

Nasleđivanje, apstraktne klase, interfejsi i klasa Object

Nasleđivanje

Nasleđivanje (relacija “IS-A”, generalizacija-specijalizacija) predstavlja jedan od osnovnih koncepata objektno-orijentisanog programiranja. Nasleđivanje podrazumeva preuzimanje svih atributa i metoda **nadklase** (klasa koja je nasleđena) i njihovo prenošenje u **podklasu** (klasa koja nasleđuje). Krajnji efekat je taj da podklasa poseduje sve karakteristike i ponašanja nadklase, ali takođe može da sadrži i još neke karakteristike i ponašanje. Jedan od osnovnih motiva nasleđivanja je **izbegavanje redundanse u kodu** (ponavljanja koda). Kod koji je napisan u nadklasi ne mora da se ponovo piše u podklasi već se automatski prenosi iz nadklase. Ovo vodi ka **ponovnom upotrebljavanju već napisanog koda (“code reuse”)**.

Grafičko predstavljanje relacije nasleđivanja se vrši usmerenom linijom od podklase ka nadklasi. Vrh ove strelice predstavlja prazan trougao i nalazi se kod nadklase, dok se kod ove vrste relacije ne navodi kardinalnost jer je uvek jedan.

Programski jezik Java dozvoljava samo **jednostruko nasleđivanje** tj. podklasa može da ima samo jednu nadklasu. U nekim drugim objektno-orijentisanim programskim jezicima (npr. C++) je omogućeno višestruko nasleđivanje, ali se tu može javiti konflikt ako dve ili više nadklasa sadrže istoimenu metodu ili atribut. Tada nije jasno koju metodu tj. atribut će naslediti podklasa.

Nasleđivanje se u Javi označava ključnom reči **“extends”**. Ova reč se navodi pri deklaraciji klase odmah posle naziva klase, a u njenom produžetku sledi naziv klase koja se nasleđuje.

```
class NazivNadklase {  
  
    ...  
  
}  
  
class NazivPodklase extends NazivNadklase {  
  
    ...  
  
}
```

Primer 1

Napraviti klasu **Kalkulator**. Ova klasa bi trebalo da ima:

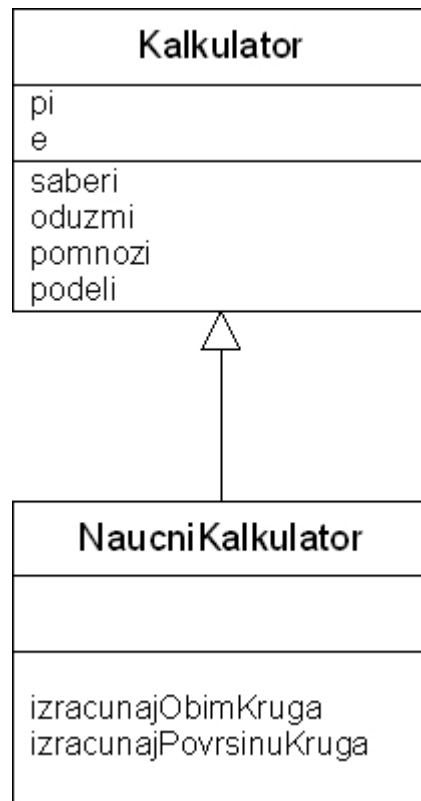
- Atribut **pi** koji ima početnu vrednost 3.14.
- Atribut **e** koji ima početnu vrednost 2.71.
- Metodu **saberi** koja sabira dva realna broja i vraća rezultat.
- Metodu **oduzmi** koja oduzima dva realna broja i vraća rezultat.
- Metodu **pomnozi** koja množi dva realna broja i vraća rezultat.
- Metodu **podeli** koja deli dva realna broja i vraća rezultat.

Napraviti klasu **NaucniKalkulator** koja nasleđuje klasu **Kalkulator** i ima:

- Metodu **izracunajObimKrug** koja kao ulazni parametar prima poluprečnik kruga i vraća

- obim kruga.
- Metodu **izracunajPovrsinuKrug** koja kao ulazni paametar prima poluprečnik kruga i vraća površinu kruga.

Napraviti klasu **TestKalkulator** koja kreira jedan objekat klase **NaucniKalkulator** i poziva sve njegove metode.



Slika 1: Primer grafičkog prikaza nasleđivanja

Na slici (Slika 1) se može videti kako se grafički predstavlja nasleđivanje. U ovom slučaju, klasa **NaucniKalkulator** (podklasa) nasleđuje klasu **Kalkulator** (nadklasa). Klasa **Kalkulator** ima dva atributa i četiri metode, dok klasa **NaucniKalkulator** ima dve svoje metode ali i sve attribute i metode iz klase **Kalkulator**.

```
class Kalkulator {  
  
    double pi = 3.14;  
    double e = 2.71;  
  
    double saberi(double a, double b){  
        return a+b;  
    }  
  
    double oduzmi(double a, double b){  
        return a-b;  
    }  
  
    double pomnozi(double a, double b){  
        return a*b;  
    }  
}
```

```

    }

    double podeli(double a, double b){
        return a/b;
    }
}

class NaucniKalkulator extends Kalkulator {

    //Atribut pi moze da se pozove jer je dobijen
    //nasledjivanjem od klase Kalkulator.
    double izracunajObimKrug(a double poluprecnik){
        return poluprecnik*2*pi;
    }

    double izracunajPovrsinuKrug(a double poluprecnik){
        return poluprecnik*poluprecnik*pi;
    }
}

class TestNaucniKalkulator {

    public static void main(String[] args) {

        NaucniKalkulator nk = new NaucniKalkulator();

        //Pozivanje metoda klase NaucniKalkulator koje su
        //napisane u toj klasi
        System.out.println("Obim: "+nk.izracunajObimKrug(5));
        System.out.println("Povrsina: "+
            nk.izracunajPovrsinuKrug(5));

        //Pozivanje metoda klase NaucniKalkulator koje su
        //nasledjene od klase Kalkulator
        System.out.println("2+3="+nk.saberi(2,3));
        System.out.println("2-3="+nk.oduzmi(2,3));
        System.out.println("2*3="+nk.pomnozi(2,3));
        System.out.println("2/3="+nk.podeli(2,3));
    }
}

```

Klasa NaucniKalkulator nasleđuje klasu Kalkulator. Može se videti da se ključna reč “extends” nalazi odmah posle naziva podklase i da se posle nje nalazi naziv nadklase. U okviru metoda “izracunajObimKrug(a)” i “izracunajPovrsinuKrug(a)” se nalaze pozivi ka atributu “pi”, što je dozvoljeno jer je taj atribut nasleđen iz klase Kalkulator. U okviru “main” metode klase TestKalkulator se vidi da je klasa NaucniKalkulator nasledila i metode “saberi”, “oduzmi”, “pomnozi” i “podeli”, a da nisu morale da budu eksplicitno napisane u okviru klase. Na taj način ne dolazi do ponavljanja koda ovih metoda u obe klase.

Metode iz nadklase se prenose u podklasu u originalnom obliku. Međutim, česta je situacija da se u podklasi zahteva da nasleđena metoda ima malo drugačije ponašanje. U tom slučaju, moguće je

napisati metodu sa istim zaglavljem (naziv, povratna vrednost i ulazni parametri) ali koja će da ima drugačije ponašanje. Ovaj postupak se naziva **redefinisane metode (“overriding”)**. Kada se u okviru podklase bude pozvala željena metoda, izvršiće se redefinisana verzija metode, a ne originalna. Sa obzirom na to da su, u suštini, konstruktori specifična vrsta metoda, moguće je redefinisati i konstruktor u okviru podklase.

Ako redefinisana metoda ili konstruktor samo “proširuju” funkcionalnost originalne metode nadklase, postoji mogućnost da se u okviru redefinisane metode pozove originalna metoda, a onda da se samo dopiše deo koda koji se odnosi na novu funkcionalnost. Pozivanje metode ili konstruktora nadklase se vrši korišćenjem **ključne reči “super”** na sledeći način:

```
//Pozivanje konstruktora nadklase
//koji nema parametre
super();

//Pozivanje konstruktora nadklase
//koji ima parametre
super(...ulazni parametri...);

//Pozivanje metode nadklase
super.nazivMetode(...ulazni parametri...);
```

Pozivanje metoda ili konstruktora nadklase je opciono i ne mora se raditi svaki put već samo kada je neophodno. Pri tome, **konstruktor nadklase se može pozvati samo iz konstruktora podklase** a naredba za poziv mora da bude **na prvoj liniji koda konstruktora podklase**. Ako nadklasa ima eksplicitno definisan parametrizovani konstruktor a nema eksplicitno definisan neparametrizovani konstruktor, onda podklasa mora da ima eksplicitno definisan konstruktor koji poziva neki konstruktor nadklase. Bitno je znati i to da se, pri inicijalizaciji objekta, uvek prvo poziva konstruktor nadklase (i njenih nadklasa), pa tek onda konstruktor podklase.

Primer 2

Napraviti klasu **Osoba** koja ima:

- Atribut ime. Početna vrednost ovog atributa je “N”.
- Atribut prezime. Početna vrednost ovog atributa je “N”.
- Atribut jmbg (String) koji predstavlja matični broj.
- Konstruktor koji kao ulazni parametar prima ime, prezime i jmbg i postavlja vrednosti odgovarajućih atributa samo ako su vrednosti sva tri parametra različita od null. U suprotnom se ispisuje poruka o grešci na ekranu.
- Metodu ispisi koja na ekranu u tri reda ispisuje podatke o imenu, prezimenu i jmbg osobe.

Napraviti klasu **Djak** koja nasleđuje klasu Osoba i ima:

- Atribut prosečnaOcena koji predstavlja prosečnu ocenu u školi (npr. 4,66).
- Konstruktor koji kao ulazni parametar prima ime, prezime, jmbg i prosečnu ocenu đaka i postavlja vrednosti odgovarajućih atributa samo ako su vrednosti prva tri parametra različita od null i ako je prosečna ocena u rasponu od 1 do 5. U suprotnom se ispisuje poruka o grešci na ekranu.
- Redefinisane metodu ispisi koja na ekranu u četiri reda ispisuje podatke o imenu, prezimenu, jmbg i prosečnoj oceni đaka.

