



Академија струковних  
студија Шумадија  
одсек у Крагујевцу

# Увод у програмирање

## Презентација 10

---

Академија струковних студија Шумадија

Одсек у Крагујевцу

Студијски програм Информатика

Крагујевац, 2020. година



# Објектно оријентисано програмирање

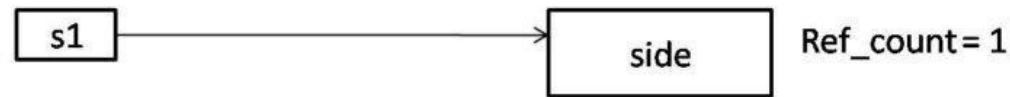
- На претходним предавањима научили смо да се креирање објекта изводи помоћу оператора **new**.
- Овим поступком се врши алоцирање меморијског простора за креирани објекат али смо такође напоменули да је количина слободне меморије коначна.
- У неком тренутку извршавања програма може да се догоди да нема више слободне меморије за коришћење (тада се "испаљује" изузетак).
- На неки начин програм мора да се ослобађа објеката који се више не користе (чиме се ослобађа и меморијски простор).
- У неким програмским језицима, нпр. C++, динамички створени објекти се морају "ручно" избрисати помоћу одговарајуће наредбе (*delete*).
- У програмском језику Јава користи се другачији приступ, тј. ослобађање непотребних објеката, а тиме и меморије, се врши аутоматски.



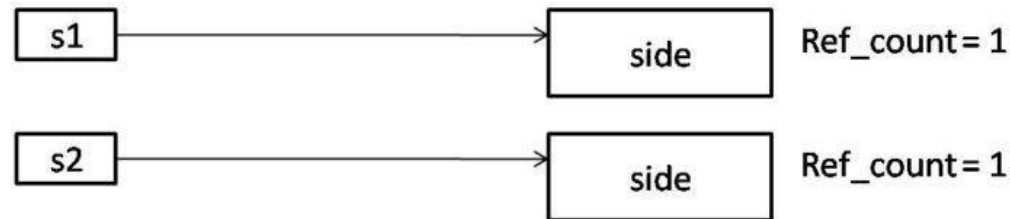
# Објектно оријентисано програмирање

- Основни критеријум на основу којег се у Јави препознаје да објекат није више потребан је да не постоји више ниједна променљива која указује на њега (не постоји ниједна референца).

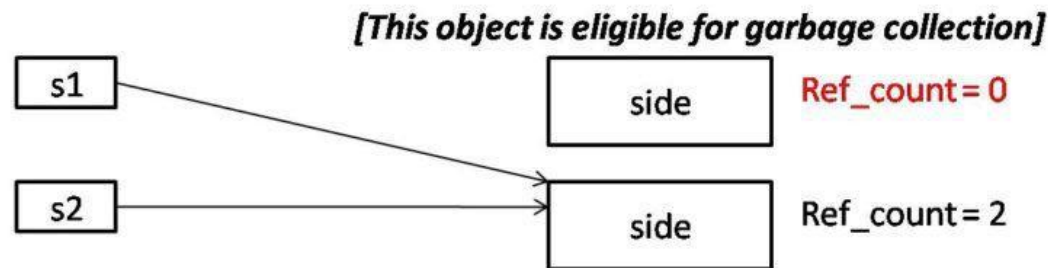
**Square s1 = new Square();**



**Square s2 = new Square();**



**s1 = s2;**





# Објектно оријентисано програмирање

- У Јави се користи посебна процедура сакупљања отпадака (енгл. ***garbage collection***) којом се аутоматски с времена на време "чисти ђубре", односно ослобађа меморија оних објеката за које се нађе да је број референци на њих у програму једнак нули.
- Дакле, није потребно писати изричите наредбе којима се покреће процедура сакупљања отпадака.
- Сакупљање отпадака се врши у посебној нити, с времена на време, ако до покретање те процедуре уопште и дође током извршавања вашег програма (зависи од сложености и времена извршавања програма).
- Различити Јава извршни системи примењују различите приступе сакупљања отпадака, о чему ви не треба да размишљате.
- Ипак, могуће је на одређеном наредбом позвати статички метод **gc()** и затражити од JVM да покрене процедуру сакупљања отпадака.



