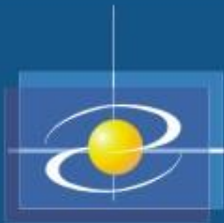


Programski jezik JAVA

PREDAVANJE 2

2022

Prezentacija kreirana na osnovu sljedeće literature :
Dejan Živković: Osnove Java programiranja; Bruce Eckel: Misliti na Javi

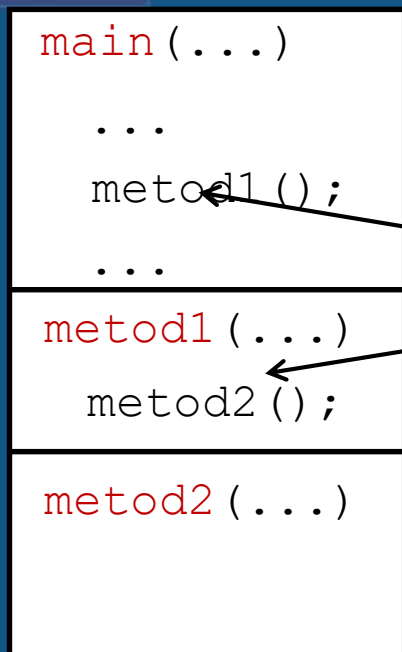


Metode

- Metod je potprogram za specifičan zadatak koji je dio glavnog zadatka programa
- Prednosti metoda
 - Manje djelove programa (metode) je lakše napisati i testirati
 - Metodi se mogu nezavisno i paralelno pisati od strane više programera
 - Metodi se mogu više puta koristiti
 - Metodi smanjuju ukupnu veličinu programskog koda
- Metod je samostalni blok Java koda koji se sastoji od deklaracija i naredbi
 - metod ima posebno ime
 - metod se može izvršiti (pozvati) u različitim delovima programa



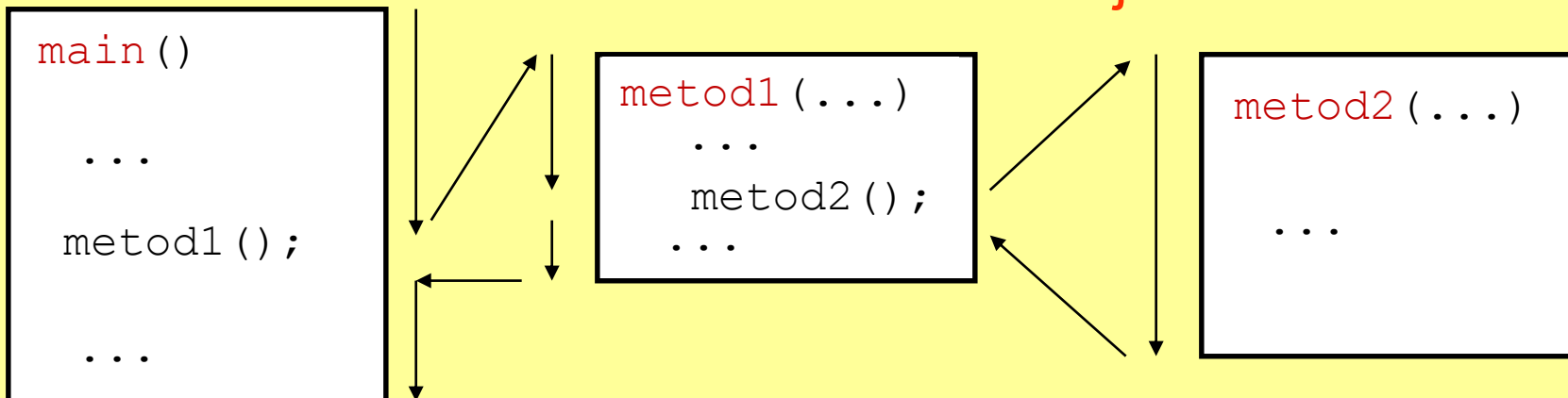
Metode



Struktura metoda

pozivi metoda metod1 i metod2

Izvršavanje metoda





Metode i njihovi parametri

- Klase sadrže dvije komponente: **promjenljive i metode**. Može se reći da metode definišu pristup podacima u većini klasa.
- Opšti oblik deklarisanja metoda glasi:

```
ModifikatorPristupa tip-rezultata ime-metoda (lista parametara) {  
    // tijelo metoda          }
```

