Poglavlje 2. Brojevi i znakovni nizovi (strings)

U prvom poglavlju pisali smo programe koji su samo ispisivali poruke na ekran. U ovome poglavlju uvodimo nove izraze koje ćemo koristiti u metodama. To uključuje upotrebu dvije vrste vrijednosti: brojevi i znakovni nizovi (strings).

SADRŽAJ

---

1. Cjelobrojne vrijednost (integers)

U prvom poglavlju smo spomenuli da nisu svi podaci u Javi objekti. Najjednostavniji tipovi podataka, poput cijelih brojeva ili brojeva u pokretnom zarezu odstupaju od logike da sve treba biti objekt. Takvi tipovi podataka nazivaju se *primitivni tipovi podataka*. Ovo poglavlje počinjemo primjerom upotrebe cijelih brojeva (integers).

Pretpostavimo da idete kupovati na tržnici. Kupili ste 2 kg jabuka po 12 kuna, 3kg krumpira po 2kune i 50lipa i 2 kg bresaka po 14 kuna. Koliko ste kuna potrošili.

Nije potrebna snaga Jave na modernom računalu za izračunati navedeni primjer jer i napamet (ili bar uz pomoć papira i olovke) da se izračunati je ukupan račun 85.5 kune.

Međutim program će rasvijetliti nekoliko točaka u računanju s cijelim brojevima.

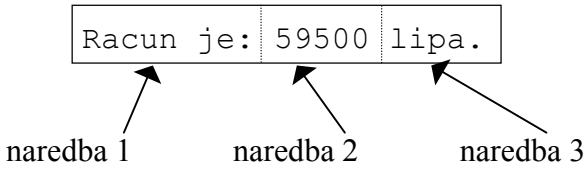
PRIMJER 1

```
public class Racun1

{  /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
   3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */

   public static void main(String[] args)
   {  System.out.print("Racun je: ");
      System.out.print(2*1200 + 3*250 + 2*1400);
      System.out.println(" lipa.");
   }
}
```

Poput svih primjera za ovo poglavlje program se sastoji od jedne klase koja sadrži samo jednu metodu `main`. U ovome primjeru metoda sadrži tri naredbe. Kad se izvrši tri metode će proizvesti dijelove linije na ekranu kako je to prikazano na slici ispod.
Primijetite da prve naredbe koriste `print` umjesto `println` što znači da ne pomiču kursor u novi red nakon ispisa.



Kada Java izvrši naredbu 2 , izračunat će vrijednost slijedećeg aritmetičkog izraza:

```
2*1200 + 3*250 + 2*1400
```

i prikazati rezultat 59500. Primijetite da možemo ispisivati brojeve koristeći `System.out.print` na isti način na koji smo koristili kod ispisivanja znakovnih nizova.

U Java programu slijedeći operatori mogu se koristiti u aritmetičkim izrazima:

+	zbrajanje (addition)
–	oduzimanje (subtraction) (koristi se i za minus predznak)
*	množenje (multiplication)
/	dijeljenje (division)
%	ostatak (remainder)

U Javi (i većini drugih programskih jezika višeg nivoa) brojevi su **cjelobrojni** ili **brojevi u pokretnom zarezu** (integers or floating-point numbers). Ove dvije vrste vrijednosti pohranjuju se u memoriji na različite načine. Ako vrijednost napišemo bez decimalne točke , npr. 4 ona će biti tretirana kao cjelobrojna vrijednost. Ako je napišemo s decimalnom točkom npr. 4.0 bit će tretirana kao broj u pokretnom zarezu. Uvijek trebamo koristiti cijele brojeve kad to možemo. Računanja s njima su brža, koriste manje memorije i nisu podložni greškama zaokruživanja poput brojeva s pokretnim zarezom.

Operator dijeljenja , / , u stvari ima dva značenja. Ako ga koristimo s cijelim brojevima tada znači **cjelobrojno dijeljenje** . Npr. 11 / 4 = 2, (–11) / 4 = –2. Ako se koristi s dva broja s pokretnim zarezom onda znači normalno dijeljenje.
Npr. 10.5 / 2 = 5.25. Operator ostatka koristi se samo s cijelim brojevima. Npr. 11%4 = 3 and (–11)%4 = –3.

Ako Javi damo da izračuna izraz m%n gdje je n jednak nuli , Java će prijaviti run-time grešku.

kada procesira složenije izraze Java se drži standardnog prioriteta operatora pa npr. izraz 3+4*5 znači isto kao i 3+(4*5).

Kada se izračunava izraz sa više operatora istog prioriteta u nizu, izraz se računa s lijeva nadesno.

Npr. 3/4/5 znači isto kao i (3/4)/5, na kao 3/(4/5) što bi proizvelo grešku (Zašto?).

