

# Објектно оријентисано програмирање



Владимир Филиповић  
[vladaf@matf.bg.ac.rs](mailto:vladaf@matf.bg.ac.rs)

Александар Картељ  
[kartelj@matf.bg.ac.rs](mailto:kartelj@matf.bg.ac.rs)

# Сопствени пакети и унутрашње класе



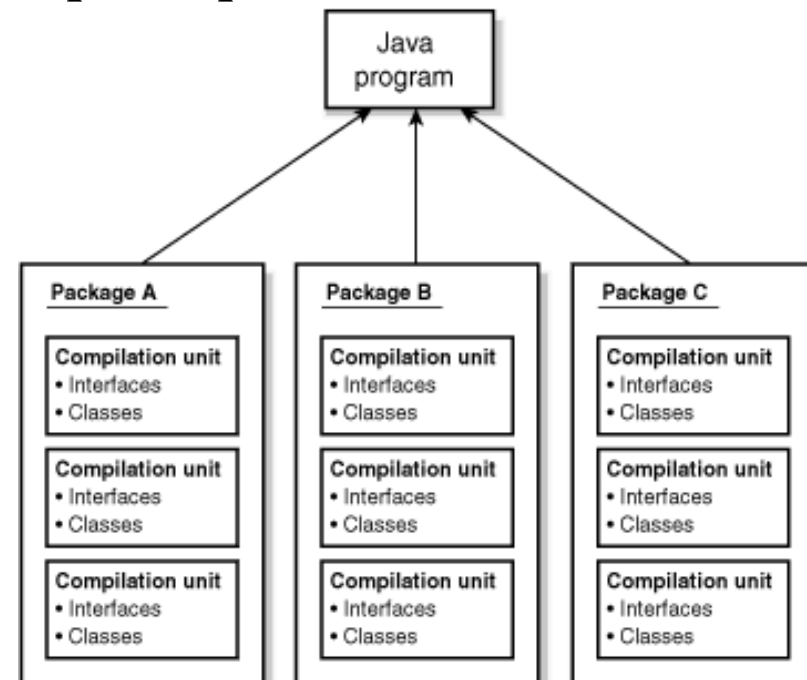
Владимир Филиповић  
[vladaf@matf.bg.ac.rs](mailto:vladaf@matf.bg.ac.rs)

Александар Картељ  
[kartelj@matf.bg.ac.rs](mailto: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, еnumerисаних типова и типова нотације) за коју је обезбеђује заштита при приступу и управљање простором имена.





## 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.



## 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.



## 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.



## 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:

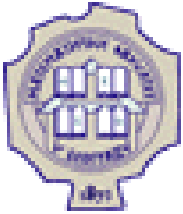
```
package rs.ac.bg.matf;
```



## 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.



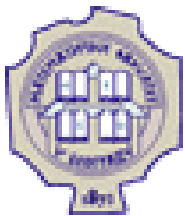


# Угнеждене класе

- Јава допушта да се дефинише класа унутар неке друге класе. Таква класа се назива **угнеждена класа**.
- Постоје два начина креирања угнеждених класа:

```
class OuterClass {  
    ...  
    static class StaticNestedClass {  
        ...  
    }  
  
    class InnerClass {  
        ...  
    }  
}
```

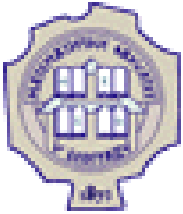
- Прва се назива **статичка угнеждена класа**, а друга само **унутрашња класа**.