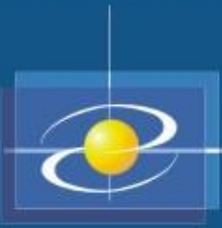
- **Modifikator pristupa** može biti *public* (označava da je metoda vidljiva za sve klase u programu), *protected* (metoda je vidljiva samo za klase nasljednice) i *private* (metoda je vidljiva samo unutar svoje klase). Modifikatori nisu obavezni.
- **Tip-rezultata** označava tip podatka koje metoda vraća. **Metod može vratiti najviše jednu vrijednost**. Rezultat se vraća pomoću naredbe **return**. Ukoliko metoda ne vraća vrijednost, njen tip povratnih podataka mora da bude označen sa *void*.
- **Ime-metoda** metode je određeno identifikatorom *ime*.
- **Lista parametara** sadrži niz parova, razdvojenih zarezima. Parametri su promjenljive koje prihvataju vrijednosti argumenata proslijeđenih metodi u trenutku njenog pozivanja.



Modifikatori pristupa

- **private**
 - Pristup je limitiran u okviru klase u kojoj je deklarisan;
 - Na primer: `private int x`.
- **default** (znači da se ne koristi nijedan modifikator)
 - Pristup je limitiran u okviru paketa u kojoj je član deklarisan;
 - Npr: `int x`;
- **protected**
 - Pristup je limitiran u okviru paketa u kojoj je član deklarisan, kao i u okviru podklasa te klase.
 - Npr: `protected void setName() {...}`
- **public**
 - Članovi su pristupačni svim klasama u svim paketima
 - Npr: `public String getName() {...}`

Pozivanje metoda



- Format:
`ime-metoda(Tista-argumenata)`
- U zagradama se umesto formalnih parametara navode stvarni argumenti
- U zagradama se navode samo argumenti, bez njihovog tipa
- Argumenti u pozivu metoda se razdvajaju zarezima
- Ako metod nema parametre, obavezne su prazne zagrade i u definiciji i u pozivu metoda
- Argumenti u pozivu i parametri u definiciji metoda moraju se slagati po broju, tipu i redosljedu



Lokalne i globalne promenljive

- Promenljive (polja) se mogu definisati van metoda u klasama
- Oblast važenja imena: dio teksta programa u kojem se može koristiti definisano ime
- Pravilo: ime (identifikator) važi u bloku u kome je definisano
- Primjer 1:

```
void neispravanMetod(int n) {  
    int x;  
    while (n > 0) {  
        int x; // GREŠKA: ime x je već definisano  
        ...  
    }  
}
```



Lokalne i globalne promenljive

- **Primjer 2:**

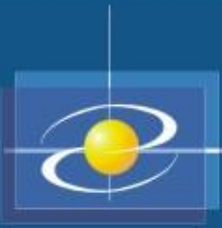
```
public class IgraSaKartama {  
    String pobjednik;        // globalna promenljiva (polje)  
    ...  
    void odigrajIgru() {  
        String igrač;        // lokalna promenljiva  
        // Ostale naredbe metoda ...  
    }  
    ...  
}
```

- **Primjer 3:**

```
for (int i = 0; i < n; i++) {  
    // Tijelo petlje  
    . . .  
}
```

```
if (i == n) // GREŠKA: ovdje ne važi ime i  
    System.out.println("Završene sve iteracije");
```


Rekurzivni metodi

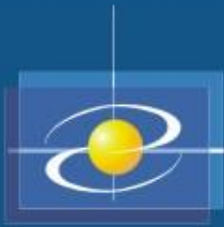


- Da li metod može da poziva sam sebe?
- Da – rekurzivan metod
- Mehanizam pozivanja rekurzivnih metoda se ne razlikuje od standardnog načina za pozivanje običnih metoda:
 1. argumenti u pozivu metoda se dodjeljuju parametrima metoda kao početne vrijednosti
 2. izvršava se tijelo pozvanog metoda
- Primer: izračunavanje stepena x^n , x realan broj i n cijeli broj
- Gotovo rješenje: `Math.pow(x, n)`



Osnovni principi OOP programiranja u Javi

- **Apstrakcija**
 - Program čine objekti koji su instance nekih klasa.
- **Enkapsulacija**
 - Podaci i metodi koje rade sa njima slažu se u cjeline – klase
 - Korisnika objekta ne zanima implementacija nego funkcionalnost
- **Nasljeđivanje**
 - Ideja: iskoristiti već postojeći kod
 - Klasa proširuje funkcionalnost već postojeće klase-`overriding`
 - Vrh hijerarhije klasa u Javi je klasa `java.lang.Object`.
- **Polimorfizam**
 - Često umjesto direktnog referenciranja na objekat iz proširene klase vrši referenciranje na objekat iz njegove bazne klase. Objekti koje pravimo u programu, mogu imati više oblika:
 - **Svaki objekat koji za svoj tip ima „stariju“ klasu, može se tretirati/poprimiti oblik i „mlađe“ klase.**



