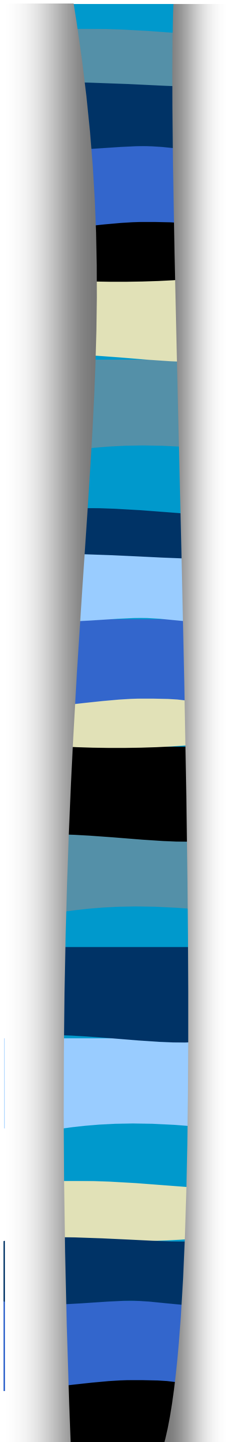


Interfejsi

Interfejsi

- Nasleđivanje klasa u Javi je jednostruko - osim klase **Object**, koja nema nadklasu, svaka klasa ima tačno jednu direktnu nadklasu.
- Da bi se ipak nekako iskoristile pogodnosti višestrukog nasleđivanja, u Javi se koriste interfejsi, pomoću kojih se "simulira" višestruko nasleđivanje.
- Interfejs je referencijalni tip podataka koji podseća na apstraktnu klasu u kojoj su svi metodi apstraktni.

Interfejsi

- Pri deklaraciji interfejsa se od modifikatora mogu koristiti samo modifikatori:
- **public** – deklarisanje javnih interfejsa – vidljivi i izvan svog paketa, interfejsi koji nisu javni mogu se koristiti samo u okviru svog paketa
- **abstract** – suvišan i njegovo navođenje/nenavođenje nema nikakvog efekta – koristi se zbog kompatibilnosti sa starijom verzijom Jave – ne preporučuje se pri deklaraciji interfejsa
- i **strictfp** **–** da bi se svi realni izrazi unutar interfejsa izračunavali striktno se pridržavajući standarda za brojeve sa pokretnom decimalnom tačkom

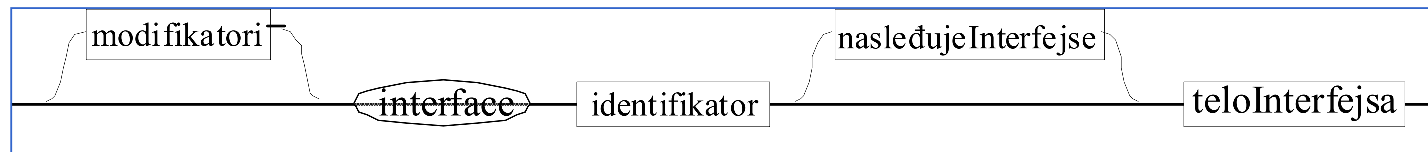
Interfejsi

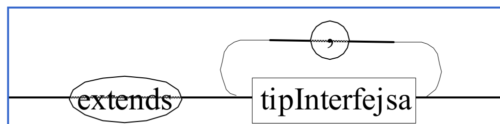
- Interfejsi mogu da se nasleđuju – jedan interfejs može naslediti više drugih interfejsa.
- Telo interfejsa se može sastojati od deklaracije apstraktnih metoda, statičkih ugnježdenih klasa i interfejsa, kao i od konstantnih polja.
- Metodi interfejsa mogu biti samo javni (**public**), apstraktni (**abstract**) metodi.

Interfejsi

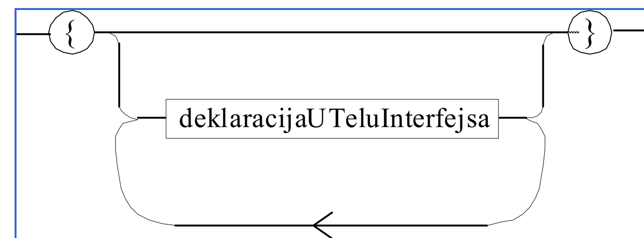
- Polja interfejsa mogu biti samo javna (**public**), statička (**static**), konstantna (**final**) polja.
- Kod deklarisanja polja interfejsa obavezno je navođenje i vrednosti polja.
- Ugnježdene klase i ugnježdeni interfejsi u interfejsu su implicitno deklarisani sa **public** i **static**, bez obzira na to da li su ovi modifikatori navedeni ili nisu.
- Osim ovih modifikatora, ugnježdena klasa u interfejsu može biti deklarisana i sa modifikatorima **abstract**, **final** i **strictfp**.
- Ugnježdeni interfejs osim modifikatora **public** i **static** može da bude deklarisan i sa modifikatorima **abstract** (iako je suvišno) i **strictfp**.

Deklaracija interfejsa





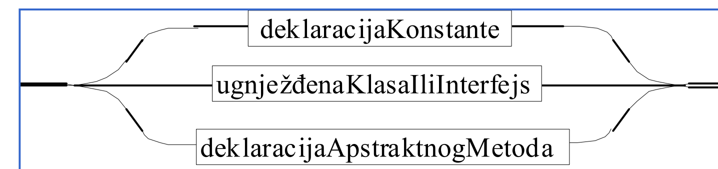
nasleđujeInterfejse



teloInterfejsa

deklaracijaKonstante

deklaracijaApstraktnogMetoda



deklaracijaUTeluInterfejsa

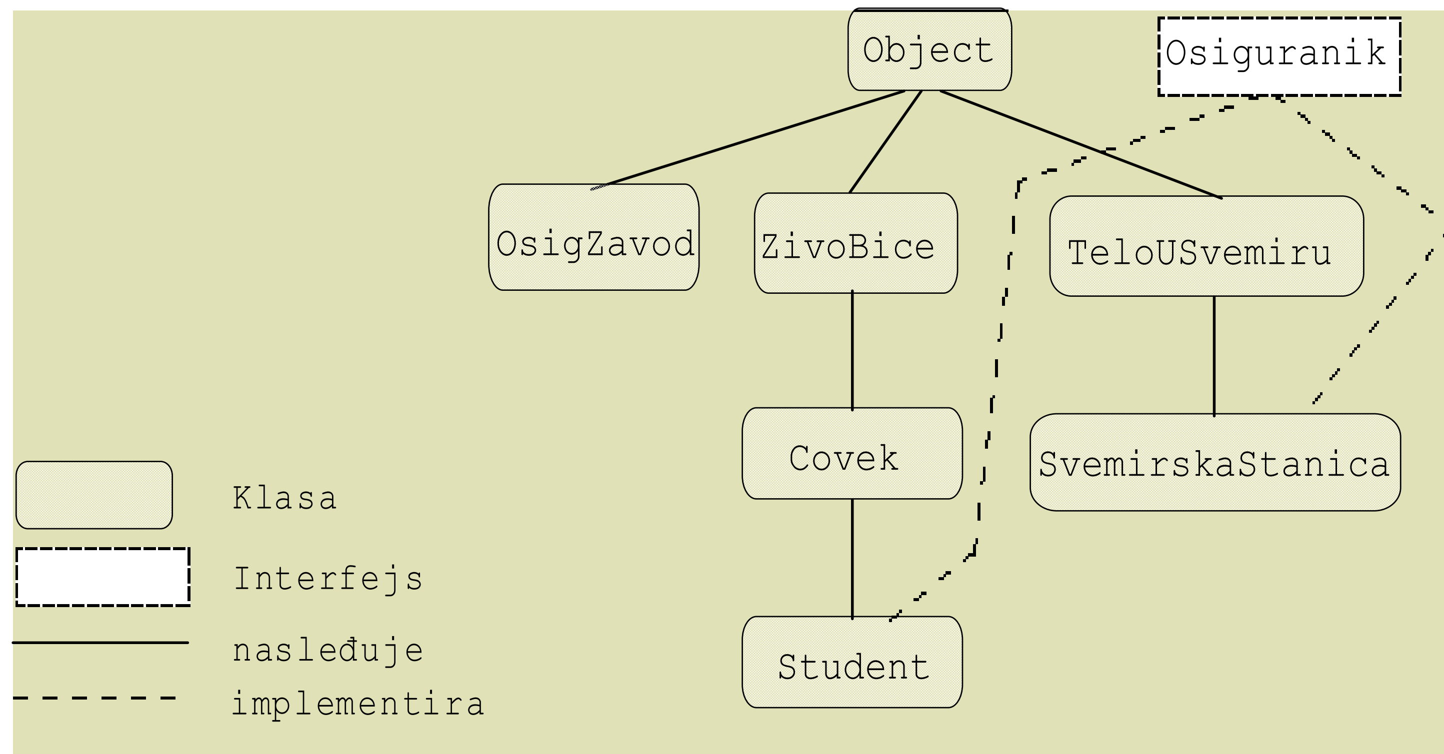
Korišćenje interfejsa (1/5)

