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

6. Prototypy

verze z 31. října 2024

V této přednášce se seznámíme s objektovým stylem programování založeným na prototypech. Na rozdíl od stylu založeného na třídách zde neexistuje pojem třídy.

1 Základy

Programování v objektovém stylu založeném na prototypech umožňuje knihovna `proto` nacházející se v přiloženém souboru `proto.py`. Manuál ke knihovně se nalézá v souboru `proto.pdf`. Knihovnu importujeme příkazem:

```
from proto import obj
```

Importovali jsme pouze proměnnou `obj` obsahující takzvaný **základní** objekt. Na začátku je to jediný objekt, který máme k dispozici.

Tisk každého objektu obsahuje jeho název a jednoznačný identifikátor:

```
>>> obj
<obj at 0x10433c610>
```

Stejně jako dříve můžeme objektu **zaslat zprávu** výrazem:

```
receiver.message(arg1, arg2, ...)
```

Například název objektu dostaneme zasláním zprávy `name` bez argumentů:

```
>>> obj.name()
'obj'
```

Pojmy **příjemce zprávy**, **argumenty zprávy**, **rozumění zprávě** a **protokol objektu** přejmeme z dřívějších přednášek. Můžeme tedy například říci, že `obj` je příjemcem zprávy `name`.

Protože v objektovém stylu založeném na prototypech neexistují třídy, nemůžeme vytvářet nové objekty jejich instanciací. Nové objekty vznikají **klonováním** již existujících objektů. Objekty, které jsou určené ke klonování, se nazývají **prototypy**. **Klon** prototypu obdržíme tak, že mu zašleme zprávu `clone` bez argumentů. Například bod můžeme vytvořit klonem základního objektu:

```
>>> point = obj.clone()
>>> point
<obj at 0x104333c50>
```

Všimněte si, že bod má stejný název jako základní objekt, ale liší se identifikátorem.

Název objektu můžeme změnit zasláním zprávy `set_name`, kde jako argument uvedeme požadovaný název. Přejmenujeme bod:

```
>>> point.set_name("point")
<point at 0x104333c50>
```

Prototyp budeme vždy uchovávat v proměnné téhož jména. Například prototyp bodu s názvem `"point"` nalezneme v proměnné `point`.

Pokud řekneme, že objekt **má vlastnost** ***property***, pak tím myslíme, že rozumí zprávám ***property*** a `set_`***property***. Zpráva ***property*** se zasílá bez argumentu a vrací hodnotu vlastnosti. Zpráva `set_`***property*** nastaví hodnotu vlastnosti na jediný argument uvedený při jejím zaslání a vrací příjemce. Jediný rozdíl oproti vlastnostem objektů ze stylu založeném na třídách je v tom, že zpráva, která čte vlastnost má stejný název jako vlastnost.

Například `name` je vlastnost určující jméno objektu. Pokud objekt nerozumí zprávě pro nastavení hodnoty vlastnosti, pak říkáme, že je vlastnost **jen pro čtení**.

Novou vlastnost definujeme zasláním zprávy

```
object.add_slot(property)
```

kde ***property*** je název vlastnosti zadaný jako řetězec. Výchozí hodnota vlastnosti je hodnota `None`.

Přidáme bodu vlastnost `x` určující jeho $x$ovou souřadnici.

```
>>> point.add_slot("x")
<point at 0x104333c50>
>>> point.x()
>>> point.set_x(3)
<point at 0x104333c50>
>>> point.x()
3
```

2 Sloty

Vnitřní stav objektu je určen pojmenovanými položkami, které se nazývají **sloty**.

Vytvoříme si bod:

```
point = obj.clone().set_name("point")
```

Objekt `point` má slot `name` s hodnotou `"point"`.

Pokud objektu zašleme zprávu, která se shoduje s názvem jeho slotu, jako návratovou hodnotu dostaneme hodnotu slotu. Tedy:

```
>>> point.name()
'point'
```

Zpráva

```
object.set_slot(name, value)
```

nastaví hodnotu slotu. V případě neexistence slotu daného názvu dojde k jeho vytvoření.

Přidáme $x$ovou souřadnici bodu:

```
>>> point.set_slot("x", 2)
<point at 0x106c8ec50>
>>> point.x()
2
```

Hodnotu slotu můžeme určit výpočtem. Uvažujme funkci:

```
def point_y(resend, self):
    return self.x()
```

kterou dáme do slotu `y` bodu:

```
point.set_slot("y", point_y)
```

Funkci, která je hodnotou slotu, říkáme **metoda**. Tedy `point_y` je metoda bodu. Jména funkcí metod budou vždy začínat názvem objektu, ve kterém se nacházejí, následovaným podtržítkem a názvem slotu, kde je metoda uložena. Tedy jméno funkce pro metodu uloženou ve slotu *`method`* objektu *`object`* je *`object_method`*.