---

2. Varijable

Slijedeći primjer je druga verzija programa prvog primjera. Ovaj primjer koristi **varijable** tj. naziv koji znači vrijednost.
Program izračunava ukupan račun u dva odvojena koraka.
PRIMJER 2

```
public class Racun2

{  /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
   3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */

   public static void main(String[] args)
   {  int total =2*1200 + 3*250 + 2*1400;
      System.out.print("Racun je: ");
      System.out.print(total/100);
      System.out.print(" kn i ");
      System.out.print(total%100);
      System.out.println(" lipa.");
   }
}
```

Izraz:

```
int total =2*1200 + 3*250 + 2*1400;
```

kaže:
kreiraj varijablu `total` koja će sadržavati vrijednost izraza `2*1200 + 3*250 + 2*1400` (=59500). Riječ `int` na početku pokazuje da je tip vrijednosti koju varijabla `total` može pohraniti je cjelobrojna vrijednost.

Ovaj izraz se naziva **deklaracija varijable (ili definicija varijable).**

Izraz:

```
System.out.print(total/100);
```

kaže: ispiši vrijednost naznačenu kao `total/100`. Radi se o cjelobrojnom dijeljenju čija je vrijednost 85 i ta će vrijednost biti ispisana.

Izraz:

```
System.out.print(total%100);
```

kaže: ispiši ostatak dijeljenja `total` sa 100. U ovome slučaju je to 50.

Ukupan ispis na ekran bi trebao biti:

Racun je:	59	kn i	50	lipa.

Ovaj primjer smo mogli realizirati i bez varijable s upotrebom slijedeće dvije naredbe:

```
System.out.print((2*1200 + 3*250 + 2*1400)/100);
System.out.print((2*1200 + 3*250 + 2*1400)%100);
```

Upotrebom varijable izbjegli smo dvostruko računanje ukupnog broja lipa. U ovako malom promjeru to je mala ušteda, ali u realnim primjerima mogu se postići značajne uštede.

Drugi razlog upotrebe varijabli je da se jasno označe vrijednosti koje se koriste u računu tako da je jasnije što se u programu radi. Za nazive varijabli možete koristiti bilo koji naziv koji već nije upotrebljen za nešto drugo. U praksi upotrebljavaju se opisni nazivi koji odgovaraju ulozi varijable.

U slijedećoj verziji programa broju kilograma pojedinog artikla dani su očiti nazivi `jabuka, krumpir, breskva.` Usput , prihvaćena je konvencija da nazivi varijabli uvijek počinju s malim slovom.

PRIMJER 3

```
public class Racun3

{  /* Izracunaj ukupni racun za 2 kg jabuka po 12 kn
   3kg krumpira po 2kn i 50lipa i 2 kg bresaka po 14 kn */

   public static void main(String[] args)
   {  int jabuka = 2;
      int krumpir = 3;
      int breskva = 2;

      int total =
         jabuka*1200 + krumpir*250 +breskva*1400;

      System.out.print("Racun je: ");
      System.out.print(total/100);
      System.out.print(" kn i ");
      System.out.print(total%100);
      System.out.println(" lipa.");
   }
}
```

Ovaj program daje izlaz isto kao i prethodni. Upotreba varijabli omogućava bolje razumijevanje koda bilo vama ili nekome drugome koji će nakon vas modificirati kod.
U programiranju postoji pojam "magični broj". To su razne konstante u programu za koje prije ili kasnije se zaboravi što znače.

Kada je varijabla kreirana za nju se alocira prostor u memoriji gdje se potom može spremiti vrijednost varijable. Količina memorije koja se alocira ovisi o tipu varijable. Npr. za tip varijable `int`, kao što je to u slučaju varijable `total` , bit će alocirana 32 bita memorije.

Format pohrane je rijetko važan programeru. Bitniji je opseg vrijednosti koji se može pohraniti u zadanu varijable. Vrijednost `int` varijable može biti u području od –2147483648 do 2147483647. Ako su potreban veći opseg vrijednosti potrebno je umjesto `int` upotrijebiti `long` tip varijable. Taj tip zauzima 64 bita, a opseg je od -9223372036854775808 do 9223372036854775807. Postoje i kraći tipovi cjelobrojnih varijabli `short` (16 bita) i `byte` (8 bita).

---

3. **Dodjeljivanje** (Assignments)

Kao što smo vidjeli, svaka varijable ima naziv i odgovarajući dio memorije. Vrijednost koja je sadržana u memoriji je **dodijeljena** varijabli. Ta vrijednost ne mora biti ista u tijeku izvršavanja programa. Može biti zamijenjena drugom vrijednošću u tijeku izvršavanja programa. U slijedećem primjeru promijenit ćemo sadržaj varijable.

