

# Java

primitivni tipovi, operatori, kontrola toka  
programa, nizovi

# Primitivni tipovi – lista

<u>Primitivni tip</u>	<u>Veličina</u>	<u>Minimum</u>	<u>Maksimum</u>
<u>boolean</u>	1-bit	–	–
char	16-bit	Unicode 0	Unicode $2^{16} - 1$
byte	8-bit	–128	+127
short	16-bit	$-2^{15}$	$+2^{15} - 1$
<u>int</u>	32-bit	$-2^{31}$	$+2^{31} - 1$
long	64-bit	$-2^{63}$	$+2^{63} - 1$
float	32-bit	IEEE754	<u>IEEE754</u>
double	64-bit	IEEE754	<u>IEEE754</u>
void	–	–	–

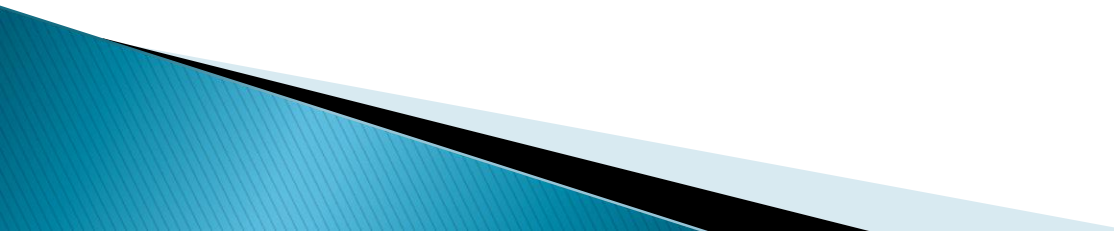
# Primitivni tipovi – literali

- ▶ Celobrojni: `2`, `2000000L`
- ▶ Razlomljeni: `3.14f`, `3.14`
- ▶ Heksadecimalni: `0xF`, `0xFF`
- ▶ Oktalni: `0123`
- ▶ Binarni: `0b1001101`
- ▶ Boolean: `true`, `false`
- ▶ Znakovni:
  - `'a'`
  - `'\n'`
  - `'\xxx'`, gde je `xxx` oktalni ASCII kod karakter

# Primitivni tipovi – deklaracija, konverzije

- ▶ Deklaracija
  - ▶ `int a;`
  - ▶ `int a = 0;`
  - ▶ `int a, b;`
  - ▶ `int a = 0, b = 3;`
  - ▶ `final int a = 55;`
  - ▶ `//konstanta`
- ▶ Implicitna konverzija
  - ▶ `int a = 5;`
  - ▶ `double b;`
  - ▶ `b = a;`
- ▶ Eksplicitna konverzija
  - ▶ `long a = 5L;`
  - ▶ `int b = a; //greška`
  - ▶ `int c = (int) a;`
  - ▶ `//ovo je dobro, to se zove cast operator`

# Operatori

- Operator dodele
  - Cast operator
  - Aritmetički operatori
  - Relacioni i logički
  - Bit-operatori
  - Operatori pomeranja
- 

# Operator dodele

- ▶ Osim dodeljivanja literala, promenljivoj se može dodeliti vrednost druge promenljive, ili neki izraz:

```
a = b;
```

```
c = (d * 10) / e;
```

- ▶ Ako su operandi primitivni tipovi, kopira se sadržaj:

```
int i = 3, j = 6;
```

```
i = j;    // u i ubačeno 6
```

# Cast operator

- ▶ Ako probamo da smestimo "širi" tip u "uži", to bi proizvelo grešku pri kompajliranju:
  - `long a = 44;`
  - `int b = a;`
- ▶ Kompajler ne može da smesti 64 bita u 32 bita i zbog toga prijavljuje grešku. To ćemo postići upotrebom kast (cast) operatora:
  - `long a = 44;`
  - `int b = (int)a;`

# Aritmetički operatori

- ▶ Osnovne operacije:

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$

- ▶ Umesto  $x = x + 1$

$x += 1$

- ▶ Automatski inkrement:  $++x$  odn.  $x++$



# Aritmetički operatori

$y = 5;$

Operator	Rezultat
$x=y+2$	$X \leftarrow 7$
$x=y-2$	$X \leftarrow 3$
$x=y\%2$	$X \leftarrow 1$
$x=++y$	$X \leftarrow 6, y \leftarrow 6$
$x=y++$	$X \leftarrow 5, y \leftarrow 6$
$x=--y$	$X \leftarrow 4, y \leftarrow 4$

# Aritmetički operatori

x = 10;  
y = 5;

Operator	Isto kao	Rezultat
x=y		x ← 5
x+=y	x=x+y	x ← 15
x-=y	x=x-y	x ← 5
x*=y	x=x*y	x ← 50
x/=y	x=x/y	x ← 2
x%=y	x=x%y	x ← 0

# Relacioni i logički operatori

- ▶ Relacioni:

- < > <= >= == !=

- ▶ Logički:

- && (I), || (ILI), ! (NE)

# Relacioni operatori

x = 5;

Operator	Rezultat
==	x == 8 je netačno (false)
!=	x != 8 je tačno (true)
>	x > 8 je netačno (false)
<	x < 8 je tačno (true)
>=	x >= 8 je netačno (false)
<=	x <= 8 je tačno (true)

# Logički operatori

- ▶ Logički: `&&` `||` `!`
- ▶ Rezultat logičkih operatora je tačno (true) ili netačno (false)
- ▶ Operandi logičkih operatora su logički izrazi

<code>&amp;&amp;</code>	false	true
false	false	false
true	false	true

<code>  </code>	false	true
false	false	true
true	true	true

<code>!</code>	
false	true
true	false

# Logički operatori

x = 6;

y = 3;

Operator	Objašnjenje	Primer
&&	konjukcija (and, i)	((x < 10) && (y > 1)) tačno (true)
	disjunkcija (or, ili)	((x==5)    (y==5)) netačno (false)
!	negacija (not, ne)	!(x==y) tačno (true)

# Ispis na ekran

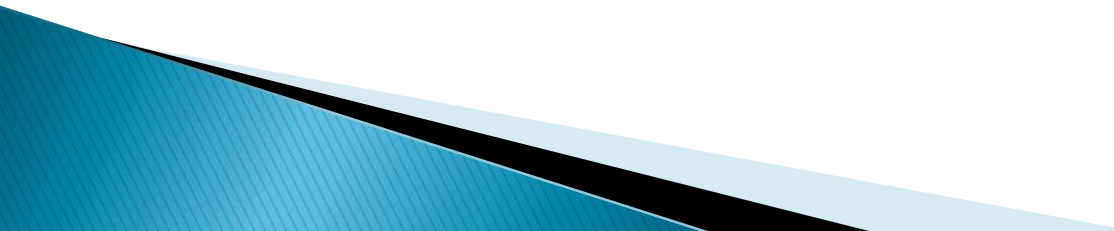
- ▶ `System.out.print("Poruka");`
- ▶ `System.out.println("Poruka");` // na kraju ima i new line
- ▶ `int ocena = 8;`
- ▶ `System.out.println("Dobili ste ocenu: " + ocena);`

# Ispis na ekran

- ▶ Sintaksa za formatizovani ispis
  - `System.out.printf("format", argumenti);`
- ▶ Primeri (`int c = 356;`)
  - `System.out.printf("celobrojni: %d\n", c);`      -->  
celobrojni: 356
  - `System.out.printf("celobrojni: %6d\n", c);`      -->  
celobrojni:    356
  - `System.out.printf("celobrojni: %+6d\n", c);`      -->  
celobrojni:   +356
  - `System.out.printf("celobrojni: %+6d\n", -c);`      -->  
celobrojni:   -356



# Kontrola toka

- ▶ `if else`
  - ▶ `switch`
  - ▶ `for`
  - ▶ `while`
  - ▶ `do while`
  - ▶ `break`
  - ▶ `continue`
- 

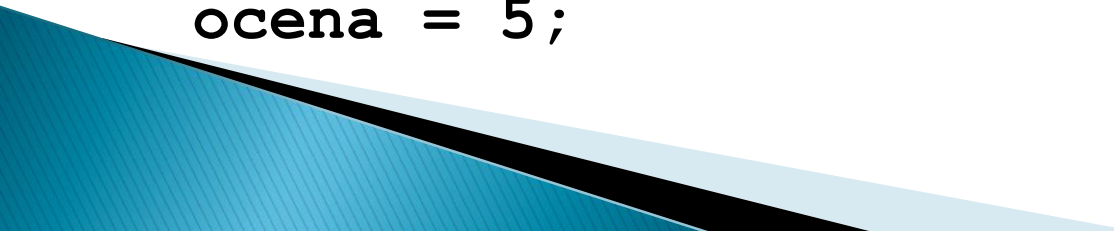
# if

- ▶ već smo radili kao deo prethodne lekcije
- ▶ akcije mogu biti blok naredbi
- ▶ uslov je kombinacija relacionih i logičkih operatora
- ▶ else deo može ponovo kao akciju imati if naredbu (postoji i ograničenje o dubini ugnježdavanja if naredbi)

```
if (uslov)  
    akcija  
else  
    druga_akcija
```

# if else – primer

```
if (bodovi >= 95)
    ocena = 10;
else if (bodovi >= 85)
    ocena = 9;
else if (bodovi >= 75)
    ocena = 8;
else if (bodovi >= 65)
    ocena = 7;
else if (bodovi >= 55)
    ocena = 6;
else
    ocena = 5;
```



