при поддержке
{а}вижн

Лекция 16

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Атрибуты

Повтор: методы и функции

В Python различают:

- *Функции* — они рассматриваются с начала курса и
- *Связанные методы* — объединяют функции с объектами, в которых осуществляется вызов.

Объект + Функция = Связанный метод

```
>>> type(Account.deposit)
<class 'function'>
>>> type(vasya_account.deposit)
<class 'method'>
```

```
>>> Account.deposit(vasya_account, 656)
666
>>> vasya_account.deposit(671)
1337
```

Функция: все аргументы в скобочках

Метод: Один объект до точки и потом аргументы в скобочках

Терминология: атрибуты, функции, методы

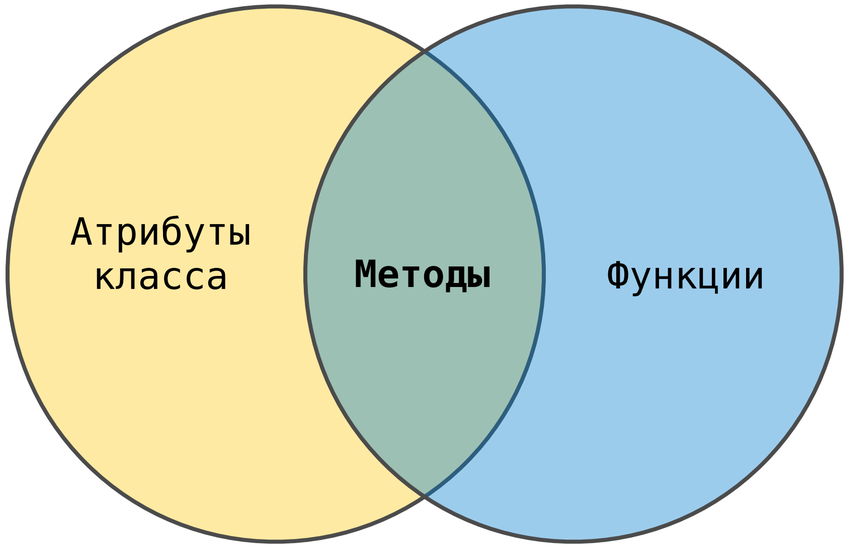
У всех объектов есть атрибуты (пары ключ-значение)

Классы — это тоже объекты, то есть у них есть атрибуты

Атрибут экземпляра — тот, что принадлежит экземпляру класса (объекту)

Атрибут класса — тот, что принадлежит непосредственно классу

Терминология:



Объектная система Python:

Функции — это объекты.

Связанные методы — тоже объекты: функция имеющая первый параметр «self» связана с экземпляром.

Выражение с точкой возвращает связанный с атрибутом класса метод, являющийся функцией.

<экземпляр>.<имя_метода>

Повторение: поиск атрибутов по имени

<выражение> . <имя>

При выполнении выражения с точкой:

1. Выполнить <выражение> слева от точки, которое вернёт объект.
2. В атрибутах объекта ищется совпадение <имени>; если атрибут с таким именем существует, то возвращается его значение.
3. В противном случае, <имя> ищется в классе с возвратом значения атрибута.
4. Если это функция, то вместо функции Python вернет связанный метод.

(Пример)

Наследование

Наследование

Наследование – это вид взаимосвязи классов друг с другом.

Применяется, если два похожих класса несколько отличаются друг от друга.

Вместе с атрибутами базового класса класс-наследник может содержать другие атрибуты, обеспечивающие некоторое дополнительное поведение.

```
class <Имя>(<Базовый класс>):
    <набор>
```

Концептуально, новый подкласс наследует атрибуты базового класса.

Подкласс может переопределять некоторые наследуемые атрибуты.

При наследовании реализуются только отличия между базовым классом и классом-наследником.

Пример наследования

Класс CheckingAccount – особый вид класса Account

```
>>> ch = CheckingAccount('Вася')
>>> ch.interest     # Меньший процент для чекового счета
0.01
>>> ch.deposit(20)  # Пополнение точно такое же
20
>>> ch.withdraw(5)  # Снятие включает дополнительную комиссию в 1 ₽
14
```

Почти всё поведение такое же как в базовом классе Account

```
class CheckingAccount(Account):
    """Банковский счет с комиссией за снятие."""
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

или

```
        return super().withdraw(      amount + self.withdraw_fee)
```

Поиск имён атрибутов в классах

Атрибуты базового класса *не копируются* в подклассы!

Поиск имени в классе:

1. Если это имя атрибута класса, вернуть значение атрибута.

2. Иначе, искать имя в базовом классе, если такой имеется.

```
>>> ch = CheckingAccount('Вася')  # Вызывает Account.__init__
>>> ch.interest      # Найдено в классе CheckingAccount
0.01
>>> ch.deposit(20)  # Найдено в классе Account
20
>>> ch.withdraw(5)  # Найдено в классе CheckingAccount
14
```

(Пример)

Объектно-ориентированный подход

Проектирование наследования

Не повторяй себя (DRY); используй готовые реализации.

Переопределённые атрибуты остаются доступными через класс.

Используй атрибуты экземпляров всегда, когда возможно.