PRIMJER 4

```
public class Mjenjacnica1

{  /* Preracunaj marke u eure */

   public static void main(String[] args)

   {  int maraka = 110;
      int pfeniga = 55;

      pfeniga = 100*maraka + pfeniga;
      double eura = pfeniga * 0.0051129;
      System.out.print("Eura: ");
      System.out.print(eura);
   }
}
```

Program započinje kreiranjem dviju varijabli nazvanih `maraka` i `pfeniga` koje su postavljene na određene iznose.
Nakon toga se izvršava slijedeći izraz:

```
pfeniga = 100*maraka + pfeniga;
```

Izraz znači: izračunaj vrijednost izraza `pfeniga = 100*maraka + pfeniga,` i pohrani rezultat u varijable `pfeniga`. Drugim riječima, ovaj izraz će zamijeniti originalnu

vrijednost pohranjenu u varijabli `pfeniga` (=55) s ukupnim iznosom koji imamo izraženim u pfenizima (= 11055). Nova vrijednost se koristi u slijedećem izrazu:

```
double eura = pfeniga * 0.0051129;
```

i predstavlja naše prvo korištenje brojeva u pokretnom zarezu.
Izraz znači: kreiraj varijablu nazvanu `eura` , tipa `double`, i dodijeli joj vrijednost `pfeniga * 0.0051129`. Varijabla tipa `double` sadržava broj u pokretnom zarezu *duple preciznosti* i alocira 64 bita memorije.
Produkt će se izračunati u aritmetici pokretnog zareza i rezultat će biti 56,5231095

Konačan ispis je:

```
Eura: 56.5231095
```

Izraz:

```
pfeniga = 100*maraka + pfeniga;
```

nazivamo **dodjeljivanje** vrijednosti .Ovaj izraz mijenja vrijednost *postojeće* varijable.

Kadgod Java obavlja dodjeljivanje ono se obavlja u dva koraka:

(1) Prvo se računa vrijednost s desne strane od znaka '='.
(2) Zatim se ta vrijednost sprema u varijablu naznačenu s lijeve strana izraza

Moguće je mijenjati vrijednost spremljenu u varijablu, ali nije moguće mijenjati tip (type) varijable.

Moguće je deklarirati varijablu bez specificiranja njene inicijalne vrijednosti. Ako Java izvrši slijedeći izraz:

```
int total;
```

kreirat će varijablu tipa `int` i dati joj naziv `total`, ali neće spremiti nikakvu vrijednost u istu. Vrijednost varijabli će biti potrebno dodijeliti kasnije u programu. U međuvremenu nije moguće znati što sadrži varijabla `total` . U njoj će biti neki bitovi koji su ostali u memoriji otprije. Kažemo da vrijednost varijable 'nije definirana'.

Opći oblik deklaracije varijable je dakle:

deklaracija (s inicijalnom vrijednosti)

TIP NAZIV = IZRAZ ;

Npr. *TIP* može biti `double`, *NAZIV* može biti `eura`, a *IZRAZ* može biti `64.45`.

dakle

```
double eura = 64.45;
```

deklaracija (bez inicijalne vrijednosti)

TIP NAZIV;

dodjeljivanje

NAZIV = IZRAZ;

Java omogućava deklaracije varijabli u kojima se istovremeno deklarira više varijabli odjednom.

Primjer: kreiranje varijabli `w`, `x`, `y` i `z`, i dodjeljivanje inicijalnih vrijednosti varijablama `x` i `z`.

```
int w, x = 1, y, z = 2;
```

Često je nekoj varijabli potrebno samo nadodati neku vrijednost i postoji skraćeno pisanje za takav tip operacije. Npr. za dodati `3*jabuka` varijabli `total` pisat ćemo slijedeći izraz:

```
total += 3*jabuka;
```

Slično možemo koristiti simbol `-=` kada želimo oduzeti vrijednost od varijable , `*=` kada želimo pomnožiti varijablu s izrazom , itd.

Općenito izraz:

`x OP= y;` ekvivalentan je izrazu
`x = x OP y;`
gdje je OP bilo koji binarni operator

Jedan od najčešćih slučajeva promjene vrijednosti varijable je povećavanje za jedan. Možemo pisati:

```
x = x+1;
```

ili kraće,

```
x += 1;
```

ili još kraće:

```
x++;
```

Slično za oduzimanje 1od `x`, pišemo `x--`.

---

4. Brojevi s pokretnom zarezom (Floating-point)

Naziv 'broj s pokretnim zarezom' proizlazi iz načina pohrane takvih brojeva u memoriji.

