Објектно орјентисано програмирање

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

Класе у програмском језику Јава

Владимир Филиповић
vladaf@matf.bg.ac.rs
Александар Картељ
kartelj@matf.bg.ac.rs

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Дефиниција класе у Јави

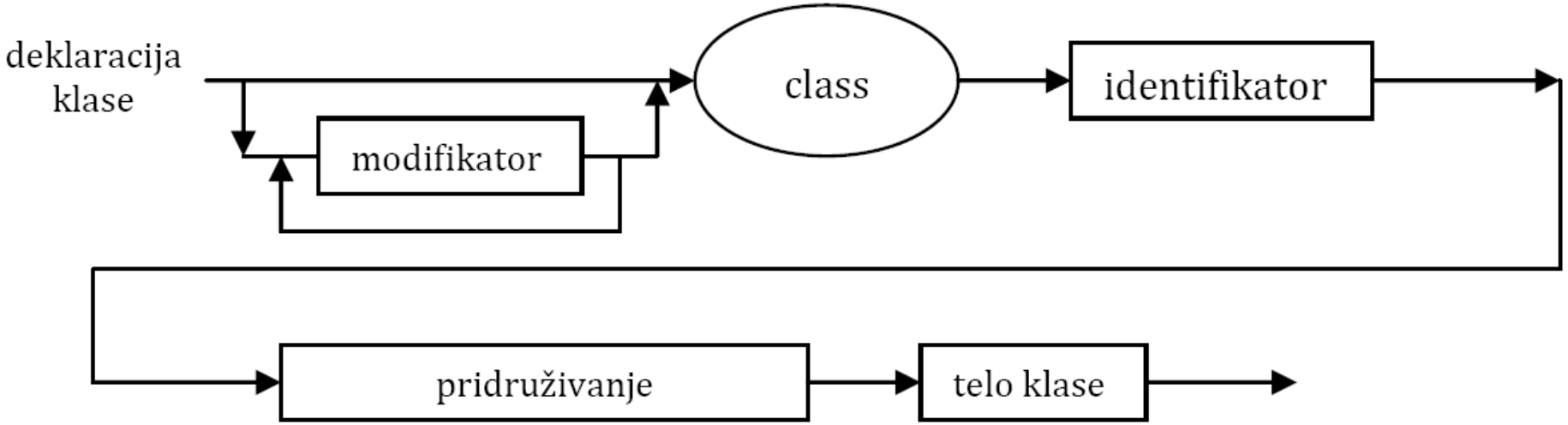
- Цео Јава код се налази унутар класа.
- Једној класи одговара једна датотека са екстензијом .java. Назив класе и назив датотеке треба да буду исти.
- Класа се дефинише коришћењем кључне речи class.
- На пример, наредбом:
  class Duzina је дефинисана нова класа са називом Duzina, а блок (тј. кôд између витичастих заграда), који следи, описује каква је структура новонаправљене класе.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Дефиниција класе у Јави (2)

- Синтаксни дијаграм за декларисање класа има следећи облик:

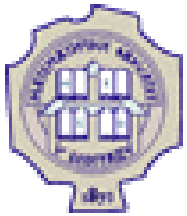


- Модификатор је резервисана реч која мења основно значење неке конструкције у Јави:

```
<modifikator>::= public | private | protected | abstract | final | static
                 | native | strictfp | synchronized | transient | volatile
```

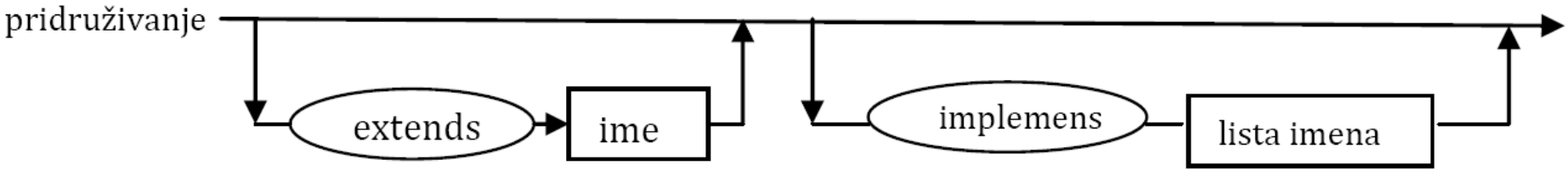
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

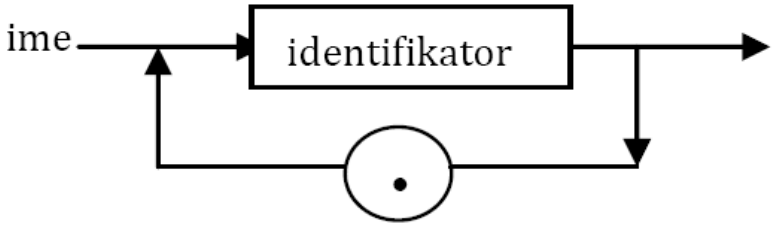


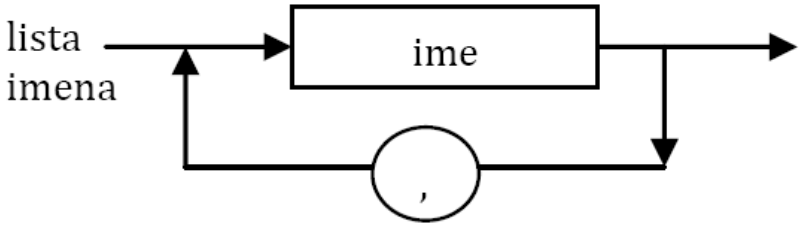
Дефиниција класе у Јави (3)

- Металингвистичка променљива pridruživanje одређује из које класе је дата класа изведена, односно, да ли дата класа имплементира неке интерфејсе:



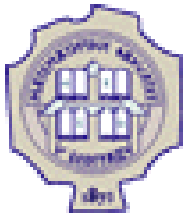
- Металингвистичка променљива ime може бити индентификатор или идентикатори раздвојени тачком, док је lista imena низ имена раздвојених зарезима:





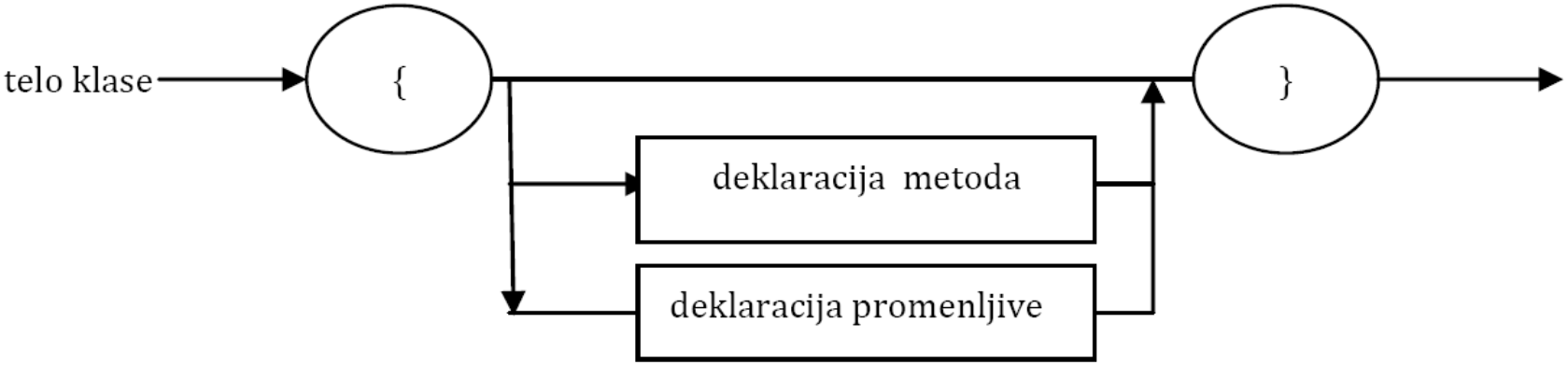
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



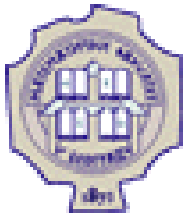
Дефиниција класе у Јави (4)

- Тело класе представља блок у којем се наводе разне декларације:



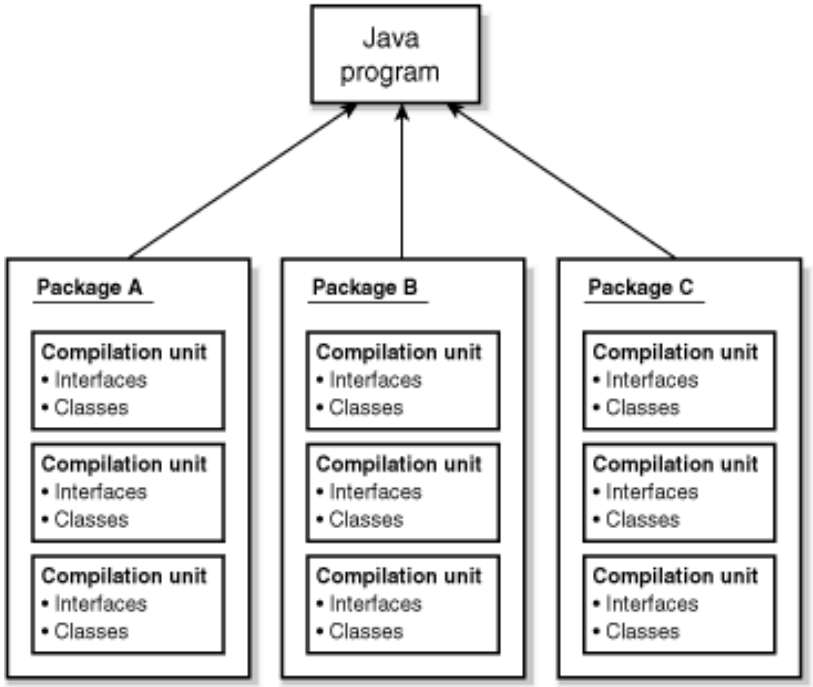
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



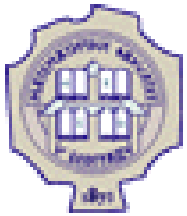
Пакети

- Програмери групишу сличне тј. повезане типове у пакете и на тај начин избегавају конфликте у именима и контролишу приступ.
- Пакет је група повезаних типова (класа, интерфејса, енумерисаних типова и типова нотације)за коју је обезбеђује заштита при приступу и управљање простором имена.



Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



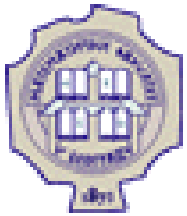
Пакети (2)

Разлози за паковање класа и интерфејса у пакете су :

1. Лакше одређивање да ли су типови повезани.
2. Лакше се могу пронаћи тражени типови.
3. Нема именских конфликта са другим типовима истог назива, јер пакет креира нови простор имена.
4. Допуштање да типови унутар пакета имају неограничен приступ један другом.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

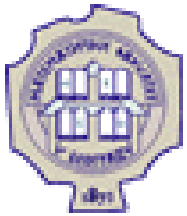


Пакети (3)

- Процес креирања сопствених пакета се може описати у три корака.
- **Први корак** је избор имена пакета.
- Препорука произвођача: коришћење назива Интернет домена са елементима поређаним по обрнутом редоследу.
  - На пример, ако је назив домена: matf.bg.ac.rs, назив пакета би требало да почне са rs.ac.bg.matf.
- На тај начин се постиже да назив пакета буде јединствен.
- По конвенцији, називи пакета почињу малим словима.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

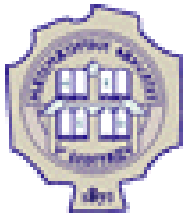


Пакети (4)

- **Други корак** је креирање структуре директоријума (фасцикли, фолдера).
- Ако је назив пакета из једног дела (нема тачака у називу), назив директоријума поклапа се са називом пакета.
- Ако се назив пакета састоји из више делова (одвојених тачком), тада за сваки део треба формирати поддиректоријум.
    - На пример, за rs.ac.bg.matf, главни директоријум треба да се зове rs, његов поддиректоријум ac, његов поддиректоријум bg и у њему треба да постоји директоријум matf.
- У сваки од ових директоријума се могу убацити датотеке, односно класе, интерфејси итд.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



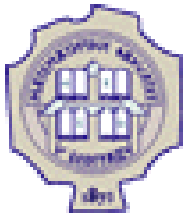
Пакети (5)

- **Трећи корак** је додавање package наредбе
- Ово треба да буде прва наредба Јава програма, дакле, пре прве наредбе import.
  - На пример, ако је назив пакета rs.ac.bg.matf, на почетку сваке датотеке у том пакету мора писати:

```
package rs.ac.bg.matf;
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

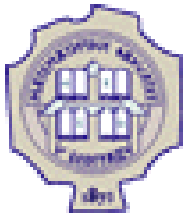


Пакети (6)

- Да би могло да се рукује уграђеним Јава класама, мора се знати где се класе налазе у оквиру система.
- Место где се класе налазе одређује се преко команде оперативног система CLASSPATH.
  - Овом командом се дефинише путања до директоријума у ком Јава окружење за извршавање тражи класе.
  - Ако CLASSPATH није дефинисан, подразумева се директоријум java\lib у конкретној инсталацији Јаве.
- Комбиновањем путање дате у CLASSPATH-у и назива пакета, Јава Виртуелна Машина проналази класе са којима се оперише.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Објекти

- За креирање примерака (конкретних објекта неке класе) користи се оператор **new**.

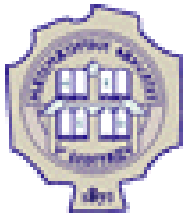
Примери:

```
String niska = new String();
Random slicajan = new Random();
Knjiga x = new Knjiga();
Point t = new Point(20 30);
```

- Описати аутоматско управљање меморијом за објекте.
- Описати улогу сакупљача отпадака.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Објекти (2)

- Компоненте објекта су: инстанцне промењљиве и методи.
- Као што је већ истакнуто, компонентама објекта приступа се преко тзв. тачка-нотације.

Пример.

(а) Приступ промењљивим примерка:

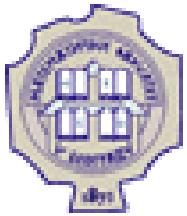
```
prviObjekt.prom; prviObjekt.prom.stanje;
prviObjekat.prom.stanje = true;
prviObjekat.stampajMe();
```

(б) Приступ методима:

```
prviObjekat.uzmiIme();
prvi objekat.postaviVelicinu(20);
prviObjekat.prom.metod3(arg1, arg2, arg3); // pozvan preko promen.
prviObjekat.getClass().getName(); // metod iz metoda
```

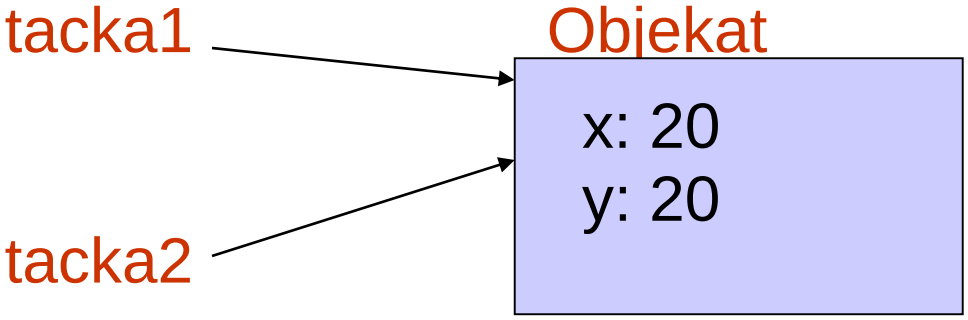
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



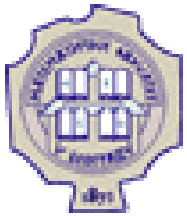
Објекти (3)

- Када се креира објекат, променљива којој се додељује представља показивач (референцу) на тај објекат.
- При позиву метода, у случају аргумената који представљају објекте, прослеђује се вредност референце на објекат, а не сам објекат!
- Обрнуто важи за аргументе метода примитивног типа, ту променљива садржи податак и дупликат текуће вредности променљиве тј. податка се прослеђује методу.
- Дакле, у оба случаја се врши супституција вредности, само што је код објектних аргумената у питању вредност референце.



Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Објекти (4)

Припадност објекта класи

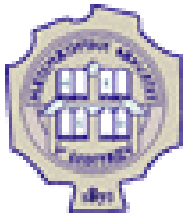
Реализује се помоћу оператора instanceof

Пример.

```
"bilo sta" instanceof String // true
Point tacka = new Point(10,10);
tacka instanceof String // false
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Конверзија типова

- Када се говори о конверзији типова, може се говорити о:

(а) конвертовању примитивних типова у примитивне

(imetipa) vrednost

Пример.

```
(int)(x/y);
```

(б) конвертовању објеката у објекте

Има смисла само код класа/интерфејса повезаних наслеђивањем, те класа и интерфејса повезаних имплементацијом.

(imeklase) objekat

Како је сваки примерак неке класе истовремено и примерак надкласе, нема потребе за експлицитном конверзијом у надкласу

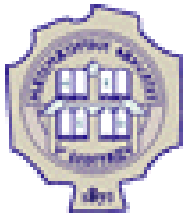
Пример. Нека је класа Kamion поткласа класе Vozilo

```
Vozilo x;
Kamion y;
y = new Kamion();
X = y; // Eksplicitna konverzija nepotrebna
```

```
Vozilo x = new Kamion();
Kamion y;
y = (Kamion) x;
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Конверзија типова (2)

(в) конвертовању примитивних типова у објекте и обрнуто.

У принципу, није могуће, осим код тзв. класа-омотача примитивних типова.

У пакету java.lang постоје класе: Integer, Float, Boolean, …

Пример.

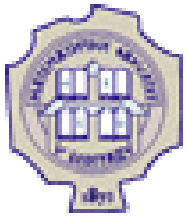
```
Integer ceoObj= new Integer(45);
int ceo = ceoObj.intValue();
Integer ceoObj2 = new Integer( 2*ceo );
```

Почев од верзије 6, програмски језика Јава подржава и аутоматску конверзију између података примитивног типа и објеката у одговарајућим класама-омотачима.

Напомена. Експлицитна конверзија крије многе опасности!

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – поља

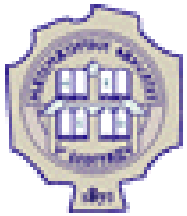
Постоји неколико врста **променљивих**:

1. поља - променљиве декларисане у самој класи
   - представљају чланове-податке унутар класе, тј. описују атрибуте објекта који је примерак дате класе;
2. локалне променљиве - променљиве декларисане у телу метода или блоку;
3. формални аргументи - променљиве декларисане у заглављу метода.

- У даљем разматрању се концентришемо на променљиве чланове, тј. поља.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

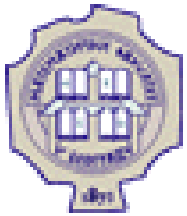


Класе у Јави – поља (2)

- Декларација поља се састоји од три компоненте:
  1. модификатора (који се опционо појављују),
  2. типа поља (тип)
  3. и имена поља (идентификатор).
- Модификатори омогућују:
  - да се подеси видљивост дате променљиве члана (тј. поља),
  - да се одреди да ли се ради о променљивој примерка (инстанцној променљивој) или о класној (статичкој) променљивој,
  - као и да ли вредност променљиве постаје непроменљива непосредно по креирању (финална тј. константна).
- У том смислу, разликују се:
  - променљива примерка,
  - класна променљива
  - и константа.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



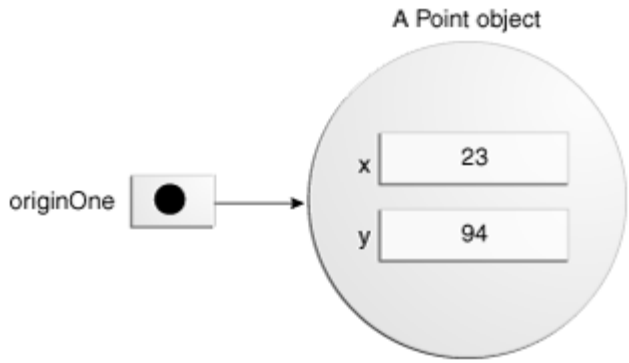
Класе у Јави – поља (3)

Променљиве примерка

- Сваки од креираних објеката дате класе садржи сопствени примерак те променљиве.
- Променљивој примерка се може приступити само ако се референцира примерак класе који садржи ту променљиву.
- Промена вредности једне променљиве примерка нема утицаја на остале.

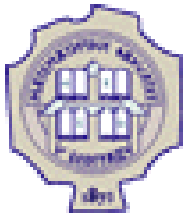
Пример.