PRIMJER

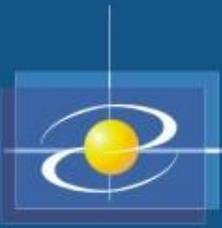
```
public class predmeti {  
  
    public void ispisati(){  
        System.out.println("Ovo je predmet");  
    }  
}
```

```
public class java extends predmeti {  
  
    public void ispisati(){  
        System.out.println("Web programiranje");  
    }  
}
```

```
public static void main(String[] args){  
    predmeti j = new predmeti();  
    predmeti k = new java();  
    j.ispisati();  
    k.ispisati();  
}
```

REZULTAT:

```
Ovo je predmet  
Web programiranje
```



KLASE I OBJEKTI

- Klase imaju sljedeću strukturu:

```
class ImeKlase {  
    promjenljive  
    metode  
}
```

- Objekat je konkretna realizacija klase u memoriji računara:
 - Instanca klase u memoriji
 - Kreira se sa operatorom `new`

Tip podatka (ime klase)

```
Figura fig = new Figura()
```

Ime promjenljive Defaultni konstruktor

- Svaki objekat prije upotrebe mora se konstruisati pomoću operatora `new` koji poziva konstruktor objekta.



Klase i objekti

- Konstruktor vrši inicijalizaciju objekta njegove klase.
- Ima isto ime kao i klasa, ali nema povratni tip (čak ni `void`).
- U jednoj klasi može postojati više konstruktora.

```
class Tacka{ double x; // x -koordinata tačke u ravni
            double y; // y -koordinata tačke u ravni
            }
```

- Konstrukcija objekta se vrši na sljedeći način:

```
Tacka toc = new Tacka();
```

- Šta ako želimo da zadamo tačku sa koordinatama (2,2)?
- Tada dodajemo još jedan konstruktor na sljedeći način:

```
class Tacka{
    double x;           // x -koordinata tačke u ravni
    double y;           // y -koordinata tačke u ravni
    //konstruktor
    public Tacka(double a, double b){
        x=a;
        y=b;
    }//kraj konstruktora
} //kraj klase
```

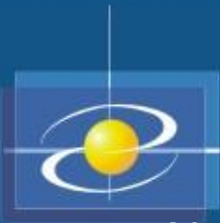


Klase i objekti

- Zašto smo u prethodnoj klasi mogli koristiti defaultni konstruktor, a da se ne piše odgovarajući kod? Odgovor: Ako nema nijednog **EKSPLICITNO** napisanog konstruktora, onda prevodilac **SAM** dodaje defaultni konstruktor.
- Ako klasu `Tacka.java` pokušamo da pozovemo na sljedeći način:

```
1 // Datoteka: Test.java
2
3 public class Test{
4
5     public static void main(String[] args){
6
7         Tacka toc = new Tacka(); // error
8     }
9
10 }
```

- U ovom slučaju javiće se greška prilikom prevođenja jer važi pravilo: **Ako imamo jedan konstruktor koji smo sami napisali tada prevodilac ne kreira defaultni konstruktor za nas!!!**



Klase i objekti

- Ako želimo možemo imati više konstruktora:

```
1 // Datoteka: Tacka.java
2
3 class Tacka{
4
5     double x; // x -koordinata
6     double y; // y -koordinata
7
8     // konstruktor
9     public Tacka(){
10         x=0.0;
11         y=0.0;
12     }
13
14     //konstruktor
15     public Tacka(double a, double
16     b){
17         x=a;
18         y=b;
19     }//kraj konstruktora
20 }//kraj klase Tacka
```

Poziv sljedeća dva konstruktora:

Tacka toc=new Tacka() i

Tacka toc=new Tacka(0,0) **kreiraju isti objekt.**

- Osnovno o konstruktorima se svodi na sljedeće:

Konstruktor ima isto ime kao i klasa

Konstruktor nema povratne vrijednosti

Klasa može imati više konstruktora koji se međusobno razlikuju po broju i tipu parametara koje uzimaju

Konstruktor bez parametara je defaultni konstruktor

Konstruktor se uvijek poziva s operatorom **new**

Klase i objekti - Primjer



```
1  class Figura {
2    // podatak
3    int boja_Figure;
4
5    // metode
6    void nacrtaj() {
7        // implementacija
8    }
9
10   void nacrtaj(int boja) {
11       // implementacija
12   }
13
14   String ime_figure(){
15       // implementacija
16   }
17 }
```

