Visoka tehnička škola Niš

Studijski program:

Savremene računarske tehnologije

Internet programiranje

(4)

Paketi i tipovi podataka u JAVI

Prof. dr Zoran Veličković, dipl. inž. el.

Oktobar, 2018.

Javin prostor imena

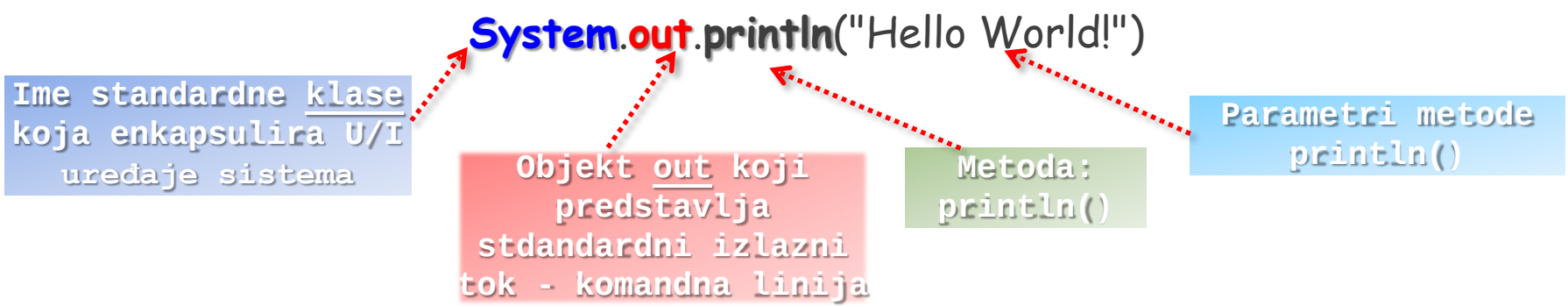
- U Javi je sve u **KLASAMA**, pa i sama aplikacija!
- U Javi se definišu **PROSTORI IMENA** u kojima se smeštaju **METODE** i **PODACI** koji se koriste u odgovarajućoj **KLASI**.
- Ovo znači da **METODE** mogu imati **ISTA IMENA** samo ako pripadaju **RAZLIČITIM IMENSKIM PROSTORIMA**.
- U tom slučaju, mora se tačno definisati **KOJA SE METODA POZIVA**, odnosno, prilikom poziva mora se navesti **PROSTORA IMENA** gde je smeštena željena metoda.
- **IMENSKI PROSTORI** su **HIJERARJISKI** organizovani nalik folderskoj organizaciji fajlova.
- Ovaj koncept omogućava **EFIKASNO POVEZIVANJE** programskih celina u kompleksni projekat bez bojazni o preklapanju imena.

Javin prostor imena i paketi

- Tako se umesto:

println("Hello World!"),

može (i treba) pisati:



- Za **SMEŠANJE** klasa i **PODELU IMENSKOG PROSTORA** u Javi se koriste **PAKETI**.
- Da bi se definisalo **KOJA** se metoda tačno poziva, mora se koristiti **HIJERARHIJSKI** pristup metodi određen preko **IMENA PAKETA** i **KLASE** (pr. klasa **System** se nalazi u paketu **java.lang** koji se podrazumevano učitava u JAVA projekte).

println( ) i escape sekvence

- U **println( )** metodi se mogu primeniti i tzv. **ESCAPE SEKVENCE** (to su sve sekvence koje počinju „backslash" karakterom "****") za **FORMATIRANO ŠTAMPANJE** na komandnoj liniji.
- Ove sekvence su nasleđene iz ere linijskih štampača i teleprintera.
- Tabela najčešće korišćenih **escape karaktera**:

CHR	ZNAČENJE	OBJAŠNJENJE
\n	Newline	Postavi poziciju kursora na početku sledeće linije.
\t	Horizontal tab	Pomeri kursor do sledeće tabulir pozicije.
\r	Carriage return	Postavi kursor na početak tekuće linije, ne prelazi se u sledeću liniju. Prethodno ispisani karakteri će biti prepisani novim.
\\	Backslash	Koristi se za štampanje "backslash" karaktera.
\"	Double quote	Koristi se za štampanje "double-quote" karaktera. Primer: System.out.println("\"in quotes\""); prikazuje "in quotes".

```
System.out.println("Welcome\nto\nJava\nProgramming!");
```

Formatirano štampanje/učitavanje

- Za **FORMATIRANO ŠTAMPANJE** može se koristiti i metoda **printf( )** - Vama već poznata iz C-a.
- Metoda **printf( )** koristi **ZAPETAMA ODVOJENU LISTU** koja može sadržavati **VIŠE ARGUMENATA** za štampanje.
- Primer formatiranog štampanja **teksta** u **DVE LINIJE**:

```
System.out.printf("%s\n%s\n", "Dobrodošli u", "programiranje na Javi!");
```

```
Dobrodošli u
programiranje na Javi!
```

