Programiranje 2

Pregled Jave za Python programerje

Iz Pythona v Javo

Struktura programa v Javi

Program v Javi sestoji iz enega ali več *razredov* in *vmesnikov*, ki so shranjeni v datotekah s končnico `.java`. Vsak razred ali vmesnik mora biti v svoji datoteki `.java`, ki ima enako ime kot razred ali vmesnik. Dveh razredov ne moremo shraniti v eno datoteko (izjema so t.i. *notranji razredi*, angl. *inner classes*), zato ni nič nenavadnega, če je program razdrobljen na desetine ali celo stotine datotek. Razrede lahko dodatno razporedimo v (gnezdene) *pakete*, pri čemer je vsak paket shranjen v svoji mapi. Na razred `Klobasa`, ki je shranjen v podpaketu `gorenjska`, ki je del paketa `slovenija`, se sklicujemo s `slovenija.gorenjska.Klobasa`.

Preden program v Javi poženemo, ga moramo *prevesti* s prevajalnikom `javac`, ki izvorne datoteke `.java` prevede v datoteke `.class`. Programer običajno uporablja razvojno okolje (IDE -- Integrated Development Environment), ki to naredi samodejno. Za zagon programa ne potrebujemo izvornih datotek `.java`, ampak samo prevedene datoteke `.class`. Eno ali več datotek `.class` lahko shranimo v datoteko `.jar` (Java ARchive), ki je primerna za distribucijo in jo lahko damo končnemu uporabniku.

Java ima *striktne statične tipe*. Tipi so *striktni* kar pomeni, da mora imeti vsaka spremenljivka, argument, funkcija in na sploh vsak izraz točno določen tip, ki se ne more spremeniti. Tipi so *statični*, kar pomeni, da se preverjajo ob prevajanju programa (pred "dinamično" fazo, ki je izjvajanje programa). Če prevajalnik odkrije, da se tipi ne ujemajo, javi napako in programa ne prevede. To pomeni, da veliko trivialnih napak odkrijemo ob prevajanju namesto ob zagonu programa. Poleg tega IDE sproti označuje napake v programu, kar dodatno olajša programerjevo delo.

Glavna metoda `main`

Natanko en razred v programu mora vsebovati funkcijo `main`, ki je vstopna točka v program. Se pravi, ko zaženemo program, se začne izvajati `main`. Program lahko poženemo tudi v spletnem brskalniku kot *applet*, vendar o tem ne bomo veliko govorili.

Hello, world!

Tradicija zahteva, da je prvi program, ki ga napišete v novem programskem jeziku "Hello, world". V Javi izgleda takole:

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello, world!");
  }
}
```

Izvorna koda mora biti shranjena v datoteki `HelloWrold.java`.

Osnovna pravila Jave

Komentarji

Java ima tri vrste komentarjev:

- enovrstični komentar se začne z `//` in sega do konca vrstice
- večvrstični komentar se začne z `/*` in konča `*/`
- Javadoc komentar za dokumentiranje izvorne kode se začne z `/**` in konča z `*/`

Javadoc je sistem za dokumentacijo kode, ki zna samodejno generirati kodo v obliki HTML. Prav tako IDE sproti prikaže Javadoc dokumentacijo, s katero opremimo svojo kodo.

Zamikanje, podpičja in zaviti oklepaji

V Pythonu blok ukazov (na primer telo zanke `for`) določimo z zamikanjem. V Javi zamikanje ni obvezno (a je zelo, zelo priporočljivo), blok ukazov pa označen z zavitimi oklepaji `{` in `}`.

V Pythonu vsak ukaz praviloma stoji v svoji vrsti. V Javi je prehod v novo vrsto povsem nepomemben, ukaz pa se zaključi s podpičjem `;`.

Na primer, pythonsko kodo

```
if x < a[i]:
   d = i - 1
else:
   l = i + 1
```

zapišemo v Javi takole (oklepaji, ki oklepajo pogoj pri `if` so obvezni):

