

Interfejsi i nabrojivi tipovi

Objektno-orjentisano programiranje 1



Interfejsi

- **Interfejsi su specifikacije funkcionalnosti klasa**
- **Interfejsi su ugovori kojima se definišu fundamentalne interakcije između klasa**
 - Klasa koja implementira interfejs se obavezuje da implementira sve funkcionalnosti specificirane interfejsom
 - Klasu koja implementira neki interfejs neka druga klasa koristi na način specificiran interfejsom
- **Interfejsi su kao i klase referencijalni tipovi**
 - Ako je **I** interfejs, a **r** promenljiva tipa **I**, tada je **r** referenca na objekat neke (bilo koje) klase koja implementira **I**
- **U telu interfejsa možemo specificirati**
 - konstante
 - **zaglavlja metoda** **[najčešće samo ovo]**
 - default metode i statičke metode
 - ugnježdene (statičke) tipove (klase, interfejse i enum-e)

● Sve deklaracije u interfejsu su implicitno *public*

- Deklaracije promenljivih u interfejsu su implicitno *static* i *final*
- Deklaracije zaglavlja metoda u interfejsu su implicitno *abstract*
- Deklaracije ugnježenih klasa i interfejsa su implicitno *static*
- Metode deklarisanе u interfejsu mogu biti ili *static* ili *default*

```
public interface Foo {  
    void nekaMetoda();  
    // public abstract void nekaMetoda();  
  
    int x = 5;  
    // public static final int x = 5;  
  
    class Bar { ... }  
    // public static class Bar { ... }  
  
    interface Baz { ... }  
    // public static interface Baz { ... }  
  
    default boolean m1() { return true; }  
  
    static boolean m2() { return false; }  
}
```

- **Klasa može implementirati više od jednog interfejsa**

```
public interface Prosiriv {  
    void prosiri(double k);  
}
```

```
public interface Translabilan {  
    void pomeri(double dx, double dy);  
}
```

```
public class Krug implements Prosiriv, Translabilan {  
    private double x, y, r;  
  
    public Krug(double x, double y, double r) {  
        this.x = x; this.y = y; this.r = r;  
    }  
  
    public void pomeri(double dx, double dy) {  
        x += dx; y += dy;  
    }  
  
    public void prosiri(double k) {  
        r *= k;  
    }  
}
```

- **Interfejsi su referencijalni tipovi promenljivih**

```
Krug k = new Krug(2, 4, 5);  
k.prosiri(2);  
k.pomeri(1, 1);
```

```
Prosiriv p = new Krug(1, 2, 4);  
p.prosiri(3);  
// p.pomeri(1, 1); -- ne kompajlira se
```

```
Translabilan t = new Krug(3, 4, 1);  
t.pomeri(2, 3);  
// t.prosiri(3); -- ne kompajlira se
```

```
void transliraj(Translabilan[] nizFig, int dx, int dy) {  
    for (int i = 0; i < nizFig.length; i++)  
        nizFig[i].pomeri(dx, dy);  
}
```

Default metode

- Klasa koja implementira interfejs nasleđuje *default* metode implementirane u interfejsu i može da ih redefiniše

```
public interface Prosiriv {  
    void prosiri(double k);  
  
    default void smanji(double k) {  
        if (k <= 0)  
            throw new IllegalArgumentException("k <= 0");  
  
        prosiri(1 / k);  
    }  
}
```

```
public class Krug implements Prosiriv, Translabilan { ... }
```

```
Krug k = new Krug(2, 4, 10);  
k.smanji(4);  
// sada je krug poluprecnika 2.5
```

Kolizije

- **Kolizije na nivou imena mogu nastati kada klasa implementira dva ili više interfejsa**
 - Na nivou zaglavlja metoda nemamo kolizije
 - Duplicirano zaglavlje koje tek treba implementirati
 - **Kolizije mogu nastati na nivou *default* metoda**
 - **Različite implementacije za ista zaglavlja**
 - Na nivou statičkih elemenata interfejsa kolizije se trivijalno razrešavaju upotrebom punih (kvalifikovanih) imena
- **Ako klasa A implementira interfejse X i Y koji oba sadrže *default* metodu istog zaglavlja tada klasa A mora redefinisati tu *default* metodu**
 - **Inače ne prolazi kompajliranje**

Primer kolizija

```
public interface Bar {  
    void m();  
    int x = 5;  
    default void d() {  
        S.o.p("Bar");  
    }  
}
```

```
public interface Baz {  
    void m();  
    String x = "Pet";  
    default void d() {  
        S.o.p("Baz");  
    }  
}
```

```
public class BarBaz implements Bar, Baz {  
    public void m() {  
        S.o.p(Bar.x);  
        S.o.p(Baz.x);  
    }  
  
    // klasa BarBaz mora redefinisati d  
    public void d() {  
        S.o.p("BarBaz");  
    }  
}
```