# Uslovni operator

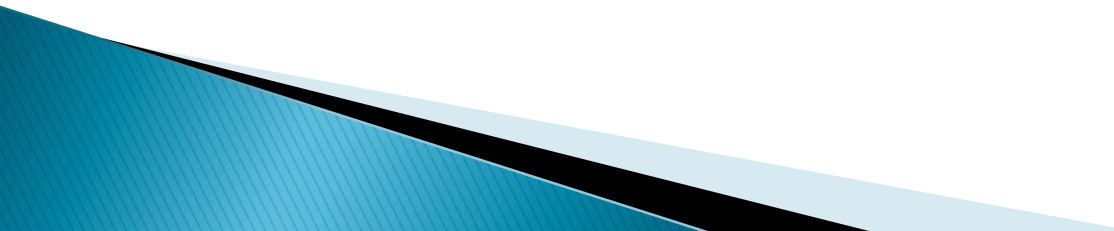
- ▶ Već smo radili kao deo prethodne lekcije

```
a = i < 10 ? i * 100 : i * 10;
```

- ▶ isto kao:

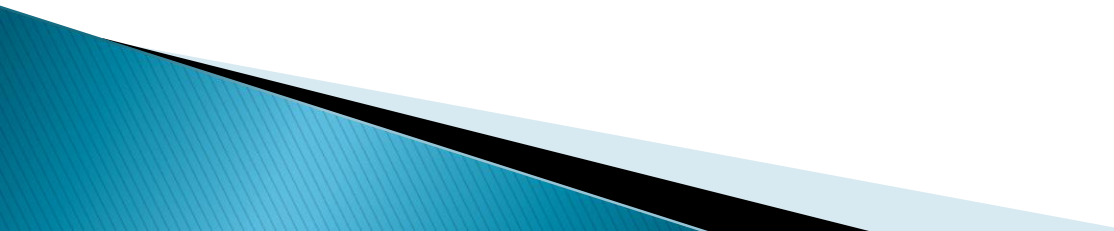
```
if (i < 10)
    a = i * 100;
else
    a = i * 10;
```

# switch

- ▶ Vrednost izraza u switch() mora da proizvede celobrojnu vrednost. Od jave 1.7 vrednost izraza može biti String.
  - ▶ Ako ne proizvodi celobrojnu vrednost, ne može da se koristi switch,() već if()!
  - ▶ Ako se izostavi break; propašće u sledeći case:
  - ▶ Kod default: izraza ne mora break; – to se podrazumeva.
- 

# switch – Primer

```
switch (ocena) {  
    case 5: System.out.println("odlican");  
        break;  
    case 4: System.out.println("vrlo dobar");  
        break;  
    case 3: System.out.println("dobar");  
        break;  
    case 2: System.out.println("dovoljan");  
        break;  
    case 1: System.out.println("nedovoljan");  
        break;  
    default: System.out.println("nepostojeca ocena");  
}
```



# switch – Primer

```
switch(c) {  
    case 'a':  
    case 'e':  
    case 'i':  
    case 'o':  
    case 'u':  
        System.out.println("samoglasnik");  
        break;  
default:  
    System.out.println("suglasnik");  
}
```

# for

- ▶ Za organizaciju petlji kod kojih se unapred zna koliko puta će se izvršiti telo ciklusa.
- ▶ Petlja sa početnom vrednošću, uslovom za kraj i blokom za korekciju.
- ▶ Opšta sintaksa:

```
for (inicijalizacija; uslov; korekcija)  
    telo
```



# for

```
for (int i = 0; i < 10; i++)
```

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

```
for (int i = 10; i >= 1; i--)
```

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

- ▶ može i višestruka inicijalizacija i step-statement:

```
for(int i = 0, j = 1;
```

```
i < 10 && j != 11; i++, j++)
```

- ▶ oprez može da se ne završi nikada, da bude beskonačna petlja

# for each petlja

- ▶ Za iteriranje kroz nizove (i kolekcije, o čemu će biti više reči kasnije) koristi se for petlja za iteriranje (for each petlja). Opšta sintaksa je sledeća:

```
for (promenljiva : niz)  
    telo
```

- ▶ Primer:

```
double[] niz = {1.0, 2.78, 3.14};  
for (double el : niz)  
    System.out.println(el);
```

# while

- ▶ Za cikličnu strukturu kod koje se samo zna uslov za prekid.
- ▶ Telo ciklusa ne mora ni jednom da se izvrši
- ▶ Opšta sintaksa:

```
while (uslov)
```

```
    telo
```

- ▶ Važno: izlaz iz petlje na false!

# while

```
int i = 0;
while (i <= 10) {
    System.out.println("Trenutno je " +
        i);
    i=i+1;
}
```

# do while

- ▶ Za cikličnu strukturu kod koje se samo zna uslov za prekid
- ▶ Razlika u odnosu na while petlju je u tome što se telo ciklusa izvršava makar jednom.
- ▶ Opšta sintaksa:

**do**

**telo**

**while (uslov) ;**

- ▶ Važno: izlaz iz petlje na false!

# do while

```
int i = 0;  
do {  
    System.out.println(i++);  
} while (i < 10);
```

# break i continue

- ▶ break – prekida telo tekuće ciklične strukture (ili case: dela) i izlazi iz nje.
- ▶ continue – prekida telo tekuce ciklične strukture i otpočinje sledeću iteraciju petlje.

