Uvod

Programski jezici II

Java

- Java: platforma za izvršavanje programa (JVM – Java Virtuelna Mašina)
- Java: programski jezik
  - Java API (Application Programming Interface)

Java

- Java programski jezik je objektno-orijentisani jezik opšte namjene, posebno pogodan za razvoj konkurentnih, mrežnih i distribuiranih programa
- razvoj jezika, pod inicijalnim nazivom Oak, počeo je 1991. godine, a vodio ga je *James Gosling* iz kompanije *Sun Microsystems*
- prva verzija jezika, pod nazivom Java, objavljena je 1995. godine
  - JDK Alpha & Beta (1995)
  - JDK 1.0 (1996)
  - JDK 1.1 (1997)
  - J2SE 1.2 (1998)
  - J2SE 1.3 (2000)
  - J2SE 1.4 (2002)
  - J2SE 5.0 (2004)
  - Java SE 6 (2006)
  - Java SE 7 (2011)
  - Java SE 8 (2014)
  - Java SE 9 (2017)
  - novi način verzionisanja od sljedeće verzije: YY.M

Java

- Jedna od najvažnijih osobina - nezavisnost od platforme
  - ova osobina omogućava da se programi pisani u ovom programskom jeziku mogu kompajlirati na jednoj računarskoj platformi, a izvršavati na različitim računarskim platformama (Windows, Unix, Linux,...)

Java

- Od maja 2007. godine, u skladu sa specifikacijama JCP-a (Java Community Process), kompanija Sun Microsystems je većinu Java tehnologije stavila pod GNU GPL (General Public License) licencu i na taj način je učinila besplatnim softverom
  - Postoje i alternativne Java implementacije:
    - OpenJDK
    - IBM J9

Java

- Java interpreter, kompajler, kao i brojni razvojni alati grupisani su u JDK (Java Development Kit) i mogu se preuzeti sa zvaničnog Web sajta kompanije Oracle (ranije Sun Microsystems)
- U okviru JRE (Java Runtime Environment) ne nalaze se kompajler i razvojni alati, već samo Java interpreter.
- Kompanija Oracle (ranije Sun Microsystems) redovno objavljuje JDK i JRE pakete za različite platforme: Linux x86, Linux x64, Solaris x86, Solaris x64, Solaris SPARC, Solaris SPARC x64, MaxOS x64, Windows x86 Online, Windows x86 Offline i Windows x64.
- http://www.oracle.com/technetwork/java/javase/downloads/index.html

Java

- JVM specifikacija – specifikacija platforme za izvršavanje Java programa
- JVM – bazirana na programskom modelu izmišljenog procesora, posjeduje odgovarajući instrukcijski set i manipuliše memorijom u vrijeme izvršavanja programa
- softver za prilagođenje konkretnoj mašini i operativnom sistemu – interpreter
  - pored prevođenja bajt-koda u mašinski jezik, interpreter obavlja i funkciju izvršavanja prevedenih mašinskih instrukcija. Interpreter prevodi i izvršava instrukcije bajt-koda jednu za drugom, tj. ne vrši prevođenje kompletnog bajt-koda odjednom
- JVM specifikacija je dostupna
  - http://docs.oracle.com/javase/specs/
- Postoji više implementacija JVM
- “Java-like” VM:
  - JamVM
  - Dalvik

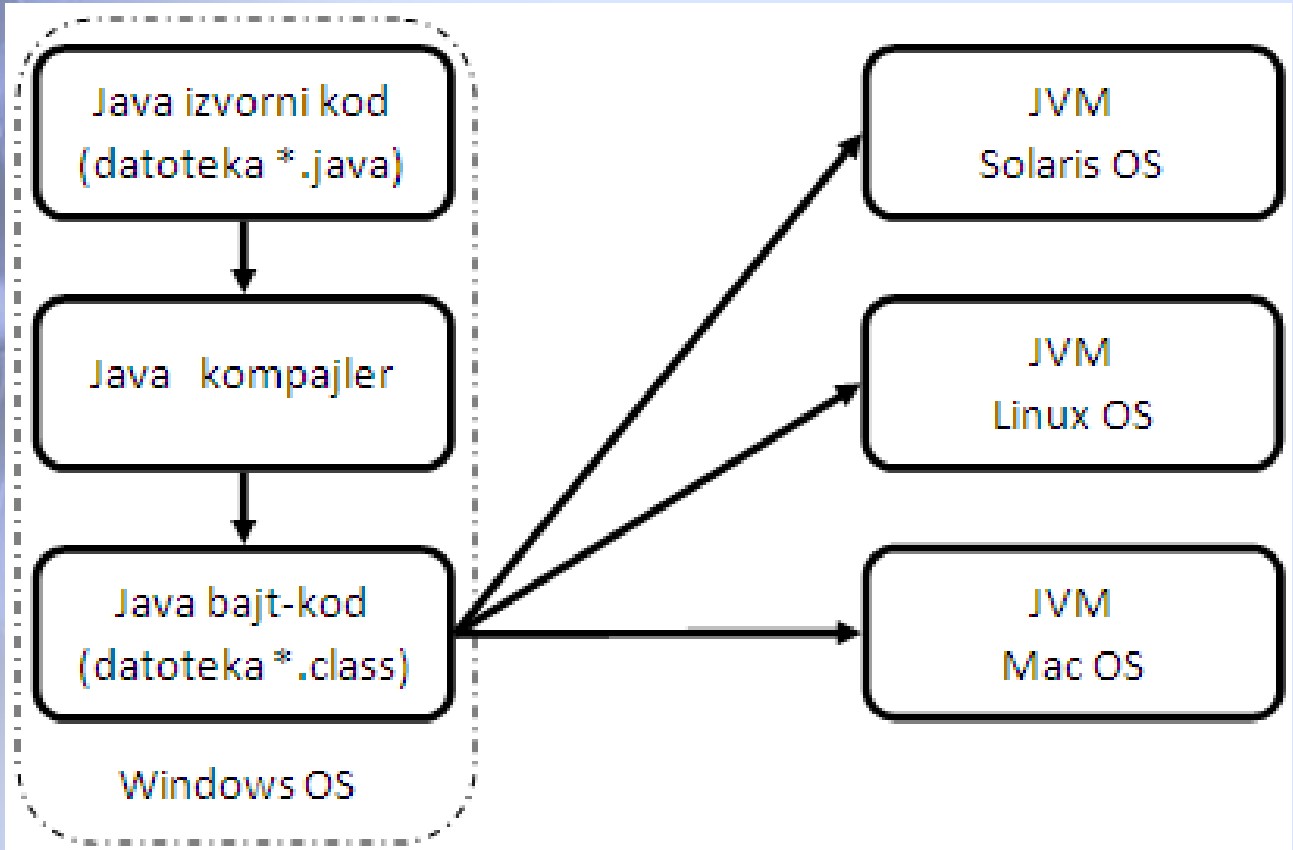
Java

- dizajnirana da što manje zavisi od specifičnih karakteristika konkretnog računarskog sistema
- jednom napisan i preveden program se izvršava na bilo kojoj platformi koja podržava Javu
- interpretirani jezik – pitanje brzine izvršavanja programa
- bajt-kod
  - specifikacija je dostupna – više implementacija kompajlera
  - visokooptimizovani mašinski jezik izmišljene računarske platforme – JVM

Java

- Prevedeni kod (bajt-kod) se smješta u datoteke sa ekstenzijom *.class*
- Ovaj kod se ne može izvršavati direktno na nekoj konkretnoj računarskoj platformi, već samo na JVM
- prenosivost programa pisanog Java programskim jezikom garantovana na nivou prevedenog (izvršnog) koda
- Java kod moguće je prevesti i za izvršavanje na nekoj konkretnoj računarskoj platformi
- Moguće je prevesti i programe pisane u drugim programskim jezicima u bajt-kod kako bi se mogli izvršavati u okviru Java virtuelne mašine

Java


Java izvorni kod
(datoteka *.java)
Java kompajler
Java bajt-kod
(datoteka *.class)
Windows OS
JVM
Solaris OS
JVM
Linux OS
JVM
Mac OS