```
class Point extends GeometryObject {

        int x; int y;
        ……..
}
```



Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – поља (4)

- Као што је раније истакнуто, компонентама објекта приступа се преко тзв. тачка-нотације
- То се односи и на инстанцне променљиве и на инстанцне методе

Пример.

```
prviObjekt.prom;
prviObjekt.prom.stanje;
prviObjekat.prom.stanje = true;
```

Пример.

```
tackaA.x = 23;
int xKoord = tackaA.x;
tackaA.y = 94;
System.out.println(tackaA.x + ",” + tackaA.y);
```

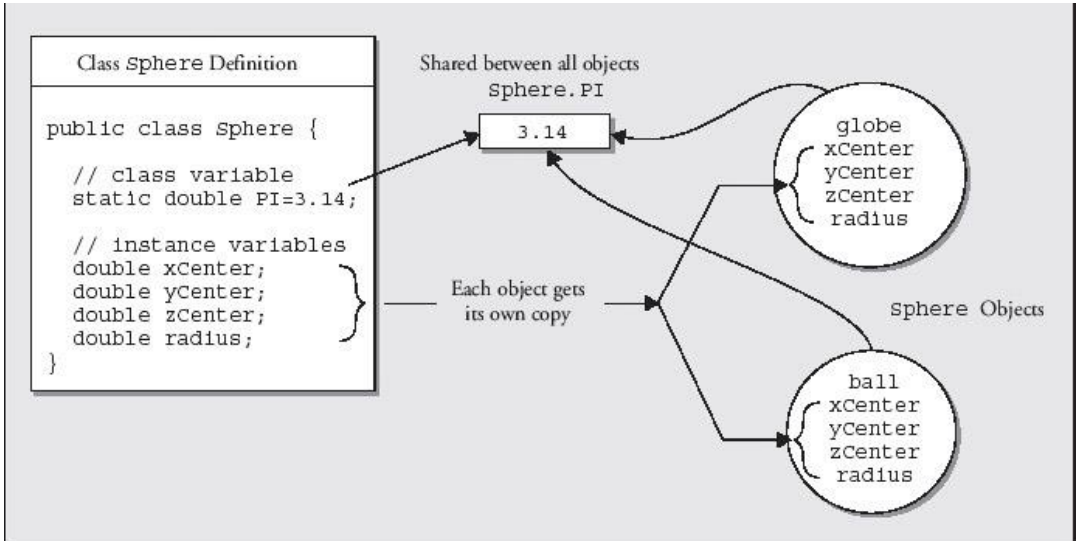
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави – поља (5)

Класне променљиве

- Класа садржи само једну копију класне променљиве и та променљива је дељена међу свим објектима дате класе.
- Она постоји чак иако се не креира ниједан примерак дате класе.
- Она припада класи и њу могу сви да референцирају, а не само примерци дате класе.
- Класна променљива се декларише коришћењем модификатора static.



Математички факултет

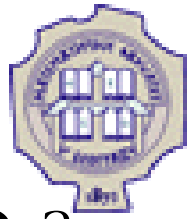
{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави – поља (6)

```
class ClanPorodice {
        static String prezime="Jankovic"; // klasna promenljiva
        int uzrast; // instancna promenljiva.
        String ime; // instancna promenljiva
        ……
        ClanPorodice otac = new ClanPorodice("Nikola", 52);
        ClanPorodice sin = new ClanPorodice("Petar", 21);
        ……
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – поља (7)

- За приступ класној променљивој се користи тачка-нотација, при чему се као прималац поруке може користити: име класе или име неке инстанце класе.
- Препоручује се коришћење имена класе.

Пример.

```
class ClanPorodice {
        static String prezime = "Jankovic";
        String ime;
        int godine;
        ……
}
```

и креиран примерак sin:

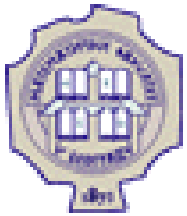
```
ClanPorodice sin = new ClanPorodice();
```

Класној променљивој prezime можемо приступити на следеће начине:

```
System.out.println( "Porodicno prezime je: " + sin.prezime);
System.out.println( "Porodicno prezime je: " + ClanPorodice.prezime);
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – поља (8)

Константе (финалне променљиве)

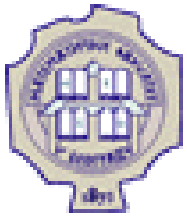
- Модификатор final дефинише да када једном променљива добије вредност није допуштена даља промена њене вредности.
- Може се примењивати и код класних и код инстанцних променљивих као и код локалних променљивих.

Пример.

```
final float STOPA = 2.3f;
final boolean NETACNO = false;
final int BROJ_STRANA = 200;
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – this

- У оквиру метода примерка или конструктора, променљива this представља референцу на сам тај објекат.
- Коришћењем this се може референцирати на ма које поље текућег објекта над којим се позива метод примерка или конструктор.

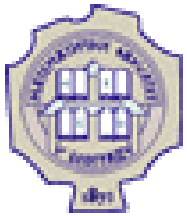
Пример.

```
{
    this.x =0;
    this.y=0;
}
```

У претходном примеру се кључна реч this може изоставити.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – this (2)

- Најчешћи разлог за коришћење ове променљиве је то што поље класе буде сакривено параметром метода или параметром конструктора.