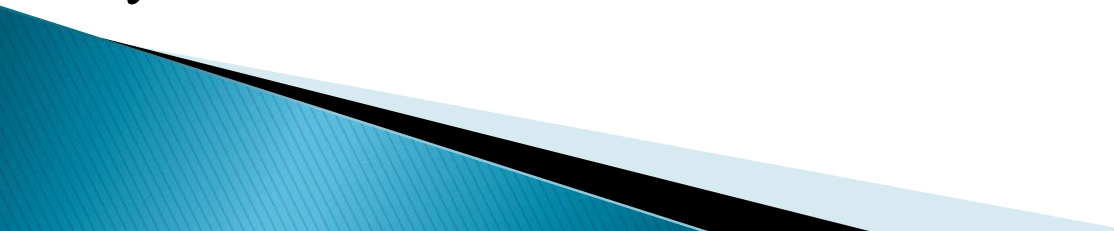
# break i continue

```
for(int i = 0; i < 10; i++) {  
    if (i==7) {  
        break;  
    }  
    if (i == 2)  
        continue;  
  
    System.out.println("Broj je:" + i);  
}
```



# break i continue

```
int i = 0;
System.out.println("Broj je:" + i);
while (i++ <= 9) {
    if (i == 7) {
        break;
    }
    if (i == 2)
        continue;
    System.out.println("Broj je:" + i);
}
```



# Izlaz iz ugnježdene petlje

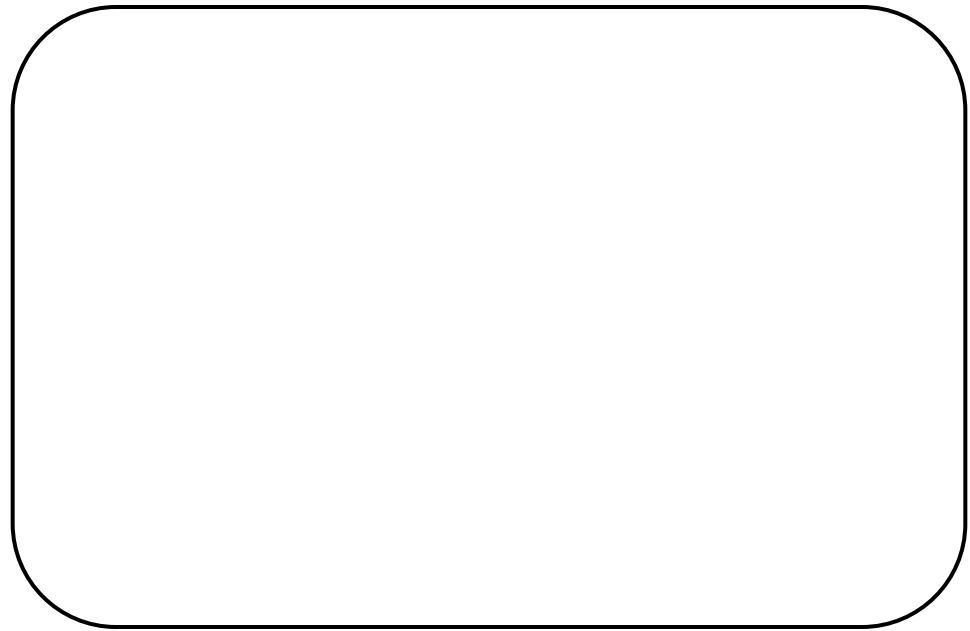
```
for (...)  
{  
    for (...)  
    {  
        ...  
        if (uslov)  
            break;  
    }  
}
```

