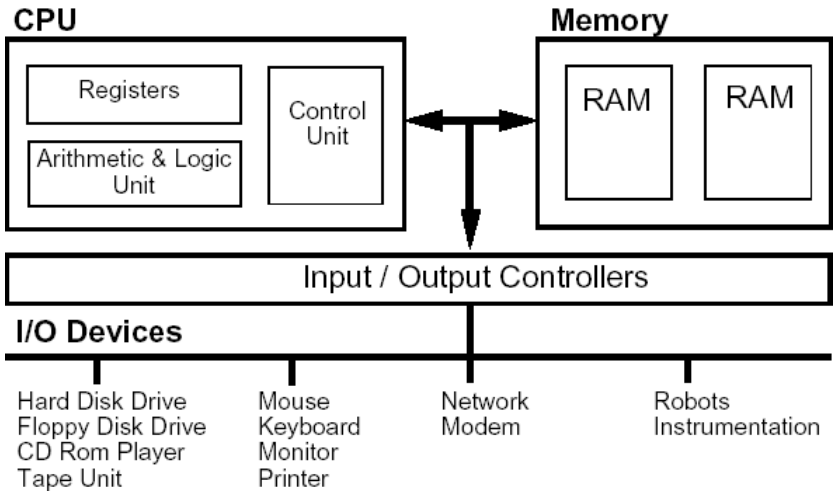
Uvod u programski jezik Java

Autori:
Goran Savić
Milan Segedinac

1. Pojam programa

Računarski program predstavlja uređen skup operacija koje računar izvršava.

U izvršavanju programa učestvuju različite komponente računara, svaka sa svojom specifičnom ulogom. Na slici su prikazane osnovne komponente savremene računarske arhitekture.



*preuzeto sa www.doc.ic.ac.uk

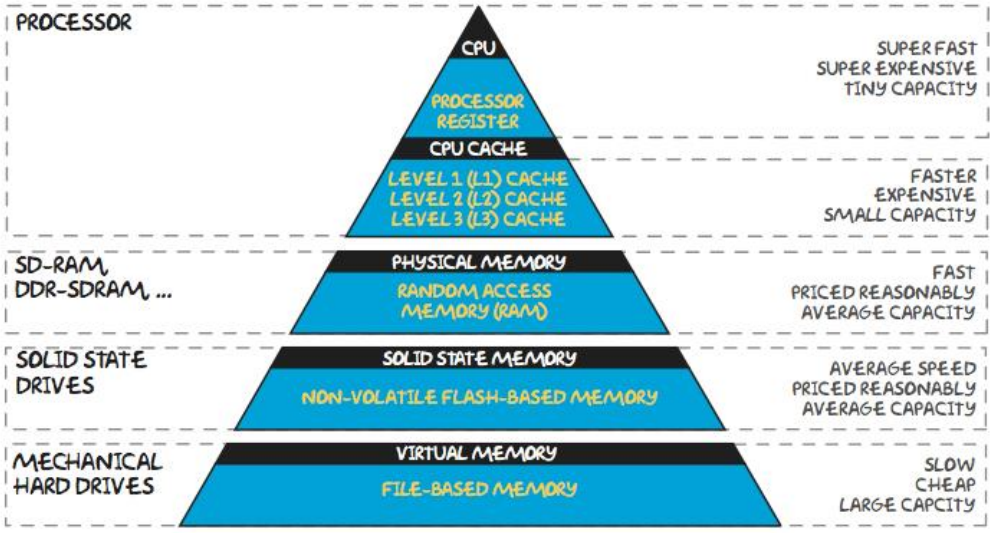
Za izvršavanje operacija definisanih u programu zadužen je procesor (CPU). Operacije se zadaju u formi procesorskih instrukcija. Procesor sadrži tri osnovne komponente. Kontrolna jedinica je zadužena za upravljanje operacijama. Ova jedinica prima, obrađuje i šalje podatke od, do i unutar procesora i na taj način upravlja ostalim računarskim komponentama. Aritmetičko-logička jedinica procesora izvršava operacije definisane programom. Procesorski registri predstavljaju memorijske lokacije u kojima procesor privremeno skladišti podatke u toku izvršavanja instrukcije programa. Pristup ovim lokacijama je veoma brz, ali je kapacitet ove memorije veoma ograničen. Na primer, procesori arhitekture x86-64 imaju 16 standardnih registara.

Glavno skladište podataka u toku izvršavanja računarskog programa je glavna memorija, predstavljena na slici u gornjem desnom uglu. Procesor ima mogućnost da direktno pristupi bilo kojoj lokaciji u radnoj memoriji. Ovakav tip memorije se naziva *random access memory* (RAM). Iz tog razloga se glavna memorija često naziva RAM memorija. Kod nas je prisutan i termin radna memorija. Pri izvršavanju programa, procesor preuzima podatke iz radne memorije i prebacuje u svoje registre. Takođe, rezultati operacija privremeno uskladišteni u registrima se upisuju u radnu memoriju. Radna memorija nije dugotrajna memorija. Podaci uskladišteni u radnoj memoriji se gube pri isključivanju računara. Ovde treba pomenuti i da osim registara, procesor sadrži i keš memoriju, kao specijalan tip memorije koja je sporija od registara, ali brža od radne memorije. Procesor u keš memoriji skladišti podatke kojima je nedavno pristupano u radnoj memoriji. Pri učitavanju podataka prvo se proverava da li je podatak dostupan u keš memoriji, kako se ne bi program usporavao pristupanjem radnoj memoriji ako nije neophodno.

Za dugotrajno skladištenje podataka koriste se različiti tipovi dugotrajne memorije. Ova memorija se nalazi na eksternim uređajima. Eksterni uređaji za skladištenje podataka su samo jedan tip eksternih uređaja, koji su prikazani u dnu slike. Ovi uređaji se nazivaju ulazno-izlazni uređaji i možemo ih podeliti u više grupa. Prva grupa su pomenuti uređaji sa dugotrajnom memorijom za skladištenje podataka. Hard disk je najvažniji uređaj ovog tipa. Ima najveći kapacitet od memorijskih skladišta

koje standardan računar sadrži, ali je sa druge strane ovo najsporija vrsta memorije. Treba pomenuti i SSD diskove kao uređaje za dugotrajno skladištenje koji su brži od hard diskova, ali su i skuplji po jedinici skladištenja. Memorija za dugotrajno skladištenje se često naziva i sekundarna memorija, za razliku od glavne memorije koja se naziva primarna memorija.

Generalno, ako govorimo o različitim tipovima memorije, ove tipove možemo hijerarhijski organizovati. Svaki sledeći nivo u hijerarhiji je sporiji od prethodnog, ali je i i jeftiniji po jedinici skladištenja, pa je samim tim u praksi raspoloživa veća količina tog tipa memorije. Na slici su prikazani različiti tipovi memorije hijerarhijski organizovani.

