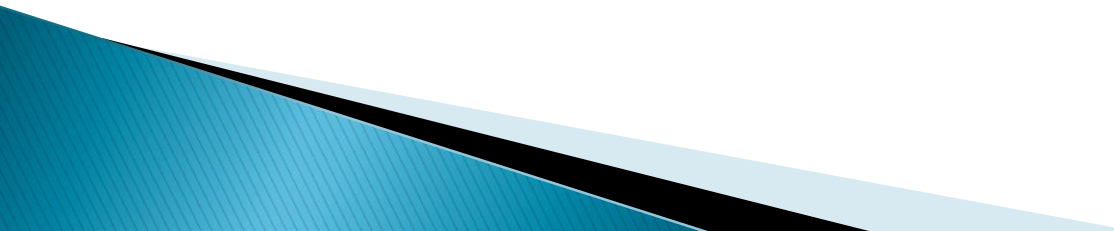


# Java

osnove objektno orijentisanog programiranja

# 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.
- 

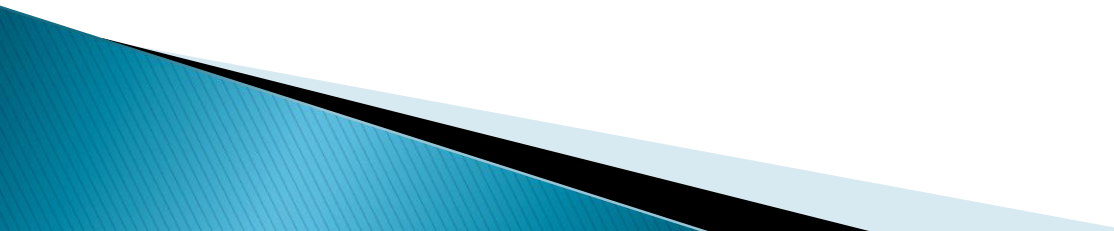
# Kako grupisati podatke o nekom entitetu?

## ► Nizovima?

```
String[][] osoba = new String[2][3] ();  
osoba[0][0] = "Pera";  
osoba[0][1] = "Perić";  
osoba[0][3] = "11.06.1990";  
  
osoba[1][0] = "Đura";  
osoba[1][1] = "Đurić";  
osoba[1][3] = "30.01.1980";
```

## ► Klasama? Mnogo prirodnije

# 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.
- 

# Sve je objekat

- ▶ Dakle, klasa je model objekta i uključuje attribute i metode
- ▶ U Javi je sve objekat, nije moguće definisati funkcije i promenljive izvan neke klase, zato listing počinje ključno reči *class*
- ▶ Instance neke klase se zovu objekti
- ▶ Objekti se kreiraju upotrebom ključne reči *new*

# Primer klase

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

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

# Primer kako se instancira i koristi

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

# Metode u klasi

- ▶ izdvojeni skup programskog koda koji se može pozvati (izvršiti) u bilo kom trenutku u programu
- ▶ najlakše objasniti ako se posmatraju kao podprogrami (podalgoritmi) specifične namene
- ▶ Definicija funckije:  
    povratni\_tip *ime\_funkcije* (parametri) {  
        programski kod  
    }



# Metode u klasi

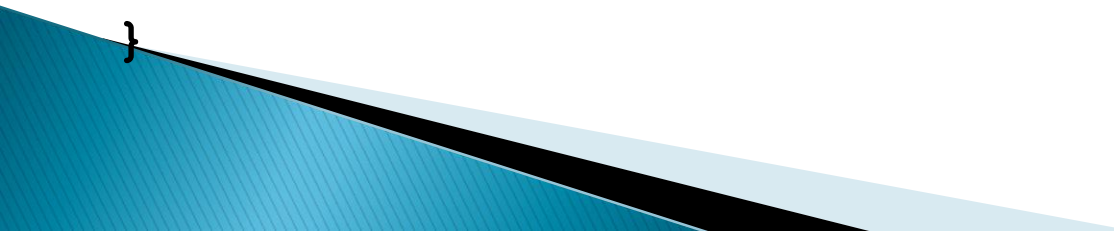
- ▶ U klasi može da postoji više metoda sa istim imenom
- ▶ Razlikuju se po parametrima, metode se nikada ne razlikuju samo po povratnoj vrednosti
- ▶ Parametri mogu biti: primitivni tipovi, reference na objekte
- ▶ Rezultat može biti: primitivni tip, referenca na objekat
- ▶ Ukoliko metoda ne vraća povratnu vrednost navodi se rezervisana reč null u deklaraciji funkcije
- ▶ Metoda vraća vrednost naredbom :
  - ▶ return vrednost;

# Ključna reč `this`

- ▶ Ključna reč `this` je referenca na objekt nad kojim je pozvana metoda

# 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`

- ▶ Ovako nešto se često koristi unutar klasa u Javi

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

# Default vrednosti atributa

- Atributi klasa imaju podrazumevane vrednosti (lokalne promenljive u metodama nemaju podrazumevane vrednosti i izazivaju grešku prilikom kompajliranja)

<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 imaju podrazumevanu vrednost null što može izazvati NullPointerException grešku u radu aplikacije (runtime exception).

