Objektno programiranje

Objekti

Objekti so podobni zapisom, le da imajo atribute in metode. Metode se lahko sklicujejo na atribute, kakor tudi na druge metode (z uporabo `this` ali `self`).

Torej so objekti neke vrste **rekurzivni zapisi**. Rekurzivni, ker se lahko objekt skliče sam nase.

Objekti v OCamlu

V OCamlu in nekaterih drugih lahko objekte naredimo nesporedno

```
object (this)
   val x = ⋯
   val y = ⋯

   method f = ⋯
   method g = ⋯
end
```

Funkcionalnost objekta (njegove metode) opišemo s tipom zapisa. S podtipi lahko objekte vmeščamo v hierarhijo.

Več primerov bomo obdelali na vajah, glej tudi datoteko ./primeri.ml.

Objektno programiranje z razredi

Mnogi objektni jeziki pa poznajo mehanizem **razredov** (Java, C++, C#). Pogljemo še enkrat primer točke:

```
let p =
  object
    val mutable x = 10
    val mutable y = 20

    method get_x = x

    method get_y = y

    method move dx dy =
      x <- x + dx ;
      y <- y + dy
  end
```

Zgornja koda naredi le *eno* točko. Če želimo narediti več točk, lahko napišemo funkcijo, ki sprejme začetno pozicijo in vrne objekt. Lahko pa definiramo razred, na primer v Javi:

```
public class Point {
    private int x ;
    private int y ;

    public Point(int x0, int y0) {
        this.x = x0;
```

```
        this.y = y0;
    }

    public int get_x() { return this.x; }

    public int get_y() { return this.y; }

    public void move(int dx, int dy) {
        this.x += dx;
        this.y += dy;
    }
}
```

Sedaj tvorimo nove točke s *konstruktorjem*: `new Point(10, 20)`.

Tudi v OCamlu lahko definiramo razred:

```
class point x0 y0 =
  object
    val mutable x = x0
    val mutable y = y0

    method get_x = x

    method get_y = y

    method move dx dy =
      x <- x + dx ;
      y <- y + dy
  end
```

In tvorimo novo točko s konstruktorjem: `new point 10 20`.

Razredi niso samo bližnjica za konstruktorje, ampak omogočajo še mnoge druge mehanizme:

- enkapsulacija
- konstruktor
- dedovanje
- vmesniki
- prekrivanje metod
- preobteževanje metod
- generične metode in razredi
- abstrakne metode in razredi
- …

Vsi ti mehanizmi lahko delo z razedi naredijo precej zapleteno. Verjetno je eden od razlogov za kompleksnost ta, da programski jeziki kot so Java, C# in C++ *vse* mehanizme za organizacijo kode izražajo s pomočjo razredov. Tako java ne pozna modulov in funktorjev, algebrajskih podatkovnih tipv, ali parametričnega polimorfizma – vse to nadomešča z razredi.

Naredimo kratek pregled osnovnih mehanizmov objektnega programiranja z razredi in jih primerjajmo s koncepti, ki smo jih spoznali do sedaj.

Enakpsulacija

Enakpsulacija je skrivanje stanja objekta, se pravi, da naredimo atribute (lahko pa tudi metode) nedostopne zunaj razreda. V Javi to naredimo z določilom `private`.

V OCamlu so atributi vedno privatni. Metodo naredimo privatno z določilom `private`.

Pojem *enkapsulacija* včasih zajema tudi idejo, da objekt poleg stanja (atributov) s seboj nosi tudi metode za delo z njimi.

Skrivanje definicij lahko implementiramo tudi z lokalnimi definicijami in signaturami, ki zakrijejo implementacijo.

Konstruktor

Konstruktor je del kode, ki nastavi začetne vrednosti atributov objekta. Običajno konstruktor sprejme argumente, se pravi, da je to funkcija.

V Javi je ime konstruktorja enako imenu razreda, prav tako v C++ in OCamlu.

Dedovanje

En razred lahko *deduje* od drugega:

```
public class A extends B { ⋯ }
```

To pomeni, da ima `A` vse, kar ima `B` (torej atribute in metode).

OCaml dopušča večkratno dedovanje.

Vmesniki

Vmesniki predpisujejo funkcionalnost, ki jo mora imeti objekt. V Javi vmesnik definiramo z

```
interface I { ⋯ }
```

in od razreda zahtevamo, da zadosti vmesniku `I` (lahko tudi večim)

```
class C implements I { ⋯ }
```

Prekrivanje

Razred lahko nekatere od podedovanih metod *prekrije* z lastnimi definicijami.

Tu se pojavi vprašanje, kako dostopati do prekritih metod, saj jih včasih potrebujemo. V Javi to naredimo s `super.imeMetode(⋯)`. Podobno vprašnje se pojavi pri konstruktorjih: kako iz konstruktorja pokličemo konstruktor iz nadrazreda?

Preobteževanje

Če imamo več metod z istim imenom `f`, pravimo, da smo `f` *preobtežili* (angl. overload). Metode se morajo med seboj razlikovati po številu argumentov ali njihovih tipih, sicer iz klice metode ne moremo razbrati, katero različico želimo.

OCaml ne pozna preobteževanja. Lahko pa uporabimo module in jih odpremo lokalno z `let open M in ⋯` ali `M.(⋯)`.

Generične metode in razredi

Generične metode in razredi so parametrizirani z razredi ali vmesniki, takole:

```
public class A<B> {
    ...
}
```

Definicija razreda `A` je *parametrizirana* z razredom `B`.

V C++ v ta namen uporabljamo predloge (angl. templates).

V OCaml poznamo dva mehanizma za generično programiranje: parametrični polimorfizem in module.

Abstraktne metode in razredi

Abstraktne (v C++ se imenujejo *virtualne*) so metode, ki jih ne implementiramo, ampak samo deklariramo. Se pravi, povemo njihov tip ne pa tudi implementacije.

Če razred vsebuje abstraktno metodo, mora biti tudi sam deklariran kot abstrakten. Objektov abstraktnega razreda ne moremo ustvarjati z `new`, saj so neke vrste "nedokončani" razredi.

Abstraktni razred je neke vrste mešanica implementacije in speficikacije (specifikacija sestoji iz abstraktnih metod).

Razredi v OCamlu so lahko abstraktni.