- Napravljeni interfejs se ne može koristiti 'samostalno', već mora postojati bar jedna klasa koja ga implementira (što se specificira u zaglavlju te klase).
- Ako ta klasa nije apstraktna tada ona mora da sadrži i pune deklaracije svih metoda čija zaglavlja su navedena u deklaraciji interfejsa i ti metodi moraju biti deklarisani sa **public** modifikatorom.
- Jedna klasa može da implementira proizvoljan broj interfejsa.
- Ako neka klasa implementira interfejs onda ga automatski implementiraju i sve njene podklase.
- Promenljiva čiji je tip neki interfejs kao svoju vrednost može dobiti samo referencu objekta čija klasa implementira taj interfejs.

Primer interfejsa

```
interface Osiguranik {
  void upisiCenuOsiguranja(double cena);
  void upisiIznos(double iznos);
  double cenaOsiguranja();
  double iznos();
}
```

Korišćenje interfejsa (2/5)



Korišćenje interfejsa (3/5)

Klasa ŽivoBiće

```
public class ZivoBice {
  private String vrsta, grupa, dise, rasprostranjenost;
  private int brojNogu;

  public ZivoBice(String v, String g, int bn, String d, String r) {
    vrsta = v;
    grupa = g;
    dise = d;
    rasprostranjenost = r;
    brojNogu = bn;
  }

  public String opis() {
    return vrsta + ", " + grupa + " sa brojem nogu " + brojNogu +
           " dise " + dise + " nalazi se: " + rasprostranjenost;
  }
}
```

Korišćenje interfejsa (3/5)

```
public class Covek extends ZivoBice {
  private String ime;
  private String prezime;

  public Covek(String imeCoveka, String prezimeCoveka) {
    super("homosapiens", "sisar", 2, "vazduh", "svuda");
    ime = imeCoveka;
    prezime = prezimeCoveka;
  }

  public String zoveSe() {
    return ime + " " + prezime;
  }
}
```

Korišćenje interfejsa (3/5)

```
public class Student extends Covek implements
Osiguranik {
  private int brIndeksa;
  private String fakultet;
  private double cenaOsig;
  private double iznosOsig;

  public Student(String ime, String prezime, int
         brIndeksa,String fakultet, double cenaOsig,
         double iznosOsig) {
    super(ime, prezime);
    this.brIndeksa = brIndeksa;
    this.fakultet = fakultet;
    this.cenaOsig = cenaOsig;
    this.iznosOsig = iznosOsig;
  }

  public void upisiCenuOsiguranja(double cena) {
    cenaOsig = cena;
  }

  public double cenaOsiguranja() {
    return cenaOsig;
  }

  public double iznos() {
    return iznosOsig;
  }

  public void upisiIznos
            (double iznos) {
    iznosOsig = iznos;
  }

  public int brojIndeksa() {
    return brIndeksa;
  }

  public String imeFakulteta() {
    return fakultet;
  }
}
```

Korišćenje interfejsa (4/5)

Klasa TeloUSvemiru

```
public class TeloUSvemiru {
  private double tezina, visina, brzina;

  public TeloUSvemiru(double tezina, double visina, double brzina) {
    this.tezina = tezina;
    this.visina = visina;
    this.brzina = brzina;
  }

  //i razni metodi u kojima se vrse sva ona komplikovana izracunavanja
}
```

Korišćenje interfejsa (4/5)

```
public class SvemirskaStanica extends TeloUSvemiru implements Osiguranik {
  private double cOsiguranja, izn;
  private int rokTrajanja;

  public SvemirskaStanica(int rokTrajanja, double osig, double izn,
                          double tezina, double visina, double brzina) {
    super(tezina, visina, brzina);
    this.rokTrajanja = rokTrajanja;
    cOsiguranja = osig;
    this.izn = izn;
  }
  public double cenaOsiguranja() {
    return cOsiguranja;
  }
  public double iznos() {
    return izn;
  }
  public void upisiCenuOsiguranja(double cena) {
    cOsiguranja = cena;
  }
  public void upisiIznos(double izn) {
    this.izn = izn;
  }
}
```

Korišćenje interfejsa (5/5)

Glavni program koji koristi sve klase

```
class program {
  public static void main(String [] args) {
    //kreiramo dva studenta i jednu svemirsku stanicu
    Student petar =
      new Student("Petar", "Petrovic", 1, "PMF", 232.1, 10000);
    Student marko =
      new Student("Marko", "Markovic", 2, "FTN", 232.1, 10000);
    SvemirskaStanica mojaStanica =
      new SvemirskaStanica(30, 230000, 5000000000, 1000, 48000, 30000);

    //devalvacija 10 posto
    OsigZavod.uvecajZbogDevalvacije(petar, 10);
    OsigZavod.uvecajZbogDevalvacije(marko, 10);
    OsigZavod.uvecajZbogDevalvacije(mojaStanica, 10);

    System.out.println("Ako se Petar povredi za vreme nastave, " +
      "bice mu isplaceno " + petar.iznos() + " dinara.");
    System.out.println("Ako se stanica sudari sa asteroidom, vlasniku " +
      "ce biti isplaceno " + mojaStanica.iznos() + " dinara.");
  }
}
```

Nasleđivanje interfejsa

- Za razliku od klasa, jedan interfejs može da nasledi više drugih interfejsa - tada on nasleđuje sve članove svih svojih prethodnika.
- Zbog višestrukog nasleđivanja, u napravljenom interfejsu možemo imati konflikt imena.
- Konflikt imena metoda nije problem jer su svi metodi apstraktni.
- Konflikt imena polja (konstantne vrednosti) se jednostavno ispravlja kastingom.

Jednostruko nasleđivanje interfejsa

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

interface Udzbenik extends Knjiga {
  String izOblasti();
}
```

Višestruko nasleđivanje interfejsa

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

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

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

Nasleđivanje interfejsa - konflikt imena (1/3)

Primer - Konflikt imena polja kod višestrukog nasleđivanja

```
class PMFprofesor implements ProfesorUniverziteta {
  private int staz;
  private int brRadova;

  PMFprofesor(int staz, int brRadova) {
    this.staz = staz;
    this.brRadova = brRadova;
  }

  public int radniStaz() {
    return staz;
  }

  public int brojNapisanihRadova() {
    return brRadova;
  }

  public String vrstaObrazovneUstanove() {
    return "Fakultet";
  }

  public String imeFakulteta() {
    return "Prirodno-matematicki fakultet";
  }
}
```

Nasleđivanje interfejsa - konflikt imena (2/3)

```
class program {
  public static void main(String[] args) {

    //Statickim clanovima pristupamo navodjenjem imena tipa,
    //tacke i imena clana:

    System.out.println(NaucniRadnik.opisPosla);          //stvaranje znanja
    System.out.println(ProsvetniRadnik.opisPosla);       //prenosenje znanja

    //Mozemo im pristupiti i navodjenjem instance tog tipa,
    //tacke i imena clana.