# Reference na objekte

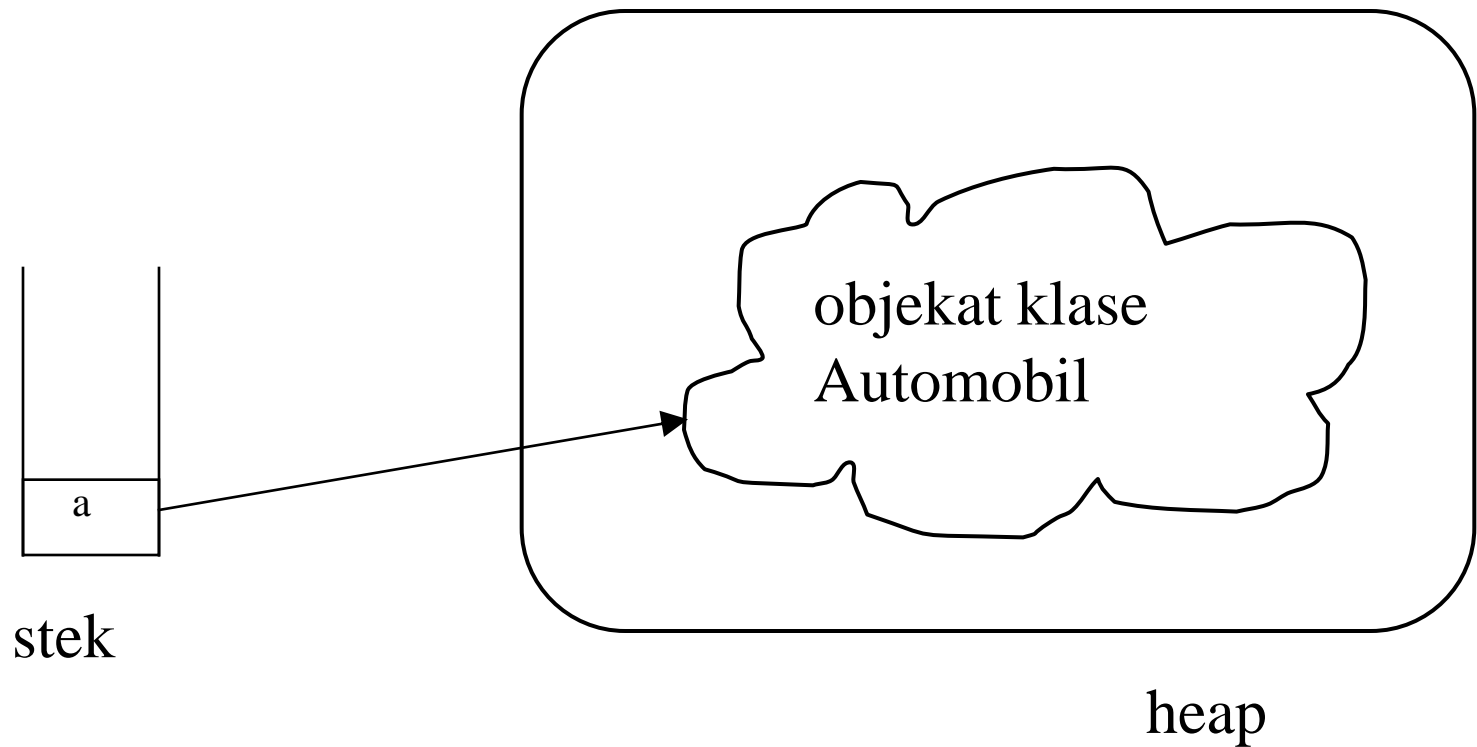
- ▶ Na steku su uvek samo reference na objekat, koji su zapravo na heap-u
- ▶ U Javi je sve referenca, nema pokazivaca
- ▶ Objekat takodje kao atribut moze imati referencu na drugi objekat koji je na heap-u

```
Automobil a;
```

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

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

# Reference na objekte

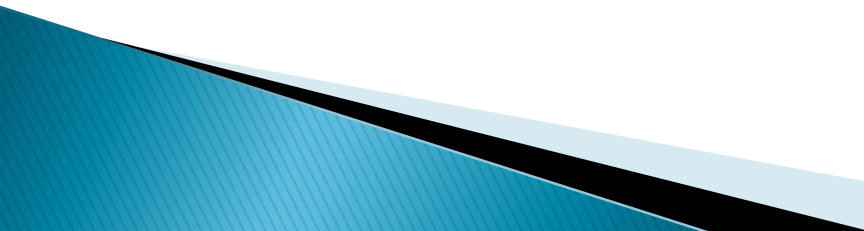


# Referenca na objekat kao parametar metode

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

...

