

Objektno orijentisano programiranje 2

Ugnežđeni tipovi

Ugnežđene klase i interfejsi

- Klasa ili interfejs može biti član druge klase ili interfejsa
 - takav tip se naziva ugnežđenim
- Primer:

```
class ObuhvatajućaKlasa{ class UgnežđenaKlasa { . . . } }
```
- Ugnežđen tip treba definisati kad on ima smisla
 - treba da ima smisla samo u kontekstu obuhvatajućeg tipa
- Primer:
 - klasa `TekstKurzor` može biti ugnežđena u klasu `Tekst`
- Ugnežđena klasa može:
 - da proširuje proizvoljnu klasu,
 - da implementira proizvoljan interfejs,
 - da bude osnova za proširivanje,
 - da se deklariše kao `final` ili kao `abstract`
- Ime ugnežđenog tipa
 - dostupno je direktno u obuhvatajućem tipu,
 - izvan mu se pristupa kvalifikacijom: `<ObuhvatajućiTip>. <UgnežđeniTip>`

Prava pristupa

- Uzajamni odnos obuhvatajuće i ugnežđene klase je prijateljski
 - potpuno uzajamno poverenje
- Kao član obuhvatajuće klase, ugnežđena klasa ima specijalnu privilegiju:
 - ima pristup svim članovima obuhvatajuće klase, čak i ako su privatni
- Ova specijalna privilegija je potpuno konzistentna sa značenjem `private`
 - specifikatori pristupa ograničavaju pristup članovima za klase koje su *izvan* obuhvatajuće klase
 - ugnežđena klasa je *unutar* obuhvatajuće klase (njen je član), pa treba da ima pristup svim njenim članovima
- Obuhvatajuća klasa takođe ima potpuni pristup članovima ugnežđene klase
- Za tipove ugnežđene u klasu, mogući specifikatori:
 - `public`, `protected`, `private`
 - mogu se koristiti da ograniče pristup ugnežđenim tipovima, kao i svim drugim članovima klase
- Tipovi ugnežđeni u interfejse su uvek (podrazumevano) javni
- Klasa koja proširuje ugnežđenu klasu ne nasleđuje prava pristupa

Statički ugnezđeni tipovi i unutrašnje klase (1)

- Kao i drugi članovi klase, ugnezđena klasa se može deklarisati kao static
 - ona se naziva statičkom ugnezđenom klasom (*static nested class*)
- Ugnezđena klasa bez modifikatora static (nestatička ugnezđena klasa)
 - se naziva unutrašnjom klasom (*inner class*)
- Primer:

```
class ObuhvatajućaKlasa{ . . .
    static class StatičkaUgnežđenaKlasa { . . . }
        class UnutrašnjaKlasa { . . . }
    }
```
- Statički ugnezđeni tipovi
 - služe samo kao mehanizam strukturiranja tipova
- Unutrašnje klase
 - omogućavaju poseban odnos između njihovih objekata i objekata spoljašnje klase

Statički ugnezđeni tipovi i unutrašnje klase (2)

- Statička ugnezđena klasa
 - ne može direktno (imenovanjem bez kvalifikacije) da pristupa nestatičkim poljima ili metodima obuhvatajuće klase
 - može preko reference na objekat
- Unutrašnja klasa ima direktni pristup nestatičkim članovima spoljašnje
- Klasa ugnezđena u interfejs je uvek statička,
 - specifikator `static` nije potreban
- Ugnezđeni interfejs je uvek statički,
 - specifikator `static` nije potreban
- Unutrašnja klasa ne može da sadrži statičke članove
 - izuzev konačnih statičkih polja inicijalizovanih konstantnim izrazom
- Objekat unutrašnje klase je uvek u vezi sa jednim objektom spoljašnje
- Više objekata unutrašnje može biti u vezi sa istim objektom spoljašnje

Primer unutrašnje klase

```
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("prenos na", iznos);  
        drugi.poslednjaAkcija=drugi.new Akcija("prenos sa", iznos);  
    }  
}
```

