# KLASE I OBJEKTI

2020/21

PRIRODNO-MATEMATIČKI FAKULTET, UNIVERZITET U KRAGUJEVCU

# KLASA

**Predstavlja implementaciju tipa.**

**Klasa je tip, objekat primerak tipa.**

- Klasom se opisuju objekti sa istim
  - karakteristikama (podaci članovi)
  - ponašanjem (funkcionalnostima – metode)
- Podaci članovi (atributi)
  - svaki objekat ima sopstvene vrednosti podataka članova
  - trenutne vrednosti podataka objekta čine trenutno **stanje objekta**
- Funkcije članice (metodi)
  - njima su definisana **ponašanja objekta**
  - poziv metoda jednog objekta – **slanje poruke**
  - obrada zahteva tj. **odgovaranje na poruku**

# DEFINISANJE KLASE

```
<modifikator> class <className> {

    <modifikator> <tip> <imepromenljive1>;
    <modifikator> <tip> <imepromenljive2>;
    ...

    <modifikator> <povratni tip> <imemetoda1> (<tip> <arg1>,...) {
      ... //implementacija metoda 1
    }
    <modifikator> <povratni tip> <imemetoda2> (<tip> <arg1>,...) {
      ... //implementacija metoda 2
    }
    ...

}   // kraj definicije klase
```

**podaci članovi**

**funkcije članice - metodi**

# ZADATAK

Definisati klasu `Robot` (bez modifikatora) na sledeći način

| **Robot** – bez modifikatora | |
|---|---|
| rbr | – tipa integer, bez modifikatora |
| setrbr(int br) | – metod koji postavlja vrednost promenljive rbr |
| sayHello() | – metod koji ispisuje poruku *Hello, I am robot no. ___* |

Definisati aplikaciju `UseRobot` (preciznije `public` klasu `UseRobot`) koja u svom main metodu:

- kreira objekat klase `Robot`,
- postavlja vrednost `rbr` na 1,
- šalje poruku kreiranom objektu da se javi (`sayHello`).

# REFERENCE I OBJEKTI

DEFINISANJE VARIJABLE PRIMITIVNOG TIPA

```
int i = 2;
```


i
Primitive variable name

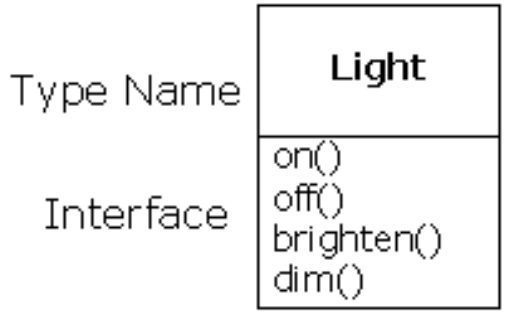
KREIRANJE OBJEKTA I REFERENCNE VARIJABLE

```
Light lt = new Light();
```


lt
Reference variable name


- **lt - referenca na objekat.**

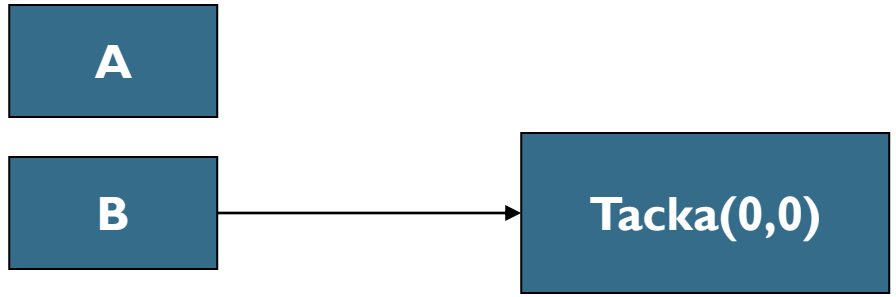
```
Light lt; // kreirana je samo referenca
Light lt = new Light(); // kreiran je i objekat
lt.on(); - slanje poruke objektu lt
```




