KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

Úvod do programovacích stylů ◇ poznámky k přednášce

3. Třídy

verze z 3. října 2023

Vnitřní stav objektů je rozdělen do pojmenovaných položek nazývaných *atributy*.

K vytváření nových tříd slouží příkaz `class`:

```
class Class:
    def __init__(self):
        self.atribute1 = None
        self.atribute2 = None
        ⋮
```

kde *`Class`* je jméno nové třídy a *`atribute1`*, *`atribute2`*, ... jsou názvy atributů. Vytvořením instance třídy *`Class`* vznikne objekt, který bude mít atributy *`atribute1`*, *`atribute2`*, ... Hodnotou každého atributu bude hodnota `None`.

Například třídu `Point` můžeme definovat takto:

```
class Point:
    def __init__(self):
        self.x = None
        self.y = None
```

Můžeme si vytvořit její instanci:

```
>>> point = Point()
>>> point
<__main__.Point object at 0x1056510f0>
```

Pokud *`object`* je objekt a *`attribute`* jeho atribut, pak

```
object.attribute
```

je výraz, jehož hodnota je hodnota atributu *`attribute`* objektu *`object`*. Například hodnota atributu `x` bodu `point`:

```
>>> point.x
>>>
```

Interpret nic nevytiskne, protože hodnota atributu `x` objektu `point` je `None`.

Pokud *`object`* je objekt, *`attribute`* jeho atribut a *`value`* hodnota, pak

```
object.attribute = value
```

je příkaz, který se vykoná tak, že nastaví hodnotu atributu *attribute* objektu *object* na hodnotu *value*.

Nastavíme atributy `x` a `y` objektu `point`:

```
>>> point.x = 3
>>> point.y = 4
```

Ověříme:

```
>>> point.x
3
>>> point.y
4
```

Objekt, kromě vnitřního stavu, obsahuje metody. *Metoda* je pojmenovaná funkce objektu.

Rozšíříme si definici třídy:

```
class Class:
    methods
```

Část *methods* je blok obsahující definice metod. Definice metody má tvar:

```
def method(self, parameter1, arameter2, ...):
    block
```

kde *method* je jméno metody, *parameter1*, *arameter2*, ... jsou proměnné a *block* je blok kódu. Definice třídy bude vždy začínat definicí metody `__init__`, která *inicializuje* instanci.

Instance třídy získá všechny metody, které třída definuje.

Rozšíříme si definici třídy `Point` o metody:

```
class Point:
    def __init__(self):
        self.x = 0
        self.y = 0

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y
```

```
    def set_x(self, x):
        self.x = x
        return self

    def set_y(self, y):
        self.y = y
        return self
```

a vytvoříme novou instanci:

```
>>> point = Point()
```

Pokud objektu *object* zašleme zprávu *message* s argumenty *arg1*, *arg2*, ..., pak se v objektu najde metoda jménem *message* a ta se zavolá s argumenty *object*, *arg1*, *arg2*, ... Vidíme, že příjemce zprávy se stává prvním argumentem volané funkce. Návratová hodnota funkce je návratovou hodnotou zaslání zprávy.

Tedy například pokud pošleme zprávu `set_x` objektu `point` s argumentem `3`, pak se najde jeho metoda `set_x`, která se zavolá s dvěma argumenty: `point` a `3`. Kód metody nastaví hodnotu atributu `x` objektu `point` na `3` a vrátí objekt `point`, který bude i návratovou hodnotou zaslání zprávy:

```
>>> point.set_x(3)
<__main__.Point object at 0x10bb8fc10>
```

Podobně zaslání zprávy `get_x` bez argumentu objektu `point` vede k zavolání jeho metody `get_x` s argumentem `point`. Metoda vrátí hodnotu atributu `x` objektu `point`:

```
>>> point.get_x()
3
```

Metoda, která se pro zaslanou zprávu v objektu nalezne, se nazývá *obsluha zprávy*. Procesu vykonávání těla obsluhy zprávy se říká *obsloužení zprávy*. Můžeme například říci, že obsloužení zprávy `set_x` způsobilo změnu atributu `x`.

Budeme dodržovat *princip zapouzdření*:

> Hodnoty atributů objektu smí přímo číst a měnit pouze metody tohoto objektu.

Ostatní můžou k datům objektu přistupovat přes jeho vlastnosti. Instancím třídy `Point` jsme definovali vlastnosti `x` a `y`.

Nově vytvořená instance třídy `Point` není v konzistentním stavu. Zprávy `get_x` a `get_y` vrací nepřípustnou hodnotu `None`:

```
>>> point = Point()
>>> point.get_x()
>>> point.get_y()
>>>
```

Objekt `point` nereprezentuje žádný geometrický bod v rovině.