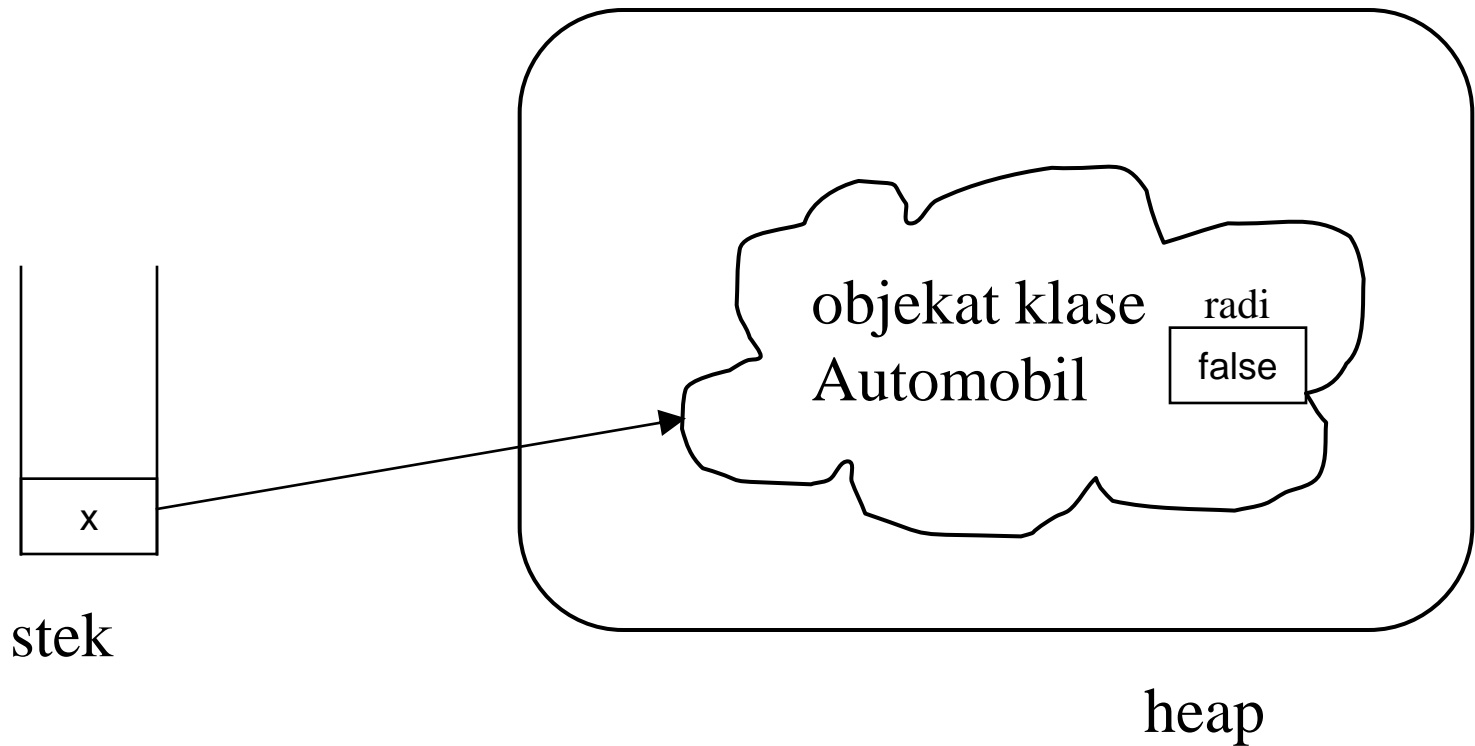
```
Automobil x = new Automobil();  
x.radi = false;  
test(x);
```