# OBJECT VARIABLES - REFERENCE

```
class Tacka{
    double x;
    double y;
    public double getX()
    {
     return x;
    }
    public double getY()
    { return y;
    }
}
```

```
int x;       // x je varijabla primitivnog tipa
Tacka A;     // A je objektna varijabla (neinicijalizovana)
Tacka B = new Tacka(); // B je objektna varijabla (inicijalizovana)
```




```
double z = A.getX();  // greska, objekat ne postoji
double w = B.getX(); // OK
```

# METODI

# METODI

- Povratni tip i ime metoda su obavezni, kao i ()

```
<modifikator> <povratni tip> <ime metoda> (<argumenti>) {
// Telo metoda.
}
```

- Metodi se definišu **samo kao deo klase**. Pozivi pogrešnih metoda za neki objekat se registruju pri kompajliranju.

```
int x = a.f(); // a je objekat odgovarajuce klase
```

- Vraćanje sa bilo koje tačke, ali sa odgovarajućim tipom.

```
float naturalLogBase() { return 2.718f; }
void nothing() { return; }
```

skup metoda - interface

Poseban tretman imaju static metodi.

varijabilan skup argumenata (pogledati)

# OVERLOADNIG – PREOPTEREĆIVANJE METODA

## OVERLOADING

- U jednoj klasi (ili pri nasleđivanju) se može definisati više metoda sa istim imenom, ali **različitim potpisom**.
- Istoimeni metodi se moraju razlikovati po broju ili bar po tipu argumenata.
- Dozvoljeno je i da vraćaju različite tipove (ali pod uslovom da se razlikuju po argumentima).

# OVERLOADNIG – PRIMER

```
public class test {

public static void main(String[] args) {
Broj b = new Broj();
b.uvecaj(1);   // 1
b.uvecaj(1.0);   // 2
b.uvecaj(1,1);   // 3
b.uvecaj(1.0, 2.0);  // ?
}

}
class Broj
{
int vrednost = 10;
int uvecaj(int i) { return vrednost + i; }    // 1
double uvecaj(double i) { return vrednost + i; }  // 2
double uvecaj(float i, float j) { return vrednost + i + j;}  // 3
}
```

# KONSTRUKTORI

- Konstruktor je specijalan vrsta metoda koji se koristi isključivo pri konstrukciji objekata.

```
Tacka B = new Tacka();
```
← Poziv konstruktora

- Karakteristike
  - ima isto ime kao i klasa
  - poziva se isključivo pri instanciranju objekata (dakle, operatorom new),
  - nema povratne vrednosti
- Klasa može imati više konstruktora (overloading)
- Konstruktor bez parametara je default konstruktor i on postoji kada klasa nema posebno implementiran (naveden) ni jedan konstuktor, u suprotnom neće postojati.

# KONSTRUKTORI

```
class Tacka{
 private double x;
 private double y;
      public Tacka() { x=0.0; y=0.0; }
      public Tacka(double a, double b) { x=a; y=b; }
 public double getX() { return x; }
 public double getY() { return y; }
}
  public class Test {
   public static void main(String[] args){
    Tacka a = new Tacka();
    Tacka b = new Tacka(1,1);
    System.out.println("A(" + a.getX() + "," + a.getY() + ")\n");
    System.out.println("B(" + b.getX() + "," + b.getY() + ")\n");
      }
  }
```

konstruktori

# TIPOVI I PROMENLJIVE

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

# PRIMITIVNI TIPOVI I VRAPERSKE KLASE

| TYPE | DESCRIPTION | DEFAULT | SIZE | EXAMPLE LITERALS | RANGE OF VALUES |
|---|---|---|---|---|---|
| boolean | true or false | false | 1 bit | true, false | true, false |
| byte | twos complement integer | 0 | 8 bits | (none) | -128 to 127 |
| char | unicode character | \u0000 | 16 bits | 'a', '\u0041', '\101', '\\', '\','\n',' β' | character representation of ASCII values 0 to 255 |
| short | twos complement integer | 0 | 16 bits | (none) | -32,768 to 32,767 |
| int | twos complement integer | 0 | 32 bits | -2, -1, 0, 1, 2 | -2,147,483,648 to 2,147,483,647 |
| long | twos complement integer | 0 | 64 bits | -2L, -1L, 0L, 1L, 2L | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| float | IEEE 754 floating point | 0.0 | 32 bits | 1.23e100f, -1.23e-100f, .3f, 3.14F | upto 7 decimal digits |
| double | IEEE 754 floating point | 0.0 | 64 bits | 1.23456e300d, -1.23456e-300d, 1e1d | upto 16 decimal digits |

