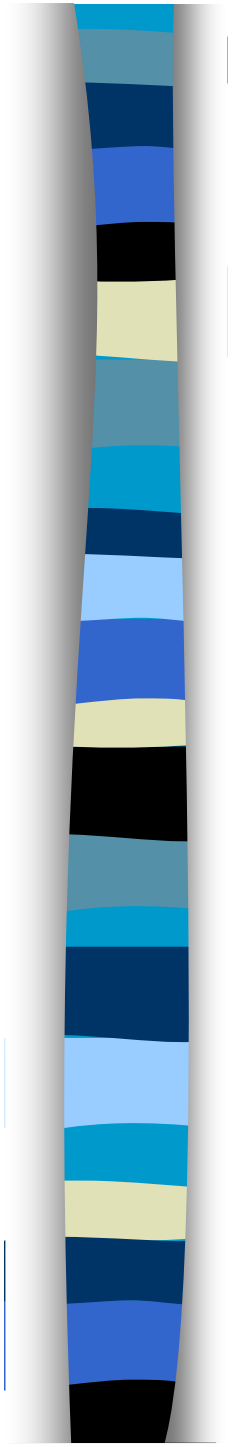
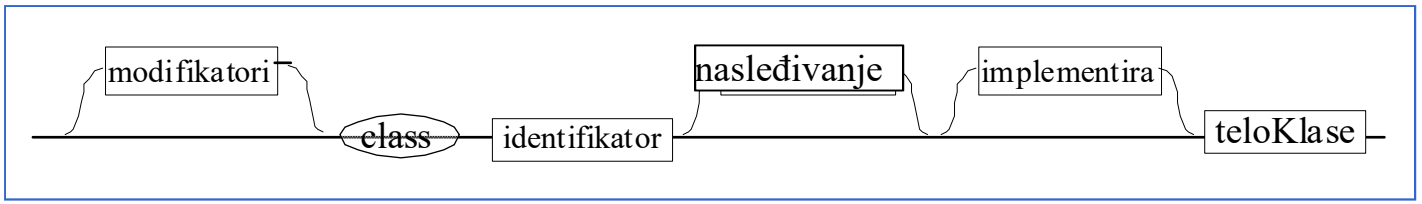
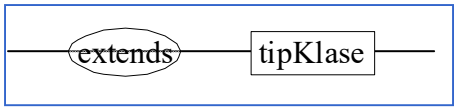


Deklaracija klase

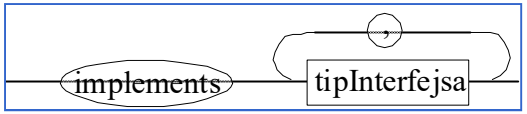
Deklaracija klase

- Deklaracija klase koja nije ugnježdena može započeti nekim od modifikatora: **`public, abstract, final, strictfp`**

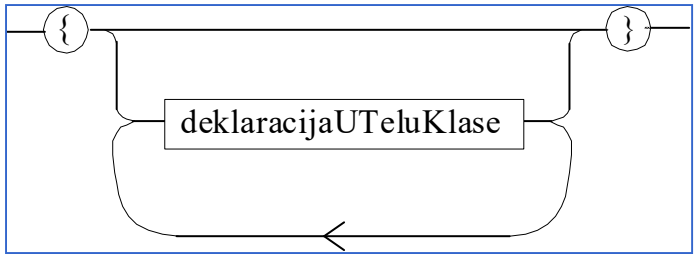




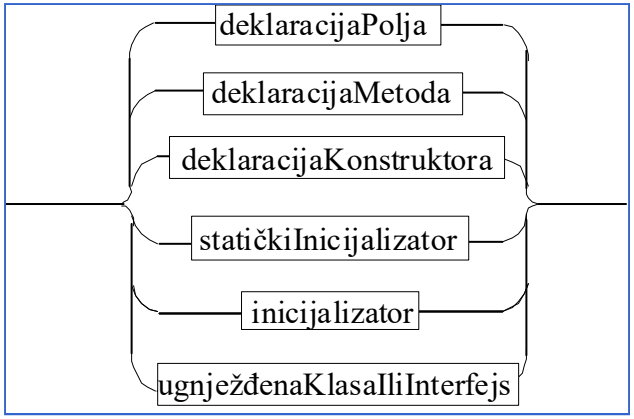
nasleđivanje



implementira



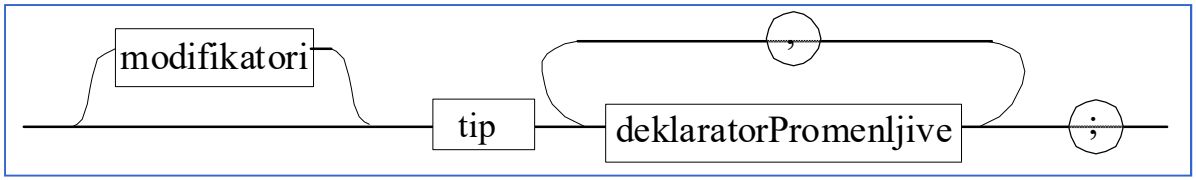
teloKlase

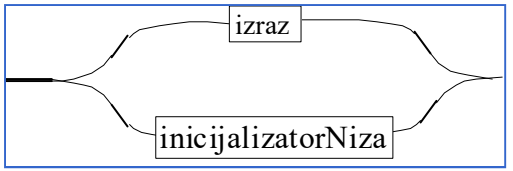


deklaracijaUTeluKlase

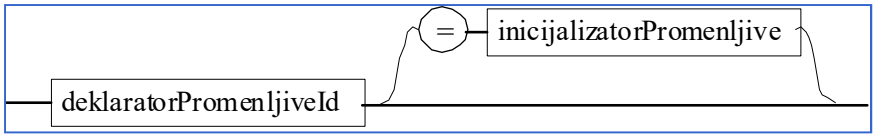
Deklaracija klase - polja

- Za deklaraciju polja mogu biti upotrebljeni modifikatori: **`public, protected, private, final, static, transient, volatile`**

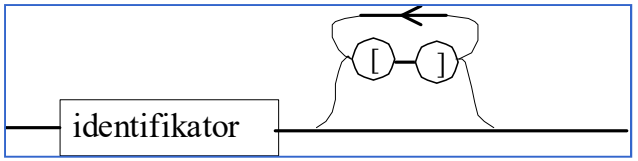




inicijalizatorPromenljive



deklaratorPromenljive



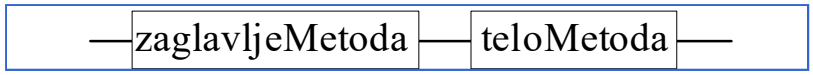
deklaratorPromenljiveId

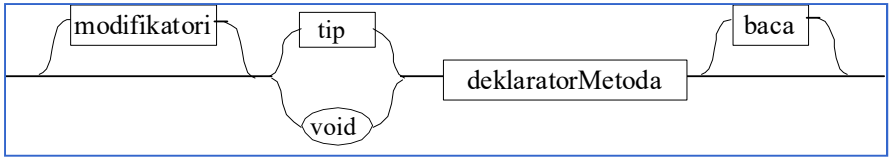
Primeri

```
protected final int x = 10, z = 9;
private double [] niz, matrica [];
volatile public String boja = "zeleno";
mojaKlasa mk1, mk2 = new mojaKlasa();
```