Java

- dvije osnovne vrste Java programa
  - aplikacije
    - izvršavaju se kao regularne aplikacije
    - neograničene mogućnosti i pristup
  - apleti
    - izvršavaju se u okviru WWW čitača
    - automatska distribucija i instalacija
    - ograničene mogućnosti apleta iz razloga sigurnosti
- jezik opšte namjene, objektno orijentisan
- konkurentni, mrežni i distribuirani programi

Osnovni koncepti

- Objektno-orijentisan jezik:
  - atributi: promjenljive unutar klase
  - metode: funkcije i procedure unutar klase
- Klasa – model objekta
  - apstrakcija zajedničkih atributa i zajedničkog ponašanja jednog skupa srodnih objekata
- Objekat – primjerak, instanca klase

Osnovni koncepti

- sintaksa: podsjeća na C++
  - programski blok je ograđen vitičastim zagradama: { ... }
- tipovi podataka
  - primitivni tipovi
    - čuvaju se na steku
    - kao parametri, uvijek se prenose po vrijednosti!
  - objekti
    - čuvaju se na heap-u
    - postoje samo reference na objekte, nikada se ne može pristupiti samom objektu
    - reference se čuvaju na steku
- metode: povratna_vrednost naziv(parametri) { }

Leksička struktura

- Kao i drugi programski jezici, Java programski jezik je definisan gramatičkim pravilima koja specificiraju kako se mogu formirati sintaksno pravilne konstrukcije koristeći elemente jezika, i semantičkom definicijom koja specificira značenje sintaksno legalnih konstrukcija
- Java podržava pisanje programskog koda korištenjem Unicode skupa karaktera
- Unicode karakteri dobijeni leksičkom translacijom izvornog koda se razdvajaju na sekvence ulaznih elemenata
- Ulazni elementi su leksički tokeni, prazni prostori (eng. white space) i komentari

Leksički tokeni

- Leksički tokeni (tokeni) predstavljaju gradivne blokove složenijih konstrukcija
- U tokene se ubrajaju identifikatori, ključne riječi, literali, separatori i operatori
- Tokeni se koriste za izgradnju složenijih konstrukcija poput izraza, naredbi, metoda i klasa
- Tokeni u Java programskom jeziku slični su kao i u mnogim drugim jezicima opšte namjene

Identifikatori

- Identifikatori – koriste se za označavanje naziva klasa, metoda, promjenljivih i labela
- Identifikatori u Java programskom jeziku moraju biti sačinjeni od slova, brojeva i znaka donje crte (*underscore*), s tim što prvi karakter ne smije biti broj
- Identifikator ne može biti sačinjen od sekvence karaktera koja je jednaka sekvenci karaktera ključne riječi jezika, boolean literala ili null literala
- Znakovi koji označavaju valute ($, ¢, ¥, ili £) su dozvoljeni, ali su rezervisani za posebne namjene, pa se ne preporučuje njihovo korištenje za ovu namjenu
- Primjeri validnih identifikatora
  - a abc a1 abc123 _a _abc a_1 abc_123 abc123def abcDEF
  - A ABC A1 ABC123 _A _ABC A_1 ABC_123 ABC123DEF ABCdef
- Preporuka je da se kod izbora naziva identifikatora vodi računa o tome šta taj identifikator označava, pa mu je u skladu s tim potrebno dati odgovarajući, smislen naziv

Identifikatori

- Primjeri nevalidnih identifikatora
  - abc.java ABC.CLASS abc-1 %abc
  - 123 123_abc 1A 123ABC
  - abstract boolean long int
- Java programski jezik razlikuje velika i mala slova u nazivima identifikatora – Java je *case-sensitive* jezik
- Primjeri
  - abc aBc abC aBC
  - ABC AbC ABc Abc

Ključne riječi

- Ključne riječi – rezervisane riječi koje su predefinisane u jeziku i ne mogu se koristiti kao identifikatori

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

- const i goto su, takođe, ključne riječi, ali se ne koriste
- Sve ključne riječi se pišu malim slovima i njihovo nepravilno korištenje rezultira greškama pri kompajliranju

Literali

- Literal označava konstantnu vrijednost
- Vrijednost koju literal predstavlja ne mijenja se tokom izvršavanja programa
- Literali su vrijednosti primitivnog tipa (numeričke, karakter i boolean), string vrijednosti i null literal koji predstavlja null referencu
- Cjelobrojni tipovi podataka su: int, long, byte i short
- Podrazumijevani tip podatka cjelobrojnog literala je int
- Tip long se može specificirati dodavanjem sufiksa L (ili l) cjelobrojnoj vrijednosti
  - Preporuka je da se koristi sufiks L, zbog velike sličnosti sufiksa l i broja 1
- Ne postoji direktan način za specificiranje short i byte literala
- Pored decimalnog brojnog sistema, cjelobrojni literali mogu biti specificirani i u binarnom, oktalnom i heksadecimalnom brojnom sistemu, navođenjem prefiksa 0b, 0 i 0x (ili 0X), respektivno
- Primjer
  - 0b1000 8 010 0x8

Literali

- Tipovi podataka sa pokretnim zarezom (floating-point tipovi) su float i double
- Podrazumijevani tip podatka floating-point literala je double
- Tip double se može i eksplicitno specificirati dodavanjem sufiksa D (ili d) datoj vrijednosti
- Tip float se može specificirati dodavanjem sufiksa F (ili f) datoj vrijednosti
- Primjeri:
  - 0.0 0.0d 0D 0.0F 0f
- Primitivni tip podataka boolean može imati true i false vrijednosti koje su označene rezervisanim literalima true i false
- Karakter literal se navodi između jednostrukih znakova navoda (‘)
  - Svi karakter literali su primitivnog tipa char i predstavljeni su prema 16-bitnom Unicode karakter setu

Literali

- Numerički literali sa znakom “_”
- Radi lakšeg uočavanja brojne vrijednosti koristi se znak “_”
- 1000000 = 1_000_000
- 1023040025 = 1_023_040_025

Separatori i operatori

- Separatori:
  - ( ) { } [ ] ; , .
- Operatori

=	>	<	!	~	?	:			
==	<=	>=	!=	&&	||	++	--		
+	-	*	/	&	|	^	%	<<	>>
+=	-=	*=	/=	&=	|=	^=	%=	<<=	>>=
>>>	>>>=								

Prazni prostori i komentari

- Prazan prostor (eng. white space) je sekvenca bjelina, tabova i karaktera koji označavaju kraj linije u Java izvornoj datoteci
- Karakteri koji označavaju kraj linije mogu biti NL (newline), CR (carriage return) ili CR-NL (carriage return-newline) sekvenca
- Prazan prostor se koristi za razdvajanje tokena, ali i za formatiranje programa radi njegovog jednostavnijeg čitanja
- Leksičkom strukturom Java programskog jezika definisana su dva tipa komentara:
  - jednolinijski i
  - višelinijski
- Pored ova dva tipa komentara, postoji i javadoc komentar koji nije definisan leksičkom strukturom Java programskog jezika

Tipovi promjenjivih

- Promjenjive instance (nestatička polja)
  - Njihove vrijednosti su jedinstvene za svaku instancu klase (za svaki objekat)
- Promjenjive klase (statička polja)
  - Postoji samo jedna kopija ove promjenjive, bez obzira na to koliko instanci klase postoji
  - Ključna riječ static
- Lokalne promjenjive
  - Slično tome kako objekat čuva svoje stanje u promjenjivim instance, tako i metoda može da čuva privremeno stanje u lokalnim promjenjivim
  - deklarišu se u metodama, konstruktorima i blokovima
  - Dostupne su samo od deklaracije do zatvarajuće zagrade metode, nisu dostupne iz ostatka klase
- Parametri
  - Parametri se uvijek klasifikuju kao „promjenjive“, ne kao „polja“

Primitivni tipovi podataka

Primitivni tipovi

- U primitivne tipove ubrajaju se: cjelobrojni tipovi, tip za karaktere, tipovi sa pokretnim zarezom i boolean tip
- Cjelobrojni tipovi su byte, short, int i long i oni su označeni tipovi (eng. signed)
- Tip za karaktere, char, predstavlja jedan karakter – ovaj tip je neoznačeni (eng. unsigned) tip
- Tipovi sa pokretnim zarezom su float i double
- Logičke vrijednosti true i false predstavljene tipom boolean