# PRIMITIVNI TIPOVI I VRAPERSKE KLASE

| Primitive type | Wrapper class |
|---|---|
| boolean | Boolean |
| byte | Byte |
| char | Character |
| float | Float |
| int | Integer |
| long | Long |
| short | Short |
| double | Double |

https://docs.oracle.com/javase/tutorial/java/data/autoboxing.html

# AUTOMATSKE PROMENLJIVE INCIJALIZACIJA I OBLAST VAŽENJA

- Incijalizacija automatskih promenljivih je obavezna, tj. forsirana od strane kompajlera.
- Oblast važenja (scope) podrazumeva vidljivost i životni vek ‘imena’ i Java je definiše slično C-u i C++-u

```
{ int x = 12;
    /* samo je x dostupno*/
    { int q = 96;
        /* x i q su dostupni */
    }
    /* samo x je dostupno a q ‘ne postoji’ */
}
```

- Za razliku od C-a u Javi nije dozvoljeno sledeće

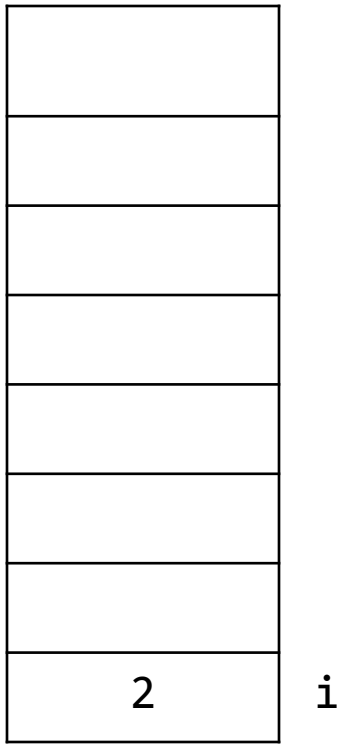
```
{ int x = 12;
    {
        int x = 96; /* nepravilno */
    }
}
```

- **STACK (stek)** je deo memorije koji je rezervisan za čuvanje lokalnih promenljivih, parametara procedura, povratnih adresa itd.