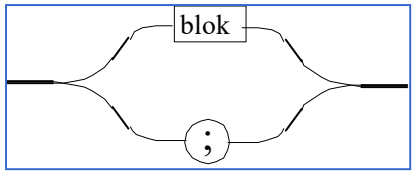
Deklaracija klase - metodi

- Zaglavlje metoda može početi navođenjem jednog ili više modifikatora metoda: **`public, protected, private, abstract, final, static, synchronized, native, strictfp`**

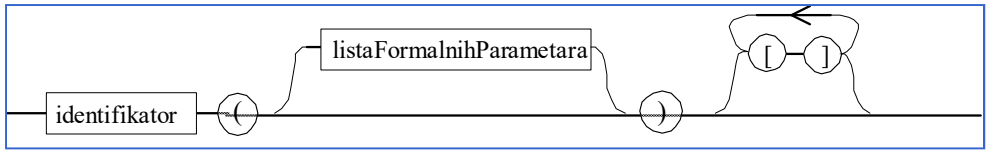




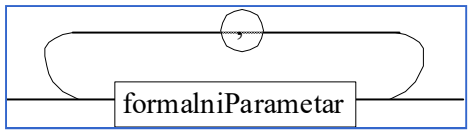
zaglavljeMetoda



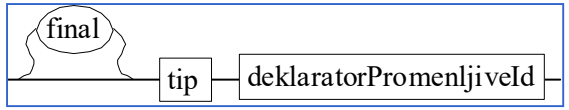
teloMetoda



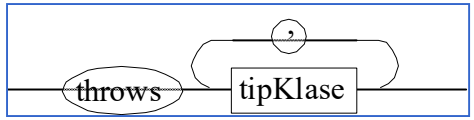
deklaratorMetoda



listaFormalnihParametara



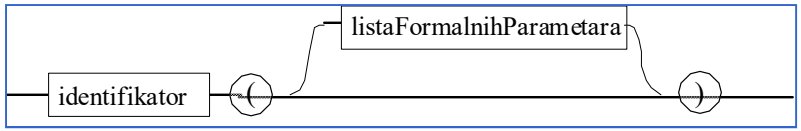
formalniParametar



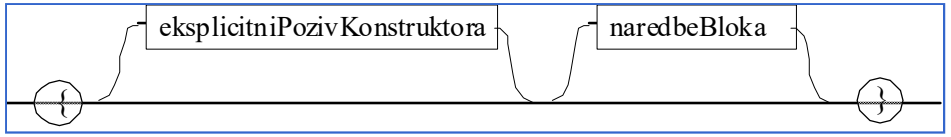
baca

Deklaracija klase - konstruktori

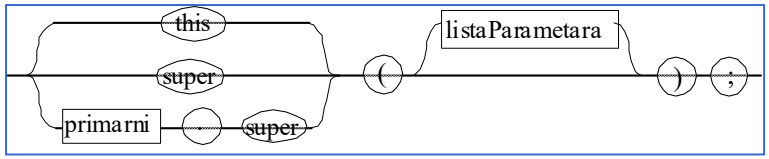
- Konstruktori su članovi klasa koji se koriste za kreiranje instanci klasa, odnosno dodelu vrednosti poljima (tj. promenljivim)
- Konstruktor može biti deklarisan bez modifikatora, ili sa jednim od sledećih modifikatora: **`public, protected, private`**



deklaratorKonstruktora

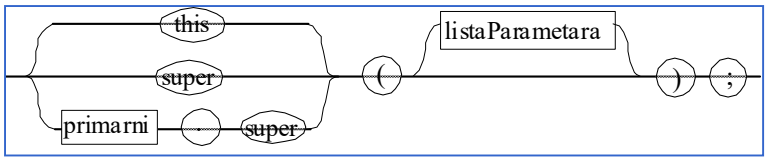


teloKonstruktora



eksplicitniPozivKonstruktora

Deklaracija klase - konstruktori



- Postoje tri vrste eksplicitnog poziva konstruktora u telu konstruktora:
  - Pozivanje drugog konstruktora iste klase - pomoću ključne reči **this**
  - Pozivanje konstruktora direktne nadklase pomoću ključne reči **super**
  - Pozivanje konstruktora direktne nadklase kada je direktna nadklasa unutrašnja klasa – koristi se primarni izraz i ključna reč **super**

Eksplicitni poziv konstruktora iste klase

- Klasa može da ima više konstruktora koji se moraju međusobno razlikovati po tipovima i/ili broju svojih parametara
- Klasa ne mora imati deklarisan ni jedan konstruktor - takvoj klasi će Java prevodilac u toku prevođenja dodati konstruktor bez parametara, koji u svom telu sadrži samo poziv **`super()`**
- Konstruktor iste klase se poziva pomoću ključne reči **`this`**
- Konstruktor direktne nadklase se uvek poziva na početku konstruktora tekuće klase, ako telo ne počinje pozivom konstruktora iste klase.

Eksplicitni poziv konstruktora iste klase

- Klasa može da sadrži više konstruktora

```
public class Student {

  private String ime;
  private int brIndeksa;
  private String vrstaFinansiranja;

  // konstruktor za sve studente
  public Student(String imeStudenta,
                 int brIndeksaStudenta,
                 String vrstaFinansiranjaStudenta) {
    ime = imeStudenta;
    brIndeksa = brIndeksaStudenta;
    vrstaFinansiranja = vrstaFinansiranjaStudenta;
  }

  // konstruktor samo za redovne studente
  public Student(String imeStudenta, int brIndeksa) {
    this(imeStudenta, brIndeksa, "iz budzeta");
```

```
class program {
 public static void main(String[] args) {
   Student A = new Student("Aca", 112);  //finansiran iz budzeta
   Student B = new Student("Nikola", 113, "samofinansirajuci");
   Student C = new Student("Eva", 114, "iz budzeta");
  }
}
```

Eksplicitni poziv konstruktora direktne nadklase

- Prva naredba u telu konstruktora može biti, a ponekad i mora biti, eksplicitni poziv konstruktora direktne nadklase, čime se inicijalizuju nasleđeni delovi klase
- Konstruktor direktne nadklase se poziva pomoću ključne reči **super**
- Konstruktor direktne nadklase se uvek poziva na početku izvršavanja konstruktora, ako telo konstruktora ne počinje eksplicitnim pozivom konstruktora iste klase (**this**)
- Ako telo konstruktora ne počinje ni eksplicitnim pozivom konstruktora iste klase, niti eksplicitnim pozivom konstruktora direktne nadklase, tada će se pri prevođenju na početku konstruktora dodati poziv konstruktora bez parametara **super()**, direktne nadklase.