    //U slucaju konflikta imena (dvosmislenosti) moramo koristiti casting.
    PMFprofesor p = new PMFprofesor(15, 75);
    System.out.println(((NaucniRadnik)p).opisPosla);     //stvaranje znanja
    System.out.println(((ProsvetniRadnik)p).opisPosla); //prenosenje znanja
    ProfesorUniverziteta pu = p;
    System.out.println(((NaucniRadnik)pu).opisPosla);    //stvaranje znanja
    System.out.println(((ProsvetniRadnik)pu).opisPosla);//prenosenje znanja
    NaucniRadnik nr = p;
    System.out.println(nr.opisPosla);                    //stvaranje znanja
    ProsvetniRadnik pr = p;
    System.out.println(pr.opisPosla);                    //prenosenje znanja
  }
}
```

Nasleđivanje interfejsa - konflikt imena (3/3)

Ispis

```
stvaranje znanja
prenosenje znanja
stvaranje znanja
prenosenje znanja
stvaranje znanja
prenosenje znanja
stvaranje znanja
prenosenje znanja
```

- Java prevodilac bi prijavio grešku zbog dvosmislenosti ako bismo u programu koristili bilo koju od sledećih naredbi:

```
System.out.println(ProfesorUniverziteta.opisPosla);
System.out.println(p.opisPosla);
System.out.println(pu.opisPosla);
```

Ugnježdene statičke klase/interfejsi (1/4)

- Korišćenje ugnježdenih statičkih referencijalnih tipova u klasama smo već opisali kod klasa. Na isti način se koriste i ugnježdeni statički referencijalni tipovi u interfejsu.
- Ugnježdenom tipu se pristupa pomoću imena interfejsa u kojem se on nalazi, tačke i imena ugnježdenog tipa. Na primer, ako se u interfejsu **I** nalazi ugnježdena klasa sa imenom **K**, tada se njoj van interfejsa **I** pristupa sa I.K

Primer - Interfejs sa ugnježdenim statičkim interfejsom

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

  interface IspisivacSadrzaja {  //ugnjezdeni staticki interfejs
    boolean imaJos();
    String sledeciNaslovIStrana();
  }
}
```

Ugnježdene statičke klase/interfejsi (2/4)

```
class SadrzajKnjige implements Knjiga.IspisivacSadrzaja {
  private java.util.Vector elementi;  //elementi sadrzaja
  private int pozicija;  //indeks elementa kojeg treba ispisati

  SadrzajKnjige() {
    elementi = new java.util.Vector();
    pozicija = 0;
  }

  void dodaj(String naslov, int strana) {
    String novi = naslov + "----" + strana;
    elementi.add(novi);  //dodaje na kraj liste
  }

  public boolean imaJos() {
    return pozicija < elementi.size();
  }

  public String sledeciNaslovIStrana() {
    if (imaJos()) {
      pozicija++;
      return (String)elementi.get(pozicija-1);
    }                    // mora konverzija, jer get vraca Object
    else
      return null;
  }
}
```

Ugnježdene statičke klase/interfejsi (3/4)

```
class Udzbenik implements Knjiga {
  private String ime;
  private String autor;
  private SadrzajKnjige sadrzaj;

  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);
  }
}
```

Ugnježdene statičke klase/interfejsi (4/4)

```
class program {
  public static void main(String[] args) {
    Knjiga k = new Udzbenik("Programski jezik Scheme",
      "Z. Budimac, M. Ivanovic, M. Badjonski, D. Tosic");
    k.dodajUSadrzaj("Uvodne napomene", 3);
    k.dodajUSadrzaj("O programskim jezicima", 3);
    k.dodajUSadrzaj("Funkcionalni stil programiranja", 6);
    k.dodajUSadrzaj("Evolucija funkcionalnih programskih jezika", 13);
    // i tako dalje
    System.out.println("Sadrzaj knjige " + k.naslov() + " autora " +
      k.autor() + " je:");
    Knjiga.IspisivacSadrzaja isp = k.sadrzaj();
    while (isp.imaJos()) {
      System.out.println(isp.sledeciNaslovIStrana());
    }
  }
}
```

Ugnježdene statičke klase i interfejsi, primer 2

```
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) {...
}
```

Ugnježdene statičke klase i interfejsi, primer 2

```
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

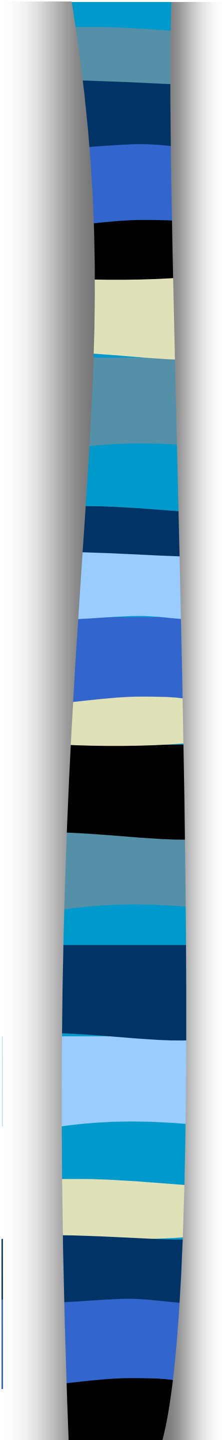
Dinamičko vezivanje: U vremenu izvršavanja formalni parametar "komparator" će biti referenca na neku instancu klase koja implementira interfejs PoređenjeRadnika

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

Ugnježdene statičke klase i interfejsi, primer 2

```java
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 opadajuce po platama
        rj.informacijeOZaposlenima(
            new RadnaJedinica.PoredjenjeRadnika() {
                public int uporedi(RadnaJedinica.Radnik                    2) {
                    return r2.getPlata() - r1.getPlata()
                }
            }
        );
    }
}
```

```
Ime, plata
Aleksandar Markovic, 55000
Milovan Petrovic, 75000
Veselin Stankovic, 50000
Zorana Vukmirovic, 60000

Ime, plata
Milovan Petrovic, 75000
Zorana Vukmirovic, 60000
Aleksandar Markovic, 55000
Veselin Stankovic, 50000
```



Nabrojivi tip podataka

Nabrojivi tip podataka

- U programiranju se često javi potreba da se radi sa tipovima čiji skup mogućih vrednosti predstavlja mali broj unapred poznatih konstanti.
- Dozvoljene vrednosti tipa podataka je moguće *nabrojati* u toku pisanja programa i ne očekuje se da će skup vrednosti značajno menjati u budućnosti.
- Neki primeri takvih skupova vrednosti su: **dani u nedelji**, **planete Sunčevog sistema**, **sorte jabuka**, itd.
- Do Jave 1.5, programeri su bili primorani da takve tipove podataka „simuliraju" na različite načine, najčešće upotrebom konstanti celobrojnog tipa, kao u sledećem primeru, gde su konstantama označene različite sorte jabuka i pomorandži.

Nabrojivi tip podataka

Primer 6.77: Simulacija nabrojivog tipa podataka pomoću celobrojnih konstanti