# Nizovi primitivnih tipova <sub>1/3</sub>

```
int a[];
```



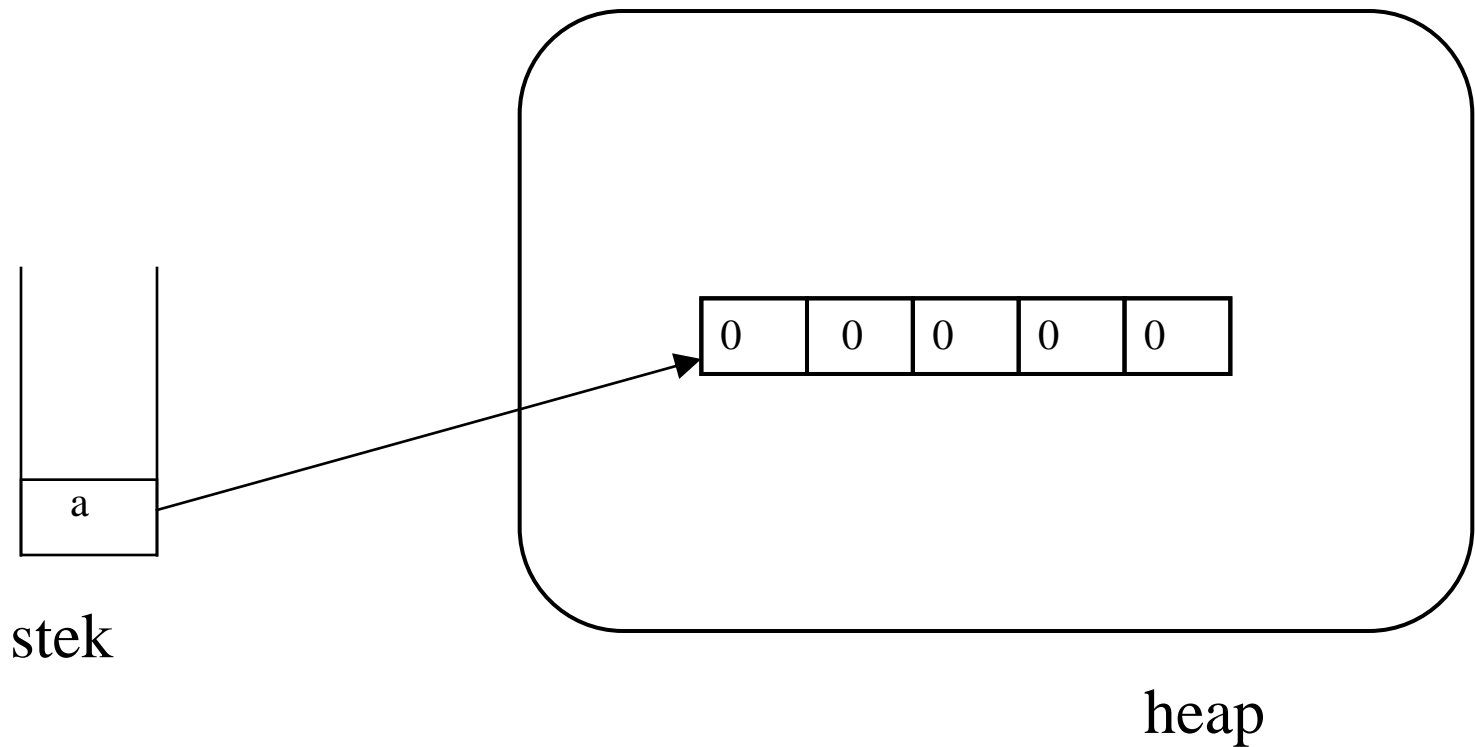
stek



heap

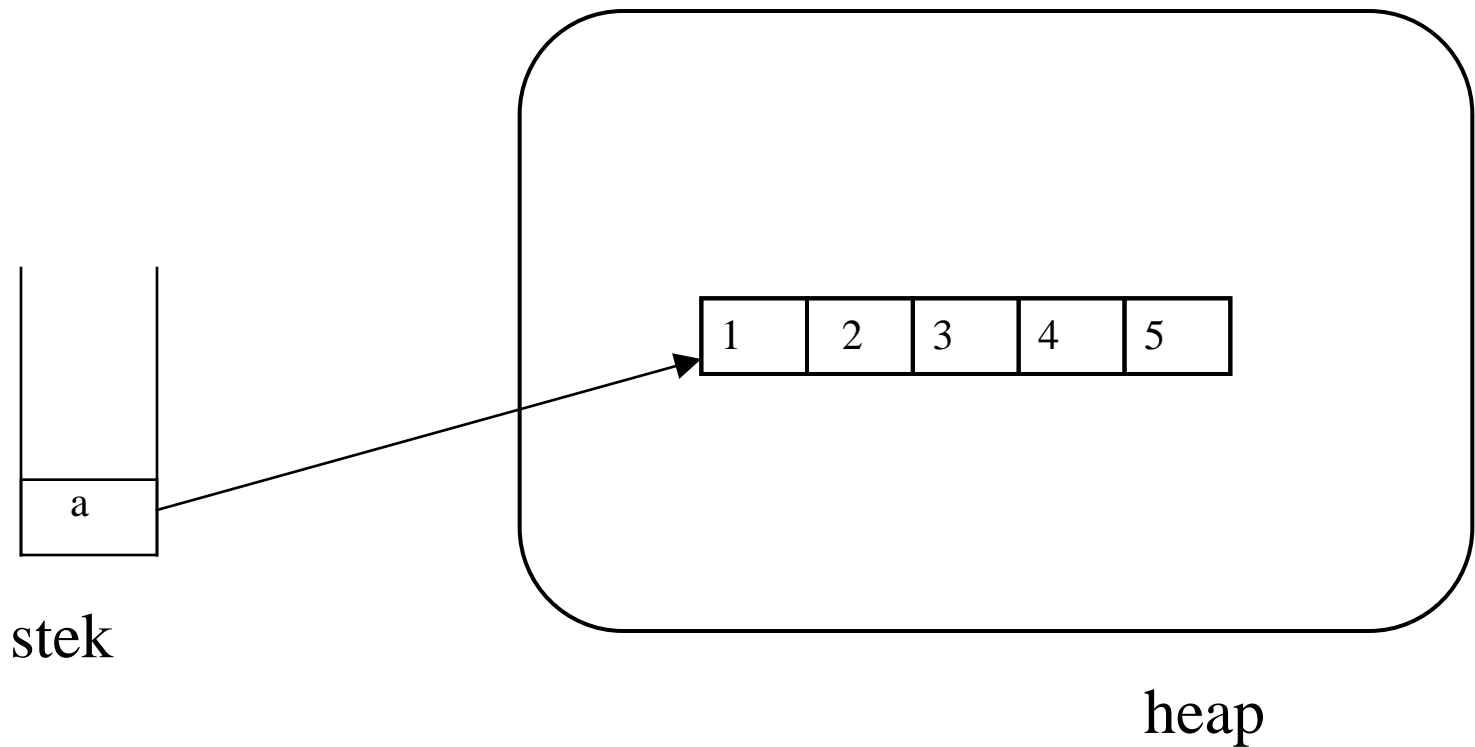
# Nizovi primitivnih tipova <sub>2/3</sub>

```
a = new int[5];
```