# Објектно оријентисано програмирање

- **Пример:** Потребно је више пута покренути програм ако се не добије резултат приказан на слици. Не постоји гаранција да ће се дефинитивно покренути метод `gc()`!

```
public class TestGC {  
  
    @Override  
    public void finalize() {  
        System.out.println("object is garbage collected " + this);  
    }  
  
    public static void main(String args[]) {  
        TestGC s1 = new TestGC();  
        TestGC s2 = new TestGC();  
        s1 = null;  
        s2 = null;  
        System.gc();  
    }  
}
```

```
Run: TestGC x  
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe"  
object is garbage collected TestGC@46d08f12  
object is garbage collected TestGC@481779b8  
Process finished with exit code 0
```



# Објектно оријентисано програмирање

- У дефиницију класе може се укључити метод **finalize()**.
- Метод **finalize()** је члан класе **Object** и њега наслеђују све подкласе класе **Object** (а то су све класе).
- Овај метод се позива аутоматски пре него се објекат коначно уништи и ослободи простор који је заузимао у меморији.
- У пракси, то може бити нешто након што објекат постане недоступан у програму.
- Метод **finalize()** је користан ако објекти класе користе ресурсе који захтевају неку специјалну акцију када се уништавају.
- Типично, то су ресурси који нису из Јава окружења и не гарантује се да ће их објекат сам ослободити.
- То могу бити графички ресурси, фонтови или други ресурси које је обезбедио оперативни систем или екстерни фајлови на "хард" диску.



# Објектно оријентисано програмирање

- Иницијално, метод **finalize()** не ради ништа.
- Он се може редефинисати тако да се у оквиру њега извршава жељена активност – ми смо исписивали поруку у нашем примеру.
- Не можемо рачунати на то да ће објекат бити уништен када више није доступан коду нашег програма (*видели смо да то није лако "изазвати"*).
- JVM ће се ослободити нежељених објеката и ослободити меморију коју они заузимају **једино ако јој понестаје меморије** или ако нема активности у нашем програму – нпр. када чека на улаз.
- Као последица тога, може се десити да се објекти не униште док се програм не заврши (*видели смо на нашем примеру*).
- **Ништа што зависи од времена** не би требало остављати **finalize()** методу да решава!



# Објектно оријентисано програмирање

- **Overloading** – коришћење истог имена или симбола за означавање више различитих конструкција у језику.
- У Јави је то **могућност** декларисања **више истоимених метода и конструктора у једној класи**.
- Ови методи се морају међусобно **разликовати** по броју или типу својих формалних **параметара**.
- Поред **Overloading**-а имена метода, у Јави се **Overloading** користи и код два операторска симбола:
  - **+** се користи као унарни плус (предзнак), као симбол сабирања бројева **и као симбол повезивања стрингова**,
  - симбол **-** се користи као унарни минус (предзнак) и као симбол одузимања бројева.

```
System.out.println("Ja " + "volim " + "da " + "gledam filmove." );
```