U Javi tip `double` se najčešće koristi za brojeve u pokretnom zarezu. Vrijednost tipa `double` zauzima 64 bita. to omogućava veoma velik opseg vrijednosti, približno $\pm 1.8 \times 10^{308}$, što izraženo preko preciznosti iznosi 15 značajnih znamenki. Naziv `double` je skraćenica za 'double precision floating-point'. Java posjeduje i drugi tip za rad s brojevima s pokretnim zarezom koji se naziva `float`. Taj tip podataka zauzima samo 32 bita tako da predstavlja ekonomičnije korištenje memorije, ali ima samo pola preciznosti tipa `double` te ekvivalentno manji opseg od oko $\pm 3.4 \times 10^{38}$.

Broj s pokretnim zarezom može se pisati s decimalnom točkom, npr. 14.56 ili 14.0. Može biti pisan i s eksponentom, npr. 14.56E-12.

Java sadržava niz metoda koje računaju širok opseg matematičkih funkcija koje koriste `double` vrijednosti. Sve su statičke metode (ne pripadaju nijednom objektu) iz klase Math koja se nalazi u Java biblioteci. Kadgod koristite statičku metodu koja pripada klasi biblioteke potrebno je kao prefiks naziva metode staviti i naziv klase.

`Math.sin(x)`	sinus od x
`Math.cos(x)`	cosinus od x
`Math.tan(x)`	tanges od x
`Math.exp(x)`	e^x
`Math.log(x)`	prirodni logaritam od x
`Math.abs(x)`	apsolutna vrijednost od x
`Math.floor(x)`	najveći cijeli broj $\leq$ x
`Math.ceil(x)`	najmanji cijeli broj $\geq$ x

U svakoj od metoda parametar x je tipa `double` , a rezultat je tipa `double`.

Java se neće buniti ako nađe vrijednost tipa `int` na mjestu gdje je predviđen tip `double`. Kadgod se to dogodi Java će konvertirati cijeli broj u odgovarajući broj s pokretnim zarezom

Razmotrite slijedeće izraze:

```
int m = 3;
int n = 4;
double x = m*n;
```

Kada Java bude izvršavala treći izraz. izračunat će m*n koristeći cjelobrojnu aritmetiku te rezultat privremeno spremiti kao 32-bitnu vrijednost odnosno cijeli broj 6. Zatim će tu vrijednost konvertirati u 64-bitni broj u pokretnom zarezu (6) te ga spremiti u varijablu x.

Što će biti rezultat izvršavanja slijedećeg izraza:

```
double x = m/n;
```

Odgovor je da će x biti postavljen na vrijednost 0. Razlog tome je da će za izračunavanje izraza m/n biti upotrijebljena cjelobrojna aritmetika, jer su oba operanda cjelobrojna

Pretpostavimo da ipak želimo koristiti normalno dijeljenje brojeva s pokretnim zarezom. Potrebno je Javu uvjeriti da vrijednosti m i n tretira kao brojeve s pokretnim zarezom. Možemo to učiniti na slijedeći način:

```
double x = ((double) m) / n;
```

Izraz `(double)` naziva se **cast operator**. Stavljanjem cast operatora ispred m kažemo Javi da konvertira vrijednost od `m` u ekvivalentnu `double` vrijednost. Kad smo to učinili operator se sada odnosi na jednu `double` vrijednost i jednu `int` vrijednost. U tom slučaju Java će koristiti dijeljenje za brojeve s pokretnim zarezom.

Kada želimo konvertirati broj u pokretnom zarezu u cijeli broj postoje dva načina.
Prvi je način da od broja s pokretnim zarezom uzmemo cijeli dio, dakle dio koji ostane kad oduzmemo decimalni dio.

Npr., broj u pokretnom zarezu 4.7 tada postaje cijeli broj 4. Način kako to postižemo je upotreba `int` cast operatora:

```
double x = 4.7;
int i = (int) x;
```

Drugi izraz će konvertirati vrijednost od `x` u cijeli broj 4 i nakon toga pohraniti ga u varijablu `i`. Da je x bio postavljen na –4.7, `i` bi primio vrijednost –4.

Drugi način konvertiranja brojeva u pokretnom zarezu u cijele brojeve je zaokruživanje na najbliži cijeli broj. Npr. najbliži cijeli broj za 4.7 je 5.

Npr. slijedeći izraz obavit će zaokruživanje pozitivnog broja x i pohraniti zaokruženu vrijednost u varijablu i :

```
int i = (int) (x + 0.5);
```

Npr., ako je `x` jednak 4.7, vrijednost od `x + 0.5` bit će 5.2, a vrijednost od `(int)(x + 0.5)` bit će 5.

(Upozorenje. Ne piši: `int i = (int)x + 0.5;`)

Kako bi bilo izvedeno zaokruživanje za negativne brojeve ?

