Объектно-ориентированное программирование на Python

Осипов Никита Алексеевич

ЛЕКЦИЯ 5. АБСТРАКТНЫЕ БАЗОВЫЕ КЛАССЫ

Учебные вопросы:

1. Понятие абстрактного класса.
2. Реализация наследования абстрактного класса.

Абстрактные базовые классы

- **Абстрактным** называется класс, который содержит один и более **абстрактных методов**
- Абстрактным называется объявленный, но не реализованный метод
- **Абстрактные классы**
  - не могут быть инстанциированы (нельзя создать экземпляр класса)
  - от них нужно наследовать свой класс, реализовать в нем все их **абстрактные методы** и только тогда можно создать экземпляр (своего) класса

Абстрактные базовые классы – реализация

oopAb.py

- Добавить в базовый класс методы по умолчанию, выбрасывающие исключение ***NotImplementedError***

```
class AbstractDocument :

    def __init__(self, name):
        self.name = name

    # Абстрактный метод
    def show(self):
        raise NotImplementedError("Subclass must implement abstract method")
```

Абстрактные базовые классы – реализация

oopAb.py

- Такое решение неполное: наследники могут не переопределить все методы базового класса

```
class PDF(AbstractDocument):
    # Переопределить метод родительского класса
    def show(self):
        print ("Show PDF document:", self.name)

class Word(AbstractDocument):
    pass  # метод не переопределен
```

```
documents = [ PDF("Python tutorial"),
              Word("Java IO Tutorial"),
              PDF("Python Date & Time Tutorial")]
for doc in documents:
    doc.show()
```

Проблема обнаружится только во время выполнения попытки вызова не переопределённого метода

Абстрактные базовые классы – реализация

oopAb_ABC.py

- С помощью модуля abc:
  - модуль определяет мета-класс и набор декораторов
  - абстрактные классы позволяет регистрировать существующие классы как часть своей иерархии

```
from abc import ABC, abstractmethod
class ChessPiece(ABC):
    # общий метод, который будут использовать все наследники этого класса
    def draw(self):
        print("Drew a chess piece")

    # абстрактный метод, который будет необходимо переопределять для каждого подкласса
    @abstractmethod
    def move(self):
        pass
```

Метод становится абстрактным, если он описан ключевым словом **@abstractmethod**

Абстрактные базовые классы – реализация

- Необходимо создать конкретный класс, например, класс ферзя, в котором требуется реализовать метод move()

```
class Queen(ChessPiece):
    def move(self):
        print("Moved Queen to e2e4")

# Можем создать экземпляр класса
q = Queen()
# И доступны все методы класса
q.draw()
q.move()
```

Абстрактные базовые классы – реализация

- Абстрактный метод может быть реализован сразу в абстрактном классе

Декоратор **abstractmethod** обяжет программистов, реализующих подкласс либо реализовать собственную версию абстрактного метода, либо дополнить существующую

Можно вызывать родительский метод при помощи **super**()

```
from abc import ABC, abstractmethod
class Basic(ABC):
    @abstractmethod
    def hello(self):
        print("Hello from Basic class")

class Advanced(Basic):
    def hello(self):
        super().hello()
        print("Enriched functionality")

a = Advanced()
a.hello()
```

ЛЕКЦИЯ 6. МНОЖЕСТВЕННОЕ НАСЛЕДОВАНИЕ

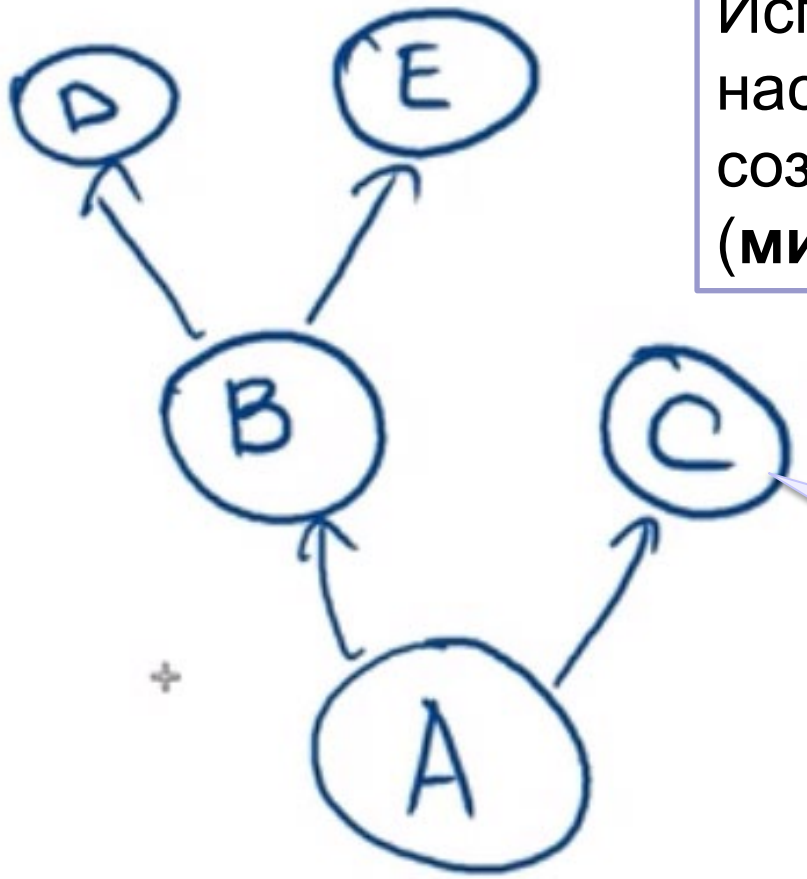
Учебные вопросы:

1. Реализация множественного наследования в Python.
2. Порядок разрешения методов (Method Resolution Order / MRO) в Python.

Множественное наследование

- Возможность у класса потомка наследовать функционал не от одного, а от нескольких родителей

```
class D: pass
class E: pass
class B(D, E): pass
class C: pass
class A(B, C): pass
```



Использование множественного наследования, позволяет создавать **классы-примеси (миксины)**

Можно вынести отдельный функционал в отдельный класс-миксин

Применение классов-миксинов

```
class Car:
    def ride(self):
        print("Riding a car")

    def play_music(self, song):
        print("Now playing: {} ".format(song))
```

Класс имеет несвойственную ему функциональность

```
class MusicPlayerMixin:
    def play_music(self, song):
        print("Now playing: {}".format(song))
```

Вынести функционал в отдельный класс-миксин

```
class Smartphone(MusicPlayerMixin):
    pass
```

Можно "домешивать" этот класс в любой, где нужна эта функция

Множественное наследование

oopMn1.py

- Для того чтобы понять для двух конкретных классов кто является предком, а кто является наследником кого-то, использовать функцию **issubclass**

```
issubclass(A, A) # True
issubclass(C, D) # False
issubclass(A, D) # True
issubclass(C, object) # True
issubclass(object, C) # False
```

- Функция **isinstance** позволяет узнать ведет ли себя объект как экземпляр какого-либо класса

```
x = A()
isinstance(x, A) # True
isinstance(x, B) # True
isinstance(x, object) # True
isinstance(x, str) # False
```

Ромбовидное наследование (The Diamond Problem)

- Если у нескольких родителей будут одинаковые методы?
- Какой метод в таком случае будет использовать наследник?
  - для определения порядка используется **алгоритм поиска в ширину,** то есть сначала интерпретатор будет искать метод hi в классе **B**, если его там нет - в классе **C**, потом **A**.

```
class A:
    def hi(self):
        print("A")
class B(A):
    def hi(self):
        print("B")
class C(A):
    def hi(self):
        print("C")
class D(B, C):
    pass

d = D()
d.hi()
```

oopMn1b.py

Ромбовидное наследование ("вызов следующего метода")

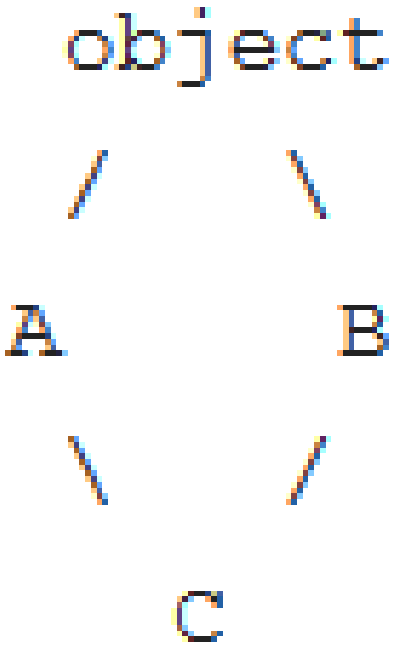
```
class A():
    def __init__(self):
        super().__init__()
        print("A")

class B():
    def __init__(self):
        super().__init__()
        print("B")

class C(A, B):
    def __init__(self):
        super().__init__()
        print("C")
```

А если убрать вызов базового класса

```
A
C
```



Какой будет порядок вызова конструкторов?

```
B
A
C
```

```
x = C()
```

oopMn2.py