Primitivni tip	Veličina	Minimum	Maksimum
boolean	1-bit	–	–
char	16-bit	Unicode 0	Unicode 2^{16}- 1
byte	8-bit	-128	+127
short	16-bit	-2^{15}	$+2^{15} - 1$
int	32-bit	-2^{31}	$+2^{31} - 1$
long	64-bit	-2^{63}	$+2^{63} - 1$
float	32-bit	IEEE 754	IEEE 754
double	64-bit	IEEE 754	IEEE 754
void	–	–	–

Deklaracija promjenljive primitivnog tipa

- Deklaracijom promjenljive se specificira njen tip i naziv
- Ovo implicitno određuje memorijsku alokaciju za datu promjenljivu, ali i vrijednosti koje mogu biti smještene u promjenljivu
- Promjenljiva se može deklarisati u bilo kom bloku – ne mora na početku metode.

```
int a;

int a = 0;

int a, b;

int a = 0, b = 3;
```

Implicitna konverzija tipova

- sa “užeg” ili “manjeg” tipa na “širi” ili “veći” tip
- nema gubitka informacije, jer “uži” tip podatka staje u “širi” tip podatka
- byte —> short —> int —> long —> float —> double

- primjer:

```
long a;
int i = 5;
a = i;
```

Eksplicitna konverzija tipova

- Sa “šireg” na “uži” tip podatka – posljedica je gubljenje informacije.
- Primjer:

```
long a = 5L;
int b = a;
```

Greška pri kompajliranju!

Eksplicitna konverzija tipova

- Pravilna eksplicitna konverzija – upotreba cast operatora:
- Primjer:

```
long a = 5L;
int b = (int)a;
```

Klase i objekti

Osnovni koncepti – klase i objekti

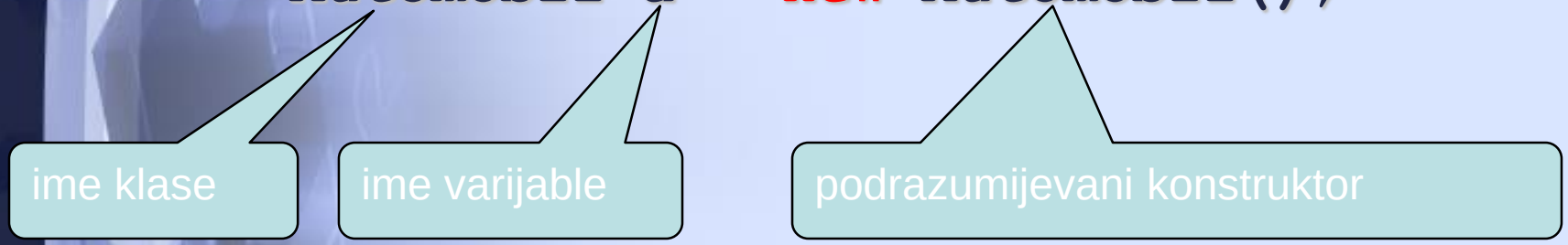
- klasa: model objekta
  - obuhvata:
    - atribute
    - metode
- objekat: primjerak, instanca klase

Primjer klase

```
class Automobil {
  boolean radi;

  void upali() {
    radi = true;
  }

  void ugasi() {
    radi = false;
  }
}
```

Konstruktor

```
Automobil a = new Automobil();
```



- svaki se objekat prije upotrebe mora konstruisati pomoću operatora new koji poziva konstruktor objekta.
- operator new:
  - rezeriše prostor u memoriji za novi objekat
  - poziva specificirani konstruktor
  - vraća referencu na novi objekat

Konstruktor

- konstruktori su specijalne metode
  - ime konstruktora jednako imenu klase
  - nemaju povratni tip
  - koriste se za inicijalizaciju promjenljivih
  - mogu imati parametre
  - “normalno” tijelo metode
  - može ih biti više u jednoj klasi
- ako nema eksplicitno napisanog niti jednog konstruktora, onda prevodilac sam dodaje podrazumijevani konstruktor

Korištenje instanci

- Poziv metoda instance:
  - a.upali()
  - a.ugasi()
- Pristup varijablama instance:
  - a.radi

Sve je objekat

- nije moguće definisati funkcije i promjenljive izvan neke klase, tj. nema globalnih promjenljivih, funkcija ili procedura
- ne postoje odvojene deklaracija i definicija klase, već samo njena definicija
- ne postoje *header* fajlovi, kao u C++

Sve je objekat

- Objekti imaju stanje
- Primjer: osoba je objekat koji ima ime, godine, pol...
- Objektima se šalju poruke (poziv metode)
  - Programer: Koliko si star?
  - Objekat: Ja imam 22 godine.
  - Programer: Kako se zoveš?
  - Objekat: Ja se zovem Marko.

Izvršavanje programa

- metoda main() – osnovna funkcija od koje počinje izvršavanje programa (kao u C/C++), ali mora biti metoda klase

Hello.java

```
class Hello {
  public static void main(String args[]) {
      System.out.println("Hello world!");
    }
}
```

Prevođenje i pokretanje

- jedna klasa – jedna datoteka, ime jednako imenu klase, ekstenzija .java
- prevođenje:

```
javac Hello.java
```

- rezultat:

```
Hello.class
```

- pokretanje:

```
java Hello
```

- [ovo važi sa standardni razvojni paket JDK (Java Development Kit)]

Primjer: program sa dvije klase

Automobil.java

```
class Automobil {
  boolean radi;
  void upali() { radi = true; }
  void ugasi() { radi = false; }
}
```

Test.java

```
class Test {
  public static void main(String args[]) {
    Automobil a;
    a = new Automobil();
    a.upali();
  }
}
```

- pokretanje: java Test
- više klasa može posjedovati metodu main

Reference na objekte

```
Automobil a; //deklaracija promjenljive
a = new Automobil();
```

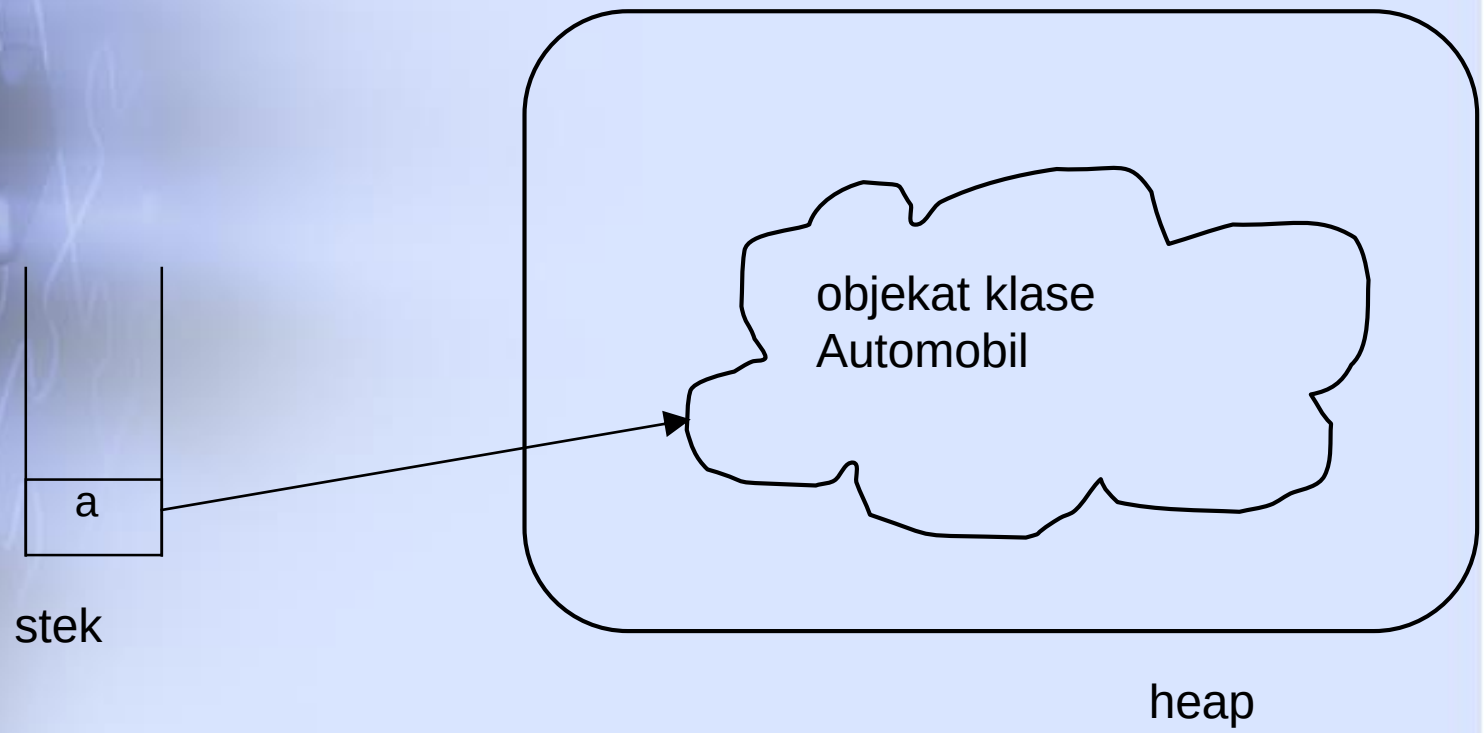
- promenljiva a nije objekat, već referenca na objekat