Pokud objektu zašleme zprávu:

```
receiver.message(arg1, arg2, ...)
```

a hodnotou slotu *`message`* je metoda, pak je metoda zavolána s argumenty *`resend`*, *`receiver`*, *`arg1`*, *`arg2`*, ... Argument *`resend`* vysvětlíme později.

Tedy `point.y()` způsobí zavolání funkce `point_y` s argumenty *`resend`* a `point`. Proto:

```
>>> point.y()
2
```

Změna slotu `x` způsobí změnu vlastností `x`, `y`.

```
>>> point.set_slot("x", 4)
<point at 0x106c8ec50>
>>> point.x()
4
>>> point.y()
4
```

Zpráva

object`.print_slots()`

vytiskne sloty příjemce. Například:

```
>>> point.print_slots()
parent: <obj at 0x10946d150>
name:   'point'
x:      2
y:      <function point_y at 0x10946a340>
```

Slot `parent` vysvětlíme později.

V textu tisk hodnot slotů můžeme učinit přehlednější tak, že nahradíme objekty a funkce v proměnných jejich názvy:

```
>>> point.print_slots()
parent: obj
name:   'point'
x:      2
y:      point_y
```

Zpráva

object`.add_slot(`*name*`)`

přesněji nastaví objektu hodnotu slotu *name* na `None` a hodnotu slotu `set_`*name* na metodu, která mění hodnotu slotu *name*.

Například pro nový bod:

```
>>> point = obj.clone().set_name("point")
>>> point.add_slot("x")
<point at 0x1026ed290>
>>> point.print_slots()
parent: obj
name:   'point'
x:      None
set_x:  <function object_add_slot.<locals>.set_method at 0x102842f20>
```

Metodu `set_x` za nás vytvoří systém. V textu budeme metodu nastavující hodnotu slotu *`slot`* zkracovat jen na `set_`*`slot`*`_method`:

```
>>> point.print_slots()
parent: obj
name:   'point'
x:      None
set_x:  set_x_method
```

Metodu nastavující hodnotu slotu si můžeme napsat sami.

```
point = obj.clone().set_name("point")

def point_set_x(resend, self, val):
    return self.set_slot("x", val)

point.set_slot("set_x", point_set_x)
```

Například dostáváme:

```
>>> point.print_slots()
parent: obj
name:   'point'
set_x:  point_set_x

>>> point.set_x(2)
<point at 0x110985410>
>>> point.print_slots()
parent: obj
name:  'point'
set_x:  point_set_x
x:      2
```

Přehledněji můžeme sloty objektu zakreslit do tabulky:

parent	obj
name	'point'
set_x	point_set_x
x	2

3 Dědičnost

Zpráva

```
object.clone()
```

vytvoří nový objekt s jediným slotem `parent` o hodnotě *`object`*.

Například:

```
>>> point = obj.clone()
>>> point.print_slots()
parent: obj
```

Objekt ve slotu `parent` se jmenuje **rodič**. Každý objekt kromě základního má rodiče. Objekt *`object1`* je **předkem** objektu *`object2`*, pokud *`object1`* je rodič *`object2`* nebo *`object2`* má rodiče *`parent`* a *`object1`* je předkem *`parent`*. Základní objekt je předkem všech ostatních objektů. Objekt *`object1`* **je typu** *`object2`*, pokud *`object2`* je předek *`object1`* nebo se jedná o týž objekt. Každý objekt je typu `obj`.

Rozebereme si detailněji mechanizmus zaslání zprávy. Po zaslání zprávy *`message`* objektu *`receiver`* s argumenty *`arg1`*, *`arg2`*, ... se nejprve hledá obsluha zprávy a poté se obsluha vyhodnotí. Pokud se nalezne obsluha zprávy, říkáme, že objekt zprávě **rozumí**, jinak zaslání zprávy skončí chybou.

Hledání obsluhy v objektu *`object`* probíhá následovně.

1. Pokud objekt *`object`* má slot *`message`*, pak je obsluhou hodnota slotu.
2. Pokud objekt *`object`* nemá slot *`message`*, ale má rodiče, pak se rekurzivně hledá obsluha zprávy v rodiči.
3. Jinak zpráva nemá obsluhu.

Vyhodnocení obsluhy *`handler`* nalezené v objektu *`owner`* probíhá následovně.

1. Pokud je *`handler`* funkce, pak se *`handler`* zavolá s argumenty *`resend`*, *`receiver`*, *`arg1`*, *`arg2`*, ... Hodnotu *`resend`* vysvětlíme později.
2. Jinak je hodnotou obsluhy přímo *`handler`*.