Nasleđivanje interfejsa

- Nasleđivanje interfejsa može biti jednostruko i višestruko

```
public interface Knjiga {  
    String ime();  
    int brojStrana();  
}
```

```
public interface Udzbenik extends Knjiga {  
    String oblast();  
}
```

```
public class InformatickiUdzbenik implements Udzbenik {  
    private String ime;  
    private int brStr;  
  
    public InformatickiUdzbenik(String ime, int brStr) {  
        this.ime = ime;  
        this.brStr = brStr;  
    }  
  
    public String ime() { return ime; }  
    public int brojStrana() { return brStr; }  
    public String oblast() { return "informatika"; }  
}
```

Višestruko nasleđivanje interfejsa

```
public interface NaucniRadnik {
    String opisPosla = "stvaranje znanja";
    int radniStaz();
    int brojNapisanihRadova();
}

public interface ProsvetniRadnik {
    String opisPosla = "prenosenje znanja";
    int radniStaz();
    String vrstaObrazovneUstanove();
}

public interface ProfUniverziteta extends NaucniRadnik, ProsvetniRadnik {
    String imeFakulteta();
}

class PMFProf implements ProfUniverziteta {
    ...
    public int radniStaz() { ... }
    public int brojNapisanihRadova() { ... }
    public String vrstaObrazovneUstanove() { ... }
    public String imeFakulteta() { return "PMF"; }
}
```

```
public interface NaucniRadnik {  
    String opisPosla = "stvaranje znanja";  
    int radniStaz();  
    int brojNapisanihRadova();  
}
```

```
public interface ProsvetniRadnik {  
    String opisPosla = "prenosenje znanja";  
    int radniStaz();  
    String vrstaObrazovneUstanove();  
}
```

```
public interface ProfUniverziteta extends NaucniRadnik, ProsvetniRadnik {  
    String imeFakulteta();  
}
```

```
class PMFProf implements ProfUniverziteta { ... }
```

```
PMFProf prof = new PMFProf(...);
```

```
NaucniRadnik np = prof;  
System.out.println(np.opisPosla);
```

```
ProsvetniRadnik pp = prof;  
System.out.println(pp.opisPosla);
```

Ugnježdjeni tipovi

- Unutar interfejsa je moguće definisati ugnježdenu statičku klasu, interfejs ili *enum* (nabrojivi tip)
- Unutar klase je takođe moguće definisati interfejs
 - **Interfejsi ugnježdjeni u klasu su implicitno statički**

```
public interface Knjiga {  
    String naslov();  
    String autor();  
    void dodajUSadrzaj(String naslov, int strana);  
    IspisivacSadrzaja sadrzaj();  
  
    interface IspisivacSadrzaja {  
        boolean imaJos();  
        String sledeciNaslovIStrana();  
    }  
}
```

```
public class SadrzajKnjige implements Knjiga.IspisivacSadrzaja {  
    // elementi sadrzaja  
    private ArrayList<String> elementi = new ArrayList<>();  
  
    // indeks elementa koga treba ispisati  
    private int pozicija = 0;  
  
    public void dodaj(String naslov, int strana) {  
        String novi = naslov + "--" + strana;  
        elementi.add(novi);  
    }  
  
    public boolean imaJos() {  
        return pozicija < elementi.size();  
    }  
  
    public String sledeciNaslovIStrana() {  
        if (imaJos())  
            return elementi.get(pozicija++);  
        else  
            return null;  
    }  
}
```

```
public class Udzbenik implements Knjiga {  
    private String ime;  
    private String autor;  
    private SadrzajKnjige sadrzaj;  
  
    public Udzbenik(String ime, String autor) {  
        this.ime = ime;  
        this.autor = autor;  
        sadrzaj = new SadrzajKnjige();  
    }  
  
    public String naslov() { return ime; }  
    public String autor() { return autor; }  
  
    public Knjiga.IspisivacSadrzaja sadrzaj() {  
        return sadrzaj;  
    }  
  
    public void dodajUSadrzaj(String naslov, int strana) {  
        sadrzaj.dodaj(naslov, strana);  
    }  
}
```

```

class RadnaJedinica {

    // ugnjezdena nestaticka klasa
    public class Radnik {
        private String ime;
        private int plata;

        public Radnik(String ime, int plata) {
            this.ime = ime;
            this.plata = plata;
        }

        public String getIme() { return ime; }
        public int getPlata() { return plata; }
    }

    // ugnjezdeni, staticki interfejs
    public interface PoredjenjeRadnika {
        int uporedi(Radnik r1, Radnik r2);
    }

    private Radnik[] radnici = null;
    private int trenutnoZaposlenih = 0;

    public RadnaJedinica(int maxZaposlenih) {
        radnici = new Radnik[maxZaposlenih];
    }

    public void zaposli(String ime, int plata) {
        if (trenutnoZaposlenih < radnici.length) {
            Radnik r = new Radnik(ime, plata);
            radnici[trenutnoZaposlenih++] = r;
        }
    }

    public void informacijeOZaposlenima(PoredjenjeRadnika komparator) {