Reference na objekte

- manipulacija objektima - putem referenci

```
Automobil a;
a.upali(); - greška
==============================
Automobil a = new Automobil();
a.upali();
```

Reference na objekte



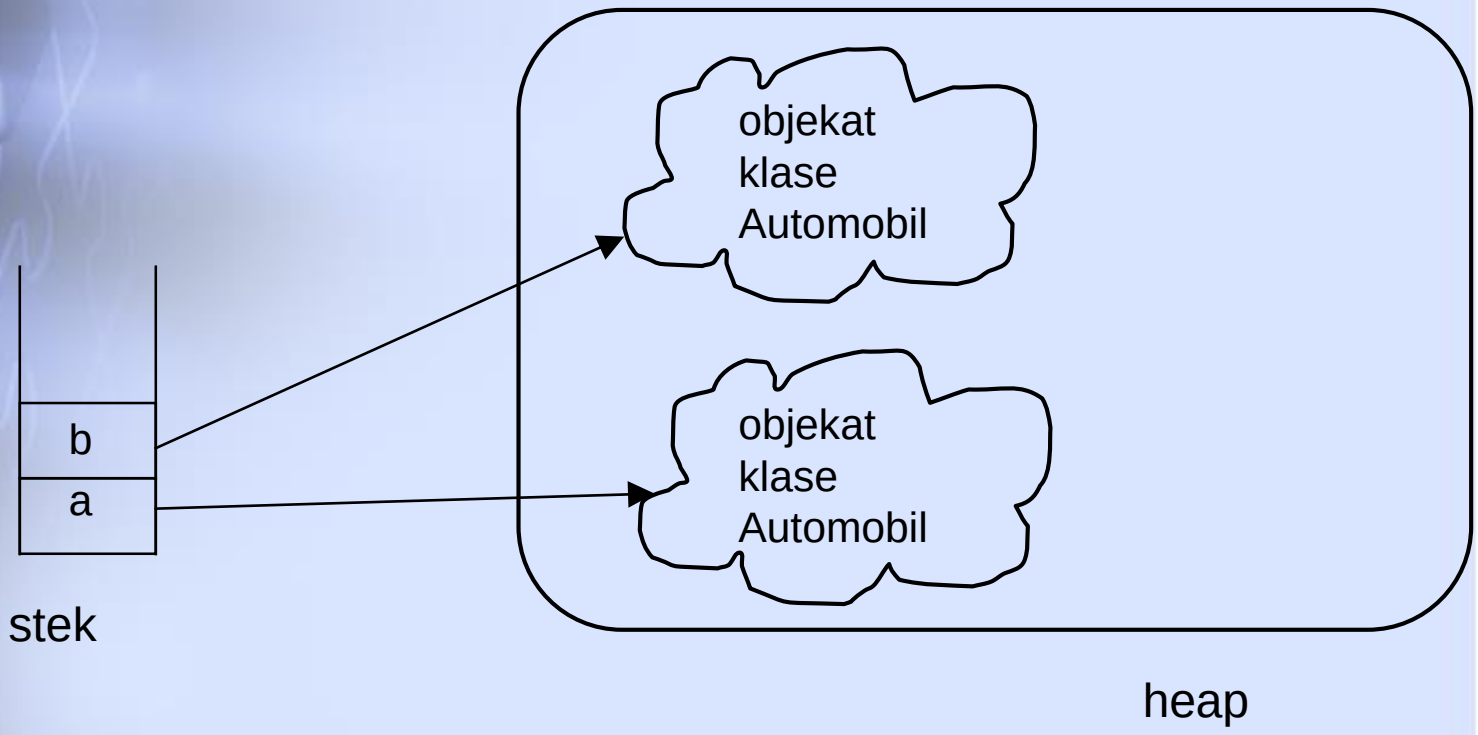
Operator dodjele vrijednosti

```
Automobil a = new Automobil();
Automobil b = new Automobil();
b = a;
```

Vrši se kopiranje vrijednosti reference!

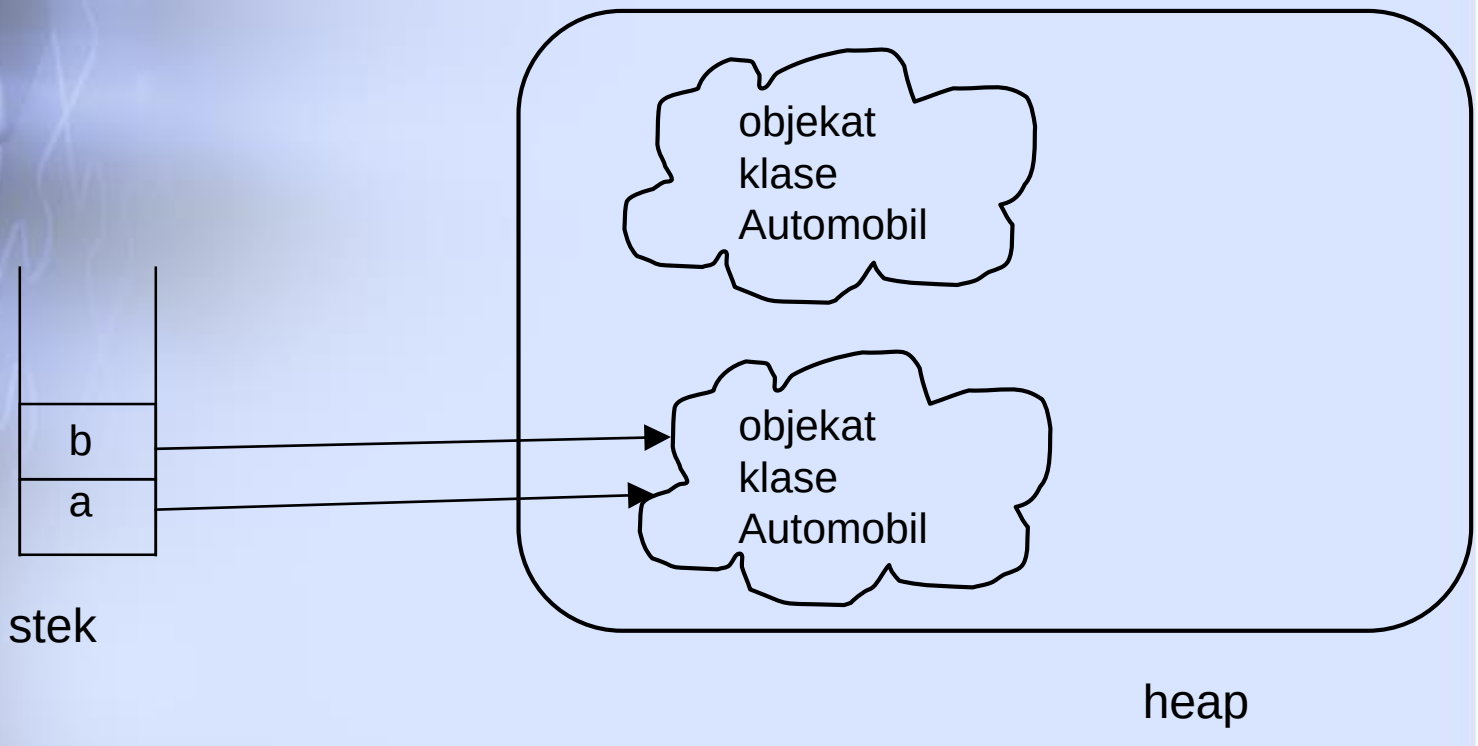
Reference na objekte

```
Automobil a = new Automobil();
Automobil b = new Automobil();
```