- Klasa je osnovna programska cjelina
- Klasa Figura: jedan podatak i tri metode
- Dvije metode istog imena
- Klasa figura nema main funkciju
- Klasa je samo nacrt



Instanciranje

- Kreiranje objekta date klase nazivamo *instanciranjem* objekta, a same objekte zovemo *instancama* date klase.
- Za instanciranje koristimo operator *new*.

```
1  class  KorisnikFigure{
2
3      public static void main(String[] args) {
4
5          // fig je objekat koji reprezentuje klasu Figura.
6
7
8          Figura fig=new Figura();
9
10         KorisnikFigure kf = new KorisnikFigure();
11         kf.ucini_nesto(fig);
12     }
13
14     void ucini_nesto(Figura x) {
15         x.nacrtaj();
16     }
17 }
```

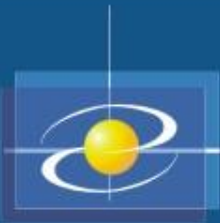


Instanciranje

- Da bi pozvali metodu neke klase, na primjer metodu `ucini_nesto` u klasi `KorisnikFigure`, ili metodu `nacrtaj` klase `Figura`, **MORAMO** prvo kreirati objekat te klase i zatim metodu pozvati na objektnoj promjenljivoj pomoću *dot* (`.`) *operatora*: na primjer,

```
kf.ucini_nesto(fig);
```

- **VAŽNO**: Statičke metode (deklarisane kao **static**) su posebne po tome što **ne** trebaju kreirati objekat da bi bile pozvane. Zato **main** mora biti **static** na početku izvršavanja programa.
- Nakon definicije klase **Figura** možemo deklarirati promjenljive i argumente metoda da budu tipa **Figura**. Svaka klasa koju formiramo postaje na taj način novi tip podatka.



Instanciranje

- Kvadrat je geometrijska figura, pa je logično modelovati tu klasu tako da naslijedi metode iz *Figure* i da definiše neke nove.

```
1  class Kvadrat extends Figura{
2
3  //nasljedjuje sve iz Figure i dodaje novu promjenljivu i metodu
4
5      int duzina_stranice;
6
7      int površina(){
8          return duzina_stranice*duzina_stranice;
9      }
10
11  //klasa Kvadrat preradjuje metodu nacrtaj
12
13      void nacrtaj(){ //neka nova implementacija
14          }
15  }
```

- Ključna riječ extends kazuje da se klasa *Kvadrat* implementira proširenjem klase *Figura*.



Instanciranje

- Dio koda koji instancira objekat date klase može izgledati ovako:
`Kvadrat kv=new Kvadrat();`
`kv.nacrtaj(); //taj nacrtaj() je preradjen u klasi Kvadrat`
`kv.nacrtaj(3); //ovaj nacrtaj(int) je iz klase Figura`
- Polimorfizam: Svaki kvadrat je ujedno figura, prema tome, klasu Kvadrat smo konstruisali proširenjem klase Figura.
- Na taj način, svaka metoda koja kao argument očekuje objekat tipa `Figura`, prihvata i objekat tipa `Kvadrat`. To se naziva **polimorfizam** – jedan objekat posmatramo na različitim nivoima apstrakcije.
- Polimorfizam ilustruje upotreba metode `ucini_nesto` klase `KorisnikFigure`

```
KorisnikFigure korisnik = new KorisnikFigure();
Kvadrat kv = new Kvadrat();
//pozovimo metodu ucini_nesto
korisnik.ucini_nesto(kv);
//pozvali smo metodu na objektu prosirene klase
```



Primjer za OOP Programiranje

- Krug.java

```
class Krug extends Figura{
    //vraca ime nase figure
    String ime_figure(){
    return "Krug";
    }
}
```

- Kvadrat.java

```
class Kvadrat extends Figura{
    //vraca ime nase figure
    String ime_figure(){
    return "Kvadrat";
    }
}
```

```
class Figura
{
    //definicija
}
```

Figura.java

```
class KorisnikFigure
{
    //definicija
}
```

KorisnikFigure.java

```
class Krug
{
    //definicija
}
```

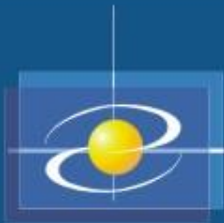
Krug.java

```
class Kvadrat
{
    //definicija
}
```

Kvadrat.java

Ukupan program se sastoji od četiri fajla

Kontrola pristupa



- Ako želimo da podatke u klasi sakrijemo onda dodajemo – private:

```
1 // Datoteka: Tacka.java
2
3 class Tacka{
4
5     private double x; // x -koordinata tačke u ravni
6     private double y; // y -koordinata tačke u ravni
7
8     public Tacka(){
9         x=0.0;
10        y=0.0;
11    }
12
13    public Tacka(double a, double b){
14        x=a;
15        y=b;
16    } //kraj konstruktora
17
18 } //kraj klase Tacka
```

- Na ovaj način pristup promjenljivima imaju samo metode (funkcije) iz iste klase.