U zaokruživanju broja u pokretnom zarezu(i pozitivnih i negativnih) na najbliži cijeli broj možete koristiti i metodu iz Math klase:

```
Math.round(x)
```

To je zgodno rješenje osim što `Math.round(x)` kao rezultat vraća vrijednost tipa `long`. Sve što je potrebno da bi se rezultat dodijelio varijabli tipa `int` je primjena `(int)` cast operatora :

```
int i = (int) (Math.round(x));
```

5. Čitanje ulaza korištenjem ConsoleReader objekta

Primjeri koji su dosada obrađivani koriste za svoj račun vrijednosti koje je programer upisao u sam kod programa. Ako u njima želimo neke druge vrijednosti moramo ih mijenjati u kodu programa.

Ljepša alternativa je da ponovo napišemo programe tako da njihove ulazne vrijednosti unosimo koristeći tipkovnicu tijekom izvršavanja programa. Npr. program koji konvertira valute mogao bi prvo ispisati poruku da korisnik upiše iznos u markama, zatim pročita ono što mu se upiše i na kraju ispiše rezultat.

U ovom dijelu napisati ćemo takav program.

Dosada smo u primjerima prikazivali informacije na ekranu koristeći `println i print` metode koje pripadaju objektu `System.out`.

Objekt `System.out` ima pristup ekranu i može proslijediti na njega bilo koje brojeve ili stringove koje mu damo kao parametre. Isto je točno kada čitamo s tipkovnice. U tom slučaju trebamo objekt koji je spojen na tipkovnicu i koji ima metode kojima možemo pristupiti znakovima koje kucamo na tipkovnici.

U Javi postoji takav objekt i naziva se `System.in`. Nažalost posjeduje veoma limitiran broj metoda. Posjeduje jednu metodu `read` koji će pročitati sve što korisnik upiše, ali ga je teško koristiti jer čita samo jedan karakter.

Nije teško konstruirati klasu s boljim setom metoda. U ovome tečaju koristit ćemo jednu klasu nazvanu `ConsoleReader.`

Kad je želimo koristiti u programu na početku programa napišemo:

```
ConsoleReader in = new ConsoleReader(System.in);
```

Primijetite da se radi o deklaraciji varijable. Ova deklaracija kreira novi objekt tipa klase `ConsoleReader` koji može pristupiti `System.in` objektu koji je spojen na tipkovnicu. U isto vrijeme kreira se varijabla `in` tipa `ConsoleReader` koja označava taj objekt.

Kad se izvrši ta naredba objekt koji nam je potreban odsad je prisutan u memoriji i njegove metode možemo pozivati koristeći njegov naziv `in` . (Možemo ga i drukčije nazvati)

`ConsoleReader` objekt posjeduje tri korisne metode:

`in.readInt()` vraća `int` vrijednost. To će biti cijeli broj koji korisnik utipka. Npr. izraz :

```
int broj = in.readInt();
```

će kreirati varijablu `broj` i pridružiti joj vrijednost koja se ukuca s tastature.

`in.readDouble()` vraća `double` vrijednost.

`in.readLine()` vraća string. Sastojat će se od znakova koje korisnik ukuca do kraja linije.

Ovo je nova verzija programa za preračun maraka u eure.

Započinje se kreiranjem `ConsoleReader` objekta, koji je označen sa `in`. Tada traži od korisnika da utipka vrijednosti maraka i pfeniga. Naredbe koje čitaju vrijednosti naznačene su podebljano (ne i u stvarnom kodu programa) .

PRIMJER 5

```
public class Mjenjacnica2

{  /* Ucitaj iznos u markama i preracunaj u eure */

   public static void main(String[] args)

   {

    ConsoleReader in =
    new ConsoleReader(System.in);
    System.out.println("Unesi iznose maraka i pfeniga");
    System.out.print("Iznos u markama =");
    int maraka = in.readInt();
    System.out.print("Iznos u pfenizima =");
    int pfeniga = in.readInt();

    pfeniga = 100*maraka + pfeniga;
    double eura = pfeniga * 0.0051129;
    System.out.print("Eura: ");
    System.out.print(eura);

   }
}
```

Na ekranu bi trebalo nakon svega pisati (ono što korisnik ukuca je ovdje naznačeno podebljano)

```
Unesi iznose maraka i pfeniga
Iznos u markama = 110
Iznos u pfenizima = 55
Eura: 56.5231095
```

`ConsoleReader` objekt nije robustan. Npr. ako unesete dva broja u jednu liniju ili ako unesete slova umjesto broja program će javiti run-time grešku.

Prije pokretanja bilo kojeg programa koji koristi klasu `ConsoleReader` potrebno je da istu imate u istome direktoriju kao i program u kojem je koristite.