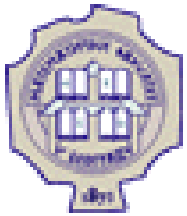
Пример.

```
public void pomeri(int x, int y) {
        this.x =x;
        this.y=y;
}
```

- Овде је кључна реч this неопходна да подвуче разлику између инстанцних променљивих и параметара метода.
- Сваки од аргумента метода сакрива по једно поље објекта, тако да је унутар метода x ознака локалне копије првог аргумента.
- Да би се реферисало на поље x примерка класе Point, унутар метода се мора користити this.x.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – this (3)

Примери:

У наредбама које следе се користи кључна реч this:

```
t = this.a; // a -instancna promenljiva tekućeg objekta
this.prviMetod(this); // poziva se metod
return this; //vraća se tekući objekt iz metoda
```

У неким од наведених случајева може се изоставити:

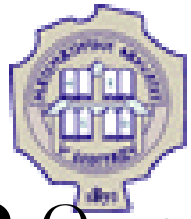
```
t = a; prviMetod(this);
```

У некима не може:

```
return this;
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Опсег важења променљиве

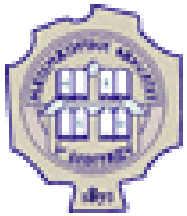
- Опсег важења променљиве је део програма у којем име променљиве може да се користи.
- Код локалних променљивих, опсег важења променљиве је блок у којем је дефинисана од места где је дефинисана.

Пример.

```
{
        int a=1; //ovde se može pozivati a, ali ne i b!
        { //ovde se može pozivati a, ali ne jos b!
                int b=2; // ovde se mogu pozivati a i b!
        } // ovde se može pozivati a, ali ne i b!
}
// ovde se ne mogu koristiti niti a, niti b!
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Опсег важења променљиве (2)

- Класним променљивима се може приступати у сваком тренутку рада програма.
- Променљивима примерка се може приступати сво време док постоји објекат – примерак дате класе.

Пример.

```
class Opseg {
        int n, probna = 10;

        void stampaProbnu() {
                int probna = 30;
                System.out.println("Lokalna probna = "+probna);
        }
        n=probna;
}
```

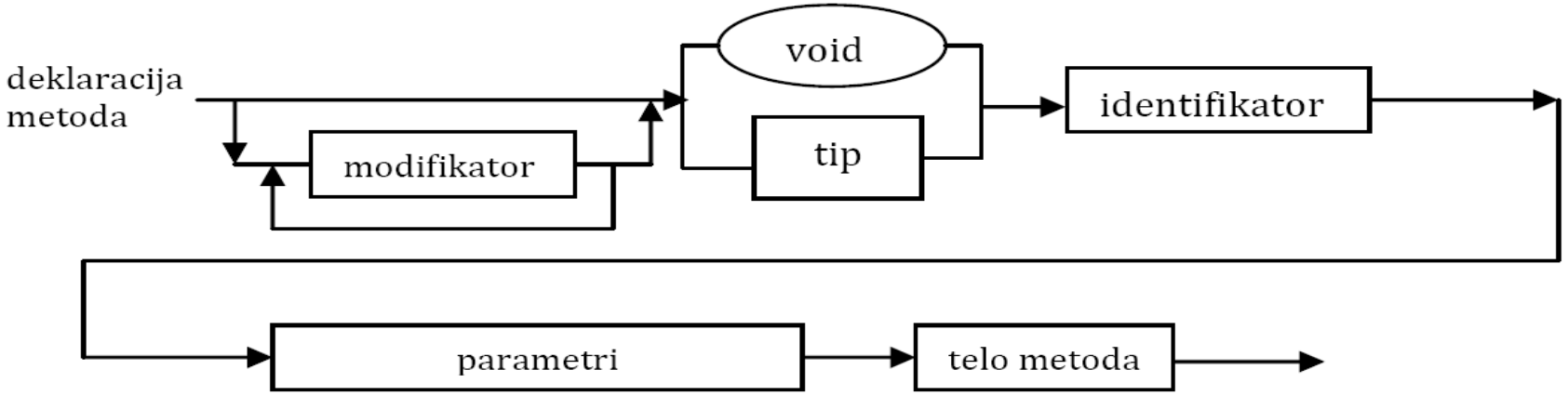
- Ако треба да се унутар метода приступи сакривеном пољу probna, то се постиже помоћу this.probna.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави - методи

- Методи се појављују у телу класе.
- Они садрже декларације локалних променљивих и друге Јава наредбе које се извршавају при позиву метода.
- Аргументи метода се наводе у заградама иза имена метода (број аргумената може бити нула).
- Приликом декларације метода у класи, наводи се тип аргумената.
- При позиву метода наводе се стварни аргументи.

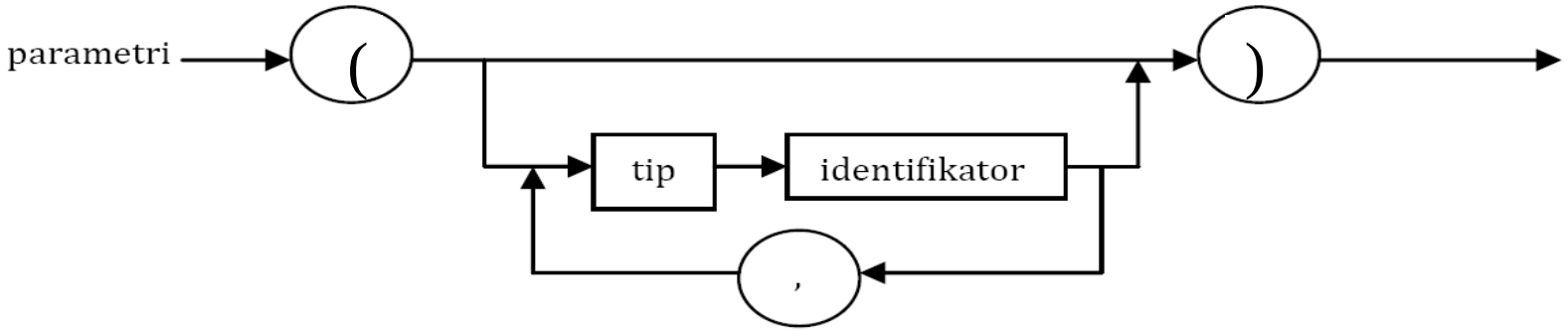


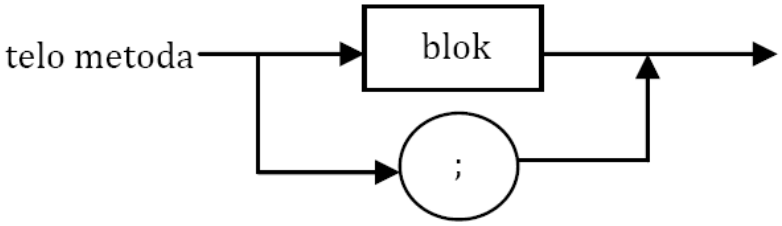
Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави – методи (2)