Kontrola pristupa



- Na ovaj način smo SAKRILI implementaciju klase, ali smo napravili problem u klijentu. Posmatrajmo kod:

```
1 // Datoteka: Test.java
2
3 public class Test{
4
5     public static void main(String[] args){
6
7         Tacka toc = new Tacka();
8         System.out.println(toc.x); // error
9     }
10
11 }
```

Prevodeći: `javac Test.java` dobijamo poruku:

```
Test.java:8: x has private access in Tacka
System.out.println(toc.x); // error
                    ^
```

1 error

- Da bi ovo popravili moramo u klasi Tacka dodati dvije javne metode `getX()` i `getY()`. Ove dvije metode predstavljaju JAVNI INTERFEJS naše klase.

Kontrola pristupa

Popravljeni primjeri

```
1 // Datoteka: Tacka.java
2
3 class Tacka{
4
5     private double x; // x -
    koordinata tačke u ravni
6     private double y; // y -
    koordinata tačke u ravni
7
8     public Tacka(){
9         x=0.0;
10        y=0.0;
11    }
12
13    public Tacka(double a, double b){
14        x=a;
15        y=b;
16    } //kraj konstruktora
17
18    public double getX(){ return x;}
19    public double getY(){ return y;}
20
21 } //kraj klase Tacka
```

```
1 // Datoteka: Test.java
2
3 public class Test{
4
5     public static void main(String[] args){
6
7         Tacka toc = new Tacka();
8         System.out.println(toc.getX());
9     }
10
11 }
```




Osnovno o kontroli pristupa

- Kontrola pristupa primjenjuje se na promjenljive članice, metode (uključujući i konstruktore) i same klase. Preciznije:

public -pristup promjenljivoj/metodi dozvoljen je iz svih klasa koje instanciraju dati objekt

protected -pristup promjenljivoj/metodi dozvoljen iz svih klasa u paketu i svih naslijedjenih klasa

private -pristup promjenljivoj/metodi dozvoljen samo unutar date klase

(ništa) defaultni pristup -pristup promjenljivoj dozvoljen iz svih klasa unutar paketa (eng. **package**)

- Uz promjenljive je moguće koristiti i modifikatore: `static` i `final`.

static -označava promjenljivu koja je zajednička svim objektima koji instanciraju datu klasu

final -definise konstante

- Uz metode možemo koristiti sledeće modifikatore:

static - metoda koja je ista za svaki objekt date klase -kaže se još i *class member* npr. takva je metoda **main**

native - metoda koja je implementirana u drugom jeziku (najčešće C ili C++) -sve fundamentalne metode u API su takve

final - ne može se preraditi u naslijeđenoj klasi

synchronized - koristi se kod rada sa nitima (eng. *thread*)



Statičke promjenljive i metode

- Modifikator `static` označava nešto zajedničko u svim instancama date klase.

```
1 class Foo{
2     int x;
3     static int y;
4 }
```

- Svaka instanca klase `Foo` će imati svoju promjenljivu `x`.
- Sve instance će DIJELITI jednu promjenljivu `y`.
- Ako jedan objekat izmijeni promjenljivu `y` i svi ostali će imati izmijenjenu vrijednost.
- Statičku promjenljivu je moguće dohvatiti ako nije instanciran NIJEDAN objekat njene klase koristeći sintaksu:
`ImeKlase.ImePromjenljive`.

```
1 public class TestFoo{
2     public static void main(String[] args) {
3         System.out.println(Foo.y);
4     }
5 }
```