```
class CheckingAccount(Account):
    """Банковский счет с комиссией за снятие."""
    withdraw_fee = 1
    interest = 0.01
    def withdraw(self, amount):
        return Account.withdraw(self, amount + self.withdraw_fee)
```

Поиск имени атрибута в базовом классе

Предпочтительнее, чем CheckingAccount.withdraw_fee

Наследование и композиция

Объектно-ориентированный подход даёт результаты при использовании метафоры.

Наследование подходит для описания связей «is-a» (является):

- например, чековый счет *является* особым видом счета
- то есть, `CheckingAccount` наследует классу `Account`

Композиция подходит для описания связей «has-a» (имеет)

- например, банк *имеет* набор счетов, которыми управляет
- то есть, у объекта `bank` есть список счетов в виде атрибута.

(Пример)

Множественное наследование

Множественное наследование

```
class SavingsAccount(Account):
    deposit_fee = 2
    def deposit(self, amount):
        return Account.deposit(self, amount - self.deposit_fee)
```

В Python класс может наследовать нескольким базовым классам

Директор по маркетингу банка CleverBank придумал идею:

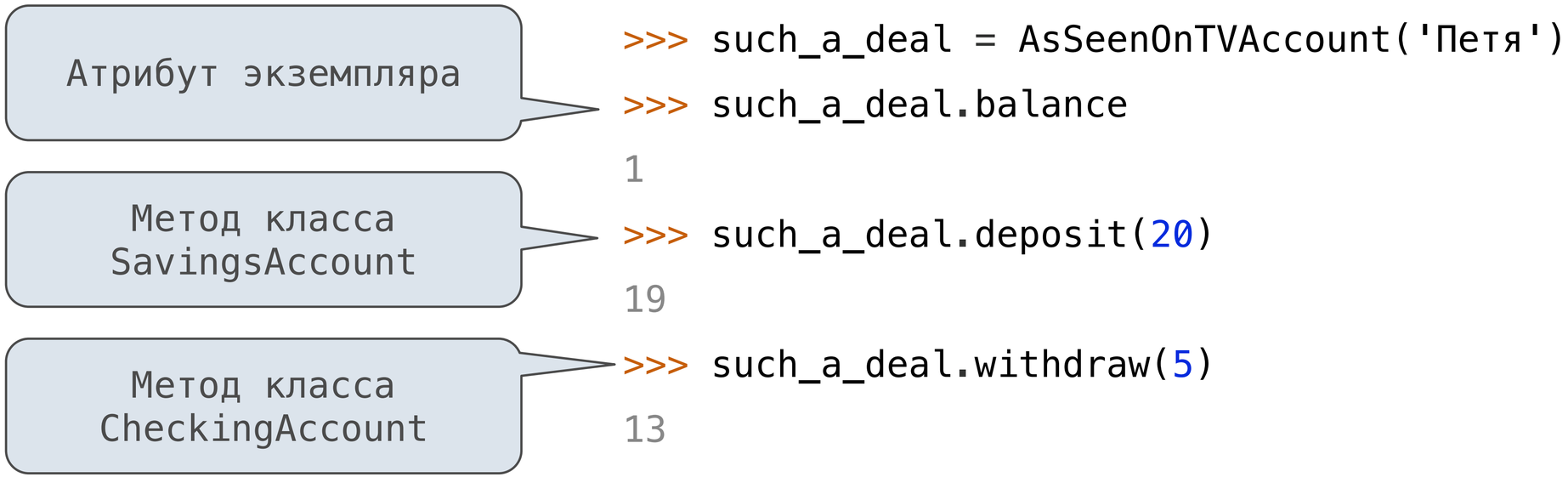
- Низкий процент: 1%
- Комиссия за снятие: 1 ₽
- Комиссия за пополнение: 2 ₽
- При открытии счета 1 ₽ в подарок

```
class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1                    # Подарок!
```

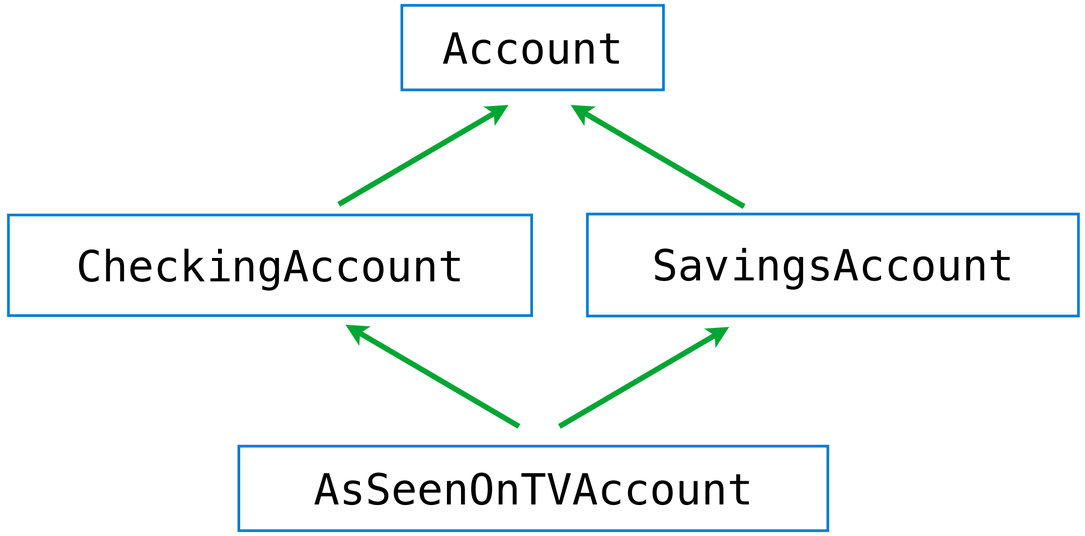
Множественное наследование

В Python класс может наследовать нескольким базовым классам.

