

Objektno orijentisano programiranje 2

Interfejsi



Uvod

- Interfejsi su način da se deklariraju tip obrazovan pretežno od
 - apstraktnih metoda i
 - konstanti
- Osnovna sličnost između klasa i interfejsa:
 - oboje definišu tipove
- Osnovna konceptualna razlika između klasa i interfejsa:
 - interfejs je stvar ugovora (do verzije Java 8 – čistog ugovora)
 - klasa je kombinacija ugovora i implementacije
- Klasa može da implementira interfejs i tada mora:
 - da implementira sve njegove apstraktne metode ili
 - da bude deklarirana kao apstraktna
- Java podržava
 - jednostruko nasleđivanje implementacije i
 - višestruko nasleđivanje interfejsa

Kada koristiti interfejse?

- Ponekad se pojavljuje dilema između apstraktne klase i interfejsa
- Važne razlike između interfejsa i apstraktnih klasa:
 - apstraktna klasa uglavnom ima delimičnu implementaciju, a interfejs uglavnom nema (izuzetno – podrazumevani i stat. metodi)
 - može se implementirati više interfejsa, a proširiti samo jedna apstraktna klasa
 - apstraktna klasa može imati privatne i zaštićene članove, dok su interfejsi ograničeni na javne članove
 - apstraktna klasa može imati konstruktore i inicijalizacione blokove, interfejs ne
- Preporuka: važne klase treba da budu implementacije interfejsa
 - ovo pogotovo važi ako se očekuje da će te klase biti proširivane
 - na ovaj način se čuva i poslovna tajna implementacije klasa
 - klijentu se stavljaju na raspolaganje samo interfejsi

Primer

- Definicija interfejsa

- sve klase koje treba da realizuju bafer treba da implementiraju interfejs `IBafer`:

```
interface IBafer{
    void isprazni();
    void stavi(Object o);
    Object uzmi();
}
```

- Implementiranje interfejsa

```
abstract class Bafer implements IBafer {
    protected int p=0;
    protected Object[] niz;
    Bafer(int velicina){niz = new Object[velicina];}
    public void isprazni(){ p=0;
        if(niz!=null)for(int i=0;i<niz.lenght;i++)niz[i]=null;}
    abstract public void stavi(Object o);
    abstract public Object uzmi();
}
```

Proširivanje implementacije interfejsa

```
// date implementacije ne vode racuna o prekoracenju
// ne vode racuna ni o postavljanju null reference pri uzimanju
class LIFObafer extends Bafer {
    LIFObafer(int velicina){super(velicina);}
    public void stavi(Object o) {niz[p++]=o;}
    public Object uzmi(){return niz[--p];}
}
class FIFObafer extends Bafer {
    private int k=0; // indeks poslednje stavljene
    private int max;
    FIFObafer(int velicina){super(velicina); max=velicina;}
    public void isprazni(){super.isprazni(); k=0;}
    public void stavi(Object o) {niz[k]=o; k=(k+1)%max;}
    public Object uzmi(){Object t=niz[p]; p=(p+1)%max; return t;}
}
```

Korišćenje implementacije interfejsa

- Referenca na objekat koji implementira interfejs može se dodeliti referenci tipa tog interfejsa

```
class FIFO_LIFO {
    public static void test(IBafer b){
// public static void test(Bafer b){ // ekvivalentno
        for (int i = 0; i<10; i++) {b.stavi(new Integer(i));}
        for (int i = 0; i<10; i++)
            System.out.print((Integer) b.uzmi()+" ");
        System.out.println();
    }
    public static void main(String [] arg){
        FIFObafer fb= new FIFObafer(10);
        LIFObafer lb= new LIFObafer(10);
        test(fb); test(lb);
    }
}
```

Metodi i polja interfejsa

- Metodi interfejsa su implicitno apstraktni i javni
 - od Java 8, mogu biti i konkretni (podrazumevani ili statički)
- Polja interfejsa su uvek javna statička i konačna (`public static final`)
 - ona su način da se definišu konstante koje se koriste pri pozivu metoda
- Primer:
 - interfejs koji ugovorom definiše srednju temperaturu za različita godišnja doba:

```
interface SezonskiTermometar{
    int PROLECE=0; int LETO=1; int JESEN=2; int ZIMA=3;
    float srednjaTemperatura(int doba);
}
```
- Imenovane konstante se mogu proslediti metodi `srednjaTemperatura(int doba)`
- Nedostatak je što vrednosti argumenta mogu biti proizvoljne celobrojne
- Od verzije Java 5.0 uvedeni tipovi nabiranja (`enum`)