Statičke promjenljive i metode

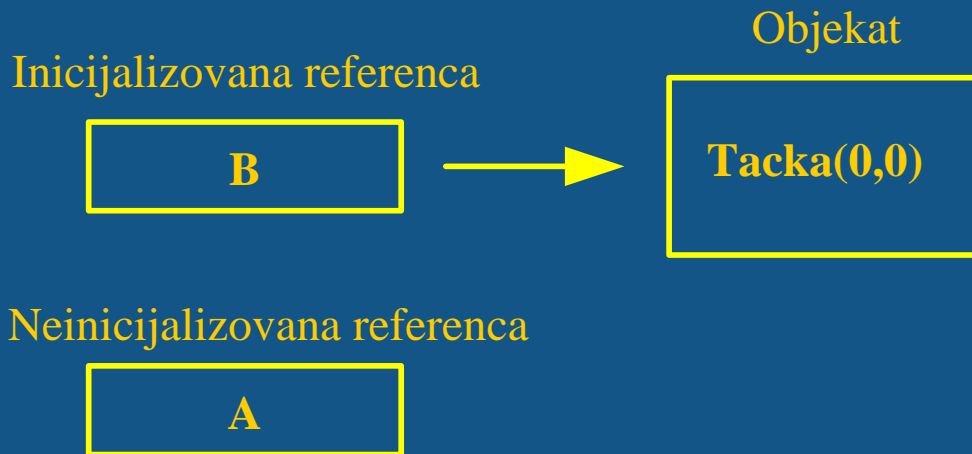
- Da bi koristili statičku metodu neke klase nije potrebno imati instancu te klase. Poziva se: `ImeKlase.ImeStatickeMetode`.
- Statičke metode ne mogu koristiti nestatičke metode iste klase direktno, niti pristupati nestatičkim promjenljivim. Primjer:

```
1  class Test{
2
3      int x;                // nestatička javna promjenljiva
4      static int y;
5
6      public void print(){ // nestatička metoda
7
8          System.out.println("Hello, World");
9      }
10
11     public static void print_static(){ // statička metoda
12
13         System.out.println("Hello, World");
14     }
15
16     public static void main(String[] args){
17
18         x=3;                //error
19         print();            //error
20         print_static();     //OK
21         new Test().x=5;     //OK
22         Test test=new Test(); //OK
23         test.print();       //OK
24     }
25 }
```

Objekti i promjenljive

- Promjenljive u kojima se čuvaju reference nazivaju se objektne promjenljive. Na primjer:

```
int x; // x je promjenljiva primitivnog tipa
Tacka A; // A je objektna promjenljiva (neinicijalizovana)
Tacka B = new Tacka(0,0); // B je objektna promjenljiva (inicijalizovana)
```



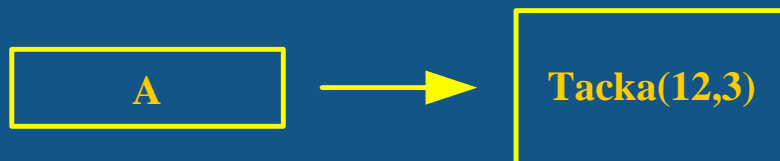
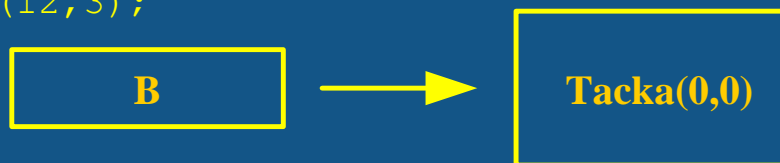
- Važno je razlikovati objekat od objektne promjenljive. Objekat se pravi pomoću operatora `new`, dok objektna promjenljiva samo sadrži referencu na objekat.

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

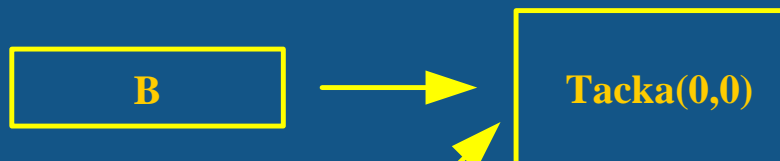
Objekti i promjenljive

- Ako objektu promjenljivu inicijalizujemo naknadno kao na primjer:

```
A = new Tacka(12,3);
```



- Ako napišemo: `A = B;`



- U Javi **dealokacija** objekata alociranih sa operatorom `new` nije zadatak programera već to radi okruženje.



Ključna riječ `this`

- Svaka nestatička metoda klase poziva se s instancom svoje klase.

```
Tacka A = new Tacka(3,3);
```

```
.....
```

```
double x = A.getX();
```

- Sintaksa poziva je: `objektnaPromjenljiva.imeMetode(...)`. Pri pozivu metoda pored explicitnih parametara dobija se i jedan implicitni parametar: to je objekat na kojem je metoda pozvana.
- U svakoj metodi ključna riječ `this` referencira na implicitni parametar.

```
1  class Tacka{
2
3      private double x; //x-koordinata tacke u ravni
4      private double y; //y-koordinata tacke u ravni
5
6      public Tacka(){
7          x=0;
8          y=0;
9      }
10
11     public Tacka(double x, double y){
12         this.x=x;
13         this.y=y;
14     }//kraj konstruktora
15
16     //javni interfejs nase klase:
17
18     public double getX(){ return x;}
19     public double getY(){ return y;}
20
21 }//kraj klase Tacka
```