- Метод је самостални блок кода који има име и својство вишеструке употребљивости.
- Параметри омогућавају да се унесу вредности у метод:





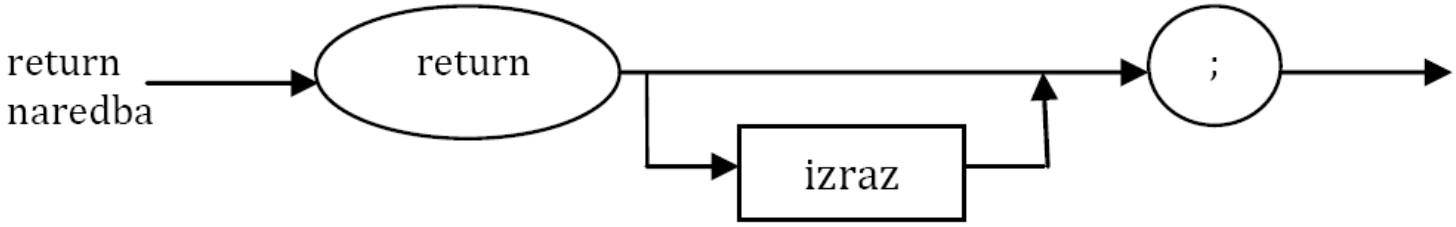
- Металингвистичка променљива blok је дефинисана у једној од претходних презентација.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави – методи (3)

- Метод може вратити вредност
  и онда је потребно навести тип повратне вреености.
- Уколико метод не враћа вредност, његов тип је void.
- Повратна вредност из метода се враћа преко return наредбе:



- Синтакса израза је дефинисана на слајдовима претходне презентације.

Математички факултет

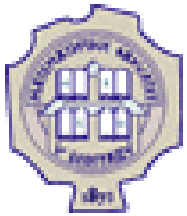
{vladaf, kartelj}@matf.bg.ac.rs

Класе у Јави – методи (4)

- Методи могу бити:
  1. инстанцни (метод примерка)
  2. и класни (статички).
- Класни методи се не односе на инстанцне променљиве, тј. не мора постојати ни једна инстанца, а метод се може користити!
- У телу класног метода се не може референсирати на променљиве примерка дате класе, већ само на класне променљиве за дату класу.
- Метод main(String args[]), који је неопходан у апликацијама, је увек класни, јер пре његовог стартовања не постоји ни једна инстанца.
- Не прави се копија класног метода за сваку инстанцу, већ се он једанпут дефинише.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Класе у Јави – методи (5)

- Име метода заједно са типом и редоследом параметара чини потпис метода.
- Потпис два метода у класи мора бити различит, како би компајлер могао да одреди који метод се позива.

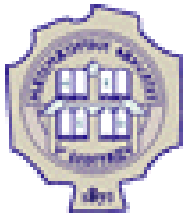
```
povratni-tip imeMetoda(arg1, arg2,…,argn) {
        // Kod metoda
}
```

Potpis metoda

- Приликом позива метода, на место формалних аргумената (параметара), наводе се стварни аргументи.
- Код навођења формалних аргумената, за сваки аргумент мора се навести тип аргумента.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



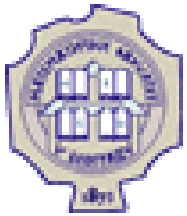
Класе у Јави – методи (6)

У наредбама у телу метода се могу користити четири потенцијална извора података:

1. формални аргументи метода,
2. инстанцне и класне променљиве,
3. локалне променљиве, дефинисане у телу метода и
4. вредности које врађу други методи који су позвани у текућем.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Преоптерећење метода

- Дефинисање метода са истим именом, али различитим параметрима назива се преоптерећење метода.

Пример.

```
class Pravougaonik {
        int x1=0;
        int x2 = 0
        int y1 = 0;
        int y2 = 0;

        Pravougaonik gradi(int x1, int y1, int x2, int y2) {
                this.x1 = x1;
                this.y1 = y1;
                this.x2 = x2;
                this.y2 = y2;
                return this;
        }
}
```

Математички факултет

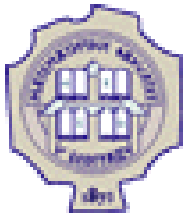
{vladaf, kartelj}@matf.bg.ac.rs

Преоптерећење метода (2)

```
Pravougaonik gradi(Point goreLevo, Point doleDesno) {
        x1 = goreLevo.x;
        y1= goreLevo.y;
        x2 = doleDesno.x;
        y2 = doleDesno.y;
        return this;
}

Pravougaonik gradi(Point goreLevo, int l, int h){
        x1 = goreLevo.x;
        y1 = goreLevo.y;
        x2 = goreLevo.x + l;
        y2 = goreLevo.y + h;
        return this;
}

void printPrav(){
        System.out.println("Pravougaonik :[" (+x1+","+y1+ "),("+x2+","+y2+")]");
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

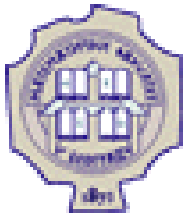


Преоптерећење метода (3)

```
public static void main (String args[]){
        Pravougaonik p = new Pravougaonik();
        p.gradi(10,20, 30,40);
        p.printPrav();
        p.gradi(new Point(10,10), new Point(30,30));
        p.printPrav();
        p.gradi (new Point (10,10), 20, 30);
        p.printPrav();
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



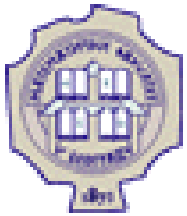
Иницијализациони блокови

- Иницијализациони блок је блок који се налази у оквиру дефиниције класе. Разликују се:
  1. Иницјализациони блок примерка (инстанцни инцијализациони блок)
  2. Иницијализациони блок класе (статички инцијализациони блок)

```
static int staticVariable;
int nonStaticVariable;
// Staticki inic. blok se izvrsava jednom po klasi.
static {
        System.out.println("Static initalization.");
        staticVariable = 5;
}
// Instanci se poziva pre konstruktora
{
        System.out.println("Instance initialization.");
        nonStaticVariable = 7;
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Конструктор

- Приликом креирања конкретног примерка неке класе (преко оператора new), увек се позива конструктор те класе.
- Конструктор је метод класе који се позива приликом креирања новог примерка дате класе.
- Карактеристике конструктора:
  - никад не враћа вредност
  - његово име се поклапа са именом класе.
- Као и код осталих типова метода, типови и бројеви формалних параметара при декларацији и позиву се морају поклапати.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Конструктор (2)

- Уколико програмер није дефинисао конструктор за дату класу, преводилац позива подразумевани конструктор.
- Подразумевани конструктор:
  - нема аргумената;
  - иницијализује све инстанцне и класне променљиве на подразумеване вредности.
- Ако је програмер дефинисао бар један конструктор за дату класу, онда подразумевани конструктор више не постоји.

Математички факултет

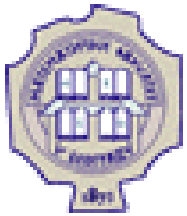
{vladaf, kartelj}@matf.bg.ac.rs

Конструктор (3)