```
class Apples {
  public static final int GOLDEN_DELICIOUS = 0;
  public static final int RED_DELICIOUS    = 1;
  public static final int GRANNY_SMITH     = 2;
  public static final int MUTSU            = 3;
}

class Oranges {
  public static final int BAHIANINHA = 1;
  public static final int JINCHENG   = 2;
  public static final int NAVELINA   = 3;
}
```

- Nedostaci ovog pristupa su višestruki:
  - Prevodilac nema informacija na osnovu kojih može da radi proveru usklađenosti tipova, pa je moguće mešati konstante različitih tipova u izrazima, na primer: double juice = 0.3 * Apples.MUTSU + 0.7 * Oranges.NAVELINA;
  - Ne postoji jednostavan i pouzdan način da se iterira kroz sve konstante;
  - Ne postoji jednostavan način da se konstantama pridruže neki podaci.

- Da bi se opisana potreba zadovoljila i problemi rešili, u Javi 1.5 uveden je novi referencijalni tip podataka, i odgovarajuća ključna reč **`enum`**, kojim je omogućeno jednostavno definisanje nabrojivih tipova podataka.
- Sintaksa definisanja novog nabrojivog tipa slična je definiciji klase, s tim da se umesto ključne reči class koristi enum.
- Konstante nabrojivog tipa navode se kao polja razdvojena zarezima, i po konvenciji im se dodeljuju imena sastavljena od velikih slova.
- Na kraju liste konstanti opciono se može ostaviti **zarez**, a lista se može završiti i znakom **tačka-zarez**.
- U sledećem primeru date su definicije nabrojivih tipova podataka za sorte jabuka i pomorandži koje smo u prethodnom primeru simulirali.

Nabrojivi tip podataka

Primer 6.78: Nabrojivi tipovi podataka bez simuliranja

```
enum Apples {
   GOLDEN_DELICIOUS, RED_DELICIOUS, GRANNY_SMITH, MUTSU
}

enum Oranges {
   BAHIANINHA, JINCHENG, NAVELINA
}
```

– Konstante nabrojivih tipova Apples i Oranges nije moguće mešati u izrazima.

– Konstantama se mora pristupati u kvalifikovanom obliku.

– Nabrojive tipove je moguće koristiti u okviru naredbe switch, gde se pri navođenju labela kvalifikovani deo imena izostavlja.

Nabrojivi tip podataka

Primer 6.79: Upotreba nabrojivih tipova

```
public class UsingEnums {
  public static void main(String[] args) {

    Oranges orange = Oranges.NAVELINA;

    // greska: mesanje brojeva, jabuka i pomorandzi
    // double juice = 0.3 * apple + 0.7 * orange;

    // poljima se pristupa kvalifikovano
    if (apple == Apples.RED_DELICIOUS)
      System.out.println("I like it :)");
    else
      System.out.println("I don't like it :(");

    // u switch naredbi mora se koristiti nekvalifikovan oblik
    switch (orange) {
      case BAHIANINHA:
        System.out.println("It's a bahianinha");
        break;
      case JINCHENG:
        System.out.println("It's a jincheng");
        break;
      default: // case NAVELINA
        System.out.println("It's a navelina");
    }
  }
}
```

Nabrojivi tipovi kao klase

- Nabrojivi tipovi u Javi se smatraju specijalnom vrstom klasa, i kao takvi spadaju u referencijalne tipove podataka.
- Dok su u većini drugih programskih jezika nabrojivi tipovi koncipirani kao prosti redni (ordinalni) tipovi u čijoj osnovi stoje celi brojevi, u Javi su konstante nabrojivog tipa predstavljene kao finalne instance (objekti) tog nabrojivog tipa.
- Ovo je ilustrovano u sledećem primeru, gde su polja nabrojivog tipa i klase definisana na principijelno isti način (što ne čini nabrojivi tip i klasu identičnim).

Primer 6.80: Nabrojivi tip i njemu principijelno slična klasa

```
enum PrimerNabrojivogTipa {
  KONSTANTA1, KONSTANTA2
}

class PrimerKlase {
  public static final PrimerKlase KONSTANTA1 = new PrimerKlase();
  public static final PrimerKlase KONSTANTA2 = new PrimerKlase();
}
```

Nabrojivi tipovi kao klase

- Za razliku od klasa, svaki definisani nabrojivi tip implicitno nasleđuje klasu `java.lang.Enum` koja nasleđuje `java.lang.Object`
- Posledica ovoga je da nabrojivi tipovi podataka ne mogu da nasleđuju druge tipove (ali mogu da implementiraju interfejse).
- Svi definisani nabrojivi tipovi su implicitno `final`, što znači da se ne mogu proširivati nasleđivanjem nabrojivog tipa.
- S druge strane, slično klasama, nabrojivi tipovi mogu da imaju polja, metode, konstruktore, unutrašnje klase i sve ostale elemente koji se mogu naći u klasama.
- Kada nabrojivi tip pored liste konstanti sadrži i druge članove, lista konstanti mora se navesti prva i mora se završiti znakom `;`

Nabrojivi tipovi kao klase

- Poljima se u nabrojivom tipu proizvoljan podatak može vezati za konstantu nabrojivog tipa.
- Ako nabrojivi tip nema `default` konstruktor, postojećem konstruktoru se moraju eksplicitno proslediti argumenti prilikom navođenja konstanti nabrojivog tipa.
- S obzirom da su nabrojivi tipovi (indirektni) potomci klase **Object**, dostupan je i metod **toString()**.

Nabrojivi tipovi kao klase

Primer 6.81: Nabrojivi tip za dane u nedelji

```
enum DaniUNedelji {
  PONEDELJAK(1, false), // eksplicitan poziv konstruktora
  UTORAK(2, false),
  SREDA(3, false),
  CETVRTAK(4, false),
  PETAK(5, false),
  SUBOTA(6, true),
  NEDELJA(7, true);

  private final int dan; // redni broj dana
  private final boolean vikend; // da li je vikend?

  private DaniUNedelji(int dan, boolean vikend) { // konstruktor
    this.dan = dan;
    this.vikend = vikend;
  }

  // posto nabrojivi tip potice od Object, imamo i toString!
  public String toString() {
    // default toString metod vraca string reprezentaciju konstante
    String str = super.toString();
    return str + " je " + dan + ". dan u nedelji, i " +
      (vikend ? "jeste" : "nije") + " deo vikenda";
  }
}

class StampajDane {
  public static void main(String[] args) {
    System.out.println(DaniUNedelji.PONEDELJAK);
    System.out.println(DaniUNedelji.UTORAK);
    System.out.println(DaniUNedelji.SREDA);
    System.out.println(DaniUNedelji.CETVRTAK);
    System.out.println(DaniUNedelji.PETAK);
    System.out.println(DaniUNedelji.SUBOTA);


    System.out.println(DaniUNedelji.NEDELJA);
  }
}
```

Navedeni kod daje sledeći izlaz:

```
PONEDELJAK je 1. dan u nedelji, i nije deo vikenda
UTORAK je 2. dan u nedelji, i nije deo vikenda
SREDA je 3. dan u nedelji, i nije deo vikenda
CETVRTAK je 4. dan u nedelji, i nije deo vikenda
PETAK je 5. dan u nedelji, i nije deo vikenda
SUBOTA je 6. dan u nedelji, i jeste deo vikenda
NEDELJA je 7. dan u nedelji, i jeste deo vikenda
```

Nabrojivi tipovi kao klase

- Iako je nabrojivi tip podataka referencijalni tip, konstantama nabrojivog tipa su ipak u pozadini automatski dodeljuju redni brojevi, počevši od 0 za prvu konstantu u listi.
- Do rednog broja konstante može se doći pozivom metoda **ordinal().**
- Korišćenje ovih vrednosti se ne preporučuje ako je potrebno neki broj pridružiti konstantama savetuje se upotreba polja (kao polje dan u prethodnom primeru)

Nabrojivi tipovi kao klase

- Iteracija kroz sve konstante nabrojivog tipa lako se postiže pomoću statičkog metoda **values()** nabrojivog tipa, koji vraća niz tih njegovih konstanti, u redosledu u kojem su navedene u definiciji tipa.
- Tako se telo **main()** metoda iz prethodnog primera može zameniti sledećim kratkim fragmentom koda:

```
For (DaniUNedelji dan : DaniUNEdelji.values())