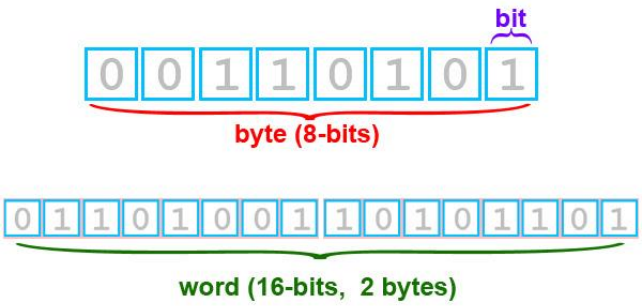


* preuzeto sa http://computerscience.chemeketa.edu

Pored memorijskih uređaja, postoje i drugi tipovi ulazno-izlaznih uređaja. Jedan tip su uređaji koji omogućuju interakciju sa korisnikom putem dobijanja informacija od korisnika (miš, tastatura, ...) ili slanja informacija korisniku (monitor, štampač, ...). Poseban tip uređaja su uređaji zaduženi za interakciju sa drugim računarskim uređajima. U uređaje ovog tipa spadaju mrežne kartice, modemi i sl.

Predstavljanje podataka u računaru

Podaci se u računaru predstavljaju u formi binarnih reči. Binarna reč je predstavljena kao niz bitova, pri čemu svaki bit može da ima vrednost nula ili jedan. 8 bitova čini jedan bajt. Ovo je ilustrovano na narednoj slici.



*preuzeto sa teach-ict.com

Bajtovi se dalje mogu grupisati u kilobajte, megabajte, gigabajte, ... prema sledećim pravilima.

1 byte	8 bits
1 kilobyte (KB)	1024 byte
1 megabyte (MB)	1024 kilobyte
1 gigabyte (GB)	1024 megabyte
1 terabyte (TB)	1024 gigabyte

Binarni brojni sistem je samo jedan od brojnih sistema koji omogućuju zapis brojeva. Ljudi koriste decimalni brojni sistem. Svaki brojni sistem ima određeni broj kao bazu sistema. Baza sistema predstavlja broj cifara koje postoje u tom brojnom sistemu. Kod binarnog brojnog sistema, baza je 2, a kod decimalnog je baza 10. Na osnovu baze i cifara zapisa se izračunava vrednost broja predstavljenog određenim brojnim sistemom. Ako vrednost broja označimo sa V, njegove cifre sa C_0, C_1,..., C_n, a bazu sa B, tada se vrednost broja dobija po sledećoj formuli:

$$V = \sum_{i=0}^{n} C_i * B^i$$

Na primer, u decimalnom brojnom sistemu, vrednost broja zapisanog ciframa 26 se dobija kao:

$$V = 6 * 10^0 + 2 * 10^1 = 6 + 20 = 26$$

Isti broj možemo u binarnom brojnom sistemu zapisati ciframa 11010. Vrednost broja (prikazana u decimalnom brojnom sistemu) se onda izračunava kao:

$$V = 0 * 2^0 + 1 * 2^1 + 0 * 2^2 + 1 * 2^3 + 1 * 2^4 = 0 + 2 + 0 + 8 + 16 = 26$$

Binarni zapis se često predstavlja i u heksadecimalnom brojnom sistemu. Ovo je brojni sistem sa bazom 16. Heksadecimalni brojni sistem ima 16 cifara: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. U tabeli su prikazane heksadecimalne cifre i zapisi njima odgovarajućih vrednosti u binarnom, odnosno decimalnom brojnom sistemu. Obzirom da jedna heksadecimalna cifra zamenjuje 4 binarne cifre, često se zbog kraćeg zapisa binarni zapis prikazuje u heksadecimalnom obliku.

Heksadecimalni broj	Binarne cifre	Decimalni broj
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

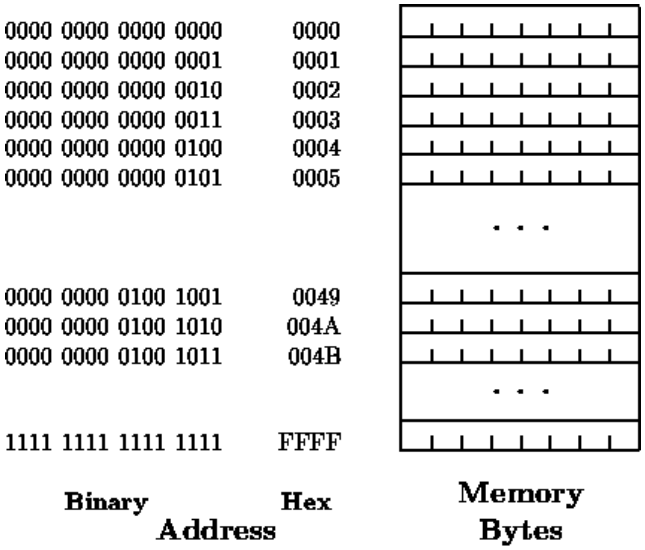
I kod heksadecimalnog brojnog sistema, način određivanja vrednosti zapisanog broja je isti kao i kod drugih brojnih sistema. Tako recimo za heksadecimalni broj A9 vrednost predstavljena decimalnim brojem bi bila 169, jer je

$$V = 9 * 16^0 + 10 * 16^1 = 9 + 160 = 169$$

Treba primetiti da su vrednost broja i njegova reprezentacija putem cifri određenog brojnog sistema, dve odvojene stvari. Svaka vrednost se može predstaviti bilo kojim brojnim sistemom koristeći cifre tog brojnog sistema prema gore prikazanom pravilu.

Skladištenje podataka u memoriji

Radna memorija predstavlja niz bajtova. Svaki bajt ima svoju adresu, preko koje se može adresirati, tj. procesor može pristupiti toj memorijskoj lokaciji i izvršiti čitanje ili upis podataka.



Kao što vidimo, svaka lokacija sadrži 8 bitova (1 bajt), a sama adresa je predstavljena i u binarnom i u heksadecimalnom obliku.

Predstavljanje instrukcija programa

Memorija računara skladišti podatke koje programi koriste, ali i same programe. Programi su predstavljeni nizom procesorskih instrukcija, pri čemu je svaka instrukcija predstavljena kao binarna reč. U ovoj binarnoj reči je predstavljena operacija koja treba da bude izvršena, kao i informacija nad kojim podacima se operacija vrši. Ovako predstavljene instrukcije se skladište u memoriji računara. Arhitektura skupa instrukcija (ISA - *instruction set architecture*) predstavlja skup svih instrukcija koje određeni procesor prepoznaje. Na personalnim računarima su danas najpopularniji procesori arhitekture x86 (Intel i AMD proizvode ovakve procesore). Postoje dve popularne verzije ove arhitekture, x86-32 i x86-64 u zavisnosti od dužine binarne reči sa kojom rade (32 bita ili 64 bita).