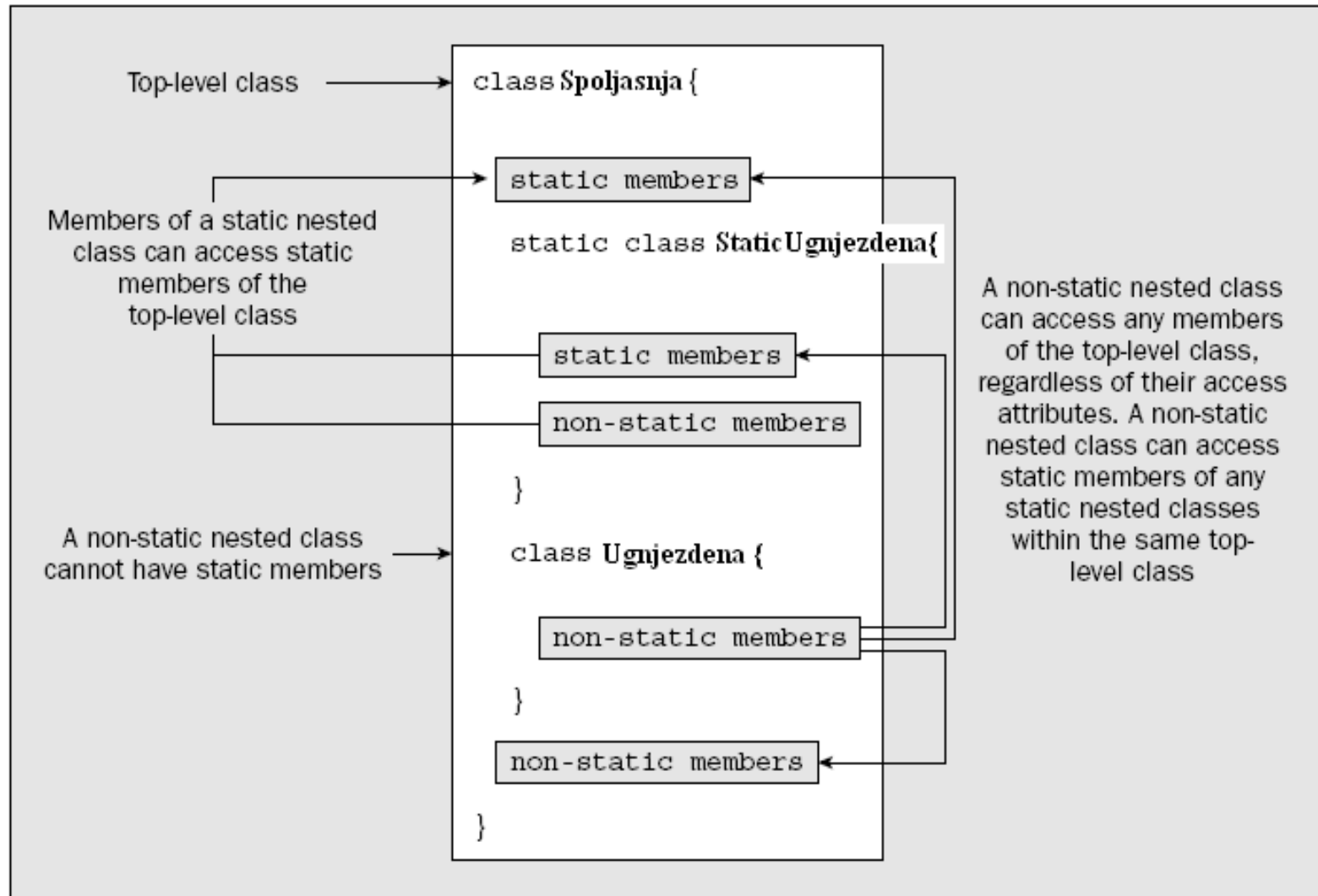
## Угнеждене класе (2)

Постоји неколико разлога за коришћење угнеждених класа:

1. То је начин логичког груписања класа које се користе само на једном месту.
2. Тиме се побољшава енкапулација (учаурење).  
Ако се класа  $B$  смести унутар класе  $A$ , поља класе  $A$  могу бити приватна а класа  $B$  и даље може да им приступа.
3. Угнеждене класе доводе до читљивијег кода који се лакше одржава.  
Угнеждавање малих класа у велику доводи до тога да код неке операције буде смештен близу места коришћења.