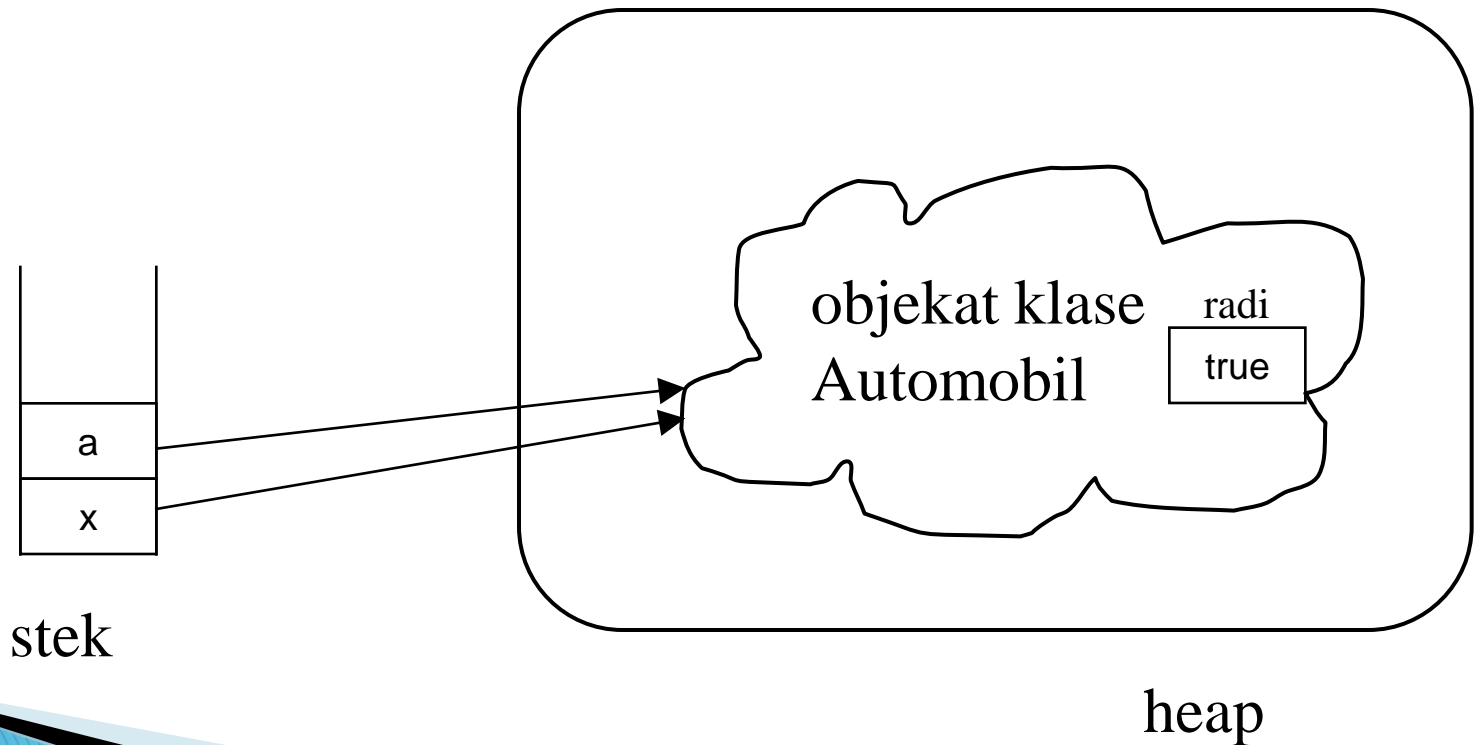
# Referenca na objekat kao parametar metode

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



# Referenca na objekat kao parametar metode

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



# null literal

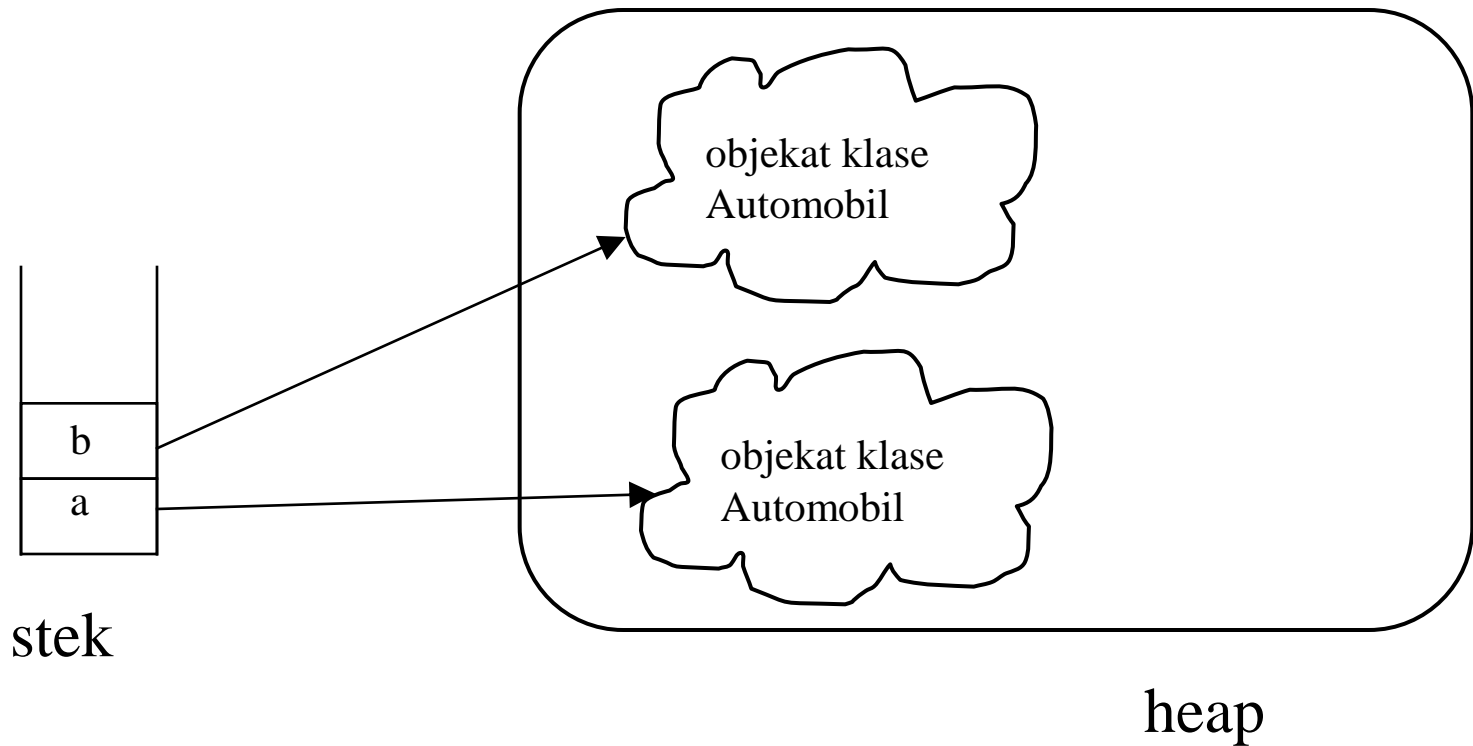
- ▶ Ako želimo da inicijalizujemo referencu tako da ona ne ukazuje ni na jedan objekat, onda takvoj promenljivoj dodeljujemo **null** vrednost, odn. **null** literal:

```
Automobil a = null;
```