U nastavku je dat primer jedne instrukcije opisane 32-bitnom binarnom reči iz MIPS arhitekture skupa instrukcija. Reč je o instrukciji `addi` koja omogućuje dodavanje određene vrednosti na postojeću vrednost nekog podatka. Instrukcija kao parametre dobija oznaku registra u kojem je postojeća vrednost, vrednost koju treba dodati, i oznaku registra u koji treba smestiti rezultat operacije.

Dat je primer ove instrukcije u formi simboličkog jezika. Instrukcija dodaje vrednost 5 na sadržaj registra r8 i smešta rezultat u registar r9.

```
addi $9, $8, 5
```

vrednost koja se dodaje

oznaka registra koji skladišti vrednost koja se uvećava

oznaka odredišnog registra

oznaka operacije

U nastavku je prikazana ista instrukcija u obliku binarnog zapisa. Označeni su delovi zapisa prema značenju. Na osnovu ovako utvrđene strukture instrukcije, procesor dekodira instrukciju i izvršava je.

```
001000 01000 01001 0000000000000101
```

vrednost koja se dodaje (5)

oznaka odredišnog registra (9)

oznaka registra koji skladišti vrednost koja se uvećava (8)

oznaka operacije

2. Programski jezik

U skladu sa objašnjenim u prethodnom poglavlju, program se sastoji od sekvence instrukcija koje su predstavljene u formi binarnih reči i uskladištene u memoriji. Program se skladišti u sekundarnoj memoriji (hard disk najčešće), a pri pokretanju se učitava u radnu memoriju odakle procesor redom učitava instrukcije i izvršava ih. Pri izvršavanju instrukcija, pristupa se podacima programa koji se takođe nalaze u radnoj memoriji.

Ovako predstavljen zapis programa naziva se mašinski jezik, jer je ovakav zapis pogodan za interpretaciju od strane mašine, tj. računara. Mašinski jezik je neophodan način zapisa programa da bi se program mogao izvršiti na procesoru. Ipak, dok programer razvija program, ovakav zapis programa nije pogodan jer je teško čitljiv za čoveka. Iz tog razloga koriste se različite simboličke notacije za predstavljanje programa. Bez obzira na koji način zapisujemo program, svaki program je izražen u nekom programskom jeziku. Programski jezik predstavlja skup izraza i gramatičkih pravila kojima se od računara zahteva da izvrši određeni zadatak.

Programski jezik najnižeg nivoa je mašinski jezik. U njemu je direktno zapisano šta procesor treba da uradi i to u binarnom formatu čitljivom za procesor. Iste naredbe se mogu navesti i u simboličkom obliku čitljivijem za čoveka. Programski jezik koji ovo omogućuje je asembler. Primer asemblerskog koda dat je u nastavku.

```
MOV AL, 61h
```

Prikazani kod učitava u registar AL heksadecimalnu vrednost 61. Iako je ovakav zapis čitljiviji za čoveka od binarnog zapisa, i dalje programer mora da poznaje skup instrukcija koje procesor prepoznaje i da napiše program kao niz ovakvih instrukcija. Ovakvi programski jezici se nazivaju programski jezici nižeg nivoa.

Programski jezici višeg nivoa su dizajnirani tako da oslobode programera obaveze da razmišlja o tome kako se program izvršava na najnižem nivou u formi procesorskih instrukcija. Ovakvi programski jezici omogućuju razvoj programa na višem nivou apstrakcije. Instrukcije i memorijske lokacije su apstrahovane kroz koncepte višeg nivoa kao što su naredbe i promenljive programa. U nastavku je dat primer programskog koda u jeziku Java koji predstavlja jezik višeg nivoa.

```
int pocetniIznos = 5;
int uvecanIznos = pocetniIznos + 9;
```

Kao što vidimo, kod se ne bavi memorijskim lokacijama u kojima će vrednosti biti smeštene niti konkretnim instrukcijama procesora koje će biti izvršene. Jezik omogućuje definisanje podataka i operacija u formi koja je čitljivija za programera.

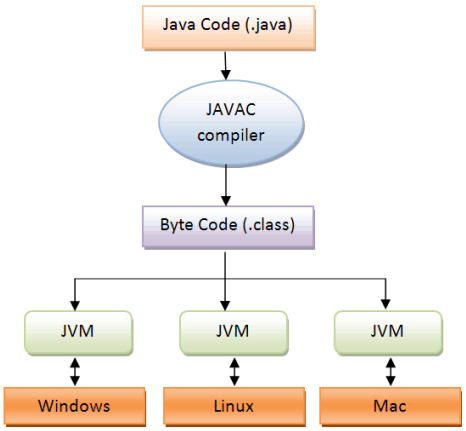
Bez obzira koji programski jezik koristimo, da bi se program izvršio neophodno je prevesti ovaj simbolički jezik u mašinski jezik koji procesor može da obradi. Za ovu svrhu se koriste specijalizovani programi koji se nazivaju prevodioci (kompajleri). Ulazni podatak u kompajler je programski kod u određenom programskom jeziku, a izlaz je prevedeni program u mašinskom jeziku.

3. Programski jezik Java

Java je programski jezik višeg nivoa opšte namene. Nastala je po uzoru na C i C++ jezike i značajan deo sintakse se poklapa sa ovim jezicima. Danas se najčešće koristi za razvoj veb i mobilnih (Android) aplikacija. Prva verzija se pojavila 1995. godine od strane Sun Microsystems kompanije, koja je sada u vlasništvu Oracle korporacije. Trenutna verzija programskog jezika Java je Java 8.

Izvršavanje Java programa

U odnosu na objašnjenja data u prethodnom poglavlju, izvršavanje Java programa ima svoje specifičnosti. Razlog je taj što se Java program ne izvršava direktno od strane procesora računara. Umesto toga, koristi se specijalizovan softver kao posrednik. Ovaj specijalizovani softver se naziva Java virtuelna mašina (JVM). Kod napisan u programskom jeziku Java se prevodi putem Java kompajlera u takozvani bajt kod. Za razliku od izvornog Java koda, bajt kod nije čitljiv za čoveka, ali ne sadrži ni opis instrukcija konkretne procesorske arhitekture da bi se mogao izvršiti od strane procesora. Umesto toga, bajt kod je čitljiv za Java virtuelnu mašinu. Java virtuelna mašina je program koji izvršava ovakav kod. Izvršavanje podrazumeva pretvaranje instrukcija programa zapisanih u formi bajt koda u instrukcije koje konkretan procesor prepoznaje, i slanje ovih instrukcija procesoru na izvršavanje. Opisano je ilustrovano na narednoj slici.