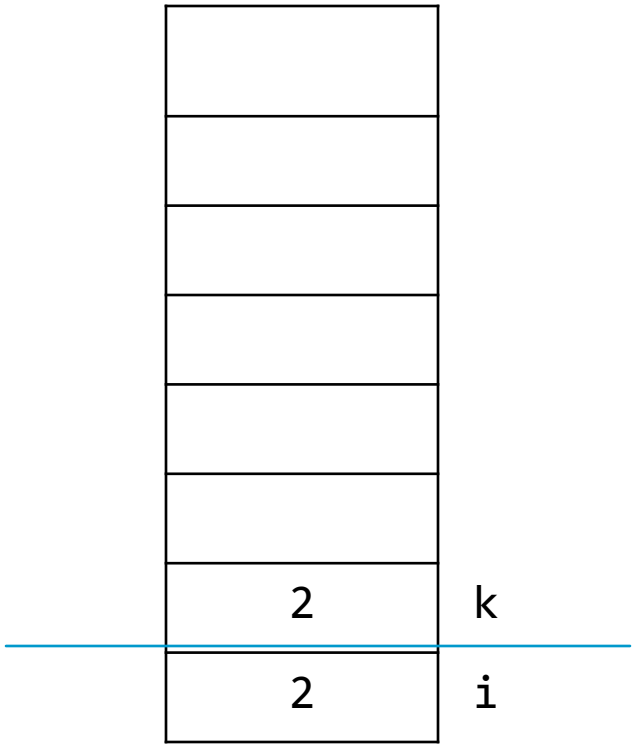
```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```

STACK (STEK)

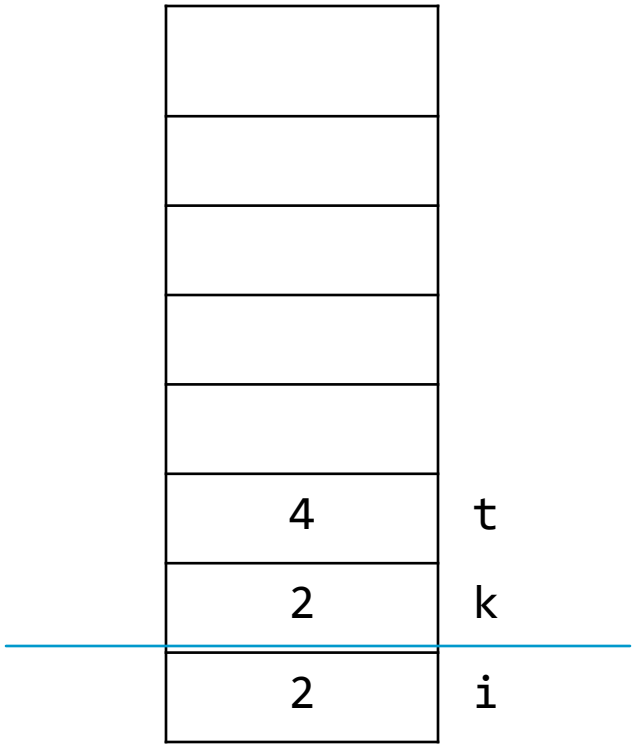
```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```



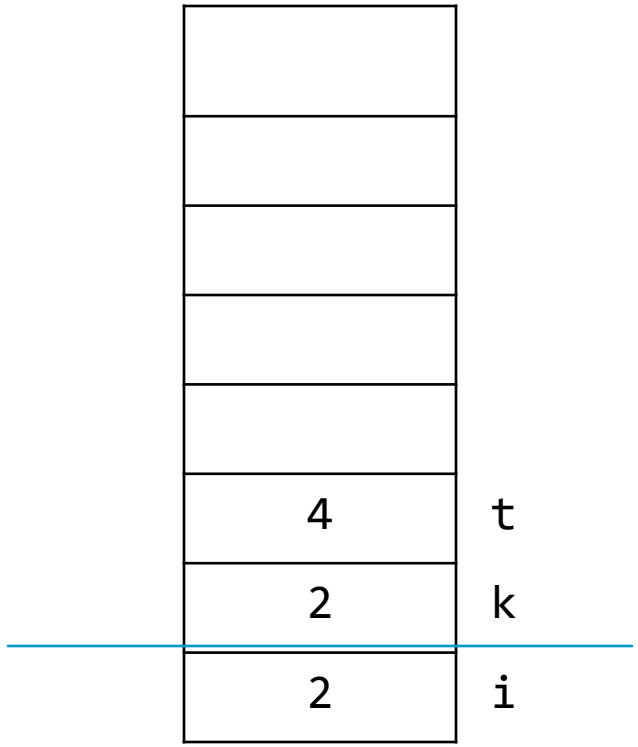
```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```



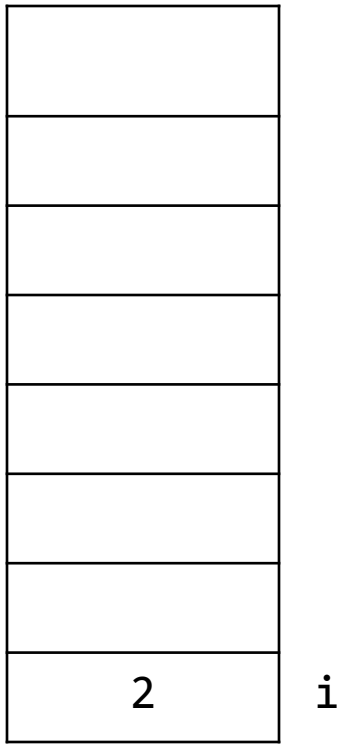
```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```



```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```



