1 Sintaksa jave

Naredbe, konvencije i komentari

Osvnovna jedinica svakog programskog jezika, pa i Jave je "naredba", nešto što treba da se u tačno odredjeno vreme izvrši. Java izvršava svaku naredbu sleva na desno i odozgo na dole (pazeći dobro na naredbe toka, koje to mogu da naruše!). Da bi Java znala da razazna šta je deo jedne, a šta deo druge naredbe, one se razdvajaju znakom tačka-zarez `;` , odnosno neke specijalne naredbe, puput naredbi kontrola toka, definišu takozvane blokove koda, i one se odvajaju unutar vitičastih (velikih) zagrada `{ }`.

Pored naredbi, svaki programski jezik ima svoju konvenciju imenovanja svojih promenjivih. Iako svaki programer ima svoj stil pisanja koda, ove konvencije su uglavnom (ne)pisana pravila kojih se treba čvrsto držati jedino iz preglednosti i lakog vizuelnog usaglašavanja sa tudjim kodovima istog programskog jezika. Neke od najpopularnijih konvencija imenovanja su:

- camelCase - Java, C# i još mnogo drugih programskih jezika koriste baš ovu konveciju
- snake_case - Python je glavni primerak programskog jezika koji koristi ovu konveciju
- cebab-case - Najmanje popularna konvencija; najčešće korišćena u HTML-u ili CSS fajlovima

Uz ove stvari jako je bitno napomenuti da svaki programski jezik dolazi sa **komentarima**. Prosto rečeno, to su naredbe koje se ne

izvršavaju ikada. One su tu samo da bi pomogle programeru, bilo onome ko je pisao taj program ili nekome drugom ko ga čita, da lakše razazna šta odredjeni delovi koda rade. U Javi, imamo 3 tipa komentara:

- Jednolinijski kometari koji se odvajaju sa `//`. Oni uglavnom služe za kratke i privremene komentare za zadatu naredbu.
- Višelinijski komentari koji se pišu izmedju simbola `/*` i `*/`. Oni služe za duže komentare koji uglavnom nagoveštavaju šta naredni delovi koda treba da odrade.
- Dokumentacioni komentari koji se pišu izmedju simbola `/**` i `*/`. Oni se mogu formatirati i pišu se da omoguće ostalim programerima koji koriste vaše funkcije u svojim programima da jasno i brzo razaznaju njihovu namenu i upotrebu.

Hajdemo da vidimo ovu teoriju primenjenu na praktičnom primeru:

```
package ime.paketa; //naredba koja deklariše ime paketa u
kome stoji naš program

import java.util.Scanner; //naredba koja importuje klasu iz
druge java datoteke (importuje se klasa Scanner iz
biblioteke utility)

public class PrviPrimer { //ime naše klase. Klase se
imenuju po CamleCase konvenciji sa prvim početnim slovom
    // Ovo je jedan jednolinijski komentar
    int x = 5; //Naredba koja deklariše i inicijalizuje
jedan ceo broj, čija je vrednost 5, a ime 'x'
    /*  Ovo je
        primer višelinisjkog
        komentara
    */
```

```
    /** A ovo
        dokumentacionog */
}
```

Tipovi primitivnih promenjivih, deklaracija i inicijalizacija

Promenjive čine srž svakog programskog jezika. Kao i u matematici, svaka promenjiva ima svoj tip, svoju vrednost i svoje ime (i ono što ne vidimo dok pišemo kod, svoje mesto u memorijskom prostoru), ali nasuprot promenjivoj u matematici, jedna ista promenjiva u različitim momentima može imati različite vrednosti. Da elaboriram, pogledajmo naredni sistem linearnih jednačina:

```
x + y = 5
x - y = 3
```

