**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Курсовая работа**

Тема: Наследование в Python

Студент: Хахин Максим

Группа: 80-303

Преподаватель: Семёнов А.С.

Дата:

Оценка:

Москва, 2021

1. Постановка задачи

Написать программу, в которой будет показаны разные свойства наследования.

1. Описание программы

В данной программе реализовано: простое наследование методов родительского класса, наследование с переопределением метода, множественное наследование, элементы расширения метода, лямбда функция, абстрактный класс, наследование абстрактного класса, делегат, конструктор с параметром, конструктор без параметра, абстрактный метод, метод прохода mro,

3. Сравнение python и c#

1) Интерфейсы. В питоне они отсутствуют 2) Единственность конструктора. В классе нельзя сделать альтернативный конструктор 3) Спецификация, Специализация, Конструирование, Конкретизация, Принцип подстановки. В питоне она отсутствует, так как это динамически типизированный язык 4) Так как в Python контейнер универсальный (не имеет типа), то методы будут выполняться того класса, которого они были созданы 5) Делегат в питон. Любая функция – делегат 6) Инкапсуляция отсутствует. Её можно легко обойти, поэтому она не имеет смысла.

4. Диаграмма классов

B

A

D

C

E

5. Диаграмма объектов

C.init()

Аналогично для объектов D и E.------🡪

c

m() - методы

C.dell()

6. Листинг программы

**from** abc **import** abstractmethod, ABC  
  
  
**class** A(ABC): *##Создаю абстрактный класс A* **def** entity(self):  
 print(**"я A"**)  
  
 @abstractmethod  
 **def** operate(self): *##Создаю абстрактный метод  
 """Вызов текущего положения"""* **pass  
  
  
class** B:  
  
 **def** \_\_init\_\_(self, \*args, \*\*kwargs): *##Создаю конструктор, который может получить на вход любое числно именованных и не именованных параметров* self.altitude = 0 *# метров* self.velocity = 0 *# км/ч* **def** take\_off(self):  
 self.altitude = 100  
 self.velocity = 300  
  
 **def** fly(self):  
 self.altitude = 5000  
  
 **def** land\_on(self):  
 self.altitude = 0  
 self.velocity = 0  
  
 **def** \_\_str\_\_(self): *##Переопределяю метод str, чтобы при печати элемента выводился заданный патерн строки* res = super().\_\_str\_\_()  
 **return** res + **' {} высота {} скорость {}'**.format(  
 self.\_\_class\_\_.\_\_name\_\_, self.altitude, self.velocity)  
  
  
**class** C(B, A): *## Создаю класс C, который наследуется от класса B и абстрактного класса A* **def** \_\_init\_\_(self, model): *##Создаю конструктор, который должен получить на вход один параметр model* super().\_\_init\_\_(model=model)  
 self.fuel = 100  
  
 **def** operate(self): *##Расширяю метод operate, наследуемую классом Drone из класса Robot* super().operate()  
 print(**"я F"**)  
  
 **def** \_\_del\_\_(self): *##Переопределение деструктора* print(**'C закончил работу'**)  
  
  
**class** D:  
  
 **def** \_\_init\_\_(self):  
 self.damage = **None** self.bullets = 0  
  
 **def** base(self, gun\_damage):  
 self.damage = gun\_damage  
 self.bullets = 100  
  
 **def** shot(self):  
 f = **lambda** q: q - 1 *## лямбда функция* **if** (self.bullets > 1):  
 print(**"Паф"**)  
 self.bullets = f(self.bullets) *##Использование лябды функции для вычитании дискретной единицы* print(**"Вы нанесли {} урона. У вас осталось {} патрон"**.format(self.damage, self.bullets))  
 **else**:  
 print(**"У вас закончились патроны"**)  
  
  
**class** E(C, D): *## Создаю класс E, который наследуется от класса C и класса D* **def** \_\_init\_\_(self, model, gun\_damage): *##Создаю конструктор, который должен получить на вход параметр model и параметр gun\_damage* super().\_\_init\_\_(model=model)  
 super().base(gun\_damage=gun\_damage)  
  
 **def** operate(self, func): *##Замещение метод operate, наследуемую классом WarDrone из класса Drone* print(**"я E"**)  
 func()  
  
 **def** \_\_del\_\_(self): *##Переопределение деструктора* print(**'E закончил работу'**)  
  
**class** F(C):*## Создаю класс F, который наследуется от класса C* **def** \_\_init\_\_(self, model):  
 super().\_\_init\_\_(model=model)  
  