```
main(…)
{
  int i = 2;
  func(i);
  . . .
  return;
}

void func(int k)
{
  int t = k * k;
  . . .
  return;
}
```



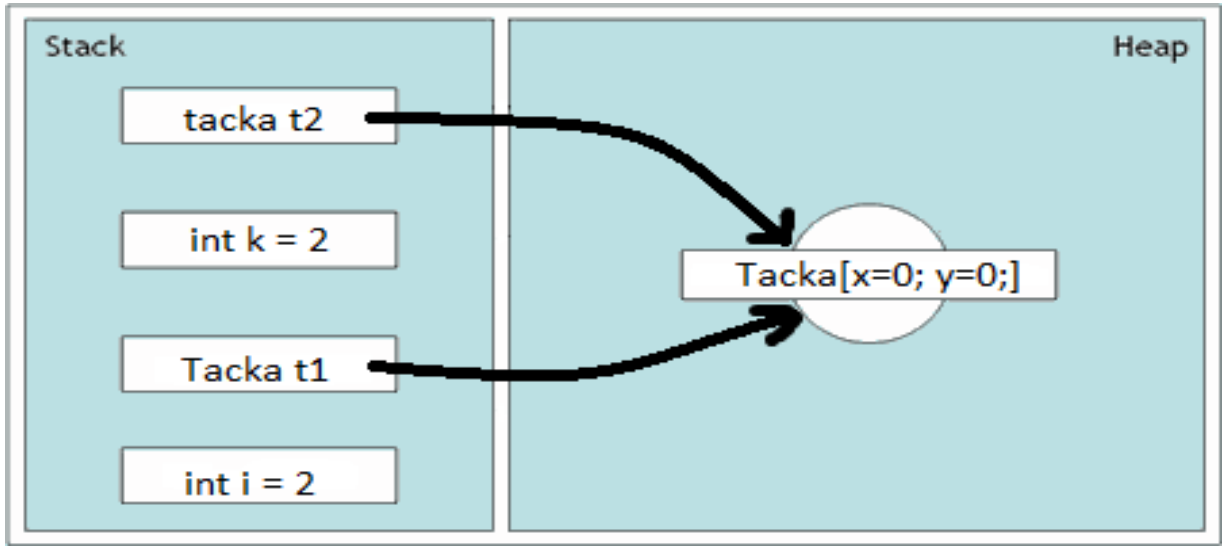
# OBLAST VAŽENJA REFERENCI I ŽIVOTNI VEK OBJEKATA

- **Životni vek objekata nije isti životnom veku primitivnih tipova.** Nakon kreiranja objekat postoji i posle }, jer je iz oblasti važenja ‘izašla‘ samo referenca