Ključna riječ `this`

- Druga uloga ključne riječi `this` je poziv konstruktora. Prvi konstruktor u klasi `Tacka` možemo napisati na sledeći način:

```
public Tacka() {                public Tacka() {
    this(0,0);                    x=0.0;
}                                  y=0.0;
                                   }
}
```

- Ključna riječ `this` je ovdje poziv drugog konstruktora. `this` ima ovakvu ulogu samo kada se pojavljuje kao prva naredba u konstruktoru.
- Statičke metode ne dobijaju implicitni parametar.
- Zbog toga, ne mogu pristupiti nestatičkim promjenljivim članicama niti mogu pozivati nestatičke metode.



Definicija klase - rekapitulacija

- Format:

```
modifikator class ImeKlase
{
    //tijelo klase
}
```

ili

```
modifikator class ImeKlase extends PostojećaKlasa
{
    //tijelo klase
}
```




Definicija klase - rekapitulacija

- **modifikator** dodatno opisuje klasu (`public`, ...)
- **ImeKlase** je identifikator (po konvenciji, sve riječi počinju velikim slovom)
- **tijelo klase** sadrži definicije
 - polja (objektne i klasne promenljive)
 - metoda (objektni i klasni)

Definicija klase - rekapitulacija



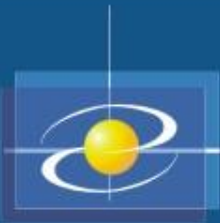
•Primjer

```
public class članPorodice
{
    static String prezime= "Marić"; // klasna promenljiva
    String ime; // objektna promenljiva
    int uzrast; // objektna promenljiva
    ...

    public članPorodice(String ime, int uzrast) { //
konstruktor
        this.ime = ime;
        this.uzrast = uzrast;
    }

    public static void verzija(...) { // klasni metod
    } ...

    public void prikaži(...) { // objektni metod
    } ...
    ...
}
```



Stvaranje objekata klase

- Operator **new**
- Format:

```
new konstruktor-klase
```

- Izvršavanje:
 - Stvara objekat rezervišući potreban prostor za njega u memoriji
 - Izvršava navedeni konstruktor klase
 - Vraća adresu rezervisanog memorijskog prostora za objekat
- Može se stvoriti nula, jedan ili više objekata definisane klase



Stvaranje objekata klase

- Primjer:

```
new ČlanPorodice("Mira",15);
```

```
ČlanPorodice sin; //referencna promenljiva
```

```
sin = new ČlanPorodice("Nikola",20);
```

```
ČlanPorodice otac; //referencna promenljiva
```

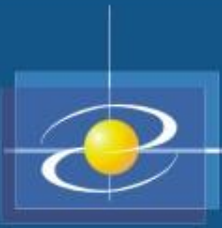
```
otac = new ČlanPorodice("Petar",52);
```

ili kraće

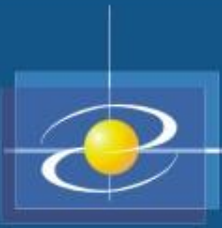
```
ČlanPorodice sin = new ČlanPorodice("Nikola",20);
```

```
ČlanPorodice otac = new ČlanPorodice("Petar",52);
```