Eksplicitni poziv konstruktora direktne nadklase

- Ako direktna nadklasa klase ne sadrži konstruktor bez parametara, tada telo svakog konstruktora klase mora početi:
  - eksplicitnim pozivom konstruktora iste klase (this) ili
  - eksplicitnim pozivom konstruktora direktne nadklase (super).
- U suprotom će se javiti greška pri prevođenju klase, jer bi Java prevodilac konstruktoru najpre dodao poziv super() na početku njegovog tela, a posle ne bi pronašao konstruktor bez parametara u direktnoj nadklasi klase i prijavio bi grešku.

Eksplicitni poziv konstruktora direktne nadklase

- Konstruktor direktne nadklase se poziva pomoću ključne reči **`super`**

```
class StudentPMF extends Student {

  private String institut;
  //matematika, fizika, hemija, geografija ili biologija

  StudentPMF(String imeStudenta,
             int brIndeksaStudenta,
             String vrstaFinansiranjaStudenta,
             String imeInstituta) {
    super(imeStudenta, brIndeksaStudenta,
          vrstaFinansiranjaStudenta);
    institut = imeInstituta;
  }

  StudentPMF(String imeStudenta,
             int brIndeksaStudenta,
             String imeInstituta) {
    this(imeStudenta, brIndeksaStudenta, "iz budzeta",
         imeInstituta);
  }

  // ostali delovi klase
}
```

Klase bez konstruktora

Klasa bez konstruktora

```
class Razlomak {

  int brojilac;
  int imenilac;

  String uString() {
    String pom = Math.abs(brojilac) + "/" + Math.abs(imenilac);
    if (brojilac * imenilac < 0)
      return "-" + pom;
    else
      return pom;
  }
}
```

Klasa može biti deklarisana bez konstruktora, ali će joj pri prevođenju biti dodat konstruktor samo sa pozivom **super()**

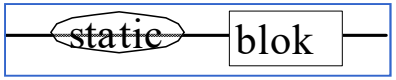
Deklaracija klase - konstruktori

Klasa sa konstruktorom

```
class Razlomak {

  int brojilac;
  int imenilac;

  Razlomak() {
    super();
  }

  String uString() {
    String pom = Math.abs(brojilac) + "/" + Math.abs(imenilac);
    if (brojilac * imenilac < 0)
      return "-" + pom;
    else
      return pom;
  }
}
```

Deklaracija klase - statički inicijalizatori

- Služi za inicijalizaciju statičkih polja klase.
- Statički inicijalizator je programski kod, koji se izvršava prvi put kada se klasa koristi u programu.
- U jednoj klasi može biti i više statičkih inicijalizatora - tada se svi oni izvršavaju redosledom kojim su navedeni u deklaraciji klase.



- Statička polja koja se koriste u statičkom inicijalizatoru moraju biti deklarisana pre deklaracije tog statičkog inicijalizatora.
- Za razliku od statičkih polja, statički metodi koji se pozivaju u statičkom inicijalizatoru mogu biti deklarisani i posle deklaracije statičkog inicijalizatora.
- Klasa u kojoj je glavni program takođe ima svoj statički inicijalizator.

Deklaracija klase - statički inicijalizatori

- Statička polja koja se koriste u statičkom inicijalizatoru moraju biti deklarisana pre deklaracije tog statičkog inicijalizatora.
- Za razliku od statičkih polja, statički metodi koji se pozivaju u statičkom inicijalizatoru mogu biti deklarisani i posle deklaracije statičkog inicijalizatora.
- Klasa u kojoj je glavni program takođe ima svoj statički inicijalizator.

```
class mojaKlasa {

  private static int brojKreiranih;

  static {
    System.out.println("mojaKlasa - staticki inicijalizator");
    brojKreiranih = 0;
  }

  static int koliko() {
    return brojKreiranih;
  }
```

Deklaracija klase - statički inicijalizatori

```
  mojaKlasa() {  // konstruktor
    brojKreiranih++;
    //...itd ...
  }

  //...ostali clanovi klase ...
}

class program {
  static {
    System.out.println("program - staticki inicijalizator");
  }
  public static void main(String[] args) {
    System.out.println("pocetak programa");
    System.out.println(mojaKlasa.koliko());
    mojaKlasa m1 = new mojaKlasa();
    System.out.println(mojaKlasa.koliko());
    mojaKlasa m2 = new mojaKlasa();
    System.out.println(mojaKlasa.koliko());
    System.out.println("kraj programa");
```

Ispis

```
program - staticki inicijalizator
pocetak programa
mojaKlasa - staticki inicijalizator
0
1
2
kraj programa
```

Deklaracija klase - inicijalizatori objekta

- Slični statičkim inicijalizatorima klasa.
- Izvršavaju se prilikom kreiranja instance klase i to **pre izvršenja tela konstruktora klase**.
- Ako se u nekoj klasi navede više inicijalizatora objekta, izvršavaće se svi, redosledom kojim su navedeni u deklaraciji klase.

Deklaracija klase - inicijalizatori objekta

- Polja koja se koriste u inicijalizatoru objekta moraju biti deklarisana pre deklaracije tog inicijalizatora. To ne važi za metode.
- Metodi koji se pozivaju u inicijalizatoru objekta mogu biti deklarisani i posle deklaracije inicijalizatora objekta.

Uvođenje inicijalizatora objekata u Javu nije značajno obogatilo jezik, jer oni imaju istu namenu kao i konstruktori klasa, te su u tom pogledu suvišni elementi. Oni su najverovatnije uvedeni samo zbog toga da bismo imali analogon statičkim inicijalizatorima.

Deklaracija klase - inicijalizatori objekta

Primer

```
class mojaKlasa {

  int x = 1;

  {
    System.out.println("prvi inicijalizator, x = " + x);
    x = 2;
  }

  mojaKlasa() {
    System.out.println("konstruktor, x = " + x);
    x = 3;
  }

  {
    System.out.println("drugi inicijalizator, x = " + x);
    x = 4;
  }
}

class program {
  public static void main(String[] args) {
    mojaKlasa m = new mojaKlasa();
    System.out.println("na kraju, x = " + m.x);
  }
}
```

Ispis

```
prvi inicijalizator, x=1
drugi inicijalizator, x=2
konstruktor, x=4
na kraju, x=3
```

Ugnježđene klase, interfejsi i nabrojivi tipovi

- Klase do sada bile su međusobno odvojene – svaka smeštena odvojeno u svom izvornom fajlu.
- Ne moraju sve klase biti definisane na taj način.
- Definicija klase može se nalaziti unutar definicije neke druge klase. Unutrašnja klasa naziva se ***ugnježđena (nested) klasa.***
- Ukoliko za to ima potrebe, ugnježđena klasa u sebi može imati ugnježđenu klasu.