Napraviti klasu **Penzioner** koja nasleđuje klasu Osoba iz prethodnog primera i ima:

- Atribut penzija koji predstavlja iznos koji pensioner prima kao penziju (npr. 14200,5

dinara).

- Konstruktor koji kao ulazni parametar prima ime, prezime, jmbg i iznos penzije i postavlja vrednosti odgovarajućih atributa samo ako su vrednosti prva tri parametra različita od null i ako je iznos penzije veći od nule. U suprotnom se ispisuje poruka o grešci na ekranu.
- Redefinisana metoda ispisi koja na ekranu u tri reda ispisuje podatke o imenu, prezimenu i iznosu penzije. Nije potrebno ispisati jmbg penzionera.

Napraviti klasu **TestOsoba** koja kreira po jedan objekat klase Osoba, Djak i Penzioner. Dodeliti osobi ime "Pera Peric" i matični broj "1212007710567". Dodeliti đaku ime "Mika Lazic", matični broj "1010000715076" i prosečnu ocenu 5.0. Dodeliti penzioneru ime "Zika Zikic", maticni broj "0909944710078" i penziju 23400.0 dinara. Ispisati podatke o svoj trojici na ekranu.

```
class Osoba {

    String ime = "N";
    String prezime = "N";
    String jmbg;

    Osoba(String ime, String prezime, String jmbg){
        if(ime!=null && prezime!=null && jmbg!=null){
            this.ime = ime;
            this.prezime = prezime;
            this.jmbg = jmbg;
        }
        else System.out.println("Greska");
    }

    void ispisi(){
        System.out.println("Ime: "+ime);
        System.out.println("Prezime: "+prezime);
        System.out.println("JMBG: "+jmbg);
    }

}

class Djak extends Osoba {

    double prosecnaOcena;

    Djak(String ime, String prezime, String jmbg,
        double prosecnaOcena){
        //Poziva se konstruktor nadklase
        //koriscenjem reci "super".
        super(ime,prezime,jmbg);

        if(prosecnaOcena<=5 && prosecnaOcena>=1)
            this.prosecnaOcena = prosecnaOcena;
        else System.out.println("Greska!");
    }

    void ispisi(){
        //Pozivanje metode "ispisi" nadklase
        //(klase Osoba) koja ce da uradi ispis
```

```

        //prva tri podatka: ime, prezime, jmbg.
        //Na ovaj nacin, kod se ne ponavlja.
        super.ispisi();

        //Ispis prosečne ocene
        System.out.println("Prosečna ocena: "+prosečnaOcena);
    }

}

class Penzioner extends Osoba {

    double penzija;

    Penzioner(String ime, String prezime, String jmbg,
               double penzija) {
        super(ime, prezime, jmbg);

        if (penzija>0) this.penzija = penzija;
        else System.out.println("Greska!");
    }

    void ispisi(){
        //Ne vrsi se poziv metodi "ispisi"
        //nadklase jer ona ispisuje i jmbg
        //a to u ovom slucaju nije potrebno.
        System.out.println("Ime: "+ime);
        System.out.println("Prezime: "+prezime);
        System.out.println("Penzija: "+penzija);
    }

}

class TestOsoba {

    public static void main(String[] args) {

        Osoba o = new Osoba("Pera","Peric",
                           "1212007710567");

        Djak dj = new Djak("Mika","Lazic",
                          "1010000715076",5.0);

        Penzioner p = new Penzioner("Zika","Zikic",
                                     "0909944710078",23400.0);

        //Poziva se metoda "ispisi" klase Osoba.
        o.ispisi();

        //Poziva se redefinisana metoda "ispisi"
        //klase Djak.
        dj.ispisi();
    }
}

```

```

        //Poziva se redefinisana metoda "ispisi"
        //klase Penzioner.
        p.ispisi();
    }
}

```

Iz primera se može uočiti nekoliko stvari. Sa obzirom na to da klasa Djak nasleđuje klasu Osoba, i da klasa Osoba ima eksplicitno napisan parametrizovani konstruktor, i klasa Djak mora da ima eksplicitno napisan konstruktor koji poziva konstruktor nadklase. Ovaj poziv se vrši u prvoj liniji koda konstruktora klase Djak korišćenjem reči “super” (linije koje sadrže komentare se ne računaju kao kod). Ovim se promoviše ponovno korišćenje koda jer originalni konstruktor vrši postavljanje vrednosti za atribute “ime”, “prezime” i “jmbg”. Prema tome, u konstruktor klase Djak je bilo dovoljno dodati kod za proveru i unos prosečne ocene.

U klasi Djak se redefiniše metoda “ispisi”. Sa obzirom na to da originalna metoda (metoda “ispisi” klase Osoba) već ispisuje na ekranu ime prezime i jmbg, redefinisana metoda poziva ovu originalnu metodu a dodatno je napisan samo onaj deo koda koji se odnosi na ispis prosečne ocene.

Redefinisana metoda “ispisi” iz klase Penzioner demonstrira slučaj u kojem nije potrebno pozvati originalnu metodu. Razlog je taj što se ne traži ispis vrednosti svih atributa već samo imena, prezimena i iznosa penzije.

Već je navedeno da podklasa putem nasleđivanja dobija sve attribute i metode nadklase. To znači da podklasa ima sve elemente kao i nadklasa, sa tim što može imati i dodatne attribute i metode. Drugim rečima, podklasa i nadklasa su donekle kompatibilne. Zbog toga je **moгуće promenljivoj tipa nadklase dodeliti objekat podklase**. Ovo je **kompatibilnost klasa**.

```

NazivNadklase objekat1 = new NazivPodklase();

```

Ako se promenljivoj tipa nadklase dodeli objekat podklase, preko te promenljive se **moгу pozivati samo metode nadklase iako je to, u suštini, objekat podklase**. Ako se želi povratiti puna funkcionalnost stvorenog objekta, potrebno je izvršiti eksplicitno konvertovanje u podtip (“class cast”) na sledeći način

```

NazivPodklase objekat2 = (NazivPodklase)(objekat1);

```

Potrebno je napomenuti i sledeću stvar: pozivi metoda podklase preko promenljive tipa nadklase će da **aktiviraju redefinisane metode iz podklase (ako postoje), a ne originalne**. Na ovaj način je u Javi omogućen **polimorfizam** - pojava da se može pristupati različitim tipovima podataka (klasama) preko zajedničkog interfejsa (zajedničke nadklase).

Primer 3

*Napraviti klasu **Vozilo** koja ima:*

- *Atribut naziv čija je početna vrednost “N”.*
- *Metodu postaviNaziv koja kao ulazni parametar prima naziv vozila i postavlja vrednost atributa naziv na tu vrednost.*
- *Metodu ispisi koja na ekranu ispisuje naziv vozila.*

Napraviti klasu **TrkackoVozilo** koja nasleđuje klasu **Vozilo** i ima:

- Atribut **maksimalnaBrzina** čija početna vrednost je 0.0.
- Metodu **postaviMaksimalnuBrzinu** koja kao ulazni parametar prima maksimalnu brzinu i postavlja vrednost atributa **maksimalnaBrzina** na tu vrednost.
- Redefinisanoj metodi **ispisi** koja na ekranu ispisuje naziv vozila i njegovu maksimalnu brzinu.

Napraviti klasu **TestVozila** koja kreira jedan objekat klase **TrkackoVozilo**, ali tako da bude dodeljen promenljivoj tipa **Vozilo**. Naziv vozila bi trebao da bude "Audi S8 GT". Pozvati metodu za ispis i videti šta se dešava. Pokušati sa unosom maksimalne brzine od 350 km/h. Izvršiti konvertovanje već postojećeg objekta u objekat klase **TrkackoVozilo**, i dodeliti maksimalnu brzinu pa ponovo izvršiti ispis.

```
class Vozilo {

    String naziv = "N";

    void postaviNaziv(String naziv) {
        this.naziv = naziv;
    }

    void ispisi() {
        System.out.println("Naziv vozila: "+naziv);
    }

}

class TrkackoVozilo extends Vozilo{

    double maksimalnaBrzina = 0.0;

    void postaviMaksimalnuBrzinu(double maksimalnaBrzina) {
        this.maksimalnaBrzina = maksimalnaBrzina;
    }

    void ispisi() {
        super.ispisi();

        System.out.println("Maksimalna brzina je: "+
                           maksimalnaBrzina);
    }

}

class TestVozilo {

    public static void main(String[] args) {

        Vozilo v = new TrkackoVozilo();

        v.postaviNaziv("Audi S8 GT");

        //Ne moze se pozvati metoda
```



```

        // "postaviMaksimalnuBrzinu" jer je
        // to metoda podklase TrkackoVozilo
        // i ne postoji u nadklasi Vozilo.

        // v.postaviMaksimalnuBrzinu();

        // Pozvace se metoda "ispisi" klase
        // TrkackoVozilo, a ne metoda "ispisi"
        // klase Vozilo.
        v.ispisi();

        // Konvertovanje postojećeg objekta u
        // klasu TrkackoVozilo.
        TrkackoVozilo tv = (TrkackoVozilo)(v);

        // Tek se sada može pozvati metoda
        // postaviMaksimalnuBrzinu.
        tv.postaviMaksimalnuBrzinu(350);
        tv.ispisi();
    }
}

```

U okviru "main" metode klase TestVozilo se kreira jedan objekat klase TrkackoVozilo ali se dodeljuje promenljivoj tipa Vozilo. Ove dve klase su međusobno kompatibilne, pa je ovakvo dodeljivanje dozvoljeno. Jedino ograničenje koje se javlja je to da se sada objekat "gleda kroz fokus klase Vozilo" pa se ne mogu pozivati metode koje ne pripadaju ovoj klasi. Prema tome, poziv metode "postaviMaksimalnuBrzinu" nije moguć. Tek kada se eksplicitno izvrši konvertovanje već postojećeg objekta u objekat klase TrkackoVozilo, moguće je pozvati i ovu metodu.

