



Funded by the  
Erasmus+ Programme  
of the European Union



---

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

---



## KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

### Osnovni elementi programskog jezika java

Lekcija 02

PRIRUČNIK ZA STUDENTE

# KI103 - JAVA 1: OSNOVE PROGRAMIRANJA U JAVI

## Lekcija 02

### *OSNOVNI ELEMENTI PROGRAMSKOG JEZIKA JAVA*

- ✓ Osnovni elementi programskog jezika java
- ✓ Poglavlje 1: Elementi programskog jezika Java
- ✓ Poglavlje 2: Objekat i klasa kao tip podataka
- ✓ Poglavlje 3: Operatori u Javi
- ✓ Poglavlje 4: Rešeni zadaci za preuzimanje
- ✓ Poglavlje 5: Domaći zadaci
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

# ▼ Uvod

## UVOD

### *Upoznavanje elemenata programskog jezika Java.*

U predavanju ćemo se upoznati sa:

- **Osnovnim elementima i osobinama** programskog jezika Java
- **Tipovima podataka** i njihovom podelom
- Pojmovima **promenljivih, konstanti, komentara i naredbi**
- **Omotačkim klasama** i njihovim ugrađenim funkcijama
- Obradom **standardnog ulaza i izlaza** koristeći klasu **Scanner** i objekat **System.out**
- Pojmom i podelom **operatora**
- **Osnovnim aritmetičkim operatorima**

Neke pojmove koje moramo koristiti kao što su klase, objekti, metod i dr. biće objašnjeni na elementarnom nivou, radi razumevanja zašto se koriste u programima. Međutim, oni će biti detaljnije objašnjeni u kasnijim lekcijama, te ako vam nije sve jasno, ne paničiti. Biće vam sve to razjašnjeno kasnije.

## ▼ Poglavlje 1

# Elementi programskog jezika Java

## ELEMENTI JAVE

*Program čini niz deklarativnih i izvršnih naredbi, a njih čine niz leksičkih simbola. Leksički simboli ili tokeni predstavljaju nizove znakova.*

Programski jezik Java sadrži **skup znakova** (engl. **character set**) koji čine:

- mala i velika slova većine alfabeta sveta
- deset decimalnih cifara
- veći broj znakova interpunkcije

Java pravi razliku između malih i velikih slova. Iako Java podržava upotrebu slova većina alfabeta sveta u službenom delu jezika koriste se samo slova engleskog alfabeta.

**Leksički simboli** ili **tokeni** (engl. **tokens**) predstavljaju nizove znakova. U programskom jeziku Java dele se na:

- **Identifikatore**
- **Literale**
- **Ključne reči**
- **Operatore**
- **Separatore**
- **Komentar**

**Leksički simboli** mogu da se pišu spojeno ili međusobno razdvojeno proizvoljnim brojem **praznina**. U **praznine** (engl. **white spaces**) spadaju :

- **Znak za razmak**
- **Tabulacija**
- **Vertikalna tabulacija**
- **Prelazak u novi red**
- **Prelazak na novu stranu**

**Komentari** (engl. **comments**) predstavljaju proizvoljne tekstove koji su namenjeni čitaocu izvornog koda programa radi lakšeg razumevanja načina funkcionisanja programa. Komentari predstavljaju posebnu vrstu praznina i njih prevodilac zanemaruje.

**Naredbe** (engl. **statements**) su nizovi leksičkih simbola i dele se na deklarativne i izvršne naredbe.

- **Deklarativnim naredbama** se definišu se neki elementi programa (metode, klase itd.).
- **Izvršnim naredbama** izvode se elementarne obrade podataka.

**Program** predstavlja **niz naredbi** pomoću kojih se ostvaruju složene obrade podataka.

## ▼ 1.1 Identifikatori i ključne reči u Javi

### IDENTIFIKATOR

*Identifikator predstavlja ime (naziv) promenljivih, tj. memorijskih lokacija, datoteka, metoda, atributa, objekata i klasa.*

Ime u programu može da označava razne stvari: memorijsku lokaciju, datoteku na disku, potprogram (metod) ili novi tip podataka (klasa). Zbog svoje važnosti u programiranju, imena imaju poseban tehnički termin i zovu se **identifikatori**.

**Identifikatori** se ne mogu davati proizvoljno, već postoje striktna pravila kako se ona grade. Termini **ime** i **identifikator** su u ovom kontekstu često sinonimi, s tim da je termin **identifikator** više tehnički termin, dok se u praksi češće javlja termin **ime**.

U Javi se identifikator sastoji od niza znakova, jednog ili više njih. Ovi znakovi od kojih se sastoji identifikator moraju biti

- **slova**
- **cifre**
- **donja crta ( \_ )**
- **znak dolar ( \$ )**

Pored toga, identifikator mora početi:

- **slovom**
- **donjom crtom**
- **znakom \$**
- Treba naglasiti da razmaci nisu dozvoljeni u identifikatorima. Na primer: ispravan identifikator u Javi je **broj\_studenata**, ali **broj studenata** nije ispravan.
- Velika i mala slova se razlikuju u Javi tako da sledeći identifikatori smatraju idetičnim: j
- **Primer,**
- **primer,**
- **PRIMER i**
- **PriMeR**
- Identifikatori mogu da budu proizvoljno dugački. Prvi znak u identifikatoru ne sme da bude cifra. Upotrebu znaka **\$** treba izbegavati. On je rezervisan za identifikatore koje razni alati za generisanje programskog koda generišu automatski. Na primer, neki ispravni identifikatori u Javi su:
- **ulaz**
- **Slika**

- **SIRINA\_EKRANA**
- **ime\_i\_prezime**
- **prosečnaOcena**
- **Primer1**
- **Jedno\_VRLO\_dugačko\_ime**
- *nePreporučujeSe*

## ČESTE GREŠKE I PRAKSA

*Greška: Identifikator sadrži prazninu ili naziv klase počinje malim slovom*

Identifikator koji sadrži prazninu, na primer: **prosek Ocena**

### Preporučena praksa:

Pored ovih formalnih pravila koje propisuje sam jezik Java, postoje preporuke o tome kako treba birati dobra imena u programima. Pre svega, **imena trebaju biti smisljena** da bismo na osnovu njih mogli lako zaključiti šta označavaju.

U programu u kome računamo aritmetičku sredinu dva broja identifikator možemo nazvati **aritmetičkaSredina** ili **sredina**, jer to zaista i predstavlja, mada smo ga mogli zvati i **x** ili **bzvz**. Davanje smislenih imena znatno doprinosi boljoj čitljivosti programa.

Dodatna konvencija koje se pridržavaju Java programeri radi bolje čitljivosti programa je:

- **Imena klase** počinju **velikim slovom**
- **Imena promenljivih i metoda** počinju **malim slovom**
- **Imena konstanti** se sastoje od **svih velikih slova**

*:Ime klase koje počinje malim slovom, na primer: **mojaKlasa***

### Preporučena praksa:

Kada se ime sastoji od nekoliko reči, recimo **prosečnaOcena**, onda se svaka reč od druge piše sa početnim velikim slovom (a i prva ako se radi o klasi). Java programeri uglavnom ne koriste donju crtu u imenima, osim u nekim izuzetnim slučajevima i to samo na prvom mestu.

Što se tiče upotrebe imena u programu radi označavanja programskih elemenata, često se koriste složena imena koja se sastoje od nekoliko običnih imena razdvojenih tačkama. Složena imena se pišu u notaciji koja naziva **tačka-notacija**. Primer ovoga možemo videti kod ispisivanja neke vrednosti na standardni izlaz:

```
System.out.println()
```

## SLOŽENA IMENA

*Uobičajeno je da se u identifikatorima koji su sastavljeni od više reči početna slova pojedinih reči pišu velikim slovima, a ostala malim slovima*

:Složena imena su potrebna zato što u Javi neke stvari mogu da obuhvataju druge stvari. Složeno ime je onda kao putokaz do neke stvari kroz jedan ili više nivoa obuhvatanja. Tako **imeSystem.out.println()** ukazuje da nešto pod nazivom System sadrži nešto pod nazivom out koje sadrži nešto pod nazivom println.

Uobičajeno je da se u identifikatorima koji su sastavljeni od više reči početna slova pojedinih reči pišu velikim slovima, a ostala malim slovima. Na primer, neki neispravni identifikatori u Javi su:

- **2Godina** Prvi znak ne sme da bude cifra
- **A, B** Znakovi interpunkcije nisu dozvoljeni
- **x-y** Operatori nisu dozvoljeni
- **int** Rezervisane reči nisu dozvoljene

## KLJUČNE REČI U JAVI

*Izvesne reči su rezervisane za specijalnu namenu u Javi tako da se one ne mogu koristiti za identifikatore koja daje programer*

Izvesne reči su rezervisane za specijalnu namenu u Javi tako da se one ne mogu koristiti za identifikatore koja daje programer. Ove reči nazivamo **službenim, rezervisanim** ili **ključnim rečima**(engl.**keywords**) jezika Java. U ovu reči spadaju: **import, class, public, static, if, else, while** - sveukupno oko pedesetak takvih reči.

<b>abstract</b>	<b>assert</b>	<b>boolean</b>	<b>break</b>	<b>byte</b>
<b>case</b>	<b>catch</b>	<b>char</b>	<b>class</b>	<b>continue</b>
<b>const</b>	<b>default</b>	<b>do</b>	<b>double</b>	<b>else</b>
<b>enum</b>	<b>extends</b>	<b>false</b>	<b>final</b>	<b>finally</b>
<b>float</b>	<b>for</b>	<b>goto</b>	<b>if</b>	<b>implements</b>
<b>import</b>	<b>instanceof</b>	<b>int</b>	<b>interface</b>	<b>long</b>
<b>native</b>	<b>new</b>	<b>null</b>	<b>package</b>	<b>private</b>
<b>protected</b>	<b>public</b>	<b>return</b>	<b>short</b>	<b>static</b>
<b>strictfp</b>	<b>super</b>	<b>switch</b>	<b>synchronized</b>	<b>this</b>
<b>throw</b>	<b>throws</b>	<b>transient</b>	<b>true</b>	<b>try</b>
<b>void</b>	<b>volatile</b>	<b>while</b>		

Slika 1.1.1 Ključne reči programskog jezika Java

Ključne reči **const** i **goto** su uvrštene u ključne reči, mada se ne koriste u jeziku Java, jer su nasleđene iz programskih jezika C i C++.

**Važna napomena:**

Spisak ključnih reči ne treba učiti napamet za sada, već ćemo se sa njima upoznavati u toku lekcija.



## PRIMER 1

### *Primena identifikatora u Java klasama*

Zadatak:

Kreirati Java klasu sa sledećim elementima:

1. Naziv klase počinje velikim slovom;
2. Klasa sadrži atribut ime i prezime specificirane malim slovom;
3. Klasa sadrži i jedno konstantno polje kojim je pokazano koliko ukupno polja klasa ima (ime, prezime i konstantno polje);
4. Klasa sadrži metodu za prikazivanje vrednosti polja čiji naziv počinje malim slovom;
5. Koristite komentare za svaku liniju koda.
6. Pokrenite kreirani program.

Rešenje:

Kreira se NetBeans projekat pod nazivom *PisanjeIdentifikatora.java* (klasa je definisana velikim slovom). Klasa pripada paketu *lo2*. Kreiraju se polja ime, prezime i BROJ\_POLJA\_U\_KLASI. Obratite pažnju na prva dva polja. Radi se o promenljivim u Javi i pišu se malim slovima (prema pravilima lepog pisanja programa). Poslednje polje je konstanta i po dogovoru naziv konstante čine sva velika polja. Konačno, kreirana je metoda pisi() čiji je zadatak da prikaže vrednosti kreiranih polja.

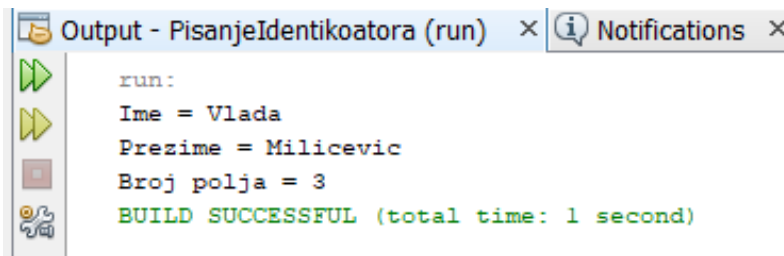
```
package lo2;

/**
 *
 * @author Vladimir Milicevic
 */
public class PisanjeIdentifikatora {
    /*imena promenljivih počinju malim slovom*/
    public String ime="Vlada";
    public String prezime="Milicevic";

    /*konstante se pišu velikim slovima*/
    public final int BROJ_POLJA_U_KLASI=3;

    //naziv metode se piše malim slovom
    public void pisi(){
        System.out.println("Ime = " + ime);
        System.out.println("Prezime = " + prezime);
        System.out.println("Broj polja = " + BROJ_POLJA_U_KLASI);
    }
    public static void main(String[] args) {
        // kreira se objekat klase. Počinje malim slovom
        PisanjeIdentifikatora pisanjeiden = new PisanjeIdentifikatora();
        /*kreirani objekat poziva metodu za pisanje*/
        pisanjeiden.pisi();
    }
}
```

```
}
```



Slika 1.1.2 Rezultat izvršavanja kreiranog programa

## ZADATAK 1

### *Vežbajte pisanje identifikatora u Java klasama*

Zadatak:

Kreirati Java klasu sa sledećim elementima:

1. Naziv klase počinje velikim slovom;
2. Klasa sadrži polja ime i prezime specificirane malim slovom;
3. Klasa sadrži i polje školska godina (piše se kao jedna reč - pogledati sekciju "Složena imena")
4. Klasa sadrži i jedno konstantno polje kojim je pokazano koliko ukupno polja klasa ima (ime, prezime, školska godina i konstantno polje);
5. Klasa sadrži metodu za prikazivanje vrednosti polja čiji naziv počinje malim slovom;
6. Koristite komentare za svaku liniju koda.
7. Pokrenite kreirani program.

## ✓ 1.2 Tipovi podataka

### ŠTA JE TIP PODATAKA?

*Tip podataka definiše veličinu i organizaciju podataka, opseg mogućih vrednosti i skup operacija koje se mogu obaviti nad tim vrednostima.*

Računarski sistemi rade sa binarnim podacima (0 i 1) bez ikakve pretpostavke šta ti nizovi bitova znače. Podaci su predmet obrade u programima. Svaki podatak ima određena svojstva koja čine **tip podatka**.

**Najmanja količina podataka** koja može da se adresira je **jedan bajt** (1 bajt = **8 bitova**) koji može da ima **256 (2<sup>8</sup>)** različitih vrednosti.

**Tip podataka** definiše **veličinu i organizaciju podataka, opseg mogućih vrednosti i skup operacija** koje se mogu obaviti nad tim vrednostima. Dakle, podatak može biti broj studenata, kao što podatak može biti i velika slika. Logično je da će ova dva podatka pripadati različitim tipovima podataka.

Podaci kojima Java programi mogu da manipulišu se dele se u dve glavne grupe:

**Primitivni (prosti) tipovi podataka** su predstavljaju tipovi podataka koji su unapred "ugrađeni" u jezik i postoje kod svih programskih jezika. Ne mogu da se rastave na manje elemente koji bi mogli nezavisno da se obrađuju, zato se kaže da oni nemaju strukturu.

**Klasni tipovi podataka** (objekti) su novi tipovi podataka koje programer sam definiše ili koristi već definisane.

U programskom jeziku Java **primitivnim tipovima podataka** obuhvaćeni su osnovni podaci koji nisu objekti. Tu spadaju:

- **Celi brojevi**
- **Realni brojevi**
- **Logičke vrednosti**
- **Znakovi alfabeta**

**Podaci klasnih (složenih) tipova** sastoje se od nekoliko elemenata koji mogu da se nezavisno obrađuju. Elementi objekata, tj. klasnih tipova, mogu da budu prosti, ali i sami mogu da budu složeni. Na taj način je broj različitih klasnih tipova podataka, koji mogu da se izvode, polazeći od prostih tipova podataka, neograničen. U jeziku Java, klasni podaci su **nizovi i klase**. Recimo, ako želite da radite sa slikama, datotekama ili da kreirate video igru, morate koristiti klasni tip podataka.

## ▼ 1.3 Primitivni tipovi podataka

### ŠTA JE PRIMITIVNI TIP PODATAKA?

*Primitivni tip podataka je ugrađen u programski jezik i predstavlja prostu vrednost koja se ne može deliti na manje delove.*

Postoje neki tipovi podataka koji su tako fundamentalni da su načini na koji se oni predstavljaju – ugrađeni u programske jezike. Takvi tipovi se nazivaju **primitivnim (ili prostim) tipovima podataka**. U frazi primitivni tipovi podataka reč primitivni znači "osnovna komponenta koja se koristi za kreiranje drugih, većih delova".

Osnovna osobina primitivnih tipova podataka je da se njima predstavljaju **proste vrednosti(brojevi, znakovi i logičke vrednosti)** koje se ne mogu deliti na manje delove. Pored toga, za svaki tip podatka su definisane i različite **operacije** koje se mogu primenjivati nad vrednostima određenog tipa - ako radimo sa znakovima, nelogično je deliti znakove.

Na primer, za vrednosti tipova koji predstavljaju **cele i realne brojeve** definisane su **operacije** sabiranja, oduzimanja, množenja i deljenja. Slično tome, za **logičke vrednosti** definisane su **logičke operacije** (AND, OR, NOT).

Tipovi podataka **byte**, **short**, **int** i **long** služe za rad sa celim brojevima (brojevima bez decimala kao što su 17, -1234 i 0) i razlikuju se po veličini memorijske lokacije koja se koristi za binarni zapis celih brojeva.

## PODRŽANI PRIMITIVNI TIPOVI U JAVI

*Java podržava sledeće primitivne tipove podataka: byte, int, float, double, char i boolean.*

Tip podataka **byte** zauzima, kao što sam reč kaže - 1 bajt (8 bitova) u memoriji. To znači da je u promenljivu tipa byte moguće smestiti 28 mogućih vrednosti, tj 256 različitih vrednosti. Radi lakšeg pamćenja bitno je napomenuti da svaki sledeći tip podataka iz ove grupe zauzima duplo veći prostor.

Tip podataka **int** se najčešće koriste u praksi.

Tipovi podataka **float** i **double** služe za predstavljanje realnih brojeva (brojeva sa decimalama kao što su 1.69, -23.678 i 5.0). Ova dva tipa se razlikuju po veličini memorijske lokacije koja se koristi za binarni zapis realnih brojeva, ali i po tačnosti tog zapisa (tj. maksimalnom broju cifara realnih brojeva).