Ugnježđene klase, interfejsi i nabrojivi tipovi

- Ugnježđene klase i interfejsi su klase i interfejsi deklarisani unutar neke klase. Noviji elementi jezika.
  - U jednostavnom obliku od Jave 1.1.
  - Sadašnji način upotrebe od Jave 1.2
  - Ugnježđeni nabrojivi tipovi – uvedeni kad i nabrojivi tipovi uopšte od Jave1.5
- Koriste se na sličan način kao i obične klase i interfejsi.
- Treba ih koristiti samo za implementaciju klase u kojoj su ugnježđeni.
- Postoje **četiri vrste** ugnježđenih klasa, interfejsa i nabrojivh tipova:
  - statičke ugnježđene klase, interfejsi i nabrojivi tipovi
  - unutrašnje klase,
  - lokalne klase i
  - anonimne klase

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

- Uvedeni su da bi se deklaracije klasa i interfejsa učinile preglednijim i čitljivijim, kao i iz praktičnih razloga da bi se male klase i interfejsi deklarisali bliže mestima na kojima se koriste.

- Uz modifikator **static** (koji je obavezan za ovakve klase) i modifikatore **public, final, abstract** i **strictfp**, mogu se koristiti i modifikatori **private** i **protected**.

- Isto važi i za interfejse, s tim što interfejs ne može biti deklarisan sa **final** modifikatorom. Ako se ugnježđeni interfejs ne deklariše sa modifikatorom **static,** on će imati iste osobine kao da je deklarisan pomoću modifikatora **static.**

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

- Ugnježđeni nabrojivi tipovi su takođe implicitno statički, a takođe i **final** (kao i svi nabrojivi tipovi).

- Vidljivost ugnježđenih statičkih klasa, interfejsa i nabrojivih tipova se određuje na osnovu upotrebljenih modifikatora, na isti način kao i vidljivost polja i metoda klase.

Sa korišćenjem ugnježđenih klasa ne treba preterivati da se ne bi pokvarila čitljivost napisanog koda.

Klasu treba koristiti kao ugnježđenu samo tada ako je ona mala i ako se ne koristi nigde osim za implementaciju spoljašnje klase.

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

Primer – bez korišćenja unutrašnje klase

```
class Lista {
  String ime;
  Lista veza;
}
public class Odeljenje {

  private String razredni;
  private Lista djaci;

  public Odeljenje(String razredni) {
    this.razredni = razredni;
    djaci = null;
  }
```

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

```
public void upisiDjaka(String ime) {
  Lista novi = new Lista();
  novi.ime = ime;
  novi.veza = djaci;
  djaci = novi;
}

public void ispisiDjaka(String ime) {
  Lista tekuci = djaci;
  Lista prethodni = tekuci;
  while ((tekuci != null) && (tekuci.ime.compareTo(ime) != 0)){
    prethodni = tekuci;
    tekuci = tekuci.veza;
  }
  if (tekuci == null) {
    System.out.println("Djak nije bio ni upisan.");
  } else if (tekuci == djaci) {
    djaci = djaci.veza;
  } else {
    prethodni.veza = tekuci.veza;
  }
}
```

Statičke ugnježdene klase, interfejsi i nabrojivi tipovi

```
  public void stampaj() {
    Lista tekuci = djaci;
    while (tekuci != null) {
      System.out.println(tekuci.ime);
      tekuci = tekuci.veza;
    }
  }
}

class program {
  public static void main(String[] args) {
    Odeljenje od = new Odeljenje("Pera Peric");
    od.upisiDjaka("Nikola Nikolic");
    od.upisiDjaka("Jovan Jovanovic");
    od.upisiDjaka("Marija Maric");
    od.stampaj();
    od.ispisiDjaka("Jovan Jovanovic");
    od.stampaj();
  }
}
```

Java API u paketu `java.util` sadrži više klasa kojima se predstavljaju liste i koje bi se mogle upotrebiti u prethodnom primeru umesto klase `Lista`.

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

Primer – pristup iznutra

```
public class Odeljenje {

  private String razredni;
  private Lista djaci;

  static private class Lista {
    String ime;
    Lista veza;
  }

  public Odeljenje(String razredni) {
    this.razredni = razredni;
    djaci = null;
  }

  // isto kao u prethodnom primeru
```

U ovom slučaju ne smemo koristiti modifikator `private`***.***

Primer – pristup od spolja

```
class Spoljasnja {
  int x = 2;
  static class Ugnjezdena {
    int y = 7;
  }
}

class program {
 public static void main(String[] args) {
    Spoljasnja.Ugnjezdena u =
           new Spoljasnja.Ugnjezdena();
    u.y++;
    System.out.println(u.y);
  }
}
```

Statičke ugnježđene klase, interfejsi i nabrojivi tipovi

- Ugnježđeni interfejsi u Javi su po definiciji statički tako da je navođenje ključne reči **static** u slučaju ugnježđenih interfejsa redundantno.
- Podrazumeva se da su svi metodi **public** i sva polja **public**, **final** (tako da navođenje tih modifikatora isto tako nije potrebno).
- Sledeći primer ilustruje upotrebu ugnježđenog statičkog interfejsa koji omogućava korisniku klase da utiče na ponašanje njenih metoda.

Unutrašnje klase

- Unutrašnje (eng. *inner*) klase - ugnježdene klase deklarisane bez modifikatora **static.**
- Instance unutrašnje klase mogu nastati samo unutar instanci spoljašnje klase.
- Kreirana instanca unutrašnje klase može pristupati svim članovima svoje spoljašnje klase, uključujući i privatne članove.
- Ako je neko ime u spoljašnjoj klasi nevidljivo zbog toga što u unutrašnjoj klasi postoji isto takvo ime, imenu spoljašnje klase se može pristupiti, korišćenjem imena spoljašnje klase i ključne reči **this.**
- Prilikom deklarisanja unutrašnje klase mogu se koristiti isti modifikatori kao i kod deklarisanja statičke ugnježdene klase, osim modifikatora **static.**
- Unutrašnja klasa može takođe biti nasleđivana - u tom slučaju u konstruktoru direktne podklase prva naredba mora biti eksplicitni poziv konstruktora direktne nadklase (unutrašnje klase) pri čemu se mora navesti i instanca spoljašnje klase.