```

```
class RadnaJedinica {
```

```
// ugnjezdena nestaticka klasa
```

```
public class Radnik {..
```

```
// ugnjezdeni, staticki interfejs
```

```
public interface PoredjenjeRadnika {  
    int uporedi(Radnik r1, Radnik r2);  
}
```

```
private Radnik[] radnici = null;  
private int trenutnoZaposlenih = 0;
```

```
public RadnaJedinica(int maxZaposlenih) {..
```

```
public void zaposli(String ime, int plata) {..
```

```
public void informacijeOZaposlenima(PoredjenjeRadnika komparator) {  
    for (int j = trenutnoZaposlenih - 1; j > 0; j--) {  
        for (int i = 0; i < j; i++) {  
            if (komparator.uporedi(radnici[i], radnici[i + 1]) > 0) {  
                Radnik tmp = radnici[i];  
                radnici[i] = radnici[i + 1];  
                radnici[i + 1] = tmp;  
            }  
        }  
    }  
}
```

```
System.out.println("Ime, plata");  
for (int i = 0; i < trenutnoZaposlenih; i++) {  
    System.out.println(radnici[i].ime + ", " + radnici[i].plata);  
}  
}
```

Inversion of control: korisnik klase diktira kako će radnici biti sortirani definišući neko uređenje za radnike

Dinamičko vezivanje: U vremenu izvršavanja formalni parametar “komparator” će biti referenca na neku instancu klase koja implementira interfejs PoređenjeRadnika

Polimorfizam: različito ponašanje metode “informacijeOZaposlenima” u zavisnosti od tipa parametra “komparator” u vremenu izvršavanja. Posledica različitog ponašanja metode “uporedi”.


```
public class Knjigovodstvo {
    public static void main(String[] args) {
        RadnaJedinica rj = new RadnaJedinica(10);
        rj.zaposli("Zorana Vukmirovic", 60000);
        rj.zaposli("Veselin Stankovic", 50000);
        rj.zaposli("Milovan Petrovic", 75000);
        rj.zaposli("Aleksandar Markovic", 55000);

        // radnici sortirani leksikografski po imenima
        //   poziv metoda ciji je parametar instanca ANONIMNE klase koja implementira
        //   ugnjezdeni interfejs "RadnaJedinica.PoredjenjeRadnika"
        rj.informacijeOZaposlenima(
            new RadnaJedinica.PoredjenjeRadnika() {
                public int uporedi(RadnaJedinica.Radnik r1, RadnaJedinica.Radnik r2) {
                    return r1.getIme().compareToIgnoreCase(r2.getIme());
                }
            }
        );

        System.out.println();

        // radnici sortirani opadajuće po platama
        rj.informacijeOZaposlenima(
            new RadnaJedinica.PoredjenjeRadnika() {
                public int uporedi(RadnaJedinica.Radnik r1, RadnaJedinica.Radnik r2) {
                    return r2.getPlata() - r1.getPlata();
                }
            }
        );
    }
}
```

Nabrojivi tipovi (enum)

- Promenljiva nabrojivog tipa uzima neku vrednost iz skupa eksplicitno nabrojanih simboličkih konstanti

- Primeri

- Dan = {Pon, Uto, Sre, Čet, Pet, Sub, Ned}
- Mesec = {Jan, Feb, Mar, Maj, Jun, Jul, Avg, Sep, Okt, Nov, Dec}
- Planete = {Merkur, Venera, Zemlja, Mars, Jupiter, Saturn, Uran, Neptun, Pluton}

- Do Java 1.5 nabrojive tipove smo simulirali celobrojnim konstantama

```
public static final int PON = 1
public static final int UTO = 2
...
public static final int NED = 7;
...
int dan;
...
if (dan == SUB || dan == NED) System.out.println("Vikend");
```

- Java 1.5 uvodi novi referencijalni tip **enum** kao način definisanja nabrojivih tipova podataka

Nabrojivi tipovi (enum)

- Za definisanje nabrojivog tipa koristimo ključnu reč enum
- U definiciji nabrojivog tipa nabrajamo vrednosti tipa

```
public enum Dani {  
    PON, UTO, SRE, CET, PET, SUB, NED  
}
```

```
public enum Meseci {  
    JAN, FEB, UTO, SRE, CET, PET, SUB, NED  
}
```

```
public enum Planete {  
    MERKUR, VENERA, ZEMLJA, MARS, JUPITER, SATURN,  
    URAN, NEPTUN, PLUTON  
}
```

```
public enum StatusVozila {  
    SLOBODAN, ZAUZET, POKVAREN  
}
```

