

Objektno Orijentisano Programiranje 1

Objektno orijentisano
programiranje

Objektno orijentisano programiranje

- U objektno orijentisanom programiranju, u sklopu opisa problema, potrebno je uočiti entitete (jedinice posmatranja) koji se nalaze u svetu (domen) u kojem se nalazi i problem koji se rešava.
- Potrebno je uočiti koji entiteti se nalaze u svetu, opisati ih i navesti operacije nad njima, a kojima se problem rešava.

Kako grupisati podatke o nekom entitetu?

- Jednu osobu:

```
String ime, prezime;
```

- Više osoba - nizovima?

```
String[][] osoba = new String[2][2]();
```

```
osoba[0][0] = "Pera"; osoba[0][1]="Perić";
```

```
osoba[1][0] = "Đura"; osoba[1][1]="Đurić";
```

- Ili duž?

```
double[] rotiraj(double x, double y, double ugaoRotacije)
```

- Klasama?

Objektno orijentisano programiranje

- Objektno orijentisano programiranje se svodi na identifikaciju entiteta u nekom domenu, navođenje njihovih osobina i pisanje operacija nad tim osobinama.
- U objektno orijentisanoj terminologiji, entiteti su opisani klasama, osobine su atributi, a operacije su metode.

Primer

- Ako pišemo program koji računa površinu kruga, svet u kojem se nalazi krug je Dekartov pravougaoni koordinatni sistem, a opis kruga se sastoji iz navođenja njegovog poluprečnika.
- Postupak rešavanja površine kruga se tada svodi na računanje prema odgovarajućoj formuli: $P = r^2 \times \pi$

Primer

```
class Krug {  
    /** Poluprečnik kruga */  
    double r;  
    /** Racuna površinu kruga */  
    double površina() {  
        return r * r * Math.PI;  
    }  
}
```

Primer

- Sa druge strane, ako je potrebno napisati program koji proverava da li je neka zadata tačka unutar, na ili izvan kruga, tada imamo dva entiteta u domenu:
 - krug i
 - zadatu tačku.
- Krug je tada potrebno opisati njegovim koordinatama i poluprečnikom, a tačku njenim koordinatama.

Primer

```
class Tacka {  
    /** x koordinata */  
    double x;  
    /** y koordinata */  
    double y;  
}  
class Krug {  
    /** Centar kruga */  
    Tacka centar;  
    /** Poluprečnik kruga */  
    double r;  
}
```

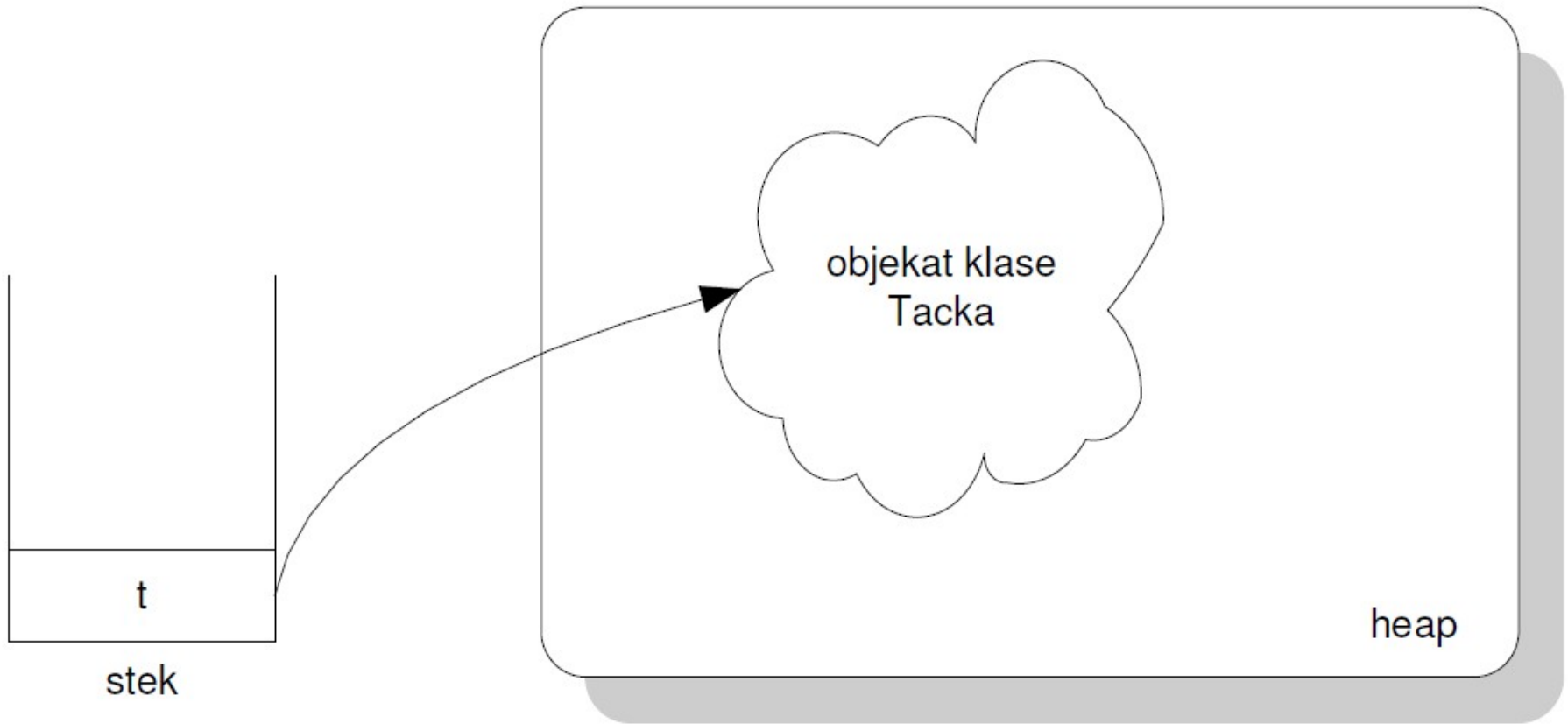

Primer

```
class Dekart {  
    /** Racuna udaljenost dve zadate tacke */  
    double udaljenost(Tacka a, Tacka b) {  
        return Math.sqrt((a.x - b.x)*(a.x - b.x) +  
                           (a.y - b.y)*(a.y - b.y));  
    }  
    /** Ispisuje da li je neka tacka unutar, izvan ili 8  
     * na krugu, na osnovu udaljenosti od centra kruga.  
     */  
    void proverj(Krug k, Tacka t) {  
        double d = udaljenost(k.centar, t);  
        if (d < k.r)  
            System.out.println("unutar kruga.");  
        else if (d == k.r)  
            System.out.println("na krugu");  
        else  
            System.out.println("Izvan kruga");  
    }  
}
```

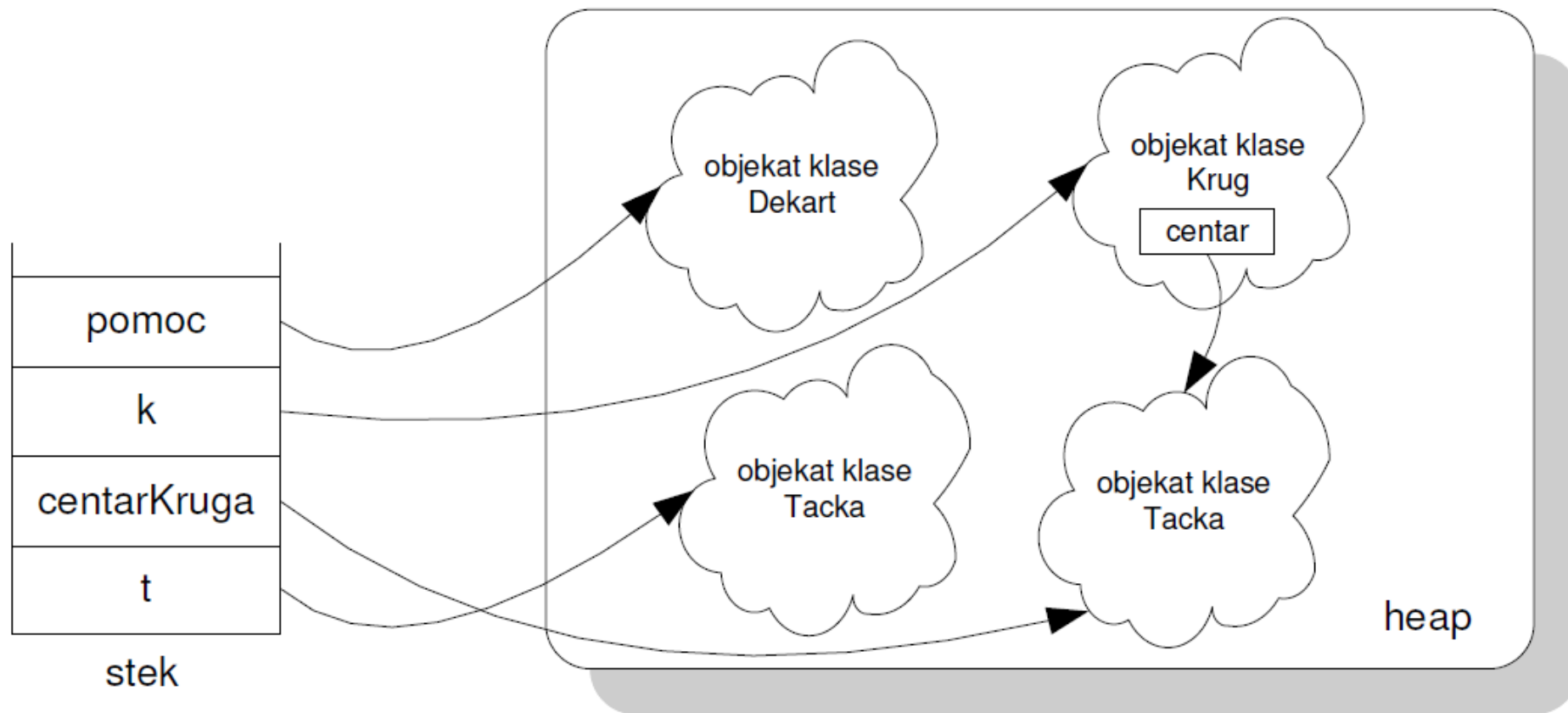
Primer

```
class Test {  
    public static void main(String[] args) {  
        Tacka t = new Tacka();  
        t.x = 5;  
        t.y = 5;  
  
        Tacka centarKruga = new Tacka();  
        centarKruga.x = 0;  
        centarKruga.y = 0;  
  
        Krug k = new Krug();  
        k.centar = centarKruga;  
        k.r = 5;  
  
        Dekart pomoc = new Dekart();  
        pomoc.proveri(k, t);  
    }  
}
```