Reference na objekte

```
b = a;
```



Konverzija referenci

- Relacija podtip-supertip između tipova referenci određuje koje konverzije su moguće među njima
- Konverzija „prema gore“ kroz hijerarhiju tipova (hijerarhija klasa) naziva se konverzija proširivanja reference (eng. *upcasting*), tj. riječ je o konverziji iz podtipa u supertip

```
public class KonverzijaReference {
	public static void main(String args[]){
		String tekst = "string";
		Object objekat = tekst;			// 1
		String tekst2 = (String) objekat;	// 2
	}
}
```

- Konverzija „prema dolje“ kroz hijerarhiju tipova naziva se konverzija sužavanja reference (eng. narrowing), tj. riječ je o konverziji iz supertipa u podtip
- Sužavanje reference podrazumijeva upotrebu operatora kastovanja, dok se proširivanje reference obično radi implicitno

Konverzija referenci

- U slučaju da pokušaj kastovanja nije legalan, desiće se greška pri kompajliranju
- Konverzija proširivanja reference nikad ne rezultira bacanjem izuzetka
- Kod konverzije sužavanja reference može doći do bacanja ClassCastException izuzetka u slučaju da konverzija nije legalna

Inicijalizacioni blokovi

- Inicijalizacioni blok je blok koda između separatora „{“ i „}“ koji se izvršava pri kreiranju objekta klase
- Dvije vrste inicijalizacionih blokova:
  - statički i
  - nestatički.
- Statički inicijalizacioni blok
  - izvršava se jedanput, prilikom učitavanja klase. Ovaj blok je označen ključnom rječju static prije separatora „{“, a u njemu se mogu inicijalizovati samo statički atribute klase.
- Nestatički inicijalizacioni blok
  - izvršava se za svaki objekat koji se kreira i inicijalizuje nestatičke atribute klase.
- Iako je moguće, ne preporučuje se inicijalizacija statičkih atributa klase nestatičkim inicijalizacionim blokom.

Podrazumijevane vrijednosti

- Svaka promjenljiva klase, promjenljiva instance ili element niza inicijalizovana je podrazumijevanom vrijednošću

Tip	Podrazumijevana vrijednost
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0F
double	0.0D
reference	null

- Ako promjenljiva klase nije inicijalizovana pri deklaraciji ili u statičkom inicijalizacionom bloku, onda se inicijalizuje podrazumijevanom vrijednošću pri učitavanju klase
- Ako promjenljiva instance nije incijalizovana pri deklaraciji ili u inicijalizacionom bloku instance, onda je inicijalizovana podrazumijevanom vrijednošću pri instanciranju objekta klase
- Polja koja su reference se uvijek inicijalizuju null referencom, ako eksplicitno nije izvršena njihova inicijalizacija

Inicijalizacija lokalnih promjenljivih primitivnog tipa

- Lokalne promjenljive su promjenljive koje se deklarišu u metodama, konstruktorima i blokovima
- Lokalne promjenljive se ne inicijalizuju prilikom njihovog kreiranja kod poziva metode, tj. kada započne izvršavanje metode

```
public class Primjer {
      public static void main(String[] args) {
            int broj1 = 10,
            int broj2;                                    // 1
            if (broj1 < 10)
                  broj2 = 100;
            if (broj1 >= 10)
                  broj2 = 1000;
            System.out.println("Broj: " + broj2);        // 2
      }
}
```

int broj2 = 0;

- Isto važi i za konstruktore i blokove
- Kompajler će kao grešku prijaviti svaki pokušaj korištenja neinicijalizovane promjenljive
- moraju biti inicijalizivane prije korišćenja – u suprotnom kompajler će prijaviti grešku

Inicijalizacija lokalnih referenci

- Za lokalne reference važe ista pravila kao i za lokalne promjenljive primitivnog tipa, s tim što je lokalnu referencu moguće inicijalizovati null referencom, što može dovesti do greške za vrijeme izvršavanja programa

```
public class KalkulatorTest {
      public static void main(String[] args) {
            Kalkulator kalkulator;  // 1
            kalkulator.zbir();      // 2
      }
}
```

Kalkulator kalkulator = null;
NullPointerException

Kalkulator kalkulator = new Kalkulator();

Parametri i rezultat metoda

- sintaksa definisanja metoda slična kao u C++:
  - primjer:

```
void metoda (String s, int i, boolean b){…}
```

- parametri mogu biti:
  - primitivni tipovi
  - reference na objekte
- rezultat može biti:
  - primitivni tip
  - referenca na objekat
- Metoda vraća vrijednost naredbom:

```
return vrijednost
```

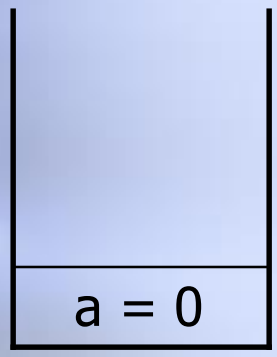
Prenos parametra po vrijednosti

- primitivni tipovi kao parametri metoda

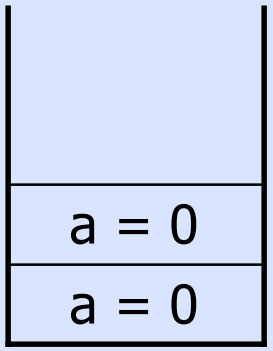
```
void test(int a) {
  a = 1; // c)
}

...
int a = 0;// deklaracija i inicijalizacija – a)
test(a); // b)
System.out.println(a); //  d)
```

