атрибута*) объекта – длина **length** и ширина **width**.

Свойства дороги можно изменить с помощью точечной записи, с которой вы познакомились, работая со структурами:

```
road.length = 60
road.width = 3
```

Полная программа, которая создает объект «дорога» (и больше ничего не делает) выглядит так:

```
class TRoad:
  def __init__( self ):       # конструктор
    self.length = 0
    self.width = 0
road = TRoad()
road.length = 60
road.width = 3
```

Начальные значения полей можно задавать прямо при создании объекта. Для этого нужно изменить конструктор, добавив два параметра – начальные значения длины и ширины дороги:

```
class TRoad:
  def __init__( self, length0, width0 ):    # конструктор
    if length0 > 0:
      self.length = length0
    else:
      self.length = 0
    if width0 > 0:
      self.width = width0
    else:
      self.width = 0
```

В этом конструкторе проверяется правильность переданных параметров, чтобы по ошибке длина и ширина дороги не оказались отрицательными. Теперь создавать объект будет проще:

```
road = TRoad ( 60, 3 )
```

Длина этой дороги – 60 единиц, она содержит 3 полосы. Обратите внимание, что мы передали только два параметра, а не три, как указано в заголовке метода **`__init__`**. Параметр **`self`**, который указан самым первым, транслятор подставляет автоматически.

Таким образом, класс выполняет роль «фабрики», которая при вызове конструктора «выпускает» (создает) объекты «по чертежу» (описанию класса).

Класс Машина

Теперь можно описать класс *Машина* (в программе назовём его **`TCar`**). Объекты класса **`TCar`** имеют три свойства: координата **`X`**, скорость **`V`** и номер полосы **`P`**:

```
class TCar:
  def __init__( self, road0, p0, v0 ): # конструктор
    self.road = road0
    self.P = p0
    self.V = v0
    self.X = 0
```

Так как объекты-машины должны обращаться к объекту «дорога», в область данных включено дополнительное поле **`road`**. Конечно, это не значит, что в состав машины

входит дорога. Напомним, что это только ссылка, в которую сразу после создания объекта-машины нужно записать адрес заранее созданного объекта «дорога».

Для того, чтобы машина могла двигаться, нужно добавить к классу один метод – процедуру **move**:

```
class TCar:
  def __init__( self, road0, p0, v0 ): # конструктор
    self.road = road0
    self.P = p0
    self.V = v0
    self.X = 0
  def move( self ):           # движение машины
    self.X += self.V
    if self.X > self.road.length:
      self.X = 0
```

Первый параметр метода **move**, как и у любого метода класса в Python, называется **self** (ссылка на сам объект), других параметров нет, поэтому при вызове этого метода никаких данных ему передавать не нужно.

В методе **move** вычисляется новая координата **X** машины и, если она находится за пределами дороги, эта координата устанавливается в ноль (машина появляется слева на той же полосе). Изменение координаты при равномерном движении описывается формулой

$$X = X_0 + V \cdot \Delta t\,,$$

где X_0 и X – начальная и конечная координаты, V – скорость, а Δt – время движения. Вспомним, что любое моделирование физических процессов на компьютере происходит в дискретном времени, с некоторым интервалом дискретизации. Для простоты мы измеряем время в этих интервалах, а за скорость V принимаем расстояние, проходимое машиной за один интервал. Тогда метод **move** описывает изменение положения машины за один интервал ($\Delta t = 1$).

Основная программа

После определения классов в основной программе создаем массив (список) объектов-машин, каждую из них «связываем» с ранее созданным объектом «*Дорога*»:

```
N = 3
cars = []
for i in range(N):
  cars.append( TCar(road, i+1, 2*(i+1)) )
```

При вызове конструктора класса **TCar** задаются три параметра: адрес объекта «дорога» (его нужно создать до выполнения этого цикла), номер полосы и скорость. В приведенном варианте машина с номером **i** идет по полосе с номером **i+1** (считаем,

что полосы нумеруются с единицы) со скоростью **2·(i+1)** единиц за один интервал моделирования.

Сам цикл моделирования получается очень простой: на каждом шаге вызывается метод **move** для каждой машины:

```
for k in range(100):    # 100 шагов
  for i in range(N):    # для каждой машины
    cars[i].move()
```

В данном случае выполнено 100 шагов моделирования. Теперь можно вывести на экран координаты всех машин:

```
print ( "После 100 шагов:" )
for i in range(N):
  print ( cars[i].X )
```

Можно ли было написать такую же программу, не используя объекты? Конечно, да. И она получилась бы короче, чем наш объектный вариант (с учетом описания классов). В чем же преимущества ООП? Мы уже отмечали, что ООП – это средство разработки больших программ, моделирующих работу сложных систем. В этом случае очень важно, что при использовании объектного подхода

• объекты реального мира проще всего описываются именно с помощью понятий «объект», «свойства», «действия» (методы);

• основная программа, описывающая решение задачи в целом, получается простой и понятной; все команды напоминают действия в реальном мире («машина № 2, вперед!»);

• разработку отдельных классов объектов можно поручить разным программистам, при этом каждый может работать независимо от других;

• если объекты *Дорога* и *Машина* понадобятся в других разработках, можно будет легко использовать уже готовые классы.