```
if (x < a[i]) {
  d = i - 1;
} else {
  l = i + 1;
}
```

Pojavi se vprašanje, kam dati zavite oklepaje. Zgornjo kodo bi nekateri zapisali takole:

```
if (x < a[i])
{
  d = i - 1;
}
else
{
  l = i + 1;
}
```

Verjetno je najbolje, da ne izumljate svojega stila, ampak upoštevate kakšna splošno sprejeta priporočila, na primer Google Java Style Guide. Ta v razdelku Braces priporoča prvega od zgornjih dveh stilov.

Kodo v Javi običajno zamikamo za 2 presledka (v Pythonu so običajni 4 presledki).

Spremenljivke

V Javi poznamo sedem vrst spremenljivk. Zaenkrat si oglejmo samo osnovno uporabo.

Vsaka spremenljivka mora biti ob prvi uporabi *deklarirana*, kar pomeni, da navedeno njen *tip* (in *določila*, glej spodaj). Na primer,

```
int a = 10;
```

je deklaracija spremenljivke `a`, ki je tipa `int` (predznačeno 32-bitno število) z začetno vrednostjo `10`. Spremenljivki ni treba določiti začetne vrednosti, lahko jo samo deklariramo:

```
int b;
```

V tem primeru je njena začetna vrednost odvisna od njenega tipa (glej Initial Values of Variables): numerične spremenljivke se inicializairajo z `0` ali `0.0`, boolove vrednosti s `false` in objekti z `null`. Pozor, spremenljivke brez začetne vrednosti so eden od virov napak. Če je le možno, kodo napišemo tako, da premenljivki ob deklaraciji že določimo začetno vrednost.

Spremenljivka je veljavna od svoje deklaracije do konca bloka, v kateri je deklarirana. Primer:

```
if (x < 10) {
  int i = x + 10;
  if (y < 10) {
    int j = x - 10;
    i = j + y;
  }
} else {
  int k = x + 2;
  z = k + 7;
}
```

Spremenljivka `i` je veljavna v vrsticah 2--6. Spremenljivka `j` je veljavna v vrsticah 4--5. Spremenljivka `k` je veljavna v vrsticah 8--9.

Konstante (določilo `final`)

Spremenljivki z določilom `final` ne moremo spreminjati vrednosti, zato je to pravzaprav *konstanta* in ne spremenljivka. Na primer,

```
final int VELIKOST = plosca.getVelikost() ;
```

je deklaracija konstante `VELIKOST`, katere vrednost se izračuna enkrat na začetku, nato pa ostane nespremenjena. Če bi jo poskusili kasneje spremeniti, bi prevajalnik javil napako.

Določilo `final` uporabimo vedno, kadar pričakujemo, da vrednosti spremenljivke ne bomo spreminjali. S tem prevajalniku povemo, kako nameravamo uporabljati spremenljivko, ta pa nas bo potem opozoril, da smo kršili svoj namen. Poleg taga lahko prevajalnik bolje optimizira kodo, če ve, da se vrednost spremenljivke ne bo spreminjala.

Dogovori o poimenovanju

Pri poimenovanju se držimo splošno sprejetega stila (glej razdelek Naming):

1. Imena razredov in vmesnikov pišemo z veliko začetnico, vsaka naslednja beseda se piše z veliko, na primer `MojRazred`. Ime razreda je v ednini.
2. Imena spremenljivk, atributov in metod pišemo z malo začetnico, vsaka nadaljna beseda se piže z veliko, na primer `mojaSpremenljivka`.
3. Imena konstant (z določilom `final`) se pišejo z velikimi črkami, na primer `MOJA_KONSTANTA`.

Ukazi `import`

Tako kot v Pythonu moramo tudi v Javi naznaniti uporabo razredov in vmesnikov iz drugih paketov z ukazom `import` na začetku datoteke `.java`. Pri tem nam je v veliko pomoč IDE, ki običajno sam poskrbi za ustrezne ukaze `import`.

Kontrolne strukture

Pogojni stavek

V Pythonu pogojni stavek pišemo