Potrebno je dodati i to da se pri svakom pozivu metode "ispisi" poziva redefinisana metoda iz klase TrkackoVozilo, a ne originalna metoda "ispisi" klase Vozilo.

Ključna reč "final" ima svoje specifično značenje u kontekstu nasleđivanja. Svaka klasa koja se označi sa ovom reči **ne može da se dalje nasleđuje tj. ne može biti nadklasa**.

```

final class NazivKlase {

    ...

}

// Nije dozvoljeno nasleđivanje prethodne klase

// class NazivKlase2 extends NazivKlase1 {
//
//     ...
//
// }

```

Zadaci

Zadatak 1

Napraviti klasu **KucniAparat** koja ima:

- Atribut marka koji predstavlja naziv proizvođača aparata.
- Atribut model koji predstavlja naziv modela aparata.
- Atribut ukljucen koji ima vrednost TRUE ako je aparat uključen, a u suprotnom FALSE.
- Konstruktor koji kao ulazne argumente prima vrednosti za marku, model i ukljucen (true ili false) i dodeljuje ih atributima. Dodeljivanje se vrši isključivo ako su unete vrednosti za marku i model različiti od null, a u suprotnom se ispisuje poruka o grešci.
- Metodu getMarka koja vraća trenutnu vrednost atributa marka.
- Metodu setMarka koja kao ulazni parametar prima naziv proizvođača i postavlja vrednost atributa marka na tu vrednost. Ako je uneta null vrednost, ispisuje se poruka o grešci na ekranu.
- Metodu getModel koja vraća trenutnu vrednost atributa model.
- Metodu setModel koja kao ulazni parametar prima naziv modela i postavlja vrednost atributa model na tu vrednost. Ako je uneta null vrednost, ispisuje se poruka o grešci na ekranu.
- Metodu ukljuci koja uključuje aparat tj. postavlja vrednost atributa ukljucen na TRUE.
- Metodu iskljuci koja isključuje aparat tj. postavlja vrednost atributa ukljucen na FALSE.
- Metodu ispisi koja na ekranu ispisuje poruku o marki i modelu aparata kao i to da li je aparat uključen.

Napraviti klasu **Televizor** koja nasleđuje klasu KucniAparat i ima:

- Atribut program koji predstavlja broj programa koji se trenutno prikazuje. Početna vrednost za atribut program je 1.
- Konstruktor koji kao ulazne argumente prima vrednosti za marku, model, ukljucen (true ili false) i program i dodeljuje ih atributima. Dodeljivanje se vrši isključivo ako su unete vrednosti za marku i model različiti od null i ako je broj programa između 1 i 40, a u suprotnom se ispisuje poruka o grešci.
- Metodu promeniProgramNavise koja menja program na sledeći. Ako se na televizoru već pušta 40. program, program dobija vrednost 1.
- Metodu promeniProgramNanize koja menja program na prethodni. Ako se na televizoru već pušta 1. program, program dobija vrednost 40.
- Metodu ispisi koja, pored poruke o marki, modelu i tome da li je uključen, ispisuje i poruku o tome koji program se trenutno prikazuje (redefinisati metodu ispisi klase KucniAparat).

Napraviti klasu **Radio** koja nasleđuje klasu KucniAparat i ima:

- Atribut frekvencija koji predstavlja trenutnu frekvenciju koju pušta radio aparat (npr. 87,5). Početna vrednost za ovaj atribut je 87.5.
- Konstruktor koji kao ulazne argumente prima vrednosti za marku, model, ukljucen (true ili false) i frekvenciju i dodeljuje ih atributima. Dodeljivanje se vrši isključivo ako su unete vrednosti za marku i model različiti od null i ako je frekvencija između 87.5 i 108.0, a u suprotnom se ispisuje poruka o grešci.
- Metodu getFrekvencija koja vraća trenutnu vrednost atributa frekvencija.
- Metodu setFrekvencija koja kao ulazni parametar prima novu frekvenciju i postavlja vrednost atributa frekvencija na tu vrednost. Ako je nova vrednost van opsega 87.5 – 108.0, ispisati poruku o grešci na ekranu.
- Redefinisati metodu ispisi klase KucniAparat tako da, ako je radio uključen, ispisuje samo poruku o frekvenciji koju radio pušta. Ako je radio isključen, ova metoda ispisuje poruku o tome da je isključen. Ni u jednom ni u drugom slučaju ne treba ispisivati podatke o marki i modelu radija.

Napraviti klasu **TestKucniAparat** koja kreira po jedan objekat klase KucniAparat, Televizor i Radio. Kucni aparat je marke Philips, model SQ2 i uključen je. Televizor je LG, model LT33, isključen je i pušta 2. program. Radio je marke Sony, model SR, isključen je i pušta radio stanicu na frekvenciji 88.9. Isključiti kućni aparat, uključiti televizor i podesiti ga na 3. program, i uključiti radio i podesiti ga na frekvenciju 105.6. Ispisati podatke o sva tri aparata na ekranu.

Rešenje:

```
class KucniAparat {  
  
    String marka;  
    String model;  
    boolean ukljucen;  
  
    KucniAparat(String marka, String model, boolean ukljucen) {  
        if (marka != null && model != null) {  
            this.marka = marka;  
            this.model = model;  
        }  
    }  
}
```

```

        this.ukljucen = ukljucen;
    }
    else System.out.println("Greska");
}

String getMarka() {
    return marka;
}

void setMarka(String marka) {
    if (marka != null) this.marka = marka;
    else System.out.println("Greska");
}

String getModel() {
    return model;
}

void setModel(String model) {
    if (model != null) this.model = model;
    else System.out.println("Greska");
}

void ukljuci() {
    ukljucen = true;
}

void iskljuci() {
    ukljucen = false;
}

void ispisi() {
    System.out.println("Marka: "+marka);
    System.out.println("Model: "+model);
    if (ukljucen) System.out.println("Aparat je ukljucen");
    else System.out.println("Aparat je iskljucen");
}
}

class Televizor extends KucniAparat {

    int program = 1;

    Televizor(String marka, String model, boolean ukljucen, int program) {
        super(marka, model, ukljucen);
        if (program >= 1 && program <= 40)
            this.program = program;
        else System.out.println("Greska");
    }

    void promeniProgramNavise() {
        if (program < 40) program++;
        else program = 1;
    }

    void promeniProgramNanize() {
        if (program > 1) program--;
        else program = 40;
    }

    void ispisi () {
        super.ispisi();
        System.out.println("Trenutno je pusten "+program+" . program");
    }
}

```

```

    }
}

class Radio extends KucniAparat {

    double frekvencija = 87.5;

    Radio(String marka, String model, boolean ukljucen, double frekvencija) {
        super(marka, model, ukljucen);
        if (frekvencija >= 87.5 && frekvencija <= 108)
            this.frekvencija = frekvencija;
        else System.out.println("Greska");
    }

    double getFrekvencija() {
        return frekvencija;
    }

    void setFrekvencija(double frekvencija) {
        if (frekvencija >= 87.5 && frekvencija <= 108)
            this.frekvencija = frekvencija;
        else System.out.println("Greska");
    }

    void ispisi(){
        if (ukljucen) System.out.println("Trenutna frekvencija je: "+
            frekvencija);
        else System.out.println("Radio je iskljucen");
    }
}

class TestKucniAparat {

    public static void main(String[] args) {

        KucniAparat k = new KucniAparat("Philips", "SQ2", true);

        Televizor t = new Televizor("LG", "LT33", false, 2);

        Radio r = new Radio("Sony", "SR", false, 88.9);

        k.iskljuci();

        t.ukljuci();
        t.promeniProgramNavise();

        r.ukljuci();
        r.setFrekvencija(105.6);

        k.ispisi();
        t.ispisi();
        r.ispisi();

    }
}

```

Zadatak 2

Napraviti klasu **Proizvod** koja ima:

- Atribut naziv.
- Atribut cena (npr. 34.5 dinara).

- Konstruktor bez ulaznih argumenata koji postavlja vrednosti atributa naziv i cena na “nepoznat” i 0.0 respektivno.
- Konstruktor koji kao ulazne argumente prima vrednosti za naziv i cenu i dodeljuje ih atributima samo ako uneti naziv nije null i ako je uneta cena veća od nule. U suprotnom se ispisuje poruka o grešci, a atributi dobijaju vrednosti “nepoznat” i 0.0 respektivno.
- Metodu getNaziv koja vraća trenutnu vrednost atributa naziv.
- Metodu setNaziv koja kao ulazni parametar prima naziv proizvoda i postavlja vrednost atributa naziv na tu vrednost. Ako je uneta null vrednost, ispisuje se poruka o grešci na ekranu.
- Metodu getCena koja vraća trenutnu vrednost atributa cena.
- Metodu setCena koja kao ulazni parametar prima cenu proizvoda i postavlja vrednost atributa cena na tu vrednost. Ako je uneta cena koja je jednaka ili manja od nule, ispisuje se poruka o grešci na ekranu.

Napraviti klasu **PrehambeniProizvod** koja nasleđuje klasu Proizvod i ima:

- Atribut rokTrajanja koji predstavlja datum do kojeg je proizvod upotrebljiv.
- Konstruktor bez ulaznih argumenata koji postavlja vrednosti atributa naziv, cena i rokTrajanja na “nepoznat”, 0.0 i trenutni datum.
- Konstruktor koji kao ulazne argumente prima vrednosti za naziv, cenu i rokTrajanja i dodeljuje ih atributima samo ako uneti naziv nije null, ako je uneta cena veća od nule i ako se rok trajanja odnosi na datum koji je posle trenutnog. U suprotnom se ispisuje poruka o grešci, a atributi dobijaju vrednosti “nepoznat”, 0.0 i trenutni datum.