*preuzeto sa viralpatel.net

Ovakvo rešenje omogućuje platformsku nezavisnost prevedenog Java programa. Obzirom da ne sadrži kod koji je specifičan za određenu arhitekturu procesora, bajt kod je moguće izvršiti na bilo kojem procesoru. Ovo je moguće zato što se bajt kod ni ne interpretira direktno od strane procesora, nego od strane JVM. Tako da je jedan Java program moguće izvršiti na bilo kojem računaru na kojem postoji instalirana JVM. Obzirom da JVM mora da prevede bajt kod u instrukcije određenog procesora, neophodno je da implementacija JVM bude specifična za određeni procesor. Takođe, obzirom da se program za određene funkcije mora obratiti operativnom sistemu, Java program je nezavisan od operativnog sistema na kojem će se izvršavati. Obzirom da je JVM zadužena za komunikaciju sa operativnim sistemom, neophodno je da na računaru postoji instalirana verzija JVM koja odgovara konkretnom operativnom sistemu.

Ovakav platformski nezavisan pristup ima jedan značajan nedostatak. Obzirom da se koristi JVM kao poseban program koji interpretira Java program, brzina izvršavanja Java programa je u pravilu manja od programa pisanih u programskim jezicima koji se direktno prevode u mašinski kod. Iz tog razloga, Java nije pogodan jezik za pisanje programa kod kojih su važne performanse pri komunikaciji sa hardverom računara.

JVM se distribuira kao softverska aplikacija u dve osnovne varijante – JRE i JDK.

Java Runtime Environment (JRE) predstavlja softverski paket koji sadrži neophodne komponente za izvršavanje Java programa prevedenih u bajt kod. Neophodno je da računar ima instaliran JRE da bi mogao da izvršava Java programe.

Java Development Kit (JDK) predstavlja nadskup JRE i sadrži i alate neophodne za razvoj Java programa. Na primer, sadrži Java kompajler za prevođenje izvornog koda u bajt kod.

JDK se može preuzeti na adresi http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html.

4. Podatak i promenljiva

Svrha programa je da obrađuje podatke. Podatak možemo definisati kao informaciju formalno predstavljenu u obliku pogodnom za obradu i prenos.

U višim programskim jezicima, podaci se predstavljaju kao promenljive. Promenljiva je apstrakcija podatka kako bi se podatku lakše pristupalo i njime upravljalo. Promenljiva je definisana

- skladišnom lokacijom u koju se podatak smešta
- simboličkim imenom (identifikatorom) kojim se lokacija podatka na apstraktnom nivou imenuje u programskom jeziku
- vrednošću samog podatka

Programski jezik Java koristi promenljive kao osnovnu jedinicu predstavljanja podataka. Podaci mogu imati različite vrednosti, pri čemu vrednosti možemo grupisati po tipu. Tip određuje skup mogućih vrednosti koje se mogu uskladištiti u promenljivoj. Tako imamo podatke koji su tipa

- celi broj (npr. 12, 178, -9)
- decimalni broj (npr. 17.5, 0.66, -9.5)
- karakter (npr. A, a, x, ?, _)
- itd.

U programskom jeziku Java, promenljiva je definisana identifikatorom i tipom podatka koji se skladišti u promenljivoj. U ovako definisanu promenljivu moguće je upisati podatak odgovarajućeg tipa.

Identifikator promenljive

Za identifikator promenljive (i drugih imenovanih elemenata u Java programu) mogu se koristiti različite kombinacije karaktera. Preciznije, dozvoljeno je koristiti slova, cifre, znak _ i znak $. Nisu dozvoljena prazna mesta. Takođe, identifikator ne može da počne cifrom. Pored toga, za identifikatore nije moguće koristiti rezervisane reči u Javi. To su reči koje imaju posebno značenje u sintaksi jezika. Neke od rezervisanih reči su class, for, int, double, if, while, ...

Važno je napomenuti da Java razlikuje velika i mala slova u identifikatoru. To znači da se na primer promenljive *index* i *Index* tretiraju kao dve različite promenljive.

Deklaracija i dodela vrednosti promenljivoj

Za definisanje promenljive, potrebno je napisati naredbu. Generalno, program se sastoji iz niza naredbi. Kod većine naredbi u Javi, piše se znak ; na kraju naredbe. Konkretno, promenljiva se definiše na sledeći način:

```
tipPromenljive nazivPromenljive;
```

Na primer, obzirom da se celobrojni tip podatka u Javi naziva *int*, promenljiva koja se zove *br* i namenjena je smeštanju celih brojeva se navodi kao:

```
int br;
```

U ovako definisanu promenljivu moguće je upisati vrednost. Vrednost se postavlja korišćenjem operatora dodele. Simbol za operator dodele u Javi je znak =. Operator dodele vrši prebacivanje vrednosti izraza sa desne strane operatora u promenljivu koja se nalazi na levoj strani operatora. Tako, ako želimo da u promenljivu *br* upišemo vrednost 9, napisaćemo:

```
br = 9;
```

Treba primetiti da sada nije naveden tip promenljive *br*, jer se tip promenljive navodi samo prvi put pri navođenju promenljive. Ovo prvo navođenje zovemo deklaracija promenljive. U Javi, da bi se promenljiva mogla koristiti, obavezno je prvo izvršiti njenu deklaraciju u kojoj se navodi tip promenljive. Za programske jezike koji imaju ovu osobinu kažemo da su statički tipizirani (za razliku od dinamički tipiziranih programskih jezika u kojima se tip promenljive može dinamički odrediti na osnovu vrednosti upisane u promenljivu bez potrebe navođenja tipa promenljive). Deklaraciju je moguće izvršiti u delu programskog koda gde želimo da promenljivu koristimo (za razliku od nekih programskih jezika u kojima postoji posebna sekcija koda u kojoj moraju da stoje sve deklaracije promenljivih).

Moguće je zajedno sa deklaracijom promenljive postaviti i početnu vrednost promenljive. Ovo se naziva inicijalizacija promenljive. Prethodni primer bi se mogao zameniti jednim redom programskog koda na sledeći način:

```
int br = 9;
```

Tipovi podataka u programskom jeziku Java

Kao što je objašnjeno, moguće je definisati podatke različitog tipa. Podržane tipove podataka možemo grupisati u dve osnovne grupe. Tako tipove podataka delimo na

- primitivne (proste) tipove i
- složene tipove

Primitivni tipovi su osnovne jedinice predstave podataka i skladište jednu prostu vrednost. Npr. jedan broj. Složeni tipovi u osnovi predstavljaju skup primitivnih vrednosti, iako je sam koncept kompleksniji od grupisanja podataka primitivnog tipa. Pojam složenog tipa je u Javi neodvojiv od koncepata objektno-orijentisanog programiranja, pa u ovom delu teksta neće biti dalje objašnjavan.