```
{
    Tacka s = new Tacka();
}
```

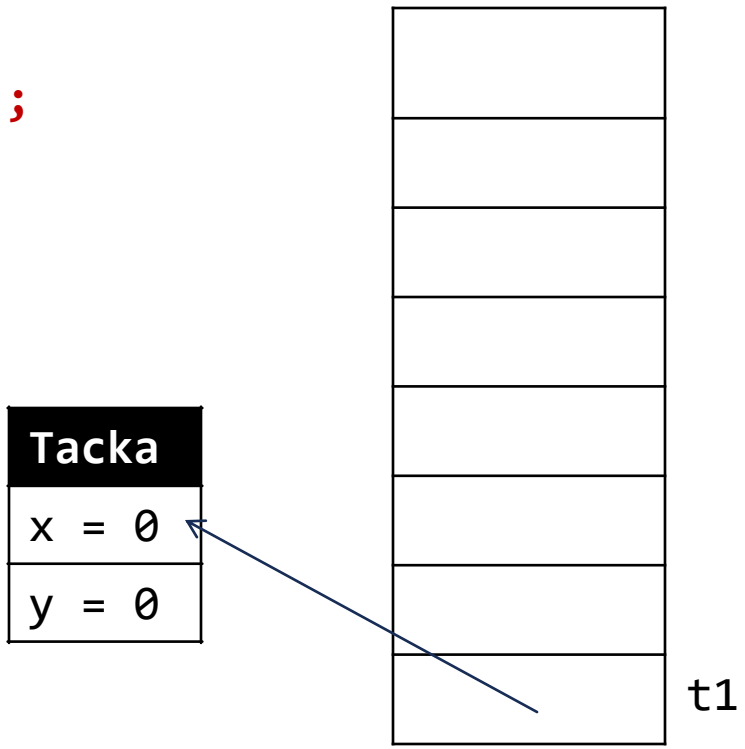
# HEAP (HRPA)

- **STACK (stek)** nije zadužen za čuvanje objekata. Na stek se pakuju samo reference.
- Objekti se čuvaju u delu memorije koji se naziva **HEAP (hrpa)**.
- **HEAP (hrpa)** je deo memorije koji je rezervisan za dinamičke promenljive, tj. za promenljive koje se stvaraju u toku izvršavanja programa.
- Pristup dinamički kreiranim promenljivama se ostvaruje pomoću pokazivača (referenci). **Na jednu istu dinamičku promenljivu može pokazivati i više pokazivača.**

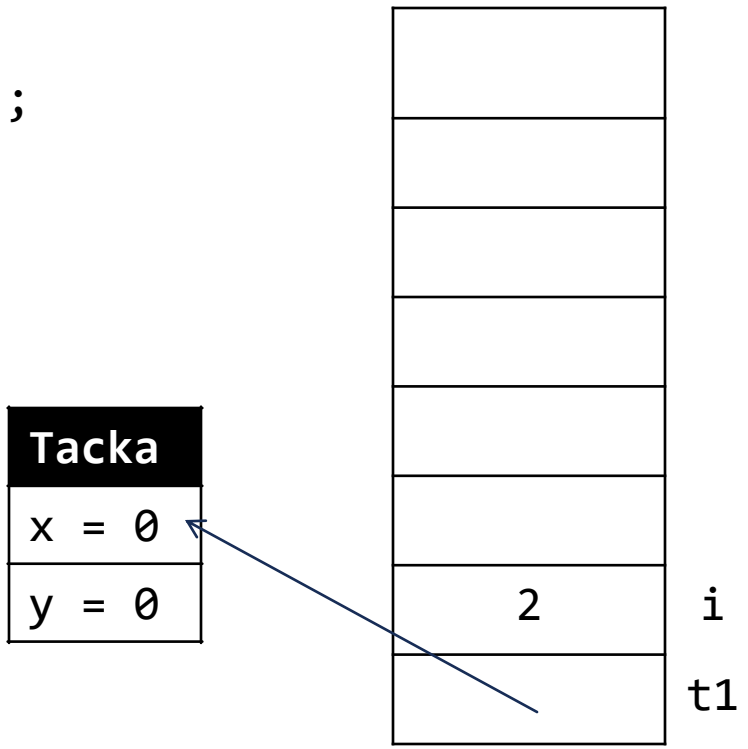