Tacka t = new Tacka();



Memorija



Ponavljjanje

- Klasa: model objekta
 - uključuje:
 - attribute
 - metode
- Objekat: instanca klase

Sve je objekat

- Nije moguće definisati funkcije i promenljive izvan neke klase
 - zato listing počinje ključnom reči ***class***
- Instance neke klase se zovu objekti
- Objekti se kreiraju upotrebom ključne reči ***new***

Primer

```
class Osoba {  
    String JMBG;  
    String ime;  
    String prezime;  
    Datum datRodj;  
}
```

Primer klase

Automobil
+ radi : boolean
+ upali () : void
+ ugasi () : void

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


Kako se instancira i koristi?

Automobil
- radi : boolean
+ upali () : void
+ ugasi () : void

```
...  
Automobil a = new Automobil();  
Automobil b = new Automobil();  
  
...  
a.upali();  
b.ugasi();
```

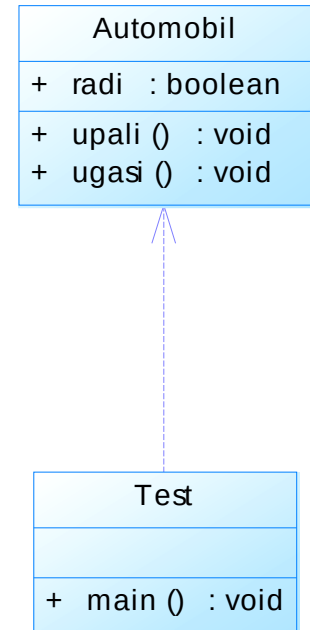
Program sa dve klase

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();  
    }  
}
```



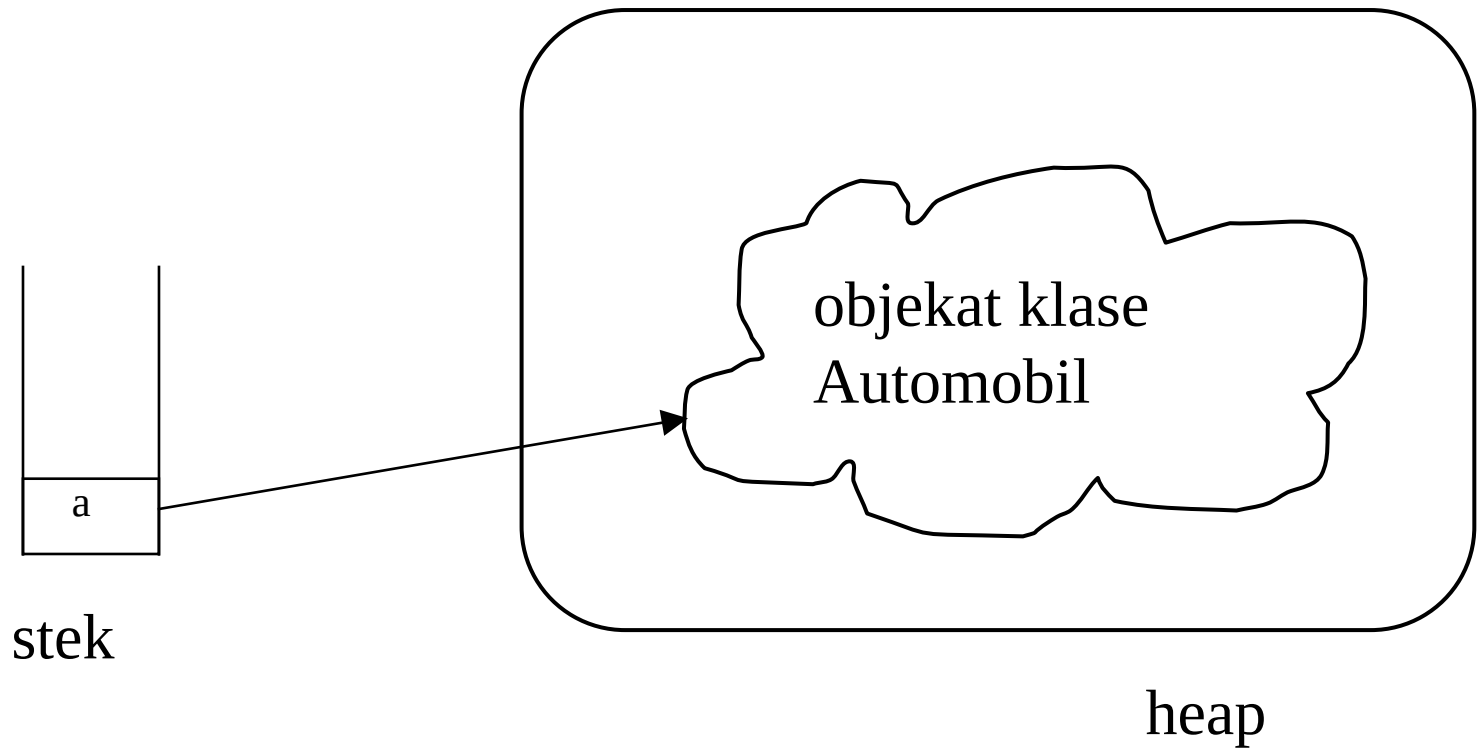
Reference na objekte

```
Automobil a;
```

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

lokalna promenljiva **a** nije objekat, već
referenca na objekat

Reference na objekte



Ključna reč **this**

- Ako je klasa Tacka zadužena za rad sa svim što ima veze sa tačkama, da li možemo da joj prepustimo i posao računanja udaljenosti neke druge tačke od nje same?

Ključna reč **this**

```
Tacka a = new Tacka();  
a.x = 0;  
a.y = 0;  
Tacka b = new Tacka();  
b.x = 5;  
b.y = 5;  
double d;  
d = a.udaljenost(b);  
// isti rezultat dobijamo i ovako  
d = b.udaljenost(a);
```

Ključna reč **this**

```
class Tacka {  
    /** x koordinata */  
    double x;  
  
    /** y koordinata */  
    double y;  
  
    /** Računa udaljenost od ove tačke do  
        * prosledjene tačke *  
    double udaljenost(Tacka b) {  
        return Math.sqrt((this.x - b.x)*(this.x - b.x) +  
                           (this.y - b.y)*(this.y - b.y));  
    }  
}
```

Ključna reč **this**