# Nizovi primitivnih tipova <sub>3/3</sub>

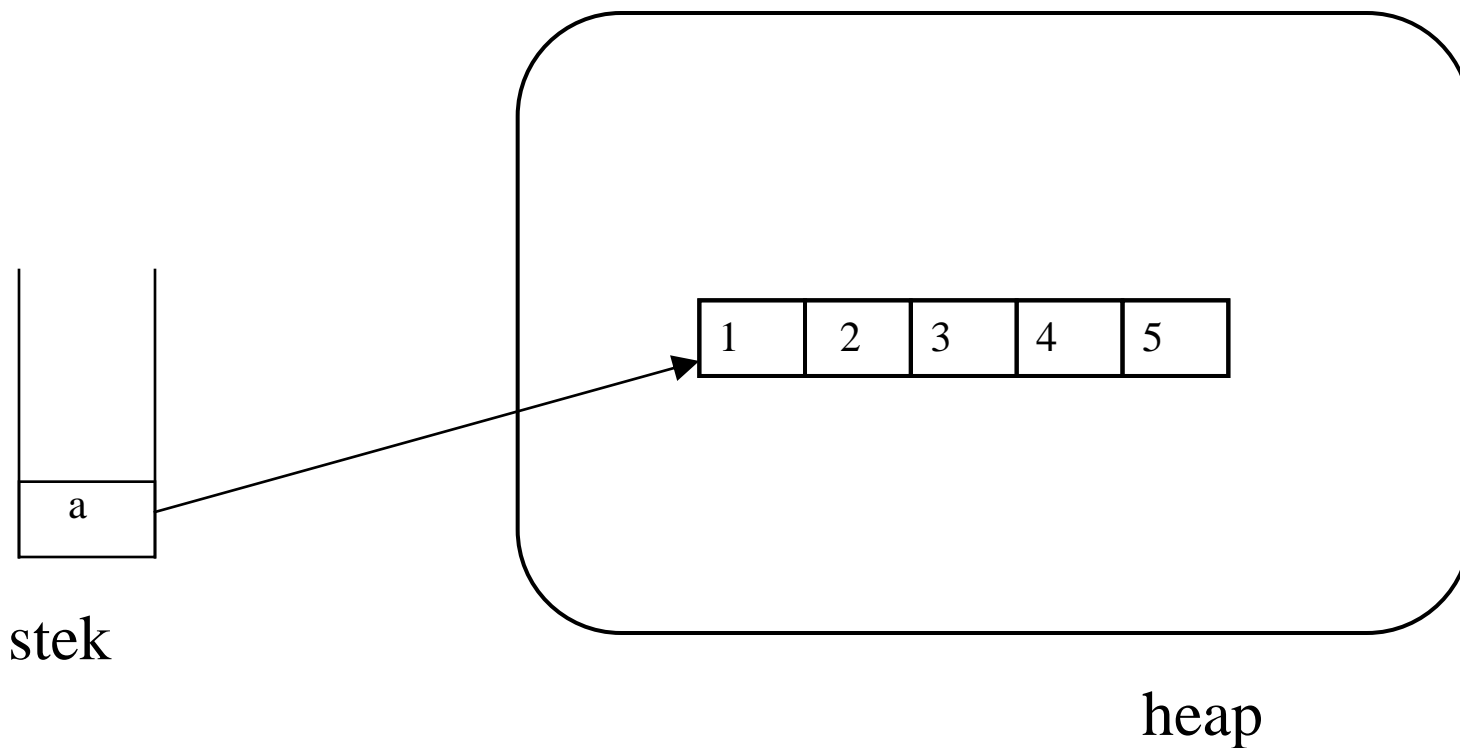
`a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5;`



# Nizovi primitivnih tipova

jednim potezom

```
int a[] = { 1, 2, 3, 4, 5 };
```



# Iteriranje kroz nizove

- ▶ Klasična for petlja:

```
int niz[] = {1, 2, 3, 4};  
for (int i = 0; i < niz.length; i++)  
    System.out.println(niz[i]);
```