System.out.println(dan)
```

Operatori nad referencijalnim tipovima

Operatori nad referencijalnim tipovima

Operator	Opis
`=`	dodela
`==`	ispitivanje jednakosti
`!=`	ispitivanje nejednakosti
`+`	konkatenacija stringova
`? :`	uslovni operator
`.`	pristup članu klase ili interfejsa
`instanceof`	ispitivanje tipa objekta
(*imeTipa*)	eksplicitna konverzija tipa
`new`	kreiranje instance klase
`[ ]`	pristup elementu niza

Dodela

- Operator dodele **=** je binarni operator kojim se promenljivoj sa leve strane operatora dodeljuje vrednost sa desne strane operatora
- Vrednost promenljive referencijalnog tipa nije objekat, već **referenca objekata** (pokazivač na objekat) – zbog toga se dodelom vrednosti jedne promenljive drugoj promenljivoj ne pravi nova kopija objekta, već se kopira samo pokazivač na objekat, tako da posle dodele obe promenljive pokazuju na isti objekat
- Vrednost koja se dodeljuje promenljivoj referencijalnog tipa može biti:
  - **null** ili
  - referenca objekta čiji je tip jednak tipu promenljive ili
  - referenca objekta čiji tip je moguće konvertovati u tip promenljive korišćenjem neke od proširujućih referencijalnih konverzija

Proširujuće referencijalne konverzije

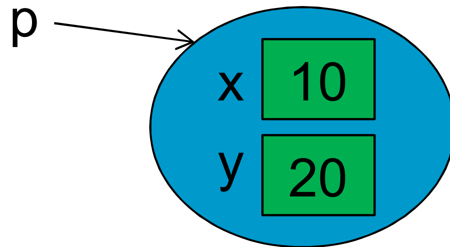
- Proširujuće referencijalne konverzije su sledeće konverzije:
  - konverzija iz bilo koje klase **P** u bilo koju klasu **N**, pod uslovom da je **P** podklasa klase **N**,
  - konverzija iz bilo koje klase **K** u bilo koji interfejs **I**, pod uslovom da klasa **K** implementira interfejs **I**,
  - konverzija vrednosti **null** u bilo koji referencijalni tip,
  - konverzija iz bilo kog interfejsa **B** u bilo koji interfejs **A**, pod uslovom da interfejs **B** nasleđuje interfejs **A**,
  - konverzija iz bilo kog interfejsa u klasu **Object**,
  - konverzija iz bilo kog niza u klasu **Object**,
  - konverzija iz bilo kog niza u interfejs **Cloneable** i u interfejs **java.io.Serializable**,
  - konverzija iz bilo kog niza **A[ ]** u bilo koji niz **B[ ]**, pod uslovom da su **A** i **B** referencijalni tipovi i da postoji proširujuća referencijalna konverzija iz tipa **A** u tip **B**.

1. Konverzija iz bilo koje klase P u bilo koju klasu N, pod uslovom da je P podklasa klase N

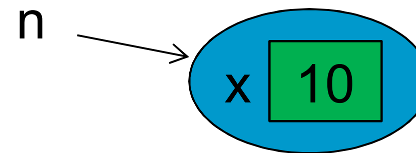
```
class N {
    int x = 10;
}
```

```
class P extends N {
    int y = 20;
}
```

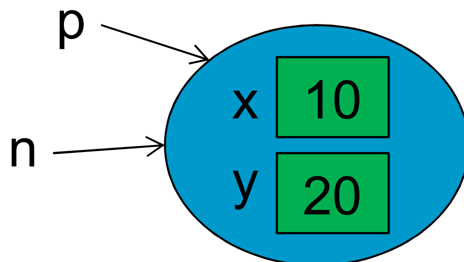
```
P p = new P();
```



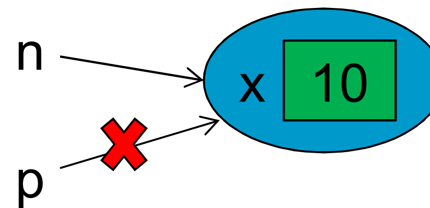

```
N n = new N();
```




```
N n = p;
```




```
P p = n;   /* ERROR */
```



2. Konverzija iz bilo koje klase K u bilo koji interfejs I, pod uslovom da klasa K implementira interfejs I.
Instanca standardnog interfejsa dobija vrednost reference na objekat klase String koja implementira taj interfejs.

```
String s = "Ovaj string ima 28 karaktera";
CharSequence cs = s;
System.out.println(cs.length());
```

3. Konverzija iz bilo kog interfejsa B u bilo koji interfejs A, pod uslovom da interfejs B nasleđuje interfejs A

```
java.util.Set skupKaraktera = new java.util.TreeSet();
for (int i = 0; i < cs.length(); i++)
    skupKaraktera.add(cs.charAt(i));

java.util.Collection kolekcija = skupKaraktera;

System.out.println("String sadrzi " + kolekcija.size() + " razlicitih karaktera");
```

4. Konverzija vrednosti null u bilo koji referencijalni tip

```
s = null;
lista = null;
```

5. Konverzija iz bilo kog niza u klasu Object, interfejs Cloneable i java.io.Serializable

Svaki nizovski tip je interno predstavljen kao klasa koja direktno nasleđuje klasu Object i implementira interfejse Cloneable i java.io.Serializable

Eksplicitna konverzija (kasting)

- Vrednost nekog tipa se može eksplicitnom konverzijom pretvoriti u odgovarajuću vrednost drugog tipa
- Ime ciljnog tipa se navodi u običnim zagradama i koristi se kao unarni operator
- Eksplicitna konverzija tipa se najčešće koristi prilikom dodele vrednosti promenljivoj kada se tip promenljive i tip vrednosti razlikuju i kada se oni ne mogu izjednačiti implicitnom proširujućom referencijalnom konverzijom. Da bismo nabrojali koje su sve eksplicitne referencijalne konverzije moguće, uočićemo tri referencijalna tipa koja učestvuju u eksplicitnoj konverziji:
  - **tip vrednosti koja se konvertuje** koji važi u toku **prevođenja** programa (neka je to tip **P**),
  - **tip vrednosti koja se konvertuje** koji važi u toku **izvršavanja** programa (tip **I**)
  - **ciljni tip u koji se vrši konverzija** (tip **C**).
- Primetimo da tipovi **P** i **I** mogu biti različiti tipovi. Na primer, **P** i **I** mogu biti klase pri čemu je **P** nadklasa klase **I**, ili tip **P** može biti interfejs koji implementira klasa **I**.

```
class I extends P {}
P src = new I();
C dst = (C) src;
```

```
class I implements P {}
P src = new I();
C dst = (C) src;
```

Eksplicitna konverzija (kasting)

Za tipove P, I i C mora da važi jedan od sledećih sedam uslova da bi eksplicitna referencijalna konverzija bila dozvoljena:

1. promenljivoj tipa **C** je moguće dodeliti vrednost tipa **P**,
2. tipovi **I** i **C** su isti tipovi,
3. tipovi **I** i **C** su klase i **I** je podklasa klase **C**,
4. tip **C** je interfejs a tip **I** je klasa koja ga implementira,
5. tip **I** je nizovski tip a tip **C** je interfejs Cloneable ili interfejs java.io.Serializable,
6. tipovi **I** i **C** su nizovski tipovi čiji su elementi istog prostog tipa,
7. tipovi **I** i **C** su nizovski tipovi čiji su elementi referencijalnih tipova (deklarisan tip elemenata niza **I** je **I1**, a deklarisan tip elemenata niza **C** je **C1**) pri čemu važi jedan od sledećih uslova:
   - **I1** je interfejs a **C1** je tip Object,
   - **I1** i **C1** su interfejsi i **I1** direktno ili indirektno nasledjuje interfejs **C1** ili je jednak interfejsu **C1**,
   - na **I1** i **C1** je moguće rekurzivno primeniti jedno od sedam gore navedenih pravila, pri čemu je **P = I1, I = I1 i C = C1**.

Pravilo 1: promenljivoj tipa **C** je moguće dodeliti vrednost tipa **P**

- **Postoje dva slučaja, u oba je eksplicitno kastovanje redundantna operacija.**
- **C i P** su isti tipovi