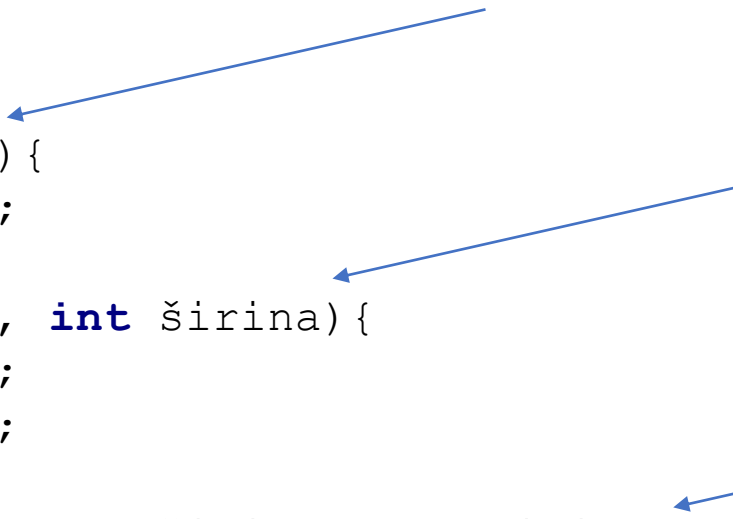




# Објектно оријентисано програмирање

## Пример: Класа "Кутија" - Overloading конструктора класе

```
public class Kutija {  
    public int dužina;  
    public int širina;  
    public int visina;  
    public Kutija(int visina) {  
        this.visina = visina;  
    }  
    public Kutija(int dužina, int širina) {  
        this.dužina = dužina;  
        this.širina = širina;  
    }  
    public Kutija(int dužina, int širina, int visina) {  
        this.dužina = dužina;  
        this.širina = širina;  
        this.visina = visina;  
    }  
    public int zapremina() {  
        return dužina * širina * visina;  
    }  
}
```





# Објектно оријентисано програмирање

- Објекат "к1" морамо креирати позивањем једног од три понуђена конструктора класе "Кутија":

```
1 public class GlavnaKlasa {  
2     Kutija k1 = new Kutija();  
3 }  
4
```

Change signature of Kutija(int)

Change signature of Kutija(int, int)

Change signature of Kutija(int, int, int)

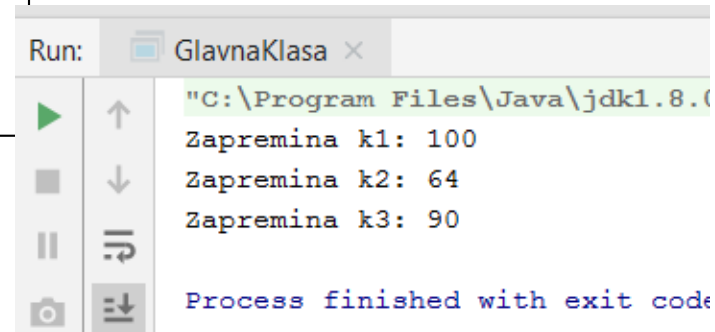
Create constructor

Cannot resolve constructor 'Kutija()'



# Објектно оријентисано програмирање

```
public class GlavnaKlasa {  
  
    public static void main(String[] args) {  
  
        Kutija k1 = new Kutija(5, 10, 2);  
        System.out.println("Zapremina k1: " + k1.zapremina());  
  
        Kutija k2 = new Kutija(4);  
        k2.dužina = 4;  
        k2.širina = 4;  
        System.out.println("Zapremina k2: " + k2.zapremina());  
  
        Kutija k3 = new Kutija(6, 3);  
        k3.visina = 5;  
        System.out.println("Zapremina k3: " + k3.zapremina());  
  
    }  
}
```





# Објектно оријентисано програмирање

## Пример: Класа "Кутија" - Overloading метода класе

```
public class Kutija {  
    public int dužina;  
    public int širina;  
    public int visina;  
  
    public Kutija(int dužina, int širina, int visina) {  
        this.dužina = dužina;  
        this.širina = širina;  
        this.visina = visina;  
    }  
  
    public int zapremina() {  
        return dužina * širina * visina;  
    }  
  
    public int zapremina(int visina) {  
        return dužina * širina * visina;  
    }  
}
```

Искористићемо могућност да задамо висину кутије, **која није атрибут класе** али је локална променљива методе која рачуна запремину.