- Štampanje decimalnih **CELIH BROJEVA**:

```
System.out.printf("Zbir je %d\n", zbir);
```

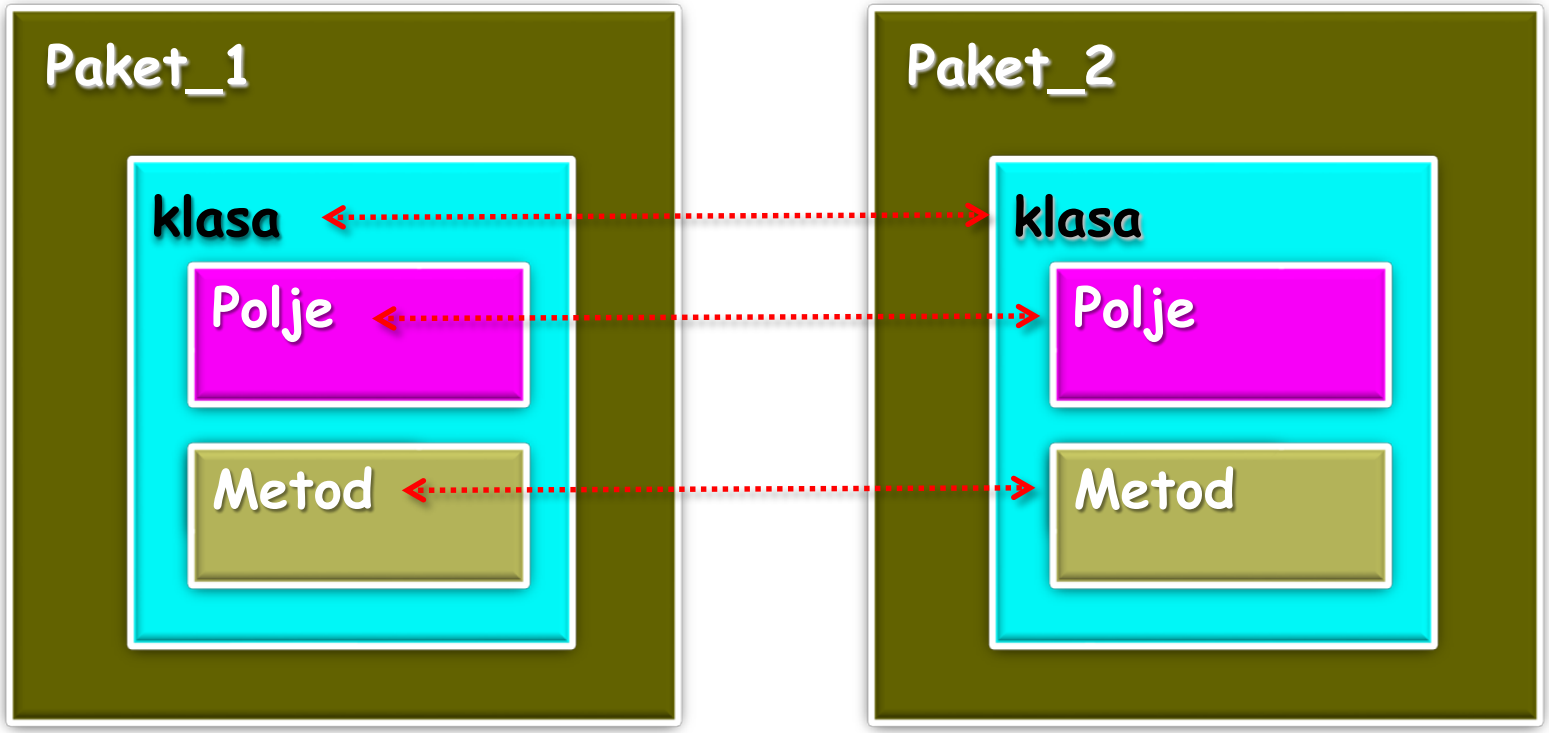
- Za **UČITAVANJE CELIH DECIMALNIH BROJEVA** sa **TASTATURE** (standardnog ulaza) može se koristi metoda **nextInt()** na sledeći način:

```
Int_Broj=input.nextInt();
```

Paketi (1)

- Već je pomenuto, **PAKETI** služe za **SMEŠANJE KLASA** i **PODELU** imenskog prostora u **Javi**.
- **PAKETI** zapravo predstavljaju **SKUP KLASA** (kaže se kontejner) sa **JEDINSTVENIM IMENOM**.
- **Svaka klasa** u Javi je **DEO** nekog paketa!
- Paketi se čuvaju **HIJERARHIJSKI**, a **IZRIČITO SE UVOZE** u definiciju **NOVE KLASE** (postoji jedan izuzetak – paket Java.lang).
- Novi **paketi se formiraju** komandom "**package**" koja mora biti **PRVA NAREDBA** u izvornoj datoteci.
- Za hijerarhijsko čuvanje paketa koristi se **SISTEM DIREKTORIJUMA**.
- Tako, datoteke sa ekstenzijom **.class** za bilo koju klasu deklarisanu u paketu **MojPaket MORA** da bude u **direktorijumu MojPaket**!