```
String s1 = “Zdravo svete”;
String s2 = (String) s1;
```

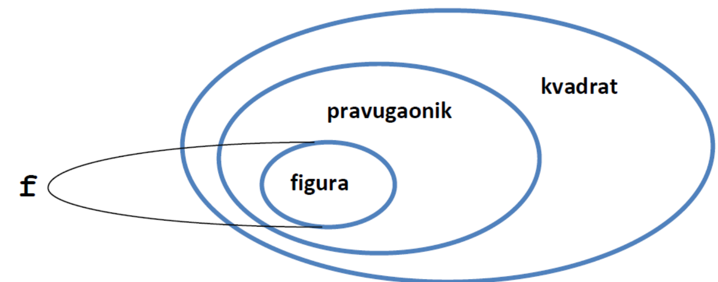
- **C i P** nisu isti tipovi → postoji proširujuća referencijalna konverzija iz **P** u **C**

```
/* C = Object, P = String */
String s = “Zdravo svete”;
Object o = (Object) s;
```

Pravila 2 i 3: I i C su isti tipovi ili je I podklasa od C

- Proširujuća (implicitna) referencijalna konverzija redukuje vidljivost sadržaja objekta, sakriva detalje (“apstrahuje” objekte).

```
class Figura {}
class Pravougaonik extends Figura {}
class Kvadrat extends Pravougaonik {}
Figura f = new Kvadrat();
```

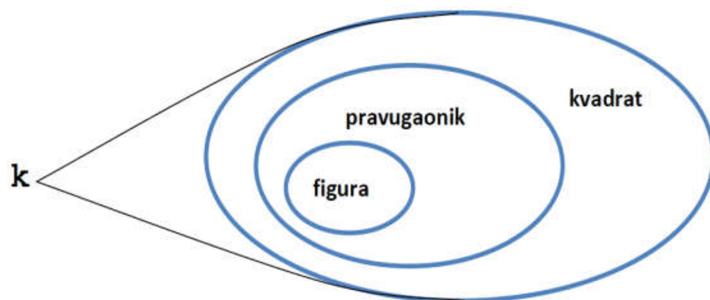


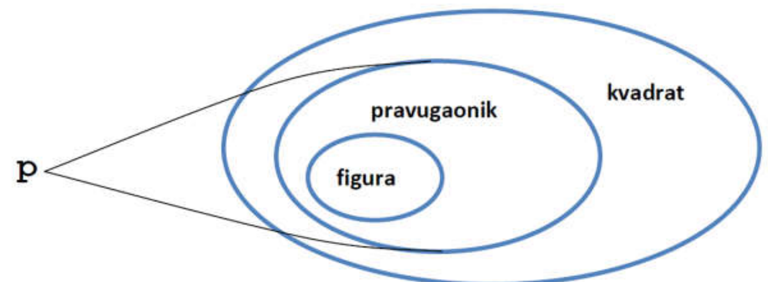
- Eksplicitnom konverzijom reference u tip podklase se otkrivaju detalji objekta (atributi, metode) koji su bili sakriveni nekom prethodnom proširujućom referencijalnom konverzijom.

```
P src = new I();
C dst = (C) src;
```

```
// pravilo 2
Kvadrat k = (Kvadrat) f;
```

```
// pravilo 3
Pravougaonik p = (Pravougaonik) f;
```





Pravila 4 i 5: I implementira C

- **Pravilo 4**: tip C je interfejs a tip I je klasa koja ga implementira,
- Ponovo eksplicitna konverzija “poništava efekat” neke prethodne proširujuće referencijalne konverzije

```
Object o = "Zdravo svete";
System.out.println(o.equals("Zdravo svete"));

CharSequence s = (CharSequence) o;        // pravilo 4
System.out.println(s.length());
```

- **Pravilo 5:** primena pravila 4 na proizvoljan nizovski tip
- tip I je nizovski tip a tip C je interfejs Cloneable ili interfejs java.io.Serializable,

Svaki nizovski tip je interno predstavljen kao klasa koja direktno nasleđuje klasu **Object** i implementira interfejse **Cloneable** i **java.io.Serializable**

Eksplicitna konverzija (kasting)

```
interface A {}

interface B {}

interface C extends A {}

class X implements A {}

class Y extends X implements C
{}

class Z extends Y implements B
{}

class program {
  public static void main
        (String[] args) {
    Object o = new X[10];
    X[] niz = (X[]) o;
// pravilo 2
// tipovi I i C su isti tipovi
```

```
P src = new I();
C dst = (C) src;
```

```
    X x = new Z();
    Y y = (Y) x;   //pravilo 3
//I je podklasa klase C  (Z je
podklasa Y)
    A a = new Y();
    C c = (C) a;   //pravilo 4
// tip C je interfejs a tip I je
klasa koja ga implementira (C je
interfejs, a Y je klasa koja ga
implementira)
```

- Operator ispitivanja jednakosti **==** i operator ispitivanja nejednakosti **!=** se pored primene na vrednosti prostih tipova mogu primeniti i na vrednosti referencijalnih tipova - njima se ispituje da li dve reference pokazuju ili ne pokazuju na isti objekat

- Rezultat primene operatora **!=** je uvek suprotan od rezultata primene operatora **==**

- Ovim operatorima nikada ne treba proveravati da li su dva objekta jednaka, već za to treba koristiti specijalno napravljene metode

Ispitivanje jednakosti i nejednakosti

Primer – ispitivanje jednakosti

```
class program {
  public static void main(String[] args) {
    Tacka A = new Tacka(2, 2);
    Tacka B = new Tacka(2, 2);
    Tacka C = A;
    Tacka D = null;
    Tacka E = null;
    if (A == B)
      System.out.println("A == B");
    else
      System.out.println("A != B");
    if (A == C)
      System.out.println("A == C");
    else
      System.out.println("A != C");
```

```
   if (B == C)
   System.out.println("B == C");
   else
       System.out.println("B != C");
     if (D == E)
       System.out.println("D == E");
     else
       System.out.println("D != E");
   }
 }
```

Ispis

```
A != B
A == C
B != C
D == E
```

Uslovni operator

- Ternarni operator **?:** može biti primenjivan i na referencijalne vrednosti
- Prvi operand ovog operatora je uvek logičkog tipa, a druga dva operanda mogu biti oba prostog tipa, ali mogu biti i oba referencijalnog tipa
- Ako su druga dva operanda ovog operatora istog referencijalnog tipa, tada će i rezultat operatora biti tog tipa. Kada su druga dva operanda različitog referencijalnog tipa, na primer jedan operand je tipa **A** a drugi operand je tipa **B**, tada za te tipove mora da važi sledeće:
  - vrednost tipa **A** je moguće dodeliti promenljivoj tipa **B**, ili
  - vrednost tipa **B** je moguće dodeliti promenljivoj tipa **A**
- Uslovni operator **?:** se sa referencijalnim vrednostima koristi slično kao i sa prostim

Primer

