Ugnježđeni tipovi

najveći deo teksta je preuzet sa slajdova
Prof. Dragana Milićeva (ETF Bg)
namenjenih pedmetu OOP2

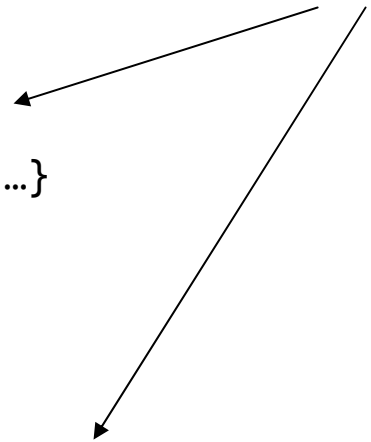
Ugneždeni tipovi

- Unutrašnje klase
- Statički ugnježđeni tipovi
- Lokalne klase
- Anonimne klase

Ugneždavanje tipova

- Definicija tipa (klasa ili interfejs) može se ugnezditi u
  - definiciju neke klase ili interfejsa
  - telo metoda ili bilo kog bloka u kodu ({…})
- Primeri:
  -
```
class MyTopLevel{
    …
    class StaticNested {…}
    …}
 }
```
  -
```
class MyClass{
    …
    public void SomeMethod(){
            …
            class StaticNested {…}
            …
    …}
}
```
  -
```
class MyClass{
    …
    public void SomeMethod(){
            …
            { class StaticNested {…} }
            …
    …}
}
```

Koja je razlika?



Ugneždavanje tipova

```
class MyTopLevel{
   private String top = "From Top level class";
   public void createNestedInMethod() {
         class Prava {
                int i=1;
                public void getPrava(){System.out.println(i);}
         }
         Prava p=new Prava();
         { class NePrava{
                int j=1;
           }
           NePrava np=new NePrava();
         }
         NePrava np=new NePrava();
   }
}
```

ugneždeno u metod

ugneždeno u blok

Ugneždene klase i interfejsi

- Klasa ili interfejs može biti član druge klase ili interfejsa
  - takav tip se naziva **ugneždenim**
- Ugnežden tip treba definisati u slučaju kada on ima smisla samo u kontekstu obuhvatajućeg tipa
  - klasa `TextCursor` može biti ugneždena u klasu `Text`
- Ugneždena klasa može:
  - da bude izvedena iz proizvoljne klase,
  - da implementira proizvoljan interfejs,
  - da bude osnova za proširivanje
  - da se deklariše kao `final` ili kao `abstract`
- Ime ugneždenog tipa
  - dostupno je direktno u obuhvatajućem tipu,
  - izvan mu se pristupa kvalifikacijom:

```
<ObuhvatajuciTip>.<UgneždeniTip>
```

Prava pristupa – unutar obuhvatajućeg tipa

Uzajamni odnos obuhvatajuće i ugneždene klase je prijateljski

- Kao član obuhvatajuće klase, ugneždena klasa ima pristup svim članovima obuhvatajuće klase, čak i ako su deklarisani kao privatni (ovo važi i u obrnutom smeru, obuhvatajuća može da pristupi članovima unutrašnje)

Ova specijalna privilegija je potpuno konzistentna sa značenjem `private` -
specifikatori pristupa ograničavaju pristup članovima za klase koje su izvan obuhvatajuće klase, a ugneždena klasa je unutar obuhvatajuće klase, pa treba da ima pristup svim njenim članovima

- Obuhvatajuća klasa takođe ima potpuni pristup članovima ugneždene klase

Napomena: `static` ugnježđeni tipovi su specifični kao i u slučaju `static` podataka ili metoda.

- Klasa koja proširuje ugneždenu klasu ne nasleđuje prava pristupa

Prava pristupa – “spolja”, primena modifikatora pristupa

- Za tipove ugneždene u klasu
  - mogući specifikatori su `public`, `protected`, `private`
  - i mogu se koristiti da ograniče pristup ugneždenim tipovima, kao i svim drugim članovima klase
- Tipovi ugneždeni u interfejse su uvek (podrazumevano) javni – zašto?

Ugneždene i unutrašnje klase

`static` ugneždeni tip	nestatički ugnježđeni tip
▪ ugneždena klasa deklarisana kao `static` se naziva statičkom ugneždenom klasom (static nested class)	▪ Nestatička ugneždena klasa se naziva unutrašnjom klasom (inner class)
▪ Statički ugneždeni tipovi služe kao mehanizam strukturiranja tipova	▪ Omogućavaju definisanje posebnog odnosa zavisnosti između njihovih objekata i objekata spoljašnje klase
▪ Statička ugneždena klasa ne može direktno (imenovanjem bez kvalifikacije) da pristupa nestatičkim poljima ili metodima obuhvatajuće klase (može preko reference na objekat) ▪ Klasa ugneždena u interfejs je podrazumevano statička ▪ Ugneždeni interfejs je podrazumevano statički	▪ Unutrašnja klasa može direktno da pristupa nestatičkim članovima spoljašnje ▪ Unutrašnja klasa ne može da sadrži statičke članove (izuzev finalnih statičkih polja inicijalizovanih konstantnim izrazom) ▪ Objekat unutrašnje klase je uvek u vezi sa jednim objektom spoljašnje klase ▪ Objekat spoljašnje klase može da bude u vezi sa više objekata unutrašnje klase