Prenos parametra po vrijednosti

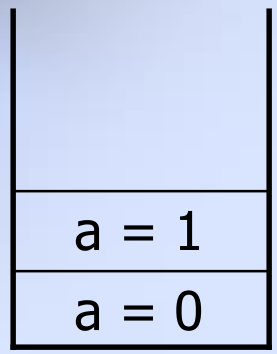
a



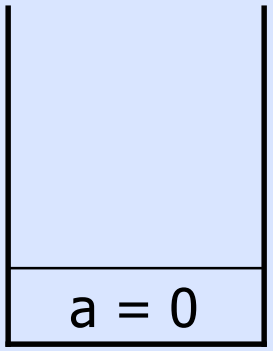
b



c



d

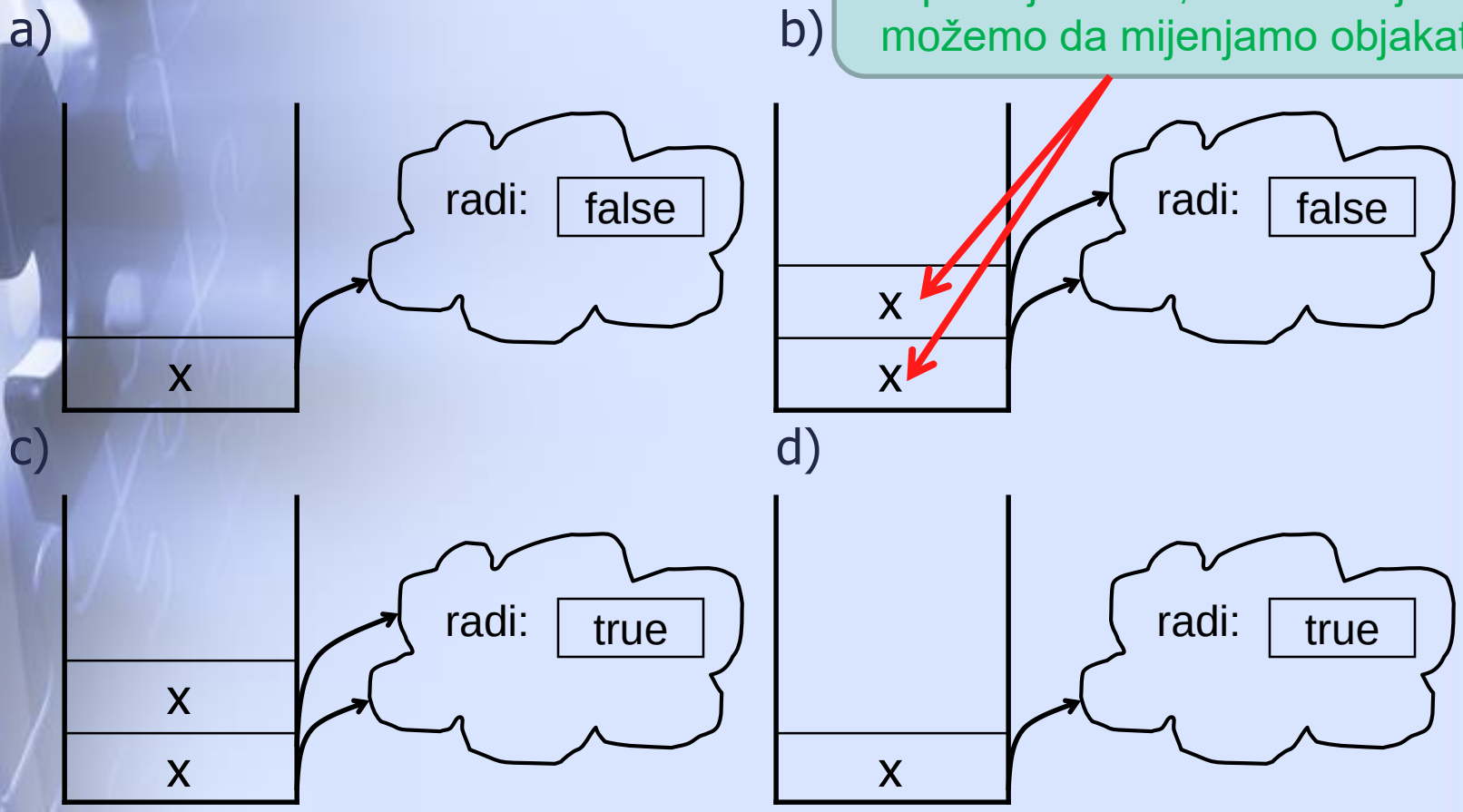


Prenos parametra po referenci

```
void test(Automobil a) {
  a.radi = true;                    c)
}

...
Automobil x = new Automobil();// referenca x
   na objekat klase Automobil   a)
x.radi = false;
test(x);                            b)
System.out.println(x.radi)          d)
```

Prenos parametra po referenci

reference na objekte se prenose po vrijednosti, ali mi i dalje možemo da mijenjamo objakat



Operatori

- operatori su specijalni simboli koji vrše određene operacije nad jednim, dva ili tri operanda i vraćaju odgovarajući rezultat
- aritmetički operatori
- relacioni operatori
- logički operatori
- bit-operatori
- operator dodjele

- razlika u odnosu na C++: postoji primitivni tip boolean; rezultat relacionih ili logičkih operatora je boolean vrijednost

Prioritet operatora

- prioritet operatora definiše kako se izraz izračunava kada je prisutno više operatora
- lista prioriteta operatora!!!
- najjednostavnije pravilo – množenje i dijeljenje vrše se prije sabiranja i oduzimanja
- dobra praksa – korištenje zagrada

```
public class Precedence {
   public static void main(String[] args){
       int x = 1, y = 2, z = 3;
       int a = x + y - 2/2 + z:
       int b = x + (y - 2)/(2 + z):
       System.out.println("a = " + a  + "b = "
   + b};}}
a = 5 b = 1
```

Aritmetički operatori

- osnovne operacije:

+, -, *, /, %

- umjesto x = x + 1

x += 1

- automatski inkrement:
  - ++x - pre-inkrement
  - x++ - post-inkrement

Unarni + i unarni - operatori

- unarni minus (-) i unarni plus (+) su isti operatori kao i binarni minus i plus
- kompajler prepoznaje namjenu na osnovu toga kako je izraz napisan

```
x = -a
```

- kompajler će prepoznati i:

```
x = a * -b
```

- ali je preporuka:

```
x = a * (-b)
```

- unarni minus invertuje znak
- unarni plus – promocija byte, short i char u int

Relacioni i logički operatori

- relacioni: < > <= >= == !=
- primitivni tipovi

Relacioni i logički operatori

```
public class Equivalence {
  public static void main(String[] args) {
      Integer n1 = new Integer(47):
      Integer n2 = new Integer(47):
      System.out.println(n1 == n2);
      System.out.println(n1 != n2);
  }
}
```

Relacioni i logički operatori

```
public class Equivalence {
  public static void main(String[] args) {
      Integer n1 = new Integer(47):
      Integer n2 = new Integer(47):
      System.out.println(n1 == n2);
      System.out.println(n1 != n2);
  }
}
false
true
```

Relacioni i logički operatori

```
public class EqualsMethod {
  public static void main(String[] args) {
      Integer n1 = new Integer(47 );
      Integer n2 = new Integer(47);
      System.out.println(nl.equals(n2));
  }
}
```

```
true
```

Relacioni i logički operatori

- logički: && (I), || (ILI), ! (NE)
- logički operatori proizvode boolean vrijednost true ili false na bazi logičke veze argumenata
- mogu se primijeniti samo na boolean vrijednosti
- short-circuiting:

```
if(test1() && test2() && test3())
```

- ako npr. **test2()** bude false, **test3()** se neće ni pozvati!
- bolje performanse

Bit operatori

- Logičko I nad bitovima: &
  - ako su oba ulazna bita 1, daje 1
- Logičko ILI nad bitovima: |
  - ako su oba ulazna bita 0, daje 0
- Ekskluzivno ILI (XOR) nad bitovima: ^
  - ako su ulazni biti različiti daje 1
- Logička negacija nad bitovima (komplement)-unarni operator: ~
- Kombinacija sa =:

  &= |= ^=
- Nad boolean tipom (1 bit) mogu se primijeniti svi osim negacije i imaju isti efekat kao i logički operatori

Bit operatori

- shift-ovanje (pomjeranje):

  a>>b – pomjera bitove u a za b mjesta
  - ako je a pozitivan, ubacuje 0
  - ako je a negativan, ubacuje 1

  a<<b – pomjera bitove u lijevo i ubacuje 0

  a>>>b – pomjera bitove u a u desno za b mjesta i ubacuje 0 bez obzira na znak a

Operator dodjele

- operator =
- vrijednost sa desne strane (rvalue) se kopira na lijevu stranu (lvalue) - rvalue je konstanta, varijabla ili izraz koji proizvodi vrijednost, lvalue je varijabla
- Ako su operandi primitivni tipovi, kopira se sadržaj:

```
int i = 3, j = 6;
i = j;  //u i ubačeno 6
3 = i;
```

- Ako su operandi reference, kopira se sadržaj reference, a ne kompletni objekti koje referenciraju!