Pretpostavimo da želite koristiti primjer 5. Prvo ćete primjer ukucati u datoteku s istim nazivom kao i klasa dakle `Mjenjacnica2.java`. U isti direktorij kopirajte i datoteku `ConsoleReader.java`. Tada prevedite i pokrenite `Mjenjacnica2` na uobičajen način. Kada Java prevodilac vidi da java koristi ConsoleReader klasu, automatski će je prevest.

---

6. Znakovni niz - string.

Nadalje ćemo se koristiti izrazom **string. String** je jedan od najčešće korištenih tipova podataka u Javi.

Niz znakova (**literal** string) možemo napisati kao niz znakova zatvorenih u dvostruke navodnike poput slijedećeg:

```
″Jedan dan″
```

Literal stringovi mogu biti proizvoljne duljine, te uključivati bilo koji znak. Čak možete imati i string bez karaktera. Takav string nazivamo **prazan** string i zadaje se kao "".

String može sadržavati **kontrolne znakove** (control characters). Ti znakovi znače stvari poput novog reda, tabulatora, ... Ako ispisujete string korištenjem `System.out.println` metode i ako ta metoda naiđe na kontrolni karakter npr. na newline kontrolni karakter ona će tada pomaknuti kursor na novi red.

Ako želite uključiti kontrolni karakter u literalni string, potrebno je korištenje \ karaktera nakon kojega slijedi slovo. '\' se naziva **escape** karakter. Koristi se i za uvođenje određenih karaktera u string poput " koji bi inače značio kraj stringa. Slijedi lista najznačajnijih kombinacija karaktera:

\n	novi red (newline)
\t	tabulator – pomak (tab)
\b	nazad (backspace)
\r	return – pomak na početak slijedeće linije
\f	line feed – pomak na slijedeću liniju bez odlaska na početak
\\	\ znak
\′	′ znak
\″	″ znak

Npr. izraz

```
System.out.print("Jedan\ndan");
```

napisat će 'Jedan' u jednu liniju te 'dan'u slijedeću liniju.

Ako želite pohraniti string da bismo ga koristili kasnije u programu koristit ćemo varijablu tipa **`String`.** Slijedi primjer izraza koji kreira novu varijablu tipa `String`, nazvanu `automobil` kojoj dodjeljuje vrijednost "BMW".

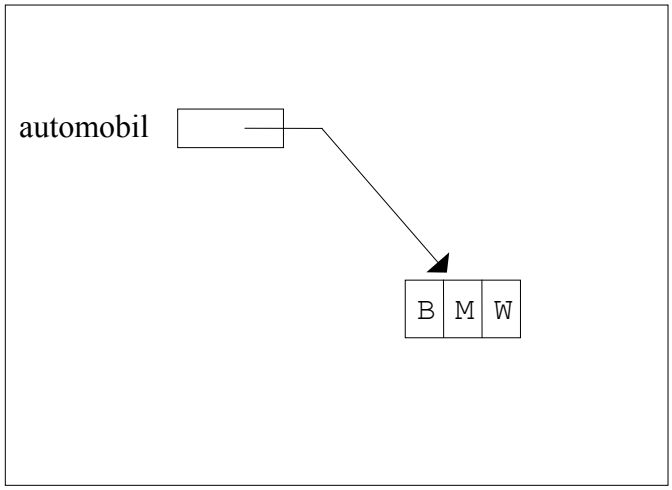
```
String automobil = "BMW";
```

Nije ispravno da će ovaj izraz pohraniti string `"BMW"` na memorijsku lokaciju varijable `automobil`. Što se stvarno događa uključuje jednu vrlo bitnu osobinu rada s memorijom Svaka lokacija u memoriji ima svoju adresu (**reference** value or **address**).
To je slično kao što svaka kuća u gradu ima svoju poštansku adresu

Kad se izvrši gornji izraz izvrše se slijedeći koraci:

(1) Kreira se varijabla nazvana `automobil`
(2) String `"BMW"` `pohrani se u` ***drugi dio memorije !***
(3) Referenca (adresa*)* gdje je string pohranjen dodijeli se varijabli `automobil`.

Rezultirajuće stanje u memoriji može se prikazati dijagramom na slici 1.



Slika 2.1. Strelica od varijable prema stringu reprezentira činjenicu da varijabla sadrži referencu na lokaciju gdje je string pohranjen.

Ovaj način tretiranja String varijable znači da je sama varijabla uvijek iste veličine – dovoljno velika da sadrži vrijednost reference (adrese)

Npr. ako kasnije izvršimo naredbu:

```
automobil = ″Alfa Romeo″;
```

varijabla `animal` će nakon toga opet sadržavati vrijednost reference, dakle istu količinu memorije. Razlika je u tome što će to biti referenca na neki drugi dio memorije koji sada sadrži drugi(duži) string ″Alfa Romeo″.