- ▶ for-each petlja:

```
int niz[] = {1, 2, 3, 4};  
for (int el : niz)  
    System.out.println(el);
```

# Višedimenzijski nizovi

```
int a[][] = { {1, 2, 3 },  
              {4, 5, 6 } };
```

```
int a[][] = new int[2][3];
```

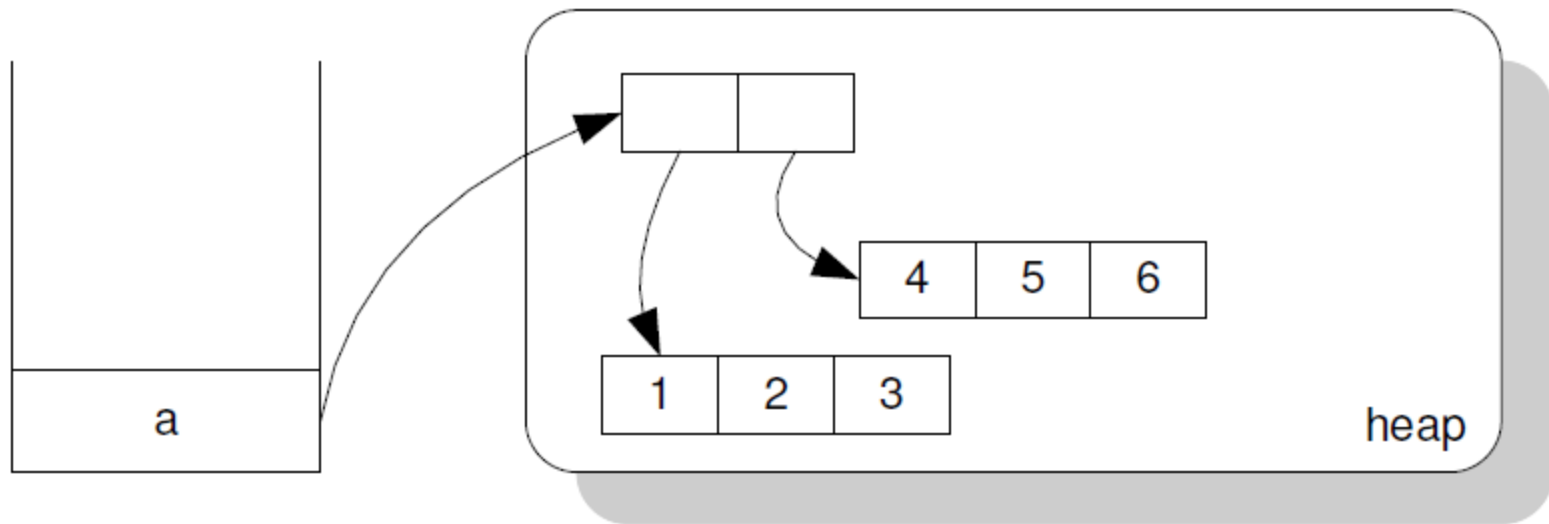
```
int a[][] = new int[2][];  
for(int i = 0; i < a.length; i++) {  
    a[i] = new int[3];  
}
```



# Višedimenzionalni nizovi

- ▶ Višedimenzionalni nizovi se predstavljaju kao nizovi nizova

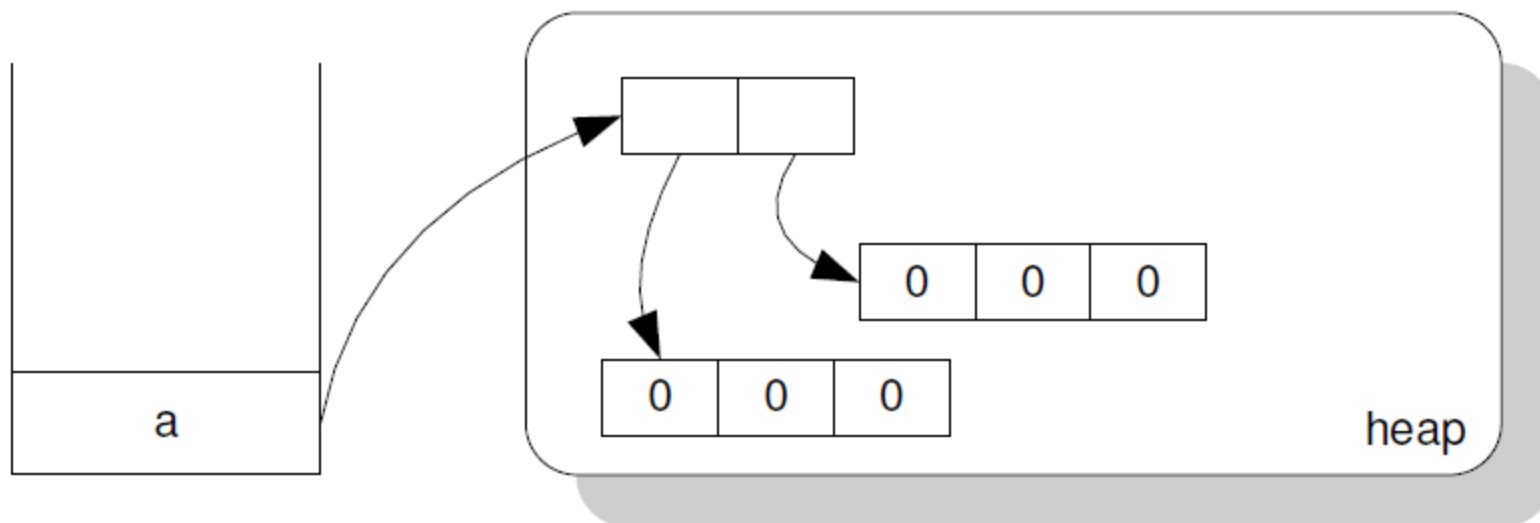
```
int[][] a = { {1, 2, 3}, {4, 5, 6} };
```