Paketi (2)



Dva nezavisna paketa Paket_1 i Paket_2 sa svojim metodama i poljima **MOGU** imati ista imena a da pri tom ne dođe do problema.

Uvoženje paketa

- Paketi se organizuju u **hijerarhiju** primenom **OPERATORA TAČKA** (.), primer:

  package java.awt.image;

- Tako, prethodni primer zahteva da paket bude **smešten** u direktorijum:

  java\awt\image.

- **UVOŽENJE** već formiranog **paketa** u programski kod se obavlja rezervisanom reči "**import**" na sledeći način:

  import java.util.Date ili **import java.lang.*;**

- Zvezdicom (*) se označava uvoženje **CELOG PAKETA**.
- Iako uvoženje čitavog paketa **produžava vreme kompajliranja**, to **NEMA NEGATIVNOG EFEKTA** u trenutku izvršavanja programa!
- **Standardne** (ugrađene) **Javine klase** se čuvaju u **PAKETU**: **java**.
- **OSNOVNE FUNKCIJE** samog jezika se čuvaju u potpaketu **java.lang** koji se **AUTOMATSKI** uvozi u **SVE** programe.
- Važno: Paketima se takođe definiše i **PRAVO PRISTUPA** metodama!

Standardni Java paketi (1)

PAKET	OBJAŠNJENJE
java.lang	Sadrži klase koje su od **fundamentalnog** značaja za Javu (npr. klasa Math). Sve promenljive su dostupne u svojim programima automatski. Nema potrebe za naredbom **uvoza** ovog paketa.
java.io	Sadrži klase koje se odnose na podršku **ulazno/izlaznim** operacijama (tokovima).
java.awt	Sadrži klase koje podržavaju Javin **grafički korisnički interfejs** (GUI). Iako se ove klase mogu koristiti za GUI programiranje, preporučuje se korišćenje alternativne klase Swing.
javax.swing	Obezbeđuje klase koje podržavaju "Swing" GUI komponente. One nisu samo fleksibilnije i lakše za korišćenje u odnosu na java.awt ekvivalente, već su **minimalno zavisne** od izvornog koda.
java.applet	Sadrži klase koje podržavaju pisanje **apleta** – programa integrisanih u Web stranicu (obrađivano u predmetu Internet tehnologije).
java.util	Sadrži klase za **široku podršku** standardnim operacijama kolekcija, vremenskim informacijama i stringovima.

Skoro svi Javini paketi!