# Објектно оријентисано програмирање

```
public class GlavnaKlasa {  
  
    public static void main(String[] args) {  
  
        Kutija k1 = new Kutija(5, 10, 2);  
  
        System.out.println("Zapremina k1: " + k1.zapremina());  
        System.out.println("Zapremina k1: " + k1.zapremina(4));  
    }  
}
```

```
Run: GlavnaKlasa x  
"C:\Program Files\Java\jdk1.8.0_121\bin  
Zapremina k1: 100  
Zapremina k1: 200  
  
Process finished with exit code 0
```



# Објектно оријентисано програмирање

## Пример:

- Креирати класу "Студент" која садржи јавне атрибуте "име", "бр. индекса", "година студија".
- Класа "Студент" треба да садржи два конструктора, са различитим параметрима (по слободном избору).
  - Приказати како се конструктори генеришу "аутоматски" (што у потпуности уклања могућност појаве синтаксне грешке).
- Класа "Студент" треба да садржи две методе "представи се", које не "враћају" никакву вредност, са различитим параметрима.
  - прва метода без параметара,
  - друга метода са параметром "година студија" који је референца на атрибут класе "година студија" (за разлику од прошлог примера класе "Кутија" где параметар није био референца ка атрибуту класе).
- У главној класи креирати објекте помоћу оба конструктора.
- Објекти треба да позову одговарајући/ће метод/е.



# Објектно оријентисано програмирање

```
public class Student {
    public String ime;
    public String brIndeksa;
    public int godStudija;
    public Student(String ime, String brIndeksa, int godStudija) {
        this.ime = ime;
        this.brIndeksa = brIndeksa;
        this.godStudija = godStudija;
    }
    public Student(int godStudija) {
        this.godStudija = godStudija;
    }
    public void predstaviSe() {
        System.out.print("Student: " + ime + ", " + brIndeksa);
        System.out.println(" upisan je u " + godStudija + ". godinu.");
    }
    public void predstaviSe(int godStudija) {
        this.godStudija = godStudija;
        System.out.print("Student: " + ime + ", " + brIndeksa);
        System.out.println(" upisan je u " + godStudija + ". godinu.");
    }
}
```



# Објектно оријентисано програмирање

```
public class GlavnaKlasa {  
    public static void main(String[] args) {  
        Student s1 = new Student("Petar Petrović", "189/2020", 1);  
        Student s2 = new Student(2);  
        s2.ime = "Ana Milić";  
        s2.brIndeksa = "100/2019";  
        s1.predstaviSe();  
        s2.predstaviSe();  
        // s1 student je upisao drugu godinu  
        s1.predstaviSe(2);  
        // sada smo promenili vrednost atributa  
        // tako da i pozivom prve metode dobijamo isti ispis na ekranu  
        // ovo naravno nije dobar način za pristup i promenu atributa klase,  
        // pa ćemo zbog toga objasniti pojam enkapsulacija  
        s1.predstaviSe();  
    }  
}
```

```
Run: GlavnaKlasa x  
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...  
Student: Petar Petrović, 189/2020 upisan je u 1. godinu.  
Student: Ana Milić, 100/2019 upisan je u 2. godinu.  
Student: Petar Petrović, 189/2020 upisan je u 2. godinu.  
Student: Petar Petrović, 189/2020 upisan je u 2. godinu.  
  
Process finished with exit code 0
```





# Објектно оријентисано програмирање

- До сада смо мало пажње усмеравали ка контроли приступа атрибутима неке класе.
- У досадашњим примерима, атрибути класе су углавном били декларисани модификатором **public**.
- Када смо причали о контроли приступа на уводном предавању (предавање 08) издвојили смо три модификатора:
  - **public**
    - Члан је приступачан са произвољног места у било којој класи дефинисаног пројекта.
  - **protected**
    - Члан је приступачан у изведеним поткласама и у коду целог пакета.
  - **private**
    - Члан је приступачан само у класи где је дефинисан.



# Објектно оријентисано програмирање

- Под дизајном класе подразумевамо начине на које ће се објектима те класе приступати, тј. како та класа изгледа споља: **којим пољима се може приступати изван тела класе, којим методима и шта ти методи треба да ураде.**
- Под имплементацијом класе подразумевамо **конкретну реализацију** свега онога што смо приликом дизајнирања класе замислили.
- Да би се програм лакше направио, одржавао и модификовао, као и да би се омогућио тимски рад на прављењу програма, **веома је значајно да имплементација сваке класе остане сакривена** колико год је то могуће.
- Како је нека класа имплементирана, то треба да зна само програмер **који ју је направио.**
- Креатори других класа **не морају знати ништа о имплементацији посматране класе** да би је користили.