Tip podataka **double** zauzima duplo više prostora u memoriji u odnosu na **float** tip podataka i češće se koristi. Razlog je jednostavan - današnji računari ne prave veliku razliku između 4 bajta i 8 bajtova, a koristeći tip **double** možemo biti sigurni da će preciznost biti veća, samim tip što zauzima više bajtova u memoriji.

Tip podataka **char** sadrži pojedinačne alfabetske znakove kao što su mala i velika slova (A,a,B,b,...), cifre (0,1,...), znakovi interpunkcije (\*,?,...) i neke specijalne znakovi (za razmak, za novi red, za tabulator,...).

Znakovi tipa **char** predstavljaju se u binarnom obliku prema standardu **Unicode**, koji je izabran zbog mogućnosti predstavljanje pisama svih jezika na svetu, odnosno zbog internacionalizacije koja je uvek bila među osnovnim ciljevima Jave. Ovaj tip podataka zauzima **2 bajta** u memoriji, tj. u promenljivu tipa char se može smestiti 216 različitih znakova, što je dovoljno za sve specijalne karaktere i znakove iz većine svetskih jezika.

Tip podataka **boolean** služi za predstavljanje samo dve logičke vrednosti: tačno (**true**) i netačno (**false**). Logičke vrednosti u programima se najčešće dobijaju kao rezultat izračunavanja relacionih operacija. Zauzima jedan bajt u memoriji (iako je logično da zauzima samo jedan bit, a ne bajt), zato što je to najmanja vrednost koja se može adresirati.

## OPSEG PRIMITIVNIH TIPOVA

*Svi primitivni tipovi podataka imaju tačno određenu veličinu memorijske lokacije u bajtovima za zapisivanje njihovih vrednosti.*

Svi primitivni tipovi podataka imaju **tačno određenu veličinu memorijske lokacije** u bajtovima za zapisivanje njihovih vrednosti. To znači da primitivni tipovi podataka imaju tačno definisan opseg vrednosti koje im pripadaju. Na primer, vrednosti koje pripadaju tipu **short** su svi celi brojevi od -32768 do 32767. Skup ovih vrednosti predstavlja **opseg vrednosti** koje mogu da se dodele podatku određenog tipa.

Ako napravimo promenljivu tipa **int**, ona će zauzimati isti memorijski prostor bez obzira da li je njena trenutna vrednost 0 ili -2000000.

Kada je ta vrednost manja od minimalne imamo slučaj **podkoračenja** vrednosti (**underflow**). Kada je vrednost veća od maksimalne onda imamo slučaj **prekoračenja** vrednosti (**overflow**).

Slika 1 prikazuje spisak svih primitivnih tipova sa njihovim osnovnim karakteristikama:

Tip	Opis	Opseg vrednosti
byte	Celi broj, 8-bitni	$-2^7$ do $2^7-1$ (-128 do 127)
short	Celi broj, 16-bitni	$-2^{15}$ do $2^{15}-1$ (-32,768 do +32,767)
int	Celi broj, 32-bitni	$-2^{31}$ do $2^{31}-1$ (-2147483648 do 2147483647)
long	Celi broj, 64-bitni	$-2^{63}$ do $2^{63}-1$ (-10E18 do +10E18)
float	Broj sa pokretnim zarezom jednostruke tačnosti, 32-bitni	-3.4E+38 do +3.4E+38
double	Broj sa pokretnim zarezom dvostruke tačnosti, 64-bitni	-1.7E+308 do 1.7E+308
char	Znak, 16-bitni, može se tretirati kao neoznačeni 16-bitni celi broj	'\u0000' do '\uFFFF'
boolean	Logička vrednost	true ili false

Slika 1.2.1 Primitivni tipovi u Javi

## KLASNI TIPOVI PODATAKA

*Svi tipovi podataka koji nisu primitivni tipovi, su klasni tipovi podataka, jer se definišu određenom klasom.*

Velika i mala slova kod imena ovih osam tipova su vrlo bitna. To znači da je **byte** ime primitivnog tipa podatka, ali **Byte** nije. Pošto smo rekli da imena klasa počinju velikim slovom, **Byte** će biti klasa, a ne primitivni tip!

Svi ostali tipovi podataka u Javi pripadaju kategoriji **klasnih tipova podataka** kojima se predstavljaju **klase objekata**. Objekti su "složeni" podaci po tome što se grade od sastavnih delova kojima se može nezavisno manipulirati. O klasama i objektima biće mnogo više reči u narednim predavanjima.

## PRIMER 2

### *Vežbanje rada sa tipovima podataka*

Kreirajte Java program, pripada paketu `lo2`, po sledećim zahtevima:

1. Klasa sadrži jedno celobrojno polje;
2. Klasa sadrži jedno polje realnog tipa podataka;
3. Klasa sadrži polje klasnog tipa u kojem se čuva celobrojna vrednost;

4. Dodeliti vrednosti promenljivim prostog tipa;
5. Kreirati objekat klase Integer i dodeliti mu vrednost;
6. Prikazati postojeće podatke kao rezultate.

Rešenje:

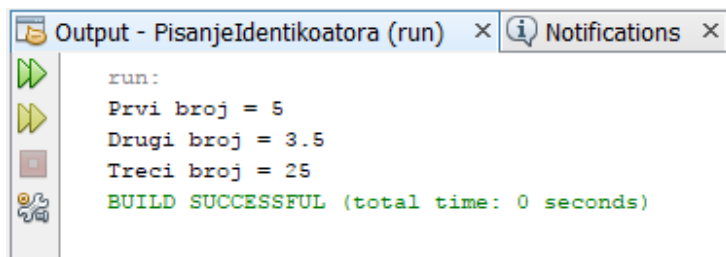
Kreirana je klasa `PrimenaTipovaPodataka.java` koja pripada paketu `lo2`. Klasa poseduje tri polja: dva prostog tipa (`int` i `double`) i jedno klasnog tipa `Integer`. Na jednostavan način (linije 9 i 10) je dodeljena vrednost promenljivim prostog tipa,

U metosi `pisi()` (o metodama će biti detaljno govora kasnije) primenom metode, koja se zove konstruktor, konačno je kreiran objekat (promenljiva klasnog tipa) klase `Integer` koja čuva celobrojnu vrednost 25 (linija koda 19).

```
package lo2;

/**
 *
 * @author Vladimir Milicevic
 */
public class PrimenaTipovaPodataka {
    /*prosti tip podataka*/
    public int broj1 = 5;
    public double broj2 = 3.5;
    /* klasa kao tip podataka*/
    public Integer broj3;

    //naziv metode se piše malim slovom
    public void pisi(){
        //kreiranje objekta i dodela vrednosti
        broj3 = new Integer(25);
        System.out.println("Prvi broj = " + broj1);
        System.out.println("Drugi broj = " + broj2);
        System.out.println("Treci broj = " + broj3);
    }
    public static void main(String[] args) {
        // kreira se objekat klase. Počine malim slovom
        PrimenaTipovaPodataka primenaTipova = new PrimenaTipovaPodataka();
        /*kreirani objekat poziva metodu za pisanje*/
        primenaTipova.pisi();
    }
}
```



```
run:
Prvi broj = 5
Drugi broj = 3.5
Treci broj = 25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Slika 1.2.2 Izvršavanje kreiranog programa

## ZADATAK 2

### *Samostalno vežbanje rada sa tipovima podataka*

Iskoristite prethodni primer i dopunite ga sledećim funkcionalnostima:

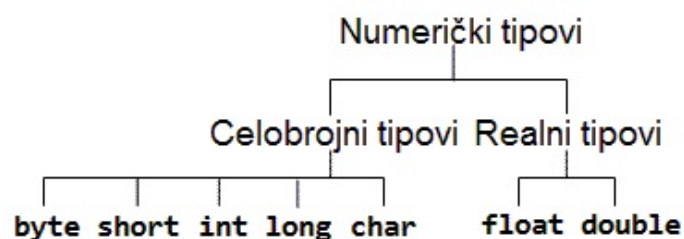
1. Dodajte još jednu konstantu (kao u Primeru 1) na primer  $\pi = 3.14$ ;
2. U metodi pisi() omogućite da na ekranu bude prikazana i ova vrednost;
3. U metodi pisi() omogućite da na ekranu bude prikazan zbir svih unetih vrednosti.

## ▼ 1.4 Numerički tipovi podataka

### CELI BROJEVI

*Java poznaje četiri vrste celih brojeva: byte, short, int i long. Oni se razlikuju prema opsegu mogućih vrednosti i veličini memorijskog prostora koji zauzimaju.*

**Numerički tipovi podataka** predstavljaju proste tipove za koje su definisane aritmetičke operacije. Među njima posebno se razlikuju **celobrojni** i **realni tipovi**.



Slika 1.3.1 Numerički tipovi

Programski jezik Java poznaje četiri vrste **celih brojeva**:

- **byte**

- **short**
- **int**
- **long**

Oni se razlikuju prema **opsegu mogućih vrednosti** i **veličini memorijskog prostora** koji zauzimaju.

## PRIMERI ZA BYTE, SHORT, INT I LONG

*byte sadrži 8 bitova, short sadrži jedan byte (16 bitova), int sadrži dva bajta (16 bitova) a long sadrži četiri bajta (32 bita).*

Svaki primitivni tip koristi određeni, nepromenljivi broj bitova. To znači da će za isti tip podataka, biti upotrebljen isti broj bitova, bez obzira na to koji broj predstavljate. Na primer: sve vrednosti tipa **short** će koristiti 16 bitova, odnosno vrednost trideset hiljada će biti predstavljena preko 16 bitova. Sve vrednosti tipa **long** će biti predstavljene sa 64 bita. Vrednost nula (tipa **long**) će biti predstavljena sa 64 bita; vrednost tri milijarde, takođe.

Vrednosti koje su veće, moraju biti predstavljene sa više bitova. Ovo je slično zapisivanju brojeva na papiru. Za zapisivanje većih brojeva, potrebno vam je više cifara. Ako je za predstavljanje neke vrednosti potrebno više bitova od broja koji se koristi za taj tip podatka, onda se ta vrednost ne može predstaviti preko tog tipa podatka.

Veći opseg kod numeričke vrednosti traži više bitova. Različite veličine kod tipova za predstavljanje celih brojeva, omogućavaju da izaberete pravi tip podatka sa kojim ćete raditi. Obično birate tip podatka čiji je opseg mnogo veći od brojeva sa kojima očekujete da ćete realno raditi. Ako program sadrži samo nekoliko promenljivih, onda će on biti podjednako brz i neće zauzimati mnogo memorije, bez obzira na tip koji koristite za svoje promenljive.

Zašto biste onda, uopšte koristili tipove podataka manje veličine? Odgovor je u činjenici da većina programa koji se stvarno koriste radi sa ogromnim količinama podataka (milijarde stavki), tako da upotreba manjih tipova može da dovede do značajne uštede prostora i vremena. I pored toga, programeri ne rade u skladu sa ovim.

Sledeća slika prikazuje osobine celobrojnih tipova podataka:

Oznaka tipa	Dužina (Bajt)	Opseg vrednosti	
byte	1	-128	127
short	2	-32768	32767
int	4	-2147483648	2147483647
long	8	-92233720368547758008	92233720368547758007

Slika 1.3.2 Osobine celobrojnih podataka

Što se tiče ove slike, naravno, ne trebate učiti vrednosti napamet, ali morate znati opsege orijentaciono, recimo: trebate znati da 13-cifreni broj ne može da se smesti u tip int.

Kada u programu želite da napišete neki broj, ne morate znati kako da ga predstavite preko bitova. Broj možete uneti direktno u program, na isti način kao što biste to uradili i na pisaćoj mašini. Ovakav broj se naziva literalom.



Literali za celobrojne tipove u jeziku Java mogu da se pišu u decimalnom, oktalnom i heksadecimalnom brojevnom sistemu.

## DECIMALNI LITERAL

*Decimalni literali koriste za osnovu brojevnog sistema broj 10 i sastoje se od niza decimalnih cifara, ispred kojeg može da se nalazi predznak + ili -.*

**Decimalni literali** sastoje se od niza decimalnih cifara, ispred kojeg može da se nalazi predznak + ili -. Prva cifra u nizu ne može da bude nula. Ukoliko nedostaje predznak, broj se smatra pozitivnim.

Na primer:

```
+125  
-32  
16  
0  
-3200000
```

### Važna napomena:

U praksi se najčešće koriste decimalni literali, ali ćemo pomenuti i oktalne i heksadecimalne literale.

Literal čija je prva cifra nula (0) pripada oktalnom brojevnom sistemu, tj. brojevnom sistemu s osnovom 8. Dozvoljene cifre su 0, 1, ..., 7.

Na primer:

```
0561
```

## HEKSADECIMALNI LITERAL

*Heksadecimalni literali koriste za osnovu brojevnog sistema broj 16.*

**Heksadecimalni literali** (osnova brojevnog sistema 16) počinju znakovima **0x** ili **0X**.

Šesnaest heksadecimalnih cifara obeležavaju se ciframa 0, 1, ..., 9 i slovima **a, b, c, d, e i f** (pri čemu je dozvoljena upotreba velikih i malih slova ravnopravno). Slova predstavljaju, redom, vrednosti 10, 11, ..., 15. Na primer:

```
0xa8fd5  
0X1B4F  
0xB0afD
```

Dodavanjem sufiksa **l** ili **L** iza niza cifara u sva tri brojeva sistema dobija se literal tipa **long**. Preporučuje se upotreba slova **L** zbog velike sličnosti slova **l** i cifre 1. Na primer:

```
3691l  
-8795L  
012345671l  
0xADCFL
```

## REALNI (DECIMALNI) BROJEVI

*Literali za realne tipove u jeziku Java mogu da se pišu u decimalnom i heksadecimalnom brojevnom sistemu.*

Programski jezik Java poznaje dve vrste realnih (decimalnih) brojeva prema IEEE standardu za realne (decimalne) brojeve:

- Tip **float** označava realan podatak **jednostruke tačnosti** (engl. **single precision**).
- Tip **double** označava realan podatak **dvostruke tačnosti** (engl. **double precision**).

Oznaka tipa	Dužina (B)	Opseg vrednosti		Značajne cifre
float	4	$1.17 * 10^{-38}$	$3.4 * 10^{38}$	6 - 7
double	8	$2.22 * 10^{-308}$	$1.79 * 10^{308}$	15 - 16

Slika 1.3.3 Osobine realnih podataka

Literali za realne tipove u jeziku Java mogu da se pišu u decimalnom i heksadecimalnom brojevnom sistemu. Za brojeve sa vrlo velikim i vrlo malim apsolutnim vrednostima koristi se eksponencijalni zapis.

Kao i kod celih brojeva - decimalni zapis se najčešće koristi, ali se u slučaju pojave ekstremno velikih ili ekstremno malih vrednosti može koristiti i eksponencijalni zapis.

Eksponencijalni zapis za decimalni brojevni sistem je oblika **mEk**, gde se m naziva mantisa, a k eksponent. Mantisa je decimalan broj koji može da ima predznak, a ceo i decimalni deo su razdvojeni decimalnom tačkom (.).

Eksponent je ceo broj sa eventualnim predznakom. Vrednost predstavljenog broja je **m \* 10<sup>k</sup>**. Pojedini delovi potpunog eksponencijalnog oblika mogu da nedostaju. Obavezno mora da postoji tačka (.), e ili E, da bi realni literali mogli da se razlikuju od celobrojnih literala. Takođe, mantisa mora da sadrži bar jednu cifru. Na primer:

```
-53.94E+2 (-53.94 * 102 = -5394)  
2e6 (2*106)  
+456.33 (456.33)  
6. (6.0)  
.12 (0.12)
```

Za realne literale podrazumeva se da su dvostruke tačnosti (tip **double**). Realni literali jednostruke tačnosti (tip float) obeležavaju se sufiksom f ili F. Na primer:

```
-654.856E+2f  
1e5F
```

```
+123.45F
6.f
.12F
```

Opšti oblik heksadecimalnog zapisa realnih literala je **0XmPk**, s tim da ravnopravno mogu koristiti mala i velika slova. Mantisa *m* se sastoji od heksadecimalnih cifara i vrednost se računa u tom brojevnom sistemu. EkspONENT *k* je decimalni celobrojni literal. Vrednost literala je **m \* 2<sup>k</sup>**. Pojedini delovi potpunog eksponencijalnog oblika mogu da nedostaju. Obavezno mora da postoji 0x ili 0X i p ili P, da bi mogao da se razlikuje od decimalnog zapisa. Na primer:

```
0xA0.Cp+2 (160,75-22 = 643,0)
0x1P5 (25 = 32)
0x.1p2 (0,0675-22 = 0,25)
```

## PRIMER 3

### *Proračun obima i površine kvadrata*

*Napisati program koji računa obim i površinu kvadrata na osnovu unete celobrojne stranice sa standardnog ulaza. Program treba da bude deo NetBeans projekta pod nazivom I02. Naziv klase treba da bude Zadatak3. Za učitavanje podataka treba koristiti objekat klase Scanner, a za ispis podataka objekat System.out.*

**Opšte napomene:** Potrebno je kreirati NetBeans projekat pod nazivom I02. Paket predstavlja folder koji može da sadrži više fajlova. U ovom paketu ćemo smestiti sve primere iz vežbe za drugu nedelju.

**Objašnjenje:** Za učitavanje podataka od korisnika sa standardnog ulaza neophodno je kreiranje novog objekta *ulaz* klase Scanner prosleđivanjem objekta System.in koji predstavlja standardni ulaz, na sledeći način:

```
Scanner ulaz = new Scanner(System.in);
```

Klasa Scanner omogućava unos podataka u Java aplikaciju metodama koje sadrži (nextInt(), nextDouble(), nextLine()...). S obzirom da je klasa Scanner deo paketa java.util, da bi se koristila, ona mora da se uveze na sledeći način:

```
import java.util.Scanner;
```

Korišćenjem metode println() objekta System.out korisniku prikazujemo poruku da je potrebno da unese stranicu kvadrata. Nakon toga, kreiramo promenljivu *stranica* koja čuva veličinu stranice kvadrata koju je korisnik uneo i dodeljujemo joj vrednost koja je uneta sa konzole pozivajući objekat *ulaz* klase Scanner i njegovu metodu nextInt() koja vraća ceo broj.

Promenljive *obim* i *površina*, koje su takođe celobrojnog tipa, treba da sačuvaju vrednosti na osnovu odgovarajućih formula po kojima se računaju. Dobijene rezultate štampano korisniku na standardnom izlazu koristeći metodu println() objekta System.out.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package l02;

import java.util.Scanner;

/**
 *
 * @author Jovana
 */
public class Primer3 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Primer3();
    }

    public Primer3() {
        Scanner ulaz = new Scanner(System.in);
        System.out.println("Unesite veličinu stranice kvadrata: ");
        int stranica = ulaz.nextInt();
        int obim = stranica * 4;
        int površina = stranica * stranica;

        /**
         * Za ispis podataka korisniku možemo koristiti metode:
         * print(), println() i printf() objekta System.out.
         * print - štampa ostavljajući kursor na narednom karakteru nakon teksta
         koji se štampa,
         * println - štampa i pomera kursor na početak sledećeg reda,
         * printf - služi za formatiranje podataka.
         */
        System.out.println("Obim kvadrata je: " + obim + ", a površina kvadrata je:
" + površina);
    }
}
```

## ZADATAK 3

### *Dodatno vežbanje prostih tipova podataka*

Uslovi zadatka odgovaraju prethodno definisanom zadatku "Zadatak1". Koristeći Primer 3, deo koji se odnosi na unos vrednosti sa tastature, nadogradite Zadatak1 tako da se vrednosti, koje odgovaraju promenljivim prostog tipa, unose sa tastature.