Napraviti klasu **TestProizvod** koja kreira dva objekta klase PrehambeniProizvod. Prvi objekat ima naziv “hleb”, cenu 30.0 dinara i rok trajanja do “15.12.2007.”. Drugi objekat ima naziv “senf”, cenu 120.5 dinara i rok trajanja koji ističe istog dana kada je i proizveden. Prvo objekat kreirati korišćenjem konstruktora koji ima ulazne argumente, a drugi korišćenjem konstruktora koji nema ulazne argumente. Ispisati nazive i cene oba proizvoda na ekranu.

Rešenje:

```
class Proizvod {

    String naziv;
    double cena;

    Proizvod() {
        naziv = "nepoznat";
        cena = 0.0;
    }

    Proizvod(String naziv, double cena) {
        if (naziv != null && cena > 0) {
            this.naziv = naziv;
            this.cena = cena;
        }
        else {
            this.naziv = "nepoznat";
            this.cena = 0;
            System.out.println("Greska");
        }
    }

    double getCena() {
        return cena;
    }

    void setCena(double cena) {
        if (cena > 0) this.cena = cena;
        else System.out.println("Greska");
    }

    String getNaziv() {
        return naziv;
    }
}
```

```

        void setNaziv(String naziv) {
            if (naziv != null) this.naziv = naziv;
            else System.out.println("Greska");
        }
    }

import java.util.GregorianCalendar;

class PrehrambeniProizvod extends Proizvod {

    GregorianCalendar rokTrajanja;

    PrehrambeniProizvod() {
        super();
        this.rokTrajanja = new GregorianCalendar();
    }

    PrehrambeniProizvod(String naziv, double cena,
        GregorianCalendar rokTrajanja) {
        super(naziv, cena);

        if (rokTrajanja != null &&
            rokTrajanja.after(new GregorianCalendar()))
            this.rokTrajanja = rokTrajanja;
        else {
            this.rokTrajanja = new GregorianCalendar();
            System.out.println("Greska");
        }
    }
}

import java.util.GregorianCalendar;

class TestProizvod {

    public static void main(String[] args) {

        GregorianCalendar datum = new GregorianCalendar();
        datum.set(2007, 11, 15);
        PrehrambeniProizvod p1 =
            new PrehrambeniProizvod("hleb", 30.0, datum);

        PrehrambeniProizvod p2 = new PrehrambeniProizvod();

        p2.setNaziv("senf");
        p2.setCena(120.5);

        System.out.println("Proizvod: "+p1.getNaziv());
        System.out.println("Cena: "+p1.getCena());

        System.out.println("Proizvod: "+p2.getNaziv());
        System.out.println("Cena: "+p2.getCena());

    }
}

```

Apstraktne klase

Već je rečeno da se metoda sastoji iz zaglavlja (povratna vrednost, naziv, ulazni parametri) i tela (kod koji obezbeđuje funkcionalnost). U ovom poglavlju je navedeno da se atributi i metode prenose nasleđivanjem i da je jedan od osnovnih motiva nasleđivanja izbegavanje ponovnog pisanja nekog koda koji je zajednički za više klasa. Međutim, kako postupiti u onim situacijama kada je unapred poznato da će klasa morati da sadrži određene metode (sa već definisanim zaglavljem) ali implementacija tih metoda može da varira u zavisnosti od nekih faktora?

Java sadrži mehanizam apstraktnih klasa koje omogućavaju rešavanje ovakvih problema. **Apstraktne klase** predstavljaju klase koje pored običnih metoda i atributa imaju i jednu ili više apstraktnih metoda. **Apstraktne metode** su one metode koje imaju samo deklaraciju ali ne i telo. **Klasa koja nasleđuje apstraktnu klasu mora da implementira sve apstraktne metode te klase** inače i ona postaje apstraktna. Deklaracija apstraktne klase se vrši korišćenjem **ključne reči “abstract”**. Ova ključna reč se mora napisati i na početku zaglavlja svake apstraktne metode u okviru apstraktne klase, a umesto tela metode se piše znak tačka-zarez (“;”).

```
abstract class NazivApstraktneKlase {  
  
    tip_atributa1 nazivAtributa1;  
    ...  
    tip_atributaN nazivAtributaN;  
  
    tip_povratne_vrednosti nazivMetode1() {  
        ...Telo metode 1...  
    }  
  
    ...  
  
    tip_povratne_vrednosti nazivMetodeN() {  
        ...Telo metode N...  
    }  
  
    abstract tip_povratne_vrednosti nazivApstraktneMetode1();  
  
    ...  
  
    abstract tip_povratne_vrednosti nazivApstraktneMetodeN();  
  
}
```

Veoma je važno napomenuti da se **ne može napraviti objekat (instanca) apstraktne klase**. Razlog je taj što bi poziv apstraktne metode vodio ka nedefinisanom ponašanju. Sledeća naredba nije dozvoljena, i bila bi prijavljena kao greška.

```
NazivApstraktneKlase promenljiva1 = new NazivApstraktneKlase();
```

Sa druge strane, i ovde važi kompatibilnost klasa pa je dozvoljeno promenljivoj koja je tipa apstraktne klase dodeliti objekat podklase (obične Java klase).

```
NazivApstraktneKlase promenljiva1 = new NazivKlase();
```

Svaka apstraktna klasa predstavlja svojevrsan ugovor jer programer koji piše klasu koja nasleđuje apstraktnu klasu mora da implementira zadate apstraktne metode. Podklasa, zauzvrat, nasleđivanjem dobija sve attribute i metode koje nisu apstraktne.

Primer 4

Napraviti apstraktnu klasu **Zaposleni** koja ima:

- Atribut plata koji je realan broj.
- Metodu `getPlata` koja vraća trenutnu vrednost atributa plata.
- Apstraktnu metodu `izracunajPlatu` koja ne vraća ništa, ali ima ulazni argument koji predstavlja broj sati koji je zaposleni radio.

Napraviti klasu **KancelarijskiRadnik** koja nasleđuje klasu **Zaposleni** i:

- Realizuje metodu `izracunajPlatu`. Ova metoda postavlja vrednost atributa plata prema formuli $PLATA = 100 * BROJ_SATI$.

Napraviti klasu **Menadzer** koja nasleđuje klasu **Zaposleni** i:

- Realizuje metodu `izracunajPlatu`. Ova metoda postavlja vrednost atributa plata prema formuli $PLATA = 1000 * BROJ_SATI$.

Napraviti klasu **TestZaposleni** i kreirati po jedan objekat klase **KancelarijskiRadnik** i **Menadzer**. Izracunati i ispisati plate obojice ako se zna da je Menadzer radio 250 sati a kancelarijski radnik 200 sati.

```
abstract class Zaposleni {  
  
    double plata;  
  
    double getPlata() {  
        return plata;  
    }  
  
    abstract void izracunajPlatu (int broj_sati);  
  
}  
  
class KancelarijskiRadnik extends Zaposleni {  
  
    void izracunajPlatu(int broj_sati) {  
        plata = 100 * broj_sati;  
    }  
  
}  
  
class Menadzer extends Zaposleni {  
  
    void izracunajPlatu(int broj_sati) {  
        plata = 1000 * broj_sati;  
    }  
  
}  
  
class TestZaposleni {  
  
    public static void main(String[] args) {
```



```

        KancelarijskiRadnik k = new KancelarijskiRadnik();
        Menadzer m = new Menadzer();

        m.izracunajPlatu(250);
        k.izracunajPlatu(200);

        System.out.println("Plata menadzera je: "+m.getPlata());
        System.out.println("Plata kancelarijskog radnika je: "+
                           k.getPlata());
    }
}

```

Klasa Zaposleni je apstraktna. Pored atributa "plata" i metode "getPlata" ona ima i apstraktnu metodu "izracunajPlatu". Sa obzirom na to da klase KancelarijskiRadnik i Menadzer nasleđuju klasu Zaposleni, one moraju da implementiraju apstraktnu metodu "izracunajPlatu". Plate menadžera i kancelarijskog radnika se izračunavaju na drugačiji način, pa su i implementacije konkretnih metoda "izracunajPlatu" posebne za svaku klasu.

Zadaci

Zadatak 1

Napraviti apstraktnu klasu **GeometrijskoTelo** koja ima:

- Atribut površina koji predstavlja površinu geometrijskog tela.
- Atribut zapremina koji predstavlja zapreminu geometrijskog tela.
- Apstraktnu metodu izracunajPovrsinu koja nema ulazne argumente, niti povratnu vrednost.
- Apstraktnu metodu izracunajZapreminu koja nema ulazne argumente, niti povratnu vrednost.
- Metodu getPovrsina koja vraća trenutnu vrednost površine geometrijskog tela. Pre nego što može da vrati tu vrednost, ova metoda mora da pozove metodu izracunajPovrsinu da izracuna površinu.
- Metodu getZapremina koja vraća trenutnu vrednost zapremine geometrijskog tela. Pre nego što može da vrati tu vrednost, ova metoda mora da pozove metodu izracunajZapreminu da izracuna zapreminu.

Napraviti klasu **Kocka** koja nasleđuje klasu GeometrijskoTelo i ima:

- Atribut duzinaStranice koja predstavlja dužinu stranice kocke.
- Metode getDuzinaStranice i setDuzinaStranice. Duzina stranice mora da bude broj veći od nule. Ispisati poruku o grešci u slučaju narušavanja ovog ograničenja.
- Realizovanu metodu izracunajPovrsinu koja na osnovu dužine stranice (atribut) postavlja vrednost atributa površina po formuli $povrsina = duzinaStranice * duzinaStranice * 6$.
- Realizovanu metodu izracunajZapreminu koja na osnovu dužine stranice (atribut) postavlja vrednost atributa zapremina po formuli $zapremina = duzinaStranice * duzinaStranice * duzinaStranice$.