Promenjive 'x' i 'y' u svakom momentu imaju jedinstvenu (do potencijalno na odabir) vrednost. Čak iako ja, recimo saberem te dve jednačine i dobijem treću:

```
2x = 8
```

vrednosti 'x' i 'y' promenjivih ostaje isti. U sve tri jednačine vrednost promenjive 'x' je 4 (a 'y' je 1). U programskim jezicima, pa i Javi, to ne mora da bude slučaj. Pogledajmo naredni kod:

```
int x = 4; //Vrednost promenjive x je 4
x = 2; //A sada je njena vrednost postala 2
```

```
int y = x + 10; //vrednost promenjive y je sada 12
x = -3; //vrednost x promenjive je -3 a y je idalje 12
```

Primetimo kako u različitim linijama koda (tj u različitim vremenskim periodima) promenjiva 'x' ima različite vrednosti!
Pored toga, možemo i izvršiti naredni matematčki prekršaj:

```
int x = 2; //U ovoj liniji koda x ima vrednost 2
x = x + 5; //a u ovoj, nakon njenog završetka x će imati vrednost 7
```

Poslednja linija koda, iako matematiči neispravna, kaže sledeće:

1. U memorijski prostor označen simbolom 'x' upiši ono što ce biti rezultat desne strane jednakosti
2. Uzmi vrednost u memorijskom prostoru označenom simbolom 'x' (što je 5 po prvoj liniji koda) i saberi to sa 2.
3. Dat rezultat, 7, upiši u memorijski prostor 'x'.

Kako je Java Objektno Orijentisan programski jezik, razlikujemo 2 tipa promenjivih:

- Primitivni (Prosti)
- Složeni (Objektni, Kompozitni, Referentni)

Prosti tipovi podataka se dalje dele na:

1. Celobrojne:
    - byte | 1B = 8bit
    - char | 2B = 16bit
    - short | 2B = 16bit
    - int | 4B = 32bit

    - long | 8B = 64bit
2. Realne:
    - float | 4B = 32bit
    - double | 8B = 64bit
3. Logičke:
    - boolean | 1bit

Primetimo da, iako se `char` koristi da čuva karaktere, oni se tretiraju kao celi brojevi i nad njima se može vršiti ista aritmetika kao i sa celim brojevima!

Da bi smo Javi naznačili da želimo da režervišemo sistem u memoriji za odjedjen tip primitivne promenjive, koristimo narednu sintaksu:

```
<tip_promenjive> <jedinstveni_identifikator_promenjive>;
```

Primeri:

```
//Deklarizacija promenjivih
int x; //deklarisali smo jedan ceo broj, dakle rezervisali
smo 4 bajta u memoriji i taj prostor od 4 bajta smo
       //nazvali 'x'
double d;
boolean nekiBoolean; //služeći se obične camelCase
konvencije, kreirali smo jedan boolean sa imenom
nekiBoolean
char karakter;
```

Ovako napisane promenjive uzimaju svoje podrazumevane vrednosti, da bi smo im dodelili neke naše vrednosti, koristimo operator dodele `=`, tako što (ovaj put bez naznačavanja tipa

promenjive; Java je vrlo pametan programski jezik!) napišemo ime promenjive, pa operator dodele i onda odgovarajući literal:

```
<identifikator_promenjive> = <vrednost_promenjive>;
```

Primeri:

```
//Inicijalizacija promenjivih
x = -5;
d = 3.14;
nekiBoolean = true; //logički tipovi mogu biti samo true ili false
karakter = 'c'; //karakteri  se navode izmedju jednostrukih navodnika ''
```

Primetimo da nešto poput:

```
int x = 12;
int x = -13;
```

nije dozvoljeno, jer se proces deklarizacije promenjive vrši samo jedanput!