Redefinisan + operator sa stringovima

- Ako je jedan od operanada klase String, cio izraz je String!

```
int i = 5;
String a = “Vrijednost i je: “ + i;
```

Kontrola toka

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

if else

```
int result = 0;
if(testval > target)
   result = -1;
else if(testval < target)
   result = +1;
else
   result = 0; // match
```

Ternarni if operator

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

- isto kao:

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

switch

- switch() radi sa byte, short, char, i int primitivnim tipovima (odgovarajućim okružujućim klasama Byte, Short, Character i Integer) podataka, sa *enum* tipovima i klasom String
- U drugim slučajevima koristi se if-else
- Ako se izostavi break, dolazi do propadanja u sljedeći case
- Kod default izraza ne mora break; - to se podrazumijeva

```
switch(c) {
  case 'a':
    System.out.println("a");
    break;
  case 'b':
    System.out.println("b");
    break;
  case 'c':
    System.out.println("c");
    break;
  default:
    System.out.println("default");
}
```

for

```
for (int i = 0; i < 10; i++)
   System.out.println(i);
```

- može i višestruka inicijalizacija, ali promjenljive moraju biti istog tipa:

```
for(int i = 0, j = 1;
i < 10 && j != 11;i++, j++)
```

while

```
double r = 0;
while(r < 0.99d) {
   r = Math.random();
   System.out.println(r);
}
```

- Važno: izlaz iz petlje na false!

do while

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

- važno: izlaz iz petlje na false!
- uvijek se izvršava bar jedanput
- u praksi se rjeđe koristi nego while

break i continue

- break – prekida tijelo tekuće ciklične strukture (ili case dijela) i izlazi iz nje
- continue – prekida tijelo tekuće ciklične strukture i otpočinje sljedeću iteraciju petlje

break i continue

```
for(int i = 0; i < 100; i++) {
  // izlaz iz for petlje
  if(i == 74) break;
  // sljedeća iteracija
  if(i % 9 != 0) continue;
  System.out.println(i);
}
```

Uništavanje objekata

- automatski, pozadinski proces (garbage collector - GC)
- radi nezavisno od pokrenutog programa
- ne postoji destruktor
- posebna metoda finalize() se poziva neposredno prije oslobađanja memorije koju je objekat zauzimao
- ne treba je koristiti za oslobađanje zauzetih resursa (otvorene datoteke, mrežne konekcije i dr.)

metoda finalize()

```
class A {
   A(){
       System.out.println(“konstruktor”);
   }
   protected void finalize() throws Throwable
   {
       System.out.println(“finalize”);
   }
   public static void main(String args[]){
       A a = new A();
       System.out.println(“main”);
   }
}
```

Apstrakcija

- Jedan od osnovnih načina rješavanja problema kompleksnosti je apstrakcija
- Apstrakcija određuje osnovne osobine i ponašanja jednog objekta koja ga razlikuju od ostalih objekata
- Apstrakcija predstavlja zanemarivanje nebitnih osobina objekata u zavisnosti od trenutnih potreba
- Apstrakcija – uprošćeni opis ili specifikacija sistema koja naglašava neke od detalja ili osobina, dok druge detalje ili osobine zanemaruje
- Za istu vrstu objekata stvarnog svijeta u model mogu da se unose različiti skupovi osobina, u zavisnosti od stvarnih potreba
- Osnova objektno-orijentisanog programiranja je modelovanje apstrakcije, pomoću klasa i objekata.
- Najteži dio ovog procesa jeste pronalazak najbolje apstrakcije.

Enkapsulacija

- Enkapsulacija je proces sakrivanja onih elemenata apstrakcije koji definišu strukturu i ponašanje
- Enkapsulacija služi da razdvoji konceptualni interfejs od implementacije apstrakcije
- Enkapsulacija se postiže tako što se izlažu samo one metode i atributi klase koji su neophodni za njeno funkcionisanje, dok se sve ostale metode i atributi klase sakrivaju
- Zahvaljujući enkapsulaciji, moguće je sakriti implementaciju klase i mijenjati način na koji je ta implementacija izvršena, bez dodatnih izmjena dijelova programa koji koriste datu klasu, odnosno objekte date klase

Deklaracija klase

```
<modifikator za deklaraciju top-level
  tipova> class <naziv klase> <lista
  tipova> <extends klauzula> <implements
  klauzula>     // zaglavlje klase
{ // tijelo klase
   <deklaracije polja>
   <deklaracije metoda>
   <deklaracije ugnježdenih klasa>
   <deklaracije ugnježdenih interfejsa>
   <deklaracije ugnježdenih enumeracija>
   <deklaracije konstruktora>
   <inicijalizacioni blok>
}
```

Deklaracija klase

- Potrebno je napraviti razliku između statičkog i nestatičkog konteksta
- Statički kontekst definisan je statičkim metodama, statičkim poljima i statičkim inicijalizacionim blokovima
- Nestatički kontekst definisan je metodama instance, konstruktorima, nestatičkim poljima i inicijalizacionim blokovima instance (nestatički inicijalizacioni blokovi)
- Pod statičkim kodom podrazumijevaju se izrazi i naredbe u statičkom kontekstu
- Pod nestatičkim kodom podrazumijevaju se izrazi i naredbe u nestatičkom kontekstu

Deklaracija metode

```
<modifikator metode> <lista parametara
   formalnog tipa> <povratni tip> <naziv
   metode>(<lista parametara>) <throws
   klauzula> // zaglavlje metode
{ // tijelo metode
    <deklaracije lokalnih promjenljivih>
    <deklaracije ugnježdenih lokalnih klasa>
    <naredbe>
}
```

Deklaracija metode

- Lista parametara metode je lista parametara razdvojenih zarezima
- Ovi parametri se koriste za prenos informacija u metodu, pri njenom pozivu
- Lista parametara metode može biti prazna
- Svaki parametar u listi parametara metode može da se sastoji iz modifikatora parametra, tipa i naziva parametra
- Naziv metode i lista parametara **čine potpis metode**, pri čemu je redoslijed parametara u listi bitan

Deklaracija klase

```
public class Kalkulator {
	private int operand1;				// 1
	private int operand2;				// 2

	Kalkulator(int op1, int op2){			// 3
		operand1 = op1;
		operand2 = op2;
	}

	public int zbir(){				// 4
		return operand1 + operand2;
	}

	public int razlika(){				// 5
		return operand1 - operand2;
	}
}
```

Deklaracija konstruktora

```
<modifikator pristupa> <naziv klase> (<lista
  parametara>)
<throws klauzula> // zaglavlje konstruktora
{ // tijelo konstruktora
   <deklaracije lokalnih promjenljivih>
   <deklaracije ugnježdenih lokalnih klasa>
   <naredbe>
}
```

Deklaracija konstruktora

- Bitno je napomenuti da se nazivi klasa i metoda nalaze u različitim prostorima imena. Iz tog razloga ne postoji konflikt naziva između naziva klasa i naziva imena – moguće je deklarisati metodu koja ima naziv identičan nazivu konstruktora (i nazivu klase)

```
public class Konstruktor {
	public Konstruktor(){		// 1
		super();
	}

	public void Konstruktor(){	// 2
		System.out.println("Metoda koja ima naziv
identican nazivu konstruktora");
	}
}
```

- Ovo se smatra lošom programerskom praksom i preporuka je da se ovakvo imenovanje metoda izbjegava

Podrazumijevani konstruktor

- Podrazumijevani konstruktor je konstruktor bez liste parametara
- **ako klasa ne specificira niti jedan konstruktor, kompajler će generisati implicitni podrazumijevani konstruktor**
- jedini zadatak implicitnog podrazumijevanog konstruktora jeste da pozove konstruktor roditeljske klase (pozivom metode super())
- programer može implementirati podrazumijevani konstruktor na proizvoljan način – u ovom slučaju kompajler **ne generiše implicitni podrazumijevani konstruktor**, a implementirani konstruktor se naziva **eksplicitni podrazumijevani konstruktor**

Nizovi

- Niz je struktura podataka koja definiše kolekciju elemenata istog tipa, kojima se pristupa na osnovu pozicije
- Prvi element niza nalazi se na poziciji 0, a posljednji na poziciji n-1, gdje je n broj elemenata niza
- Nizovi u Javi su objekti, pri čemu elementi niza mogu biti primitivnog tipa ili reference
- Svaki ovakav objekat ima final polje length koje specificira veličinu niza, tj. broj elemenata niza
- Nizovi mogu biti jednodimenzionalni ili višedimenzionalni, pri čemu su višedimenzionalni nizovi implementirani kao nizovi nizova
- Reference na nizove se čuvaju na steku, dok se elementi nizova čuvaju na heap-u

Nizovi

- deklaracija

```
<tip elemenata niza>[] <naziv niza>;
<tip elemenata niza> <naziv niza>[];
```

- deklaracijom veličina niza nije specificirana, što znači da promjenljivoj <naziv niza> može biti dodjeljena vrijednost reference niza bilo koje dužine, sve dok su njegovi elementi tipa <tip elemenata niza>

```
int[] niz;                          // 1
Kalkulator nizKalkulatora[];        // 2
```

- postoje dva ravnopravna načina deklaracije niza, u zavisnosti od lokacije operatora [].
- deklaracijom niza se ne kreira niz, već se samo deklariše referenca koja može referencirati objekat niza.

