Парадигма наследования

Наследование — одна из самых важных и мощных парадигм ООП. При наследовании мы оперируем такими понятиями, как родительский класс и класс потомок. Класс-потомок наследует методы и переменные, определенные в родительском классе. Рассмотрим на примере:

```
class A:
    var_A = 1

    def method_A(self):
        print("A")


class B(A):
    var_B = 2

    def method_B(self):
        print("B")


class C(B):
    var_C = 3

    def method_C(self):
        print("C")


print("A:\t", list(filter(lambda x: "__" not in x, dir(A))))
print("B(A):\t", list(filter(lambda x: "__" not in x, dir(B))))
print("C(B):\t", list(filter(lambda x: "__" not in x, dir(C))))
print()
```

При наследовании класс-потомок может переопределять методы и переменные родительского класса:

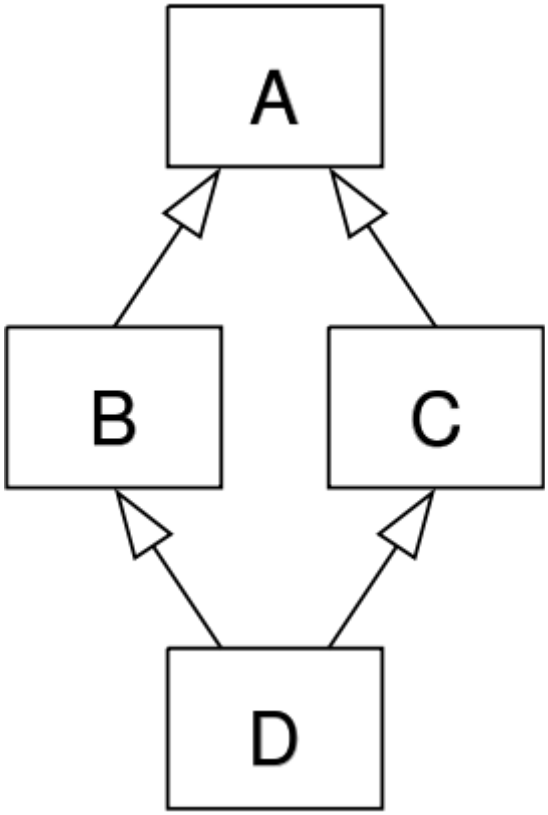
```
class A:
    value = 13

    def some_method(self):
        print(f"Method in A, value = {self.value}")


class B(A):

    def some_method(self):
        print(f"Method in B, value = {self.value}")


class C(B):
```

```
40      value = 6
41
42      def some_method(self):
43          print(f"Method in C, value = {self.value}")
44
45
46  A().some_method()
47  B().some_method()
48  C().some_method()
49  print()
```

Множественное наследование позволяет получить доступ к атрибутам нескольких родительских классов:

```
50  class A:
51
52      def some_function(self):
53          print("First function")
54
55      def other_function(self):
56          print("Second function")
57
58
59  class B:
60
61      def method_in_B(self):
62          print("Third function")
63
64
65  class C(A, B):
66
67      pass
68
69
70  # Посмотрим все атрибуты класса, не являющиеся служебными
71  print("A:\t", list(filter(lambda x: "__" not in x, dir(A))))
72  print("B:\t", list(filter(lambda x: "__" not in x, dir(B))))
73  print("C(A,B):\t", list(filter(lambda x: "__" not in x, dir(C))))
74  print()
```

Одна из проблем множественного наследования — Ромб Смерти (Diamond Inheritance).



При этом класс-наследник не знает, какую из реализаций некоторого метода следует выбрать. Эта проблема решается при помощи виртуального наследования. При этом вместо наследования реализации метода в класс-потомок передается ссылка на метод родительского класса. Однако, если классы-потомки первого уровня оба переопределяют некоторый метод, все еще остается вопрос, какая из реализаций должна передаваться потомку второго уровня. Для этого в Python версии 2.3 и выше применяется механизм C3-линеаризации. Рассмотрим несколько примеров.

```
75  # Пример без переопределения метода
76  class A:
77      value = 13
78
79      def some_method(self):
80          print(f"Method in A, value = {self.value}")
81
82
83  class B(A):
84      pass
85
86
87  class C(A):
88      pass
89
90
91  class D(B, C):
92      pass
```

```
93
94  # Рассмотрим реализацию в D
95  D().some_method()
96  print()
97
98
99  # Переопределим метод в D
100 class A:
101     value = 13
102
103     def some_method(self):
104         print(f"Method in A, value = {self.value}")
105
106
107 class B(A):
108     pass
109
110
111 class C(A):
112     pass
113
114
115 class D(B, C):
116
117     def some_method(self):
118         print(f"Method in D, value = {self.value}")
119
120 # Рассмотрим реализацию в D
121 D().some_method()
122 print()
123
124
125 # Переопределим метод в C
126 class A:
127     value = 13
128
129     def some_method(self):
130         print(f"Method in A, value = {self.value}")
131
132
133 class B(A):
134     pass
135
136
137 class C(A):
138
139     def some_method(self):
140         print(f"Method in C, value = {self.value}")
141
142 class D(B, C):
143     pass
144
```

```
145 # Рассмотрим реализацию в D
146 D().some_method()
147 print()
148
149
150 # Переопределим метод в  B и C b значение в C
151 class A:
152     value = 13
153
154     def some_method(self):
155         print(f"Method in A, value = {self.value}")
156
157
158 class B(A):
159
160     def some_method(self):
161         print(f"Method in B, value = {self.value}")
162
163
164 class C(A):
165     value = 6
166
167     def some_method(self):
168         print(f"Method in C, value = {self.value}")
169
170 class D(B, C):
171     pass
172
173 # Рассмотрим реализацию в D
174 D().some_method()
175 print()
```