```
if pogoj:
  A
else:
  B
```

pri čemer lahko `else` in `B` izpustimo. V Javi pogojni stavek pišemo

```
if (pogoj) {
  A
} else {
  B
}
```

Oklepaj pri pogoju je obvezen. Tudi tu lahko `else` in `B` izpustimo.

Python pozna `elif`, ki ga Javi ni. Namesto tega uporabimo kar `else`, ki mu sledi `if`. Torej pišemo

```
if pogoj1:
   A
elif pogoj2:
   B
elif pogoj3:
   C
else:
   D
```

v Javi

```
if (pogoj1) {
  A;
} else if (pogoj2) {
  B;
} else if (pogoj3) {
  C;
} else {
  D;
}
```

Zanka `while`

Zanko `while` pišemo v Pythonu takoe:

```
while pogoj:
  A
```

V Javi jo pišemo

```
while (pogoj) {
  A
}
```

Ukaza `break` in `continue` delujeta tako kot v Pythonu, le pozabiti ne smemo, da jima sledi podpičje.

Zanka `for`

Zanka `for` v Javi ima dve obliki. Prva je enaka kot Pythonova zanka

```
for x in iterator:
  A
```

in jo pišemo

```
for (T i : iterator) {
  A
}
```

Kot vidimo, moramo navesti tip `T` spremenljivke `i`. Druga oblika izhaja iz programskega jezika C:

```
for (A; pogoj; B) {
  C
}
```

in je ekvivalentna programu

```
A;
while (pogoj) {
  C;
  B
}
```

Na primer, zanko, ki šteje s števcem `i` od `0` do vključno `10` napišemo takole:

```
for (int i = 0; i <= 10; i += 1) {
  ..
}
```

Zanka `do`

Iz jezika C je Java prevzela tudi zanko `do`:

```
do {
  A
} (pogoj);
```

ki je ekvialentna programu:

```
A;
while (pogoj) {
  A;
}
```

Tipi

Java ima dve vrsti podatkovnih tipov:

1. **Osnovni tipi** so številski tipi in boolove vrednosti:
    - `boolean` sta boolovi vrednosti `true` in `false`
    - `char` so znaki '\u0000' in '\uffff', to je cela števila med 0 in 65535
    - `byte` so cela števila med -128 in 127
    - `short` so cela števila med -32768 in 32767
    - `int` so cela števila med -2147483648 in 2147483647
    - `long` so cela števila med -9223372036854775808 in 9223372036854775807
    - `float` so realna števila v plavajoči vejici
    - `double` so realna šteivla v plavajoči vejici z dvojno natančnostjo
2. **Referenčni tipi** (angl. reference types):
    - razredi (angl. class)
    - vmesniki (angl. interface)
    - tabele (angl. array)

Številski tipi

Osnovne aritmetične operacije delujejo tako, kot v Pythonu.

Boolean

- Namesto `p and q` pišemo Javi `p && q`.
- Namesto `p or q` pišemo v Javi `p || q`.
- Namesto `x if p else q` pišemo v Javi `x ? p : q`.

Znaki in nizi

Java loči med znaki (angl. characters) in nizi (angl. strings). Niz je zaporedje znakov in ima tip `String`, posamezen znak pa ima tip `char`.

- znaki so zapisani z *enojnimi* narekovaji: `'a'`, `'b'`, `'\n'`, `'\''` itd.
- nizi so zapisani z *dvojnimi* narekovaji:
  `"This string contains forty-two characters."`

Torej je `'a'` znak in `"a"` niz.

Tabele

Tabela v Javi ima v naprej določeno velikost, ki je ne moremo spreminjati. Elementi tabele imajo vsi isti tip. Tabela elementov tipa `T` ima tip `T[]`. Tabelo tipa `T[]` z `n` elementi naredimo z ukazom `new T[n]`. Elementi tabele imajo naslednje začetne vrednosti:

- če je `T` številski tip, so nastavljeni na `0` ali `0.0`,
- če je `T` tip `boolean`, so nastavljeni na `false`,
- če je `T` razred, so nastavljeni na posebno vrednost `null`
- če je `T` tabela, se uporabijo ta pravila rekurzivno

Nekaj primerov, kako naredimo novo tabelo:

```
int[] a = new int[10];           // tabela celih števil dolžine 10, vredn
osti elementov so 0
int[] b = {1, 2, 4, 8};          // tabela štirih celih števil z dano vse
bino
double[][] c = new int[30][20]   // matrika s 30 vrsticami in 20 stolpci,
začetne vrednosti so 0.0
double[][] d = {{1.0, 0.0, 0.0}, // identična matrika velikosti 3 × 3
                {0.0, 1.0, 0.0},
                {0.0, 0.0, 1.0}}
```

Dolžina tabele `t` je `t.length`. Indeksi elementov tabele `t` tečejo od `0` do `t.length - 1`. Element z indeksom `i` dobimo s `t[i]`. Negativni indeksi ne delujejo.

Funkcije

Na tem mestu obravnavamo *statične* funkcije, ki ustrezajo Pythonovim običajnim funkcijam (brez `self`). Na sploh v Javi določilo `static` pomeni, da funkcija, spremenljivka ali konstanta *ni* del objekta, ampak obstaja samostojno. Če imamo v razredu `Krava` definirano *statično* funkcijo ali spremenljivko `rogovi`, do nje dostopamo s `Krava.rogovi` -- in v programu obstaja natanko ena različica spremenljivke `rogovi`. Nasprotno pa ne-statične spremenljivke (ki jim rečemo *atributi*) in funkcije (ki jim rečemo *objektne metode*) ne obstajajo same po sebi: vsak objekt, ki ga naredimo, ima svoje atribute.

Za ne-objektno programiranje torej velja, da funkcijo definiramo takole:

```
public static tipRezultata imeFunkcije(tip1 arg1, ..., tipN argN) { ... }
```

Določilo `public` pomeni, da je funkcija vidna iz vseh razredov (glej spodaj). Povedati moramo, kakšen tip vrača funkcija, kako ji je ime, in kakšni so njeni argumenti, skupaj s tipi argumentov.

Možno je definirati več različic iste funkcije, če različice sprejmejo različne argumente. Na primer:

```
public static int f(int x) { return x; }

public static int f(int x, int y) { return x + y; }
```

Temu pravimo *preobteževanje* (angl. *overloading*). Ko funkcijo kličemo, bo Java ugotovila, katero različico smo imeli v mislih, glede na to, kašne argumente smo podali. Na primer, `f(42)` je očitno prva različica in `f(42, 23)` druga.

Določila `public`, `private` in `protected`

V Pythonu je vsak kos programa in vsak objekt popolnoma odprt: vsi ostali deli programa se lahko vtikajo vanj. To je pri večjih programih nadležno, saj želimo vedeti, kako so kosi kode odvisni drug od drugega. V ta namen ima Java določila, ki omejujejo dostop do atributov in metod v razredu:

- `public` -- dostopno povsod
- `protected` -- dostopno v tem razredu, paketu in podrazredih
- (brez določila) -- dostopno samo v tem razredu in paketu
- `private` -- dostopno samo v tem razredu

Podrobnejšo razlago najdete v poglavju Controlling Access to Members of a Class.

Načeloma vedno uporabimo najbolj restriktivno smiselno določilo. Torej uporabimo `private` vedno, razen v primeru, ko imamo dober razlog, da ga ne uporabimo. Določilo `public` samo za metode in konstante, ki morajo biti zares dostopne od povsod.

Zadnja sprememba: torek, 23. maj 2017, 09:20

◀ Zaviti in oglati oklepaji v Eclipse (skrito)

Skok na...

Vaje: prvi koraki v Javi (skrito) ▶