# Operator dodele vrednosti

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

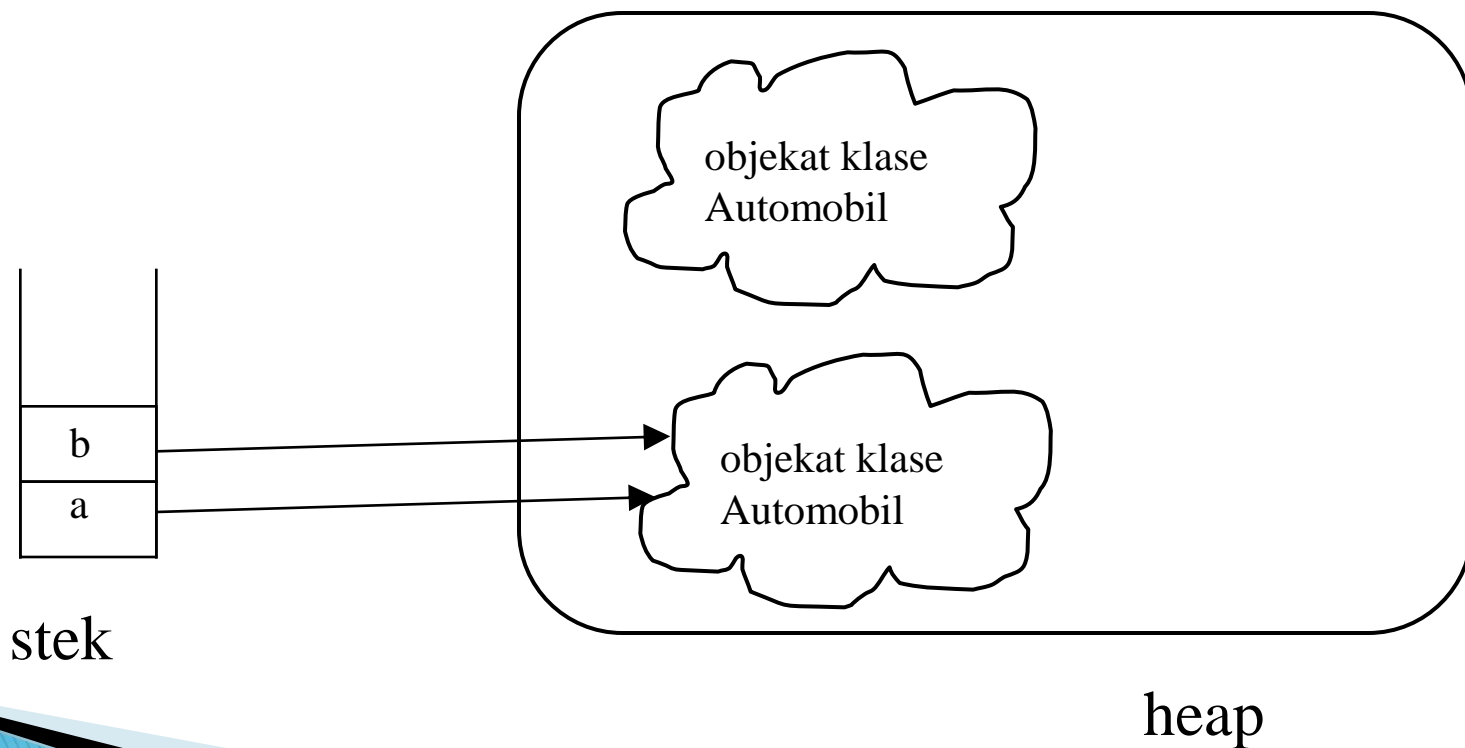
```
Automobil b = new Automobil();
```



# Reference na objekte

$b = a$

Operator dodele vrednosti nad objektima  
samo kopira referencu, ne i objekat



# Nizovi i objekti

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

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

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

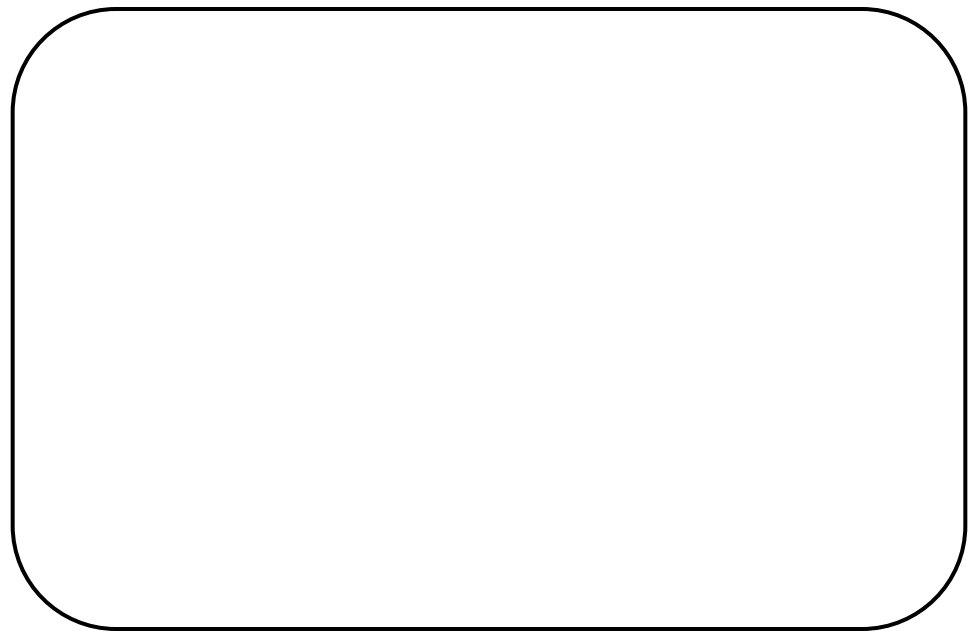
```
Automobil[] parking = new Automobil[20];  
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

# Nizovi referenci na objekte 1/3