java.applet, **java.awt**, java.awt.color, java.awt.datatransfer, java.awt.dnd, java.awt.event, java.awt.font, java.awt.geom, java.awt.im, java.awt.im.spi, java.awt.image, java.awt.image.renderable, java.awt.print, java.beans, java.beans.beancontext, **java.io**, **java.lang**, java.lang.annotation, java.lang.instrument, java.lang.management, java.lang.ref, java.lang.reflect, java.math, java.net, java.nio, java.nio.channels, java.nio.channels.spi, java.nio.charset, java.nio.charset.spi, java.rmi, java.rmi.activation, java.rmi.dgc, java.rmi.registry, java.rmi.server, java.security, java.security.acl, java.security.cert, java.security.interfaces, java.security.spec, java.sql, java.text, java.text.spi, **java.util**, java.util.concurrent, java.util.concurrent.atomic, java.util.concurrent.locks, java.util.jar, java.util.logging, java.util.prefs, java.util.regex, java.util.spi, java.util.zip, javax.accessibility, javax.activation, javax.activity, javax.annotation, javax.annotation.processing, javax.crypto, javax.crypto.interfaces, javax.crypto.spec, javax.imageio, javax.imageio.event, javax.imageio.metadata, javax.imageio.plugins.bmp, javax.imageio.plugins.jpeg, javax.imageio.spi, javax.imageio.stream, javax.jws, javax.jws.soap, javax.lang.model, javax.lang.model.element, javax.lang.model.type, javax.lang.model.util, javax.management, javax.management.loading, javax.management.modelmbean, javax.management.monitor, javax.management.openmbean, javax.management.relation, javax.management.remote, javax.management.remote.rmi, javax.management.timer, javax.naming, javax.naming.directory, javax.naming.event, javax.naming.ldap, javax.naming.spi, javax.net, javax.net.ssl, javax.print, javax.print.attribute, javax.print.attribute.standard, javax.print.event, javax.rmi, javax.rmi.CORBA, javax.rmi.ssl, javax.script, javax.security.auth, javax.security.auth.callback, javax.security.auth.kerberos, javax.security.auth.login, javax.security.auth.spi, javax.security.auth.x500, javax.security.cert, javax.security.sasl, javax.sound.midi, javax.sound.midi.spi, javax.sound.sampled, javax.sound.sampled.spi, javax.sql, javax.sql.rowset, javax.sql.rowset.serial, javax.sql.rowset.spi, **javax.swing**, javax.swing.border, javax.swing.colorchooser, javax.swing.event, javax.swing.filechooser, javax.swing.plaf, javax.swing.plaf.basic, javax.swing.plaf.metal, javax.swing.plaf.multi, javax.swing.plaf.synth, javax.swing.table, javax.swing.text, javax.swing.text.html, javax.swing.text.html.parser, javax.swing.text.rtf, javax.swing.tree, javax.swing.undo, javax.tools, javax.transaction, javax.transaction.xa, javax.xml, javax.xml.bind, javax.xml.bind.annotation, javax.xml.bind.annotation.adapters, javax.xml.bind.attachment, javax.xml.bind.helpers, javax.xml.bind.util, javax.xml.crypto, javax.xml.crypto.dom, javax.xml.crypto.dsig, javax.xml.crypto.dsig.dom, javax.xml.crypto.dsig.keyinfo, javax.xml.crypto.dsig.spec, javax.xml.datatype, javax.xml.namespace, javax.xml.parsers, javax.xml.soap, javax.xml.stream, javax.xml.stream.events, javax.xml.stream.util, javax.xml.transform, javax.xml.transform.dom, javax.xml.transform.sax, javax.xml.transform.stax, javax.xml.transform.stream, javax.xml.validation, javax.xml.ws, javax.xml.ws.handler, javax.xml.ws.handler.soap, javax.xml.ws.http, javax.xml.ws.soap, javax.xml.ws.spi, javax.xml.ws.wsaddressing, javax.xml.xpath, org.ietf.jgss, org.omg.CORBA, org.omg.CORBA_2_3, org.omg.CORBA_2_3.portable, org.omg.CORBA.DynAnyPackage, org.omg.CORBA.ORBPackage, org.omg.CORBA.portable, org.omg.CORBA.TypeCodePackage, org.omg.CosNaming, org.omg.CosNaming.NamingContextExtPackage, org.omg.CosNaming.NamingContextPackage, org.omg.Dynamic, org.omg.DynamicAny, org.omg.DynamicAny.DynAnyFactoryPackage, org.omg.DynamicAny.DynAnyPackage, org.omg.IOP, org.omg.IOP.CodecFactoryPackage, org.omg.IOP.CodecPackage, org.omg.Messaging, org.omg.PortableInterceptor, org.omg.PortableInterceptor.ORBInitInfoPackage, org.omg.PortableServer, org.omg.PortableServer.CurrentPackage, org.omg.PortableServer.POAManagerPackage, org.omg.PortableServer.POAPackage, org.omg.PortableServer.portable, org.omg.PortableServer.ServantLocatorPackage, org.omg.SendingContext, org.omg.stub.java.rmi, org.w3c.dom, org.w3c.dom.bootstrap, org.w3c.dom.events, org.w3c.dom.ls, org.xml.sax, org.xml.sax.ext, org.xml.sax.helpers