```
Object o = (1*2*3 != 1+2+3) ? new Object() : "abc";
```

Pristup članu klase ili interfejsa (1/3)

- Operatorom **.** (tačka) se pristupa poljima, metodima i unutrašnjim klasama neke klase, kao i metodima i poljima nekog interfejsa.
- Moguće je pristupiti samo onim članovima klase koji su u datom kontekstu vidljivi.
- Nestatičkim članovima klase se pristupa navođenjem imena objekta, tačke i imena člana.
- Statičkim članovima klase se pristupa navođenjem imena klase, tačke i imena člana, a moguće im je pristupiti i na isti način kao nestatičkim članovima, navođenjem imena nekog objekta te klase, tačke i imena statičkog člana.
- Poljima i metodima interfejsa se pristupa na isti način kao poljima i metodima klasa.
- Operator **.** je levo asocijativan. Na primer: Izraz poli.izvod().stepen() je ekvivalentan izrazu (poli.izvod()).stepen()

Pristup članu klase ili interfejsa (2/3)

- Metodu klase se pristupa da bi se izvršile naredbe njegovog tela. Da bi se metod mogao izvršiti sa datim parametrima (stvarni parametri - **S**), tipovi tih parametara se moraju slagati sa tipom parametara navedenih prilikom deklaracije metoda (formalni parametri - **F**).
- Parametri su usklađeni ako su tipovi **S** i **F** isti, ili postoji proširujuća prosta konverzija tipa **S** u tip **F**, ili postoji proširujuća referencijalna konverzija tipa **S** u tip **F.**
- Kod pozivanja metoda ne vrše se sužavajuće proste konverzije konstanti tipa **int** u tip **byte**, **short ili char** kao što se to radi kod operatora dodele.
- Zbog toga nije dozvoljeno pozivati metod sa stvarnim parametrom tipa **int** ako je odgovarajući formalni parametar tipa **byte, short ili char**.
- U takvoj situaciji mora se primeniti eksplicitna konverzija.

Pristup članu klase ili interfejsa (3/3)

Primer - poziv metoda uz primenu eksplicitne konverzije

```
public class MaliBroj {
  private byte broj;
  public void stavi(byte br) {
    broj = br;
  }
  public byte citaj() {
    return broj;
  }
}

class program {
  public static void main(String[] args) {
    MaliBroj m = new MaliBroj();
    m.stavi( (byte) 7 );
  }
}
```

instanceof operator

- Binarni operator **instanceof** se koristi samo kod referencijalnih tipova, njime se ispituje da li je tip prvog operanda jednak drugom operandu.
- Prvi operand može biti samo neki objekat ili konstanta **null.**
- Drugi operand je ime nekog referencijalnog tipa.
- Ako je tip prvog operanda moguće eksplicitnom konverzijom konvertovati u tip naveden u drugom operandu, tada je vrednost operatora **true**, a inače je **false.**

```
class A {}

interface I {}

class B extends A implements I {}

class program {
  public static void main(String[] args) {
    A a = new B();
    if (a instanceof I)
      System.out.println("Jeste tip interfejsa I");
    else
      System.out.println("Nije tip interfejsa I");
  }
}
```

Primer primene instanceof operatora

```
class Parent {}
class Child extends Parent implements MyInterface {}
interface MyInterface {}

class InstanceofDemo {
    public static void main(String[] args) {

        Parent obj1 = new Parent();
        Parent obj2 = new Child();

        System.out.println("obj1 instanceof Parent: "
            + (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: "
            + (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: "
            + (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: "
            + (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: "
            + (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: "
            + (obj2 instanceof MyInterface));
    }
}
```

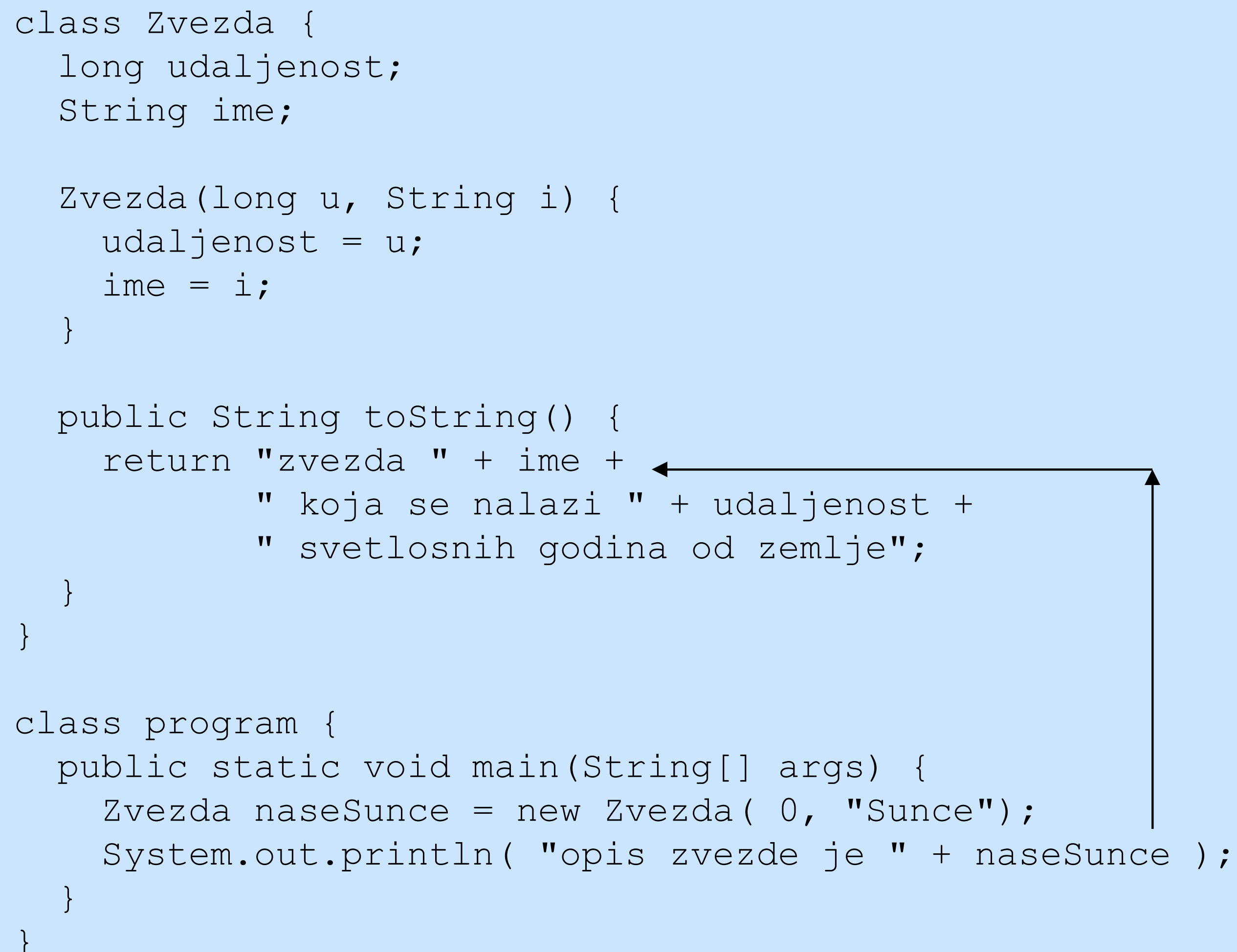
```
Output:
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof MyInterface: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

- Konkatenacija (spajanje) stringova se vrši binarnim operatorom **+**
- Ako su oba operanda stringovi, onda je rezultat novi string koji je jednak stringu koji bi nastao spajanjem stringova operanada
- Ako je samo jedan operand tipa **String** a drugi je nekog drugog tipa, onda se najpre vrednost operanda nestringovskog tipa konvertuje u tip **String** nakon čega se rezultat kreira isto kao u slučaju kada su oba operanda tipa **String**
- Konvertovanje u tip **String** je uvek moguće izvršiti:
  - Prosti tipovi – uobičajena konverzija
  - **null** – u string **“null”**
  - Bilo koji objekat – u string nastao pozivom metoda **toString()** koji vraća vrednost tipa String. Ovaj metod sadrži svaki objekat, zato što je on deklarisan u klasi **Object** sa **public** modifikatorom, pa ga sve klase nasleđuju

Primer - sabiranje i konkatenacija – oba operatora su levo asocijativna

```
------------------------------------------------------------------------
System.out.println( "abc" + 1 + 2); \\dve konkatenacije, ispisuje abc12
System.out.println( 1 + "abc" + 2 ); \\dve konkatenacije, ispisuje 1abc2
System.out.println( 1 + 2 + "abc" ); \\sabiranje i konkatenacija,
                                      \\ispisuje 3abc
```