Relacije objekata

- Termin "unutrašnja" reflektuje relaciju između objekata ugneždene i obuhvatajuće klase - objekat unutrašnje klase može postojati samo u vezi sa objektom obuhvatajuće klase
  - za razliku od "ugneždene“ koja reflektuje sintaksnu relaciju između dve klase - kod jedne klase se pojavljuje unutar koda druge klase

- Definicija unutrašnje klase:
  - unutrašnja klasa je ugneždena klasa čiji objekat postoji “unutar” nekog objekta njene obuhvatajuće klase (ima implicitnu referencu na njega) i ima direktan pristup nestatičkim članovima obuhvatajuće klase
  - Relacija između objekta obuhvatajuće klase i unutrašnje klase se uspostavlja prilikom kreiranja objekta unutrašnje klase

```
this.new Unutrašnja(...); // može i bez this
this – objekat obuhvatajuće klase
```

Primer

```
public class RacunUbanci {
   private long broj; private long stanje;
   private Akcija poslednjaAkcija;
   public class Akcija {
         private String akcija; private long iznos;
         Akcija (String akcija, long iznos) {
                  this.akcija=akcija; this.iznos=iznos;}
         public String toString() {
                  return broj + ": " + akcija + " " + iznos;}
   }
   public void uplata (long iznos){
         stanje+=iznos;
         poslednjaAkcija=new Akcija("uplata", iznos);}
   public void isplata(long iznos){
         stanje-=iznos;
         poslednjaAkcija=new Akcija("isplata", iznos);}
   public void prenos(RacunUbanci drugi, long iznos){
         drugi.isplata(iznos);
         uplata(iznos);
         poslednjaAkcija=this.new Akcija("prenosna",iznos);
         drugi.poslednjaAkcija=drugi.new Akcija("prenossa", iznos);}
}
```

Uzajamni pristup

- Unutrašnja klasa može da pristupi bez kvalifikovanja članu spoljašnje
- Spoljašnja klasa može da pristupi članu unutrašnje samo preko kvalifikacije (imena objekta)
- Pri kreiranju objekta unutrašnje klase
  - uspostavlja se referenca prema objektu spoljašnje
  - referenca na spoljašnji objekat se ponaša kao sekundarni `this` i omogućava pristup članovima spoljašnje klase preko imena, bez kvalifkacije

    primer: navođenje `broj` u metodi `toString()`

    puna kvalifikacija bi bila: `RačunUbanci.this.broj`

Lokalne klase

- Lokalna klasa se definiše unutar proizvoljnog programskog bloka
  - unutar metoda,
  - unutar konstruktora ili
  - unutar inicijalizacionog bloka

- Lokalne klase nisu članovi okružujuće klase
- Lokalne klase su nepristupačne izvan bloka u kojem su definisane
- Instance lokalne klase su normalni objekti koji se mogu
  - prenositi kao argumenti ili
  - vraćati kao rezultati metoda
- Iz okružujućeg bloka, lokalna klasa ima pristup samo:
  - `final` lokalnim varijablama ili
  - `final` argumentima metoda

Lokalne klase

- U paketu `java.util` postoji interfejs:

```
public interface Iterator{
boolean hasNext();
Object next() throws NoSuchElementException;//...
}
```

- Može se pisati metod koji vraća `Iterator`:

```
public static Iterator obilazak(final Object[] objekti){
 class Iter implements Iterator{
   private int p = 0; //pozicija u nizu objekti
   public boolean hasNext() {return (p<objekti.length);}
   public Object next() throws NoSuchElementException {
      if (p>=objekti.length) throw new NoSuchElementException();
      return objekti[p++];
   }
 }
 return new Iter();
 }
```

- Klasa `Iter` je lokalna klasa, klijenti metoda `obilazak()` nisu svesni tipa `Iter`
- Metoda može da vraća tip `Iter` - zašto?

Anonimne klase

- Ako ime lokalne klase nije potrebno može se deklarisati anonimna klasa
- Anonimna klasa proširuje drugu klasu ili implementira neki interfejs
- Anonimna klasa se definiše u izrazu new, kao deo naredbe

  Ime supertipa (osnovne klase ili interfejsa) se koristi za anonimnu klasu
- Anonimna klasa ne može da ima eksplicitne klauzule extends i implements
- Primer:

```
public static Iterator obilazak(final Object[] objekti){
   return new Iterator{
      private int p = 0; //pozicija u nizu objekti
      public boolean hasNext(){returnp<objekti.length;}
      public Object next() throws NoSuchElementException {
       if (p>=objekti.length) throw newNoSuchElementException();
       return objekti[p++];}
   }
 }
```

Anonimne klase

- Anonimna klasa ne može imati konstruktor, jer konstruktor nosi ime klase koje ne postoji
- Ako je potreban konstruktor superklase, iza imena apstraktne klase se dodaju argumenti:

Primer:

Neka postoji klasa `Atribut` sa konstruktorom koji ima argument tipa `String`

```
Atributime = new Atribut("Ime"){/*...*/}
```

- biće pozvan `super("Ime")`, odnosno konstruktor klase `Atribut` sa argumentom tipa String
- Kada anonimna klasa implementira ineterfejs, onda se poziva samo konstruktor klase `Object`