Tipovi podataka u Javi

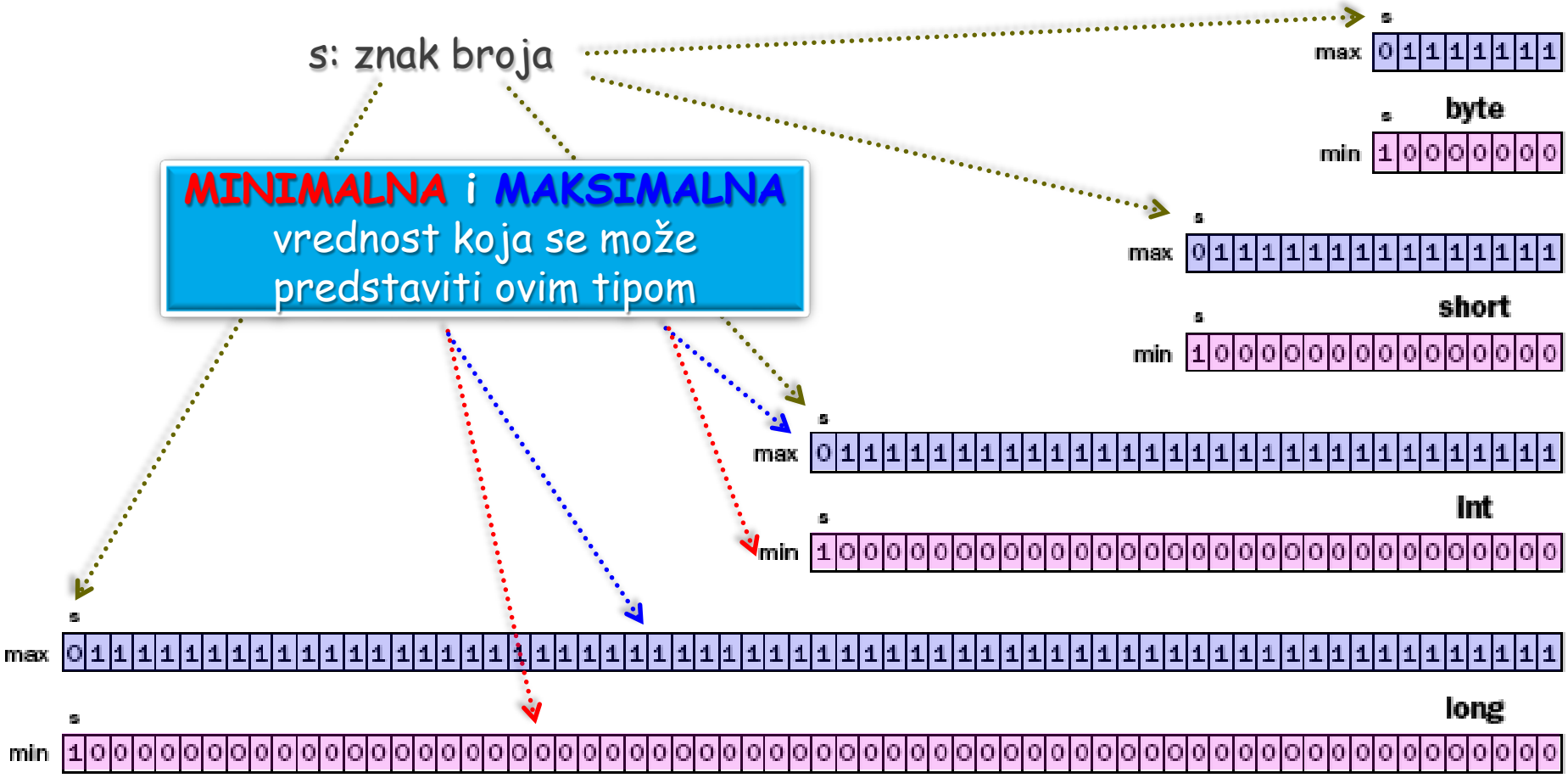
- Java je strogo **TIPIZIRAN JEZIK** (šta to zapravo znači?). Evo odgovora: Svaka **promenljiva** i svaki **izraz** u Javi moraju da imaju **svoj TIP**!
- Pri **dodeljivanju vrednosti PROMENLJIVAMA** ili **IZRAZIMA** Java vrši **strogu PROVERU TIPA**.
- U Javu su ugrađeni **ELEMENTARNI** (ili vrednosni, odnosno, prosti) **tipovi podataka** kao što su: **bajt**, **short**, **int**, **long**, **char**, **float**, **double** i **boolean**.
- Svi **ELEMENTARNI TIPOVI** su svrstani u sledeće **GRUPE**:
  - **Celi brojevi**,
  - **Brojevi u pokretnom zarezu**,
  - **Znakovi** i
  - **Logičke vrednosti**.
- Setite se da je u Javi sve u **KLASAMA**, međutim, prosti tipovi podataka **NISU OBJEKTI** (odmah pa izuzetak!) razlog ovome je efikasnost **PREDSTAVLJANJA** u memeoriji i **operativnog rada** sa njima.

Elementarni tipovi u Javi (1)

Tip podataka	Veličina [bajt]	Opseg
bajt	1	-128 do +127
boolean	1	Tačno ili netačno
char	2	A-Z, a-z, 0-9, spec char.
short	2	-32768 do +32767
int	4	-2 miliona do +2 miliona
long	8	-10^{18} do $+10^{18}$
float	4	$-3.4 \cdot 10^{38}$ do $+3.4 \cdot 10^{38}$
double	8	$-1.7 \cdot 10^{308}$ do $+1.7 \cdot 10^{308}$

- Primetite, **VIŠE BITOVA**, odnosno, **BAJTOVA** je potrebno rezervisati za **veći** brojni opseg (engl. range).
- Programer **NE MOŽE** kreirati prikazane **ELEMENTARNE TIPOVE**, već samo može **izabrati odgovarajući**.