Pisanje koda prvenstveno služi za olakšanje i asistenciju čovečanstvu, te se svaki programski jezik trudi da bude što dostupniji i jednostavniji u svojoj primeni, pa tako, Java dopušta da se proces deklarizacije i inicijalizacije objedini u jednu komandu:

```
//Inicijalizacija sa deklarizacijom
int ceoBroj = 5;
double realanBroj = 2.71;
```

```
boolean tačno = false;
char nebuloza = 'Y';
```

Objektni tipovi se razlikuju od primitivnih po tome kako se čuvaju u memorijskom prostoru, pa zato i po svojoj inicijalizaciji/deklaraciji. Oni se uvek pišu sa početnim velikim slovom. Dalja diskusija o objektnim tipovima će uslediti u dogledno vreme, za sada je dovoljno da se upoznamo sa jednim 'hibridnim' tipom podataka, niskom, odnosno nizom karaktera - `String`. String je imutabilni (nepromenjivi) tip podataka koji se ponaša i kao objektni tip i kao primitivni tip. Za sve naše potrebe možemo ga postrati kao primitivni tip.

```
String s;
s = "Zdravo svete!";
String nekiDrugiString = "Pozdrav svima! Kako ste?";
```

String se piše izmedju običnih (duplih) navodnika `" "` i interpretira se kao niz karaktera. Dakle svaki karakter koji se moze upisati u `char` može se upisati i u String.
Primetimo da ikao sam naglasio da su niske imutabilne, ta imutabilno se odnosi samo na onaj deo koji se odnosi na njihovo ponašanje u memoriji, pa je nešto poput narednog dozvoljeno:

```
String s = "Pozdrav!"; //U ovom momentu promenjiva s ima vrednost "Pozdrav!"
s = "Zbogom."; //a u ovom momentu promenjiva s ima vrednost "Zbogom."
```

Operatori

Operatori čine osnovni deo sintakse Jave koji nam omogućava da baratamo sa vrednostima naših promenjivih. Najlakše je videti njihovu upotrebu kroz neke jednostavne primere.

1. Operator dodele `=` | infiksno
   Operator dodele dodeljuje levoj strani "jednakosti" vrednost koja se nalazi sa njene desne strane. Naravno tipovi tih vrednosti moraju da se podudaraju; ako leva strana jednakosti očekuje ceo broj, onda i krajnje rešenje desne strane mora takodje biti ceo broj!

```
int x = 5; //Dodelili smo promenjivoj x broj 5
```

Nešto poput narednog nije dozvoljeno:

```
int x = 5.2;
```

jer se tipovi ne slažu: sa leve strane se očekuje ceo broj, a sa desne se nalazi realan!

1. Aritmetički operatori:
   - Sabiranje/oduzimanje `+/-` | infiksno

```
int x = 5 + 2; //vrednost x-a nakon ove naredbe je 7
int y = x + 3; //vrednost y-a nakon ove naredbe je 10
x = x + y //vrednost x-a nakon ove naredbe je 17
int z = 2 - 10 //vrednost z-a nakon ove naredbe je -8
```

   - Obrtaje znaka `-` | prefiksno

```
double d1 = 3.14;
double d2 = -d1;
```

- Množenje/deljenje `* /` | infiksno

```
double m = 5.0, n = 3.3; //više promenjivih istog tipa se mogu deklarisati u jednoj komandi
double nekiSlozeniIzraz = ((m*30.2)/(n*0.11) + 15*m);
double d = 3; //Ovo je dozvoljeno, ali vrednost d promenjive će biti 3.0!
int x = 12/3; //Celobrojno deljenje, rezultat je ceo broj 4
double y = 12.0 / 3; //ili 12/3.0 <- Realno deljenje, rezultat je 4.0
```

- Moduo (ostatak pri deljenju) `%` | infiksno

```
int x = 37;
int y = 7;
int z = x % y; //z = 2, jer je 37 / 7 = 5, a 5 * 7 = 35, pa je ostatak 2
```