Relacije objekata

- Termin "ugnežđena" reflektuje sintaksnu relaciju između dve klase
 - kod jedne klase se pojavljuje unutar koda druge klase
- Termin "unutrašnja" reflektuje relaciju između objekata ugnežđene i obuhvatajuće klase
 - objekat unutrašnje može postojati samo u vezi sa objektom obuhvatajuće klase
 - relacija se naziva sekundarni `this`
 - primer: navođenje polja `broj` u metodi `toString()`
 - puna kvalifikacija bi bila: `RačunUbanci.this.broj`
- Definicija unutrašnje klase: unutrašnja klasa je ugnežđena klasa čiji objekat
 - ima implicitnu referencu na odgovarajući objekat spoljašnje klase (sek. `this`) i
 - ima direktni pristup (bez kvalifikacije) nestatičkim članovima obuhvatajuće klase
- Relacija između objekta unutrašnje i obuhvatajuće klase se uspostavlja prilikom kreiranja objekta unutrašnje klase:

```
this.new Akcija( "prenos" , iznos ); // može i bez eksplisitnog this
```

 - prosleđuje se referenca na objekat spoljašnje novom objektu unutrašnje klase

Proširivanje unutrašnjih klasa (1)

- Unutrašnje klase se mogu proširivati kao i statičke ugnezđene ili obične klase
- Objekat izvedene unutrašnje klase mora da je u vezi sa objektom
 - originalne spoljašnje klase (klase u okviru koje je definisana osnovna unutrašnja) ili
 - izvedene klase iz originalne spoljašnje klase
- Česta situacija: da se proširena unutrašnja klasa nalazi unutar proširene spoljašnje

```
class Spoljna { class Unutrasnja{} }  
class IzvedenaSpoljna extends Spoljna {  
    class IzvedenaUnutrasnja extends Unutrasnja {}  
    Unutrasnja u = new IzvedenaUnutrasnja();  
}
```
- Referenca `u` se inicijalizuje kada se kreira objekat `IzvedenaSpoljna`
- Pri konstrukciji objekta `IzvedenaUnutrasnja`
 - poziva se podrazumevani konstruktor:
`IzvedenaUnutrasnja() {IzvedenaSpoljna.this.super();}`
 - `super()` poziva konstruktor `Unutrasnja` koji zahteva ref. na objekat tipa `Spoljna`
 - `IzvedenaSpoljna.this` pokazuje na objekat tipa `IzvedenaSpoljna`
 - `IzvedenaSpoljna` je kompatibilan tip sa `Spoljna`

Proširivanje unutrašnjih klasa (2)

- Eksplisitna referenca na objekat spoljne klase je potrebna ako:
 - potklasa unutrašnje klase nije i sama unutrašnja klasa, već je samostalna ili
 - obuhvatajuća klasa unutrašnje potklase nije potklasa spoljne klase
- Primer:

```
class Samostalna extends Spoljna.Unutrasnja {  
    Samostalna(Spoljna s){s.super();}  
}
```

 - `s.super()` poziva konstruktor `Unutrasnja` i prosleđuje mu referencu na objekat klase `Spoljna`
 - referenca će inicijalizovati sekundarni `this` podobjekta tipa `Unutrasnja` u okviru stvorenog objekta tipa `Samostalna`

Lokalne unutrašnje klase

- Lokalna klasa se definiše unutar proizvoljnog programskog bloka
 - unutar metoda,
 - unutar konstruktora ili
 - unutar inicijalizacionog bloka
- Lokalne klase nisu članovi okružujuće klase
 - ali objekat lokalne klasne ima sekundarni `this` na objekat spoljašnje klase
- Lokalne klase su nepristupačne izvan bloka u kojem su definisane
- Reference na objekte lokalne klase se mogu:
 - prenositi kao argumenti ili
 - vraćati kao rezultati metoda
- Iz okružujućeg bloka, lokalna unutrašnja klasa ima pristup samo:
 - `final` lokalnim promenljivama ili
 - `final` argumentima metoda

Primer 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 objekat tipa `Iter`, jer je to podtip tipa `Iterator`,
a u izrazu odakle se poziva metoda očekuje se izraz tipa `Iterator`

Anonimne lokalne 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 nadtipa (osnovne klase ili interfejsa) se koristi za anonimnu klasu
- Anonimna klasa ne može da ima eksplisitne 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(){return p<objekti.length;}  
        public Object next() throws NoSuchElementException {  
            if (p>=objekti.length) throw new NoSuchElementException();  
            return objekti[p++];  
        }  
    }  
}
```

Konstruktor anonimne klase

- Anonimna klasa ne može imati konstruktor, jer konstruktor nosi ime klase koje ne postoji
- Ako je potreban poziv specifičnog konstruktora natklase, iza imena apstraktne klase se dodaju argumenti:
- Primer:
 - neka postoji klasa `Atribut` sa konstruktorom koji ima argument tipa `String`
 - piše se anonimna klasa koja se izvodi iz klase `Atribut`