# Угнеждене класе (3)





# Угнеждене класе (4)

```
package ugnjezdeneklase;

public class Spoljasnja {
    private static int sp;
    private int np;
    private void nm() {}
    private static void sm() {}

    //instanca ugnjezdene staticke klase nije uslovljena postojanjem instance spoljne klase
    static class StatickaUgnjezdena{
        private static int sp;
        private int np;

        private void nm(){
            //pristup spoljnoj statickoj
            Spoljasnja.sp=5;
            //ne mozemo pristupiti nestatickoj promenljivoj iz
            //staticke klase
            //Spoljasnja.np = 8;
            //ne mozemo pristupiti ni nestatickoj metodi
            //spoljne klase
            //Spoljasnja.nm();
            //ali mozemo statickoj metodi spoljne klase
            Spoljasnja.sm();
        }
        private static void sm(){
            //slicno kao i u nestatickoj nm() metodi,
            //mozemo pristupiti sledecim elementima spoljne klase
            Spoljasnja.sp=5;
            Spoljasnja.sm();
        }
    }
}
```



# Угнеждене класе (5)

```

class Unutrasnja{
    //unutrasnja klasa ne moze imati staticke promenljive i metode
    //private static int sp;
    private int np;
    private void nm(){
        //pristup spoljnim statickim i nestatickim promenljivama
        Spoljasnja.sp=5;
        //ovo je np iz objekta Spoljasnje klase
        Spoljasnja.this.np=7;
        //a ovo je np iz objekta klase Ugnjezdjena
        np=8;
        //slicno i za metode
        Spoljasnja.sm();
        //nestaticka metoda spoljne klase
        Spoljasnja.this.nm();
        //nestaticka metoda ove klase (rekurzija!)
        nm();
        //pristup statickim elementima klase StatickaUgnjezdjena
        StatickaUgnjezdjena.sp=5;
        StatickaUgnjezdjena.sm();
    }
}

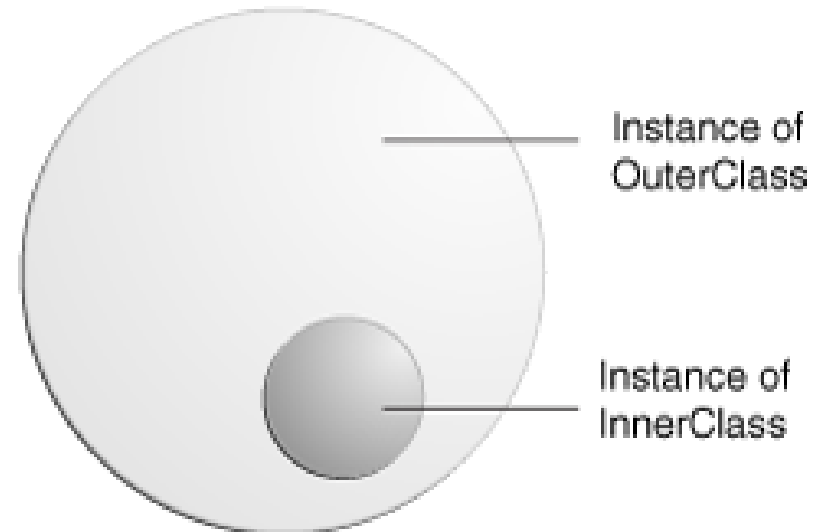
public static void main(String[] args) {
    //ne moze ovako
    //Unutrasnja un=new Unutrasnja();
    Spoljasnja sp=new Spoljasnja();
    Unutrasnja un=sp.new Unutrasnja();
    StatickaUgnjezdjena su=new StatickaUgnjezdjena();
}
}

```



# Унутрашње класе

- Унутрашња класа придружује примерку класе која је обухвата, па и она има директан приступ до свих поља и метода објекта који је садржи.
- Како је унутрашња класа придружена примерку тј. објекту, то она сама не може садржати статички члан.
- Објекти примерци унутрашње класе постоје само у оквиру примерка спољашње класе.





## Унутрашње класе (2)

- Када се дефинише унутрашња класа, она је члан спољашње класе на исти начин као и остали чланови (поља и методе).
- Унутрашња класа може имати модификаторе приступа као и остали чланови.
- Угњеждена класа би требало да има специфичну повезаност са спољашњом класом. Произвољно угњеждење једне класе у другу нема много смисла.
- Класа највишег нивоа је класа која садржи угњеждenu класу, али сама по себи није угњеждена



# Пример унутрашње класе

- Последом наредбом је креиран објекат типа Spoljasnja.

```
public class Spoljasnja {  
    // Unutrasnja  
    public class Unutrasnja {  
        //Detalji unutrasnje klase ...  
    }  
    // Ostali clanovi Spoljasnje klase  
}  
  
Spoljasnja sp = new Spoljasnja();
```





## Пример унутрашње класе (2)

- Последњом наредбом на претходном слајду није креиран објекат класе `Unutrasnja`.
- Ако је потребно да се креира примерак унутрашње класе, то се постиже тако што се реферише на име унутрашње класе са именом спољашње класе као квалификатором:

```
// definisanje objekta unutrasnje klase  
Spoljasnja.Unutrasnja unut = sp.new Unutrasnja();
```

- У горњем примеру је креиран објекат унутрашње класе који је придружен раније креираном објекту `sp`.