## PRIMITIVNI TIPOVI PODATAKA (VIDEO)

*Video prikazuje sve primitivne tipove podataka u Javi*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 1.5 Logički tipovi podataka

## ŠTA JE BOOLEAN?

*.Promenljiva tipa boolean može imati samo jednu od dve logičke vrednosti: tačno (true) ili netačno (false) i zauzima jedan bajt*

U Javi postoji i primitivan tip podatka **boolean**; koristi se da predstavi vrednost **tačno** ili **netačno**. Promenljiva tipa **boolean** može imati samo jednu od dve logičke vrednosti: tačno (**true**) ili netačno (**false**) i zauzima jedan bajt.

U programu pisanom u Javi reči **true** i reč **false** uvek označavaju vrednosti tipa **boolean**.

Nad podacima logičkog tipa **boolean** se primenjuju **logičke operacije** (**NOT**, **AND**, **OR**, **XOR**). O ovim operacijama ćemo više govoriti kasnije, a ako niste upoznati sa ovim terminima možete pogledati sledeći link:

[http://sr.wikipedia.org/  
sr/%D0%91%D1%83%D0%BB%D0%BE%D0%B2%D0%B0\\_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%](http://sr.wikipedia.org/sr/%D0%91%D1%83%D0%BB%D0%BE%D0%B2%D0%B0_%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%)

Logičke vrednosti u programima se najčešće dobijaju kao rezultat izračunavanja logičkih i relacionih operacija, u zavisnosti da li je rezultat tačan ili netačan. Koriste se i kao uslovi kod naredbi ***if***, ***else if***, ***for***, ***while*** i ***dowhile***.

## PRIMER 4

## Vežbanje korišćenja logičkih tipova podataka

Zadatok:

1. Kreira se klasa `PrimenaLogickihTipovaPodataka.java` koja pripada paketu `lo2`.
2. Klasa sadrži dva logička polja `a` i `b`;
3. Pridružiti proizvoljne logičke vrednosti navedenim poljima;
4. Prikazati vrednosti izraza: `ne a`, `a i b`, `a ili b`.
5. Pokrenuti kreirani program

Rešenje:

U nastavku je cilj dodatnog vežbanja upotrebe komentara. Rešenje zadatka je opisano kroz komentare u kodu.

```
package l02;

/**
 *
 * @author Vladimir Milicevic
 */
public class PrimenaLogickihTipovaPodataka {
    /*prosti logički tip podataka*/
    public boolean a = true;
    public boolean b = false;

    //naziv metode se piše malim slovom
    public void pisi(){
        //prikaz vrednosti rezultata logičkih izraza

        System.out.println("ne a = " + (!a));
        System.out.println("a i b = " + (a
&amp;&amp;&amp;&amp;&amp;&amp;&amp;b));
        System.out.println("a ili b = " + (a || b));

    }

    public static void main(String[] args) {
        // kreira se objekat klase. Počine malim slovom
        PrimenaLogickihTipovaPodataka primenaTipova = new
PrimenaLogickihTipovaPodataka();
        /*kreirani objekat poziva metodu za pisanje*/
        primenaTipova.pisi();
    }
}
```

```
run:
ne a = false
a i b = false
a ili b = true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Slika 1.4.1 Izlaz iz pokrenutog programa

## ZADATAK 4

## Samostalno vežbanje korišćenja logičkih tipova podataka

Prethodni primer pokušajte da proširite sa još dve logičke promenljive c i d. Dodelite im vrednost po slobodnom izboru.

Kao rezultat prikažite vrednosti sledećih logičkih izraza:

1. ne a i b - u Javi (!a && b)
2. ne (a ili b) i (c ili d) - u Javi !(a || b) && (c || d)

## BOOLEAN TIP PODATAKA (VIDEO)

*Video objašnjava šta je boolean tip podataka*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 1.6 Znakovni tip podataka

### ŠTA JE CHAR?

*Znakovni tip koji se naziva char, može da predstavlja mala i velika slova, cifre, znakove interpunkcije i kontrolne znake.*

Imajući u vidu da se različiti znakovi kod računara često koriste u programskom jeziku Java za predstavljanje pojedinačnih znakova postoji **znakovni tip** koji se naziva **char**.

Tip **char** u memoriji zauzima **2 bajta**, tj. za kodiranje svakog znaka se koristi 16-bitni **Unicode**, to omogućava, korišćenje većine alfabeta koji se koriste u svetu.

Tip podataka char sadrži pojedinačne znakove alfabeta kao što su:

- mala i velika slova (A,a,..., Z, z)
- cifre (0,1,...)
- znakove interpunkcije (\*,!, ?,...)
- kontrolne znake (razmak, novi red, tabulator...)

Znakovni literali se u programu uokviruju jednim apostrofom:

```
'm'  
'y'  
'A'
```

## UNICODE I ASCII KOD

*Unicode je šema za šifriranje znakova u bitne brojeve koju podržava Java.*

Računari interno koriste binarne brojeve. Znaci se čuvaju u računaru kao sekvence nula i jedinica (0 i 1). Mapiranje znaka u njegov binarni prikaz se naziva "**Encoding sheme**" ili šema šifrovanja znaka. Kako su znakovi šifrovani u niz nula i jedinica nam govori upravo ova šema

za znakova. Java podržava *Unicode* šemu šifrovanja znakova koja je kao standard donešena od strane Unicode konzorcijuma da bi se podržale sve razlike u jezicima na svetu.

Prvobitno je *Unicode* šifrovanje zamišljeno da radi 16-bitno, međutim ispostavilo se da to nije dovoljno pa je uvedeno i 32-bitno šifrovanje kako bi se podržali i jezici poput kineskog. Pošto je 16 bita, tj. 2 bajta, možemo ga predstaviti i sa četiri heksadecimalne vrednosti. Naprimer engleska reč *Welcome*, na kineskom se piše sa dva znaka koji se heksadecimalno mogu predstaviti kao: `\u6B22\u8FCE`. `\u` predstavlja UNICODE vrednost.

## KONVERZIJA ZNAKOVNIH U NUMERIČKE TIPOVE I OBRNUTO

*Znakovni tip -char - se može pretvoriti u broj kao što se i broj može pretvoriti u znakovni tip*

Znakovni tip (*char*) se može pretvoriti u broj kao što se i broj može pretvoriti u znakovni tip. Ovo može zvučati čudno ali pošto svaki broj koji je između 0 i 65536 (16 bita) predstavlja određeni znak po UNICODE-u možemo izvršiti konverziju i broj pretvoriti u znak.

Primer 1: Heksadecimalni broj 41 predstavlja slovo A

```
char karakter = (char)0XAB0041;  
System.out.println(karakter); // ovaj kod će ispisati veliko slovo A.
```

Ukoliko pokušamo da pretvorimo broj sa decimalnim zareзом u znakovni tip prvo će se otpisati decimalni deo a zatim će se broj konvertovati u karakter.

Primer 2: Decimalan broj 65.25 predstavlja slovo A

```
char karakter = (char)65.25;  
System.out.println(karakter); // Ispisuje slovo A
```

Ukoliko uradimo obrnuto konverziju pa karakter A pretvorimo u broj takođe ćemo dobiti broj 65.

```
int i = (int)'A';  
System.out.println(i); // Ispisuje 65
```

## KONVERZIJA TIPOVA PODATAKA (VIDEO)

*Video pokazuje kako možete da promenite tip nekog podatka*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**



## KONTROLNI ZNACI

*Kontrolni znaci formatiraju prikaz strigova i drugih znakova na izlaznom uređaju računara (npr. Da monitoru ili štampaču).*

Kontrolni znaci se u programima predstavljaju sa nekoliko znakova koji se nalaze unutar apostrofa:

```
'\n'  
'\t'
```

Svaki od prethodnih primera predstavlja po jedan znak. Prvi je 16-bitni znakovni tip koji označava **novi red**, a drugi znakovni tip **tabulator**. Na sledećoj slici prikazani su kontrolni (specijalni) znakovni tipovi sa svojim nazivima na engleskom, ASCII oznakama i objašnjenjima.

Spec. karakter	Naziv	ASCII vrednost i značenje
\a	Bell (beep)	007 zvuk bipa
\b	Backspace	008 pomera pokazivač ulevo
\t	Horizontal Tab	009 pomeranje do sledećeg tab-a
\n	New line	010 počinje novu liniju
\v	Vertical Tab	011 vrši vertikalno tabuliranje
\f	Form feed	012 pomera se za jednu liniju dole
\r	Carriage return	013 pomeraj na početak linije
\0	Null	000 karakter nule
\"	Double Quote	034 za upotrebu unutar stringa sa "
\'	Single Quote	039 za upotrebu unutra char sa '
\\	Backslash	092 obratna dupla kosa crta

Slika 1.5.1 Kontrolni znaci sa njihovim ASCII oznakama i značenjem

Bilo koja (celobrojna) vrednost tipa **char** može da se izrazi u obliku oktalnog ili heksadecimalnog literala. Na taj način mogu da se izraze i vrednosti koje ne predstavljaju nijedan od štampanih ili upravljačkih znakova na datom računaru.

Heksadecimalne vrednosti pišu se nizom od tačno četiri heksadecimalne cifre (0, 1, 2,..., 9, **A**, B, ..., **F**; mala slova su dozvoljena) iza znaka **\u**. Na taj način može da se predstavlja bilo koji **Unicode** znak, čak i ako to ne podržava okruženje.

### Važna napomena:

- Tip podataka **char** ne sadrži nikakve informacije o fontu.
- Tip podataka **char** ne može da sadrži celu reč, već samo jedan znak!
- **Stringovi** (tj. nizovi znakova), koji su jako često korišćena vrsta podataka u programiranju, nemaju odgovarajući primitivni tip u Javi.
- Skup stringova se u Javi smatra klasnim tipom, odnosno stringovi su objekti unapred definisane klase **String**.
- String literal se uokviruje duplim apostrofom - primer: "CS101", dok se znakovni literal uokviruje jednim apostrofom - primer 'ž'

## PRIMER 5

### *Vežbanje rada sa tipom char i specijelnim znacima*

Zadatak:

1. Kreirati klasu RadSaKarakterima.java koja pripada paketu lo2;
2. Klasa sadrži dva znakovna polja zn1 i zn2;
3. Dodelite im proizvoljne znake;
4. Konvertovati unete znake u celobrojne vrednosti;
5. Prikazati kao rezultat zbir tih celobrojnih vrednosti;
6. Konvertovati zbir u odgovarajući znak;
7. Pokazati kako se koriste navodnici u naredbi println() (u okviru već otvorenih znaka navoda).

Rešenje:

Kreiran je NetBeans projekat RadSaKarakterima.java koja pripada paketu lo2. U linijama 9 i 10, kreirana su dva znakovna polja klase i pridružena im je inicijalna vrednost.

Linije koda 16, 17 i 18 pokazuju kako se u navodnicima pišu specijalni karakteri, kao što su novi navodnici (ili apostrofi) i ostali znaci navedeni u sekciji "Kontrolni znaci". Da ne bi došlo do sintaksne greške, obavezno ispred takvog znaka se stavlja karakter '\\'.

U dva slučaja je vršena konverzija: iz karaktera u int (linije 16, 17 i 18) i, obrnuto iz int u karaktere (linija koda 18).

```
package lo2;

/**
 *
 * @author Vladimir Milicevic
 */
public class RadSaKarakterima {
    /*prosti char tip podataka*/
    public char zn1 = 'a';
    public char zn2 = 'b';

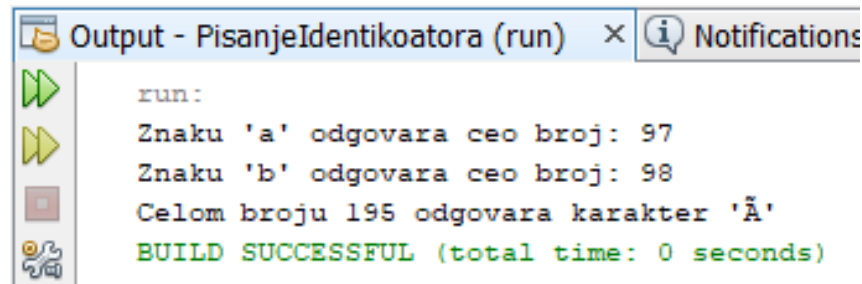
    //naziv metode se piše malim slovom
    public void pisi(){
        //prikaz vrednosti konverzije iz int u char i obratno

        System.out.println("Znaku '\\a\\' odgovara ceo broj: " + (int)zn1);
        System.out.println("Znaku '\\b\\' odgovara ceo broj: " + (int)zn2);
        System.out.println("Celom broju " + (int)(zn1+zn2) + " odgovara karakter "
+ "\\'" + (char)((int)(zn1+zn2))+ "\\");

    }

    public static void main(String[] args) {
        // kreira se objekat klase. Počine malim slovom
```

```
RadSaKarakterima primenaKaraktera = new RadSaKarakterima();  
/*kreirani objekat poziva metodu za pisanje*/  
primenaKaraktera.pisi();  
}  
  
}
```



Slika 1.5.2 Izlaz iz pokrenutog programa

## ZADATAK 5

### *Samostalno vežbanje rada sa tipom char i specijelnim znacima*

Kombinovanjem primera Primer 3 i Primer 5, pokušajte da uradite prethodni zadatak tako što će se vrednosti za zn1 i zn2 unositi sa tastature.

## PRIMITIVNI TIP PODATAKA - CHAR (VIDEO)

### *Video objašnjava šta je char primitivni tip podataka*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 1.7 Tip String

### ŠTA JE STRING?

*String je predefinisana klasa u Java biblioteci koja omogućava formiranje teksta koji čini skup znakovnih tipova, tj. char.*

Tip **char** predstavlja samo jedan znakovni tip. Da bi predstavili ceo tekst postoji tip podataka koji se zove **String**. Naprimer, sledeći kod deklariše poruku koja ima vrednost "Dobro došli u Javu"

**String** poruka = "Dobro došli u Javu";

**String** je predefinisana klasa u Java biblioteci kao i klase poput **Scanner**, **System** i **JOptionPane**. **String** nije primitivni tip podataka. **String** je ustvari referentni tip. Svaka klasa može da bude referentni tip nekog tipa ali o tome ćemo pričati u kasnijim lekcijama. Kao što se kod brojeva + za sabiranje kod **String** tipa on omogućuje da se dva teksta spoje.

Primer:

```
//Spajanje više poruka
String poruka = "Dobro došli" + " u " + " Javu";
//Spajanje poruke i broja u poruku
String poruka2 = "Poglavlje: " + 2;
//Spajanje poruke i karaktera u poruku
String poruka3 = "Sektor " + 'B';
```

Pored mogućnosti spajanja vrednosti **String** tipa postoji i mogućnost dodavanja na postojeći **String**. Ovo se radi sa +=.

Primer:

```
String poruka4 = "Java je";
poruka+=" programski jezik";
```

Ukoliko želite da izčitajte sa konzole reč možete koristiti Scanner javinu klasu.

Primer:

```
Scanner ulaz = new Scanner(System.in);
System.out.println("Unesite rec:");
String s1 = ulaz.next();
System.out.println("Upisali ste reč " + ulaz);
```

## STRINGOVI

*String nije primitivni tip podataka, već predstavlja niz znakova (tipa char) kojime se definiše neki tekst.*

U svakom programskom jeziku često radimo sa nizovima znakova, recimo - "CS101" ili "Uvod u programiranje". Ovakvi nizovi karaktera se nazivaju **stringovima**, s obzirom da nemamo adekvatniju reč u srpskom jeziku. Pre dvadesetak godina se koristio termin **niska** kao sinonim za string, ali nije zaživeo.

Treba na kraju primetiti da **stringovi** (tj. nizovi znakova), koji su često korišćena vrsta podataka u programiranju, nemaju odgovarajući primitivni tip u Javi.

Kao što smo rekli - kad za određeni tip podataka nemamo odgovarajući primitivni tip - koristimo neki klasni tip, tj. klasu, koja je već ugrađena u programski jezik Java ili definišemo sopstveni tip - recimo: Student, Igrač, Kladionica...

Skup stringova se u Javi smatra klasnim tipom, odnosno stringovi se tretiraju kao objekti koji pripadaju unapred definisanoj klasi **String**. Po sličnom principu ćemo raditi i sa datotekama, slikama, ikonama, bazama podataka i slično.

**Važna napomena:**

*Naš primarni cilj je da, u početku, ovladamo radom sa primitivnim tipovima i (samo) korišćenjem funkcionalnosti klasa koje su već ugrađene u programski jezik Java, a tek kasnije ćemo praviti sopstvene klase. Ovaj pristup se pokazao kao lakši za razumevanje kod početnika, imajući u vidu da težimo da se u početku uči što manje teorije, a da se radi što više zadataka.*

**PRIMER 6*****Vežbanje rada sa klasom String***

O klasi String će biti detaljno govora u daljem izučavanju Jave. Ovde je cilj da se klasa String prikaže kao tip podataka i da njeni objekti (stringovi) mogu da izvode određene operacije.

Zadatak:

Prikazati kako se izvodi spajanje dva stringa. Pre spajanja sve karaktere prvog stringa prikazati velikim slovima, a drugog malim.

Rešenje:

String je klasa, prema tome kreirane promenljive str1 i str2 su objekti klase String. Primenom metode toUpperCase() svi znaci u stringu se prevode u velika slova (linija koda 15). Primenom metode toLowerCase() svi znaci u stringu postaju mala slova (linija koda 17).

Spajanje dva, ili više, stringova obavlja se pomoću operatora "+" (linija koda 19).

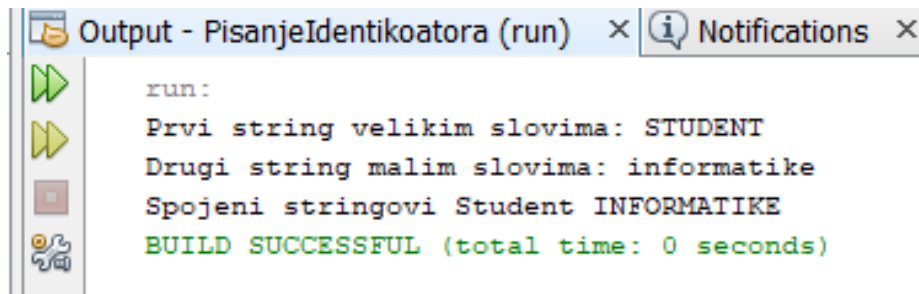
```
package l02;