Unutrašnje klase

Primer – korišćenje unutrašnje klase

```
public class DvaBroja {
  private int x = 1;
  private int y = 1;
  private class Uvecivac {
    void uvecaj() {
      int x = 1 + (int) ( Math.random() * 10);
      DvaBroja.this.x = DvaBroja.this.x + x; //pristup spoljasnjem x
      y = y + 2 * x;
    }
  }
  public void povecaj(int puta) {
    Uvecivac u = new Uvecivac();
    for (int i = 1; i <= puta; i++)
      u.uvecaj();
  }
  public void stampaj() {
    System.out.println("x = " + x);
    System.out.println("y = " + y);
  }
}
class program {
  public static void main(String[] args) {
    DvaBroja db = new DvaBroja();
    db.povecaj(7);
    db.stampaj();
  }
}
```

Unutrašnje klase

Primer – nasleđivanje unutrašnje klase

```
class Spoljasnja {
  Spoljasnja() { //konstruktor spoljasnje
    System.out.println("Spoljasnja");
  }
  class Unutrasnja { //unutrasnja klasa
    Unutrasnja() {
      System.out.println("Unutrasnja");
    }
  }
}
class Podklasa extends Spoljasnja.Unutrasnja {
  Podklasa(Spoljasnja s) {
   s.super();
   //poziv konstruktora nadklase uz navodjenje instance spoljasnje klase
   System.out.println("Podklasa");
  }
}
class program {
  public static void main(String[] args) {
    Spoljasnja s = new Spoljasnja();
    Podklasa p = new Podklasa(s);
  }
}
// Ispis: Spoljasnja
//        Unutrasnja
//        Podklasa
```

Lokalne klase

- Lokalne klase – ugnježdene klase deklarisane u bilo kom bloku. U telu nekog metoda ili konstruktora, slično kao lokalne promenljive.
- Lokalna klasa može biti deklarisana sa modifikatorima **abstract, final, strictfp** ili bez modifikatora i ponaša se slično kao i unutrašnja klasa.
- Lokalne klase ne mogu biti statičke.

Primer – ispisivanje "Dobar dan."

```
class program {
  public static void main(String[] args) {
    String pozdrav = "Dobar dan.";

    class malaLokalna {
      String poruka;

      malaLokalna(String poruka) {
        this.poruka = poruka;
      }
      void ispisiPoruku() {
        System.out.println("Poruka je: " + poruka);
      }
    }

    malaLokalna mala = new malaLokalna( pozdrav );
    mala.ispisiPoruku();
  }
}
```

Početak važenja deklarisanih imena

- Java podržava **rekurziju** i **uzajamnu rekurziju** - imena klasa i metoda mogu se koristiti i pre završetka njihove deklaracije.
- **Rekurzija** – pojava da se u telu nekog metoda nalazi poziv istog tog metoda.

Primer rekurzije – faktorijel broja

```
public static long faktorijel(int n) {
  if (n <= 1)
    return 1;
  else
    return n * faktorijel(n-1);
}
```

Početak važenja deklarisanih imena

- **Uzajamna rekurzija** – kada se dva ili više metoda međusobno pozivaju tako da se uspostavlja ciklus poziva

Primer uzajamne rekurzije – pet puta "Dobar dan"

```
class Pozdrav {

  String prvaRec = "Dobar";
  String drugaRec = "dan";

  void prviDeo(int puta) {
    System.out.print(prvaRec + " ");
    drugiDeo(puta);
  }

  void drugiDeo(int puta) {
    System.out.println(drugaRec + ".");
    if (puta > 1)
      prviDeo(puta-1);
  }
}

class program {
  public static void main(String[] args) {
    Pozdrav p = new Pozdrav();
    p.prviDeo(5);
  }
}
```

Početak važenja deklarisanih imena

- Rekurzija i uzajamna rekurzija su moguće u Javi – imena svih metoda klase postoje u svim delovima klase – pre navođenja deklaracije nekog metoda i u toku njegove deklaracije i posle deklaracije ime tog metoda može da se koristi za njegovo pozivanje.
- Mesto i redosled navođenja deklaracija polja u klasi takođe ne utiče na njihovu vidljivost u konstruktorima, metodima i unutrašnjim klasama, ali utiče na njihovu vidljivost u **statičkim inicijalizatorima** i u **inicijalizatorima objekta.**
- **Statički inicijalizator** pristupa samo statičkim poljima i to samo onim koji su deklarisani pre njegove deklaracije.
- **Inicijalizator objekta** vidi sva statička polja bez obzira na mesto njihove deklaracije, a od nestatičkih polja vidi samo ona koja su deklarisana pre njega.

Početak važenja deklarisanih imena

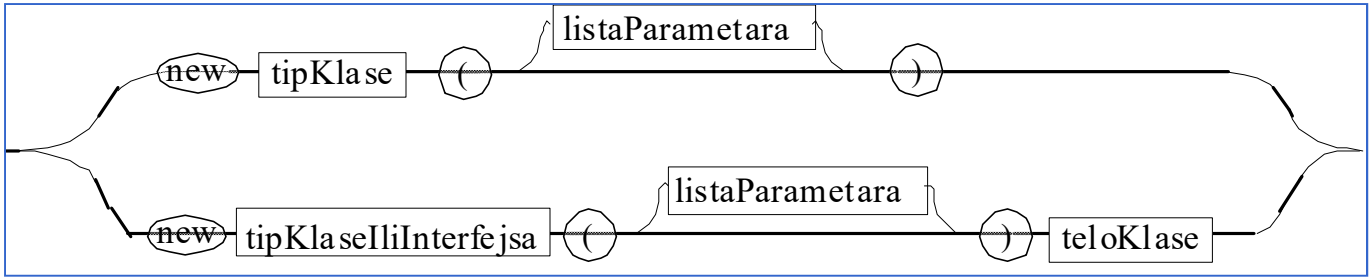
- Mesto deklaracije statičke ugnježdene klase ili interfejsa unutar deklaracije neke klase ne utiče na korišćenje statičke ugnježdene klase, odnosno interfejsa.
- Isto važi i za unutrašnje klase.
- Klasa (ili interfejs) kod koje se prilikom deklaracije njenih članova koristi ista klasa je takođe rekurzivna.
- Ime klase može da se koristi i pre njene deklaracije.

Početak važenja deklarisanih imena

Primer – uzajamno rekurzivni tipovi

```
public class Pas {
  String ime;
  Macka neprijatelj;
}

public class Macka {
  String ime;
  Pas neprijatelj;
}

class program {
  public static void main(String[] args) {
    Pas Fifi = new Pas();
    Fifi.ime = "Fifi";
    Macka Zizi = new Macka();
    Zizi.ime = "Zizi";
    Fifi.neprijatelj = Zizi;
    Zizi.neprijatelj = Fifi;
  }
}
```

Kreiranje instanci klasa



- Nova instanca (objekat) neke klase se kreira primenom **new** operatora.
- U toku kreiranja instance klase izvršava se konstruktor klase, poziva se i konstruktor direktne nadklase, dodeljuju se početne vrednosti poljima i izvršavaju se svi inicijalizatori objekta.
- Donji deo služi za kreiranje anonimnih klasa.