## Пример унутрашње класе (3)

- Унутар нестатичких метода класе `Spoljasnja`, можемо користити име класе `Unutrasnja` без квалификовања.
- Тако се унутар нестатичког метода класе `Spoljasnja` може креирати нови примерак класе `Unutrasnja` са:

```
Unutrasnja unut = new Unutrasnja ();
```

што је еквивалентно са:

```
this.Unutrasnja unut = this.new Unutrasnja ();
```



# Локалне унутрашње класе

- Класа се може дефинисати унутар метода, и то је тзв. локална унутрашња класа (енг. local inner class).
- Примерци тј. објекти локалне унутрашње класе могу се креирати само локално, тј. унутар метода у коме је и дефиниција класе.
- То је корисно када израчунавање у методу захтева употребу специјализоване класе.
- Добар пример су ослушкивачи догађаја (енг. event listener).
- Локална унутрашња класа може реферисати променљиве декларисане у методу у чијој дефиницији се појављује, али само ако су те променљиве финалне.



# Локалне унутрашње класе (2)

```
public class PrimerLokalneKlase {
    static String reg= "[^0-9]";
    public static void validirajBrojTelefona(String broj) {
        //mora da bude finalna kako bi se videla unutar lokalne metode
        final int duzinaBroja = 10;
        //lokalna klasa jer je koristimo samo za potrebe ove metode
        class BrojTelefona {
            String formatiraniBroj = null;
            BrojTelefona(String broj) {
                String trenutniBroj =broj.replaceAll(reg, "");
                if (trenutniBroj.length() == duzinaBroja)
                    formatiraniBroj = trenutniBroj;
                else
                    formatiraniBroj = null;
            }
            public String vratiBroj() {
                return formatiraniBroj;
            }
        }
        BrojTelefona mojBroj = new BrojTelefona(broj);
        if (mojBroj.vratiBroj() == null)
            System.out.println("Broj nije dobar");
    }

    public static void main(String... args) {
        validirajBrojTelefona("123-456-7890");
    }
}
```



# Анонимне класе

- Анонимне класе омогућују програмеру да пише концизан код.
- Оне допуштају да се истовремено декларише класа и креира њен примерак, при чему та класа нема име.
- Другим речима, оне су као локална класа, само што немају име. Анонимне класе се користе када се локална класа користи само једном.



# Анонимне класе (2)

```
public class Student {
    private int godinaRodjenja;
    private String ime, prezime;

    public Student(int godinaRodjenja, String ime, String prezime) {
        this.godinaRodjenja = godinaRodjenja;
        this.ime = ime;
        this.prezime = prezime;
    }

    public static void main(String[] args) {
        Student[] nizStudenata = new Student[3];
        nizStudenata[0] = new Student(1992, "Ana", "Pesic");
        nizStudenata[1] = new Student(1990, "Mirko", "Petrovic");
        nizStudenata[2] = new Student(1985, "Drgana", "Djordjevic");
        /*Koristimo anonimnu klasu koja nasledjuje Comparator jer nam treba specifican
        nacin sortiranja niza i to samo jedanput. Neefikasna alternativa bi bila da smo
        nasledili klasu komparator nasom klasom npr. MojKomparator, potom redefinisali
        njen compare metod, ali bi to bilo previse koda za jednokratnu upotrebu*/
        Arrays.sort(nizStudenata, new Comparator() {
            @Override public int compare(Object o1, Object o2) {
                Student s1 = (Student) o1; Student s2 = (Student) o2;
                return s1.godinaRodjenja - s2.godinaRodjenja;
            }
        });
    }
}
```



# 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 оријентисано програмирање) направио проф. др Душан Тошић.

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

Надаље, један део материјала је преузет од колегинице Марије Милановић.

Хвала Марији Милановић на помоћи у реализацији ове презентације.