Napraviti klasu **Kvadar** koja nasleđuje klasu GeometrijskoTelo i ima:

- Atribut duzina koja predstavlja dužinu kvadra.
- Atribut sirina koji predstavlja širinu kvadra.
- Atribut visina koji predstavlja visinu kvadra.
- Odgovarajuće get i set metode za sva tri atributa. Dužina, visina i širina moraju da budu brojevi veći od nule. Ispisati poruku o grešci u slučaju narušavanja ovih ograničenja.
- Realizovanu metodu izracunajPovrsinu koja na osnovu dužine, visine i širine (atributi) postavlja vrednost atributa površina po formuli $povrsina = 2 * (duzina * sirina) + 2 * (duzina * visina) + 2 * (sirina * visina)$.
- Realizovanu metodu izracunajZapreminu koja na osnovu dužine, visine i širine (atributi) postavlja vrednost atributa zapremina po formuli $zapremina = duzina * sirina * visina$.

Napisati klasu **TestGeometrijskihTela** koja kreira jedan objekat klase Kocka i jedan objekat klase Kvadar. Postaviti dužinu stranice kocke na 25.5cm. Postaviti dužinu, širinu i visinu kvadra na vrednosti 20 10 i 30 respektivno. Ispisati površine i zapremine kocke i kvadra na ekranu.

Rešenje:

```

abstract class GeometrijskoTelo {

    double povrsina;
    double zapremina;

    abstract void izracunajPovrsinu();

    abstract void izracunajZapreminu();

    double getPovrsina() {
        izracunajPovrsinu();
        return povrsina;
    }

    double getZapremina() {
        izracunajZapreminu();
        return zapremina;
    }

}

class Kocka extends GeometrijskoTelo {

    double duzinaStranice;

    double getDuzinaStranice() {
        return duzinaStranice;
    }

    void setDuzinaStranice(double duzinaStranice) {
        if (duzinaStranice > 0)
            this.duzinaStranice = duzinaStranice;
        else System.out.println("Greska");
    }

    void izracunajPovrsinu() {
        povrsina = duzinaStranice*duzinaStranice*6;
    }

    void izracunajZapreminu() {
        zapremina = duzinaStranice*duzinaStranice*duzinaStranice;
    }

}

class Kvadar extends GeometrijskoTelo {

    double duzina;
    double sirina;
    double visina;

    double getDuzina() {
        return duzina;
    }

    void setDuzina(double duzina) {
        if (duzina > 0) this.duzina = duzina;
        else System.out.println("Greska");
    }

    double getSirina() {
        return sirina;
    }

}

```

```

    void setSirina(double sirina) {
        if (sirina > 0) this.sirina = sirina;
        else System.out.println("Greska");
    }

    double getVisina() {
        return visina;
    }

    void setVisina(double visina) {
        if (visina > 0) this.visina = visina;
        else System.out.println("Greska");
    }

    void izracunajPovrsinu() {
        povrsina = 2*(duzina*sirina)+2*(duzina*visina)+2*(sirina*visina);
    }

    void izracunajZapreminu() {
        zapremina = duzina*visina*sirina;
    }
}

class TestGeometrijskihTela {

    public static void main(String[] args) {

        Kocka ko = new Kocka();

        ko.setDuzinaStranice(25.5);

        Kvadar kv = new Kvadar();

        kv.setDuzina(20);
        kv.setSirina(10);
        kv.setVisina(30);

        System.out.println("Povrsina kocke je: "+ko.getPovrsina());
        System.out.println("Zapremina kocke je: "+ko.getZapremina());

        System.out.println("Povrsina kvadra je: "+kv.getPovrsina());
        System.out.println("Zapremina kvadra je: "+kv.getZapremina());

    }

}

```

Zadatak 2

Napraviti apstraktnu klasu **Vozilo** koja ima:

- Atribut marka koji predstavlja marku vozila.
- Atribut model koji predstavlja model vozila.
- Odgovarajuće get i set metode za ova dva polja. Nedoovoljene vrednosti za ova dva polja su null Stringovi. U slučaju da se pokuša sa unosom ovih nedovoljenih vrednosti, potrebno je ispisati poruku o grešci na ekranu.
- Apstraktnu metodu daLiIdePoKopnu koja vraća boolean vrednost koja označava da li vozilo može da se kreće po kopnu.
- Apstraktnu metodu daLiIdePoVodi koja vraća boolean vrednost koja označava da li vozilo može da se kreće po vodi.
- Apstraktnu metodu daLiIdePoVazduhu koja vraća boolean vrednost koja označava da li vozilo može da leti.
- Metodu ispiši koja na ekranu ispisuje podatke o vozilu – marku, model i da li ide po kopnu, vodi ili vazduhu.

Napraviti klasu **Automobil** koja nasleđuje klasu Vozilo i ima:

- Implementiranu metodu daLiIdePoKopnu koja uvek vraća TRUE jer automobil može da se kreće po kopnu.
- Implementiranu metodu daLiIdePoVodi koja uvek vraća FALSE jer automobil ne može da se kreće po vodi.
- Implementiranu metodu daLiIdePoVazduhu koja uvek vraća FALSE jer automobil ne može da leti.

Napraviti klasu **Avion** koja nasleđuje klasu Vozilo i ima:

- Implementiranu metodu daLiIdePoKopnu koja uvek vraća FALSE jer avion ne može da se kreće po kopnu.
- Implementiranu metodu daLiIdePoVodi koja uvek vraća FALSE jer avion ne može da se kreće po vodi.
- Implementiranu metodu daLiIdePoVazduhu koja uvek vraća TRUE jer avion može da leti.
- Redefinisanu metodu ispisi koja ispisuje na ekranu poruku o tome da je u pitanju avion i podatke o marki i modelu aviona.

Napraviti klasu **TestVozila** koja kreira po jedan objekat klase Automobil i Avion. Automobil je marke Mercedes, model E220, dok je avion Boeing model 747. Ispisati podatke o oba vozila na ekranu.

Rešenje:

```
abstract class Vozilo {

    String marka;
    String model;

    String getMarka() {
        return marka;
    }
    void setMarka(String marka) {
        if (marka != null) this.marka = marka;
        else System.out.println("Greska");
    }
    String getModel() {
        return model;
    }
    void setModel(String model) {
        if (model != null) this.model = model;
        else System.out.println("Greska");
    }

    abstract boolean daLiIdePoKopnu();

    abstract boolean daLiIdePoVodi();

    abstract boolean daLiIdePoVazduhu();

    void ispisi() {
        System.out.println("Marka vozila: "+marka);
        System.out.println("Model vozila: "+model);

        if (daLiIdePoKopnu())
            System.out.println("Vozilo moze da ide po kopnu");

        if (daLiIdePoVodi())
            System.out.println("Vozilo moze da ide po vodi");

        if (daLiIdePoVazduhu())
            System.out.println("Vozilo moze da leti");
    }
}

class Automobil extends Vozilo {

    boolean daLiIdePoKopnu() {
        return true;
    }
}
```

```

        boolean daLiIdePoVazduhu() {
            return false;
        }

        boolean daLiIdePoVodi() {
            return false;
        }
    }

    class Avion extends Vozilo {

        boolean daLiIdePoKopnu() {
            return false;
        }

        boolean daLiIdePoVazduhu() {
            return true;
        }

        boolean daLiIdePoVodi() {
            return false;
        }

        void ispisi(){
            System.out.println("U pitanju je avion");
            System.out.println("Marka aviona: "+getMarka());
            System.out.println("Model aviona: "+getModel());
        }
    }

    class TestVozila {

        public static void main(String[] args) {

            Automobil a = new Automobil();

            a.setMarka("Mercedes");
            a.setModel("E220");

            Avion av = new Avion();

            av.setMarka("Boeing");
            av.setModel("747");

            a.ispisi();
            av.ispisi();
        }
    }
}

```

Interfejsi

Interfejsi predstavljaju korak dalje u odnosu na apstraktne klase. Kod interfejsa su **sve metode apstraktne** tj. nemaju telo već samo deklaraciju. Zbog toga se ispred metode u interfejsu čak **ni ne navodi ključna reč “abstract”**. Kao i apstraktne klase, ni interfejsi **ne mogu da imaju pojavljivanja**. Pored toga što nemaju nijednu konkretnu metodu, interfejsi **ne mogu da imaju ni attribute već samo konstante**. Deklaracija interfejsa se vrši korišćenjem **ključne reči “interface”**.

```
interface NazivInterfejsa {

    tip_povratne_vrednosti nazivMetode1();

    ...

    tip_povratne_vrednosti nazivMetodeN();

}
```

Intefejs je koncept uz pomoću koga se razdvaja specifikacija metoda (deklaracija) od njihove implementacije. Svaka klasa koja implementira interfejs obavezno mora da sadrži implementacije svih njegovih metoda. Jedna klasa može istovremeno da implementira više interfejsa, i to se označava uz pomoć ključne reči “implements”.

```
class NazivKlase implements NazivInterfejsa1, NazivInterfejsa2... {

    ...

}
```

Moguće su čak i situacije u kojima klasa nasleđuje neku drugu klasu i istovremeno implementira jedan ili više interfejsa.

```
class NazivKlase1 extends NazivKlase2 implements NazivInterfejsa1,
NazivInterfejsa2... {

    ...

}
```

Još je potrebno napomenuti i to da metode koje implementiraju metode iz interfejsa moraju da budu javne tj. označene ključnom reči “public” (više o paketima i nivoima pristupa u narednom poglavlju). Iako interfejsi nisu klase u uobičajenom smislu, i kod njih važi kompatibilnost tipova. U kontekstu interfejsa to znači da se promenljivoj koja je tipa nekog interfejsa uvek može proslediti objekat bilo koje klase koja nasleđuje taj interfejs. Naravno, u tom slučaju se mogu pozivati samo one metode koje se nalaze u interfejsu.

Primer 5

Napraviti interfejs *AutomatNovcaInterfejs* koji ima:

- Metodu *uloziNovac* koja ne vraća ništa, a ima kao ulazni argument iznos koji se želi uneti (npr. 123,45 din).
- Metodu *podigniNovac* koja ne vraća ništa, a ima kao ulazni argument iznos koji se želi podići (npr. 123,45 din).