```
class AsSeenOnTVAccount(CheckingAccount, SavingsAccount):
    def __init__(self, account_holder):
        self.holder = account_holder
        self.balance = 1                    # Подарок!
```



Неопределенность в именах атрибутов класса



Атрибут экземпляра

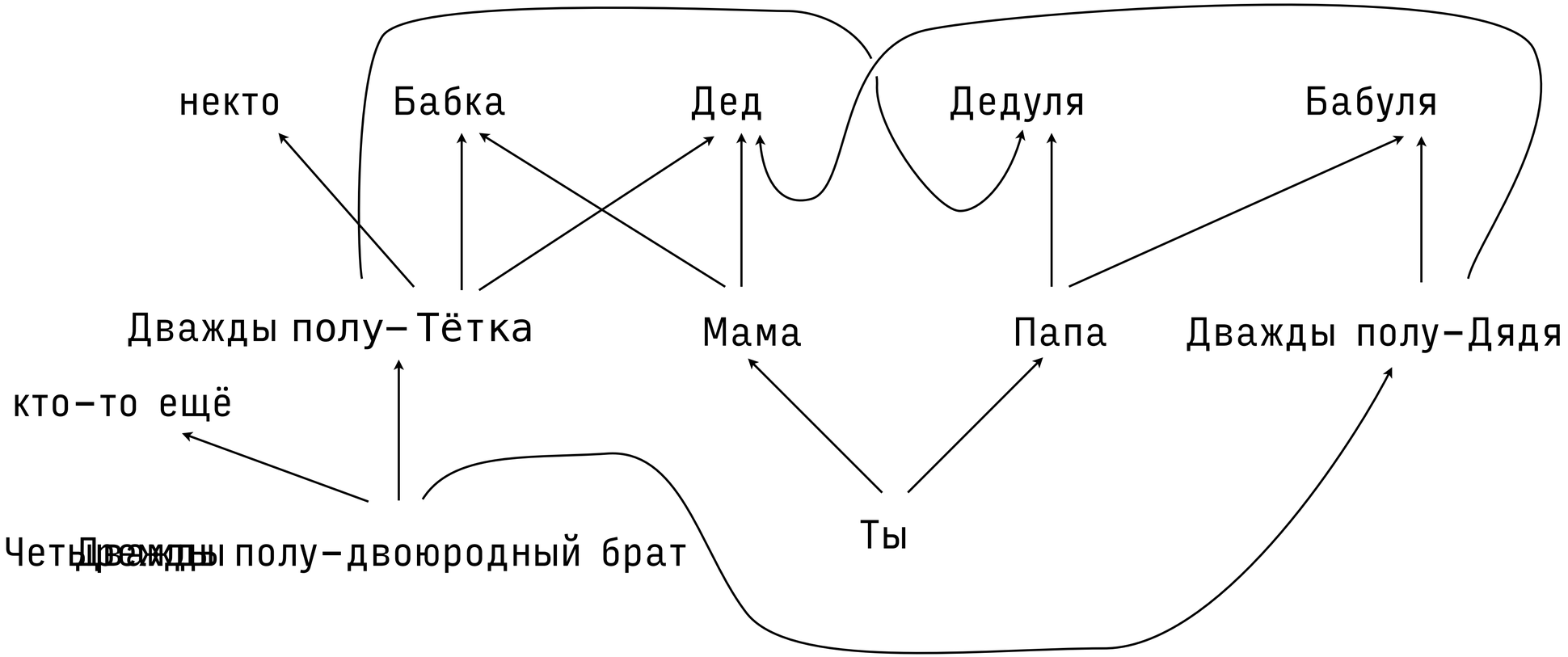
Метод класса SavingsAccount

Метод класса CheckingAccount

```
>>> such_a_deal = AsSeenOnTVAccount('Петя')
>>> such_a_deal.balance
1
>>> such_a_deal.deposit(20)
19
>>> such_a_deal.withdraw(5)
13
```

Сложное наследование

Биологическое наследование



Мораль: Наследование может быть сложным, так что применяй его с умом!