 **def** delivery(self):  
 print(**"я F"**)  
  
 **def** \_\_del\_\_(self): *##Переопределение деструктора* print(**'F закончил работу'**)  
  
  
**def** pr():  
 print(**"say hello to my little friend"**)  
*##Принцип подстановки*c = C(model=**'Orbiter'**) *##Создаём объект родительского класса*print(c)  
c.take\_off()  
print(c)  
c.fly()  
print(c)  
c.operate()  
print(c)  
c.land\_on()  
print(c)  
print(**"########################################################################"**)  
c = F(model=**'Orbiter I'**) *##Заменяем на объект класса наследника*print(c)  
c.take\_off()  
print(c)  
c.fly()  
print(c)  
c.operate()  
print(c)  
c.land\_on()  
print(c)  
c.delivery()  
**if** (isinstance(c, C)): *##Определение типа который в объект подставлен* print(**"Объект orbiter принадлежит классу C"**)  
**else**:  
 print(**"Объект orbiter не принадлежит классу C"**)  
**del**(c) *##Принудительный вызов деструктора  
##Конец принципа подстановки*print(**"########################################################################"**)  
e = E(model=**'Orbiter II'**, gun\_damage=27) *##Создаём объект класса с двумя параметрами*e.operate(pr) *##Метод с параметром*e.shot()  
e.shot()  
print(e)  
e.fly()  
print(e)  
print(e.\_\_class\_\_.\_\_mro\_\_) *##Демонстрация принципа обхода mro***if** (isinstance(e, C)): *##Определение типа который в объект подставлен* print(**"Объект e принадлежит классу C"**)  
**else**:  
 print(**"Объект e не принадлежит классу C"**)  
**if** (isinstance(e, F)): *##Определение типа который в объект подставлен* print(**"Объект e принадлежит классу F"**)  
**else**:  
 print(**"Объект e не принадлежит классу F"**)

7. Результаты выполнения программ

<\_\_main\_\_.C object at 0x0000023F19D10288> C высота 0 скорость 0  
<\_\_main\_\_.C object at 0x0000023F19D10288> C высота 100 скорость 300  
<\_\_main\_\_.C object at 0x0000023F19D10288> C высота 5000 скорость 300  
я F  
<\_\_main\_\_.C object at 0x0000023F19D10288> C высота 5000 скорость 300  
<\_\_main\_\_.C object at 0x0000023F19D10288> C высота 0 скорость 0  
*########################################################################*C закончил работу  
<\_\_main\_\_.F object at 0x0000023F19D107C8> F высота 0 скорость 0  
<\_\_main\_\_.F object at 0x0000023F19D107C8> F высота 100 скорость 300  
<\_\_main\_\_.F object at 0x0000023F19D107C8> F высота 5000 скорость 300  
я F  
<\_\_main\_\_.F object at 0x0000023F19D107C8> F высота 5000 скорость 300  
<\_\_main\_\_.F object at 0x0000023F19D107C8> F высота 0 скорость 0  
я F  
Объект orbiter принадлежит классу C  
F закончил работу  
*########################################################################*я E  
say hello to my little friend  
Паф  
Вы нанесли 27 урона. У вас осталось 99 патрон  
Паф  
Вы нанесли 27 урона. У вас осталось 98 патрон  
<\_\_main\_\_.E object at 0x0000023F19D10288> E высота 0 скорость 0  
<\_\_main\_\_.E object at 0x0000023F19D10288> E высота 5000 скорость 0  
(<**class '\_\_main\_\_.E'**>, <**class '\_\_main\_\_.C'**>, <**class '\_\_main\_\_.B'**>, <**class '\_\_main\_\_.A'**>, <**class 'abc.ABC'**>, <**class '\_\_main\_\_.D'**>, <**class 'object'**>)  
Объект e принадлежит классу C  
Объект e не принадлежит классу F  
E закончил работу  
  
Process finished **with** exit code 0

8. Вывод

Наследование в Python считается одним из наиболее важных аспектов ООП, поскольку оно служит функции повторного использования, что делает часть кода более надежной.

Все в Python является **объектом**. Модули — это объекты, определения классов и функции — это объекты, и, конечно, объекты, созданные из классов, тоже являются объектами.

Начиная с версии Python 3 в языке перестали существовать интерфейсы. Абстрактные классы создаются путем наследования класса ABC (библиотека abc).

Из-за динамической типизации некоторые парадигмы ООП работают специфично или не работаю вовсе.

Python очень прост в освоении и хорошо подходит для прототипирования, так как код на нём пишется намного быстрее, чем на низкоуровневых языках, однако он не подходит для высоконагруженных систем (Python интерпретируемый, а не компилируемый) или систем с высоким требованием к безопасности (из-за серьёзных проблем с инкапсуляцией).

9. Ссылки на источники

<https://qna.habr.com/q/237887>

<https://habr.com/ru/post/72757/>

<https://pythonru.com/primery/primery-raboty-s-klassami-v-python>

<https://docs.python.org/3/library/abc.html>