# Višedimenzionalni nizovi

- ▶ Višedimenzionalni niz se može kreirati i na sledeći način:

```
int[][] a = new int[2][3];
```



# Višedimenzionalni nizovi

- ▶ Dvodimenzionalni niz se može kreirati i postupno:

```
int[][] a = new int[2][];  
for (int i = 0; i < a.length; i++)  
    a[i] = new int[3];
```

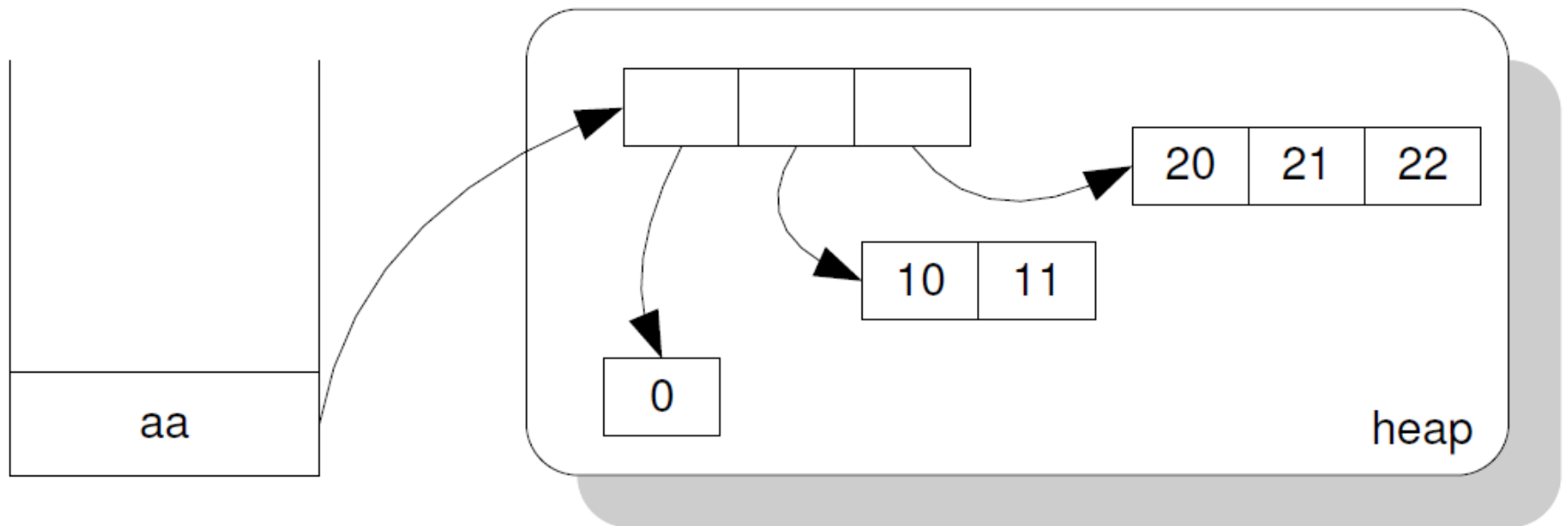
# Višedimenzijski nizovi

- ▶ Moguće napraviti dvodimenzijski niz koji ima različit broj kolona u svakoj vrsti:

```
int[][] aa = new int[3][];  
for (i = 0; i < aa.length; i++) {  
    aa[i] = new int[i + 1];  
    for (int j = 0; j < aa[i].length; j++)  
        aa[i][j] = i*10 + j;  
}
```

0
10 11
20 21 22

# Višedimenzionalni nizovi



# Iteriranje kroz višedimenzionalne nizove

- ▶ Klasična for petlja:

```
int[][] a = { {1, 2, 3}, {4, 5, 6} };  
for (int i = 0; i < a.length; i++) {  
    for (int j = 0; j < a[i].length; j++) {  
        System.out.println(a[i][j]);  
    }  
    System.out.println();  
}
```