Napraviti klasu *AutomatNovca* koja implementira interfejs *AutomatNovcaInterfejs* i ima:

- Atribut *stanje* koji predstavlja količinu novca koja se nalazi u automatu. Početno stanje je 5000.0 din.
- Implementiranu metodu *uloziNovac*. Ova metoda vrši ulaganje tj. dodavanje unetog iznosa na trenutno stanje samo ako je uneti iznos veći od nule. U suprotnom, ispisuje se poruka o grešci.
- Implementiranu metodu *podigniNovac*. Ova metoda vrši podizanje željenog iznosa tj. smanjivanje stanja samo ako je uneti iznos veći od nule i ako u automatu ima dovoljno novca. U suprotnom, ispisuje se poruka o grešci.
- Metodu *ispisi* koja ispisuje koliko još novca ima u automatu.

Napraviti klasu *NapredniAutomatNovca* koja implementira interfejs *AutomatNovcaInterfejs* i ima:

- *Atribut stanje koji predstavlja količinu novca koja se nalazi u automatu. Početno stanje je 20000.0 din.*
- *Implementiranu metodu uloziNovac. Ova metoda vrši ulaganje tj. dodavanje unetog iznosa na trenutno stanje. Maksimalni iznos koji se može uplatiti je 20.000 dinara. Ako se ovaj iznos prekorači, potrebno je ispisati poruku o tome da je maksimalni iznos koji se može uplatiti 20.000 dinara. Ako je uneta vrednost za iznos manja od nule ispisuje se poruka o grešci.*
- *Implementiranu metodu podigniNovac. Ova metoda vrši podizanje željenog iznosa tj. smanjivanje stanja. Maksimalni iznos koji se može podići je 10.000 dinara. Ako se ovaj iznos prekorači, potrebno je ispisati poruku o tome da je maksimalni iznos koji se može podići 10.000 dinara. Ako je uneta vrednost za iznos manja od nule ispisuje se poruka o grešci.*
- *Metodu ispisi koja ispisuje koliko još novca ima u automatu i koji su maksimalni iznosi koji se mogu uplatiti tj. podići u jednoj transakciji.*

Napraviti klasu **TestAutomatNovca** koja pravi po jedan objekat klase **AutomatNovca** i **NapredniAutomatNovca** i ulaže u prvi 3000 din, a iz drugog podiže 9.999 dinara. Ispisati stanje svakog automata novca.

```
public interface AutomatNovcaInterfejs {

    public void uloziNovac(double iznos);

    public void podigniNovac (double iznos);

}

public class AutomatNovca implements AutomatNovcaInterfejs{

    double stanje = 5000.0;

    public void uloziNovac(double iznos){
        if (iznos > 0) stanje += iznos;
        else System.out.println("Greska");
    }

    public void podigniNovac(double iznos){
        if (iznos > 0 && stanje >= iznos) stanje -= iznos;
        else System.out.println("Greska");
    }

    void ispisi(){
        System.out.println(
            "U automatu ima jos "+stanje+" dinara");
    }

}

public class NapredniAutomatNovca {

    double stanje = 20000.0;

    public void uloziNovac(double iznos){
```

```

        if (iznos > 0) {
            if (iznos <= 20000) stanje += iznos;
            else System.out.println("Maksimalni iznos"+
                " koji se moze uplatiti je 20000 dinara");
        }
        else System.out.println("Greska");
    }

    public void podigniNovac(double iznos){
        if (iznos > 0 && stanje >= iznos){
            if (iznos <= 10000) stanje -= iznos;
            else System.out.println("Maksimalni iznos"+
                " koji se moze podici je 10000 dinara");
        }
        else System.out.println("Greska");
    }

    void ispisi(){
        System.out.println(
            "U automatu ima jos "+stanje+" dinara");
        System.out.println(
            "Maksimalni iznos koji se moze uplatiti je 20000 dinara");
        System.out.println(
            "Maksimalni iznos koji se moze podici je 10000 dinara");
    }
}

public class TestAutomatNovca {

    public static void main(String[] args) {

        AutomatNovca a = new AutomatNovca();

        a.uloziNovac(3000);

        NapredniAutomatNovca na = new NapredniAutomatNovca();

        na.podigniNovac(9999);

        a.ispisi();
        na.ispisi();
    }
}

```

Klase AutomatNovca i NapredniAutomatNovca implementiraju interfejs AutomatNovcaInterfejs, pa moraju da sadrže metode “uloziNovac” i “podigniNovac” koje moraju da budu označene kao javne (“public”).

Zadaci

Zadatak 1

Napraviti interfejs **SluzbaNabavke** koji ima:

- Metodu dodajRobu koja ne vraća ništa, a kao ulazni argument prima broj komada određenog proizvoda koji je potrebno dodati.

Napraviti interfejs **SluzbaProdaje** koji ima:

- Metodu prodajRobu koja ne vraća ništa, a kao ulazni argument prima broj komada određenog proizvoda koji je potrebno prodati.

Napraviti klasu **Prodavnica** koja implementira interfejse SluzbaNabavke i SluzbaProdaje i ima:

- Atribut brojProizvoda koji predstavlja koliko ima komada određenog proizvoda u prodavnici. Početna vrednost za ovaj atribut je 0.
- Implementiranu metodu dodajRobu koja kao ulazni argument prima broj komada određenog proizvoda koji je potrebno dodati i dodaje ga na brojProizvoda. Dodavanje se vrši samo ako je vrednost ulaznog argumenta veća od nule, a u suprotnom se ispisuje poruka o grešci.
- Implementiranu metodu prodajRobu koja kao ulazni argument prima broj komada određenog proizvoda koji je potrebno prodati i oduzima ga od atributa brojProizvoda. Prodaja se vrši samo ako je vrednost ulaznog argumenta veća od nule i ako ima dovoljno proizvoda u prodavnici, a u suprotnom se ispisuje poruka o grešci.
- Metodu ispiši koja ispisuje koliko ima proizvoda u prodavnici.

Napraviti klasu **Skladiste** koja implementira interfejs SluzbaNabavke i ima:

- Atribut stanjeNaSkladistu koji predstavlja koliko ima komada određenog proizvoda u prodavnici. Početna vrednost za ovaj atribut je 1000.
- Implementiranu metodu dodajRobu koja kao ulazni argument prima broj komada određenog proizvoda koji je potrebno dodati i dodaje ga na stanjeNaSkladistu. Dodavanje se vrši samo ako je vrednost ulaznog argumenta veća od nule, a u suprotnom se ispisuje poruka o grešci.
- Metodu skiniSaStanja koja kao ulazni argument prima broj komada određenog proizvoda koji je potrebno skinuti sa stanja i oduzima ga od atributa stanjeNaSkladistu. Skidanje sa stanja se vrši samo ako je vrednost ulaznog argumenta veća od nule i ako ima dovoljno proizvoda u skladištu, a u suprotnom se ispisuje poruka o grešci.

Napraviti klasu **Dobavljac** koja ima:

- Metodu izvršiIsporuku koja kao ulazni argument dobija interfejs SluzbaNabavke i broj proizvoda koji je potrebno isporučiti. Isporuka robe se vrši pozivom metode dodajRobu interfejsa SluzbaNabavke sa brojem proizvoda kao ulaznim argumentom.

Napraviti klasu **TestProdavnica** koja kreira po jedan objekat klase Prodavnica, Skladiste i Dobavljac. Ispisati koliko ima proizvoda u prodavnici, a onda dodati robu (100 komada) u prodavnicu korišćenjem metode izvršiIsporuku klase Dobavljac. Potom je potrebno dodati robu (500 komada) u skladište korišćenjem metode izvršiIsporuku klase Dobavljac. Ponovo ispisati koliko ima proizvoda u prodavnici.

Rešenje:

```
public interface SluzbaNabavke {

    public void dodajRobu(int kolicina);

}

public interface SluzbaProdaje {

    public void prodajRobu(int kolicina);

}

public class Prodavnica implements SluzbaProdaje, SluzbaNabavke {

    int brojProizvoda = 0;

    public void prodajRobu(int kolicina) {
        if (kolicina > 0 && brojProizvoda >= kolicina)
            brojProizvoda -= kolicina;
        else System.out.println("Greska");
    }
}
```

```

    }

    public void dodajRobu(int kolicina) {
        if (kolicina > 0) brojProizvoda += kolicina;
        else System.out.println("Greska");
    }

    public void ispisi(){
        System.out.println("U prodavnici ima: "+
            brojProizvoda+" proizvoda");
    }
}

public class Skladiste implements SluzbaNabavke {

    int stanjeNaSkladistu = 1000;

    public void dodajRobu(int kolicina) {
        if (kolicina > 0) stanjeNaSkladistu += kolicina;
        else System.out.println("Greska");
    }

    public void skiniSaStanja(int kolicina) {
        if (kolicina > 0 && stanjeNaSkladistu >= kolicina)
            stanjeNaSkladistu -= kolicina;
        else System.out.println("Greska");
    }
}

public class Dobavljac {

    //Ovako napisana metoda izvrši isporuku omogućava klasi Dobavljac
    //da vrši isporuku bilo kakvoj prodavnici tj. skladistu. Jedino je
    //bitno da ta prodavnica tj. klasa kojom se predstavlja implementira
    //interfejs SluzbaNabavke.
    public void izvršiIsporuku(SluzbaNabavke n, int kolicina){
        n.dodajRobu(kolicina);
    }
}

public class TestProdavnica {

    //Ovo je primer kako ista metoda može da opslužuje više klasa ako
    //se koriste interfejsi. Kada interfejsi ne bi bili korisćeni,
    //morale bi da se u okviru klase Dobavljac napisu dve metode
    //izvršiIsporuku koje bi, u sustini, radile isto samo bi jedna
    //bila zadužena za klasu Skladiste a druga za klasu Prodavnica.
    public static void main(String[] args){

        Prodavnica p = new Prodavnica();

        Dobavljac d = new Dobavljac();

        Skladiste s = new Skladiste();

        p.ispisi();

        //Metodi izvršiIsporuku se može kao ulazni argument proslediti
        //objekat klase Prodavnica jer ova klasa implementira interfejs
        //SluzbaNabavke. Onda se u okviru metode izvršiIsporuku preko
        //interfejsa SluzbaNabavke poziva metoda dodajRobu klase Prodavnica.
    }
}