# Објектно оријентисано програмирање

- Долазимо до појма **енкапсулације** - скривање детаља имплементације класе.
- Значај енкапсулације: несметано модификовање класа и одржавање објектно оријентисаних програма (посебно великих).
- Сви атрибути класе (поља) треба да буду сакривена и дефинисана са модификатором **private**.
- Приступ вредностима тих поља из других класа треба омогућити **само преко јавних метода**.
- Дакле, овим приступом, сви итрибути (и интерне методе, по потреби) су заштићени унутар чауре класе и могу се мењати само на контролисан начин.
- У ту сврху се дефинишу **јавне методе** са именима која почињу речима **get** и **set**.



# Објектно оријентисано програмирање

- Методе **get** и **set** су део **јавног интерфејса** класе.
- Методе чија имена почињу са **get** само **враћају вредности атрибута класе (поља)** – скраћено се често називају "гетери".
- Методи чија имена почињу са **set** увек **мењају вредност атрибута класе (поља)** – скраћено се често називају "сетери" (мутатори).
- Конвенција – име "гет" или "сет" метода за неку променљиву се прави додавањем речи **get** или **set** испред имена те променљиве са великим почетним словом.
- Применом енкапсулације осигурано је да се не може ни у једном делу програма ван класе **случајно или намерно** доделити нека неконзистентна вредност атрибутима класе.
- У нашем *IDE* постоје начини да се аутоматски генеришу **get** и **set** методе, чиме је додатно олакшана њихова имплементација у коду класе.



# Објектно оријентисано програмирање

## Пример:

- Креирати класу "Студент" која садржи **приватне атрибуте** "име", "бр. индекса", "година студија".
- Класа "Студент" треба да садржи конструктор који **иницијализује** све приватне атрибуте класе.
- Класа "Студент" треба да садржи **get** и **set** методе које обезбеђују приступ атрибутима.
- Класа "Студент" треба да садржи методе "представи се" који не "враћа" никакву вредност.
  - **САВЕТ:** Водити рачуна о дозволама за приступ приватним члановима класе.
  - **Да ли је могуће приступити приватним атрибутима класе у методи "представи се"?**
  - **Зашто?**
- У главној класи креирати објекте "s1" и "s2".
- Објекти треба да позову одговарајући метод "представи се".
- Након тога променити вредност атрибута објектима (по слободном избору) и поново позвати одговарајући метод "представи се".



# Објектно оријентисано програмирање

```
Student.java x GlavnaKlasa.java x
1 public class Student {
2     private String ime;
3     private String brIndeksa;
4     private int godStudijs;
5
6     public Student(String ime, String brIndeksa, int godStudijs) {
7         this.ime = ime;
8         this.brIndeksa = brIndeksa;
9         this.godStudijs = godStudijs;
10    }
11
12
13
14
15
16
17
18
19
20
21    public void ispis() {
22        System.out.print("Student: " + ime + ", " + brIndeksa);
23    }
24 }
```