U tabeli su navedeni primitivni tipovi u programskom jeziku Java.

Naziv tipa	Opis tipa
byte	Celi broj između -128 i 127
short	Celi broj između -32,768 i 32767
int	Celi broj između -2^{31} i $2^{31} - 1$. Ovaj tip se najčešće koristi za cele brojeve.
long	Celi broj između -2^{63} i $2^{63} - 1$. Ovaj tip se koristi za veoma velike cele brojeve koji izlaze iz opsega predviđenog int tipom
float	Realni broj. Decimalni brojevi se u računaru predstavljaju u formi $1.xxxxx * 2^{yyy}$. Tako da se za ovaj broj skladišti baza (xxxxx) i eksponent (yyy).
double	Realni broj. Zauzima duplo više mesta u memoriji u odnosu na float. Obzirom da realni brojevi mogu imati beskonačan broj decimalnih mesta, double tip može da vrednost izrazi duplo preciznije nego float tip.

boolean	Logička vrednost – tačno ili netačno.
char	Karakter. Izražen prema Unicode kodnom sistemu koji omogućuje predstavu karaktera iz većine pisanih sistema koji se na svetu koriste. To znači da su podržana i slova iz srpske azbuke i abecede.

U nastavku je prikazano kako se korišćenjem ovih tipova podataka vrši inicijalizacija promenljivih.

```
byte b = 100;
short s = 10000;
int i = 100000;
long x = 22334L;
float f = 122.4f;
double d = 224.17;
boolean result = true;
char capitalC = 'C';
```

Izrazi prikazani sa desne strane znaka jednakosti u prethodnom primeru nazivaju se literali. Literal je reprezentacija fiksne vrednosti. Treba primetiti da se literal tipa char definiše korišćenjem znakova '. Takođe, za literal tipa long je potrebno dodati sufiks L, a za float f.

Iako je objašnjeno da će o složenim tipovima biti reči kasnije, ovde će biti spomenut jedan važan složeni tip. Kada je potrebno predstaviti niz karaktera, koristi se složeni tip String. I za ovaj tip je moguće definisati literal korišćenjem znakova ". Inicijalizacija promenljive ovog tipa prikazana je u nastavku.

```
String s = "Novi Sad";
```

U okviru stringa moguće je definisati i specijalne karaktere koji su označeni znakom \. Na primer, pošto navodnici označavaju kraj stringa, ukoliko želimo da string sadrži navodnike, to ćemo napisati kao \". Pored toga, znak \n predstavlja novi red u stringu, a znak \t predstavlja TAB karakter u stringu.

Konverzija tipova

Ranije je objašnjeno da operator dodele vrši upis vrednosti izraza koji stoji sa desne strane operatora u promenljivu koja se nalazi na levoj strani operatora. U principu, da bi ova operacija mogla da se izvrši, neophodno je da izraz sa desne strane ima rezultat istog tipa kao promenljiva u koju vrednost treba da bude upisana. Zato su dodele prikazane u prethodnim primerima ispravne, a naredni primer bi rezultovao greškom pri kompajliranju programa.

```
double d = "Novi Sad";
```

Ipak, naredni primer neće izazvati grešku, iako je vrednost sa desne strane tipa int, a promenljiva tipa double.

```
double d = 34;
```

Razlog je taj što u ovoj situaciji kompajler vrši implicitnu konverziju int vrednosti u double. Ovo je moguće zato što se int može posmatrati kao podtip tipa double.

S druge strane, naredni primer će izazvati grešku pri kompajliranju.

```
double d = 13.46;
int x = d;
```

Tip double je širi tip od tipa int i nije moguće implicitno izvršiti konverziju iz jednog tipa u drugi, jer bi moglo doći do gubitka podataka. Ipak, moguće je eksplicitno naznačiti kompajleru da ovu konverziju izvrši. Ovo se naziva eksplicitna konverzija i vrši se *cast* operatorom. *Cast* operator u zagradama navodi ime tipa u koji je potrebno konvertovati vrednost. Primer korišćenja cast operatora pri eksplicitnoj konverziji double vrednosti u int prikazan je u narednom primeru.

```
double d = 13.45;
int x = (double) d;
```

Blok koda i doseg promenljive

U Java programu kod je organizovan u blokove koda. Blok može da predstavlja određeni element Java jezika, npr. telo petlje, metodu, klasu. Ovo su sve pojmovi koji će kasnije biti objašnjeni. Bez obzira na značenje bloka koda, kod se definiše između znakova { i }. Blok koda se ne završava znakom ;.

Ranije je pomenuto da promenljiva može da se koristi nakon što se deklariše. Drugo pitanje je do kada može da se koristi. Promenljiva može da se koristi od deklaracije, pa do kraja bloka koda u kojem je deklarisana. Ovo se naziva doseg promenljive (eng. *scope*). Primer bloka koda i dosega promenljive je prikazan u nastavku.

```
{
      x = 5;
      int x;
      x = 5;
}
x = 5;
```

Greška! Promenljiva nije deklarisana

Greška! Promenljiva više ne može da se koristi

5. Struktura Java programa

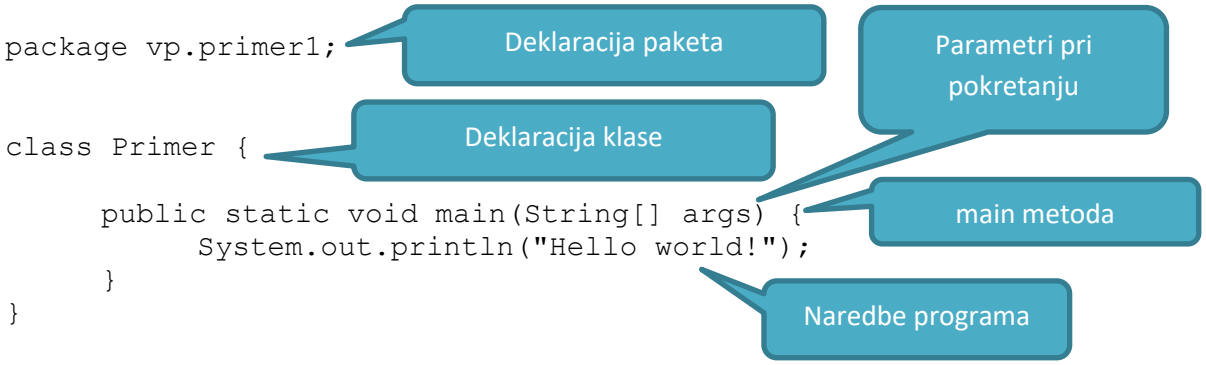
Java program se sastoji od niza naredbi organizovanih u blokove koda.