Nabrojivi tipovi (enum)

- Vrednosti nabrojivog tipa se navode kvalifikovano uz navođenje naziva nabrojivog tipa (**osim u switch naredbi**)

```
public enum StatusVozila {  
    SLOBODAN, ZAUZET, POKVAREN  
}  
  
public class Vozilo {  
    private StatusVozila status;  
  
    public Vozilo() {  
        status = StatusVozila.SLOBODAN;  
    }  
  
    public void napraviVoznju() {  
        if (status == StatusVozila.SLOBODAN) {  
            status = StatusVozila.ZAUZET;  
            vozi();  
            if (status != StatusVozila.POKVAREN)  
                status = StatusVozila.SLOBODAN;  
        }  
    }  
  
    private void vozi() { ... }  
}
```

- Vrednosti nabrojivog tipa se navode kvalifikovano uz navođenje naziva nabrojivog tipa (**osim u switch naredbi**)

```
public enum StatusVozila {  
    SLOBODAN, ZAUZET, POKVAREN  
}  
  
public class Vozilo {  
    private StatusVozila status;  
  
    public Vozilo() {  
        status = StatusVozila.SLOBODAN;  
    }  
  
    public void napraviVoznju() {  
        if (status == StatusVozila.SLOBODAN) {  
            status = StatusVozila.ZAUZET;  
            vozi();  
            if (status != StatusVozila.POKVAREN)  
                status = StatusVozila.SLOBODAN;  
        }  
    }  
  
    private void vozi() { ... }  
}
```

- Vrednosti nabrojivog tipa se navode kvalifikovano uz navođenje naziva nabrojivog tipa (**osim u switch naredbi**)

```
public enum StatusVozila {  
    SLOBODAN, ZAUZET, POKVAREN  
}
```

```
public class Vozilo {  
    private StatusVozila status;  
    ...  
  
    public void ispisiStatus() {  
        switch (status) {  
            case SLOBODAN:  
                System.out.println("Vozilo slobodno");  
                break;  
            case ZAUZET:  
                System.out.println("Vozilo zauzeto");  
                break;  
            default:  
                System.out.println("Vozilo pokvareno");  
        }  
    }  
}
```

Nabrojivi tipovi (enum)

- Enumi su specijalna vrsta klasa
- **Svaki enum implicitno nasleđuje klasu *java.lang.Enum*** koja nasleđuje *java.lang.Object*
 - Enumi ne mogu da nasleđuju druge tipove
 - Enumi mogu da implementiraju proizvoljan broj interfejsa
- **Svaka vrednost nabrojivog tipa je objekat koji može imati stanje i ponašanje**
 - U enimima pored obaveznog navođenja liste konstanti možemo definisati
 - Attribute (polja)
 - Metode
 - ***Non-public* konstruktore (eksplicitno se pozivaju prilikom navođenja konstanti)**
 - Unutrašnje tipove

```
public enum Planet {  
    MERCURY (3.303e+23, 2.4397e6), ← eksplicitan poziv  
    VENUS   (4.869e+24, 6.0518e6),   konstruktora  
    EARTH   (5.976e+24, 6.37814e6),  
    MARS    (6.421e+23, 3.3972e6),  
    JUPITER (1.9e+27,   7.1492e7),  
    SATURN  (5.688e+26, 6.0268e7),  
    URANUS  (8.686e+25, 2.5559e7),  
    NEPTUNE (1.024e+26, 2.4746e7);  
  
    private final double mass;    // in kilograms  
    private final double radius; // in meters  
  
    private Planet(double mass, double radius) {  
        this.mass = mass;  
        this.radius = radius;  
    }  
  
    // universal gravitational constant  
    public static final double G = 6.67300E-11;  
  
    public double surfaceGravity() {  
        return G * mass / (radius * radius);  
    }  
}
```


Nabrojivi tipovi (enum)

- Svaki enum ima implicitno definisan statički metod **values()** koji vraća niz koji sadrži sve konstante definisane enumom

```
public class Planete {  
    public static void main(String[] args) {  
        Planet[] p = Planet.values();  
        for (int i = 0; i < p.length; i++)  
            S.o.p(p[i] + ", " + p[i].surfaceGravity());  
    }  
}
```

MERCURY, 3.7030267229659395

VENUS, 8.871391908774457

EARTH, 9.802652743337129

MARS, 3.7126290961053403

JUPITER, 24.80617666947324

SATURN, 10.44978014597121

URANUS, 8.8726476241638

NEPTUNE, 11.158634802412358