```
/** Atribut koji opisuje visinu */  
private double visina;  
/** Postavlja vrednost atributa */  
void setVisina(double visina) {  
    this.visina = visina;  
}  
double getVisina() {  
    return this.visina;  
}
```


Default vrednosti atributa

- Primitivni tipovi kao atributi klase

<u>Primitivni tip</u>	<u>Default</u>
boolean	false
char	'\u0000'
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

- Reference kao atributi klase
 - null
- Lokalne promenljive
 - nemaju default vrednost – upotreba pre inicijalizacije izaziva grešku kod kompajliranja!

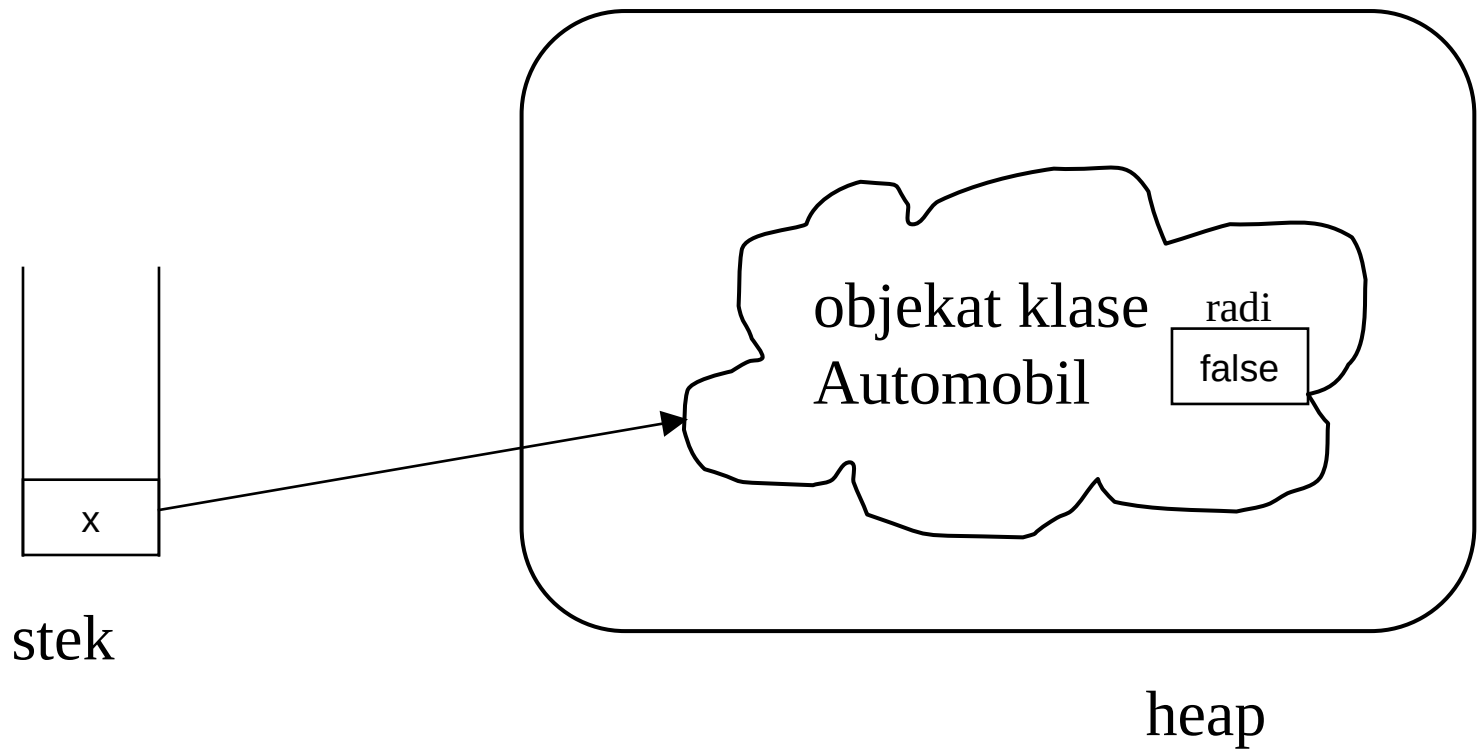
Referenca na objekat kao parametar metode

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

...

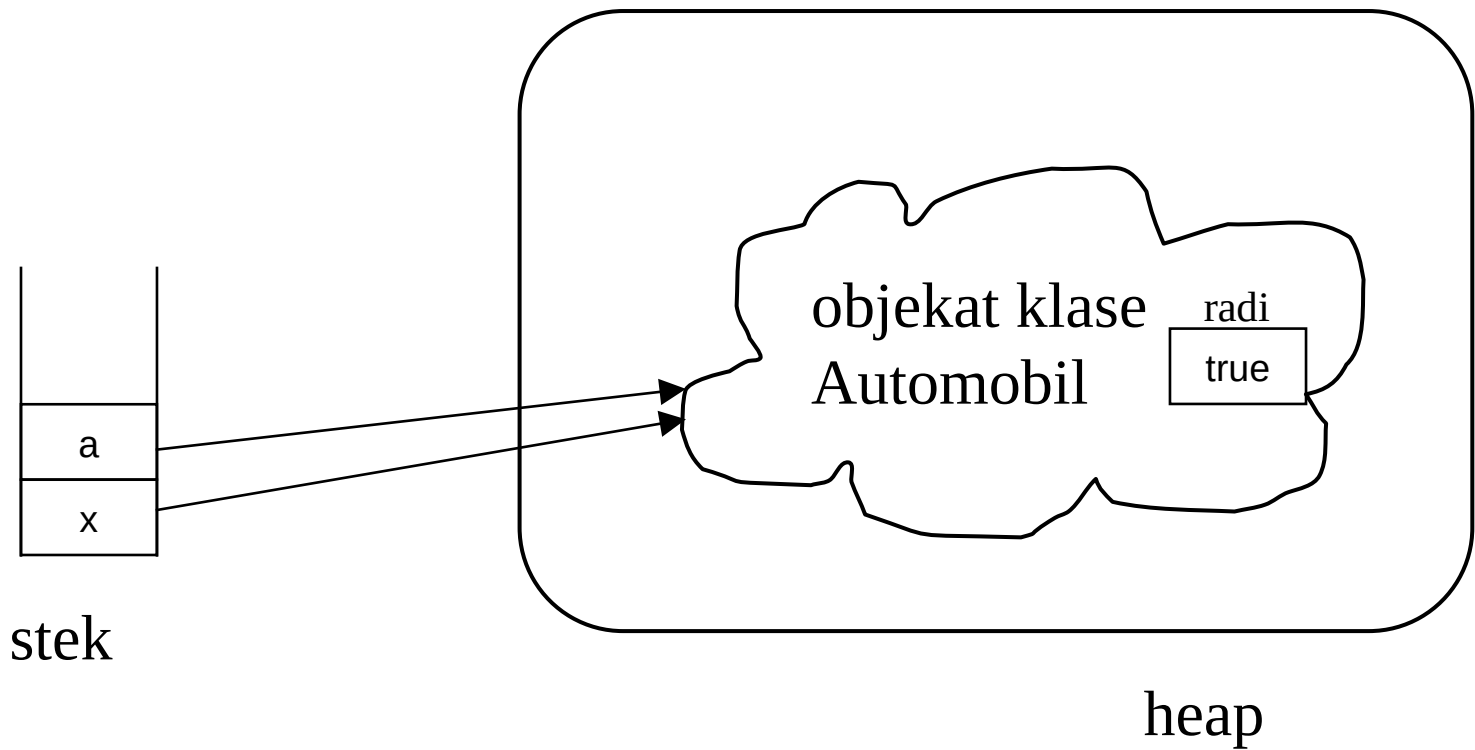
```
Automobil x = new Automobil();  
x.radi = false;  
test(x);
```

Referenca na objekat kao parametar metode



```
Automobil x = new Automobil();  
x.radi = false;
```

Referenca na objekat kao parametar metode



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

Method overloading

- U klasi može da postoji više metoda sa istim imenom
- Razlikuju se po parametrima
- Metode se nikada ne razlikuju po povratnoj vrednosti

Method overloading

```
class A {  
    int metoda() { ... }  
    int metoda(int i) { ... }  
    int metoda(String s) { ... }  
}
```

- Metode se nikada ne razlikuju po povratnoj vrednosti!

null literal

- Ako želimo da inicijalizujemo referencu tako da ona ne ukazuje ni na jedan objekat, onda takvoj promenljivoj dodeljujemo **null** vrednost, odn. **null** literal:

```
Automobil a = null;
```

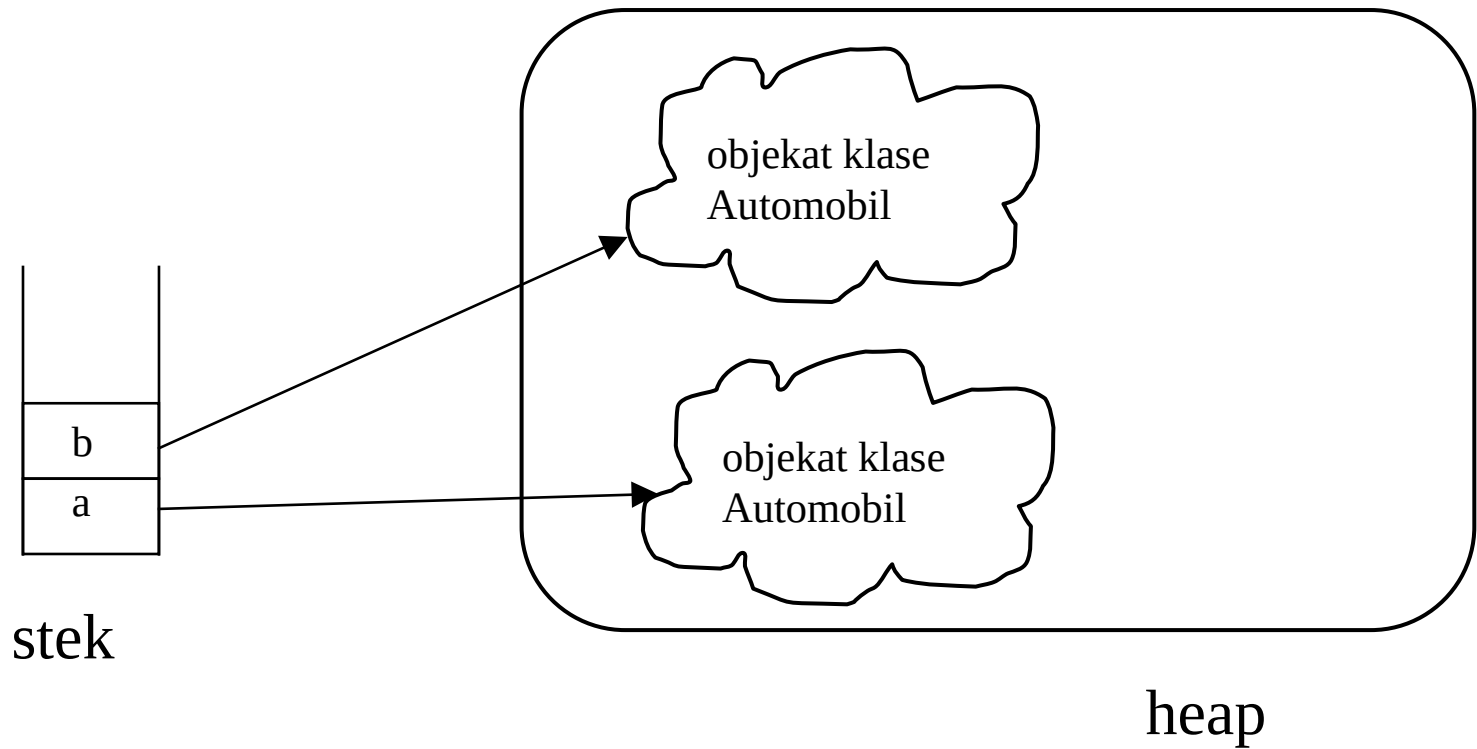
Operator dodele vrednosti

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



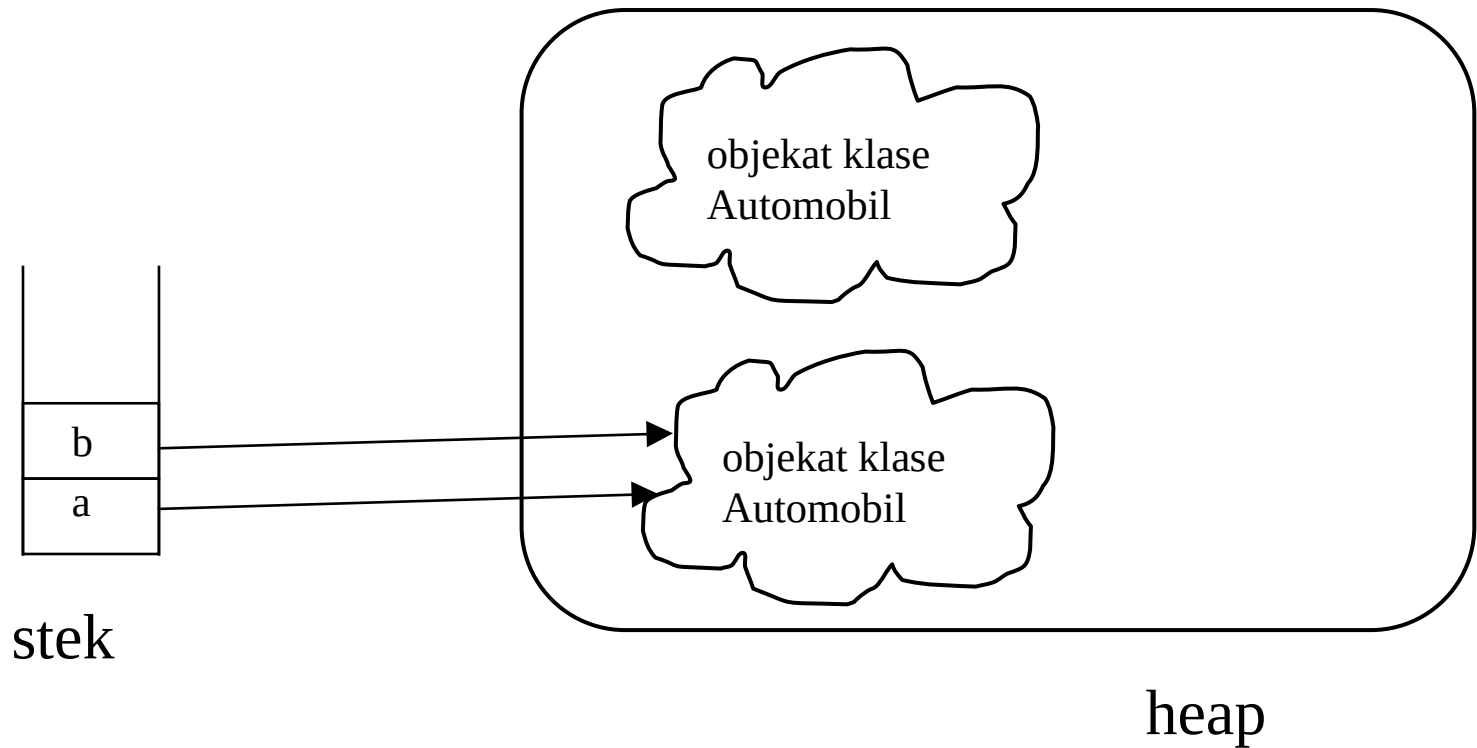
Vrši se kopiranje
vrednosti reference!

Reference na objekte



Reference na objekte

$b = a$



Nizovi i objekti

```
int a[]; // još uvek nije napravljen niz!
```

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

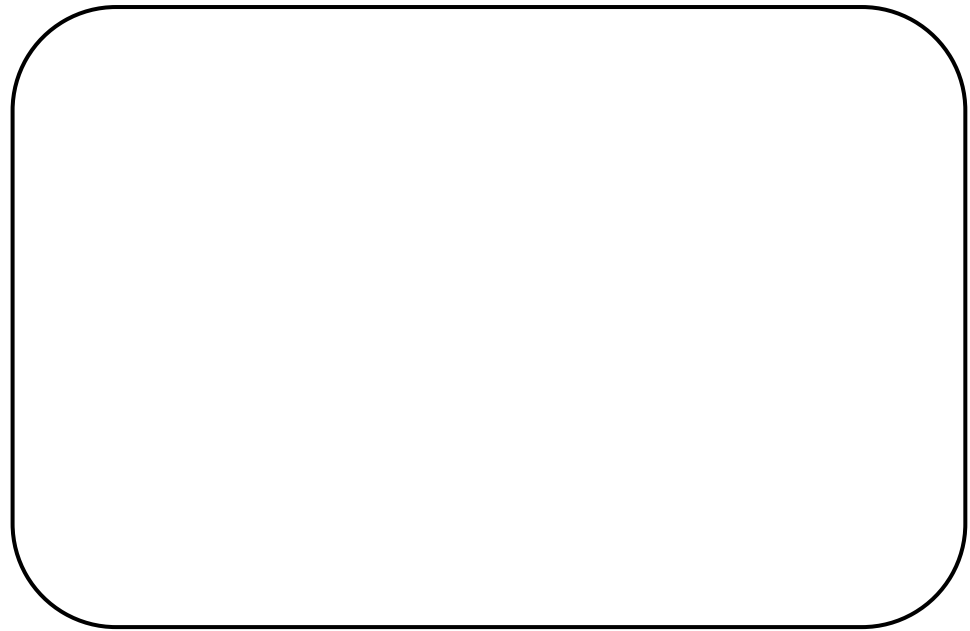
```
int a[] = { 1, 2, 3, 4, 5 };
```

```
Automobil[] parking = new Automobil[20];  
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