Postoji jedan vrlo koristan operator koji se može primijeniti na stringove. Naziva se operator spajanja (concatenation) i označava se znakom '+'.

Ako su st1 i st2 stringovi, onda st1+ st2 znači string koji se sastoji od svih karaktera st1 nakon kojih slijede karakteri iz stringa st2.
Npr. ako je varijabli `automobil` dodijeljena referenca na string "BMW" onda će izraz

```
″Novi ″ + automobil + ″ je skup.″
```

znači string:

```
″Novi BMW je skup.″
```

Izrazi poput ovoga mogu se koristiti bilo gdje u programu gdje je potreban neki složeni izraz.
Npr. slijedeći izraz :

```
System.out.println(″Novi ″ + automobil + ″ je skup.″);
```

će ispisati na ekranu ″Novi BMW je skup.″

Kako često trebamo ispisivati poruke koje sadrže i brojeve Java dopušta da se kombiniraju brojevi i stringovi korištenjem + operatora.
Npr. `cijena` je cjelobrojna vrijednost koja sadrži vrijednost 30000. Tada će izraz:

```
″Cijena novog ″ + automobil + ″ je ″ + cijena + ″ eura″.
```

značiti string:

```
″Cijena novog BMW je 30000 eura″.
```

Primijetite da je Java uzela vrijednost iz varijable cijena i pretvorila je u string sa znamenkama pogodnim prikazu cijelih brojeva.

I u primjerima 2 i 3 mogli smo koristiti sličan način :

Izraze:

```
System.out.print("Racun je: ");
System.out.print(total/100);
System.out.print(" kn i ");
System.out.print(total%100);
System.out.println(" lipa.");
```

Mogli smo zamijeniti s jednim izrazom:

```
System.out.println
   ("Racun je: "+ (total/100) + " kn i " +
   (total%100) + " lipa." );
```

Točnije, zagrade oko izraza `total/100` i `total%100` nisu potrebne jer Java primjeni pravilo da operatori / i % imaju veći prioritet od operatora +.

Slijedi program koji obavlja luckastu konverzaciju s korisnikom. Program čita korisnikovo ime i dodjeljuje ga varijabli `ime`. Također program starost korisnika i dodjeljuje ga varijabli `godina`. Nakon toga koristi te dvije vrijednosti da bi konstruirao par rečenica koje će ispisati na ekran.

Program započinje konstrukcijom `ConsoleReader` objekta da bi mogao čitati što korisnik utipka. U ovom slučaju taj objekt je označen nazivom `korisnik`. Metoda `korisnik.readLine()` koristi se za čitanje imena , a metoda `user.readInt()` za čitanje broja godina.

EXAMPLE 6

```
public class Prica

{  /* Pricaj s korisnikom. */

   public static void main(String[] args)

   {  ConsoleReader korisnik =
         new ConsoleReader(System.in);

      System.out.println
         ("Cao.  kako se zoveš?");
      String ime = korisnik.readLine();
      System.out.println
         ("Koliko imaš godina " + ime + "?");
      int godina = korisnik.readInt();
      System.out.print(godina + " su lijepe godine, ");
      System.out.println
         ("ali " + (godina+1) + " je bolje.");
      System.out.println
         ("Vidimo se kasnije " + ime + "!");
   }
}
```

Ovo bi se trebalo pojaviti na ekranu kad se program pokrene. (Korisnikov unos je podebljan.)

```
Cao.  kako se zoveš?
Ivica.
koliko imaš godina Ivica?
20
```

```
20 su lijepe godine , ali 21 je bolje.
Vidimo se kasnije Ivica!
```

Stringovi imaju mnogo korisnih metode koje su vezane za njih.
Jedna od vrlo korisnih se naziva `substring`. Java koristi konvenciju da su karakteri unutar stringa numerirani s 0, 1, 2, … s lijeva nadesno. Pretpostavimo da st označava string , a da su m i n cjelobrojne vrijednosti. Tada izraz :

```
st.substring(m,n)
```

daje novi string koji se sastoji od karaktera stringa `st`, počevši od pozicije m i završavajući s pozicijom n-1. (n je prva pozicija koja nije uključena !). Npr. ako varijabla `automobil` označava string `"Mercedes"`, onda izraz

```
automobil.substring(5,8)
```

će dati string `"des"`,kako je to dolje pokazano.

M	e	r	c	e	**d**	**e**	**s**
0	1	2	3	4	5	6	7	8

Navest ćemo još neke standardne metode koje možemo koristiti u radu sa stringovima. U svakom primjeru st označava string.