```
class Knjiga {
        String autor;
        String naslov;
        int brojStrana;

        Knjiga(String a, String n, int bs) {
                autor = a;
                naslov = n;
                brojStrana = bs;
        }

        void stampaPod() {
                System.out.println("Autor: "+autor);
                System.out.println("Naslov: "+naslov);
                System.out.println("Bro strana: "+brojStrana);
        }

        public static void main(String args[]) {
                Knjiga jedina;
                jedina = new Knjiga("Ivo Andric", "Gospodjica", 257);
                jedina.stampPod();
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



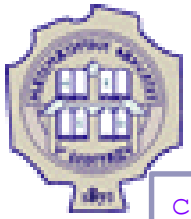
Преоптерећење конструктора

- Конструктори могу бити преоптерећени, исто као и остали методи.
- Ако постоји додатни конструктор, који има неке нове особине, у њему се може позвати већ постојећи конструктор употребом кључне речи this на следећи начин:

```
this(arg1, arg2, …);
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

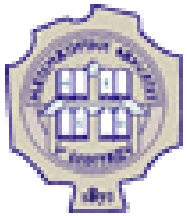


Преоптерећење конструктора (2)

```
class Knjiga2 {
        String autor;
        String naslov;
        int brojStrana;

        Knjiga2(String a, String n, int bs) {
                autor = a;
                naslov = n;
                brojStrana = bs;
        }

        Knjiga2(String a, String n) {
                this( a, n, 200);
        }

        Knjiga2(String a) {
                this( a, ”Gorski vijenac”, 140);
        }

        Knjiga2() {
                this(”Njegos”, “Gorski vijenac”, 140);
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

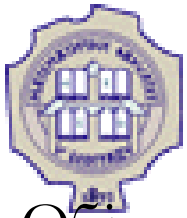


Преоптерећење конструктора (3)

```
void stampaPod() {
        System.out.println("Autor: "+autor);
        System.out.println("Naslov: "+naslov);
        System.out.println("Bro strana: "+brojStrana);
}

public static void main(String args[]) {
        Knjiga2 prva;
        prva = new Knjiga2("Ivo Andric", "Gospodjica", 257);
        prva.stampPod();
        Knjiga2 druga = new Knjiga2();
        druga.stampPod();
        prva = new Knjiga2("Dzordz Orvel");
        prva.stampPod();
        druga = new Knjiga2("Borislav Pekic", "Atlantida");
        druga.stampPod();
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

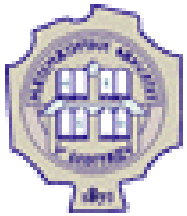


Копирајући конструктор

- Објекат чије унутрашње стање (тј. вредност неког поља) може да се промени назива се мутирајући објекат.
- У супротном се ради о немутирајућем објекту.
- Стога треба пажљиво радити са конструкторима мутирајућих објеката.
- Наиме, може се догодити да се, при извршавању конструктора, вредност поља новокреираног објекта постави тако да садржи вредност аргумента конструктора:
  - У том случају стварни аргумент конструктора и поље новокреираног објекта постају „везани“ и реферишу на исти објекат.
  - Због тога промена објекта аргумента конструктора доводи до промене поља новокреираног објекта и обрнуто.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

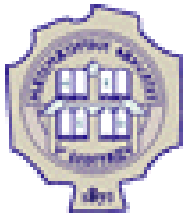


Копирајући конструктор (2)

```
class Line extends GeometryObject {
        Point a;
        Point b;

        Line( Point a, Point b ) {
                this.a = a;
                this.b = b;
        }

        public static void main( String[] argsKL ) {
                Point tackaA = new Point( 19, 20 );
                Point tackaB = new Point( 7, 52 );
                Line linijaAB = new Line( tackaA, tackaB );
                tackaA.setX(200);
        // ovde je doslo do promene vrednosti linijaAB
        // sa [(19,20)-(7, 52)] na [(200,20)-(7, 52)]
        // iako nije radjeno sa linijaAB, vec sa tackaA
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Копирајући конструктор (3)

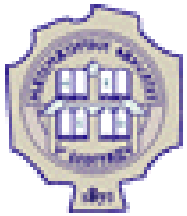
- Да би се избегле такве ситуације, потребно је уместо употребе референце на исти објекат направити копију објекта.

Пример.

```
class Line extends GeometryObject {
        Point a;
        Point b;

        Line( Point a, Point b ) {
                this.a = new Point( a.x, a.y );
                this.b = new Point( b.x, b.y );
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

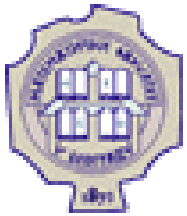


Копирајући конструктор (4)

- Сада више нема „везивања“ поља и аргумента.
- Међутим, запис је рогобатан:
  - нарочито у случају када мутирајући објекат који представља поље новог објекта има сложену структуру.
- У прошлом случају било би јако добро када би постојала могућност да се једноставном наредбом направи нова копија.
- Да би се то постигло, потребно је да у класи која описује тип поља постоји тзв. копирајући конструктор.
- Да би се истакло да копирајући конструктор не може да модификује своје аргументе користи се модификатор final.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

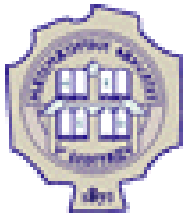


Копирајући конструктор (5)

```
class Point extends GeometryObject {
        int x; int y;

        Point( int x, int y ) {
                this.x = x;
                this.y = y;
        }

        Point( final Point tacka ) {
                this.x = tacka.x;
                this.y = tacka.y;
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

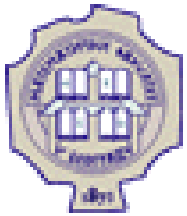


Копирајући конструктор (6)

```
class Line extends GeometryObject {
        Point a;
        Point b;

        Line( Point a, Point b ) {
                this.a = new Point( a );
                this.b = new Point( b );
        }

        public static void main( String[] argsKL ) {
                Point tackaA = new Point( 19, 20 );
                Point tackaB = new Point( 7, 52 );
                Line linijaAB = new Line( tackaA, tackaB );
                tackaA.setX(200);
                // promena kod tackaA ne menja vrednost linijaAB
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

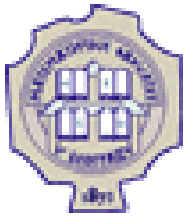


Превазилажење метода

- Процес дефинисања метода у потклаcи који има исти потпис као и метод у наткласи назива се превазилажење метода.
- Способност подкласе да превазиђе метод омогућује да нека класа:
  1. наследи „довољно блиску“ надкласу
  2. и да по потреби модификује њено понашање.
- Нови метод има исто име, исти број и типове параметара и враће резултат истог типа као метод надкласе који се превазилази.
- Нови метод може да врати и подтип типа који враћа метод надкласе, што се назива коваријантни тип резултата.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



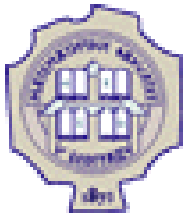
Превазилажење метода (2)

- Када се превазилази метод пожељно је користи анотацију @Override.
- На тај начин се даје компајлеру да желимо да урадимо превазилажење и он нас упозорава уколико метод са таквим потписом не постоји у надкласи.