```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```

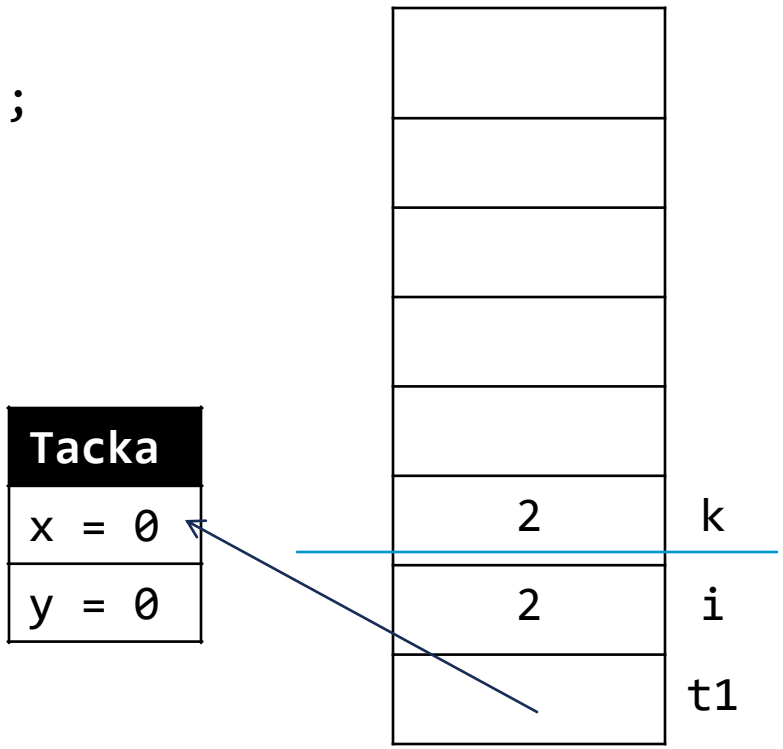
```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```



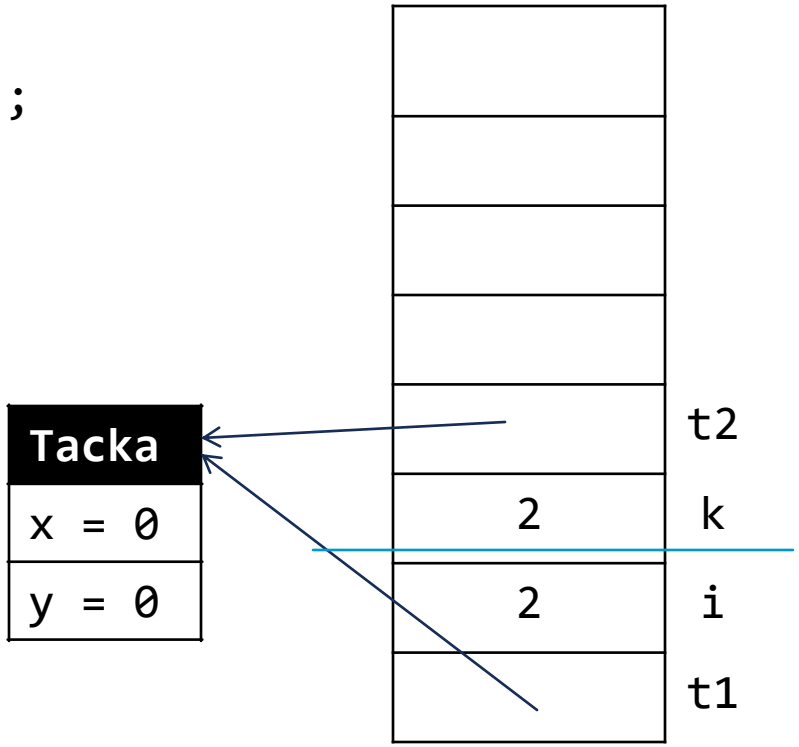
```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```



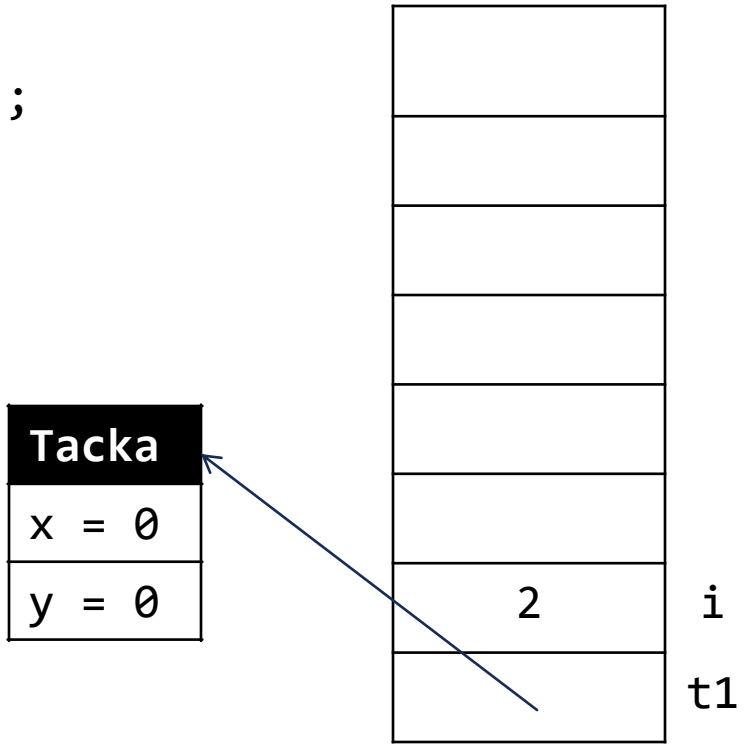
```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```