Nizovi

- deklaracija i kreiranje niza

```
<tip elemenata niza>[] <naziv niza> = new <tip elemenata
niza> [<veličina niza>];
<tip elemenata niza> <naziv niza>[] = new <tip elemenata
niza> [<veličina niza>];
```

- deklaracija i kreiranje niza može se izvršiti zajedno.

```
int[] niz = new int[10];
Kalkulator nizKalkulatora[] = new Kalkulator[15];
```

- i u ovom slučaju postoje dva ravnopravna načina istovremene deklaracije i kreiranja niza

Nizovi

- inicijalizacija
- korištenjem petlje

```
for (int i = 0; i < niz.length; i++) {
      niz[i] = i;
}
```

- deklaracija, kreiranje i inicijalizacija niza može se izvršiti zajedno

```
<tip elemenata niza>[] <naziv niza> = { <inicijalizaciona
lista> };
<tip elemenata niza> <naziv niza>[] = { <inicijalizaciona
lista> };
```

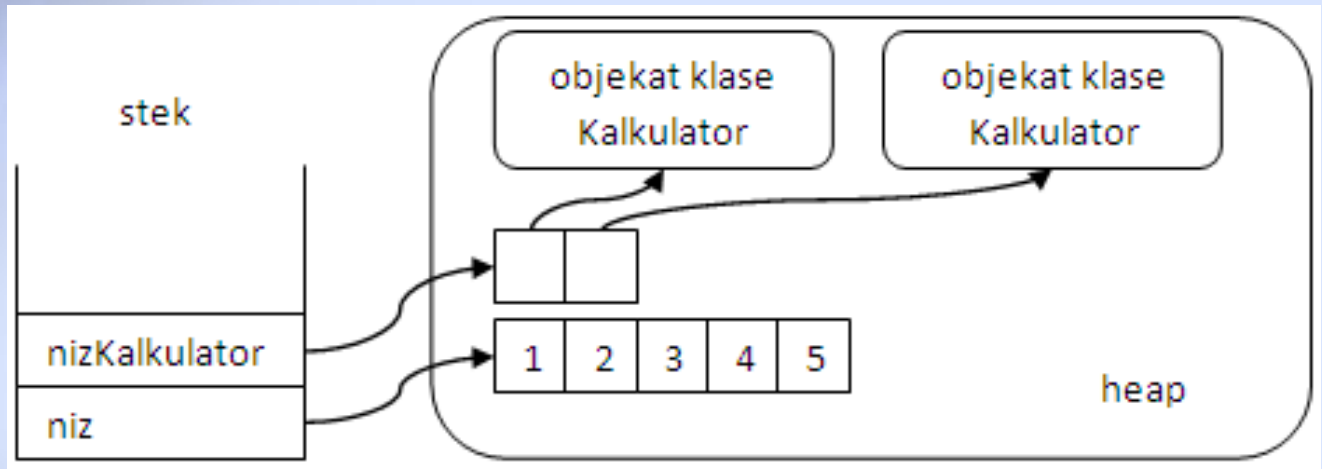
- i u ovom slučaju postoje dva ravnopravna načina istovremene deklaracije, kreiranja i inicijalizacije niza

Nizovi

- primjer deklaracije, kreiranja i inicijalizacije

```
int niz[] = {1, 2, 3, 4, 5};
Kalkulator[] nizKalkulatora = {new Kalkulator(1, 2), new
Kalkulator(0, 0)};
```

- stanje u memoriji



Višedimenzionalni nizovi

- predstavljeni kao nizovi nizova
- deklaracija

```
<tip elemenata niza>[][]...[] <naziv niza>;
<tip elemenata niza> <naziv niza>[][]...[];
```

- višestruke ekvivalentne deklaracije

```
int[][] matrix;
int[] matrix[];
int matrix[][];
```

- deklaracija sa kreiranjem

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

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

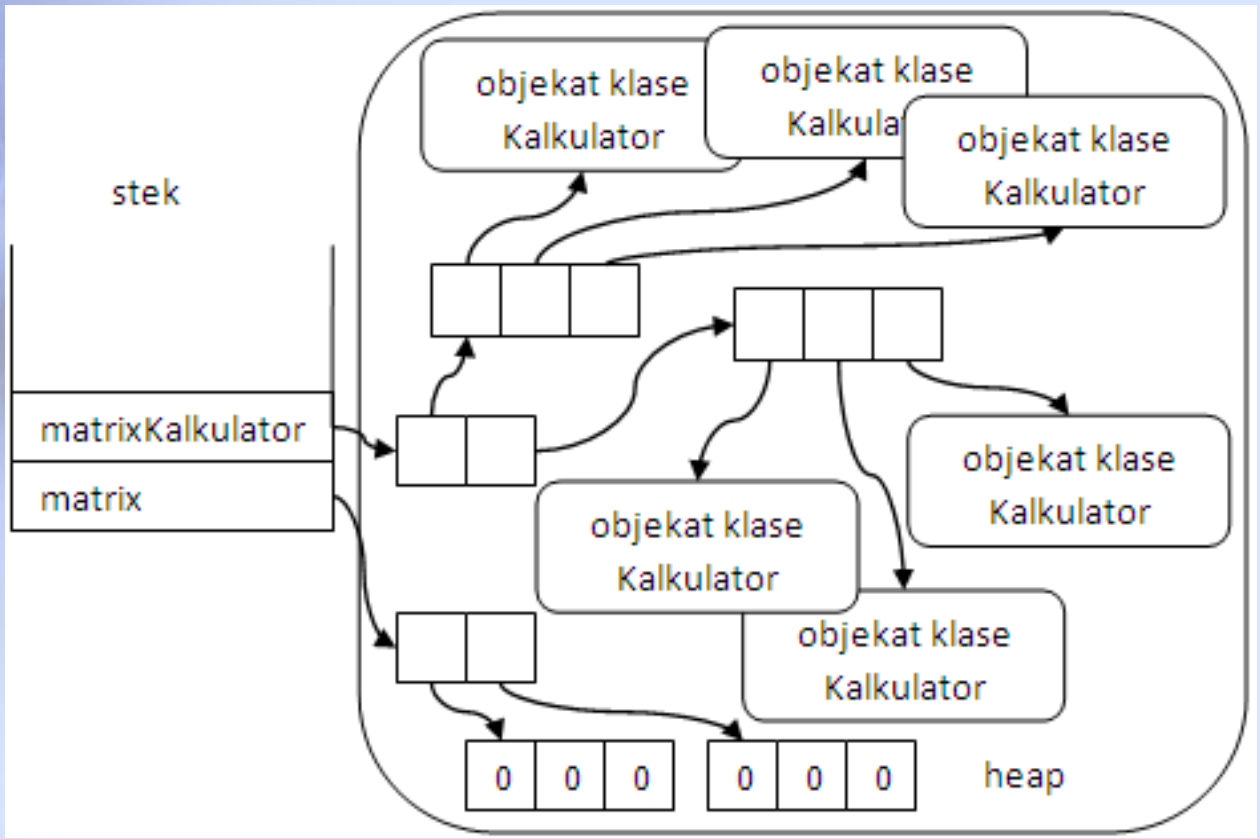
Višedimenzionalni nizovi

- Kod višedimenzionlanih nizova čiji su elementi reference, potrebno je izvršiti još jedan dodatni korak, tj. potrebno je eksplicitno kreirati i same objekte
- primjer

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

Višedimenzionalni nizovi

- stanje u memoriji



Konvencije davanja imena

- nazivi klasa (`MojaKlasa`)
- nazivi metoda (`mojaMetoda`)
- nazivi atributa (`mojAtribut`)
- nazivi paketa (`mojpaket.drugipaket`)
- set/get metode (`setAtribut/getAtribut`)

javadoc

- specijalni komentari u izvornom kodu
- automatsko generisanje programske dokumentacije
- HTML format na izlazu
- Kompletna dokumentacija Jave je generisana javadoc alatom
  - Lokacija: %JAVA_HOME%\doc

javadoc

- komentari počinju sa /**, a završavaju sa */

```
@author
@version
@param
@return
@deprecated
@throws
@see
@exception
@since
@deprecated
@serial
@serialField
@serialData
{@link}
```