```
Automobil[] parking;
```



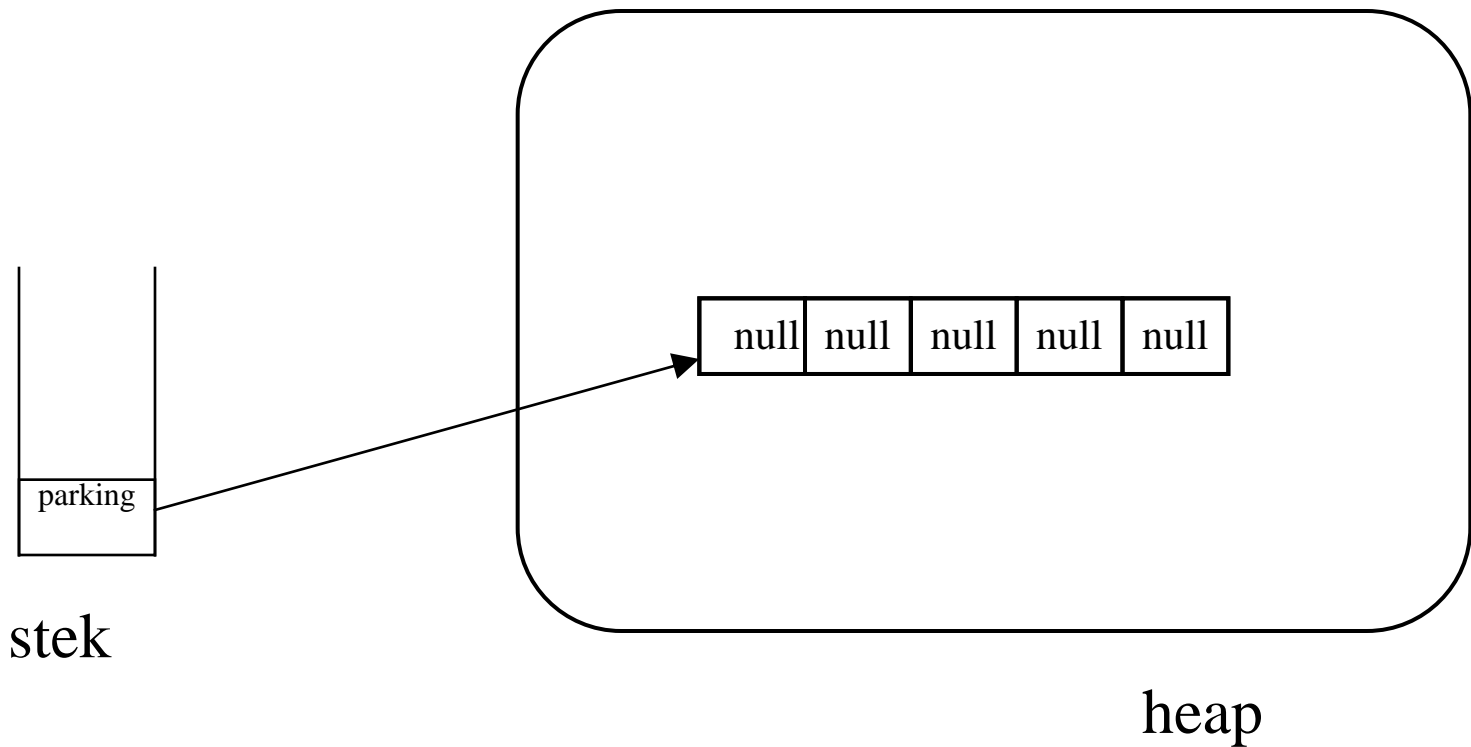
stek



heap

# Nizovi referenci na objekte <sub>2/3</sub>

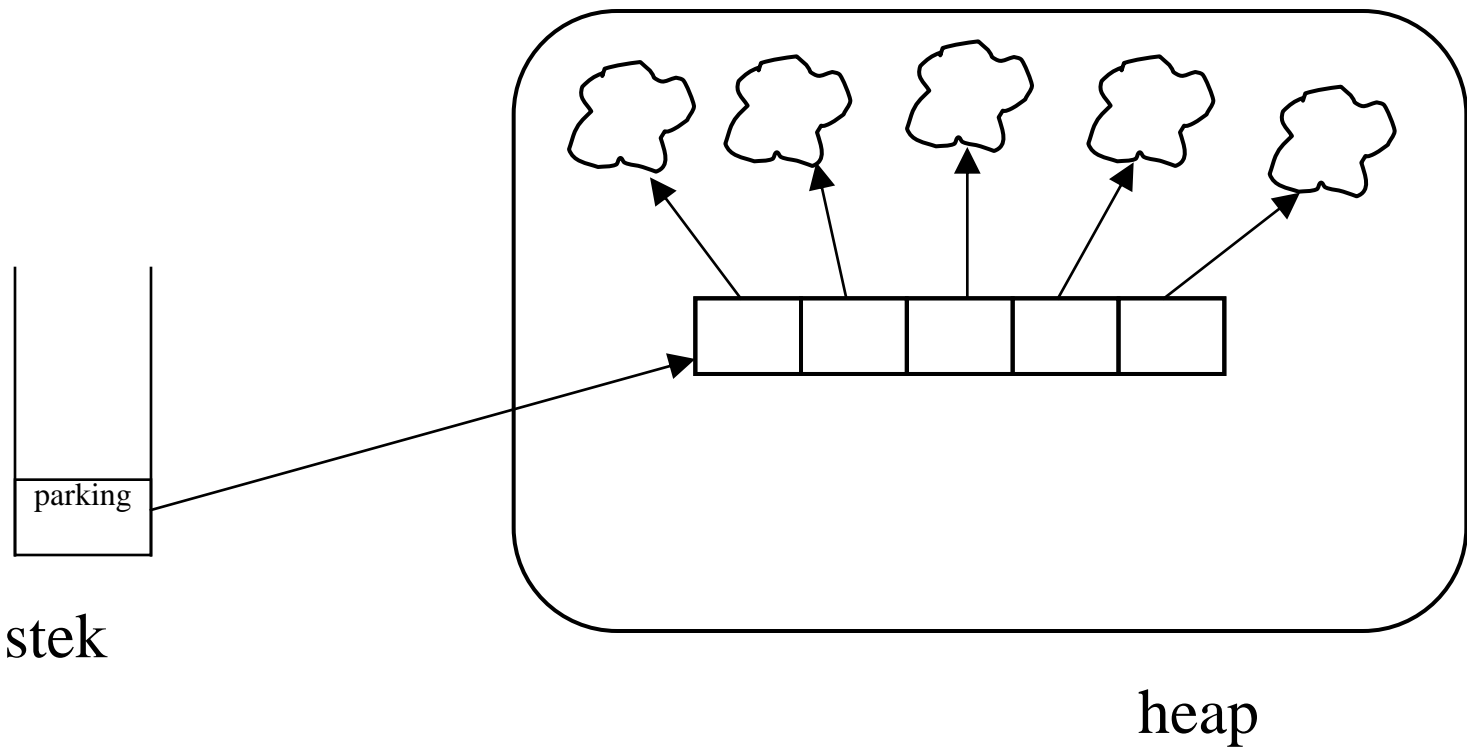
```
parking = new Automobil[5];
```





# Nizovi referenci na objekte 3/3

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



# Kreiranje i popunjavanje

- ▶ Može i jednostavnije kreiranje i popunjavanje:

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

- ▶ ili:

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