Osnovna jedinica organizacije koda u Javi je klasa. Klasa je pojam iz objektnog programiranja i, obzirom da ovi pojmovi još nisu objašnjavani, možemo za sada posmatrati klasu kao jedan imenovani blok koda. Kod klase se piše u fajlu sa ekstenzijom `.java`. U jednom fajlu je dozvoljeno definisati jednu klasu, pri čemu ime fajla i ime klase moraju biti jednaki.

Klase se grupišu u pakete. Paket je imenovana jedinica organizacije klasa. Svaki paket odgovara jednom folderu na disku. Ovaj folder sadrži fajlove u kojima se nalaze klase za koje je definisano da pripadaju tom paketu. U isti paket se najčešće grupišu logički srodne klase. U skladu sa organizacijom foldera na disku, i paketi su hijerarhijski organizovani. Tako jedan paket može imati više potpaketa.

Klasa sadrži blokove koda organizovane u metode. Ovo je takođe pojam iz objektnog programiranja, koji za sada može biti smatran imenovanim blokom koda koji izvršava određeni niz naredbi. Poseban status pri izvršavanju programa ima metoda sa nazivom `main`. Ukoliko klasa sadrži ovu metodu, klasu je moguće pokrenuti i pri startovanju programa krenuće da se izvršava programski kod definisan u `main` metodi.

Java program ima unapred definisanu strukturu. Ova struktura prikazana je u nastavku na primeru.



Deklaracija paketa mora biti prva linija u Java fajlu. Ona označava naziv paketa kojem klasa pripada. U konkretnom primeru, klasa se nalazi u paketu primer1, koji je potpaket paketa vp. To znači da se prikazani fajl mora na disku nalaziti u folderu primer1 koji se nalazi unutar foldera vp.

Nakon toga dolazi import sekcija koja je u prikazanom primeru prazna. Ova sekcija će biti objašnjena kasnije i tiče se navođenja drugih klasa koje ova klasa koristi.

Nakon import sekcije, dolazi deklaracija klase putem rezervisane reči class. Klasa mora da se zove isto kao i fajl u kojem se nalazi.

Unutar klase se definiše njen sadržaj. Za sada ćemo se zadržati na main metodi kao jedinom sadržaju klase. Ovde se navodi proizvoljan niz naredbi koji će se izvršiti pri pokretanju programa. U prikazanom primeru, definisana je naredba za ispisivanje teksta `Hello world!` na ekran.

Pri pokretanju programa moguće je poslati niz ulaznih parametara programu. Ovo je slučaj za bilo koji program na računaru. Parametri se pri pokretanju navode tako što se nakon naziva programa navedu vrednosti parametara programa. Program može onda da prilagodi ponašanje u skladu sa ulaznim parametrima (npr. pri pokretanju *browsera* se prosledi parametar *start-maximized* da bi se *browser* startovao sa maksimizovanim prozorom).Ako je reč o Java programu, ovi parametri će biti dostupni u `args` promenljivoj definisanoj u zaglavlju `main` metode.

6. Kompajliranje i izvršavanje Java programa

Program prikazan u prethodnom primeru moguće je izvršiti. Za to je neophodan preduslov da se program prvo prevede u bajt kod. Prevođenjem .java fajla u bajt kod dobija se fajl sa ekstenzijom .class.

Pri instalaciji Java virtuelne mašine u JDK verziji, instalira se i program pod nazivom javac. Kompajliranje se vrši putem ovog programa. Potrebno je iz komande linije (command-prompt na Windows operativnom sistemu) izvršiti ovaj program i kao parametar proslediti naziv .java fajla koji

se prevodi. Primer je u nastavku za slučaj kada je Java instalirana na navedenoj lokaciji i ako se kompajlira klasa Primer prikazana u prethodnom primeru na Windows operativnom sistemu.

```
C:\temp\vp\primer1>"c:\Program Files\Java\jdk1.7.0_67\bin\javac.exe" Primer.java
```

Kao što smo objasnili, izvršavanjem prikazane komande dobija se fajl Primer.class.

Prevedeni Java program se izvršava programom java, koji se takođe instalira pri instalaciji java virtuelne mašine (i JDK i JRE varijanta sadrže ovaj program). Izvršavanje programa se vrši izvršavanjem programa java, pri čemu se ime klase koja se izvršava prosleđuje kao parametar. Da bi se klasa mogla izvršiti, mora da sadrži main metodu. U nastavku je primer pokretanja klase kompajlirane u prethodnom primeru na Windows operativnom sistemu.

```
C:\temp>"c:\Program Files\Java\jdk1.7.0_67\bin\java.exe" vp.primer01.Primer
```

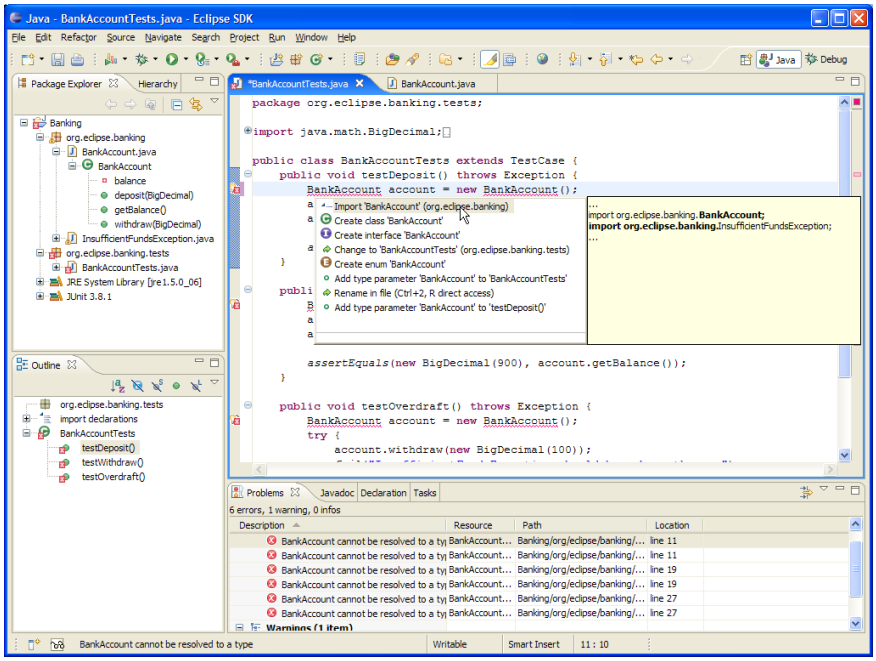
Da ne bismo morali svaki put pisati punu putanju do programa java i javac, obično se putanje do ovih programa navedu u PATH promenljivoj okruženja operativnog sistema (eng. *environment variables*). Ako je u komandnoj liniji navedeno samo ime programa, operativni sistem će automatski potražiti navedeni program u folderima koji su navedeni u PATH promenljivoj.