Například uvažujme objekty:

```
>>> point.add_slot("x")
<obj at 0x100f0d610>
>>> point.set_x(3)
<obj at 0x100f0d610>
>>> p1 = point.clone()
```

Objekty `p1` a `point` mají sloty:

```
>>> p1.print_slots()
parent: point

>>> point.print_slots()
parent: obj
x:      3
set_x:  set_x_method
```

Nyní objektu `p1` zašleme zprávu `x`. Nejprve se hledá obsluha zprávy. Systém se podívá, zda objekt `p1` má slot `x`. Protože se slot nenajde, pokračuje hledání obsluhy u jeho rodiče, kterým je objekt `point`. Ten má slot `x` s hodnotou 3. Proto je číslo 3 obsluhou zprávy `x` zaslanou objektu `p1`. Jelikož číslo 3 není metoda, je i návratovou hodnotou zaslané zprávy:

```
>>> p1.x()
3
```

Pokud objektu `p1` zašleme zprávu `set_x`, pak se obsluha najde v objektu `point`. Obsluhou je metoda, která se zavolá. Přestože se obsluha našla v objektu `point`, druhým argumentem volaní metody bude příjemce zprávy, tedy objekt `p1`. Proto zpráva `set_x` změní objekt `p1` a ne objekt `point`:

```
>>> p1.set_x(4)
<obj at 0x1007f4d90>
>>> p1.x()
4
>>> p1.print_slots()
parent: point
x:      4

>>> point.print_slots()
parent: obj
x:      3
set_x:  set_x_method
```

4 Přeposlání zprávy

Pokud objekt má slot *`message`* a jeho rodič rozumí zprávě *`message`*, pak říkáme, že objekt **přepisuje obsluhu** zprávy *`message`*.

Vytvoříme prototyp popisku `label` s vlastností `text`.

```
label = obj.clone().set_name("label")
label.add_slot("text")
label.set_text("")
```

Zde `label` přepisuje obsluhu zprávy `name`.

Dále vytvoříme prototyp okna `window` s vlastností `widget` pro jeho obsah.

```
window = obj.clone().set_name("window")
window.add_slot("widget")
```

Vytvoříme si prototyp okna s popiskem.

```
labeled_window = window.clone().set_name("labeled_window")
labeled_window.set_widget(label.clone())
```

Zavedeme konvenci, že objekty, které nejsou prototypy, budeme ukládat do proměnných, jejichž název končí číslem. Vytvoříme klon okna s popiskem:

```
labeled_window1 = labeled_window.clone()
```

Nastavením textu popisku klonu vzniká problém:

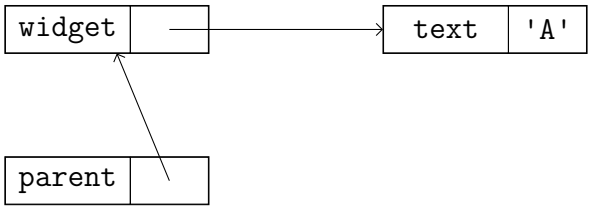
```
>>> labeled_window1.widget().set_text("A")
<label at 0x10a750f50>
>>> labeled_window.widget().text()
'A'
```

Vidíme, že došlo i k změně textu popisku u prototypu.

Zdroj problému se nachází v tom, že klon okna i jeho prototyp sdílí obsah:

```
>>> labeled_window.widget() is labeled_window1.widget()
True
```

Vztahy mezi objekty přehledně znázorňuje následující obrázek, kde tabulky jsou objekty a šipka znázorňuje hodnotu slotu. V tabulce jsou uvedeny jen pro příklad významné sloty.



Objekt vlevo nahoře je `labeled_window`, vlevo dole se nachází objekt `labeled_window1` a vpravo je sdílený obsah okna.

Potřebujeme změnit klonování okna tak, aby se během klonování naklonoval i jeho obsah. Tedy potřebujeme přepsat obsluhu klonování tak, aby nejprve provedla standardní klonování a poté naklonovala obsah okna. K tomu je potřeba umět přeposlat zprávu `clone` rodiči.

Uvažujme zaslání zprávy *message* objektu *receiver* s argumenty *arg1*, *arg2*, ... vedoucí k nalezení obsluhy *handler* nalezené ve slotu objektu *owner*, která je metodou. Jak již víme, metoda se zavolá s argumenty *resend*, *receiver*, *arg1*, *arg2*, ... Argument *resend* je funkce bez parametrů, která přepošle zprávu rodiči objektu *owner*.

Přeposlání zprávy objektu *object* probíhá tak, že se nejprve nalezne obsluha zprávy *message* v objektu *object* a poté se obsluha vyhodnotí.