Kreiranje instanci klasa

Kada se pravi nova instanca neke imenovane klase, izvršavaju se sledeći koraci:

1. Poziva se konstruktor klase i prelazi se na korak 2.
2. Ako telo konstruktora počinje eksplicitnim pozivom nekog drugog konstruktora iste klase (navođenjem ključne reči **this** i parametara u zagradama), tada se pomoću ovih istih pet koraka vrši kreiranje instance klase tim konstruktorom, nakon čega se prelazi na korak 5. Inače se prelazi na korak 3.
3. Telo konstruktora ne počinje pozivom nekog drugog konstruktora iste klase. Eksplicitno se (pomoću ključne reči **super**) ili implicitno poziva konstruktor direktne nadklase čime se pomoću ovih istih pet koraka kreira deo instance koji odgovara nasleđenim delovima klase i prelazi se na korak 4.
4. Redosledom kojim su navedeni u deklaraciji klase se inicijalizuju polja objekta i izvršavaju se inicijalizatori objekta. Ako je polje deklarisano bez inicijalne vrednosti, njegova početna vrednost će biti: **false** – ako je polje tipa boolean**,** ***nula odgovarajućeg tipa*** – ako je polje nekog numeričkog tipa, ili **null** – ako je polje referencijalnog tipa. Prelazi se na korak 5.
5. Izvršava se telo konstruktora do kraja čime se završava kreiranje instance klase

Kreiranje instanci klasa

Kada se pravi nova instanca neke anonimne klase:

- Prilikom kreiranja njene instance prvo se poziva konstruktor njene direktne nadklase – pomoću navedenih 5 koraka napravi se deo instance koja odgovara nasleđenim delovima klase
- Nakon toga izvršava se korak 4 iz prethodnog postupka

Kada je u pitanju inicijalizacija statičkih članova klase:

- ona se uvek dešava pre inicijalizacije nestatičkih članova i to samo jednom – pri prvom obraćanju klasi
- Kada se po prvi put kreira instanca neke klase ili se po prvi put pristupa njenom statičkom metodu ili polju – Java interpreter mora da učita traženu klasu
- Već tokom učitavanja, inicira se inicijalizacija svih njenih statičkih članova

Anonimne klase

- Postoje situacije kada hoćemo da definišemo klasu za koju želimo da definišemo samo jedan objekat u programu i jedina upotreba tog objekta je da se prosledi direktno kao argument metoda.
- U ovom slučaju, sve dok ta klasa nasleđuje postojeću klasu ili implementira neki interfejs, imamo opciju da je definišemo kao *anonimnu klasu.*
- Definicija anonimne klase se pojavljuje u izrazu *new*, u naredbi u kojoj kreiramo i koristimo, pa tako nema ni potrebe da obezbedimo ime za klasu.

Anonimne klase

- Ako nam je potrebna samo jedna instanca neke klase, tada tu klasu ne moramo deklarisati na uobičajeni način, već je možemo deklarisati prilikom kreiranja same instance ne navodeći ime – otud i naziv anonimna klasa.

- Deklaracija anonimne klase vrši se u telu neke druge klase, pa zato one spadaju u ugnježdene klase.

- Deklaracija anonimne klase uvek počinje pozivom konstruktora klase koju anonimna klasa nasleđuje, ili navođenjem imena interfejsa i navođenjem (), ako klasu koristimo da bismo njome implementirali neki interfejs. Nakon toga se u vitičastim zagradama navode članovi anonimne klase.

- Anonimna klasa ne sme da sadrži ni jedan konstruktor.

Anonimne klase

- Ako nam je potrebna samo jedna instanca klase tada tu klasu ne moramo deklarisati na uobičajen način, već je možemo deklarisati prilikom kreiranja same instance.
- Pri tome ne navodimo ime klase, zbog čega se takva klasa i zove **anonimna klasa**.
- Instanca anonimne klase se kreira navođenjem ključne reči **new** nakon koje sledi deklaracije anonimne klase.

Korišćenje anonimnih klasa se preporučuje samo u slučajevima kada je deklaracija anonimne klase veoma kratka, jer će u protivnom napisani kod izgubiti svoju čitljivost.

Anonimne klase

Primer

```
class Akcija {

  private String ime;

  Akcija(String ime) {
    this.ime = ime;
  }

  void izvrsi() {
    System.out.println
       ("Nista");
  }

  String citajIme() {
    return ime;
  }

}
```

```
class program {
  public static void main(String[] args) {
    Akcija a = new Akcija("Slucajan zbir") {
     //anonimna klasa, dodavanje njenih članova
      void izvrsi() {
        int x = (int) (Math.random() * 100);
        int y = (int) (Math.random() * 100);
        int zbir = x + y;
        System.out.println("Zbir dva slucajna
                    broja je: " + zbir);
      }
    };
    a.izvrsi();
  }
}
```

Anonimne klase

- Pretpostavimo da želimo da definišemo objekat klase koja implementira interfejs *ActionListener* za jednokratnu upotrebu.

```
pickButton.addActionListener(new ActionListener()      {

                    // Code to define the class

                    // that implements the

                    // ActionListener interface

}
```

- Definicija klase pojavljuje se u izrazu new koji kreira argument za metod *AddActionListener*(). Ovaj metod zahteva referencu tipa ActionListener – drugim rečima referencu na klasu koja implementira ActionListener interfejs.

Uništavanje instanci klasa

- Svaki kreirani objekat zauzima odeđeni memorijski prostor. Da bi se memorija računara racionalno koristila, objekti koji se u programu više ne koriste se uništavaju, a memorijski prostor koji oni zauzimaju se oslobađa.
- O uništavanju nepotrebnih instanci klasa ne brine programer već deo Java virtuelne mašine koji se zove sakupljač otpadaka (engl. *garbage collector*).
- Pre nego što uništi objekat, sakupljač otpadaka će pozvati njegov metod **finalize.**
- Metod **finalize** je član klase **Object** i njega nasleđuju sve podklase klase **Object** (a to su sve klase).
- Inicijalno, **finalize** ne radi ništa, ali se može redefinisati tako da se u okviru njega izvršava željena aktivnost – najčešće da bismo oslobodili resurse operativnog sistema koje je objekat zauzimao.

Uništavanje instanci klasa

- U definiciju klase može se uključiti metod **finalize**().
- Ovaj metod se poziva automatski pre nego se objekat konačno uništi i oslobodi prostor koji je zauzimao u memoriji.
- U praksi, to može biti nešto nakon što objekat postane nedostupan u programu.
- Kada JVM uništava objekat, ona zove metod finalize() za objekat.

```
protected void finalize(){
// nas kod ...
    }
```

- Metod **finalize()** je koristan ako objekti klase koriste resurse koji zahtevaju neku specijalnu akciju kada se uništavaju.
- Tipično, to su resursi koji nisu iz Java okruženja i ne garantuje se da će ih objekat sam osloboditi.