```
public class Line {
        void drawLine(){
                //implementacija crtanja linije u nadklasi
        }
}

public class DottedLine extends Line{

        @Override
        void drawLine() {
                //prevazilazimo implementaciju iz nadklase
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Превазилажење метода (3)

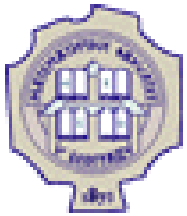
- Позивање одговарајућег метода дефинисаног у натклаcи дате класе, а који је превазиђен у потклаcи, реализује се коришћењем кључне речи super.

Пример.

```
void mojMetod( int x, int y) {
        // neke naredbe
        super.mojMetod(x,y);
        // jos naredbi
}

void mojMetod2( int x, int y) {
        // neke naredbe
        mojMetod(x,y);
        // poziva se metod definisan u istoj ovoj klasi
        super.mojMetod(x,y);
        // poziva se metod definisan u nadklasi
        // jos naredbi
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



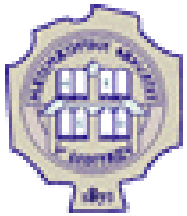
Превазилажење метода (4)

- Превазилажење конструктора се не може извршити, јер конструктор има исто име као класа у којој се налази.
- Приликом конструисања примерка поткласе, тј. извршења конструктора поткласе бива позван конструктор наткласе.
- Ако у наткласи није дефинисан конструктор, тада бива позван подразумевани имплицитни конструктор наткласе.
- Експлицитно позивање конструктора наткласе се може реализовати кључном речју super као у случају позива метода.

```
super(arg1, arg2, …)
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



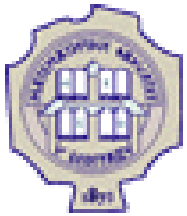
Превазилажење метода (5)

Пример.

```
class LabeledPoint extends Point {
        String label;
        LabeledPoint (int x, int y, String label) {
                super(x,y);
                this.label = label;
        }
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



Метод финализатор

Метод финализатор је дефинисан у оквиру класе Object, па свака класа у Јави може да га превазиђе.

- Овај метод се може бити позван при сакупљању отпадака.
- Имплементација овог метода у класи Object не ради ништа.

Програмер може одлучити да у методу класе који га превазилази извршава нпр. ослобађање ресурса са којима не управља Јава виртуелна машина.

- Метод се позива са finalize().
- Може се превазићи у сопственој класи са:

```
protected void finalize() {
        ……
}
```

- Обично није неопходно његово коришћење.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Реализација класе Object

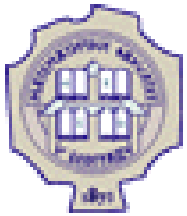
- Могуће је примењивати два оператора за поређење објеката:

  = = !=

- У овом случају, објекти се сматрају једнаким ако заузимају исти простор у меморији.
- Не постоји могућност преоптерећења оператора у програмском језику Јава, па самим тим ни оператора = = и !=.
- Други начин не употребом метода equals који је дефинисан у оквиру класе Object, па свака класа у Јави може да га превазиђе.
- Имплементација овог метода у класи Object проверава да ли су објекти идентички једнаки.
- Програмер може одлучити да у својој класи превазиђе овај метод нпр. тако да враћа true нпр. када су садржаји свих поља исти.

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



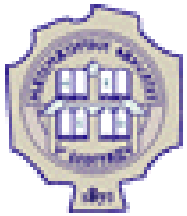
Реализација класе Object (2)

- Метод за превођење у ниску је дефинисан у оквиру класе Object, па свака класа у Јави може да га превазиђе.
- Метод toString() у класи Object враће ниска-репрезентацију датог објекта.
- Ниска-репрезентација сваког од објеката потпуно зависи од структуре тог објекта и то је разлог због код се метод toString() обично превазилази у новонаправљеним класама.
- Дакле, препоручује се да свака од класа има своју реализацију метода за превођење у стринг.

```
@Override
public String toString() {
        ….
}
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs



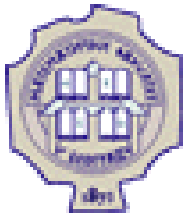
Реализација класе Object (3)

- Одређивање класе објекта се може се постићи коришћењем метода getClass(), који је дефинисан у класи Object и тај метод се не може се превазићи
- Овај метод враће имутабилни објекат типа Class, који садржи методе (више од 50) помоћу којих се могу добити информације о датој класи (надкласама, интерфејсима, пољима, методама, њиховим параметрима итд.) и/или вршити све операције над примерцима класе (креирање, повдешавање вредности поља, позивање метода и сл.)
- О томе ће бити више речи у делу који се односи на рефлексију.

Пример.

```
String runtimeClassName = objekat.getClass().getName();
```

Математички факултет
{vladaf, kartelj}@matf.bg.ac.rs



Реализација класе String

- Тип String је објектни тип који се користи за представљање текста, који се чува као низ знакова.
- Примерци класе String не могу да мутирају, тј. да мењају вредност.
- Не може се правити поткласа класе String.
- Када се извршава операција над постојећим стринг објектом, као резултат се увек креира нови примерак класе String.
- Као код других објеката, може се користити литерал null за одбацивање објекта на који тренутно реферише дата String променљива.

Пример. Постављање променљиве тако да не указује ни на шта.

```
String nulaString = null;
/* String promenljiva koja ne referiše ni na jedan string */
```

Математички факултет

{vladaf, kartelj}@matf.bg.ac.rs

Захвалница

Велики део материјала који је укључен у ову презентацију је преузет из презентације коју је раније (у време када је он држао курс Објектно орјентисано програмирање) направио проф. др Душан Тошић.

Хвала проф. Тошићу што се сагласио са укључивањем тог материјала у садашњу презентацији, као и на помоћи коју ми је пружио током конципцирања и реализације курса.