```
Atribut ime = new Atribut( "Ime" ) { /*...*/ }
```
 - poziva se `super("Ime")`, odnosno konstruktor klase `Atribut` sa argumentom tipa `String`
- Kada anonimna klasa implementira ineterfejs, onda se kao super-konstruktor poziva samo konstruktor klase `Object`

Lambda izrazi

- Jednostavan način rada sa metodima anonimnih objekata anonimnih lokalnih klasa koje implementiraju interfejse sa samo 1 metodom
- Sličnost sa globalnom funkcijom
 - nema (eksplicitno) definisanu klasu kojoj pripada
- Razlike u odnosu na globalnu funkciju
 - postoji automatski genrisana klasa “omotača” funkcije i njen objekat
 - funkcija, odnosno njena klasa, je lokalna
- Doprinose konciznoj notaciji i eleganciji programa
- Često se metod svede na izračunavanje izraza – odatle naziv “izraz”
 - naziv “izraz” ne odražava prirodu, u pitanju je funkcija, a ne izraz
- Na mestu definisanja – ne izračunava se izraz
 - stvara se objekat anonimne klase koja implementira interfejs sa 1 metodom

Definisanje lambda izraza

- Opšti oblik definicije lambda izraza:
 $(\text{parametri}) \rightarrow \{\text{telo}\}$
 - parametri – uobičajena lista tipova i imena
 - tipovi mogu da se izostave, ako se o njima može zaključiti iz konteksta
 - moraju se ili navesti svi tipovi ili izostaviti svi tipovi
 - ako u listi postoji samo 1 parametar bez tipa – zagrade () mogu da se izostave
 - zagrade () su obavezne ako nema parametara
 - telo – uobičajeno telo funkcije (blok koji koristi parametre)
 - ako je samo jedna naredba (return), mogu da se izostave zagrade {} i reč return
 - tada se telo svodi na izraz koji izračunava vrednost lambda izraza
 - tip lambda izraza – ne navodi se
 - o njemu se zaključuje na osnovu tipa izraza u return naredbi
 - tip može biti i void

Primeri definicija lambda izraza

```
(int k) -> {return k*k; }  
k -> k * k; // skraćeni oblik
```

```
(double a, double b) -> {return a+b; }  
(a,b)-> a+b;
```

```
(int x, y) -> x+y // ! Greška
```

```
n -> {int s=0; for(int i=0; i<n; s+=i++); return s; }
```

Ciljni tip i cilj lambda izraza

- Cilj (*target*) lambda izraza – referenca na anonimni lambda objekat
- Lambda objekat – objekat anonimne klase lambda izraza
- Lambda klasa – automatski definisana, ostvaruje funkcijski interfejs
- Funkcijski interfejs – interfejs sa 1 apstraktnom metodom
 - može imati anotaciju: @FunctionalInterface
- Funkcijski interfejs predstavlja tip cilja lambda izraza (ciljni tip)
 - treba ga razlikovati od tipa rezultata lambda izraza
- Zaključivanje o parametrima lambda izraza i tipu rezultata
 - na osnovu parametara i tipa rezultata metoda ciljnog tipa
 - tip rezultata lambda izraza mora da bude kompatibilan sa tipom metoda

Definisanje cilja lambda izraza

- U definiciji cilja (reference na lambda objekat) koristi se ciljni tip
- Primer:

```
@FunctionalInterface  
interface FI{ int fi(int x); }  
interface FD{ double fd(double x); }  
class A{  
    void m( ) {  
        FI ciljFi=x->x*x; // int fi(int x){return x*x; }  
        ...  
    }  
}
```

Izračunavanje lambda izraza

- U naredbi u kojoj se izračunava lambda izraz
 - funkciji se pristupa preko cilja (reference) i imena metoda interfejsa
- Primer (nastavak prethodnog):

```
...
int i=ciljFi.fi(3);
FD ciljFd = x -> Math.exp(-0.1*x)*Math.sin(x);
double d=ciljFd.fd(2.0);
ciljFd = x -> x * x;
d=ciljFd.fd(2.0);
ciljFd = (int x) -> x * x; // ! Greška
}
```