Nizovi primitivnih tipova ^{1/3}



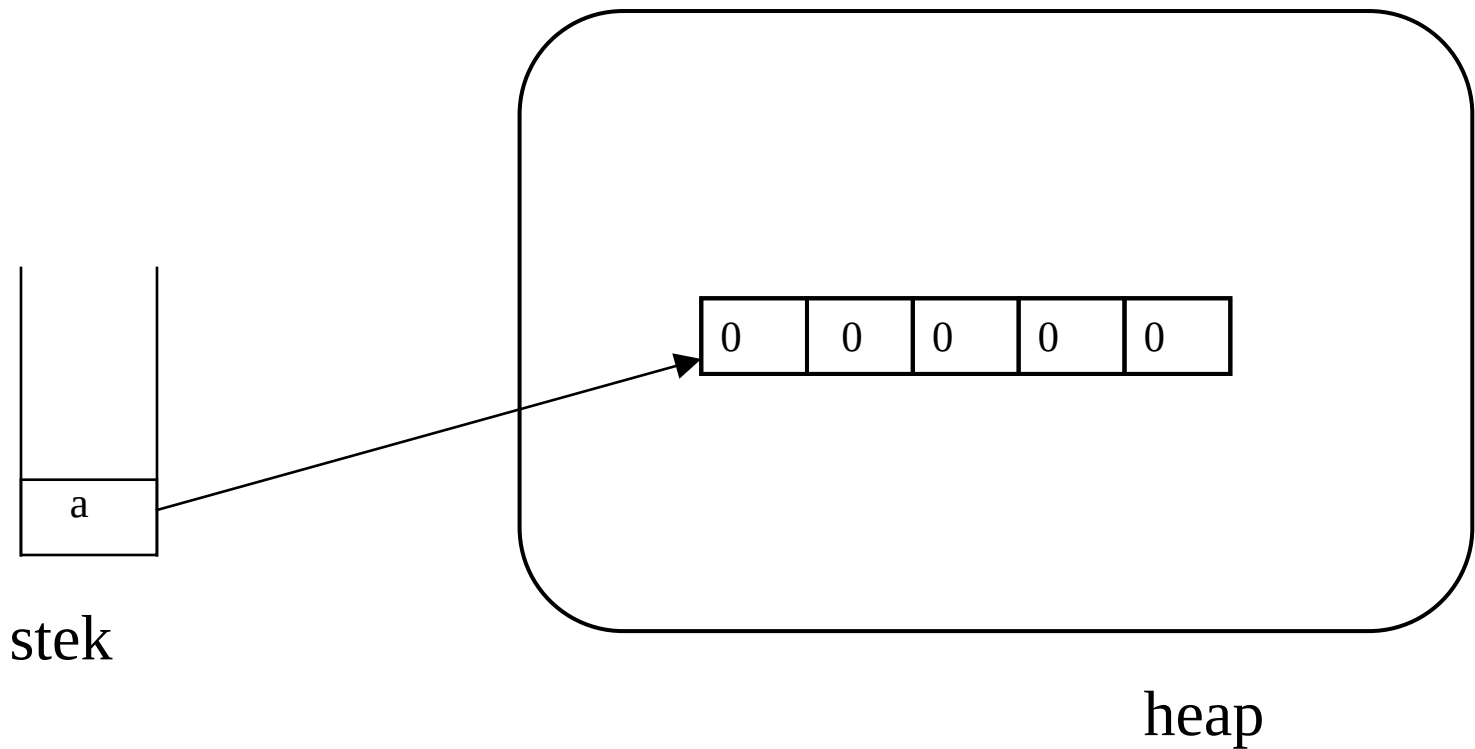
stek



heap

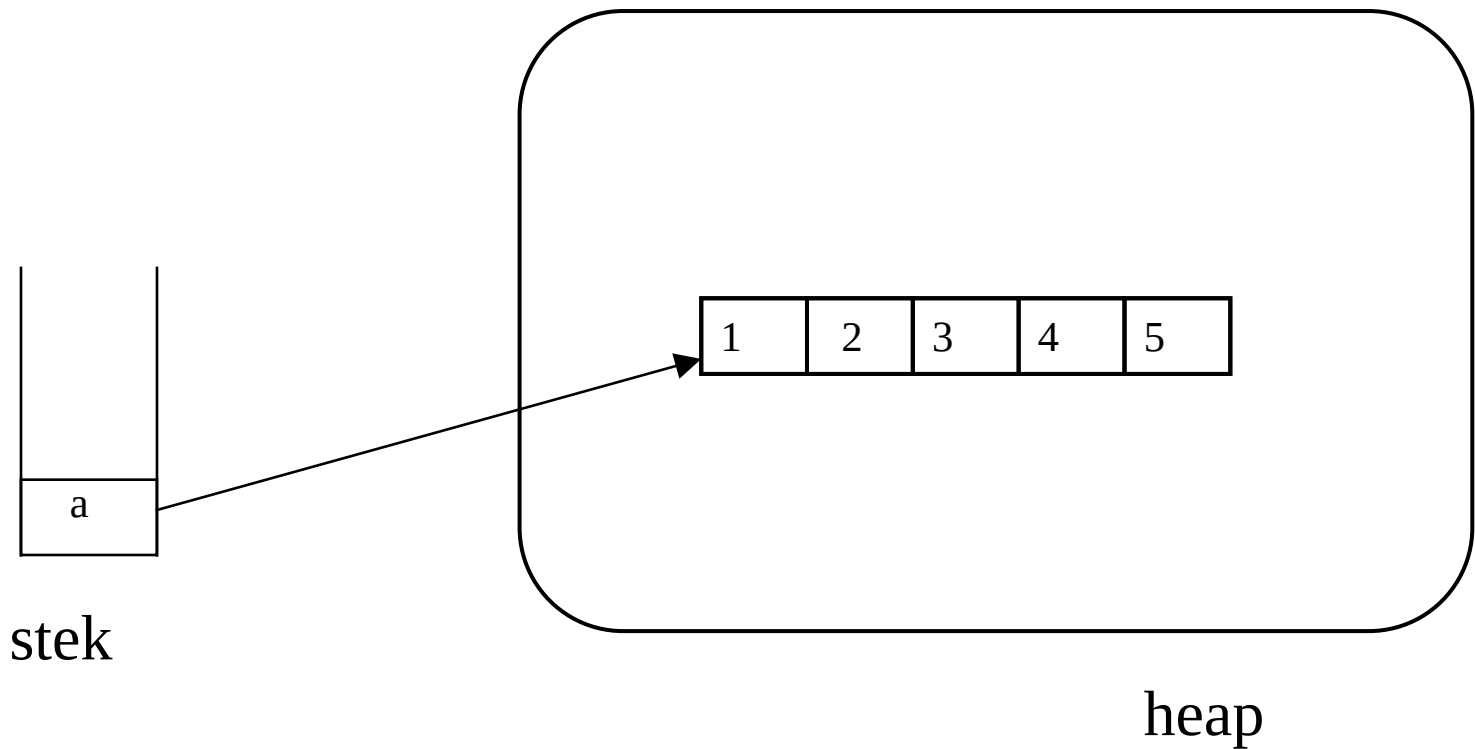
```
int a[];
```

Nizovi primitivnih tipova ^{2/3}



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

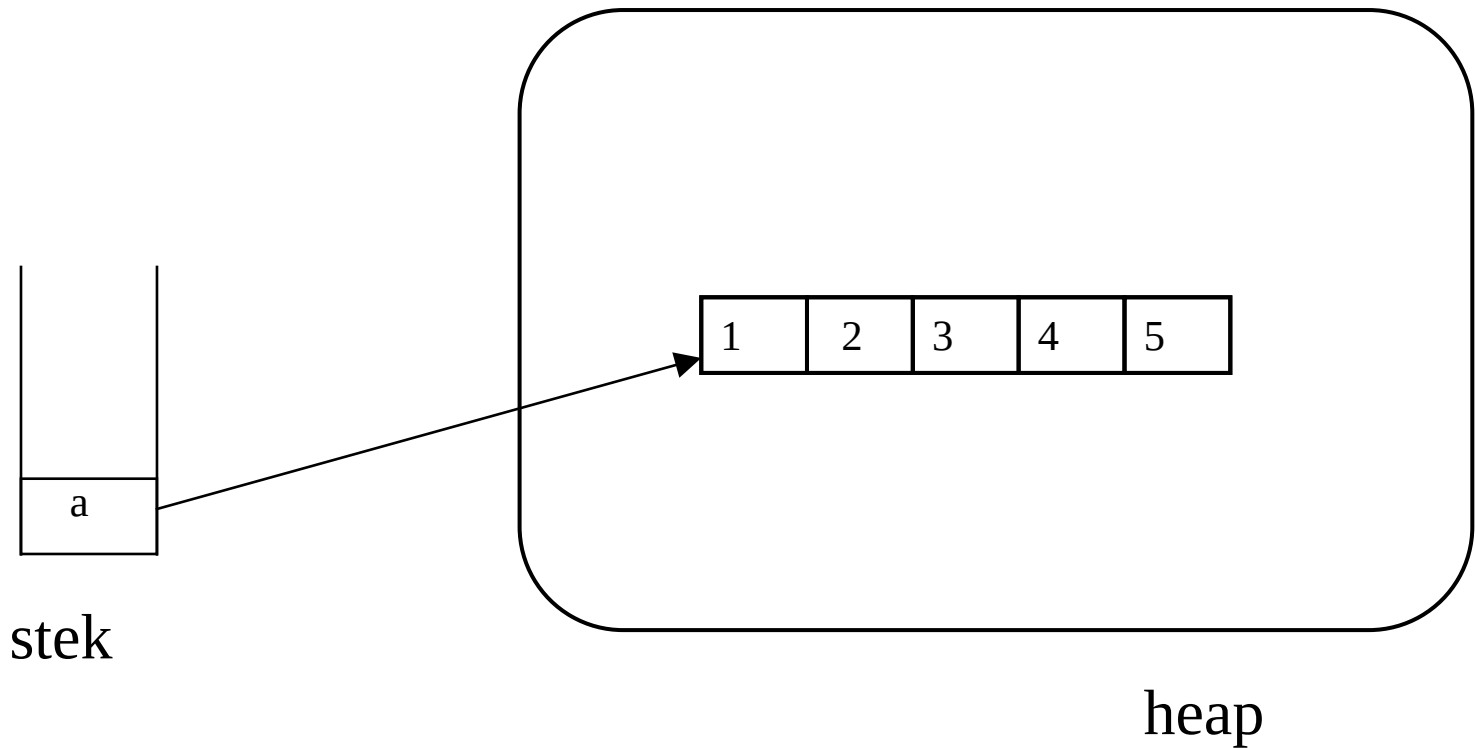
Nizovi primitivnih tipova ^{3/3}



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

Nizovi primitivnih tipova

jednim potezom

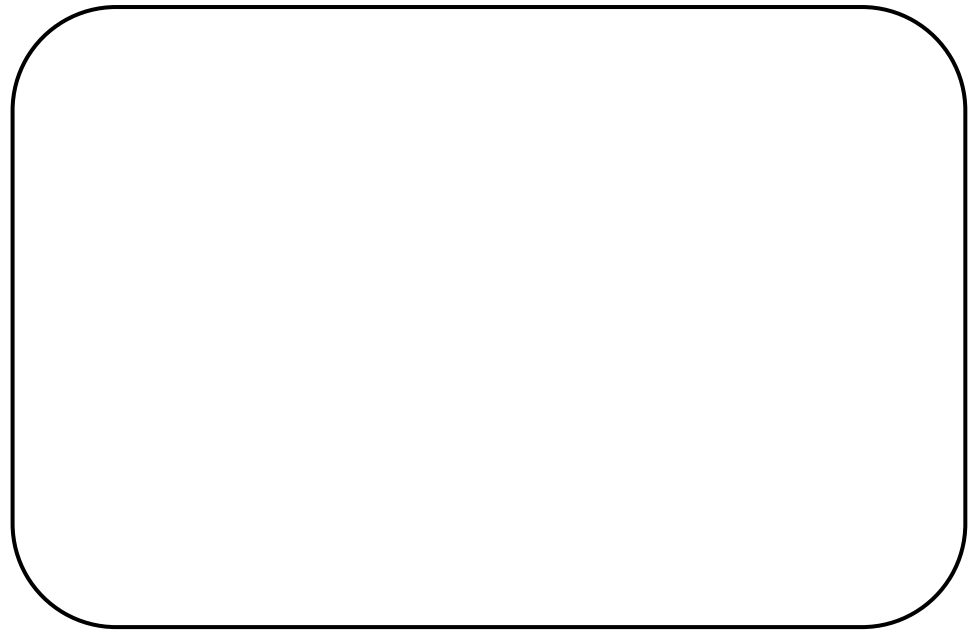


```
int a[] = { 1, 2, 3, 4, 5 };
```

Nizovi referenci na objekte ^{1/3}



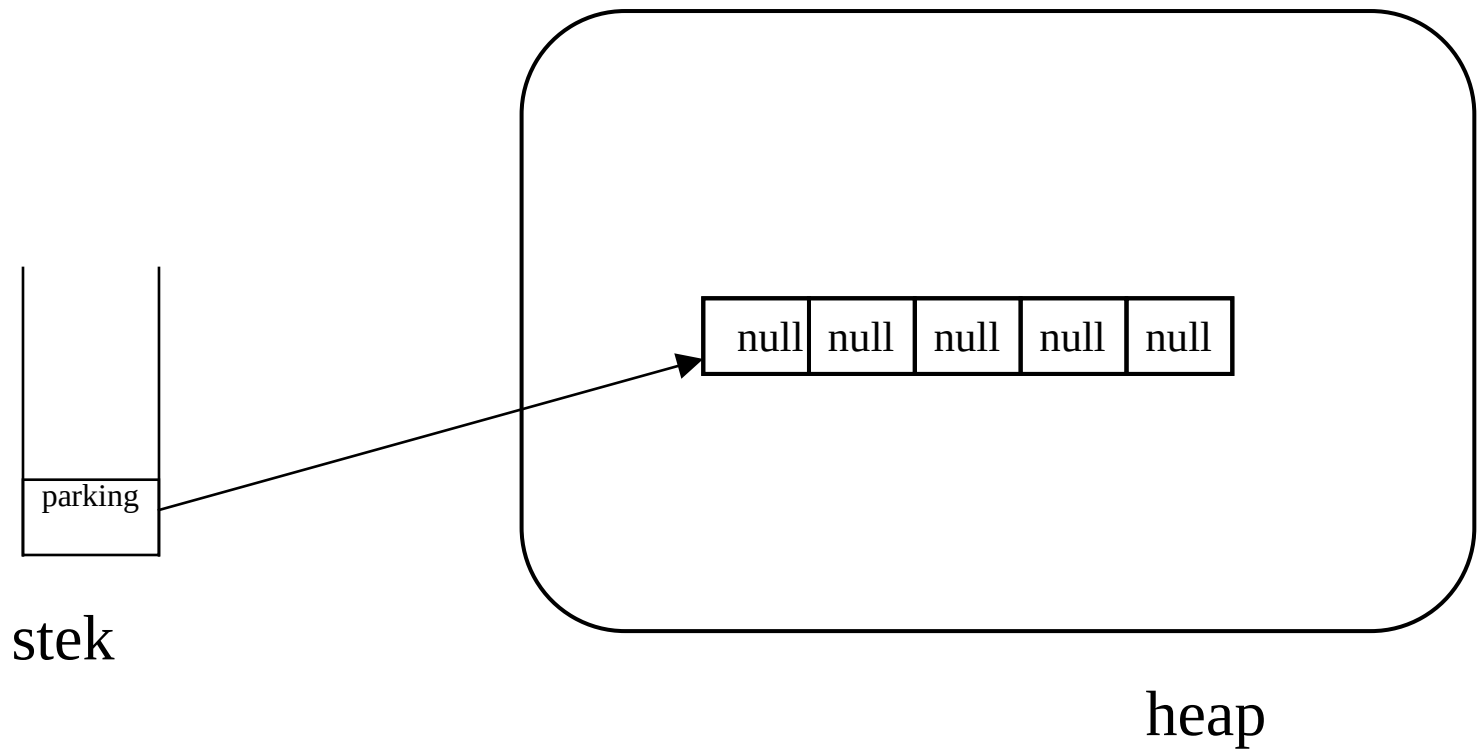
stek



heap

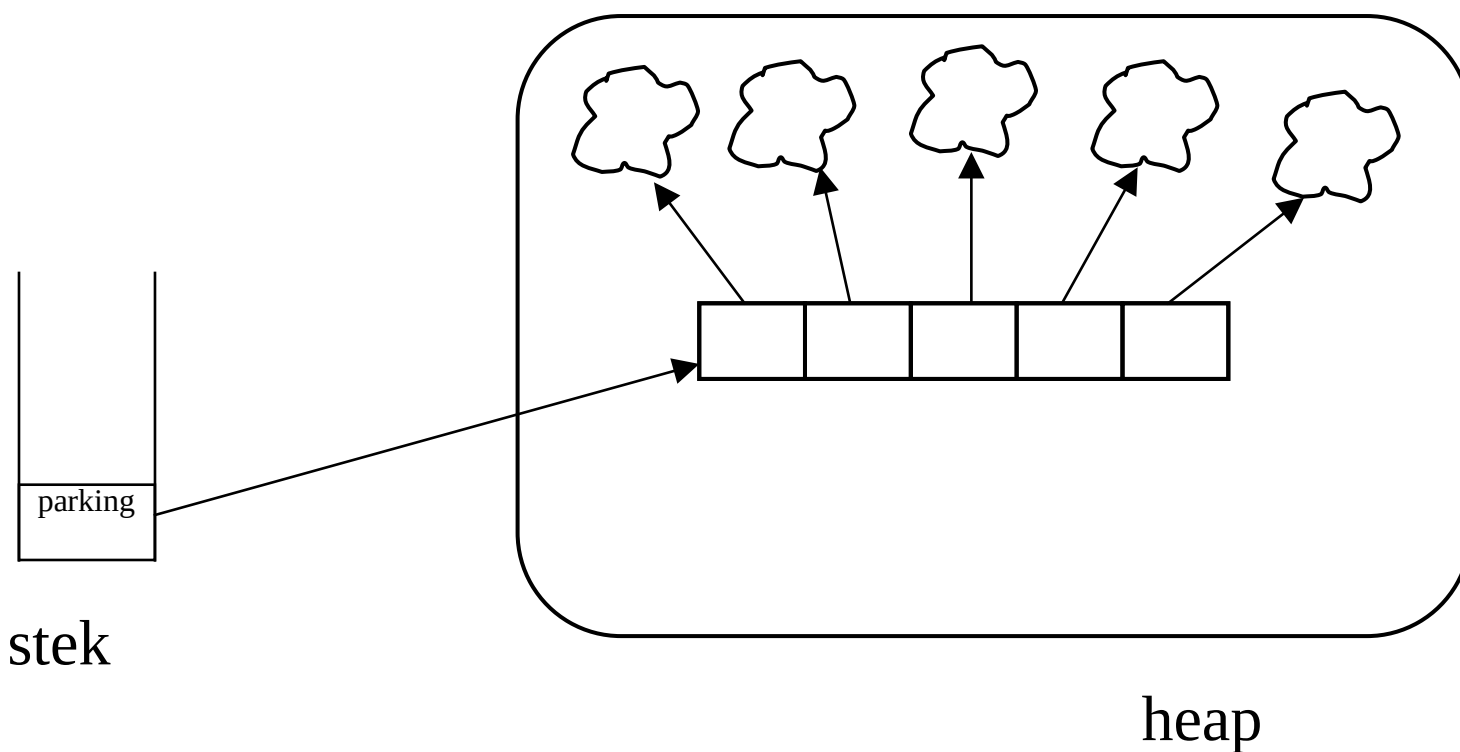
```
Automobil[] parking;
```


Nizovi referenci na objekte ^{2/3}



```
parking = new Automobil[5];
```

Nizovi referenci na objekte 3/3



```
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

Kreiranje i popunjavanje

- Samo prilikom deklaracije promenljive (inicijalizacija prilikom deklaracije)

```
Automobil[] parking = {  
    new Automobil(),  
    new Automobil(),  
    new Automobil()};
```

- Bilo gde u programu:

```
Automobil[] parking;  
...  
parking = new Automobil[] {  
    new Automobil(),  
    new Automobil(),  
    new Automobil()  
};
```

Višedimenzijski nizovi

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

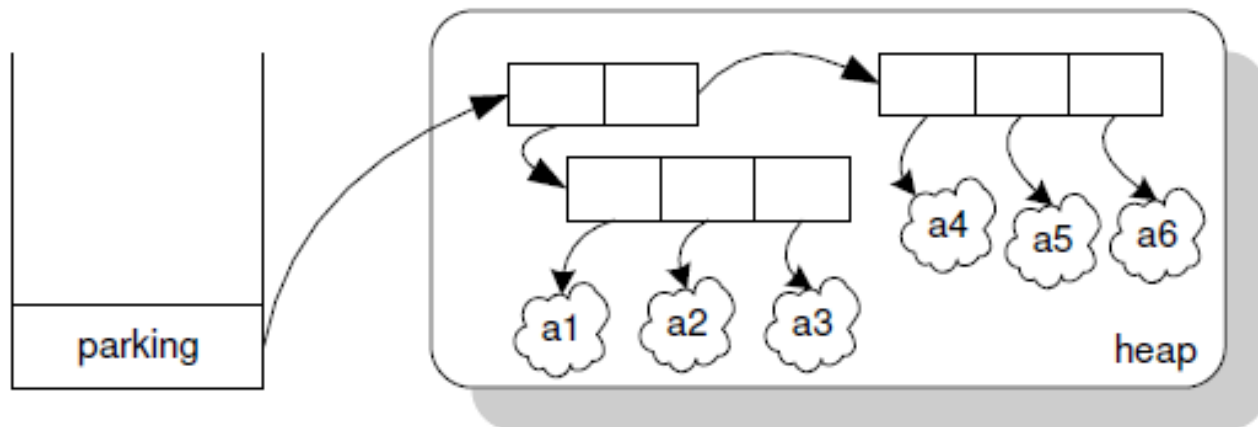
```
int a[][] = new int[2][3];
```

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

```
Automobil[][] a = {  
    { new Automobil(), new Automobil() },  
    { new Automobil(), new Automobil() }  
};
```

Višedimenzijski nizovi

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