/**
 *
 * @author Vladimir Milicevic
 */
public class RadSaStringovima {
    /*String tip podataka*/
    public String str1 = "Student";
    public String str2 = "INFORMATIKE";

    //naziv metode se piše malim slovom
    public void pisi(){
        //prevodjenje u velika slova
        System.out.println("Prvi string velikim slovima: " + str1.toUpperCase());
        //prevodjenje u mala slova
        System.out.println("Drugi string malim slovima: " + str2.toLowerCase());
        //spajanje stringova
        System.out.println("Spojeni stringovi " + str1 + " " + str2);
    }

    public static void main(String[] args) {
        // kreira se objekat klase. Počine malim slovom
    }
```

```
RadSaStringovima primenaStringova = new RadSaStringovima();  
/*kreirani objekat poziva metodu za pisanje*/  
primenaStringova.pisi();  
  
}  
  
}
```



Slika 1.6.1 Izlaz iz kreiranog programa

## ZADATAK 6

### *Samostalno vežbanje rada sa klasom String*

Kreirajte program koji ima sledeće funkcionalnosti:

1. Klasa sadrži 4 stringa str1, str2, str3, str4:
2. Dodelite im redom sledeće vrednosti "Kratak", "PROGRAM", "Java", "PROGRAMER";
3. Stringovi str2 i str4 dobijaju nove vrednosti tako što se originalni stringovi konvertuju u mala slova;
4. Spojiti ove stringove tako što će između susednih stringova biti po jedan blanko znak;
5. Prikazati ovaj string na ekranu;
6. Primenom metode length() prebrojati i prikazati broj karaktera u poslednjem stringu (pod 5).

## TIP PODATAKA - STRING (VIDEO)

### *Video objašnjava String tip podataka*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ✓ 1.8 Promenljive i konstante

### PROMENLJIVE

*Promenljiva je ime lokacije u glavnoj memoriji koja koristi konkretan tip podatka za skladištenje određene vrednosti*

Podaci programa se čuvaju u memorijskim ćelijama u posebnom delu programa. Programer ne mora da vodi računa o tačnim memorijskim lokacijama (adresama) gde se nalaze podaci, već se za memorijske lokacije koriste simbolička imena preko kojih se u programu ukazuje na podatke u njima. Ova imena kojima su u programu pridruženi podaci se nazivaju **promenljive**.

**Promenljiva** je **ime lokacije** u glavnoj memoriji koja koristi konkretan tip podatka za skladištenje određene vrednosti.

Već smo definisali da je **tip podatka** konkretna šema za upotrebu obrasca bitova u glavnoj memoriji za predstavljanje podataka. Promenljivu možemo shvatiti kao **kutiju** napravljenu od jednog ili više bajtova, u koju možete smestiti **vrednost konkretnog tipa**.

Pošto se podaci u kutiji mogu menjati tokom izvršavanja programa, promenljiva može ukazivati na različite podatke u različitim trenucima izvršavanja programa, ali se promenljiva uvek odnosi na istu kutiju.

Samo ime "promenljiva" naznačava da se vrednost može menjati više puta.

Promenljiva se može koristiti u programu samo ako je prethodno **deklarisana** (ili definisana) u programu.

### DEKLARACIJA PROMENLJIVE

*Deklaracijom promenljive se navodi njen tip i njeno ime.*

**Deklaracijom** promenljive se navodi njen **tip** i njeno **ime**.

**Tip promenljive** određuje zapravo **tip podataka** koje promenljiva može sadržati. Ta informacija je potrebna radi rezervisanja memorijske lokacije (kutije) čija veličina je dovoljna za smeštanje podataka odgovarajućeg tipa.

Već smo konstatovali da sintaksa označava gramatiku programskog jezika. Sintaksa se može odnositi i na pojedine delove programa, pa tako možemo govoriti i o sintaksi deklaracije promenljive.

Deklaracija promenljive se može ostvariti na nekoliko načina.

Sledi opšti oblik prvog načina za deklaraciju promenljive.

```
tipPodatka imePromenljive;
```

Njime se ne dodeljuje vrednost, već se samo zadaje tip podatka i daje ime promenljive.

U sledećem primeru su deklarisanе promenljive: **godina**, **brojSekundi**, **prosek** i **zapremina** odgovarajućeg tipa koji je naveden ispred imena svake promenljive:

```
int godina;  
long brojSekundi;  
float prosek;  
double zapremina;
```

Jedini način u Javi da se dodeli neka vrednost promenljivoj, tj. da se neki podatak "stavi" u kutiju koju promenljiva predstavlja - jeste pomoću naredbe dodele.

Naredba dodele ima opšti oblik:

```
ime = izraz;
```

gde ime predstavlja ime odgovarajuće promenljive, izraz je konstrukcija kojom se navodi ili izračunava neka vrednost, a znak = predstavlja operator dodele.

Kada se izvršava neka naredba dodele prilikom izvršavanja programa, najpre se izračunava izraz i zatim se dobijena vrednost dodeljuje promenljivoj nazvanoj ime.

Sledi opšti oblik drugog načina za deklaraciju promenljive

```
tipPodatka imePromenljive = inicijalnaVrednost;
```

Ovim iskazom se deklarira promenljiva, zadaje se njen tip podatka, rezervira se memorija za nju i u tu memoriju se stavlja vrednost. Inicijalna vrednost mora po tipu biti odgovarajuća.

## ŠTA JE KONSTANTA?

*Konstanta je promenljiva koja ne može promeniti vrednost tokom izvršavanja programa.*

Jedna promenljiva može sadržati promenljive vrednosti istog tipa tokom izvršavanja programa.

U izvesnim slučajevima, međutim, vrednost neke promenljive u programu **ne treba da se menja nakon dodele** početne vrednosti toj promenljivoj.

U Javi se u naredbi deklaracije promenljive može dodati službena reč **final** kako bi se obezbedilo da se promenljiva ne može promeniti nakon što se inicijalizuje.

Na primer, iza deklaracije:

```
final int MAX_BROJ_POENA = 100;
```

svaki pokušaj promene vrednosti promenljive MAX\_BROJ\_POENA proizvodi sintaksnu grešku.



Ove "nepromenljive promenljive" se nazivaju **konstante**, jer njihove vrednosti ostaju konstantne za sve vreme izvršavanja programa.

Obratite pažnju na konvenciju u Javi za pisanje imena konstanti:

**njihova imena se sastoje samo od velikih slova, a za eventualno razdvajanje reči služi donja crta.**

Ovaj stil se koristi i u standardnim klasama Jave u kojima su definisane mnoge konstante.

Tako, na primer, u klasi **Math** se nalazi konstanta **PI** koja sadrži vrednost broja Pi, ili u omotačkoj klasi **Integer** se nalazi konstanta **MIN\_VALUE** koja sadrži minimalnu vrednost tipa **int**.

## PRIMER 7 - KORIŠĆENJE PROMENLJIVE

*Jednom dodelom vrednosti promenljivoj je moguće deklarirati i postaviti vrednost više promenljivih*

U sledećem primeru ćemo inicijalizovati (dodeliti im vrednosti) promenljive iz prethodnog primera:

```
int godina = 19;
long brojSekundi = 100000000;
float prosek = 0.0;
double zapremina = 148.999;
```

Važno je primetiti da promenljiva može imati različite vrednosti tokom izvršavanja programa, ali sve te vrednosti moraju biti istog tipa - onog koji je naveden u deklaraciji promenljive.

Na primer, možemo uraditi sledeće:

```
int godina = 19;
godina = 20;
godina = 21;
godina = 35;
```

U prethodnom primeru, vrednosti promenljive godina mogu biti samo celobrojni brojevi tipa **int** - nikad znakovi ili logičke vrednosti.

Recimo, ne možemo uraditi sledeće:

```
godina = true;
```

Takođe, bitno je napomenuti da ne možemo dodeliti vrednost promenljivoj ukoliko je nismo deklarirali.

Jednom dodelom je moguće deklarirati i postaviti vrednost više promenljivih. Sledi opšti oblik trećeg načina za deklaraciju promenljivih:

```
tipPodatka imePrvePromenljive, imeDrugePromenljive;
```

Ovim se deklariraju dve promenljive istog tipa, ali se tim promenljivama ne daje nikakva vrednost. Ako želite, ovo možete da uradite i za više od dve promenljive. Primer:

```
int stranicaA, stranicaB, stranicaC;
double obim, površina, zapremina;
```

Sledi opšti oblik četvrtog načina za deklaraciju promenljivih:

```
tipPodatka imePrvePromenljive = inicijalnaVrednostJedan, imedrugePromenljive =
inicijalnaVrednostDva;
```

Ovim se deklariraju dve promenljive. Obe su istog tipa i u obe se stavljaju inicijalne vrednosti. Na isti način možete deklarirati i više promenljivih. Primer:

```
int stranicaA = 5, stranicaB = 6, stranicaC = 7;
double obim = 15.0, površina = 37.5, zapremina = 260.89;
```

Iako ova mogućnost donosi uštedu u pisanju, jedna od odlika dobrog stila programiranja je da se promenljive posebno deklariraju. O ovoga pravila treba odstupiti samo ako je više promenljivih povezano na neki način.

## ZADATAK 7

### *Samostalno vežbanje primene promenljivih*

Na osnovu dosadašnjeg rada, iskoristite poznate primere za kreiranje programa koji koristi deklaracije promenljivih iz primera "Primer 7".

Za sve slučajeve, kreirajte posebne NetBeans projekte, prevedite programe i prikažite rezultate njihovog izvršavanja.

## PROMENLJIVE U JAVI (VIDEO)

### *Video objašnjava primenu promenljivih u vašem programu*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 1.9 Naredbe

### ŠTA JE NAREDBA?

*Naredba (iskaz) je osnovna jedinica obrade u programima.*

**Naredba** (engl. **statement**) je osnovna jedinica obrade u programima.

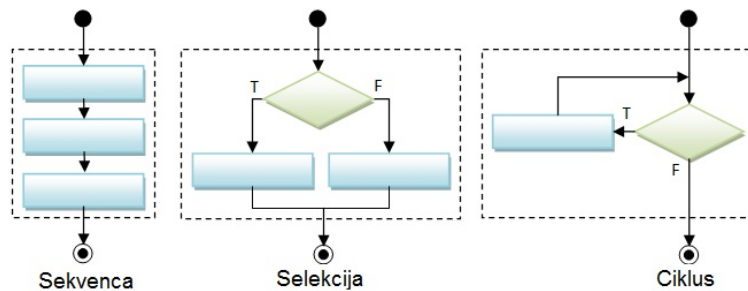
Izvršne naredbe se dele na:

- **Proste naredbe**
- **Složene naredbe**
- **Upravljačke naredbe**

**Proste naredbe** predstavljaju elementarne obrade koje ne mogu da se podele na manje delove koji bi i sami bili naredbe.

**Složene naredbe** predstavljaju strukture naredbi kojima se određuje redosled izvršavanja naredbi sadržanih u strukturi. Te strukture naredbi nazivaju se **upravljačke strukture**. One mogu da se podele u sledeće tri grupe:

- Sekvenca
- Selekcija
- Ciklusi ili petlja



Slika 1.7.1 Upravljačke naredbe

**Upravljačke naredbe** ne vrše nikakvu obradu već samo prenose tok upravljanja na neko mesto u programu. Ovoj grupi pripadaju razne naredbe skokova. Naredbe unutar složenih naredbi i same mogu da budu proste ili složene. Na taj način mogu da se ostvaruju vrlo složene obrade. Pod pojmom **naredba** se najčešće podrazumeva bilo koja vrsta naredbi: prosta, složena ili upravljačka. Ponekad, u širem smislu obuhvataće se i **deklarativne naredbe** kao što su naredbe za definisanje podataka, tipova podataka itd.

## PROSTA NAREDBA

### *Prosta naredba predstavlja elementarnu obradu u programu*

Prosta naredba predstavlja elementarnu obradu u programu.

Njen opšti oblik je:

```
izraz;
```

Izvršavanje proste naredbe sastoji se u izračunavanju izraza. Dobijena vrednost izraza ne koristi se ni za šta.

Trajni rezultati naredbe dobijaju se kao bočni efekti pojedinih operatora unutar izraza.

Operatori s bočnim efektima jesu operatori za dodelu vrednosti i operatori `++` i `--`, o kojima ćemo detaljno govoriti u sledećem predavanju.

Specijalnu, i ređe korišćenu, naredbu čini takozvana prazna naredba (engl.empty statement).

Ona se sastoji samo od tačke-zareza (;). Prazna naredba nema nikakvog efekta.

## BLOK NAREDBI

*Blok ili sekvenca je niz naredbi koje se izvršavaju jedna za drugom.*

**Blok** ili **sekvenca** je **niz naredbi** koje se izvršavaju jedna za drugom.

Blok naredbi u jeziku Java piše se u obliku niza naredbi unutar vitičastih zagrada (`{ }`).

Na taj način se jasno ističe gde su početak i kraj sekvence:

```
{
    naredba1
    naredba2
    ...
    naredbaN
}
```

## PRIMER 8

*Programe čine blokovi naredbi, tj. instrukcija kojim se saopštava procesoru računara šta se od njega traži da uradi.*

**Primer 8-1 :**

```
{ p = a; a = b; b = p; }
```

Prvi primer je jedan jednostavan blok naredbi kojim se međusobno zamenjuju vrednosti promenljivih **a** i **b** korišćenjem pomoćne promenljive **p**.

Treba posebno obratiti pažnju na to da je znak **;** ispred zagrade **}** neophodan. Naime, **b = p** je samo izraz i tek je **b = p;** naredba koja može da bude deo sekvence.

**Primer 8-2 :**

```
{
    int broj1 = 10;
    int broj2 = 20;
    double aritmetičkaSredina = (broj1 + broj2) / 2.0 ;
    System.out.println("Aritmetička sredina je: " + aritmetičkaSredina);
}
```

Drugi primer predstavlja jedan malo složeniji bloka naredbi. U njemu se na osnovu celobrojnih vrednosti dva broja računa aritmetička sredina ta dva broja.

## ZADATAK 8

### *Samostalno vežbanje korišćenja naredbi i blokova*

Iskoristite blok naredbi i primera Primer 8-2 (prethodna sekcija) i ugradite ga u odgovarajući program. Neophodno je da se kreira nov NetBeans projekat, da se izvrši njegovo prevođenje i dokumentuje rezultat njegovog izvršavanja.

## JEDNOSTAVNI I BLOK ISKAZI (VIDEO)

### *Video objašnjava šta je iskaz u Javi i vrste iskaza koji se koriste*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 1.10 Komentari

### ŠTA SU KOMENTARI?

*Komentar predstavlja tekst na prirodnom jeziku u programu koji objašnjava delove programa drugim programerima*

**Komentar** predstavlja **tekst na prirodnom jeziku** u programu koji objašnjava delove programa drugim programerima. Java prevodilac zanemaruje komentare tokom prevođenja programa.

Važnost komentara se ogleda u tome što drugi programeri mogu razumeti delove koda, sam programer se posle izvesnog vremena može podsetiti šta je hteo da uradi određenim kodom, kao i za izradu projektne dokumentacije.

U Javi se mogu koristiti **tri vrste komentara**.

**Komentar u jednoj liniji:** Tosu kratki komentari koji se pišu u jednom redu teksta. Njihov početak u nekom redu se označava simbolom `//` i obuhvataju tekst do kraja tog reda.

Na primer:

```
int prosecnaOcena = 10; // inicijalizacija procečne ocene
```

Kod nekih programera možete videti i ovakvu vrstu komentara iznad samog koda.

Na primer:

```
// inicijalizacija procečne ocene  
int prosecnaOcena = 10;
```

**Komentar u više linja:** Predstavlja duže komentare koji se prostiru u više redova. Oni počinju simbolom `/*`, obuhvataju tekst u više redova i završavaju se simbolom `*/`.

Na primer:

```
/*
Uzimanje korisničkog imena i lozinke od korisnika.
Ukoliko korisnik unese pogrešnu lozinku, biće obavešten o tome.
Ukoliko korisnik unese pogrešnu lozinku tri puta, biće mu zabranjen pristup 15
minuta.
*/
```

**Dokumentacioni komentar:** Počine sa `/**` a završava se sa `*/`. Služi za pisanje dokumentacije direktno unutar samog koda. Pomoćni program **javadoc** od ovih komentara kreira **html** dokument koji se može čitati bilo kojim veb pretraživačem. Za razliku od običnih komentara, dokumentacioni komentari mogu sadržati **HTML** tagove i specijalne reči koje počinju znakom `@`.

Na primer:

```
/**
Računanje zbira dva broja
@param prvi broj
@param drugi broj
@return rezultat
@author student
*/
```

## PRIMER 9

### *Vježbanje rada sa komentarima*

Zadatak:

Napisati Java program i u njega ugraditi sledeće komentare:

*Komentar 1:*

```
/**
Računanje zbira dva broja
@param prvi broj
@param drugi broj
@return rezultat
@author student
*/
```

*Komentar 2:*

```
/*Unos dva broja*/
```

*Komentar 3:*

```
//Računanje i prikazivanje rezultata
```

NAPOMENA: Rad sa obektima i pozivima metoda će biti detaljno izučavan u nastavku. Ovde je metoda iskorišćena za primer upotrebe dokumentacionih komentara u Javi.