Važno je napomenuti i kako java virtuelna mašina pronalazi klase koje treba da izvrši. U promenljivoj okruženja CLASSPATH se navode folderi u kojima će JVM tražiti klase koje program izvršava ili ih koristi pri izvršavanju. Ako ova promenljiva nije navedena, podrazumevano će JVM tražiti klase u trenutnom folderu. Iz ovog razloga je pokretanje prethodnog primera ispravno radilo iako nije podešavana CLASSPATH promenljiva.

7. Eclipse IDE

Java programi su obični tekstualni fajlovi tako da mogu biti pisani u bilo kojem tekstualnom editoru (Notepad, Gedit, Ultraedit, Sublime Text, ...). Takođe, napisane programe je moguće kompajlirati i izvršiti korišćenjem komandne linije. Ipak, najčešće se koriste specijalizovani softverski alati za razvoj i izvršavanje Java programa. Ovi alati se zajednički nazivaju IDE – Integrated Development Environment.

Jedno od najčešće korišćenih razvojnih okruženja za Java aplikacije je Eclipse. Eclipse je besplatan IDE otvorenog koda. Najnovija verzija Eclipse za razvoj Java veb aplikacija dostupna je na adresi https://eclipse.org/downloads/eclipse-packages/ na linku „Eclipse IDE for Java EE Developers“.



*preuzeto sa www.eclipse.org

Eclipse sadrži tekstualni editor za prikaz i pisanje programskog koda. Ima ugrađeni kompajler koji vrši kompajliranje koda. Greške i upozorenja se prikazuju u editoru. Editor ima podršku za automatsko dopunjavanje teksta (*auto-completion*) koje je zavisno od konteksta. Takođe, moguće je automatski formatirati programski kod, kao i izgenerisati delove koda.

Pored tekstualnog editora, Eclipse sadrži različite panele koje korisnik može slobodno da raposređuje. Najčešće korišćeni panel je Package explorer (na slici na levoj strani) koji prikazuje strukturu projekata, paketa i klasa u projektu. Eclipse ima predefinisane rasporede panela koje naziva perspektiva. Perspektiva predstavlja pogled na projekat i za svaki tip projekta ili tip zadatka postoji odgovarajuća perspektiva. Za razvoj Java aplikacija se najčešće koristi Java perspektiva. Perspektiva se može promeniti klikom na Window stavku glavnog menija i opciju Open perspective.

Osim za kompajliranje, Eclipse ima ugrađenu podršku i za pokretanje Java programa. Potrebno je kliknuti desnim tasterom miša na klasu koju želimo da pokrenemo (u Package exploreru ili na kodu klase u tekst editoru) i izabrati opciju Run As->Java Application.

Kreiranje novog projekta vrši se preko glavnog menija stavkama File->New->Project… Moguće je izabrati različite tipove projekata, pri čemu se za standardan Java projekat koristi opcija Java->Java Project.

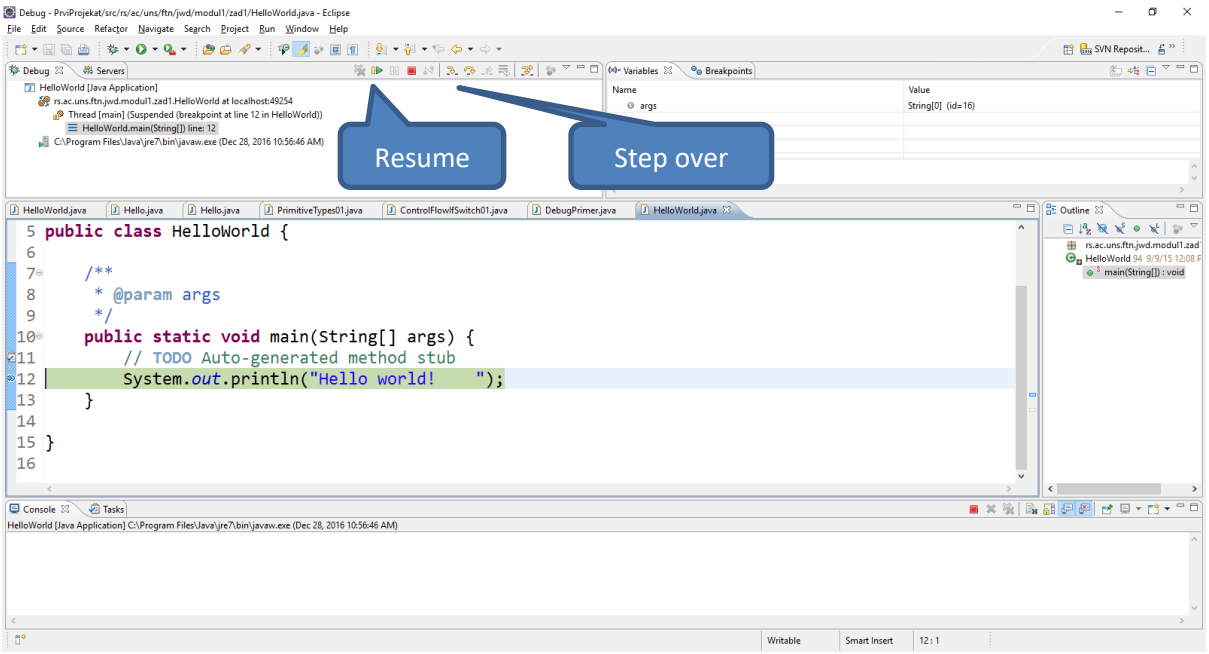
Postojeći projekat se sa diska ubacuje u Eclipse opcijom File->Import existing project into workspace… Nakon toga se klikom na browse izabere folder u kojem je željeni eclipse projekat (to je folder koji sadrži eclipse .project fajl).

Debug programskog koda u Eclipse IDE

Eclipse pruža i podršku za debug programskog koda. Debug predstavlja praćenje izvršavanja programa, najčešće da bi se otklonila greška u programu. Za otklanjanje greške, prvi korak je lociranje greške. Ovo se najčešće radi tako što se program izvršava liniju po liniju dok se prate vrednosti promenljivih i tok programa.