```

```

        d.izvrsiIsporuku(p, 100);

        //Metodi izvrsiIsporuku se moze kao ulazni argument proslediti
        //objekat klase Skladiste jer ova klasa implementira interfejs
        //SluzbaNabavke. Onda se u okviru metode izvrsiIsporuku preko
        //interfejsa SluzbaNabavke poziva metoda dodajRobu klase Skladiste.
        d.izvrsiIsporuku(s, 500);

        p.ispisi();
    }

}

```

Zadatak 2

Napraviti apstraktnu klasu **MuzickiUredjaj** koja ima:

- Atribut ukljucen koji ima vrednost TRUE ako je uređaj ukljucen, a FALSE ako nije.
- Atribut jacinaTona koji predstavlja jačinu tona na uređaju. Raspon vrednosti za ovaj atribut je od 0 do 40. Početna vrednost je 0.
- Metodu ukljuci koja uključuje uređaj.
- Metodu iskljuci koja isključuje uređaj.
- Metodu pojačajTon koja pojačava ton za 1 (ako je ton bio 5, posle poziva metode ima vrednost 6). Pojačavanje tona se vrši samo ako je ton manji od 40.
- Metodu smanjiTon koja smanjuje ton za 1 (ako je ton bio 5, posle poziva metode ima vrednost 4). Smanjivanje tona se vrši samo ako je ton veći od 0.
- Apstraktnu metodu ispisi koja ništa ne vraća a nema ni ulaznih argumenata.

Napraviti interfejs **RadioKomponenta** koji ima:

- Metodu promeniStanicu koja ne vraća ništa ali ako ulazni argument prima novu frekvenciju radio stanice.

Napraviti interfejs **CDKomponenta** koji ima:

- Metodu pustiPesmu koja ne vraća ništa ali kao ulazni argument prima broj pesme na CD-u koji je potrebno pustiti.

Napraviti klasu **MuzickiStub** koja nasleđuje klasu MuzickiUredjaj, implementira interfejsse RadioKomponenta i CDKomponenta i ima:

- Atribut frekvencija koji predstavlja trenutnu frekvenciju radio stanice. Početna vrednost za ovaj atribut je 87.5.
- Atribut brojPesme koji predstavlja broj pesme sa CD-a koju muzički stub trenutno pušta.
- Atribut ukljucenRadio koji ima vrednost TRUE ako muzički stub trenutno pušta muziku sa radija, a inače FALSE.
- Atribut ukljucenCD koji ima vrednost TRUE ako muzički stub trenutno pušta muziku sa CD-a, a inače FALSE.
- Implementiranu metodu promeniStanicu koja uključuje radio komponentu muzičkog stuba (postavlja vrednost atributa ukljucenRadio na TRUE), isključuje CD komponentu (postavlja vrednost atributa ukljucenCD na FALSE), i postavlja vrednost atributa frekvencija na unetu vrednost. Dodeljivanje nove vrednosti za frekvenciju se vrši samo ako je uneta frekvencija u rasponu 87.5-108.0. U suprotnom, potrebno je dodeliti frekvenciju 87.5.
- Implementiranu metodu pustiPesmu koja uključuje CD komponentu muzičkog stuba (postavlja vrednost atributa ukljucenCD na TRUE), isključuje radio komponentu (postavlja vrednost atributa ukljucenRadio na FALSE), i postavlja vrednost atributa brojPesme na unetu vrednost. Dodeljivanje nove vrednosti za broj pesme se vrši samo ako je uneti broj u rasponu 1 do 21. U suprotnom, potrebno je pustiti prvu pesmu sa CD-a.
- Implementiranu metodu ispisi koja na ekranu ispisuje poruku o tome da li je muzički stub uključen. Ako jeste, metoda ispisuje i poruku o tome koja je trenutna jačina tona. Pored toga, ova metoda bi trebalo da, ako je uključena radio komponenta muzičkog stuba, ispiše frekvenciju radio stanice, a ako je uključen CD broj pesme koja se trenutno pušta. Ako stub nije uključen ne ispisuje se ništa drugo osim poruke da je isključen.

Napraviti klasu **TestMuzickiStub** koja kreira jedan objekat klase MuzickiStub, uključuje ga, pojačava ton za 2, podešava frekvenciju radija na 88.9 i ispisuje na ekranu podatke o muzičkom stubu.

Rešenje:

```
public abstract class MuzickiUredjaj {
```

```

    boolean ukljucen;
    int jacinaTona = 0;

    void ukljuci() {
        ukljucen = true;
    }

    void iskljuci() {
        ukljucen = false;
    }

    void pojačajTon() {
        if (jacinaTona < 40) jacinaTona++;
    }

    void smanjiTon() {
        if (jacinaTona > 0) jacinaTona--;
    }

    abstract void ispisi();
}

public interface RadioKomponenta {

    void promeniStanicu(double frekvencija);
}

public interface CDKomponenta {

    void pustiPesmu (int brojPesme);
}

public class MuzickiStub extends MuzickiUredjaj implements RadioKomponenta,
    CDKomponenta {

    double frekvencija = 87.5;
    int brojPesme;
    boolean ukljucenRadio;
    boolean ukljucenCD;

    public void promeniStanicu(double frekvencija) {
        ukljucenRadio = true;
        ukljucenCD = false;
        if (frekvencija >= 87.5 && frekvencija <=108.0)
            this.frekvencija = frekvencija;
        else
            this.frekvencija = 87.5;
    }

    public void pustiPesmu(int brojPesme) {
        ukljucenCD = true;
        ukljucenRadio = false;
        if (brojPesme >= 1 && brojPesme <= 21)
            this.brojPesme = brojPesme;
        else
            this.brojPesme = 1;
    }

    void ispisi() {

```

```

        if (uključen) {
            System.out.println("Muzicki stub je uključen");
            System.out.println("Jacina tona je: "+jacinaTona);
            if (uključenRadio)
                System.out.println("Uključen je radio i pusta stanicu na "+
                                    frekvencija+" MHz");
            if (uključenCD)
                System.out.println("Uključen je CD i pusta "+
                                    brojPesme+". pesmu");
        }
        else System.out.println("Muzicki stub je isključen");
    }
}

public class TestMuzickiStub {

    public static void main(String[] args) {

        MuzickiStub ms = new MuzickiStub();

        ms.ukljuci();
        ms.pojacajTon();
        ms.pojacajTon();
        ms.promeniStanicu(88.9);
        ms.ispisi();
    }
}

```

Klasa Object

Klasa **Object je, u Javi, nadklasa svih klasa**. Iako se ni u jednoj Java klasi eksplicitno ne navodi ovo nasleđivanje ono je uvek prisutno. Prirodna posledica toga je da **promenljiva tipa Object može da primi instancu bilo koje druge Java klase** (kompatibilnost tipova). Ovo može da bude izuzetno korisno u onim situacijama kada nije unapred poznato kog će tipa biti neka promenljiva ili ulazni parametar (pogledati “equals” metodu dalje u tekstu).

Klasa Object nije ni apstraktna klasa ni interfejs već obična klasa sa konkretnim atributima i metodama. U narednih nekoliko pasusa će detaljno biti objašnjene samo dve metode koje se često koriste: “toString” i “equals”.

Ideja metode **“toString”** je da vrati vrednosti svih atributa klase spojene u jedan String. Na ovaj način se lako i brzo može prikazati unutrašnje stanje svakog objekta na ekranu. U originalnoj formi, ova metoda vraća samo String sa nazivom klase i memorijskom adresom objekta koji je pozvao metodu. Drugim rečima, da bi metoda “toString” bila korisna, **potrebno ju je redefinisati** tako da vraća String sa vrednostima odgovarajućih atributa konkretne klase u kojoj se nalazi. Zaglavlje ove metode je uvek isto i glasi:

```
public String toString()
```

Metoda **“equals”** je zamišljena tako da poredi dva objekta iste klase i vraća “true” ako su jednaki, a “false” ako nisu. Jednakost dva objekta iste klase zavisi od toga o kojoj klasi se radi: dva automobila su “jednaka” ako imaju istu registraciju, dve osobe su “jednake” ako imaju isti matični broj, itd. U svojoj originalnoj formi, ova metoda poredi memorijske lokacije oba objekta i smatra ih jednakim ako se nalaze na istoj poziciji u memoriji. Slično kao i sa “toString” metodom, i “equals”

metodu je potrebno redefinisati da bi vršila upoređivanje na pravi način. Ova metoda je već korišćena u prethodnim primerima kada je bilo potrebno porediti dve String vrednosti. Redefinisana varijanta “equals” metode iz klase String poredi dve String vrednosti slovo po slovo i vraća “true” ako su jednaki, a “false” ako nisu. Zaglavlje “equals” metode glasi ovako:

```
public boolean equals (Object o)
```

Može se videti da metoda kao ulazni parametar ima objekat tipa Object. Ovo je i dobro i loše. Dobro je zbog toga što se metodi može proslediti objekat bilo koje klase kao ulazni parametar (Object je nadklasa svih klasa) pa je ovakvo zaglavlje odgovarajuće za bilo koju klasu. Loše je zbog toga što se, pre bilo kakve provere, uneti objekat mora eksplicitno konvertovati u odgovarajući tip.

Kada se govori o konvertovanju tipova, potrebno je napomenuti i da postoji operator kojim se može proveriti koje je klase neki objekat. U pitanju je **“instanceof” operator**. Sledeći izraz vraća “true” ako je objekat zaista te klase, a “false” ako nije.

```
objekat instanceof NazivKlase
```

Šta se zapravo dobija korišćenjem ove dve metode? One bi trebalo da posluže kao konvencija tj. orijentir pri programiranju: ako se žele uporediti dva objekta, potrebno je pozvati “equals” metodu a ako se želina ekranu ispisati unutrašnje stanje nekog objekta, potrebno je pozvati “toString” metodu.