```
package l02;

/**
 * *
 * @author Vladimir Milićević
 */
public class SabiranjeDvaBroja {

    /**
     * *
     * @param args
     * @param prvi broj
     * @param drugi broj
     * @return rezultat
     */
    int saberi (int prvi, int drugi) {
        int zbir = prvi + drugi;
        return zbir;
    }

    public static void main(String[] args) {
        /*Unos dva broja*/
        int prvi = 10;
        int drugi = 20;
        //sabiranje dva broja
        SabiranjeDvaBroja sdb = new SabiranjeDvaBroja();
        int rezultat= sdb.saberi(prvi, drugi);
        System.out.println("prvi + drugi = " + rezultat);
    }
}
```

## ZADATAK 9

### *Utvrđivanje rada sa komentarima*

Za sve kreirane primere i zadatke za samostalni rad, koji su rađeni u okviru ove lekcije, definišite i ugradite odgovarajuće dokumentacione komentare po uzoru na Primer 9.

## ▼ Poglavlje 2

# Objekat i klasa kao tip podataka

## SADRŽAJ

*Klasa je tip podatka koji korisnik može sam da definiše i koji je složen jer sadrži druge podatke različitih tipova. Objekat je podatak čiji je tip definisala klasa koja ga stvara.*

1. Pojam softverskog objekta
2. Načini pristupa podacima
3. Omotačke klase
4. Prikazivanje podataka
5. Unos podataka
6. Paketi u Javi

## ▼ 2.1 Pojam softverskog objekta

### ŠTA JE OBJEKAT?

*Objekat predstavlja entitet iz stvarnog života sa svojim jedinstvenim identitetom predstavljeni sa skupom podataka koji predstavljaju jednog činioca objekto-orijentisanog modela.*

Kao što smo ovlaš napomenuli, pored primitivnih tipova podataka, postoji i **objektni tip podataka**.

Objektno-orijentisan pristup programiranju se zasniva na **manipulisanju objektima** - **definisanjem objekata** i **njihovom interakcijom**. Objekti su profesor, fakultet, bankovni račun, student i slično, odnosno sve stvari koje poseduju neku funkcionalnost ili nad kojima se može izvoditi neka aktivnost u određenoj situaciji.

Ove ideje je najbolje potkrepiti jednim primerom iz svakodnevnog života, jer se u objektno-orijentisanoj metodologiji teži da se rešavanjem problema simulira pristup koji se primenjuje u realnim situacijama. Razmotrimo zato kako radi, recimo, tipični restoran: Kada gost dođe u restoran, on izabere sto i poručuje jelo i piće iz jelovnika. Jedan kelner prima porudžbinu i donosi poručeno jelo i piće. Kuvar sprema jelo i šanker priprema piće. Na kraju, isti kelner donosi račun i gost ga plaća.



Koji objekti učestvuju u radu restorana? To su očigledno: gost, kelner, kuvar, šanker, ali to nije sve što je dovoljno za rad restorana. Druge neophodne stvari su: sto, jelovnik, jelo, piće i račun. Sve su to objekti čijim se međusobnim manipulisanjem realizuje rad restorana.

U primeru vidimo da postoje razni tipovi objekata. Njihova imena (gost, kelner, jelovnik, račun ...) nama otprilike govore šta je njihova uloga, ali računari ne znaju šta se podrazumeva pod tim imenima. Da bismo to opisali za računare, definišemo **klase objekata**.

## ŠTA JE OBJEKAT U JAVI? (VIDEO)

*Video objašnjava šta je objekat u Javi*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ŠTA JE KLASA?

*Klasa u Javi opisuje računaru koje podatke svi objekti klase imaju i šta takvi objekti mogu da urade*

**Klasa** u Javi opisuje računaru **koje podatke svi objekti klase imaju i šta takvi objekti mogu da urade**. **Svaki objekat jedne klase ima iste podatke** (stvari koje ga obeležavaju) i **iste mogućnosti** (stvari koje može uraditi).

Na primer, svaki objekat klase **Porudžbina** treba da sadrži gosta koji je napravio određenu porudžbinu i spisak jela i pića sa cenama od kojih se porudžbina sastoji, kao i mogućnost da se izračuna ukupna vrednost porudžbine. Primetimo da može postojati više objekata iste klase.

Restoran može imati dva kuvara, tri kelnera, dvadeset jela na meniju, četiri gosta u datom trenutku i tako dalje. Zašto restoran nema samo jednog zaposlenog, jer bi u principu sve poslove kuvara, šankera i kelnera mogao obavljati jedan čovek? To bi možda bilo izvodljivo ako restoran ima samo jednog gosta, ali je praktično nemoguće ako ima više njih. Funkcionisanje restorana je podeljeno u više poslova kako bi se oni izvodili efikasnije i bolje.

Slično tome, *objektno-orjentisani programi distribuiraju zadatke na više objekata* kako bi programi bili efikasniji i bolji. Pored toga, na taj način se programi mogu lakše menjati i proširivati.

## SOFTVERSKI OBJEKAT

*Softverski objekti oponašaju stvarne objekte po tome što imaju pridružene podatke ( tj. atributi objekta) i mogućnosti (koji određuju šta objekat može da uradi).*

**Objekat** predstavlja **entitet iz stvarnog života** sa svojim jedinstvenim identitetom. Tako, student, sto, stolica, taster na tastaturi, telefon i tako dalje mogu se posmatrati kao objekti.

Pored ovih "fizičkih" stvari, pojam objekta obuhvata i apstraktne entitete kao što su krug, kredit, brak i tako dalje.

Jedinstven identitet objekta je određen njegovim **svojstvima(atributima)** i **ponašanjem**. Na primer, student se može identifikovati po imenu i prezimenu, fakultetu koji studira, broju indeksa i slično, a njegovo ponašanje se sastoji od mogućnosti da polaže ispite iz određenih predmeta, da bude član studentskih organizacija i slično.

**Softverski objekti** oponašaju stvarne objekte po tome što imaju **pridružene podatke** (koji određuju šta su **svojstva** ili **atributi** objekta) i **moćgućnosti** (koji određuju šta objekat može da uradi). **Podaci** koje objekti sadrže se predstavljaju **promenljivim (atributima ili poljima)**, a njihove **mogućnosti metodima (procedurama)**..

Svaki **atribut** ima **ime** i **tip podatka**, recimo **brojIndeksa** kao ceo broj (primitivni tip podataka **int**). Svaka **metoda** ima **ime** i **tip podatka** koji vraća (postoji mogućnost i da metoda ne vraća ništa), recimo **prikažiPodatke** kao metoda koja ne vraća ništa (u Javi se obeležava sa **void**). O metodama ćemo govoriti detaljnije u narednim predavanjima

## PRIMER SOFTVERSKOG OBJEKTA: OBJEKAT STUDENT

*Objekti iste vrste se definišu koristeći zajedničku klasu. Klasa definiše sve attribute objekata iste klase i operacije (metode) koje se mogu nad njima izvršiti.*

Slika prikazuje objekat **obj** klase **Klasa** koji ima sledeća svojstva:

- **svojstvo1** tipa **tip1** i vrednost **vrednost1**
- **svojstvo2** tipa **tip2** i vrednost **vrednost2**
- **svojstvo3** tipa **tip3** i vrednost **vrednost3**

i sledeće metode:

- **metod1** koji vraća podatak tipa **tip1**
- **metod2** koji vraća podatak tipa **tip2**
- **metod3** koji vraća podatak tipa **tip3**

obj:Klasa		
svojstvo1	: tip1	= vrednost1
svojstvo2	: tip2	= vrednost2
svojstvo3	: tip3	= vrednost3
metod1	() : tip1	
metod2	() : tip2	
metod3	() : tip3	

Slika 2.1.1 Tipični objekat

Objekti iste vrste se **definišu koristeći zajedničku klasu**. Na primer, svaki objekat klase **Student** može imati atribute: **ime**, **prezime**, **brojIndeksa** i **prosek** čije vrednosti označuju konkretnog studenta. Pored toga, svaki takav objekat može imati metode:

**prikažiPodatke**, **dodeliStipendiju**, **smanjiZaduženje**, **suspendujStudenta** kojima se može nešto uraditi sa konkretnim studentom. Softverski oblik objekta **s1** klase **Student** je prikazan na slici:

s1:Student		
ime	: String	= "Milovan"
prezime	: String	= "Petrović"
brojIndeksa	: int	= 2089
prosek	: double	= 8.14
prikažiPodatke ()	: void	
dodeliStipendiju ()	: void	
smanjiZaduženje ()	: void	
suspendujStudenta ()	: void	

Slika 2.1.2 Jedan objekat klase Student

## PRIMER SOFTVERSE KLASE - KLASA STUDENT

*Klasa je opis objekata sa zajedničkim svojstvima.*

**Klasa** je **opis objekata sa zajedničkim svojstvima**. **Definicijom neke klase** se određuju **atributi** ili polja (promenljive) i **metodi** (procedure) koje poseduju **svi objekti te klase**. Klase dakle definišu **šablon** kako izgledaju pojedini objekti tih klase. Vrlo je važno razumeti da se u Java programu pišu **klase**, a objekti se ne pišu nego se **konstruišu (kreiraju) na osnovu tih klase**.

Softverski oblik klase **Student** je prikazan na slici 3 :

Student		
ime	: String	
prezime	: String	
brojIndeksa	: int	
prosek	: double	
prikažiPodatke ()	: void	
dodeliStipendiju ()	: void	
smanjiZaduženje ()	: void	
suspendujStudenta ()	: void	

Slika 2.1.3 Klasa Student

Na osnovu prethodne dve slike možete uočiti jasnu razliku između klase i objekata: objekat ima konkretne vrednosti atributa, dok klasa samo definiše atribut i tip atributa.

Dobra analogija klase i objekata u Javi je građevinski nacrt zgrade na papiru i izgrađenih zgrada u arhitekturi i građevinarstvu:

Zajednička svojstva i procedure za sve studente predstavljaju klasu **Student**, konkretni studenti koji su uneti u sistem predstavljaju **objekte klase Student**. Na osnovu jedne klase može konstruisati više objekata. Ako posmatramo primer klase Student o kojoj smo govorili, njena definicija bi u programskom jeziku Java izgledala otprilike ovako:

```
class Student {
    String ime;
    String prezime;
    int brojIndeksa;
    double prosek;
    public void prikaziPodatke() {
    }
    public void dodeliStipendiju() {
    }
    public void smanjiZaduzenje() {
    }
    public void suspendujStudenta() {
    }
}
```

## DVA STUDENTA KAO DVA OBJEKTA

*Svaki konstruisani objekat neke klase ima svoje primerke (objektnih) promenljivih i metoda te klase.*

Detalji definicije ove klase za sada nisu važni, već je bitno to da na osnovu nje možemo konstruisati više objekata klase **Student**. Ako konstruišemo, recimo, dva objekta **s1** i **s2** te klase sa svojim individualnim podacima, onda je njihov izgled u memoriji računara prikazan na sledećoj slici:

s2:Student				s1:Student			
ime	: String	=	"Dragana"	ime	: String	=	"Milovan"
prezime	: String	=	"Pajović"	prezime	: String	=	"Petrović"
brojIndeksa	: int	=	2287	brojIndeksa	: int	=	2089
prosek	: double	=	9.00	prosek	: double	=	8.14
prikažiPodatke ()	: void			prikažiPodatke ()	: void		
dodeliStipendiju ()	: void			dodeliStipendiju ()	: void		
smanjiZaduzenje ()	: void			smanjiZaduzenje ()	: void		
suspendujStudenta ()	: void			suspendujStudenta ()	: void		

Slika 2.1.4 Dva objekta klase Student

Primitimo sa slike da svaki konstruisani objekat neke klase ima svoje primerke (objektnih) promenljivih i metoda te klase. To je glavna odlika objekata i jedna od karakteristika koja objektno-orjentisano programiranje razlikuje od proceduralnog programiranja.

## METODI

*Metod u Javi predstavlja program koji obavlja neki specifičan zadatak i čini ga niz naredbi i promenljivih koje predstavljaju neku funkcionalnu celinu sa svojim posebnim imenom.*

Kao što smo pomenuli, **objekti su složeni podaci** i sastoje se od sastavnih delova kojima se može nezavisno manipulirati, a to su **atributi** i **metode**.

**Metode** u Javi predstavljaju potprograme koji obavljaju neki specifičan zadatak, tj. metoda je niz naredbi i promenljivih koja predstavlja neku funkcionalnu celinu sa svojim posebnim imenom. Izvršavanje metoda se postiže navođenjem **imena** i **argumenata metoda** u programu na mestu gde je potrebno uraditi zadatak koji metoda obavlja - ovaj postupak se naziva **pozivanjem metoda**. **Argumenti metoda** su podaci koji se prosleđuju metodi.

Java sadrži veliki broj klasa sa unapred napisanim metodama koje se mogu koristiti u programima, bez ikakvog razumevanja kako su oni napisani ili kako rade. Recimo, nizovi znakova (stringovi) su u programskom jeziku Java predstavljeni klasom String. Svaki konkretan niz znakova je predstavljen objektom klase String.

:Literale kojima se označavaju stringovi smo koristili u našem prvom programu:

```
"Zdravo, svete!"
```

Svaki string literal mora imati dvostruke apostrofe na početku i na kraju niza znakova. Broj znakova u stringu je dužina stringa koji može biti nula ili više - dužina praznog stringa ("") je nula, a dužina stringa "CS101" je pet. Unutar string literala se mogu koristiti specijalni znakovi \n, \t, \\ i ostali, kao i Unicode znakovi. Nad stringovima je dozvoljena operacija spajanja (konkatenacija) stringova i njena oznaka je +, jer podseća na spajanje stringova.

Na primer, rezultat izraza

```
"Zdravo, " + "svete" + "!"
```

je string:

```
"Zdravo, svete!"
```

## ATRIBUTI

*Klasa definiše nov tip podataka atributa, a vrednosti tog novodefinisanog tipa su objekti te klase.*

Prva uloga klase je da budu sredstvo za opis objekata - klasa definiše attribute i metode koje imaju svi objekti koji pripadaju toj klasi. Klasa definiše novi tip podataka, a vrednosti tog novodefinisanog tipa su objekti te klase. Najčešće svaki objekat ima različite attribute i rezultat rada svake metode zavisi od atributa objekta.

Ovakve atribute i metode nazivamo **objektnim atributima i metodima**, pošto se **različito ponašaju u zavisnosti od stanja objekta**.

Recimo, u klasi **String** postoji definisana metoda **length()** koja vraća različitu dužinu za svaki objekat klase **String**, tj. broj znakova jednog stringa, kao na primer:

```
"CS101".length()
```

vraća broj 5, dok

```
"Zdravo, svete!"
```

vraća broj 14.

Dakle, rezultat rada metode vraća različitu vrednost u zavisnosti od stanja objekta koji je poziva.

Druga uloga klase je da definišu atribute i metode koji su zajednički i pritom se isto ponašaju za sve objekte određene klase. Ovi atributi i metodi se nazivaju **statički članovi klase**, tj. **statički atributi i statički metodi**. Na primer, u klasi **String** postoji **statička metoda valueOf(parametar)** koja vraća prosleđeni parametar kao string i ne zavisi od stanja konkretnog objekta, već od stanja parametra, na primer:

```
String.valueOf(5)
```

vraća string:

```
"5"
```

Kao što možete primetiti:

- statičke metode se pozivaju po nazivu klase
- objektne metode po nazivu objekta

## PRIMER 10 - PRIKAZIVANJE TRENUTNOG VREMENA

### *Prikazivanje podataka na konzoli razvojnog okruženja*

Cilj ove pokazne vežbe jeste da se pokaže kako u kreiranoj Java klasi može da se kreira objekat neke klase. Objekat može da se kreira pomoću konstruktora, primenom operatora new (linija koda 19), a može i pomoću produkcione (factory) metode (linija 17). Sledi listing navedene klase nakon čega će uslediti objašnjenje koda.

```
package com.metropolitan.lekcija2;

import java.text.SimpleDateFormat;
import java.util.Calendar;