Podrazumevani metodi

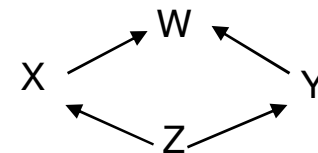
- Interfejs može sadržati konkretne podrazumevane metode
- Podrazumevani metodi – modifikator `default`
- Svrha – da se izbegne modifikacija postojećih aplikacija kada je potrebno bibliotečkom interfejsu dodati nove metode
- Objekat tipa interfejsa ne može da se kreira čak ni kad bi interfejs sadržao samo konkretne (podrazumevane) metode
- Postoje problemi sa konfliktima imena podrazumevanih metoda

Statički metodi

- Interfejs može sadržati i statičke metode
- Modifikator `static`
- Svrha – čiste funkcije – bez bočnih efekata, kao matematičke
- U klasi koja implementira interfejs
 - statički metod sa istim potpisom sakriva metod interfejsa
 - statički metod interfejsa se mora pozivati sa `Interfejs.metod()`
 - to ne važi za imenovane konstante – dozvoljeno direktno imenovanje ukoliko se ne implementira više interfejsa sa istoimenim konstantama
 - ne važi za izvedene klase – imenovanje statičkog m. natklase – direktno
- Nije greška ako u interfejsima koji se implementiraju i natklasi postoje statički metodi sa istim potpisom

Jednostruko/višestruko nasleđivanje

- Nasleđivanje u OO programskim jezicima može biti:
 - jednostruko – klasa neposredno nasleđuje samo jednu klasu
 - višestruko – klasa može imati dve ili više neposrednih roditeljskih klasa
- Proširivanjem klasa nasleđuje ugovor i implementaciju natklase
- Problem je nasleđivanje implementacije (u “dijamant” strukturi):
 - ako W čuva stanje (polje f)
 - šta će značiti $z.f$ (ako je: $Z z;$)?
 - da li postoje dve kopije ili samo jedna?
- Java dozvoljava samo jednostruko nasleđivanje za klase da izbegne ovaj problem



Višestruko nasleđivanje u Javi

- Sprečavanje višestrukog nasleđivanja
 - sprečava i neke korisne dizajne
- Problemi višestrukog nasleđivanja potiču od višestrukog nasleđivanja implementacije
- Java nudi način da se nasledi samo ugovor
- Interfejsi omogućavaju višestruko nasleđivanje u Javi
- Supertipovi neke klase su:
 - klase koje su proširene (u jednom koraku izvođenja – smo jedna) i
 - interfejsi koji su implementirani

Proširivanje interfejsa

- Interfejsi se mogu proširivati
- Jedan interfejs može da proširuje više drugih:

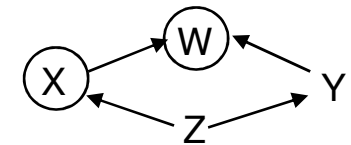
```
interface I extends BI1, BI2{  
    double i_method();  
}
```

- Sve metode i konstante osnovnih interfejsa su deo novog interfejsa
- Višestruko nasleđivanje kod klasa:
 - klasa koja proširuje jednu klasu i implementira bar jedan interfejs
 - klasa koja implementira više interfejsa

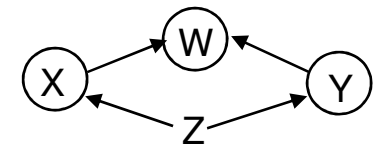
Primeri

- Primeri višestrukog nasleđivanja u romboidnoj (dijamant) strukturi:

```
interface W{}  
interface X extends W{}  
class Y implements W{}  
class Z extends Y implements X{}
```



```
interface W{}  
interface X extends W{}  
interface Y extends W{}  
class Z implements X,Y{}
```