Primer 6

Napraviti klasu **Knjiga** koja ima:

- Atribut naslov.
- Atribut autor koji sadrži ime i prezime autora.
- Atribut ISBN koji predstavlja jedinstveni identifikator knjige (String).
- Metode getNaslov, getAutor i getISBN koje vraćaju vrednosti odgovarajućih atributa.
- Metode setNaslov, setAutor i setISBN koje kao ulazne parametre dobijaju nove vrednosti za odgovarajuće attribute i postavljaju ih.
- Redefinisanu metodu toString klase Object koja vraća jedan String. Ovaj String sadrži podatke o knjizi: naslov, autor i ISBN uz odgovarajući tekst.
- Redefinisanu metodu equals klase Object koja kao ulazni argument prima objekat klase Object, ali se smatra da će se zaista unositi objekti klase Knjiga. Ova metoda vraća true ako je vrednost atributa ISBN jednaka ISBN knjige koja je uneta kao ulazni argument. U suprotnom, metoda vraća false.

Napraviti klasu **TestKnjiga** koja kreira dva objekta klase Knjiga. Prva knjiga je “Istorija lepote”, njen autor je Umberto Eko, a ISBN je 0-234-567. Druga knjiga je “Istorija ruznoce”, njen autor je takođe Umberto Eko, a ISBN je 0-567-890. Proveriti da li je u pitanju ista knjiga i ispisati poruku o tome. Nakon toga, ispisati podatke o svakoj knjizi na ekranu.

```
public class Knjiga {  
  
    String naslov;  
    String autor;  
    String ISBN;  
  
    public String getAutor() {  
        return autor;  
    }  
}
```

```

    }
    public void setAutor(String autor) {
        if (autor != null) this.autor = autor;
        else System.out.println("Greska");
    }
    public String getISBN() {
        return ISBN;
    }
    public void setISBN(String isbn) {
        if (isbn != null) ISBN = isbn;
        else System.out.println("Greska");
    }
    public String getNaslov() {
        return naslov;
    }
    public void setNaslov(String naslov) {
        if (naslov != null) this.naslov = naslov;
        else System.out.println("Greska");
    }
}

public String toString(){
    return "Naslov: '"+naslov+"' Autor: "+autor+" ISBN "+ISBN;
}

//"equals" metoda poredi dva objekta iste klase i vraca true
//ako su jednaki a false ako nisu.
public boolean equals (Object o){
    //Ovako se vrši konvertovanje (cast) iz klase Object u
    //klasu Knjiga. Konvertovanje je moguće samo ako se kao
    //ulazni argument zaista unese objekat klase Knjiga. Ako
    //se unese objekat bilo koje druge klase, metoda će da
    //prekine sa radom i pojaviće se greska.
    Knjiga k = (Knjiga) (o);

    if (ISBN.equals(k.getISBN())) return true;
    else return false;
}

}

public class TestKnjiga {

    public static void main(String[] args) {

        Knjiga k1 = new Knjiga();

        k1.setNaslov("Istorija lepote");
        k1.setAutor("Umberto Eko");
        k1.setISBN("0-234-567");

        Knjiga k2 = new Knjiga();

        k2.setNaslov("Istorija ruznoce");
    }
}

```

```

        k2.setAutor("Umberto Eko");
        k2.setISBN("0-567-890");

        if (k1.equals(k2)) System.out.println("Knjige su iste");
        else System.out.println("Knjige nisu iste");

        System.out.println(k1.toString());
        System.out.println(k2.toString());

        //Moglo je i ovako jer bi se metoda
        //"toString" automatski pozvala
        //System.out.println(k1);
        //System.out.println(k2);

    }

}

```

Zadaci

Zadatak 1

Napraviti klasu **Monitor** koja ima:

- Atribut marka
- Atribut model
- Atribut velicinaEkрана koji predstavlja dužinu dijagonale ekrana u inčima (npr. 17").
- Odgovarajuće get i set metode za ova tri atributa. Nedoovoljene vrednosti za marku i model su null Stringovi a za dijagonalu svi brojevi manji od 12. U slučaju unosa ovih nedovoljenih vrednosti, potrebno je ispisati poruku o grešci na ekranu.
- Redefinisanu metodu toString klase Object koja vraća jedan String. Ovaj String sadrži sve podatke o monitoru: marka, model i dijagonala uz odgovarajući tekst.
- Redefinisanu metodu equals klase Object koja kao ulazni argument prima objekat klase Object, ali se smatra da će se zaista unositi objekti klase Monitor. Prvo, potrebno je proveriti da li je zaista unet objekat klase Monitor, pa ako nije ispisati poruku o grešci na ekranu i vratiti false. Ako jeste unet objekat klase Monitor, ova metoda vraća true ako su vrednosti atributa marka, model i dijagonala jednaki marki modelu i dijagonali monitora koji je unet kao ulazni argument. U suprotnom, metoda vraća false.

Napraviti klasu **Test Monitor** koja kreira dva objekta klase Monitor. Prvi je Samsung model SyncMaster 151n sa dijagonalom 15", a drugi je Sony model WS sa dijagonalom 17". Proveriti da li je u pitanju isti monitor i ispisati poruku o tome. Nakon toga, ispisati podatke o svakom monitoru na ekranu.

Rešenje:

```

public class Monitor {

    String marka;
    String model;
    int dijagonala;

    public int getDijagonala() {
        return dijagonala;
    }

    public void setDijagonala(int dijagonala) {
        if (dijagonala >= 12) this.dijagonala = dijagonala;
        else System.out.println("Greska");
    }

    public String getMarka() {
        return marka;
    }

}

```



```

public void setMarka(String marka) {
    if (marka != null) this.marka = marka;
    else System.out.println("Greska");
}
public String getModel() {
    return model;
}
public void setModel(String model) {
    if (model != null) this.model = model;
    else System.out.println("Greska");
}

public String toString(){
    return "Marka monitora: "+marka+" Model: "+model+
        " Dijagonala: "+dijagonala;
}

public boolean equals (Object o){
    //Operatorom instanceof se proverava da li ulazni argument
    //o zaista sadrzi objekat klase Monitor.
    if (o instanceof Monitor){
        Monitor m = (Monitor) (o);

        if (marka.equals(m.getMarka()) && model.equals(m.getModel()) &&
            dijagonala == m.getDijagonala()) return true;
        else return false;
    }
    else {
        System.out.println("Greska");
        return false;
    }
}

}

public class TestMonitor {

    public static void main(String[] args) {

        Monitor m1 = new Monitor();

        m1.setMarka("Samsung");
        m1.setModel("SyncMaster 151n");
        m1.setDijagonala(15);

        Monitor m2 = new Monitor();

        m2.setMarka("Sony");
        m2.setModel("WS");
        m2.setDijagonala(17);

        if (m1.equals(m2)) System.out.println("U pitanju je isti monitor");
        else System.out.println("U pitanju su razliciti monitori");

        System.out.println(m1);
        System.out.println(m2);

    }

}

```

Napraviti klasu **Osoba** koja ima:

- Atribut ime. Početna vrednost ovog atributa je "N".
- Atribut prezime. Početna vrednost ovog atributa je "N".
- Atribut JMBG koji predstavlja matični broj predstavljen kao String..
- Odgovarajuće get i set metode za ove attribute. Nedozvoljene vrednosti za ime i prezime i JMBG su null Stringovi. U slučaju prekoračenja bilo kog od ovih ograničenja potrebno je ispisati poruku o grešci.
- Redefinisati metodu toString klase Object. Ova metoda bi trebalo da vraća String koji sadrži vrednosti imena i prezimena osobe, kao i odgovarajući tekst uz to.
- Redefinisati metodu equals klase Object koja kao ulazni argument prima objekat klase Object ali se smatra da će se zaista unositi objekti klase Osoba. Prvo, potrebno je proveriti da li je zaista unet objekat klase Osoba, pa ako nije ispisati poruku o grešci na ekranu i vratiti false. Ako jeste unet objekat klase Osoba, ova metoda vraća true ako su vrednosti atributa ime, prezime i JMBG jednaka imenu, prezimenu i JMBG unete osobe, a u suprotnom false.

Napraviti klasu **TestOsoba** koja kreira dva objekta klase Osoba. Dodeliti i jednoj i drugoj osobi ime "Pera Perić" i JMBG "1010980710034"pa proveriti i ispisati poruku o tome da li je u pitanju ista osoba. Nakon toga, ispisati podatke o prvoj osobi.

Rešenje:

```
public class Osoba {

    String ime = "N";
    String prezime = "N";
    String JMBG;

    public String getIme() {
        return ime;
    }
    public void setIme(String ime) {
        if (ime != null) this.ime = ime;
        else System.out.println("Greska");
    }
    public String getJMBG() {
        return JMBG;
    }
    public void setJMBG(String jmbg) {
        if (jmbg != null) JMBG = jmbg;
        else System.out.println("Greska");
    }
    public String getPrezime() {
        return prezime;
    }
    public void setPrezime(String prezime) {
        if (prezime != null) this.prezime = prezime;
        else System.out.println("Greska");
    }

    public String toString(){
        return "Ime osobe je: "+ime+"; Prezime osobe je: "+prezime+
            " JMBG: "+JMBG;
    }

    public boolean equals (Object o){
        if (o instanceof Osoba){
            Osoba os = (Osoba) (o);
            if (ime.equals(os.getIme()) &&
                prezime.equals(os.getPrezime()) &&
                JMBG.equals(os.getJMBG())) return true;
            else return false;
        }
        else{
            System.out.println("Greska");
        }
    }
}
```

```

        return false;
    }
}

public class TestOsoba {

    public static void main(String[] args) {

        Osoba os1 = new Osoba();

        os1.setIme("Pera");
        os1.setPrezime("Peric");
        os1.setJMBG("1010980710034");

        Osoba os2 = new Osoba();

        os2.setIme("Pera");
        os2.setPrezime("Peric");
        os2.setJMBG("1010980710034");

        if (os1.equals(os2))
            System.out.println("U pitanju je ista osoba");
        else
            System.out.println("U pitanju su razlicite osobe");

        System.out.println(os1);
    }
}

```