/**
 *
```

```
* @author Vladimir Milicevic
*/
public class CurrentTime {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //kreiranje objekta Factory metodom
        Calendar cal = Calendar.getInstance();
        //Kreiranje objekta konstruktorom
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
        System.out.println( sdf.format(cal.getTime()) );
    }
}
```

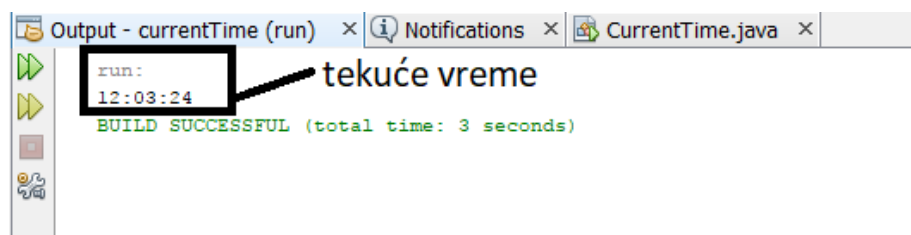
Prvo na šta treba obratiti pažnju su linije koda 16 i 17. Prvo je kreiran objekat klase Calendar čiji je zadatak da preuzme trenutno vreme u formatu koji je definisan u liniji 17.

Za definisanje formata, u kojem će trenutno vreme biti pokazano, zadužena je klasa SimpleDateFormat. Kreiran je objekat ove klase koji diktira prikazivanje trenutnog vremena u formatu "HH:mm:ss".

Konačno, linija koda 19 prikazuje na konzoli tekuće vreme.

Da bi bilo moguće koristiti klase Calendar i SimpleDateFormat, bilo je neophodno učitati pakete u kojima se te klase nalaze. Pogledati listing i linije koda 3 i 4.

Konačno, iz razvojnog okruženja NetBeans IDE se pokreće kreirana aplikacija i rezultat izvršavanja (tekuće vreme) je prikazan sledećom slikom.



Slika 2.1.5 Rezultat izvršavanja kreiranog programa na konzoli

## ZADATAK 10

### *Vežbanje kreiranja i primene objekata klase.*

Kreirajte klasu Osoba sa sledećim poljima:

- String ime;
- String prezime;
- int broj godina;

- Kreirajte prazan konstruktor klase i u njemu pomoću naredbe `println` omogućiti prikazivanje podataka o Osobi;

Kreirajte klasu `Main` koja sadrži poznatu `main()` metodu.

U `main()` metodi kreirajte objekat klase `Osoba`.

Pokrenite program i dokumentujete rezultate izvršavanja.

## UPOTREBA OBJEKATA (VIDEO)

### *Video objašnjava kreiranje i primenu objekata*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ŠTA JE KLASA? (VIDEO)

### *Video pokazuje kako se kreira klasa u Javi koja predstavlja automobil*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 2.2 Načini pristupa podacima

### PROMENLJIVA VREDNOSNOG TIPa

*Promenljiva vrednosnog tipa sadrži vrednost podatka koga predstavlja. Pod vrednošću promenljive podrazumeva se vrednost sadržanog podatka.*

Prema načinu uskladištavanja i pristupanja, podaci se dele na:

- **Vrednosne podatke**
- **Pokazne podatke**

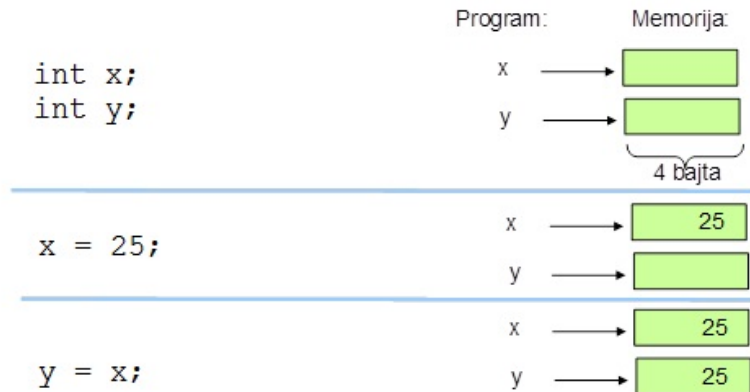
**Promenljiva vrednosnog tipa** sadrži **vrednost podatka** koga predstavlja. Pod vrednošću promenljive podrazumeva se vrednost sadržanog podatka. Operacije nad promenljivom obrađuju taj, neposredno sadržani podatak. Promena vrednosti jedne promenljive nikad ne utiče na vrednost druge.

Ovo možete zamisliti kao kutiju određene veličine u bajtovima koja sadrži **konkretnu vrednost** koja se može menjati.

U jeziku Java **primitivni tipovi** su **vrednosni tipovi podataka**.



Na slici 1 je prikazan način smeštanja vrednosti primitivnih promenljivih u memoriju računara, tj. promenljivih veličina koje su deklarisanе da predstavljaju primitivan tip podataka u memorijske lokacije definisane svojim adresama. Vidimo da se vrednosti ovih primitivnih veličina nalaze u odgovarajućim adresama memorije računara:



Slika 2.2.1 Primer smeštanja vrednosti primitivnih promenljivih u memoriju

U jeziku Java **klasni tipovi** su **pokazni tipovi podataka**. O njima ćemo govoriti nešto kasnije.

## PROMENLJIVA POKAZNOG TIPRA

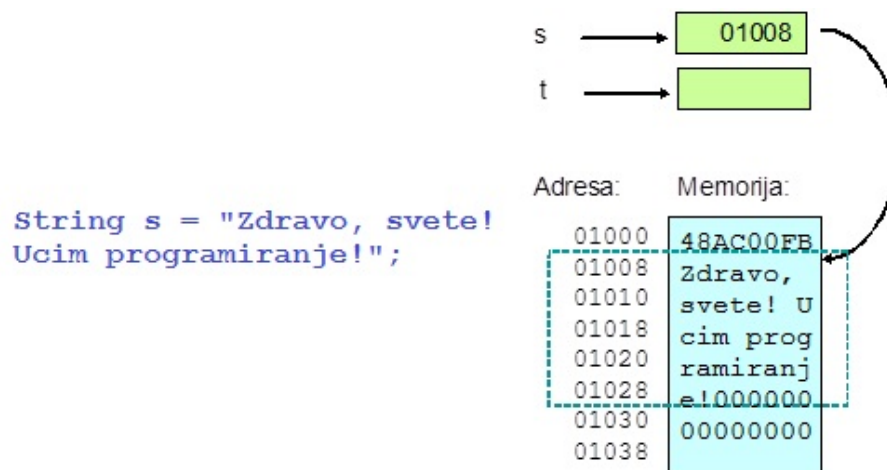
*Promenljiva pokaznog tipa sadrži samo adresu mesta (referencu) u memoriji gde se nalazi vrednost predstavljenog podatka*

Promenljiva pokaznog tipa sadrži samo adresu mesta (referencu) u memoriji gde se nalazi vrednost predstavljenog podatka. Pod vrednošću promenljive podrazumeva se vrednost pokazanog podatka, a ne vrednost neposredno sadržanog pokazivača.

Operacije nad promenljivom obrađuju pokazani podatak, a ne neposredno sadržani pokazivač. Više promenljivih mogu da pokazuju na fizički isti podatak. U tom slučaju promena vrednosti jedne promenljive promeniće i vrednost ostalih promenljivih koje predstavljaju (pokazuju na) isti podatak. Ovo možete zamisliti kao kutiju određene veličine u bajtovima koja sadrži adresu koja se može menjati.

Imajući u vidu da su objekti u Javi složen tip podataka, prilikom deklaracije promenljive koja ukazuje na objekat, nećete automatski dobiti i objekat.

Na j slici 2 je prikazana je situacija koja se dobija prilikom kreiranja objekta s klase String, koji predstavlja neki tekst. Tekst se smešta u slobodni deo memorije, a Java obezbeđuje zapisivanje u memoriji računara, kao vrednost promenljive tipa objekat, **adresu** na kojoj se nalazi sam objekat, tj. u ovom slučaju, tekst. Znači, za klasne tipove podataka, promenljiva je **referenca na objekat**, i ne obezbeđuje memorisanje objekta!



Slika 2.2.2 Primer smeštanja adrese prvog objekata tipa string u memoriju

## PRIMER SMEŠTAJA U MEMORIJU OBJEKTA TIPRA STING

*Ime objekta ukazuje na adresu u memoriji gde se on smešta.  
Izjednačavanje dva objekta je izjednačavanje adresa gde se oni smeštaju u memoriju.*

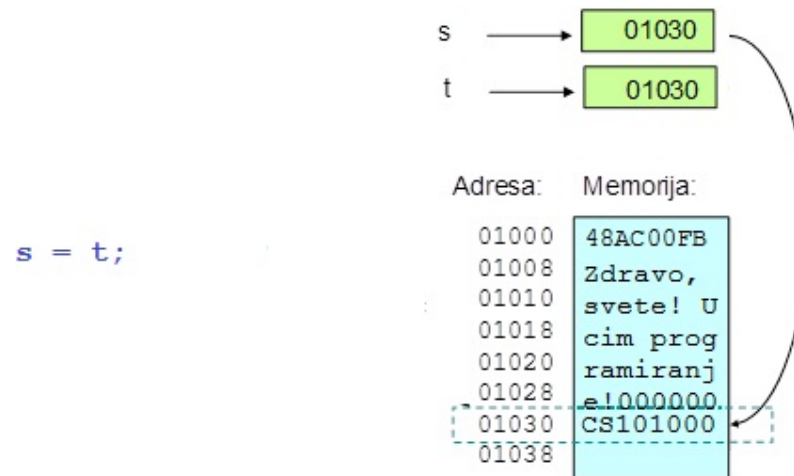
Slično se dešava kada se kreira neki drugi objekat klase String, sistem memoriše adresu gde se nalazi objekat t, tj. početak teksta koji predstavlja ovaj objekat. Slično se dešava kada se kreira neki drugi objekat klase String, sistem memoriše adresu gde se nalazi objekat t, tj. početak teksta koji predstavlja ovaj objekat.



Slika 2.2.3 Primer smeštanja adrese drugog objekata tipa String u memoriju

Ako se vrši "izjednačavanje" dva objekta tipa String, onda se ustvari vrši dodeljivanje adresa na kojima se nalaze ovi objekti, tj. u memoriji. Ako se vrši "izjednačavanje" dva objekta tipa String, onda se ustvari vrši dodeljivanje adresa na kojima se nalaze ovi objekti, tj. u memoriju za objekat **t** upisuje se adresa na kojoj se nalazi objekat. U memoriju za objekat **t** upisuje se adresa na kojoj se nalazi objekat **s**. Na taj način, objekti **t** i **s** sada sadrže iste podatke, a podaci koje je sadržao objekat **s** (slika 3) postaju "otpad" je ih više ne može da koristi objekat

s, a ni jedan drugi objekat, jer njihva adresa u memoriji nije povezana više ni sa jednim objektom (slika 4).



Slika 2.2.4 Primer izjednačavanja dva objekata koji se nalaze u memoriji

## PRIMER 11

### Primer smeštaja u memoriju objekta tipa Sting

Zadatak:

1. Kreiraju se dva stringa sa referencama: prvi i drugi;
2. Dodeliti konkretne stringove ovim referencama;
3. Štampati prvi string, drugi strina i string nastao spajanjem prvoj i drugog stringa;
4. Prepisati referencu drugi u prvi;
5. Štampati prvi string, drugi strina i string nastao spajanjem prvoj i drugog stringa;

Sledećom slikom je prikazan očekivani izlaz.

```
run:
Prvi string = String broj 1
Drugi string = String broj 2
Spojeni stringovi = String broj 1 String broj 2
-----
Prvi string = String broj 2
Drugi string = String broj 2
Spojeni stringovi = String broj 2 String broj 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Slika 2.2.5 Očekivani izlaz iz programa

```
package l02;
```

```
/**
```

```
* *
* @author Vladimir Milićević
*/
public class RadSaStingovima {

    public static void main(String[] args) {
        /*kreiranje Sting referenci kojima su pridružena dva stinga*/
        String prvi = "String broj 1";
        String drugi = "String broj 2";
        //Prva grupa rezultata
        System.out.println("Prvi string = " + prvi);
        System.out.println("Drugi string = " + drugi);
        System.out.println("Spojeni stringovi = " + prvi + " " + drugi);
        //prepisivanje jedne u drugu ref. prvi i drugi ukazuju na istu lokaciju u
        memoriji
        prvi=drugi;
        //Prva grupa rezultata
        System.out.println("-----");
        System.out.println("Prvi string = " + prvi);
        System.out.println("Drugi string = " + drugi);
        System.out.println("Spojeni stringovi = " + prvi + " " + drugi);
    }
}
```

## ZADATAK 11

### *Samostalno vežbanje smeštaja u memoriju objekta tipa Sting*

- Koristite primer Primer11;
- Prepravite kod tako da postoji i treća Sting referenca pod nazivom pomocniString;
- Zapamtite sadržaj memorijske lokacije na koju ukazuje prvi pomoću pomocniString;
- Nakon izvršenja druge grupe rezultata vratite originalnu vrednost u prvi;
- Kreirajte po analogiji treću grupu rezultata.

## ▼ 2.3 Omotačke klase

### ŠTA JE OMOTAČKA KLASA?

*Omotačka klasa sadrži “umotan” primitivni tip i time ih pretvara u objekte i omogućava da se nad njima sprovedu operacije kao i nad drugim objektima.*

Osam primitivnih tipova podataka u Javi nisu klase, a ni vrednosti primitivnih tipova nisu objekti. Primitivni tipovi nemaju ugrađene funkcionalnosti, na primer: ne možemo konvertovati ceo broj u string. Ponekad je potrebno premostiti taj jaz, tj. tretirati primitivne vrednosti kao da su to objekti. To se rešava tako što vrednost primitivnog tipa možemo "umotati" u objekat odgovarajuće **omotačke klase**(engl. **wrapper**).

Za svaki od osam primitivnih tipova postoji odgovarajuća omotačka klasa:

- **Byte,**
- **Short,**
- **Integer,**
- **Long,**
- **Float,**
- **Double,**
- **Characteri**
- **Boolean.**

Primitivni tip	Omotačka klasa
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

Slika 2.3.1 Primitivni tipovi i omotačke klase

## PRIMENA OMOTAČKIH KLASA

*Namena omotačkih klasa se ogleda u tome što obuhvataju neke korisne statičke konstante imetode koji olakšavaju rad sa primitivnim vrednostima*

Namena omotačkih klasa se ogleda u tome što obuhvataju neke korisne **statičke konstante** i **metode** koji olakšavaju rad sa primitivnim vrednostima, a pored toga za konstruisanje objekata koji predstavljaju vrednosti primitivnih tipova.

Kako promenljiva primitivnog tipa nije objekat, za nju ne možemo pozvati metodu neke klase. Dakle, ako pozovemo statičku metodu i njoj prosledimo promenljivu primitivnog tipa, rešićemo ovaj problem.

Klasa **Integer**, na primer, sadrži konstante:

**MIN\_VALUE** i **MAX\_VALUE**

koje su jednake minimalnoj i maksimalnoj vrednosti tipa **int**. Ovo je korisno jer je svakako lakše zapamtiti imena nego numeričke vrednosti.

Iste konstante se nalaze i u klasama

### Byte, Short, Long, Float, Double i Character.

#### PRIMER:

Dat je niz od  $n$  elemenata i treba pronaći minimalni element. Predpostavićemo da je maksimalni element niza jednak najmanjoj mogućoj vrednosti (**Integer.MIN\_VALUE**), pa ćemo sve naredne elemente porediti sa ovim trenutnim maksimalnim elementom niza. U slučaju da je  $i$ -ti element niza veći od trenutnog maksimalnog elementa, maksimalni element će dobiti vrednost  $i$ -tog elementa.

Klasa **Double** sadrži i neke konstante koje ne predstavljaju realne brojeve u matematičkom smislu: Konstanta **POSITIVE\_INFINITY** označava beskonačnu vrednost koja se može dobiti kao rezultat nekih izraza.

Na primer, deljenje pozitivnog broja sa nulom ili proizvod  $10200 * 10200$  koji prevazilazi vrednost **MAX\_VALUE**, u Javi su definisani i kao rezultat daju **POSITIVE\_INFINITY**. Slično važi i za konstantu **NEGATIVE\_INFINITY**.

Konstanta **NaN** (engl. *not a number*) predstavlja nedefinisanu vrednost koja se dobija, recimo, izračunavanjem kvadratnog korena od negativnog broja ili deljenjem nule sa nulom.

## PRIMER 12 - UPOTREBA OMOTAČKE KLASSE

*Omotačka klasa omogućava pretvaranje stringa u primitivni tip i obrnuto.*

Formirati omotačku klasu da bi se izvršila:

- proveriti da li je rezultat nekog izraza u validnom opsegu
- proveriti da li je izvršeno deljenje nulom
- proveriti da li je rezultat nekog izraza validan broj

Klase omotači sadrže i neke statičke metode za rad sa odgovarajućim tipovima podataka.

Klasa **Integer** sadrži metod **parseInt()** koji pretvara string u vrednost tipa **int**.

Na primer:

```
Integer.parseInt("17")
```

kao rezultat daje ceo broj 17 tipa **int**.

Napomena: Ako argument metoda **parseInt()** nije string koji predstavlja ceo broj, u programu nastaje greška tipa izuzetak pod nazivom **NumberFormatException**.

Slično, klasa **Double** sadrži metod **parseDouble()** koji pretvara string u vrednost tipa **double**. Na primer, iskaz:

```
Double.parseDouble("2.61e1")
```

kao rezultat daje realni broj 26.1 tipa **double**.

Klasa **Integer** sadrži i metode

- *Integer.toBinaryString(i)*,
- *Integer.toHexString(i)*, i
- *Integer.toOctalString(i)*

vraćaju broj *i* kao string u binarnoj, heksadecimalnoj i oktalnoj reprezentaciji. Metoda *Integer.reverse()* vraća sa obrnutim rasporedom cifara.

Klasa **Character** sadrži, između ostalog, veliki broj statičkih metoda kojima se može ispitivati dati znak tipa **char**. Na primer, metodi *isLetter()*, *isLetterOrDigit()*, *isLowerCase()* i tako dalje kao rezultat daju logičku vrednost tačno ili netačno prema tome da li dati argument tipa **char** predstavlja znak koji je slovo, slovo ili cifra, malo slovo i tako dalje.

Ove metode možemo i sami implementirati, ali ipak treba znati da one postoje i kao ugrađene metode u omotačke klase.

## ZADATAK 12 - UPOTREBA OMOTAČKE KLASA

### *Samostalno vežbanje upotrebe omotačkih klasa*

- Kreirati stringove prvi, drugi i treći;
- Upisati u stringove redom vrednosti: "100"; "12.56" i "-11.8" redom.
- Primenom omotačkih klasa konvertovati stringove u odgovarajuće numeričke vrednosti;
- Izvršiti sabiranje i prikazati rezultat;
- Izvršiti spajanje stringova (operatorom +);
- U čemu se razlikuju rezultati primene operatora "+" u ova dva različita slučaja?

## ŠTA JE KLASA OMOTAČ? (VIDEO)

*Video daje animaciju koja objašnjava kako klase predstavljaju omotače za primitivne tipove podataka*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 2.4 Prikazivanje podataka

### METODE PRINT() I PRINTLN()

*Objekat `System.out` sadrži metode za prikazivanje vrednosti na ekranu, kao što su `print()` i `println()`*

Klase mogu imati kako **statičke metode**, tako i **statičke promenljive** (atribute, polja).

Klasa **System** sadrži statički atribut **out**, koji je zapravo objekat klase **PrintStream**, o kojoj ćemo govoriti kasnije. Statički atribut **out** klase **System** pozivamo sa: `System.out`. Statički atribut `System.out` ukazuje na objekat koji u Javi predstavlja **standardni izlaz**. **Standardni izlaz** je apstraktni model prikazivanja raznih tipova podataka na fizičkom uređaju ekrana. Analogno tome, postoji objekat **System.in** koji predstavlja **standardni ulaz** i o njemu ćemo govoriti kasnije. Objekat **System.out** sadrži metode za prikazivanje vrednosti na ekranu.

Jedna od njih je metod **print()** koji služi za prikazivanje podataka na ekranu. Složeno ime `System.out.print()` se dakle odnosi na metod **print()** u objektu na koga ukazuje statički atribut **out** u klasi **System**.

Potpuno isto važi i za drugi metod **println()**: složeno ime `System.out.println()` se odnosi na metod **println()** u objektu na koga ukazuje statički atribut **out** u klasi **System**.

### PRIMER KORIŠĆENJE NAREDBE PRINT I PRINTLN

*Metod `println()` prikazuje tekst u sledećoj liniji, a metod `print()` u istoj liniji.*

U Javi se podaci na ekranu najčešće prikazuju na sledeći način:

```
System.out.print(...);  
System.out.println(...);
```

Podaci koji se žele prikazati na ekranu se navode kao argumenti u zagradama umesto tri tačke, a uprintln() slučaju se nakon njihovog prikazivanja prelazi u sledeći red na ekranu.

Na primer, naredbama:

```
System.out.print("Zdravo ! ");  
System.out.print("Dobrodošli u svet programiranja! ");
```

Prikazuje se tekst:

"Zdravo ! Dobrodošli u svet programiranja!"

dok se naredbama:



```
System.out.println("Zdravo ! ");  
System.out.println("Dobrodošli u svet programiranja! ");
```

Prikazuje se tekst:

“Zdravo !

Dobrodošli u svet programiranja!

Argument ovih metoda, odnosno vrednosti koje se žele prikazati na ekranu, u programu se navodi u zagradama. Zagrade se pri pozivanju nekog metoda uvek moraju pisati, čak i ako metod nema argumenata.

Na primer:

```
System.out.println();
```

ukoliko samo želimo da pređemo u sledeći red, tj. da kursor pomerimo za jedan red.

Ako se iza složenog imena nalazi leva zagrada, onda ono predstavlja metod; u suprotnom slučaju, ono predstavlja promenljivu.

## METOD PRINTF()

*Metod printf() omogućava prikaz u željenom formatu.*

Jedan problem sa metodima **print()** i **println()** je to što su prikazani brojevi ponekad u formatu koji nije pregledan. Na primer, izvršavanjem naredbi

```
double prosekOcena = 89.0 / 9;  
System.out.println("Prosek ocena je: " + prosekOcena);
```

na ekranu bismo dobili sledeći rezultat:

*Prosek ocena je: 9.888888888888889*

Pošto je promenljiva *prosekOcena* tipa double, njena vrednost se izračunava sa 15-16 decimala i sve se one prikazuju na ekranu.