Uništavanje instanci klasa

- To mogu biti grafički resursi, fontovi ili drugi resursi povezani sa crtanjem koje je obezbedio host operativni sistem ili eksterni fajlovi na hard disku.
- Ukoliko se oni ne oslobode pre uništenja objekta, zauzimaju sistemske resurse i ukoliko ih potrošimo u dovoljnoj količini, naš program, a možda i drugi programi na sistemu mogu da prestanu da rade.
- Za većinu klasa ovo nije neophodno, ali ako je npr. objekat otvorio fajl sa diska, a ne garantuje njegovo zatvaranje, želimo da budemo sigurni da će fajl biti zatvoren kada se objekat uništi. Možemo implementirati metod ***finalize()*** tako da se pobrine za to.
- Druga upotreba metoda ***finalize()*** je da upamti činjenicu da je objekat uništen.

Uništavanje instanci klasa

- Ne možemo računati na to da će objekat biti uništen kada više nije dostupan kodu našeg programa.
- Osim ako naš program ne poziva System.gc() metod, JVM će se osloboditi neželjenih objekata i osloboditi memoriju koju oni zauzimaju jedino ako joj ponestaje memorije, ili ako nema aktivnosti u našem programu – npr. kada čeka na ulaz.
- Kao posledica toga, može se desiti da se objekti ne unište dok se program ne završi. Takođe, nemamo garancije kada će biti pozvan metod **finalize()**. Jedino je sigurno da će biti pozvan pre nego što se oslobodi memorija koju je objekat zauzimao.
- Ništa što zavisi od vremena ne bi trebalo ostavljati **finalize()** metodu!

Uništavanje instanci klasa

Primer – Metod `finalize`

```
class A {

  private int broj;
  private int [] niz;

  A(int br) {
    broj = br;
    niz = new int [1048576]; //niz kojem treba 4 MB memorije
  }

  protected void finalize() throws Throwable {
    super.finalize();
    System.out.println("unistavanje " + broj);
  }
}

class program {
  public static void main(String [] args) {
    for (int i = 1; i <= 5; i++)
      new A(i);
  }
}
```

Ispis (za 64 MB memorije)

```
unistavanje 1
unistavanje 2
unistavanje 3
unistavanje 4
```

Teorijske vežbe 4

Objektno-orijentisano programiranje

Apstraktne klase

- Deklarišu se rečju **`abstract`**.
- Ne može se instancirati objekat apstraktne klase.
- Apstraktni metod je metod koji se deklariše bez implementacije.
- Ako klasa sadrži apstraktni metod, i sama mora biti apstraktna.
- Ako je klasa apstraktna, nije nužno deklarisati apstraktni metod u njoj.
- Klasa koja nasleđuje apstraktnu klasu mora implementirati sve njene apstraktne metode, ili alternativno i sama mora biti apstraktna.

Zadatak 9

- Apstraktna klasa `PolovnoVozilo` opisuje polovno vozilo koje se prodaje.
- Ova klasa sadrži polje `godiste` i sledeće metode:
  - `baznaCena` – Računa baznu cenu vozila koja je jednaka `(20 - min(starostVozila, 19)) * 500`.
  - `toString` – Override toString metoda.
  - `preporucenaCena` - apstraktni metod koji treba da vrati cenu vozila.

```java
public abstract class PolovnoVozilo {

    private int godiste;

    public PolovnoVozilo(int godiste) {
        this.godiste = godiste;
    }

    public String toString() {
        return "Polovno vozilo godista " + godiste + ".";
    }

    protected int baznaCena() {
        int starostVozila =
                Calendar.getInstance().get(Calendar.YEAR) - godiste;
        return (20 - Math.min(starostVozila, 19)) * 500;
    }

    public abstract int preporucenaCena();
}
```

Zadatak 9

- Klasa `Automobil` nasleđuje klasu `PolovnoVozilo`.
- Sadrži polje `brojVrata` i implementira sledeće metode:
  - `preporucenaCena` – Implementacija apstraktnog metoda klase `PolovnoVozilo`. Preporučena cena je jednaka baznoj ceni ukoliko vozilo ima više od 3 vrata. U suprotnom se cena smanjuje za 5 posto.
  - `toString` – Override toString metoda. Implementacija sadrži i poziv `toString` metoda natklase.

```java
public class Automobil extends PolovnoVozilo {

    private int brojVrata;

    public Automobil(int godiste, int brojVrata) {
        super(godiste);
        this.brojVrata = brojVrata;
    }

    public int preporucenaCena() {
        int cena = baznaCena();
        if (brojVrata <= 3) {
            cena = Math.round(0.95f * cena);
        }
        return cena;
    }

    public String toString() {
        return super.toString() + " Tip polovnog vozila: Automobil. "
            + "Broj vrata: " + brojVrata;
    }
}
```

Zadatak 9

- Klasa `Kombi` nasleđuje klasu `PolovnoVozilo`.
- Sadrži polje `brojSedista` i implementira sledeće metode:
  - `preporucenaCena` – Implementacija apstraktnog metoda klase `PolovnoVozilo`. Preporucena cena je jednaka baznoj ceni uvećanoj za `brojSedista * 100`.
  - `toString` – Override toString metoda. Implementacija sadrži i poziv `toString` metoda natklase.

```java
public class Kombi extends PolovnoVozilo {

   private int brojSedista;

   public Kombi(int godiste, int brojSedista) {
       super(godiste);
       this.brojSedista = brojSedista;
   }

   public int preporucenaCena() {
       return baznaCena() + brojSedista * 100;
   }

   public String toString() {
       return super.toString() + " Tip polovnog vozila: "
             + "Kombi. Broj sedista: " + brojSedista;
   }
}
```

Zadatak 9

- U tekstualnom fajlu su navedena vozila. Svako vozilo je zadato u jednoj liniji fajla.
  - Primer automobila: a,2007,5
  - Primer kombija: k,2001,8
- U glavnom programu instancirati svako vozilo iz fajla i ispisati o kom se vozilu radi, a potom i njegovu preporučenu cenu.

```java
import java.io.*;

public class Program {

    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader("vozila.txt"));
        String linija = reader.readLine();
        while (linija != null) {
            PolovnoVozilo polovnoVozilo = null;
            String[] vrednosti = linija.split(",");
            String tipVozila = vrednosti[0].trim();
            int godiste = Integer.parseInt(vrednosti[1].trim());
            if (tipVozila.equals("a")) {
                int brojVrata = Integer.parseInt(vrednosti[2].trim());
                polovnoVozilo = new Automobil(godiste, brojVrata);
            } else if (tipVozila.equals("k")) {
                int brojSedista = Integer.parseInt(vrednosti[2].trim());
                polovnoVozilo = new Kombi(godiste, brojSedista);
            }
            if (polovnoVozilo != null) {
                System.out.println(polovnoVozilo + " Preporucena cena: " +
                        polovnoVozilo.preporucenaCena());
            } else {
                System.out.println("Format vozila u fajlu nije ispravan. (" +
                        linija + ")");
            }
            linija = reader.readLine();
        }
        reader.close();
    }
}
```