Inicijalizacija objekata

- Ako želimo posebnu akciju prilikom kreiranja objekta neke klase, napravićemo konstruktor
- Konstruktor se automatski poziva prilikom kreiranja objekta

Automobil a = new Automobil();



- Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi

Inicijalizacija objekata

```
Automobil a = new Automobil();  
a.prednjiLevi = new Tocak();  
a.prednjiDesni = new Tocak();  
a.zadnjiLevi = new Tocak();  
a.zadnjiDesni = new Tocak();
```

- Umesto da se svaki put ovo piše u programu, napravi se konstruktor u klasi Automobil u kojem će pisati:

```
Automobil() {  
    prednjiLevi = new Tocak();  
    prednjiDesni = new Tocak();  
    zadnjiLevi = new Tocak();  
    zadnjiDesni = new Tocak();  
}
```

Inicijalizacija objekata

- konstruktor

```
class A {  
    A() {  
        System.out.println("konstruktor");  
    }  
}  
...  
A promenljiva = new A();
```

na konzoli će pisati:
konstruktor

Konstruktor sa parametrima

- Može da se napravi i konstruktor sa parametrima

```
class A {  
    A(String s) {  
        System.out.println(s);  
    }  
}  
...  
A promenljiva = new A("blabla");
```

na konzoli će pisati:

blabla

Podrazumevani konstruktor

- Ukoliko se unutar definicije klase ne navede nijedan konstruktor, kompajler će sam generisati tzv. podrazumevani konstruktor koji nema parametre, a telo mu je prazno.
- Taj konstruktor se neće kreirati na nivou izvornog koda, već na nivou bajt-koda (prevedenog koda).

Automobil a = new Automobil();



Constructor overloading

- Možemo imati više konstruktora