Naravno, u većini slučajeva toliko broj decimala u izlaznim podacima samo smeta, pa je potrebno na neki način imati veću kontrolu nad formatom prikazanih brojeva. Zato je od verzije Java 5 dodat metod printf() koji ima slične mogućnosti kao ista funkcija u programskom jeziku C. Broj opcija metoda System.out.printf() za formatizovanje podataka je vrlo velik, ali ćemo ovde pomenuti samo nekoliko. Metoda *System.out.printf()* može imati jedan argumenat ili više njih razdvojenih zapetama. Prvi argument je formatirani string (objekat tipa String) kojim se određuje format izlaznih podataka. Preostali argumenti predstavlja>ju vrednosti koje se prikazuju.

Opšti oblik je:

```
System.out.printf(formatiraniString, argument1, argument2...argumentN);
```

Ako bismo prethodni primer napisali koristeći metod **printf()** umesto **println()**:

```
double prosekOcena = 89.0 / 9;
System.out.printf("Prosek ocena je: %5.2f\n", prosekOcena);
```

na ekranu bismo dobili sledeći rezultat:

*Prosek ocena je: 9.89*

Željeni izlazni format neke vrednosti se navodi u prvom argumentu metoda **printf()** zapisom koji počinje znakom procenta (%), završava se određenim slovom, a između mogu biti još neke informacije. Slovo na kraju ukazuje na tip podatka koji se prikazuje, pri čemu se **d koristi za cele brojeve, f za realne brojeve, s za stringove** i tako dalje. Između početnog znaka % i slova na kraju može se navesti minimalan broj mesta za ispis podataka.

Na primer, razlika između **%d** i **%8d** je to što se u prvom slučaju prikazuje ceo broj sa onoliko cifara koliko ih ima (računajući i znak minus za negativne brojeve), dok se u drugom slučaju koristi tačno osam mesta.

U ovom drugom slučaju, ako broj ima manje od osam cifara, dodaju se prazna mesta ispred cifara broja kako bi se popunilo svih osam mesta; ako broj ima više od osam cifara, koristi se tačno onoliko mesta koliko broj zapravo ima cifara.

## PRIMER KORIŠĆENJA NAREDBE PRINTF

***%5.2f prikazuje realan broj sa 5 mesta i dve decimale.***

U sledećem konkretnom primeru, izvršavanjem naredbi:

```
int n = 230;
System.out.printf("Broj: %d\n", n);
System.out.printf("Broj: %8d\n", n);
```

na ekranu dobijamo:

Broj: 230

Broj: 230

Primetimo da se u drugom slučaju dodaju pet praznih mesta (raz¬maka) ispred broja 230 kako bi se on prikazao u polju dužine osam mesta. Pored dela koji počinje znakom % za opis formata podataka, prvi argument metoda printf() može sadržati druge znakove (uključujući specijalne znakove koji počinju sa \, recimo za prelazak u novi red - \n). Ovi znakovi se na ekranu prikazuju neizmenjeni i mogu poslužiti da se prikaže proizvoljni tekst koji bliže opisuje podatke.

Sledeće naredbe, na primer:

```
int n = 230;
System.out.printf("Kvadrat broja %d je %8d\n", n, n * n);
```

na ekranu proizvode prikaz teksta:

Kvadrat broja 230 je 52900

U ovom primeru, format **%d** se odnosi na vrednost drugog argumenta metoda **printf()** (promenljiva *n*), a format **%8d** se odnosi na vrednost trećeg argumenta (izraz *n\*n*). Ovaj primer takođe pokazuje da argumenti metoda **printf()** mogu biti proizvoljni izrazi.

Za prikazivanje realnih brojeva se dodatno može navesti željeni broj decimala prikazanog broja. Ovu mogućnost smo koristili u naredbi

```
System.out.println("Prosek ocena je: %5.2f\n", prosekOcena);
```

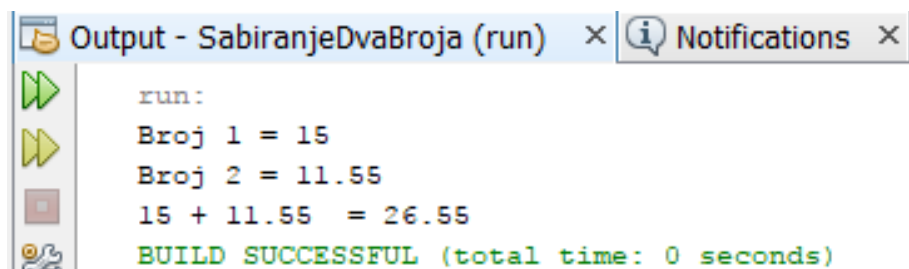
gde smo vrednost promenljive *prosekOcena* prikazali u formatu **%5.2f**. Deo između **%** i slova **f** se u opštem slučaju sastoji od ukupnog broja mesta i broja decimala razdvojenih tačkom. Tako **5.2** znači da se realan broj koji je sadržaj promenljive *prosekOcena* prikazuje u polju od pet mesta sa dve decimale.

Pomenimo još slovo **s** koje se može koristiti za bilo koji tip podataka. Tim slovom se neka vrednost prikazuje u svom podrazumevanom obliku. Na primer, **%10s** znači da se neka vrednost prikazuje u polju od (minimalno) deset mesta.

## PRIMER 13

### *Vežbanje upotrebe naredbi print, println i printf*

Korišćenjem naredbi za prikazivanje rezultata kreirati program koji daje sledeći izlaz.



Slika 2.4.1 Formatiran izlaz primenom naredbe printf

U primeru dodatno vežbamo i povezivanje stringa sa drugim tipom podataka u naredbi **println** (linije 18 i 19). Posebno je primenjena naredba **printf** kojom je formatiran izlaz koji je prikazan na konzoli. Detalj **%d** ukazuje da je na tom mestu predviđeno prikazivanje celog broja. Detalj **%4.2f** rezervoar 4 mesta, od toga 2 za decimalni deo, za prikazivanje realnog broja.

```
package l02;

/**
 * *
 * @author Vladimir Milićević
 */
public class PrikazivanjePodataka {
```

```
public static void main(String[] args) {  
    /*kreiranje Sting referenci kojima su pridružena dva stinga*/  
    String str1 = "Broj 1 =";  
    String str2 = "Broj 2 =";  
    //Prva grupa rezultata  
    int prvi = 15;  
    double drugi = 11.55;  
    System.out.println(str1 + prvi);  
    System.out.println(str2 + drugi);  
    System.out.printf("%d + %4.2f = %4.2f \n",prvi,drugi,prvi+drugi );  
  
}  
}
```

## ZADATAK 13

### *Samostalno vežbanje prikazivanja podataka*

Pomoću naredbe printf formatirajte i prikažite rezultate sledećeg izraza, identično kao u prethodnom primeru:

- $(11+12.8899)/35.999$  - celobrojni deo neka zauzima 2 i decimalni 2 mesta za svaki od realnih brojeva prilikom prikazivanja rezultata;

UPUTSTVO:

Izlaz bi trebalo da izgleda ovako:

Broj 1 = 11

Broj 2 = 12.8899

Broj 3 = 35.999

$(11 + 12.88) / 35.99 = 0,66$

## ▼ 2.5 Unos podataka

### UNOS PODATAKA METODAMA KLASE SCANNER

#### *Klasa Scanner omogućava unos podataka u Java aplikaciju.*

U verzijama programskog jezika Java pre verzije 5 se učitavanje vrednosti osnovnih tipova podataka sa standardnog ulaza zasnivalo na kompleksnom pristupu koji je sadržao napredije elemente objektno-orijentisanog programiranja, pa je zato od verzije Java 5 u paketu *java.util* dodata klasa **Scanner** koja olakšava učitavanje ulaznih podataka od korisnika.

Učitavanje vrednosti u Java programu ipak nije tako jednostavno kao njihovo prikazivanje, za primenu ugrađenih metoda klase **Scanner** koje će nam omogućiti učitavanje podataka od korisnika je neophodno prvo kreirati objekat tipa **Scanner**.

## KREIRANJE OBJEKTA KLASSE SCANNER

### *Za kreiranje objekta bilo koje klase služi operator **new** Scanner*

Za kreiranje objekta bilo koje klase služi operator **new** koji ima opšti oblik:

```
new Klasa()
```

gde je **Klasa()** poziv specijalnog metoda koji se naziva **konstruktor klase** čije je ime **Klasa**.

Konstruktor je specijalni metod neke klase kojim se vrši konstruisanje objekta te klase u memoriji i njihova inicijalizacija.

Ako operator **new** primenimo za kreiranje objekta klase **Scanner**, možemo pisati recimo:

```
Scanner ulaz = new Scanner(System.in);
```

Ovom naredbom dodele se kreira objekat klase **Scanner** pod nazivom **ulaz**. Argument konstruktora klase **Scanner** naveden u zagradama jeste objekat **System.in**. Time se kreirani objekat klase **Scanner** povezuje sa standardnim ulazom kao izvorom podataka koji će se čitati, tj. objektom u klasi **System**, baš kao što je standardni izlaz predstavljen objektom **out** u klasi **System**. Na sličan način se objekat klase **Scanner** može povezati sa nekom datotekom,

Klasa **Scanner** sadrži metode koji se mogu primeniti na objekat te klase radi učitavanja podataka različitog tipa.

Kad jednom kreiramo objekat klase **Scanner**, možemo ga pozivati onoliko puta koliko nam je potrebno.

Ako definišemo objekat **ulaz** na sledeći način:

```
Scanner ulaz = new Scanner(System.in);
```

Možemo pozivati sledeće metode ovog objekta:

```
ulaz.nextBoolean(), ulaz.nextByte(), ulaz.nextInt(), ulaz.nextLong(),  
ulaz.nextFloat(), ulaz.nextDouble()
```

Ove metode služe za učitavanje vrednosti preko tastature tipa **boolean**, **byte**, **int**, **long**, **float** i **double** i svaka metoda vraća konkretan tip (**boolean**, **byte**, **int**, **long**, **float** i **double**)

```
ulaz.next()
```

Učitava se niz znakova preko tastature do prve beline (znaka razmaka, tabulatora ili novog reda) i vraća se kao vrednost tipa **String**

```
ulaz.nextLine()
```

Učitava se niz znakova preko tastature do kraja reda (znaka novog reda) i vraća se kao vrednost tipa `String`.

## PRIMER 14 - UNOS DVA CELA BROJA

*U klasi `Scanner` su definisani i metodi kojima se može proveravati to da li su ulazni podaci dostupni ili da li su oni određenog tipa*

Sledeći primer prikazuje učitavanje dva cela broja sa standardnog ulaza i štampanje njihovog zbira:

```
public class PrimerUlaza {
    public static void main(String[] args) {
        Scanner ulaz = new Scanner(System.in);
        System.out.println("Unesi prvi broj: ");
        int broj1 = ulaz.nextInt();
        System.out.println("Unesi drugi broj: ");
        int broj2 = ulaz.nextInt();
        int suma = broj1 + broj2;
        System.out.println("Suma je: " + suma);
    }
}
```

Pri čemu se, na primer, naredbom:

```
int broj1 = ulaz.nextInt();
```

učitava ceo broj preko tastature kao tip `int` i dodeljuje celobrojnoj promenljivoj `broj1`.

U klasi `Scanner` su definisani i metodi kojima se može proveravati to da li su ulazni podaci dostupni ili da li su oni **određenog** tipa:

```
ulaz.hasNext()
```

Vraća logičku vrednost tipa `boolean`, i to `true` (tačno) ako je neki ulazni niz znakova, različit od znakova beline, raspoloživ preko tastature.

```
ulaz.hasNextBoolean(), ulaz.hasNextByte(), ulaz.hasNextInt(), ulaz.hasNextLong(),
ulaz.hasNextFloat(), ulaz.hasNextDouble()
```

Vraća logičku vrednost tipa `boolean`, i to `true` (tačno) ako je odgovarajuća vrednost tipa `boolean`, `byte`, `int`, `long`, `float` i `double` raspoloživa preko tastature.

```
ulaz.hasNextLine()
```

Vraća logičku vrednost tipa `boolean`, i to `true` (tačno) ako je ulazni niz znakova raspoloživ preko tastature do kraja reda.

Ovim smo obradili samo osnovne mogućnosti klase **Scanner** za učitavanje podataka preko tastature, s obzirom da je ova klasa je mnogo opštija i može poslužiti za učitavanje podataka i iz drugih izvora podataka kao što su datoteke, mrežne konekcije ili čak stringovi.

## ZADATAK 14

### *Samostalno vežbanje unosa sa tastature*

Modifikujte primer "Primer 13" tako da se vrednosti za brojeve prvi i drugi unose sa tastature. Pokušajte isto tako da realizujete i zadatak "Zadatak 13".

## UNOS PODATAKA PRIMENOM KLASSE SCANNER (VIDEO)

*Video daje primer korišćenja klase Scanner radi unosa podatak u program sa konyole*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ 2.6 Paketi u Javi

### POJAM PAKETA

*Paket u programskom jeziku Java je kolekcija klasa koje su namenjene jednoj vrsti posla i koje zato čine funkcionalnu celinu*

Java platforma sadrži veliki broj unapred napisanih klasa koje se mogu koristiti u programima. Da bi se olakšalo pronalaženje i upotreba potrebnih klasa, sve klase Java platforme su grupisane u **pakete**, slično organizaciji datoteka i foldera.

**Paket** u programskom jeziku Java je kolekcija klasa koje su namenjene jednoj vrsti posla i koje zato čine funkcionalnu celinu. Neki od osnovnih paketa čije klase ćemo koristiti su:

- *java.lang*
- *java.util*
- *java.math*
- *java.awt*
- *java.io*

Prilikom definisanja nove klase u Javi se mogu koristiti druge klase samo iz istog paketa. Klase iz drugih paketa se mogu koristiti uz navođenje punog imena klase koje se sastoji iz dva dela: imena paketa i imena klase. Na primer, za rad sa datumima postoji klasa **Date** koja se nalazi u paketu *java.util*, ako je potrebna, klasa se može upotrebiti pod imenom *java.util.Date*

S obzirom da upotreba punog imena klase zahteva više pisanja, postoji mogućnost "uvoženja" pojedinih klasa na početku programa, čime bi klasu mogli da upotrebimo upotrebom samo imena klase, bez naziva paketa.

Ova mogućnost se postiže pisanjem deklaracije **import** na početku teksta klase, na primer, ako želimo da definišemo klasu **Student** koja koristi objekat **datumUpisa** klase **Date**, umesto da pišemo:

```
class Student{
    ...
    java.util.Date datumUpisa;
    ...
}
```

možemo kraće pisati:

```
import java.util.Date;
class Student{
    ...
    Date datumUpisa;
    ...
}
```

## PRIMER 15 - KORIŠĆENJA PAKETA

*Deklaracijom package definišemo paket klase koja se pojavljuje u radnu memoriju programa iskazom import*

Ako želimo da uvezemo više klasa iz istog paketa možemo upotrebiti znak \* čime se uvoze sve klase navedenog paketa. Prethodni primer možemo napisati na sledeći način:

```
import java.util.*;
class Student{
    ...
    Date datumUpisa;
    ...
}
```

možemo kraće pisati:

```
import java.util.Date;
class Student{
    ...
    Date datumUpisa;
    ...
}
```

Osnovni paket java.lang automatski uvozi klase koje se nalaze u njemu, pa se ne moraju uvoziti, recimo klasa String.



Svaka klasa u Javi mora pripadati nekom paketu, ako prilikom definisanja klase ne navedemo paket ona će pripadati podrazumevanom paketu bez imena (anonimni ili default paket).

Ako želimo da klasu dodamo imenovanom paketu, koristimo deklaraciju `package` kojom se definiše paket kome klasa pripada. Ova deklaracija se navodi na samom početku klase, pre deklaracije `import`. Na primer, ako želimo da klasa `Student` bude deo paketa `l02.prvipaket`, to definišemo na sledeći način:

```
package l02.prvipaket;

import java.util.*;
class Student{
    ...
    Date datumUpisa;
    ...
}
```

## ZADATAK 15

### *Vežbanje korišćenja paketa*

Za izradu zadatka uzor može biti primer sa linka: <https://www.javatpoint.com/package>

- Kreirajte klasu `MojiPodaci` u paketu `l02.prvaklasa`;
- Klasa sadrži metodu `pisi()` koja daje za rezultat sledeći String "Pozdravlja Vas Ime i Prezime" - o metodama će više biti reči kasnije. Metoda `pisi()` u ovom obliku je već korišćena u prethodnim primerima;
- Kreirati klasu `PrikazPodataka` koja pripada drugom paketu, npr. `kratkiprogrami.l02`;
- Importovati paket sa prvom klasom u drugu klasu;
- Kreirati u klasi `PrikazPodataka` objekat klase `MojiPodaci`;
- Pozvati ovim objektom metodu `pisi()`;
- Pokrenuti program i prikazati rezultate;

## ▼ Poglavlje 3

# Operatori u Javi

## ŠTA SU OPERATORI?

*Operatori predstavljaju operacije koje se izvršavaju nad operandima dajući određeni rezultat*

**Operatori** predstavljaju operacije koje se izvršavaju nad operandima dajući pri čemu daju određeni rezultat. **Izraz** (engl. expression) predstavlja proizvoljno složen niz operanada i operatora. Najjednostavniji izraz se sastoji od jednog literala ili promenljive.

Primer izraza:

```
"CS101"  
    ulaz
```

Složeniji izrazi se dobijaju primenom operatora na operandom, tj. podatkom.

Operatori mogu da se primenjuju na jedan operand (unarni operatori), dva operanda (binarni operatori) ili tri operanda (ternarni operatori).

**Unarni operatori** mogu da stoje ispred operanda (prefiksni operatori) ili iza operanda (postfiksni operatori).

Primer izraza:

`++brojac`

`suma--`

`(int) 5.66`

**Binarni operatori** uvek stoje između svoja dva operanda (infiksni operatori).

Primer izraza:

`5 + 7`

`10 * 14.23`

Izraz po potrebi i po želji može da se stavlja unutar para obliha zagrada (). Njihova uloga je da izdvoje deo složenog izraza kao neku manju celinu i da time utiču na redosled izračunavanja operatora.

Primer izraza:

`5 * (10 + 12 * 9)`

# TERNARNI OPERATORI

*Ternarni operator radi sa tri operanda.*

U jeziku Java postoji i jedan operator s tri operanda (**ternarni operator**).

Upotreba ternarnog operatora **?:** označava se stavljanjem znaka pitanja između prvog i drugog operanda i dve tačke između drugog i trećeg operanda.