Aby mohly být nově vytvořené instance rovnou v konzistentním stavu, rozšíříme si definici inicializace instance (metodu `__init__`):

```
def __init__(self):
    self.atribute1 = expr1
    self.atribute2 = expr2
    ⋮
```

Části *expr1*, *expr2*, ... jsou výrazy. Atributy *atribute1*, *atribute2*,... nově vytvořené instance třídy budou mít hodnoty odpovídající hodnotám výrazů *expr1*, *expr2*, ... Výrazy se vyhodnocují postupně.

Změníme definici inicializace instance třídy `Point`:

```
def __init__(self):
    self.x = 0
    self.y = 0
```

Nově vytvořené instance reprezentují bod o souřadnicích $(0, 0)$:

```
>>> point = Point()
>>> point.get_x()
0
>>> point.get_y()
0
```

Bod můžeme snadno uvést do nekonzistentního stavu:

```
>>> point = Point()
>>> point.set_x("1")
<__main__.Point object at 0x7f906a3cb9d0>
>>> point.get_x()
'1'
```

Chyba se může projevit úplně v jiném místě programu za nějaký čas:

```
>>> point.get_x() + 5
Traceback (most recent call last):
  File "<pyshell#24>", line 1, in <module>
    point.get_x() + 5
TypeError: can only concatenate str (not "int") to str
```

Lepší by bylo nedovolit nastavit hodnotu atributu na nečíselnou hodnotu.

Změníme definice metod třídy `Point`:

```
def set_x(self, x):
    if type(x) != int:
        raise TypeError("x coordinate of a point should be an integer")
    self.x = x
    return self

def set_y(self, y):
    if type(y) != int:
        raise TypeError("y coordinate of a point should be an integer")
    self.y = y
    return self
```

Nyní se již chyba projeví v momentě, kdy se pokusíme uvést objekt do nekonzistentního stavu:

```
>>> point = Point()
>>> point.set_x(1)
<__main__.Point object at 0x7fd0c62f8100>
>>> point.get_x()
1
>>> point.set_x("1")
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    point.set_x("1")
  File "point.py", line 34, in set_x
    raise TypeError("x coordinate of a point should be an integer")
TypeError: x coordinate of a point should be an integer
>>> point.get_x()
1
```

Zavedeme si třídu pro popisky:

```
class Label:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.text = ""

    def get_x(self):
        return self.x

    def get_y(self):
        return self.y

    def set_x(self, x):
        if type(x) != int:
            raise TypeError("x coordinate of a label should be an integer")
```

```
        self.x = x
        return self

    def set_y(self, y):
        if type(y) != int:
            raise TypeError("y coordinate of a label should be an integer")
        self.y = y
        return self

    def get_text(self):
        return self.text

    def set_text(self, text):
        if not type(text) == str:
            raise TypeError("text of a label should be a string")
        self.text = text
        return self
```

Popisek je podobný bodu (má také vlastnosti `x` a `y`), ale navíc má vlastnost `text`.

Dále si zavedeme třídu pro skupiny ovládacích prvků.

```
class Group:
    def __init__(self):
        self.items = []

    def get_items(self):
        return self.items[:]

    def set_items(self, items):
        for item in items:
            if not (isinstance(item, Label)
                    or isinstance(item, Group)):
                raise TypeError("items of a group have to be an array of widgets")
        self.items = items[:]
        return self
```

Popisky i skupiny jsou ovládací prvky. Každý ovládací prvek by mělo jít posunout. Přesněji by každý ovládací prvek měl rozumět následující zprávě.

widget`.move(`*dx*`, `*dy*`)`

widget: ovládací prvek
dx, *dy*: čísla

Přidáme obsluhu zprávy `move` do třídy `Label`:

```
def move(self, dx, dy):
    self.set_x(self.get_x() + dx)
    self.set_y(self.get_y() + dy)
    return self
```

a do třídy `Group`:

```
def move(self, dx, dy):
     for item in self.get_items():
         item.move(dx, dy)
     return self
```

Tedy posunout popisek znamená změnit jeho kartézské souřadnice a posun skupiny uskutečníme tak, že posuneme každý prvek ve skupině. Vyzkoušíme:

```
>>> label = Label()
>>> label.move(10, 20)
<__main__.Label object at 0x10f3bcb90>
>>> print(label.get_x(), label.get_y())
10 20
>>> label2 = Label()
>>> group = Group().set_items([label, label2])
>>> group.move(5, 10)
<__main__.Group object at 0x10f906f50>
>>> print(label.get_x(), label.get_y())
15 30
>>> print(label2.get_x(), label2.get_y())
5 10
```

Posun popisku i skupiny provedeme v obou případech zasláním zprávy `move`. Ovšem v každém z případů se zavolá jiná metoda zodpovědná za posun.

Právě jsme si demonstrovali obecný princip polymorfizmu v objektovém programování:

> Různé třídy mohou mít definovány pro tutéž zprávu různé metody.