Zadatak 10

- Klasom Kurs predstavljen je jedan kurs u školi jezika.
- Ova klasa kao atribute ima:
  - jezik (String),
  - nivo, koji moze imati vrednosti "osnovni", "srednji" i "napredni" (set metoda ne dozvoljava neku drugu vrednost, setuju se samo ispravne vrednosti, ili "osnovni" ako vrednost nije ispravna), i
  - broj polaznika (set metoda ne dozvoljava negativne vrednosti, ako je vrednost negativna, broj polaznika je 0).

Zadatak 10

- Klasa Kurs jos sadrži i konstruktor koji inicijalizuje sve atribute, get i set metode za svaki atribut i toString() metod.

```java
public class Kurs {

    private String jezik;
    private String nivo;
    private int brojPolaznika;

    public Kurs(String jezik, String nivo, int brojPolaznika) {
        this.jezik = jezik;
        setNivo(nivo);
        setBrojPolaznika(brojPolaznika);
    }

    public String getJezik() {
        return jezik;
    }

    public void setJezik(String jezik) {
        this.jezik = jezik;
    }

    public String getNivo() {
        return nivo;
    }
```

```
    public void setNivo(String nivo) {
        if (nivo.equals("osnovni") || nivo.equals("srednji") || nivo.equals("napredni")) {
            this.nivo = nivo;
        } else {
            this.nivo = "osnovni";
        }
    }

    public int getBrojPolaznika() {
        return brojPolaznika;
    }

    public void setBrojPolaznika(int brojPolaznika) {
        if (brojPolaznika < 0) {
            this.brojPolaznika = 0;
        } else {
            this.brojPolaznika = brojPolaznika;
        }
    }

    @Override
    public String toString() {
        return jezik + " jezik, " + nivo + " nivo, broj polaznika: " + brojPolaznika;
    }

}
```

Zadatak 10

- Klasom SkolaJezika predstavljena je škola jezika u kojoj se održavaju kursevi.
- Podaci o kursevima se učitavaju iz tekstualnog fajla koji je organizovan na sledeći način: u prvoj liniji fajla nalazi se broj kurseva u ponudi škole, a u svakoj sledećoj podaci o jednom kursu razdvojeni zarezom (jezik, nivo, broj polaznika).

Zadatak 10

- Klasa SkolaJezika definiše i sledeće metode:
  - **String najnaprednijiJezik()** - vraca jezik na cijem naprednom kursu ima najvise polaznika. Ako ima vise takvih jezika, metod vraca prvi na koji je naisao.
  - **void unaprediGrupu(String jezik, String nivo)** - za zadati jezik i nivo, povecava nivo kursa za jedan stepen (osnovni u srednji, srednji u napredni). Ukoliko je zadati nivo napredni, ili ne postoji kurs sa unetim parametrima, metoda ne radi nista. Ukoliko nakon povecanja postoje dva kursa za isti jezik i nivo, potrebno je ispisati poruku da je potrebno spojiti te dve grupe.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class SkolaJezika {

    private Kurs[] kursevi;

    public SkolaJezika() throws IOException {
        BufferedReader in = new BufferedReader(new FileReader("kursevi.txt"));
        int brojKurseva = Integer.parseInt(in.readLine().trim());
        kursevi = new Kurs[brojKurseva];
        for (int i = 0; i < brojKurseva; i++) {
            String[] elementiLinije = in.readLine().split(",");
            String jezik = elementiLinije[0].trim();
            String nivo = elementiLinije[1].trim();
            int brojPolaznika = Integer.parseInt(elementiLinije[2].trim());
            kursevi[i] = new Kurs(jezik, nivo, brojPolaznika);
        }
        in.close();
    }
```

```
String najnaprednijiJezik() {
    Kurs najnaprednijiJezik = null;
    for (int i = 0; i < kursevi.length; i++) {
        if (najnaprednijiJezik == null
                || (kursevi[i].getNivo().equals("napredni")
                && kursevi[i].getBrojPolaznika() >
                najnaprednijiJezik.getBrojPolaznika())) {
            najnaprednijiJezik = kursevi[i];
        }
    }
    return najnaprednijiJezik == null ? "ne postoji" : najnaprednijiJezik.getJezik();
}
```

```
    void unaprediGrupu(String jezik, String nivo) {
        String noviNivo = null;
        if (nivo == "osnovni") {
            noviNivo = "srednji";
        } else if (nivo == "srednji") {
            noviNivo = "napredni";
        } else {
            return;
        }
        boolean postojiNapredniji = false;
        boolean unapredjenKurs = false;
        for (int i = 0; i < kursevi.length; i++) {
            if (kursevi[i].getJezik().equals(jezik)) {
                if (kursevi[i].getNivo().equals(noviNivo)) {
                    postojiNapredniji = true;
                }
                if (kursevi[i].getNivo().equals(nivo)) {
                    kursevi[i].setNivo(noviNivo);
                    unapredjenKurs = true;
                }
            }
        }
        if (postojiNapredniji && unapredjenKurs) {
            System.out.println("Potrebno je spojiti grupe za "
        + jezik + " jezik, " + noviNivo + " nivo.");
        }
    }

}
```

Zadatak 10

- U glavnom programu potrebno je napraviti instancu klase SkolaJezika za tekstualni fajl "kursevi.txt" i ispisati rezultate izvršavanja metoda:
  - najnaprednijiJezik() i
  - unaprediGrupu().

Zadatak 10

```
import java.io.IOException;

public class Glavna {

    public static void main(String[] args) throws IOException {
        SkolaJezika skolaJezika = new SkolaJezika();
        String najnaprednijiJezik = skolaJezika.najnaprednijiJezik();
        if (!najnaprednijiJezik.equals("ne postoji")) {
            najnaprednijiJezik = "je " + najnaprednijiJezik;
        }
        System.out.println("Najnapredniji jezik " + najnaprednijiJezik);
        skolaJezika.unaprediGrupu("engleski", "osnovni");
    }
}
```

```
Console
<terminated> Glavna (1) [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (Nov 1, 2017, 1:15:25 PM)
Najnapredniji jezik je engleski
Potrebno je spojiti grupe za engleski jezik, srednji nivo.
```