Primer izraza:

```
a > b ? a : b
```

O ternarnom operatoru ćemo više govoriti u narednom predavanju.

Uopšteno govoreći, rezultat izračunavanja može biti:

- promenljiva
- vrednost,
- ništa – označava se sa void

Redosled izvršavanja operatora prvenstveno se određuje oblim zagradama ( **()** ). Unutar para oblih zagrada mogu da budu ugnežđeni drugi parovi zagrada do proizvoljne dubine. Operatori unutar para zagrada izvršavaju se pre operatora izvan tih zagrada.

Operatori unutar zagrada izvršavaju se po redosledu prioriteta. Od dva susedna operatora, operator s višim prioritetom izvršava se pre operatora s nižim prioritetom. U jeziku Java postoje ukupno 16 nivoa prioriteta - najviši prioritet ima nivo 16, a najniži nivo 1. U slučaju niza operatora međusobno jednakih prioriteta, operatori će biti izvršavani sleva nadesno ili zdesna nalevo, u zavisnosti od njihovog smeru grupisanja.

O prioritetu operatora ćemo detaljnije govoriti u narednom predavanju.

## ▼ 3.1 Aritmetički operatori

### VRSTE ARITMETIČKIH OPERATORA

*Osnovni aritmetički operatori u Javi odgovaraju matematičkim operacijama sabiranja, oduzimanja, množenja i deljenja bio kog numeričkog tipa podataka.*

Osnovni aritmetički operatori u Javi odgovaraju matematičkim operacijama sabiranja, oduzimanja, množenja i deljenja sa oznakama **+**, **-**, **\*** i **/** i mogu primenjivati na vrednosti bilo kog numeričkog tipa (**byte**, **short**, **int**, **long**, **float** i **double**). Prilikom izvođenja samih operacija obe vrednosti moraju biti istog tipa. O aritmetičkom operatoru **%** ćemo govoriti u narednom predavanju.

Operator	Opis	Primena	Primer
*	Množenje	izraz1 * izraz2	2 * 3 (6) 3.3 * 1.0 (3.3)
/	Deljenje	izraz1 / izraz2	1 / 2 (0) 1.0 / 2.0 (0.5)
%	Ostatak (Deljenje po modulu)	izraz1 % izraz2	5 % 2 (1)
+	Sabiranje (ili unarni +)	izraz1 + izraz2 +izraz1	7 + 2 (9) 7.2 + 2.0 (9.2) +(-5) (-5)
-	Oduzimanje (ili unarni -)	izraz1 - izraz2 -izraz1	9 - 7 (2) -5.5 (-5.5)

Slika 3.1.1 Aritmetički operatori

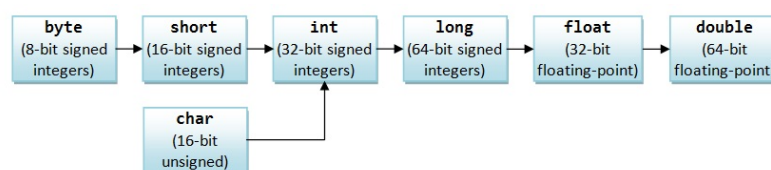
## PRIMERI KORIŠĆENJA 1

*Konverzija tipa se automatski vrši prilikom izračunavanja izraza.*

Kod izračunavanja sledećeg izraza:

27.16 + 3

se najpre ceo broj 3 pretvara u ekvivalentni realni broj 3.0, a zatim se izračunava 27.16 + 3.0. Ovo se naziva konverzija tipa i predstavlja pretvaranje vrednosti "manjeg" tipa u ekvivalentnu vrednost "većeg" tipa. Konverzija tipa se automatski vrši prilikom izračunavanja izraza.



Slika 3.1.2 Konverzija tipa

Kada se jedan od osnovnih aritmetičkih operatora primenjuje na dve vrednosti istog tipa (nakon eventualne konverzije tipa), dobija se i rezultat istog tipa:

Ako se množe dva cela broja tipa int, rezultat je ceo broj tipa int; ako se sabiraju dva realna broja tipa double, rezultat je realni broj tipa double. Ovo je prirodno pravilo, osim u slučaju operatora deljenja /. Kada se dele dva cela broja, rezultat toga je opet ceo broj i eventualne decimale rezultata se odbacuju. Na primer, izraz:

7 / 2

kao rezultat daje ceo broj 3, a ne realni broj 3.5.

Slično, ako je **n** celobrojna promenljiva tipa **int**, tada je i **1 / n** ceo broj **int**. Tako, ako **n** ima vrednost veću od 1, rezultat izračunavanja **1 / n** je 0. Tačna vrednost celobrojnog deljenja se može dobiti na dva načina. Prvi način je da se jedan od operandi napiše kao realan broj. Na primer, kod izračunavanja izraza:

```
(3 + 2) / 2.0
```

zbog konverzije tipa se najpre levi operand pretvara u realni broj, pa se kao rezultat dobija tačan realni broj. To znači da se rezultat sabiranja  $3 + 2$ , tj. broj 5 pretvara u realni broj 5.0, pa stoga izraz  $5.0 / 2.0$  daje tačan rezultat 2.5.

Drugi način za dobijanje tačnog rezultata celobrojnog deljenja je korišćenje operatora eksplicitne konverzije tipa (engl. type cast). Oznaka tog operatora u Javi je ime odgovarajućeg tipa podataka u zagradama, što se piše ispred vrednosti koju želimo da pretvorimo u navedeni tip. Na primer, ako su **n** i **m** celobrojne promenljive tipa **int**, u izrazu:

```
(double) n / m
```

zahteva se pretvaranje vrednosti promenljive **n** u tip **double** pre njenog deljenja sa vrednošću promenljive **m**, tako da se na kraju dobija tačan rezultat deljenja tipa **double**.

## PRIMERI KORIŠĆENJA 2

*EksPLICITNA KONVERZIJA tipa se može vršiti i između znakovnog tipa **char** i celobrojnog tipa **int***

Date su vrednosti zbira elemenata niza i broja elemenata niza, ispisati aritmetičku sredinu elemenata niza

```
int zbirElemenataNiza = 111;
int brojElemenataNiza = 10;
double aritmetickaSredinaNiza = (double) zbirElemenataNiza /
    brojElemenataNiza;
System.out.println("Aritmetička sredina je: " + aritmetickaSredinaNiza);
```

EksPLICITNA konverzija tipa se može vršiti i između znakovnog tipa **char** i celobrojnog tipa **int**, s obzirom da znakovni tip **char** ima ujedno i numeričku vrednost.

Numerička vrednost nekog znaka je njegov Unicode kodni broj tako da izraz

```
(int) '+'
```

daje ceo broj 43 i sledeći izraz:

```
(char) 97
```

daje znak 'a'.

## ARITMETIČKI OPERATORI (VIDEO)

### *Video objašnjava primenu aritmetičkih operatora u Javi*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## PRIMER 16

### *Odrediti nove numeričke vrednosti promenljive, za dati procenat od početne vrednosti.*

*Napisati program koji na određeni broj dodaje određeni procenat tog broja. Na primer ukoliko imamobroj 100 i procenat 20%, rezultat treba da bude 120. Unos cifre treba uraditi **korišćenjem grafičkog dijaloga**. Program treba da bude deo NetBeans projekta pod nazivom **KI103-V02** i deo paketa **cs101.v02**. Naziv klase treba da bude **Zadatak6**. Za učitavanje podataka od korisnika treba koristiti metodu **showInputDialog()** klase **JOptionPane**, a za ispis podataka objekat **System.out**. Za parsiranje stringova u brojeve koristiti adekvatnu omotačku klasu.*

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cs101.v02;

import javax.swing.JOptionPane;

/**
 *
 * @author Jovana
 */
public class Zadatak6 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak6();
    }

    public Zadatak6() {
        double subtotal = Double.parseDouble(JOptionPane.showInputDialog(null,
"Unesite pocetnu cifru: "));
        double gratuityPercent =
Double.parseDouble(JOptionPane.showInputDialog(null, "Unesite procenat uvećanja:
"));
    }
}
```

```

        double gratuity = subtotal * gratuityPercent / 100;
        double total = subtotal + gratuity;
        System.out.println("Uvećanje: " + gratuity);
        System.out.println("Ukupno: " + total);

    }

}

```

## PRIMER 17

### *Proračun energije zagrevanja vode od početne do željene temperature.*

*Napisati program koji računa ukupnu energiju potrebnu da se zagreje voda od početne do finalne temperature. Vaš program bi trebalo da od korisnika zahteva da unese količinu vode u kilogramima i da unese početnu i krajnju temperaturu vode korišćenjem grafičkog dijaloga. Ukupna energija se računa kao:*

*energija u džulima = količina vode u kilogramima \* (finalna temperatura - početna temperatura) \* 4184*

*Program treba da bude deo NetBeans projekta pod nazivom KI103-V02 i deo paketa cs101.v02. Naziv klase treba da bude Zadatak8. Za učitavanje podataka od korisnika treba koristiti metodu showInputDialog() klase JOptionPane, a za ispis podataka objekat System.out. Za parsiranje stringova u brojeve koristiti adekvatnu omotačku klasu.*

```

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package cs101.v02;

import javax.swing.JOptionPane;

/**
 *
 * @author Jovana
 */
public class Zadatak8 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak8();
    }

    public Zadatak8() {
        double tezina = Double.parseDouble(JOptionPane.showInputDialog(null,

```

```
"Unesite tezinu vode u kilogramima: ");
    double pocetnaTemperatura =
Double.parseDouble(JOptionPane.showInputDialog(null, "Unesite pocetnu vrednost
temperature: "));
    double krajnjaTemperatura =
Double.parseDouble(JOptionPane.showInputDialog(null, "Unesite konacnu vrednost
temperature: "));
    double energija = tezina * (krajnjaTemperatura - pocetnaTemperatura) * 4184;
    System.out.println("Potrebna energija je: " + energija);
}
}
```

## PRIMER 18

### *Proračun površine i obima kruga*

*Napisati program koji računa obim i površinu kruga na osnovu poluprečnika, double vrednosti učitane sa standardnog ulaza. Rezultat treba da bude zaokružen na dve decimale. Program treba da bude deo NetBeans projekta pod nazivom l02. Rešeni zadaci 02. Naziv klase treba da bude Zadatak18. Za učitavanje podataka od korisnika treba koristiti objekat klase Scanner, a za ispis podataka objekat System.out.*

#### **Objašnjenje:**

Da bismo izbegli korišćenje vrednosti PI kao promenljivu koju definišemo ručno, možemo koristiti Math javinu biblioteku i njenu konstantu PI (Math.PI).

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package l02;

import java.util.Scanner;

/**
 *
 * @author Jovana
 */
public class Zadatak18 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        new Zadatak18();
    }

    public Zadatak3() {
```



```
Scanner ulaz = new Scanner(System.in);
System.out.println("Unesite poluprečnik: ");
double radius = ulaz.nextDouble();
double obim = 2 * radius * Math.PI;
double površina = Math.pow(radius, 2) * Math.PI;
System.out.printf("Obim kruga poluprečnika %.2f iznosi %.2f, a površina %.2f.", radius, obim, površina);

}

}
```

## ZADATAK 16

### *Proračun obima i površine kvadrata.*

#### **Tekst zadatka:**

Napisati program koji računa obim i površinu kvadrata na osnovu unete veličine stranice (ceo broj) sa standardnog ulaza. Program treba da bude deo NetBeans projekta pod nazivom **I02**. Naziv klase treba da bude **Zadatak16**. Za učitavanje podataka od korisnika treba koristiti objekat klase **Scanner**, a za ispis podataka objekat **System.out**.

## PRIMER 19

### *Aritmetička sredina dva uneta broja*

#### **Zadatak:**

Napisati program koji računa aritmetičku sredinu za dva uneta broja (celi brojevi) sa standardnog ulaza. Rezultat treba da bude zaokružen na dve decimale. Program treba da bude deo NetBeans projekta pod nazivom **I02**. Naziv klase treba da bude *Primer19*. Za učitavanje podataka od korisnika treba koristiti objekat klase **Scanner**, a za ispis podataka objekat **System.out**.

#### **Postupak rešavanja:**

Prateći istovetnu proceduru kao kod prethodnog zadatka potrebno je kreirati novog objekta ulaz klase Scanner.

Ispisati poruku korisniku da unese prvi broj, kreirati promenljivu prviBroj koja čuva prvi broj i dodeliti vrednost koja je unesena iz konzole koristeći objekat ulaz klase Scanner i njegovu metodu nextInt(). Isti postupak ponoviti za drugi broj.

Zatim je potrebno kreirati promenljivu aritmetickaSredina tipa double koja čuva aritmetičku sredinu dva broja i dodeliti joj vrednost deljenjem zbira dva broja sa 2.0 ili eksplicitnom konverzijom tipa na sledeći način

```
double aritmetickaSredina = (double) (prviBroj + drugiBroj) / 2;
```

Na kraju je potrebno ispisati poruku korisniku sa izračunatom aritmetičkom sredinom zaokruženom na dve decimale korisniku koristeći metodu printf().

## PRIMER 19 - PROGRAMSKI KOD

### *Programski kod koji određuje aritmetičku sredina dva uneta broja*

```
package l02;

import java.util.Scanner;

public class Primer19 {
    public static void main(String[] args) {
        Scanner ulaz = new Scanner(System.in);
        System.out.println("Unesite prvi broj: ");
        int prviBroj = ulaz.nextInt();
        System.out.println("Unesite drugi broj: ");
        int drugiBroj = ulaz.nextInt();
        double aritmetickaSredina = (prviBroj + drugiBroj) / 2.0;
        System.out.printf("Aritmeticka sredina je: %.2f\n", aritmetickaSredina);
    }
}
```

## ZADATAK 17

### *Zadatak za određivanje aritmetička sredina tri uneta broja*

Napisati program koji računa aritmetičku sredinu i zbir za tri uneta broja (realni brojevi) sa standardnog ulaza. Rezultat treba da bude zaokružen na tri decimale. Program treba da bude deo NetBeans projekta pod nazivom **I02**. Naziv klase treba da bude **Zadatak17**. Za učitavanje podataka od korisnika treba koristiti objekat klase **Scanner**, a za ispit podataka objekat **System.out**.

## PRIMER 20

### *Konverzija dinara u evre.*

#### **Tekst zadatka:**

Napisati program koji računa količinu novca u evrima na osnovu unete količine novca u dinarima (ceo broj) sa standardnog ulaza. Rezultat treba da bude zaokružen na tri decimale. Odnos dinara i evra se računa na osnovu podataka sa sledećeg linka: <http://www.nbs.rs/kursnaListaModul/srednjiKurs.faces?lang=lat> Program treba da bude deo NetBeans projekta pod nazivom **I02.konverzija**. Naziv klase treba da bude **Primer20**. Za učitavanje podataka od korisnika treba koristiti objekat klase **Scanner**, a za ispit podataka objekat **System.out**.

#### **Postupak rešavanja:**

Prateći proceduru iz prethodnih zadataka treba učitati podatak od korisnika i učitanu vrednost smestiti u promenljivu `novacDinari`.

Deljenjem količine novca u dinarima sa tekućim kursom dobijamo količinu novca u evrima

kao realan broj (promenljiva `novacEvri`) koji je potrebno ispisati korisniku zaokruženu na tri decimale koristeći koristeći metodu `printf()`.

## PRIMER 20 - KOD

*Programski kod vrši konverziju dinara u evre u skladu sa opisanim postupkom rešavanja zadatka..*

```
package l02.konverzija;

import java.util.Scanner;

public class Primer20 {
    public static void main(String[] args) {
        Scanner ulaz = new Scanner(System.in);
        System.out.println("Unesite količinu novca u dinarima: ");
        int novacDinari = ulaz.nextInt();
        double novacEvri = novacDinari / 119.5;
        System.out.printf("Količina novca u evrima je: %.3f\n", novacEvri);
    }
}
```

## ZADATAK 18

*Konverzija temperatura izraženih u Celzijusovim stepenima u Farenhajtove stepene.*

### Tekst zadatka:

Napisati program kojim se prevodi celobrojna temperatura iz Celzijusove skale u skalu Farenhajta po sledećoj formuli:

```
tempFarenhajt = tempCelzijus * 1.8 + 32
```

Program treba da bude deo NetBeans projekta pod nazivom `l02.konverzija2`. Naziv klase treba da bude `Zadatak18`. Za učitavanje podataka od korisnika treba koristiti objekat klase `Scanner`, a za ispit podataka objekat `System.out`.

## ✓ Poglavlje 4

### Rešeni zadaci za preuzimanje

#### ZA DODATNU VEŽBU

##### *Pokazna vežba - rešeni zadaci*

U ovoj sekciji možete preuzeti java projekat sa rešenim zadacima iz ove lekcije.

Srećno u radu!!!

## ▼ Poglavlje 5

### Domaći zadaci

#### ZADACI ZA DOMAĆI RAD

*Za ove zadatke se ne daje rešenje i očekuje se da svaki student pokuša samostalno rešavanje istih*

**Zadatak 1.** Napisati program za konverziju temperature zadate u Celzijus stepenima u Farenhajt stepene. Vrednost u Celizijusima pročitati sa konzole kao double vrednost, a zatim je konvertovati u Farenhajte. Rezultat takođe treba biti prikazan na konzoli. Program treba da bude deo NetBeans projekta pod nazivom KI103-Dz1. Naziv klase treba da bude Zadatak 1.

**Zadatak 2.** Napisati program koji od korisnika zahteva da unese minute i na osnovu njih računa broj dana i godina. Pretpostavimo da svaka godina ima 365 dana. Rezultat takođe treba biti prikazan na konzoli. Program treba da bude deo NetBeans projekta pod nazivom KI103-Dz1. Naziv klase treba da bude Zadatak 2.

**Zadatak 3.** Napisati program koji od korisnika zahteva početnu cenu čokolade. Pretpostavimo da cena čokolade svakim danom raste za 2%. Izračunati cenu čokolade nakon 3 dana. Rezultat takođe treba biti prikazan na konzoli. Program treba da bude deo NetBeans projekta pod nazivom KI103-Dz1. Naziv klase treba da bude Zadatak 3.

## ▼ Zaključak

### ZAKLJUČAK

*Rezime šta ste naučili u ovoj lekciji.*

U predavanju smo se upoznali sa:

- Osnovnim elementima i osobinama programskog jezika Java
- Tipovima podataka i njihovom podelom
- Pojmovima promenljivih, konstanti, komentara i naredbi
- Omotačkim klasama i njihovim ugrađenim funkcijama
- Obradom standardnog ulaza i izlaza koristeći klasu Scanner i objekat System.out
- Pojmom i podelom operatora
- Osnovnim aritmetičkim operatorima
- Konverzijom tipova podataka