V následující metodě tedy nejdříve naklonujeme okno standardně a poté mu změníme obsah na klon obsahu klonovaného okna.

```
def window_clone(resend, self):
    clone = resend()
    if self.widget():
        clone.set_widget(self.widget().clone())
    return clone

window.set_slot("clone", window_clone)
```

Znovu vytvoříme prototyp i jeho klon:

```
labeled_window = window.clone().set_name("labeled_window")
labeled_window.set_widget(label.clone())
labeled_window1 = labeled_window.clone()
```

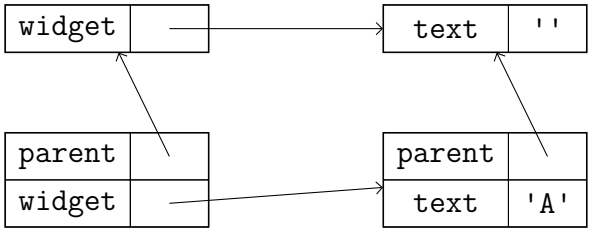
Nyní již k problému nedojde:

```
>>> labeled_window1.widget().set_text("A")
<label at 0x10dc45410>
>>> labeled_window.widget().text()
''
```

Klon již nesdílí obsah s prototypem:

```
>>> labeled_window.widget() is labeled_window1.widget()
False
```

Následující obrázek zachycuje vztahy mezi objekty.



Vidíme, že zde přibyl objekt nacházející se na obrázku vpravo dole, který je obsahem naklonovaného okna.

5 Prototypové uživatelské rozhraní

Knihovna Prototype Micro Widget (`pmw`) umožňuje definovat uživatelské rozhraní v prototypovém objektovém stylu. Manuál ke knihovně se nalézá v souboru `pmw.pdf`. Pro úvod se podívejte do programu `09_prototype_micro_widget.py`.

Objekty knihovny mohou zasílat události. Systém událostí má stejné chování jako ten v minulé přednášce. Tedy zaslání události *`event`* objektem *`sender`* s argumenty *`arg1`*, *`arg2`*, ... vede k zpracování události v objektu *`sender`*.

Zpracování události v objektu *`object`* probíhá ve dvou krocích:

1. Pokud objekt *`object`* rozumí zprávě *`event`*, pak je mu zaslána zpráva *`event`* s argumenty *`sender`*, *`arg1`*, *`arg2`*, ...
2. Pokud je za objekt *`object`* přímo zodpovědný objekt *`object2`*, pak pokračuje zpracování události v *`object2`*.

Vztah přímé zodpovědnosti jednoho objektu za druhý přebíráme z minulé přednášky. Tedy skupina je přímo zodpovědná za své prvky a okno je přímo zodpovědné za svůj obsah.

Otázky a úkoly na cvičení

1. Vytvořte prototyp počítadla `counter`. Objekty typu `counter` budou mít vlastnost `value` udávající hodnotu počítadla. Výchozí hodnota počítadla bude nula. Například:

```
>>> counter1 = counter.clone()
>>> counter1.value()
0
>>> counter1.set_value(2)
<counter at 0x10b4f96d0>
>>> counter1.value()
2
```

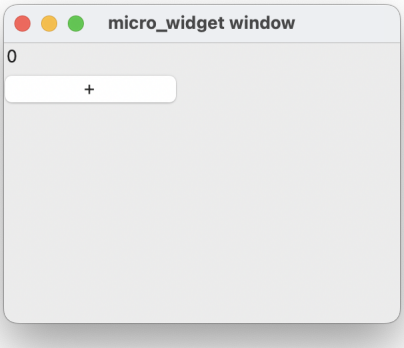
2. Zařiďte, aby se zasláním zprávy `inc` počítadlu zvedla jeho hodnota o jedna. Například:

```
>>> counter1 = counter.clone()
>>> counter1.inc()
<counter at 0x10a75ab10>
>>> counter1.value()
1
```

3. Vytvořte prototyp počítadla jako klon skupiny ovládacích prvků. Počítadlo bude tvořit popisek s hodnotou a tlačítko pro zvětšení jeho hodnoty. Například program:

```
window1 = window.clone()
counter1 = counter.clone()
window1.set_widget(counter1)
window1.display()
```

otevře okno:



4. Přidejte počítadlům vlastnosti pro čtení `label` a `button`, kde hodnotou vlastnosti je popisek a tlačítko počítadla. Například:

```
>>> counter1.label()
<label at 0x107f3d190>
>>> counter1.button()
<button at 0x106118f90>
```

5. Přidejte počítadlům vlastnost `value` s hodnotou počítadla a zařiďte, aby nastavení vlastnosti změnilo i text popisku. Například:

```
>>> counter1.set_value(1)
<counter at 0x10f1f8fd0>
>>> counter1.value()
1
>>> counter1.label().text()
'1'
```

6. Zařiďte, aby se klikem na tlačítko počítadla inkrementovala jeho hodnota. Otestujte řešení pro více počítadel v jednom okně.

7. Přidejte do počítadla tlačítko na dekrementaci jeho hodnoty.