Relacija sa klasom Object

- Interfejsi nemaju neki jedinstveni koreni interfejs (kao što klase imaju klasu `Object`)
- Klasa `Object` jeste nadtip i svim interfejsima
 - reference na objekte klase koje implementiraju neki interfejs mogu se dodeliti referencama na tip `Object`
- Primeri:
 - parametar tipa interfejsa `I` se može dodeliti referenci tipa `Object`

```
void a(I iRef){ Object obj = iRef; /*...*/}
```
 - argument tipa interfejsa `I` se može proslediti metodi sa parametrom tipa `Object`

```
void b(Object o){ /*...*/ }  
/*...*/  
I iRef; /*...*/ x.b(iRef);
```

Konflikti imena metoda

- Šta se događa kada se apstraktan metod istog imena pojavljuje u dva ili više implementiranih interfejsa?
 - metodi imaju različit potpis (broj i/ili tipove parametara):
=> dva ili više metoda preklopljenih imena
 - metodi imaju tačno jednake potpise i povratne tipove
=> jedan metod sa istim potpisom i povratnim tipom
 - metodi se razlikuju jedino u povratnom tipu
=> **ne mogu se implementirati oba interfejsa**
 - metodi se razlikuju samo u `throws` klauzuli
=> jedan metod koji zadovoljava sve `throws` klauzule

Različite throws klauzule

- Metod može da baca manje tipova izuzetaka nego što je deklarirano u odgovarajućoj metodi nadtipa

- Primer:

```
interface X { void m() throws Greska; }
interface Y { void m(); }
class Z implements X,Y {
    public void m() { /*...*/ }
}
```

- Z implementira metod `m()` tako da ne baca izuzetke, što zadovoljava i `X.m()` i `Y.m()`
 - uža lista izuzetaka u odnosu na `X.m()`

Konflikti podrazumevanih metoda

- Ako se podrazumevani metod nasleđuje po više linija izvođenja (npr. u dijamant strukturi) – nije problem
 - postoji samo jedna implementacija metoda
- Problem nastupa kod implementacije više interfejsa ako njihovi podrazumevani metodi imaju isti potpis
 - nije greška ako se u klasi implementira takav metod
 - greška je ako se podrazumevani metodi samo nasleđuju
- Ako u natklasi postoji konkretan nestatički metod sa potpisom istim kao potpis podrazumevanog metoda implementiranog interfejsa
 - nasleđuje se metod natklase

Konflikti imena konstanti

- Dva implementirana interfejsa mogu da imaju istoimene konstante
 - za takve konstante treba koristiti imena sa kvalifikatorima
- Primer:
 - interfejsi `PreferansSpil` i `RemiSpil` sadrže konstante `VELICINA` različitih vrednosti
 - klasa `MultiSpil` implementira oba interfejsa
 - u klasi `MultiSpil` moraju se koristiti imena `PreferansSpil.VELICINA`, odnosno `RemiSpil.VELICINA`
 - samo `VELICINA` je dvosmisleno
- Ako se imenovana konstanta nasleđuje po više linija izvođenja
 - ne postoji problem – to je ista konstanta
- Isto važi za podrazumevane i statičke metode
 - ako se nasleđuju po više linija izvođenja – to je isti metod

enum tip umesto konstanti

- Problem sa imenovanim konstantama – nisu bezbedne
 - provera tipa ne pomaže, jer se može pojaviti vrednost van opsega
 - na primer, kao stvarni argument metoda `srednjaTemperatura(int)` nema bezbednog načina da se `int` ograniči na vrednosti 0-3, koje se očekuju
- Problem se može prevazići pomoću `enum` tipa (uvedenog u Java 5.0)

```
interface SezonskiTermometar{  
    enum GodisnjeDoba{PROLECE, LETO, JESEN, ZIMA}  
    float srednjaTemperatura(GodisnjeDoba doba);  
}
```
- Tip `enum` je klasni tip
 - pored konstanti može imati konstruktore i metode
 - objekti tipa nabrajanja se alociraju na hipu
 - konstante tipa nabrajanja se ponašaju kao reference na objekte tog tipa

Konstruktori i metodi nabiranja

- Argumenti koji se navode uz konstantu su argumenti konstruktora
- Konstruktor se izvršava za konstantu prilikom učitavanja klase

```
enum Nabiranje{
    PRVI(1), DRUGI(2), TRECI(3);
    int vrednost;
    Nabiranje(int i){vrednost=i;}
    int vrednost(){return vrednost;}
}
class ProbaEnum{
    public static void main(String[] a){
        System.out.println(Nabiranje.PRVI+"="
            +Nabiranje.PRVI.vrednost());
    }
}
```