```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```



```
main(…)
{
  Tacka t1 = new Tacka();
  int i = 2;

  if(i  > 1 )
  {
    k = 2;
    Tacka t2 = t1;
  }

  return;
}
```



# GARBAGE COLLECTOR

- Kada prestane oblast važenja reference briše se samo referenca sa steka, ali ne i objekat. **Objekat ostaje u memoriji na heapu**.

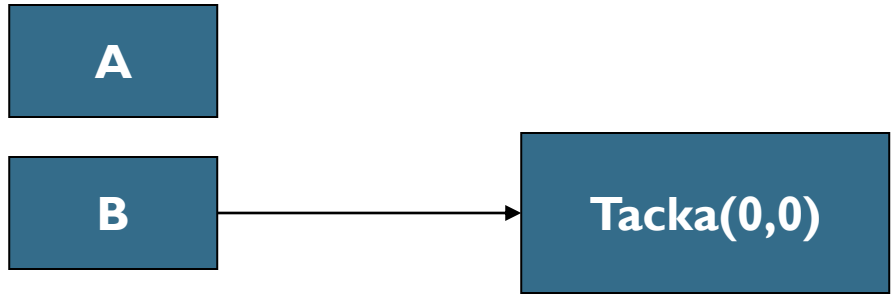
- JVM vodi računa o broju referenci na objekat, te stoga objekat ne sme biti obrisan iz memorije ukoliko je obrisana jedna referenca. Objekat  sme biti obrisan samo kada više ne postoje reference na isti.

- Postoji posebna nit koja vodi računa i uklanja objekte koji nisu referencirani - **Garbage Collector**. Ona se izvršava kada to dozvoljava procesorsko vreme ili kada je neophodno osloboditi memoriju.

# OBJECT VARIABLES - REFERENCE

```
class Tacka{
    double x;
    double y;
    public double getX()
    {
     return x;
    }
    public double getY()
    { return y;
    }
}
```

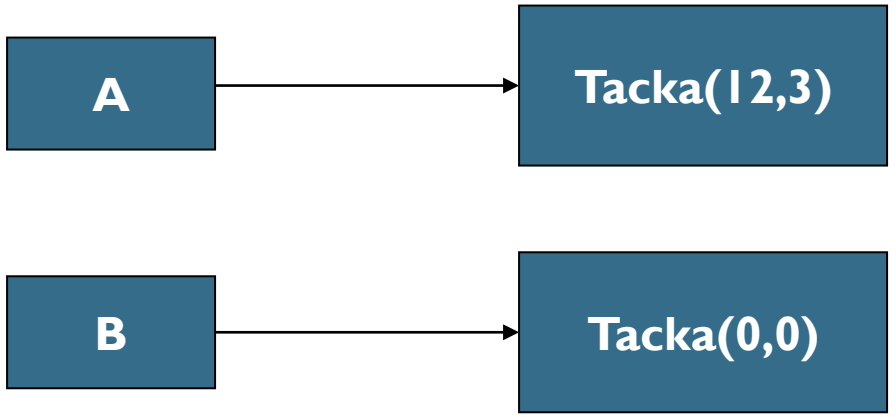
```
int x;       // x je varijabla primitivnog tipa
Tacka A;     // A je objektna varijabla (neinicijalizovana)
Tacka B = new Tacka(); // B je objektna varijabla (inicijalizovana)
```




```
double z = A.getX();  // greska, objekat ne postoji
double w = B.getX(); // OK
```

# OBJECT VARIABLES - REFERENCE

```
A = new Tacka(); A.x = 12; A.y = 3;
```




```
A = B; - ne kreira se novi objekat vec
         A postaje referenca na vec
         postojeci objekat
```