3. Operatori uvećanja i umanjenja (inkrementacija i dekrementacija) | prefiksno i postfiksno

```
/*
Uvecanje za 1: ++
*/
int x = 5;
x++; //Uvećava x za jedan nakon izvršavanja čitave naredbe! x će imati vrednost 6
int y = 10;
```

```
++y; //Uvećava y za jedan pre izvršavanja ostatka naredbe! y će imati vrednost 11
int z1 = 5;
int z2 = 5;
int z = (++z1) + (z2++);
//U ovom momentu z će imati vrednost 11! z1 i z2 će imati vrednost 6
//Ovo se desilo jer ++z1 je uvećao z1 na 6 odmah, zatim ga je sabrao sa z2, što je 5, pa zatim
//tu vrednost (11) sačuvao u promenjivu z, a tek nakon toga je uvećao z2 na 6!!

int w = 10;
w--; //Analogno, w će imati vrednost 9
```

4. Relacioni operatori
   Rešenje svakog relacionog operatora je uvek tipa boolean!
   Razlikujemo više relacionih operatora

- Poredjenje jednakosti `==` | infiksno

```
boolean b = 5 == 3; //b je false
b = 5 == 5; //b je true
b = 5 == 6; //b je false
boolean b1 = true == false //ako i samo ako -> vrednost b1 je false!
```

- Manje (manje ili jednako) `<=` | infiksno

```java
boolean b = 5 <= 3; //b je false
b = 5 <= 5; //b je true
b = 5 <= 6; //b je false
```

- Strogo manje ```<``` | infiksno

  ```java
boolean b = 5 < 3; //b je false
b = 5 < 5; //b je false
b = 5 < 6; //b je true
```

- Veće (veće ili jednako) `>=` | infiksno

```java
boolean b = 5 >= 3; //b je true
b = 5 >= 5; //b je true
b = 5 >= 6; //b je false
```

- Strogo veće `>` | infiksno

```java
int x = 5;
boolean b = x > 3; //b je true
b = x > 5; //b je false
b = x > 6; //b je false
```

5. Logički operatori
   I ovde je rešenje uvek logički literal!

- (Logička) negacija (ne) `!` | infiksno

```
/*
!|  p    |  !p
---------------
   true  |  false
   false |  true
*/
boolean b = !true; //resnje je false
b = !(2 > 3); //resenje je true
```

- (Logička) konjukcija (i) `&&` | infiksno

```
/*
&&      | true   false
----------------------
true    | true  false
false   | false false
*/
boolean b = true && false; //resenje je false
b = (5 == 3) && (5 < 2); //resenje je false
b = (true == true) && true; //resenje je true
```

- (Logička) disjunkcija (ili) `||` | infiksno

```
/*
||      | true   false
--------------------
true    | true   true
false   | true   false
*/
boolean b = true || false; //resenje je true
b = (5 == 3) || (5 < 2); //resenje je false
```

6. Bitovni operatori
   Resenja su logički literali ili zadati literali sa svojim bitovima reorganizovanim

- Bitovna negacija `~` | prefiksno
  Negira (obrće) svaki bit

```
   ~  | true  false
--------------
      | false true

  ~(01011) = 10100
```

- Bitovna konjukcija `&` | infiksno
  Konjuktuje bit po bit. 1 interpretira kao tačno, 0 kao netačno.

```
   &  | true  false
---------------
true  | true  false
false | false  false

  01011
| 10010
--------
  00010
```

- Bitovna disjunkcija `|` | infiksno
  Disjunktuje bit po bit. 1 interpretira kao tačno, 0 kao netačno

```
   |  | true  false
---------------
```

```
true  | true  true
false | true  false

  01011
| 10010
--------
  11011
```

- Bitovna ekskluzivna disjunkcija (xor) `^` | infiksno
  Ekskluzivno disjunktuje bit po bit. 1 interpretira kao tačno, 0 kao netačno

```
   ^  | true  false
---------------
true  | false true
false | true  false

  01011
^ 10010
--------
  11001
```

- Aritmetičko pomeranje (bitova) u levo `<<` | infiksno
  Bitovi se pomeraju onoliko mesta u levo koliko je to zadato drugim operandom aritmetičkog pomeranja u levo, tako da se umeće onoliko 0 kolika je vrednost drugog operanda na najniža mesta.

```
11101001 << 1 = 11010010
11101001 << 3 = 01001000
```

