Αντικειμενοστραφής Προγραμματισμός - Μέρος 3ο

Μιχάλης Δρακόπουλος

Ιούνιος 2020

ΚΛΗΡΟΝΟΜΙΚΟΤΗΤΑ

Ιεραρχία κλάσεων με σκοπό την **εξειδίκευση**

Σχέση κληρονομικότητας Η κλάση `B` (**υποκλάση**) είναι μια εξειδίκευση της κλάσης `A` (**υπερκλάση**)

- Η `B` **κληρονομεί** τα ιδιοχαρακτηριστικά και τις μεθόδους της `A`
- Η `B` μπορεί να προσθέσει νέα ιδιοχαρακτηριστικά και μεθόδους που την εξειδικεύουν σε σχέση με την `A`
- Η `B` μπορεί να **υπερκαλύψει** μεθόδους της `A`: να έχει μεθόδους με το *ίδιο όνομα* οι οποίες χρησιμοποιούνται για εξειδικευμένες ανάγκες της.

Η υπερκλάση `Person`

[1]:
```
class Person:

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

    def get_name(self):
        return self.name

    def __str__(self):
        return self.name
```

Η υποκλάση `Student`

[2]:
```
class Student(Person):

    def __init__(self, name, AM):
        Person.__init__(self, name)
        self.AM = str(AM)
        self.courses = {}

    def __str__(self):
        return Person.__str__(self) + ' (' + self.AM + ')'

    def get_AM(self):
        return self.AM
```

```
    def add_course_grade(self, course, grade):
        self.courses[course] = int(grade)

    def show_grades(self):
        for course in self.courses:
            print(course, self.courses[course])

    def get_MO(self):
        return sum(self.courses.values())/len(self.courses)
```

[3]:
```
s1 = Student('Mary', 1112201900035)
s1.add_course_grade('251', 8)
s1.add_course_grade('141', 9)
s1.add_course_grade('617', 6)
```

[4]:
```
s1.get_name()
```

[4]:
```
'Mary'
```

[5]:
```
s1.show_grades()
```

```
251 8
141 9
617 6
```

[6]:
```
s1.get_MO()
```

[6]:
```
7.666666666666667
```

Σύνθεση-έχει, Κληρονομικότητα-είναι

ΠΟΛΥΜΟΡΦΙΣΜΟΣ

Η συμπεριφορά μιας υπερκαλυμμένης μεθόδου εξαρτάται από το είδος των δεδομένων της.

```
# method() overriden in subclass

subclass_object.method()      # call subclass method

superclass_object.method()    # call superclass method
```

[7]:
```
class Player:

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

    def get_name(self):
        return self.name
```

```
    def job(self):
        print('I play games')
```

[8]:
```
class FootballPlayer(Player):

    def __init__(self, name, goals):
        Player.__init__(self, name)
        self.total_goals = goals

    def get_goals(self):
        return self.total_goals

    def job(self):
        print('I play football')
```

[9]:
```
class BasketballPlayer(Player):

    def __init__(self, name, points):
        Player.__init__(self, name)
        self.total_pointss = points

    def get_points(self):
        return self.total_goals

    def job(self):
        print('I play basketball')
```

[10]:
```
def player_sport(x):
    print('I am', x.get_name(), end = ':')
    x.job()

def main():
    p = Player('John Doe')
    f = FootballPlayer('Messi', 800)
    b = BasketballPlayer('Gallis', 24805)

    player_sport(p)
    player_sport(f)
    player_sport(b)

main()
```

```
I am John Doe:I play games
I am Messi:I play football
I am Gallis:I play basketball
```

Η `player_sport` δουλεύει μόνο με αντικείμενα που έχουν μεθόδους `get_name` και `job`

[11]:
```
def player_sport(x):
    if isinstance(x, Player):
        print('I am', x.get_name(), end = ':')
        x.job()
    else:
        print('Not a player')
```

[12]:
```
p = Player('John Doe')
f = FootballPlayer('Messi', 800)
b = BasketballPlayer('Gallis', 24805)
```

[13]:
```
isinstance(p, Player)
```

[13]: True

[14]:
```
isinstance(f, Player)
```

[14]: True

[15]:
```
isinstance(b, Player)
```

[15]: True

[16]:
```
isinstance(f, FootballPlayer)
```

[16]: True

[17]:
```
isinstance(f, BasketballPlayer)
```

[17]: False

[18]:
```
isinstance(p, FootballPlayer)
```

[18]: False