`st.length()` Daje broj znakova u stringu ("duljina" stringa st)
Npr. `"Mercedes".length()`
vraća vrijednost 8.

`st.toLowerCase()` Daje string sa svim velikim slovima pretvorenim u mala slova.
Npr. `"Mercedes".toLowerCase()`
vraća string `"mercedes"`

`st.UpperCase()` Daje string sa svim malim slovima pretvorenim u velika slova.
Npr. `"Mercedes".toUpperCase()`
vraća string `"MERCEDES"`

Ako ste izračunali broj i želite ga pretvoriti u odgovarajući string sve što je potrebno je kombinirati prazni string i broj koristeći + operator. Npr. ako varijabla `rezultat` sadrži vrijednost 142, onda izraz `"" + rezultat` će proizvesti string `"142"`.

To radi jer Java koristi pravilo koje kaže da ako kombinirate string i broj pomoću operatora +, broj će biti konvertiran u string, a zatim će oba stringa biti spojena. U ovome slučaju string nastao iz zbroja je spojen s praznim stringom.

U drugom smjeru nije tako jednostavno. Potrebno je koristiti neki od slijedećih metoda.

`Integer.parseInt(st)`	Ako string `st` predstavlja cjelobrojnu vrijednost ovo će dati odgovarajuću vrijednost
`Double.parseDouble(st)`	Ako string `st` predstavlja broj u pokretnom zarezu ovo će dati odgovarajuću vrijednost.

Na kraju ovog dijela još pokoje pravilo o kombiniranju brojeva i stringova korištenjem + operatora. Razmotrimo slijedeće izraze:

```
"Rezultat je " + 3 + 7

3 + 7 " je rezultat"
```

na prvi pogled izgleda kao da želimo reći istu stvar. Međutim ako ih ispišemo koristeći metodu `System.out.println` vidjet ćete da ćete dobiti sasvim nešto drugo.

Da biste predvidjeli što će se u ovakvim slučajevima potrebno je koristiti slijedeće pravila:

Ako Java treba odrediti vrijednost izraza x+y gdje su x i y bilo brojevi ili stringovi, tada će se dogoditi slijedeće:

(1) Ako su x i y oboje stringovi onda će ih spojiti.
(2) Ako je jedan string, a drugi broj onda će pretvoriti broj u string i spojiti ga s drugim stringom
(3) Ako su x i y oboje brojevi bit će zbrojeni.

Ako se računa izraz koji sadrži više + operatora izraz se računa slijeva nadesno !

Ponovo razmotrimo izraz:

```
"Rezultat je " + 3 + 7
```

Prvo što će Java obraditi je `"rezultat je " + 3`. Pretvorit će broj 3 u string "3", spojiti ga s prvim stringom i dati `"Rezultat je 3"`. Zatim će obraditi `"Rezultat je 3" + 7`, i konačno dati string:

```
"Rezultat je 37"
```

Sad razmotri:

```
3 + 7 " je rezultat"
```

Ovaj put će Java prvo zbrojiti 3 i 7 i dati broj 10. Zatim će obraditi `10 + " je rezultat "` i konačno dati:

```
"10 je rezultat "
```

---

7. Zadaci za poglavlje 2

Za ove zadatke kreirajte direktorij P2 unutar direktorija JProg.
Potrebno je unijeti i prevesti i izvršiti 4 zadatka.

1. Zadatak (Koristi naziv klase i datoteke Aritmetika)
   Napiši program koji traži od korisnika da ukuca dva cijela broja i zatim ispiši:

   - sumu
   - razliku
   - umnožak
   - prosjek (double !)
   - udaljenost (apsolutna vrijednost razlike)

   - Koristi metodu `Math.abs` za račun apsolutne vrijednosti

2. Zadatak (Koristi naziv klase i datoteke Mjenjacnica)

   Napiši program koji učitava iznos u kunama i preračunava ih u eure (kurs npr. 1 euro = 7.50 kn)

3. Zadatak. (Koristi naziv klase i datoteke Upoznavanje)
   Napiši program koji će ispisati prva dva reda kao što je prikazano, zatim učitati ime i nakon toga ispisati pozdrav. Podebljan je upis od strane korisnika . Ime može biti bilo koje.

```
Bok. Moje ime je Java.
Kako se ti zoveš ?
Ivica
Ivica ! Drago mi je što smo se upoznali.
```

4. Zadatak. (Koristi naziv klase i datoteke Tablica)
   Napiši program koji će nacrtati tablicu na ekranu..

```
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
|  |  |  |
+--+--+--+
```

   Učini to na način da kreiraš dvije string varijable za oba uzorka pa onda te varijable naizmjenično ispiši.