- Aritmetičko pomeranje (bitova) u desno `>>` | infiksno
  Bitovi se pomeraju onoliko mesta u desno koliko je to zadato drugim operandom aritemtičkog pomeranja u desno, tako da se zadržava znak broja .

```
11101001 >> 1 = 11110100
01101001 >> 1 = 00110100
10000001 >> 3 = 11110000
```

- Logičko pomeranje (bitova) u desno `>>>` | infiksno
  Bitovi se pomeraju onoliko mesta u desno koliko jeto zadato drugim operandom logičkog pomeranja u desno, tako da se umeću nule.

```
11101001 >>> 1 = 01110100
11101001 >>> 2 = 00111010
```

7. Uslovni i instancioni operatori

- Uslovni operator dodele `?:` | ternarno
  Dodeljuje jednu od dve zadate vrednosti u zavisnosti od logičkog uslova. Šablon za korišćenje ovog operatora je:

```
<tip_promenjive_X> <ime_promenjive> = <logički_uslov> ?
<literal_tipa_X_1> : <literal_tipa_X_2>;
gde se dodeljuje:
<literal_tipa_X_1>; ukoliko je uslov ispunjen
<literal_tipa_X_2>; ukoliko uslov nije ispunjen
```

```java
boolean b1 = true, b2 = false;
int x1 = (b1 && b2) ? 10 : -5;
int x2 = (b1 || b2) ? 10 : -5;
//vrednost x1 = -5, posto je b1 && b2 = false
//vrednost x2 = 10, posto je b1 || b2 = true
```

- Instancioni operator `instanceof` | infiksno
  Proverava da li je zadati objekat instanca neke klase i vraća logički literal kao rešenje. Objekat je instanca druge klase ako im se izvedeni i nominalni tip poklapaju ili ako je izvedeni tip n-ta potklasa nominalne klase.

  ```java
  boolean jesteInstancaOd = nekiObjekat instanceof NekaKlasa;
  ```

````
Pored svega ovoga, moguće je i skratiti, tj objediniti ostale operatore sa operatorom dodele u jednu komandu. Takve komande su infiksne i izvrše zadatu operaciju sa prvim i dugrim operandom i rešenje upišu u prvi operand.
Primeri:
```java
int x = 5;
x += 7; //Skraceno za x = x + 7.Vrednost x = 12
int y = 14;
int z = 2;
y /= z; //Skraceno za: y = y / z. Vrednos t = 7
````

Dokle god se vrednosti poklapaju sa tipom prvog operanda u opticaj dolaze:
````

`+=, -=, /=, %=, ~=, &=, |=, ^=, <<=, >>=, >>>=`

Ključna reč final

Ukoliko želimo neku promenjivu da inicijalizujemo samo jedanput i da zagarantujem da njena vrednost ostane nepromenjena tokom izvršavanja programa, drugim rečima ako želimo da kreiramo neku konstantu, koristimo ključnu reč `final`. Ključna reč `final` je **modifikator** i kao takva navodi se pre navodjenja tipa promenjive.

Definicija 1.1 (Modifikatori)**.**

Modifikatori su ključne reči koji dodatno opisuju ponašanje klasa, funkcija ili promenjivih. Modifikatori se uvek navode pre početka deklaracije klase, funkcije ili promenjive.

Po konvenciji, konstante u javi se pišu sa svim velikim slovima. Ukoliko želimo sa više reči da nazovemo našu konstantu, odvajamo ih sa dodnjom crtom `_` i svaku reč pišemo sa svim velikim slovima.
Primer:

```
final double PI = 3.14;
final String PORUKA_DOBRODOSLICE = "Dobrodosli narode!";
final double E = 2.71;
final int MAX_POENA = 100;
final boolean ISTINA = true;
final double G = 9.81;
```