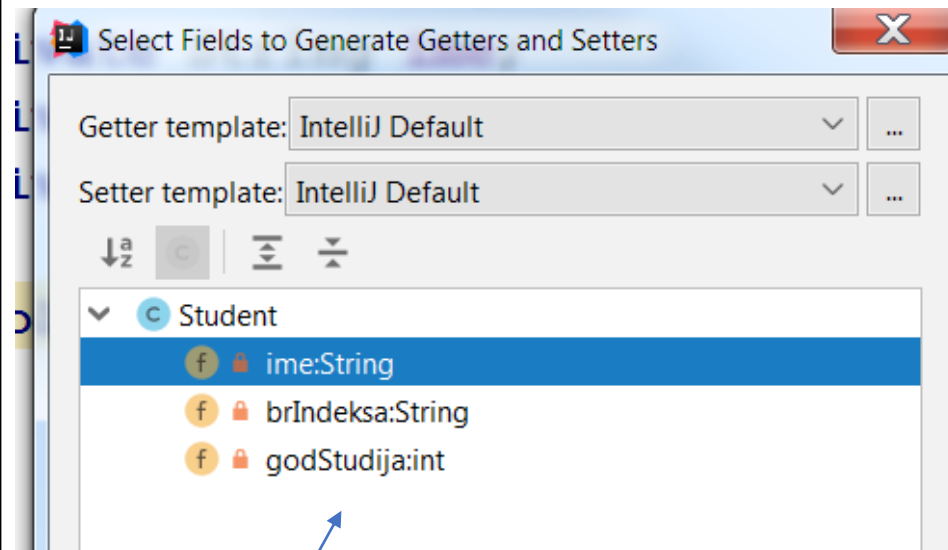
Generate

- Constructor
- Getter
- Setter
- Getter and Setter
- equals() and hashCode()
- toString()
- Override Methods... Ctrl+O
- Delegate Methods...
- Test...
- Copyright



# Објектно оријентисано програмирање

```
public class Student {  
    private String ime;  
    private String brIndeksa;  
    private int godStudija;  
  
    public Student(String ime, String brIndeksa, int godStudija) {  
        this.ime = ime;  
        this.brIndeksa = brIndeksa;  
        this.godStudija = godStudija;  
    }  
  
    public String getIme() {  
        return ime;  
    }  
  
    public void setIme(String ime) {  
        this.ime = ime;  
    }  
  
    public String getBrIndeksa() {  
        return brIndeksa;  
    }  
  
    public void setBrIndeksa(String brIndeksa) {  
        this.brIndeksa = brIndeksa;  
    }  
}
```



У нашем примеру селекујемо сва три атрибута.

Такође, у зависности од потреба, могуће се креирати само "гетере" или само "сетере" за одређене атрибуте класе.



# Објектно оријентисано програмирање

```
public int getGodStudija() {  
    return godStudija;  
}  
  
public void setGodStudija(int godStudija) {  
    this.godStudija = godStudija;  
}  
  
public void predstaviSe() {  
    System.out.print("Student: " + ime + ", " + brIndeksa);  
    System.out.println(" upisan je u " + godStudija + ". godinu.");  
}  
}
```

У нашем примеру атрибутима класе могу да приступају сви чланови класе иако је модификатор приступа дефинисан као **private**.

Погледати дефиницију на слајду 17.





# Објектно оријентисано програмирање

```
public class GlavnaKlasa {  
  
    public static void main(String[] args) {  
  
        Student s1 = new Student("Petar Petrović", "189/2020", 1);  
        Student s2 = new Student("Ana Milić", "100/2019", 2);  
  
        s1.predstaviSe();  
        s2.predstaviSe();  
  
        s1.setGodStudijska(2);  
        System.out.println();  
        s1.predstaviSe();  
  
        System.out.println();  
        System.out.println("Student: " + s2.getIme() + " radi kolokvijum.");  
    }  
}
```

```
Run: GlavnaKlasa x  
"C:\Program Files\Java\jdk1.8.0_121\bin\java.exe" ...  
Student: Petar Petrović, 189/2020 upisan je u 1. godinu.  
Student: Ana Milić, 100/2019 upisan je u 2. godinu.  
  
Student: Petar Petrović, 189/2020 upisan je u 2. godinu.  
  
Student: Ana Milić radi kolokvijum.  
  
Process finished with exit code 0
```



# Литература

Градиво осмог предавања :

- Поглавље 7:

<https://singipedia.singidunum.ac.rs/izdanje/40716-osnove-java-programiranja>

- Одабрана поглавља из књиге: **Јава JDK9: Комплетан приручник**
  - Аутор: *Herbert Schildt* (може и старије издање *JDK7*).
- Књига: **Објектно оријентисани начин мишљења**
  - Аутор: *Matt Weisfeld*
- Не морате да купујете наведене књиге.