Elementarni tipovi u Javi (2)



Referencni tipovi u Javi (1)

- Šta su zapravo **TIPOVI PODATKA**?
- **TIPOVI PODATKA** se mogu smatrati **ŠABLONIMA** koji opisuju **FUNKCIONALNOST** kolekcije podataka.
- Pored **VREDNOSNIH TIPOVA** podataka, u Javi postoje i **REFERENCNI TIPOVI**.
- Osnovna razlika između ovih tipova je u **NAČINU SMEŠTANJA** (u memoriji) i **PRISTUPA** podacima.
- Generalno, memorija za podatke je Javi podeljena na **STEK** i **HIP** deo.
- **VREDNOSNI** (elementarni) tipovi podataka se smeštaju u **STEK DEO MEMORIJE** (integer, boolen, char, struct).
- **Posle korišćenja**, ovi tipovi podataka **OSLOBAĐAJU** zauzetu memoriju.
- Ovaj **način korišćenja** memorije je bio prisutan i u **starijim** programskim jezicima.

Referencni tipovi u Javi (2)

- Za razliku od vrednosnih tipova, promenljive **REFERENCNOG** tipa postoje **ISTOVREMENO** u **OBA** memorijska dela (dakle, i u hipu i na steku).
- Konkretni podaci **OBJEKTA** se nalaze u **DINAMIČKOM DELU MEMORIJE** - hip-u (engl. heap).
- **ISTOVREMENO** se formira promenljiva (naziva se **REFERENCA** te otuda i naziv) na **STEKU** koja **ukazuje** na **objekt** u **HIP** memoriji.
- Kada neka funkcija **pristupa referencnoj promenljivoj**, vraća se **MEMORIJSKA ADRESA** objekta (a ne sam objekt) na koji ona ukazuje!
- Kada se referencna promenljiva **više ne koristi**, referenca objekta se **UNIŠTAVA** ali **NE** i sam objekt (mada ovo ima za posledicu oslobađanje pridružene memorije).
- Ova procedura je slična **brisanju datoteke** sa hard-diska.
- Kada objekat **NEMA** svojih **referenci** podložan je **SKUPLJANJU OTPADAKA** (softveru koji je zadužen za skupljanje "đubreta").

Promenljive u Javi

- Šta su to **promenljive** u Javi?
- **PROMENLJIVE** su **oznake** koje opisuju pojedine **lokacije u memoriji** kojoj je **pridružen TIP PODATAKA**.
- Promenljive su **osnovna jedinica** za **ČUVANJE VREDNOSTI** u **Javi**.
- **Pre upotrebe** promenljive se **MORAJU DEKLARISATI**!
- Evo nekoliko primera ispravne **deklaracije** elementarnih promenljivih u Javi:

```
int a, b, c;                  // deklaracija tri integera a, b, and c.
int d = 3, e, f = 5;          // dekl. i inic. integera d and f.
byte z = 22;                  // inicijalizacija z.
double pi = 3.14159;          // deklaracija Rudolfovog brija π.
char x = 'x';                 // promenljiva x ima vrednost 'x'.
```

Java: int promenljive

```
class Example2 {
  public static void main (String args[]) {
    int num;                          // deklaracija promenljive num
    num = 100;                        // pridružena vrednosti num, 100
    System.out.println("This is num: " + num);
    num = num * 2;                    // pridružena vrednosti num*2
    System.out.print("The value of num * 2 is ");
    System.out.println(num);
  }
}
```

Program za prikaz zbira dva broja

```
import java.util.Scanner;              // koristi se klasa Scanner iz java.util
public class Addition
 {
    public static void main( String[] args )
    {
      // kreira se objekt input tipa Scanner da se dobije ulaz sa tastature
    Scanner input = new Scanner(System.in);                 // referencna promenljiva
    int number1;            // prvi sabirak                 // vrednosna promenljiva
    int number2;            // drugi sabirak                // vrednosna promenljiva
    int sum;                // zbir dva broja number1 i number2, vredn.promenljive
    System.out.print("Unesi prvi broj: " );                 // ispiši
    number1 = input.nextInt( );                    // učitaj prvi int. br. sa tast.
    System.out.print( " Unesi drugi broj: " );        // ispiši
    number2 = input.nextInt( );                  // učitaj drugi int. br. sa tastat.
    sum = number1 + number2;      // saberi brojeve i smesti zbir u prom. sum
    System.out.printf("Zbir je %d\n", sum ); // prikaži promeljivu sum
    }    // kraj metode main
}        // end class Addition
```

Operatori poređenja u Javi

- Rezultat svakog poređenja je **Bulova promenljiva** koja može uzeti vrednosti **true** (istina) ili **false** (laž).