Primetimo da konstante ne moraju biti samo brojevi kaošto smo na to navikli u matematici! Takodje, inicijalizaciju konstanti možemo da odložimo, ali nakon sto je jednom inicijalizujemo, nemožemo je

inicijalizovati ponovo nikad više (ne možemo joj promeniti vrednost)!

```
final double KONSTANTA;
final double PI = 3.14;
final double TAU = 2 * PI;
KONSTANTA = 5.0; //Sasvim dozvoljeno
TAU = -100; //Greska, TAU je vec inicijalizovano!
KONSTANTA = 123; //Greska, KONSTANTA je vec incijalizovana!
```

Celobrojno i realno deljenje

Već smo se kratko dotakli celobrojnog i realnog deljenja u delu kada smo pričali o operatoru deljenja. U praksi smo navikli da kada delimo dva broja tako da delilac deli deljenik bez ostatka kao rezultat zapišemo ceo broj, a ukoliko delilac ne deli deljenik kao rezultat zapišemo realan broj. Kako je Java strogotipiziran programski jezik, on ovakvu distinkciju mora da izvrši pre dobijanja rezultata.
Celobrojno deljenje se u Javi vrši kada su oba operanda celi brojevi i kao rešenje se uvek dobija ceo deo količnika bez ostatka, drugim rečima, zanemarujemo sve posle zareza. Takodje, ako želimo da količnik dva realna broja sačuvamo u promenjivu tipa `int`, dobijamo isti rezultat - ostatak se zanemaruje.
Realno deljenje se u Javi vrši kada je barem jedan od operanada realan broj. Tada kao rešenje dobijamo ono što i očekujemo, decimalan broj sa zapetom, odnosno tačkom u slučaju Jave. Napomenimo da ako delilac deli deljenik u ovom slučaju, dobićemo ceo broj sa pridodatim `.0` pošto se očekuje realan, a ne ceo broj.

Ostaje pitanje šta da radimo kada imamo dva cela broja a kao rezultat želimo da dobijemo realan broj? Da bi smo ovako nešto

postigli u Javi, moramo da izvršimo *konverziju* iz celih u realan broj i to možemo da uradimo na dva načina:

- Eksplicitno
- Implicitno

Eksplicitna konverzija u neki tip podataka se vrši tako što se ispred identifikatora ili literala u običnim zagradama navodi tip podatka u koji želimo da konvertujemo zadati litera. Naravno, na nama je da zagarantujemo da takva konverzija uopšte ima smisla: konverzija celog u realan broj je trivijalna, ali konverzija izmedju baba i žaba ne bi bila moguća. Primer eksplicitne konverzije bi izgledao:

```
double d = (double)22/3;
```

U ovom primeru `(double)` se odnosi na celobrojni literal 22 i konvertuje ga u realan broj tako što mu dodaje `.0` na kraj, da dobijemo broj 22.0. Posle toga, izvršava ce realno deljenje pošto je prvi operand u tom slučaju sada realan broj i kao rešenje dobijamo broj 7.33333... Primetimo da je ovakva konverzija *trenutna*, ta ako imamo nešto poput:

```
int x = 22;
int y = 3;
double d = (double)x/y + x;
```

konverzija `literala` koji se nalazi u promenjivoj `x` se dešava samo na tom mestu; u momentu kada na snagu dolazi operator sabiranja, literal upisan u promenjivoj `x` će idalje biti celobrojnog tipa!

Implicitna konverzija se dešava implicitnim putem. Kao i kod deljenja, proizvod dva broja će biti realni literal kada god je barem jedan od operanada realan broj. Koristeći se time, gornji primer možemo rešiti i implicitnom konverzijom na sledeći način:

```
int x = 22;
int y = 3;
double d  = 1.0*x/y;
```

Kako množenje ima prednost u odnosu na deljenje i pošto je prvi operand množenja realan broj, kao rešenje dobijamo realan literal koji se poklapa po vrednošću sa literalom upisanim u promenjivu `x`, te kada se izvršava deljenje, prvi operand će biti literal tipa `double` i izvršiće se realno deljenje.