- Ako ne napravimo default konstruktor, a napravimo barem još jedan konstruktor, default konstruktor se više **ne pravi** u bajtkodu!
 - preporuka: napraviti default konstruktor uvek

Constructor overloading

```
class Tacka {  
    /** x koordinata */  
    double x;  
    /** y koordinata */  
    double y;  
    /** Podrazumevani konstruktor  
    * x i y će postaviti na nulu  
    */  
    Tacka() {  
        x = 0;  
        y = 0;  
    }  
}
```


Constructor overloading

```
/** Konstruktor koji prima koordinate */
Tacka(double x, double y) {
    this.x = x;
    this.y = y;
}
/** Konstruktor koji prima tačku */
Tacka(Tacka t) {
    this.x = t.x;
    this.y = t.y;
}
}
```

Constructor overloading

```
class TestTacka {  
    public static void main(String[] args) {  
        Tacka t1 = new Tacka();  
        Tacka t2 = new Tacka(2, 2);  
        Tacka t3 = new Tacka(t2);  
    }  
}
```

Ključna reč **this** i konstruktori



```
class Krug {  
    Tacka centar;  
    double r;  
    Krug() {  
        centar = new Tacka();  
    }  
    Krug(double x, double y, double r) {  
        this();  
        centar.x = x;  
        centar.y = y;  
        this.r = r;  
    }  
}
```

Uništavanje objekata – Garbage collector

- Radi kao poseban proces u pozadini
- Automatska dealokacija memorije
- Automatska defragmentacija memorije
- Poziva se po potrebi
 - možemo ga eksplicitno pozvati sledećim kodom:
`System.gc();`
ali Garbage Collector će sam "odlučiti" da li će dealocirati memoriju
 - poziv ove metode je samo sugestija GC-u da bi mogao da otpočne čišćenje

Garbage collector