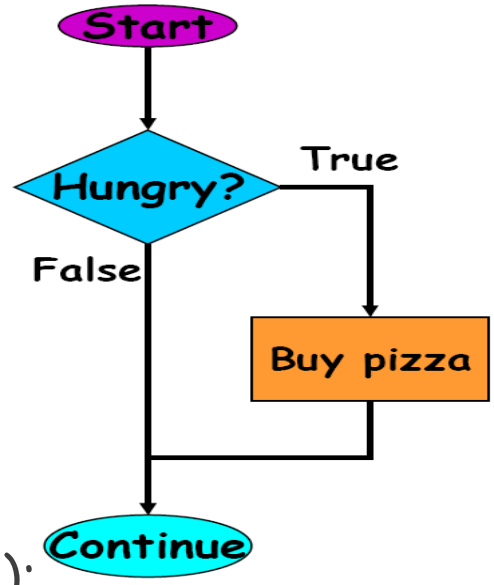
Operatori	Značenje operatora
<	Manje od (engl. less than)
<=	Manje ili jednako (engl. less than or equal to)
==	Jednako (engl. equal to)
>=	Veće ili jednako (engl. greater than or equal to)
>	Veće od (engl. greater than)
!=	Nije jednako (engl. not equal)

Upravljačke naredbe u Javi (1)

```
class IfSample {
  public static void main (String args[]) {
   int x, y;
   x = 10;
   y = 20;
    if(x < y) System.out.println("x je manje od y");
          x = x * 2;
    if(x == y) System.out.println("x je sada jednako y");
          x = x * 2;
    if(x > y) System.out.println("x je sada veće od y");
        // ovaj kod ispod neće ništa prikazati, zašto?
    if(x == y) System.out.println("you won't see this");
  }
}
```

False

true



Upravljačke naredbe u Javi (2)

```
class ForTest {
    public static void main (String args[]) {
    int x;                                  // deklaracija prom. x
     for(x = 0; x<10; x = x+1)              // petlja za prikaz
     System.out.println("Vrednost x-a je: " + x);
  }
}
```

Telo naredbe **for**

Parametri naredbe for:

1. Inicijalizacija petlje, x=0
2. Ispitivanje uslova izlaska iz petlje, x<10
3. Korak uvećanja promenljive, x=x+1

Java: blok koda

```
/*  Demonstracija blok koda. */
class BlockTest {
  public static void main(String args[]) {
    int x, y;
    y = 20;
    // For petlja u ovom slučaju predstavlja blok koda
    for(x = 0; x<10; x++) {
      System.out.println("This is x: " + x);
      System.out.println("This is y: " + y);
      y = y - 2;
    }
  }
}
```

Java: dodela float vrednosti

```
// Računa površinu kruga.
class Area {
  public static void main (String args[]) {
    double pi, r, a;                    // deklaracija promenljivih

    r = 10.8;                           // prečnik kruga
    pi = 3.1416;                        // Rudolfov broj pi
    a = pi * r * r;                     // Računa površinu kruga
    System.out.println("Površina kruga je " + a);
  }
}
```

Java: dodela char vrednosti

```
// Primer char tipa podataka
class CharDemo {
  public static void main (String args[]) {
    char ch1, ch2;
    ch1 = 88;                    //dec. Predstava broja X
    ch2 = 'Y';                   // ASCII kod za Y
    System.out.print("ch1 i ch2: ");
    System.out.println (ch1 + " " + ch2);
  }
}
```

Jednostavno **formatiranje** izlaza

Šta se dobija na izlazu?

Java: dinamička inicijalizacija

```
// Primer dinamičke inicijalizacije promenljivih.
class DynInit {
    public static void main (String args[]) {
        double a = 3.0, b = 4.0;
            // c je dinamički inicijalizovana promenljiva
        double c = Math.sqrt(a * a + b * b);
      System.out.println("Hypotenuse is " + c);
    }
}
```

Class Math, sqrt - metoda !