# Inicijalizacija objekata

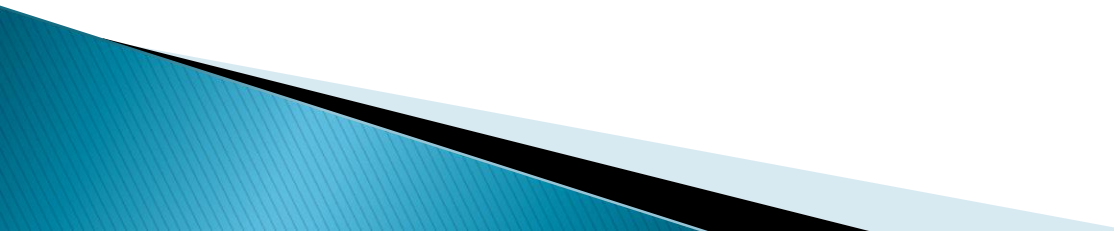
- ▶ Ako želimo posebnu akciju prilikom kreiranja objekta neke klase, napravićemo konstruktor
- ▶ Konstruktor se automatski poziva prilikom kreiranja objekta

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

- ▶ Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi
- ▶ U konstruktoru inicijalizujemo attribute koji bi trebalo da su inicijalizovani
- ▶ Konstruktor može primiti i parametre

# Konstruktor sa parametrima