```
class GCTest {  
    GCTest() {  
        System.out.println("Konstruktor");  
    }  
    protected void finalize() throws Throwable {  
        System.out.println("finalized");  
    }  
    public static void main(String[] args) {  
        GCTest test = new GCTest();  
        System.out.println("main running...");  
        System.gc();  
    }  
}
```

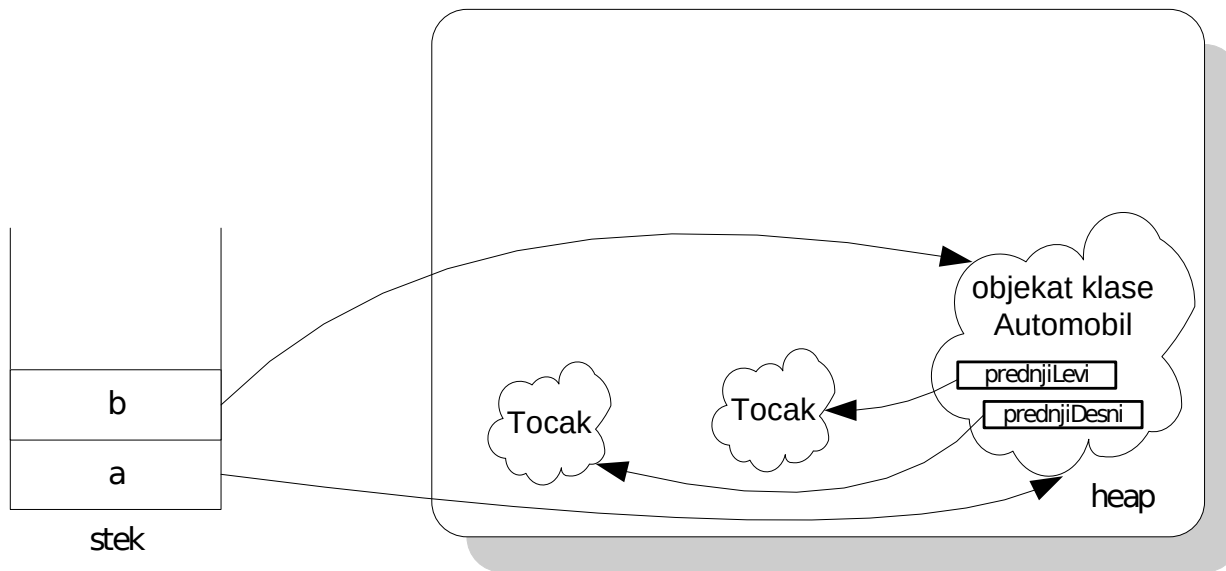
Garbage collector

- Ne postoji destruktor
- Možemo napisati posebnu metodu **finalize()**, koja se poziva neposredno pre oslobađanja memorije koju je objekat zauzimao
 - nemamo garanciju da će biti pozvana
- **Da li je moguć memory leak?**

Deep vs. shallow copy

- Šta se događa kada primenimo operator '=' na reference?
 - tada i originalna promenljiva i nova ukazuju na isti objekat.

Deep vs. shallow copy



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

Deep vs. shallow copy

- Šta ako napravimo metodu `copy()` koja vraća kopiju originalnog objekta?
 - sve zavisi od atributa:
 - ako su atributi primitivni, sve je OK
 - ako atributi nisu primitivni, tada dobijamo *shallow copy*

Deep vs. shallow copy

- Ako operatorom '=' zapravo pravimo kopiju vrednosti promenljive, gde je problem?

– ako napravimo metodu:

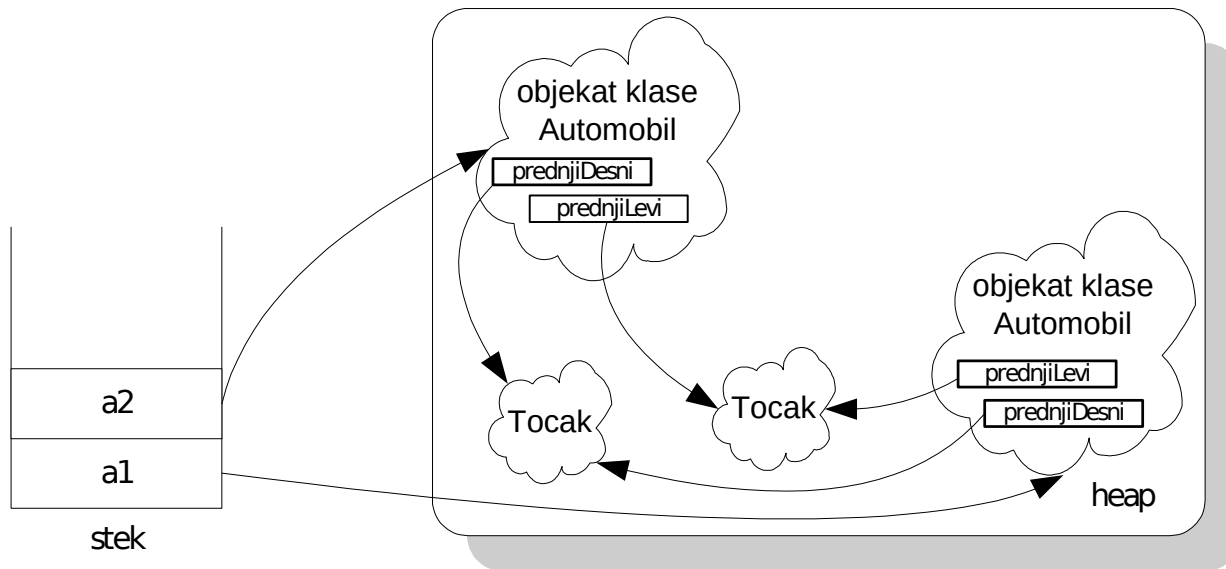
Automobil copy() {...}

koja će praviti kopiju objekta, onda moramo da napravimo kopiju svih atributa iz originala

- operator '=' kod primitivnih tipova radi deep copy
- kod referenci ne radi deep copy objekta, već kopira reference

Plitka kopija

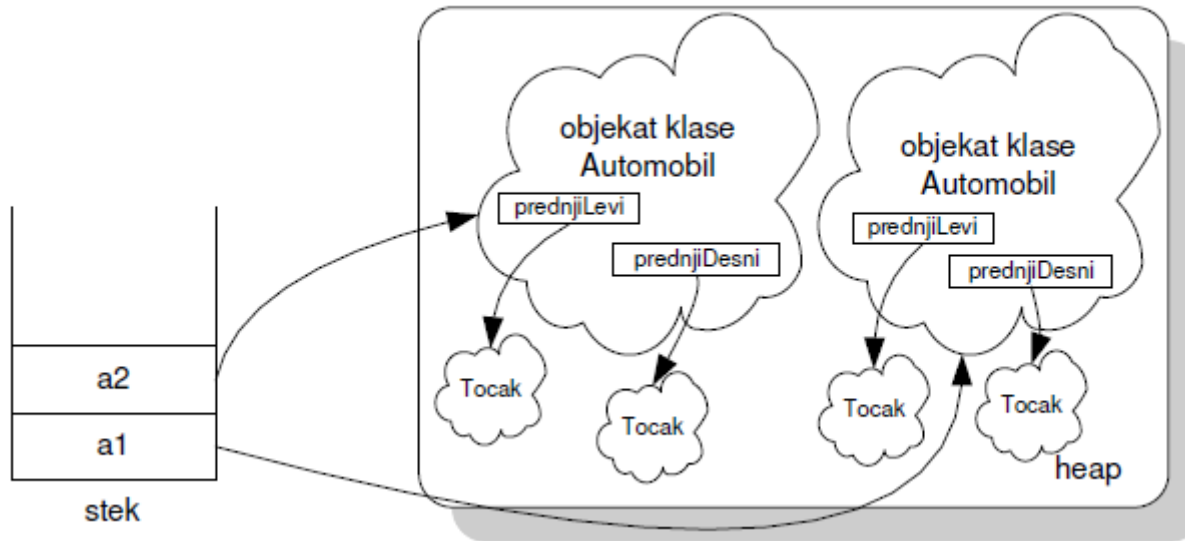
```
Automobil copy () {  
    Automobil ret = new Automobil();  
    ret.prednjiLevi = this.prednjiLevi;  
    ret.prednjiDesni = this.prednjiDesni;  
    ret.zadnjiLevi = this.zadnjiLevi;  
    ret.zadnjiDesni = this.zadnjiDesni;  
    return ret;  
}
```



Duboka kopija

```
Tocak copy() {  
    Tocak ret = new Tocak();  
    ret.pritisak = this.pritisak;  
    return ret;  
}
```

```
Automobil copy () {  
    Automobil ret = new Automobil();  
    ret.prednjiLevi = this.prednjiLevi.copy();  
    ret.prednjiDesni = this.prednjiDesni.copy();  
    ret.zadnjiLevi = this.zadnjiLevi.copy();  
    ret.zadnjiDesni = this.zadnjiDesni.copy();  
    return ret;  
}
```



Parametri i rezultat metoda

- Parametri mogu biti:
 - primitivni tipovi
 - reference na objekte
- Rezultat može biti:
 - primitivni tip
 - referenca na objekat
- Metoda vraća vrednost naredbom:
return vrednost
ili
return (vrednost)

Ključna reč **static**

- Definiše statičke attribute i metode
- Statički atributi i metode postoje i bez kreiranja objekta
 - zato im se može pristupiti preko imena klase
 - `StaticTest.i++`;
- Statički atributi imaju istu vrednost u svim objektima
 - ako promenim statički atribut u jednom objektu, on će se promeniti i kod svih ostalih objekata
- Namena statičkih metoda:
 - pristup i rad sa statičkim atributima
 - opšte metode za koje nije potrebno da se kreira objekat
 - `Math.sin(x)`

Ključna reč **static**

```
class StaticTest {  
    static int i = 47;  
    static void metoda() { i++; }  
}
```

```
...  
StaticTest st1 = new StaticTest();  
StaticTest st2 = new StaticTest();
```

```
...  
st1.i++;           // isto što i st2.i++;  
StaticTest.i++;  
StaticTest.metoda();
```

Ključna reč **static**

- `System.out.println();`
- `Math.random();`
- `Math.sin();`
- `Math.PI;`
- `public static void main(String[] args) {...}`
- `java Hello` → `Hello.main(args)`

Statički blok

- Statički blok se izvršava samo jednom, prilikom prvog korišćenja klase
- Unutar statičkog bloka može se pristupati samo statičkim atributima i mogu se pozivati samo statičke metode

Statički blok

```
class Test {  
    static int a;  
    static int b;  
    static void f() {  
        b = 6;  
    }  
    static {  
        a = 5;  
        f();  
    }  
}
```