Da bi se Debug izvršio, najpre je potrebno postaviti tačku u programu gde će se program zaustaviti pri izvršavanju. Ovo nazivamo prekidna tačka (eng. *break point*). Prekidna tačka se postavlja klikom desnim tasterom miša na prostor sa leve strane tekstualnog editora pored linije na kojoj želimo da postavimo prekidnu tačku. Nakon klika, izabere se opcija Toggle Breakpoint. Na isti način se prekidna tačka i sklanja.

Kada je postavljena prekidna tačka, može se pokrenuti Debug programa klikom desnim tasterom miša na klasu i izborom opcije Debug As->Java Application. Eclipse će nakon toga preporučiti otvaranje Debug perspektive kao pogodnije perspektive za praćenje izvršavanja programa. Eclipse u ovoj perspektivi je prikazan na slici.



Perspektiva prikazuje u kojoj liniji je program trenutno zaustavljen (ta linija je markirana). U gornjem desnom uglu je panel na kojem se prikazuju trenutne vrednosti promenljivih u kodu. Klikom na dugmad “Step over” može se izvršiti naredna linija koda. Klikom na dugme Resume, izvršavaju se sve linije do naredne prekidne tačke.

Dakle, debug proces se vrši tako što se izvršavaju delovi koda i prati se tok programa i vrednosti promenljivih kako bi se utvrdilo na kom mestu program počinje da se neočekivano ponaša.

8. Ispis na ekran

Ispis poruke na konzolu se vrši naredbom

```
System.out.println(tekst za ispis);
```

Ova naredba ispisuje tekst i prelazi na novi red. Ukoliko je potrebno samo ispisati tekst bez prelaska na novi red, koristi se naredba

```
System.out.print(tekst za ispis);
```

Kao tekst za ispis može da se pošalje literal ili promenljiva. Ako je reč o stringovima, moguće je vršiti spajanje stringova operatorom +. Tako naredni primer ispisuje tekst *21000 Novi Sad.*

```
int ptt = 21000;
System.out.println(ptt + "Novi" + " Sad");
```

9. Aritmetički operatori

Java podržava standardne aritmetičke operatore sabiranja, oduzimanja, množenja i deljenja. Simboli za operatore su +,-, * i /. Primer korišćenja aritmetičkih operatora je dat u nastavku.

```
int b = 5;
int c = 2;
int a = b + c;
```

Promenljiva `a` će imati vrednost 7 nakon izvršavanja prethodnog primera.

Ukoliko je na postojeću vrednost promenljive potrebno dodati novu vrednosti, to se može napisati kao

```
a = a + 3;
```

Ovo se može i kraći zapisati kao

```
a += 3;
```

Pored operatora += , postoje i operatori -=, *= i /= koji rade po istom principu kao operator +=, samo što vrše operacije oduzimanja, množenja i deljenja.

Specifično, ako je na postojeću vrednost promenljive potrebno dodati vrednost 1, to se može zapisati i kao

```
a++;
```

Operator ++ se zove inkrement operator i efekat primene operatora na promenljivu je uvećavanje promenljive za 1. Inkrement operator ima svoju prefiksnu i postfiksnu varijantu, tj. moguće je operator napisati pre i posle promenljive. U oba slučaja efekat operatora je isti – promenljiva se uvećava za 1. Razlika između prefiksnog i postfiksnog operatora je rezultat koji će izraz da vrati. Konkretno kod prefiksnog inkrement operatora, rezultat je vrednost promenljive uvećana za 1. Kod postfiksnog inkrement operatora, rezultat je originalna vrednost promenljive pre uvećavanja. Razjasnićemo ovo na narednom primeru koda.

```
int x = 10;
int y = 10;
int b = ++x;
int c = y++;
```

Nakon izvršavanja koda iz primera, promenljive x i y će obe imati vrednost 11, jer je nad njima primenjen inkrement operator koji uvek ima efekat da promenljiva bude uvećana za 1. Dilema je koje će vrednosti imati promenljive b i c obzirom da su one dobile vrednost izraza koje vraća prefiksni, odnosno postfiksni inkrement operator. Promenljiva b će imati vrednost 11, jer prefiksni inkrement operator kao rezultat vraća novu vrednost promenljive. Promenljiva c će imati vrednost 10, jer postfiksni inkrement operator vraća originalnu vrednost promenljive pre uvećavanja.

Slično kao što postoji inkrement operator, postoji i dekrement operator za umanjenje promenljive za 1. Simbol za ovaj operator je --.

Ovde ćemo pomenuti još i moduo operator % koji vraća ostatak pri deljenju dva cela broja. Tako na primer izraz 7%2 vraća rezultat 1, jer 7 podeljeno sa 2 daje rezultat 3 i ostatak 1.

10. Komentari

Pored programskog koda koji se izvršava, program treba da sadrži i dodatne komentare koji objašnjavaju programski kod. Komentari se pišu u formi prirodnog jezika i računar ih ne interpretira pri izvršavanju programa. Komentari bi trebali da povećaju čitljivost koda. Važno je naglasiti da komentari ne treba da forme programskog jezika izražavaju u prirodnom jeziku, nego treba da objašnjavaju koja je funkcija određenog dela programskog koda.

U Javi postoje dve vrste komentara. Ako je ceo tekst komentara u jednoj liniji, to se naziva jednolinijski komentar. Ovakav komentar počinje znakovima // nakon čega sledi tekst komentara. Ako komentar ima više linija, onda se piše višelinijski komentar. Ovaj komentar počinje znakovima /*. Nakon toga sledi tekst komentara koji može da ima više linija, a na kraju komentara se znakovima */ označava da je komentar završen. U nastavku je dat primer jednolinijskog i višelinijskog komentara.

```
int brojac = 0; // promenljiva za evidenciju trenutnog studenta
/*
Prolazak kroz kolekciju studenata i
pronalaženje studenta sa najboljim prosekom
*/
```

11. Konvencije pisanja Java programskog koda

Iako je Java programski jezik u kojem je slobodan format pisanja programskog koda, strogo se preporučuje poštovanje određenih konvencija kako bi kod bio čitljiv.

Identifikatori se u Javi najčešće definišu korišćenjem *camel case* notacije. U ovoj notaciji, ako se identifikator sastoji iz više reči, reči se pišu spojeno pri čemu svaka nova reč počinje velikim slovom. Dat je primer jedne promenljive imenovane u skladu sa ovom notacijom.

```
int maxElement = 0;
```

Druga važna stvar je poravnanje programskog koda. Počeci naredbi koje se nalaze u istom bloku treba da budu vertikalno poravnati. Obzirom da blokovi koda mogu da se ugnježdavaju, naredbe koje se pišu u podbloku treba da budu pomerene za jedan TAB karakter u odnosu na naredbe iz nadbloka. Dat je primer poravnjanja programskog koda.

```
int a = 5;
int b = 3;
{
      int c = 4;
      int d = 5;
}
int e = 6;
```