```
/** 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;  
}  
}
```



# Konstruktor sa parametrima

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

# Uništavanje objekata – Garbage collector

- ▶ Ne postoji destruktor
- ▶ 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

- ▶ Možemo napisati posebnu metodu `finalize()`, koja se poziva neposredno pre oslobađanja memorije koju je objekat zauzimao
  - nemamo garanciju da će biti pozvana
- ▶ I pored Garbage Collector-a može doći do `OutOfMemory` ako ne vodimo računa

# Modifikatori pristupa

- ▶ **public** – vidljiv za sve klase
- ▶ **protected** – vidljiv samo za klase naslednice i klase iz istog paketa
- ▶ **private** – vidljiv samo unutar svoje klase
- ▶ nespecificiran (*friendly*) – vidljiv samo za klase iz istog paketa (direktorijuma, foldera)



# getters & setters

- ▶ Ponekad je potrebno obezbediti kontrolisan pristup atributima, kako za čitanje, tako i za pisanje.
- ▶ To se postiže posanjem odgovarajućih metoda kroz koje se pristupa atributima:

```
public class Student {  
    private String ime;  
    public String getIme() {  
        return ime;  
    }  
    public void setIme(String ime) {  
        this.ime = ime;  
    }  
}
```

# getters & setters

- ▶ Ova kombinacija atributa i njegovog getter-a i setter-a se još zove i svojstvo (*property*).
- ▶ Ovim je omogućeno da se čitanje vrednosti svojstva sprovodi kroz njegov getter, a izmena kroz setter.
- ▶ Ako izostavimo setter, dobijamo *read only* svojstvo.

# Ključna reč `static`

- ▶ Definiše statičke attribute i metode

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

- ▶ Statički atributi i metode postoje i bez kreiranje 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

# Ključna reč `static`

- ▶ Namena statičkih metoda:
  - pristup i rad sa statičkim atributima
  - opšte metode za koje nije potrebno da se kreira objekat
- ▶ Primeri upotrebe:
- ▶ `System.out.println();` // out je staticki atribut, ovo ostalo su staticke metode
- ▶ `Math.random();`
- ▶ `Math.sin();`
- ▶ `Math.PI;`
- ▶ `public static void main(String[] args) {...}`
- ▶ `java Hello` → `Hello.main(args)` //ovo se u stvari desava kada pokrenemo program

# 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();  
    }  
}
```

# Klasa Object

- ▶ Sve Java klase direktno ili indirektno nasleđuju klasu Object
- ▶ Klasa Object definiše osnovne metode koje imaju svi objekti u Javi a to su
  - equals(o), toString(), hashCode(), getClass(), clone(), finalize(), wait(), notify(), notifyAll()

# Konvencije davanja imena

- ▶ Nazivi klasa pišu se malim slovima, ali sa početnim velikim slovom (npr. Automobil, ArrayList).
- ▶ Ukoliko se naziv klase sastoji iz više reči, reči se spajaju i svaka od njih počinje velikim slovom (npr. HashMap).
- ▶ Nazivi metoda i atributa pišu se malim slovima (npr. size, width). Ako se sastoje od više reči, one se spajaju, pri čemu sve reči počevši od druge počinju velikim slovom (npr. setSize, handleMessage).
- ▶ Nazivi paketa pišu se isključivo malim slovima. Ukoliko se sastoje iz više reči, reči se spajaju (npr. mojpaket, velikipaket.malipaket).
- ▶ Detaljan opis konvencija nalazi se na adresi <http://www.oracle.com/technetwork/java/codeconv-138413.html>.



# Konvencije davanja imena

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