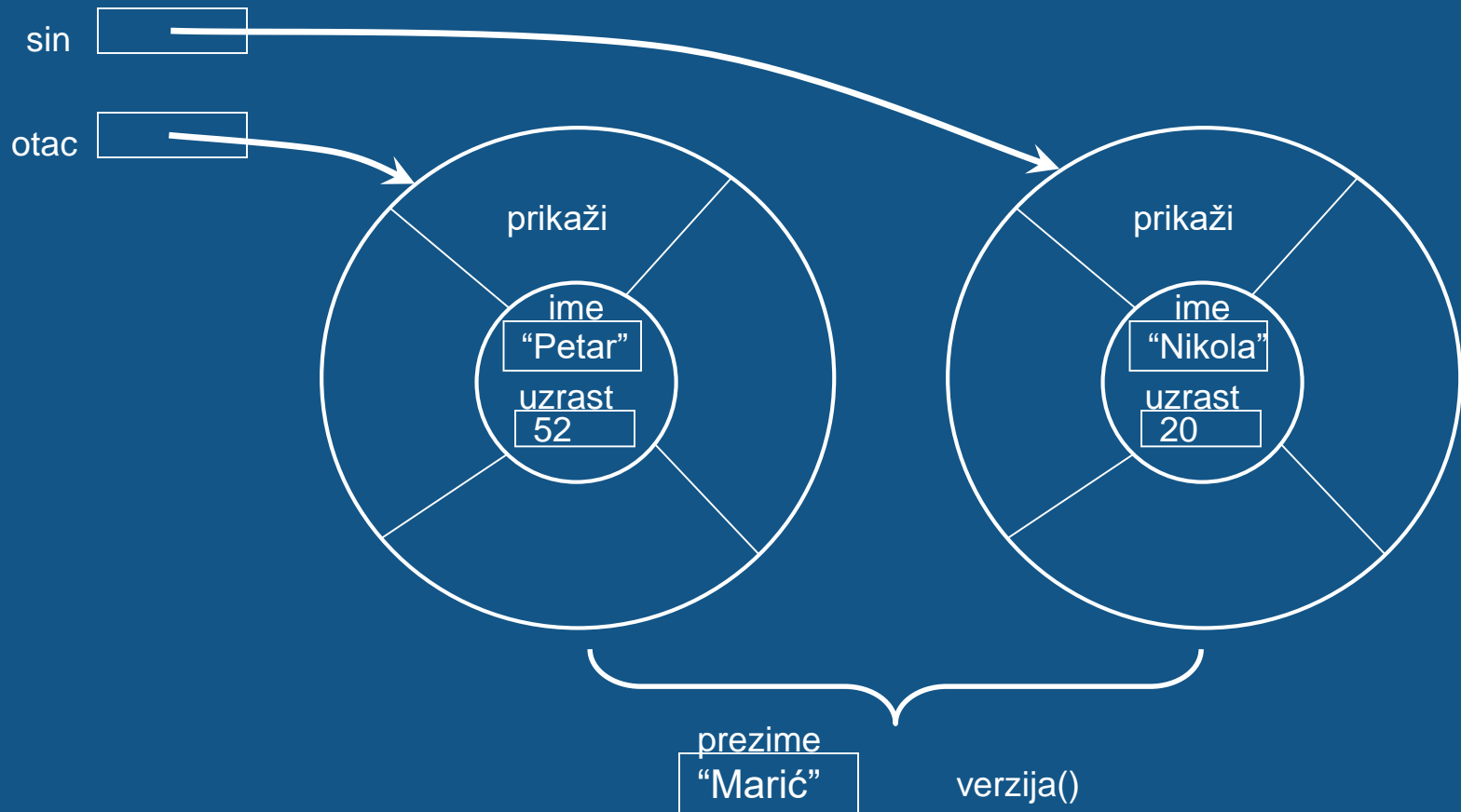
Stvaranje objekata klase

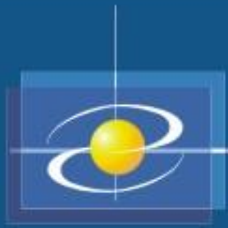


- Klasne (statičke) promjenljive i metodi su zajednički za sve objekte klase
- Objekti klase sadrže vlastite kopije objektnih promjenljivih i metoda
- Referencna promjenljiva sadrži pokazivač (referencu) na novostvoreni objekat (tj. sadrži njegovu adresu u memoriji)



Stvaranje objekata klase





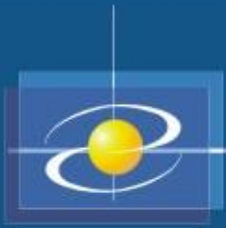
Korišćenje polja i metoda

- Objektna i klasna polja (kao i metodi) se koriste pomoću tačka-notacije
- Ispred imena polja se navodi
 - Ime klase za klasna polja
 - Ime objektne promenljive konkretnog objekta za objektna polja



Pozivanje metoda

- Objektni metodi
 - Odnose se na pojedinačne objekte i indirektno kao parametar imaju konkretni objekat klase u kojoj su objektni metodi definisani
 - Pozivaju se samo preko odgovarajućeg konkretnog objekta korišćenjem tačka-notacije
- Klasni (statički) metodi
 - Odnose se na cijelu klasu, a ne na pojedinačne objekte
 - Sadrže modifikator **static**
 - Pozivaju se preko imena klase korišćenjem tačka-notacije (može i kao objektni metodi, ali se ne preporučuje)



Podaci metoda

- U metodu se mogu koristiti četiri potencijalna izvora podataka:
 - Parametri metoda
 - Objektne i klasne promenljive klase u kojoj je metod definisan
 - Lokalne promenljive, definisane u telu metoda
 - Rezultati drugih metoda koji se pozivaju