Oblast važenja u Javi

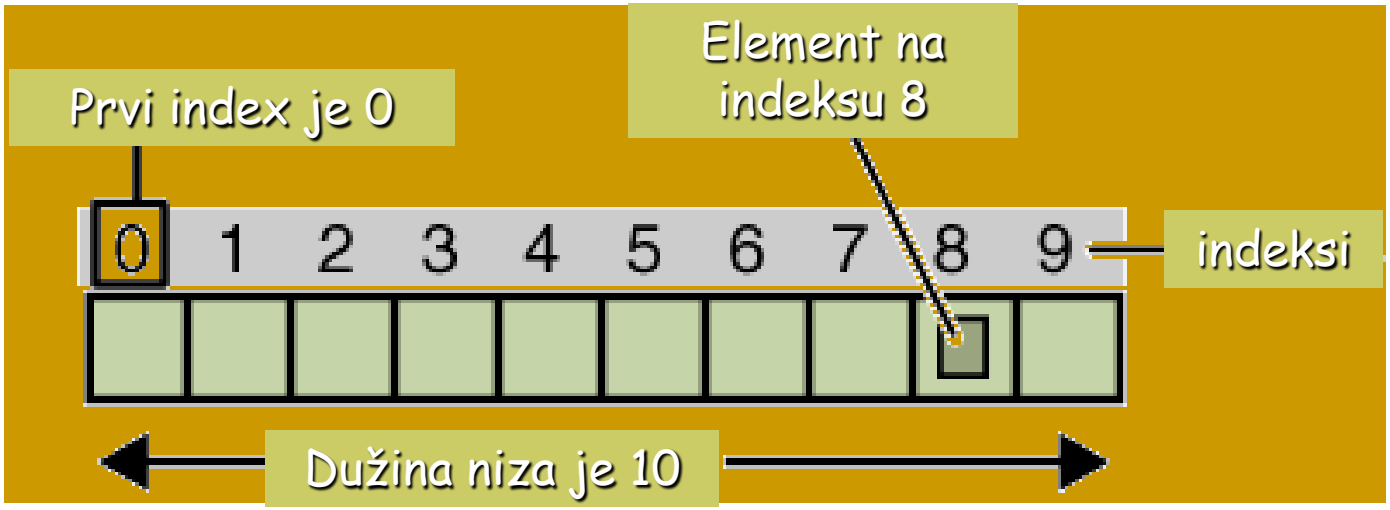
- **Oblast važenja** (ili vidljivosti) promenljivih je značajan pojam za kapsuliranje podataka.
- Promenljive se mogu definisati u okviru **bilo kog** bloka.
- **Zapamtie: BLOK** definiše **oblast važenja**!
- **Oblast važenja** određuje objekte koji će biti **vidljivi drugim** delovima programa.
- **Oblast važenja** može biti definisan **KLASOM**, **METODOM** i **PAKETOM**.
- **Oblast važenja** definisana **METODOM** počinje sa „ **{** „.
- Promenljive definisane **unutar** oblasti važenja **NISU VIDLJIVE** naredbama koje si definisane **izvan** te oblasti.

Java: Oblast važenja (2)

```
class Scope {
  public static void main(String args[]) {
    int x;                    // Poznato svim delovima koda unurtar main-a
    x = 10;
    if(x == 10) {             // start oblasti važenja
      int y = 20;             // y poznato samo u ovom bloku!
                              // x i y su poznati ovde
      System.out.println("x and y: " + x + " " + y);
      x = y * 2;
    }
    // y = 100;               // Greška! Promenljiva y ovde nije poznata
                              // x je još uvek ovde poznata !
    System.out.println("x is " + x);
  } }
```

Java: nizovi (1)

- **NIZ** (engl. array) predstavlja **GRUPU PROMENLJIVIH ISTOG TIPA** koje se pojavljuju pod **ZAJEDNIČKIM NAZIVOM**.
- Zapamtite: Nizovi su **referencni tipovi podataka**!
- Koje su **posledice** (ili **benefiti**) ove činjenice?



Java: nizovi (2)

- Mogući su nizovi **svih** tipova!
- Pristup **elementima niza** moguć je preko njihovog **rednog broja - INDEXA**.
- Moguće je definisati sledeće tipove nizova:
  - **JEDNODIMENZIONE** (1D) i
  - **VIŠEDIMENZIONE** nizove (xD):
- Primer dekleracije 1D, 2D i 3D nizova:

```
month_days[] = new int[12];          // 1D niz
int twoD[][] = new int[4][5];        // 2D niz
int threeD[][][] = new int[3][4][5]; // 3D niz
```

Operator **new** je neophodno primeniti prilikom formiranja **REFERENCNIH** tipova!

Java: 2D nizovi

```
class Matrix {
  public static void main (String args[]) {
    double m[][] = {
      { 0*0, 1*0, 2*0, 3*0 },
      { 0*1, 1*1, 2*1, 3*1 },
      { 0*2, 1*2, 2*2, 3*2 },
      { 0*3, 1*3, 2*3, 3*3 }
    };
    int i, j;
    for(i=0; i<4; i++) {
      for(j=0; j<4; j++)
        System.out.print(m[i][j] + " ");
      System.out.println();
    } }
}
```

Dekleracija i inicijalizacija 2D niza

Štampanje elemenata 2D niza !

Znakovni nizovi u Javi

- Java podržava **ZNAKOVNE NIZOVE** svojom klasom **String**.
- **STRING JE OBJEKT**, što podrazumeva da su mu pridružene **METODE** za rad sa ovim tipom promenljive.
- Mogu se definisati i **NIZOVI ZNAKOVNIH NIZOVA** (setite se dekleracije metode **main**, koja ima kao parametar **niz znakovnih nizova**:

```
main(String args[]).
```

- Objekt tipa string se može koristiti kao **ARGUMENT** metode **println()**.

```
String str = "Inicijalizacija string promenljive";

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

- Objekti tipa string poseduju **SPECIFIČNE OSOBINE** i **ATRIBUTE** koje olakšavaju njihovo korišćenje.
- Veći deo standardnih klasa i metoda za obradu stringova je obrađen u C#-u.