Konkatenacija stringova - primer



Primer konkatenacije stringova

```
class Zvezda {
  long udaljenost;
  String ime;

  Zvezda(long u, String i) {
    udaljenost = u;
    ime = i;
  }

  public String toString() {
    return "zvezda " + ime +
           " koja se nalazi " + udaljenost +
           " svetlosnih godina od zemlje";
  }
}

class program {
  public static void main(String[] args) {
    Zvezda naseSunce = new Zvezda( 0, "Sunce");
    System.out.println( "opis zvezde je " + naseSunce );
  }
}
```

Teorijske vežbe 5

Objektno-orijentisano programiranje

Inicijalizacija

- Inicijalizacija nestatičkih polja klase se može obaviti na jedan od sledećih načina:
  - Pri deklaraciji polja
  - Unutar konstruktora
  - Unutar inicijalizatora
- Statička polja se inicijalizuju na sledećim mestima:
  - Pri deklaraciji polja
  - Unutar statičkog inicijalizatora

Inicijalizacija (2)

- Unutar jedne klase može biti proizvoljan broj konstruktora, pri čemu ne smeju postojati dva konstruktora sa istim parametrima (mora postojati ili različit broj parametara, ili da parametri budu različitog tipa).
- Unutar jednog konstruktora može se pozvati drugi konstruktor, i to na sledeći način: `this(p1,p2,...);` Poziv drugog konstruktora se može odviti samo u okviru prve linije koda konstruktora.
- Unutar jedne klase može biti proizvoljan broj inicijalizatora i statičkih inicijalizatora, pri čemu je redosled izvršavanja određen redosledom kojim su inicijalizatori navedeni.
- Finalna polja klase moraju biti inicijalizovana.

Inicijalizacija klase

- Redosled izvršavanja koda za inicijalizaciju klase:
  - Prilikom prve upotrebe klase inicijalizuju se sva njena statička polja. Ova polja se inicijalizuju na default vrednosti ukoliko pri deklaraciji polja nije naveden kod za inicijalizaciju, a ukoliko jeste, polja se incijalizuju onako kako je zadato.
  - Nakon toga (takođe pri prvoj upotrebi klase) izvršavaju se statički inicijalizatori, i to redosledom kojim su navedeni.

Inicijalizacija objekta

- Redosled izvršavanja koda za inicijalizaciju objekta:
  - Poziv konstruktora natklase ili drugog konstruktora trenutne klase (u zavisnosti od toga šta se nalazi u prvoj liniji konstruktora)
  - Inicijalizacija svih polja (ukoliko polja nisu inicijalizovana pri deklaraciji, njihova vrednost postaje default vrednost odgovarajućeg tipa)
  - Pozivanje svih inicijalizatora
  - Nastavak izvršavanja tela konstruktora

Zadatak 11

- Klasa BankovniRacun treba da modelira racun u banci.
- Klasa treba da sadrži informacije o broju računa, nosiocu računa, informaciju o tome da li je aktivirana usluga dozvoljenog minusa i trenutno stanje na računu.
- Klasa treba da ima dva statička polja najmanjiPromet i najveciPromet u kojima u svakom trenutku treba da stoji najmanji i najveći promet koji je banka na bilo kom računu do tog trenutka imala.
- Klasa takođe treba da sadrži statičku konstatu koja će nositi default vrednost broja računa. Ukoliko neki račun ima ovu vrednost, to znači da pravi broj računa još nije dodeljen.

```java
public class BankovniRacun {

   private static final int NEVALIDAN_BROJ_RACUNA = -1;

   private static int najmanjiPromet;
   private static int najveciPromet = Integer.MIN_VALUE;

   private int brojRacuna;
   private String nosilacRacuna;
   private boolean dozvoljenMinus;
   private int trenutnoStanje = 0;

   . . .
}
```

Zadatak 11

- Klasa BankovniRacun treba da ima više konstruktora različitih parametara.
- Pri implementaciji treba koristiti različite pristupe inicijalizacije polja klase.

```java
public BankovniRacun(int brojRacuna, String nosilacRacuna, boolean dozvoljenMinus, int
    trenutnoStanje) {
    System.out.println("Konstruktor 1");
    this.brojRacuna = brojRacuna;
    this.nosilacRacuna = nosilacRacuna;
    this.dozvoljenMinus = dozvoljenMinus;
    this.trenutnoStanje = trenutnoStanje;
}

public BankovniRacun(int brojRacuna, String nosilacRacuna, boolean dozvoljenMinus) {
    this(brojRacuna, nosilacRacuna, dozvoljenMinus, 0);
    System.out.println("Konstruktor 2");
}

public BankovniRacun(int brojRacuna, String nosilacRacuna) {
    this(brojRacuna, nosilacRacuna, false);
    System.out.println("Konstruktor 3");
}

public BankovniRacun(String nosilacRacuna) {
    this.nosilacRacuna = nosilacRacuna;
    System.out.println("Konstruktor 4");
}

static {
    System.out.println("Staticki inicijalizator");
    najmanjiPromet = Integer.MAX_VALUE;
}

{
    System.out.println("Inicijalizator");
    brojRacuna = NEVALIDAN_BROJ_RACUNA;
}
```

Zadatak 11

- Klasa BankovniRacun treba da ima sledeće metode:
  - boolean unesiPromet (int iznos) – Dodaje prosleđeni iznos na trenutno stanje, i pritom ažurira statička polja najmanjiPromet i najveciPromet. Metod vraća false ukoliko bi nakon dodavanja iznosa trenutno stanje postalo negativno a da pritom račun ne podržava dozvoljen minus (u tom slučaju se promet ne primenjuje); metod vraća true u suprotnom.
  - boolean brojRacunaJeDodeljen() – Vraća true ukoliko je broj računa dodeljen.
  - int nevalidniRacun() – Vraća vrednost konstante koja čuva ovu informaciju.
  - void stampajStatistikuBanke() – Štampa vrednosti polja najmanjiPromet i najveciPromet.
  - String toString() – Implementacija toString metoda.

```java
public boolean unesiPromet(int iznos) {
    if (trenutnoStanje + iznos < 0 && !dozvoljenMinus) {
            return false;
    }
    trenutnoStanje += iznos;
    if (iznos < najmanjiPromet) {
            najmanjiPromet = iznos;
    }
    if (iznos > najveciPromet) {
            najveciPromet = iznos;
    }
    return true;
}

public boolean brojRacunaJeDodeljen() {
    return brojRacuna != NEVALIDAN_BROJ_RACUNA;
}

public static int nevalidniRacun() {
    return NEVALIDAN_BROJ_RACUNA;
}

public static void stampajStatistikuBanke() {
    System.out.println("Najmanji promet dosad je: " + najmanjiPromet + ". Najveci promet dosad
    je: " + najveciPromet);
}

public String toString() {
    return "Racun broj " + brojRacuna + ". Ime nosioca racuna: " + nosilacRacuna +
            ". Dozvoljeni minus: " + dozvoljenMinus + ". Trenutno stanje: " + trenutnoStanje +
    ".";
}
```

```java
public static void main(String[] args) {
    System.out.println("Nevalidni racun ima kod: " +
            BankovniRacun.nevalidniRacun());
    BankovniRacun racun = new BankovniRacun("Jovan Jovanovic");
    BankovniRacun racun2 = new BankovniRacun(123, "Pera Peric", true, 200);
    System.out.println(racun);
    System.out.println(racun2);
    System.out.println("Broj prvog racuna je dodeljen: " +
            racun.brojRacunaJeDodeljen() + ", broj drugog racuna je dodeljen: "
            + racun2.brojRacunaJeDodeljen());
    racun.unesiPromet(-10);
    racun2.unesiPromet(-1038);
    racun2.unesiPromet(266);
    System.out.println(racun);
    System.out.println(racun2);
    BankovniRacun.stampajStatistikuBanke();
}
```