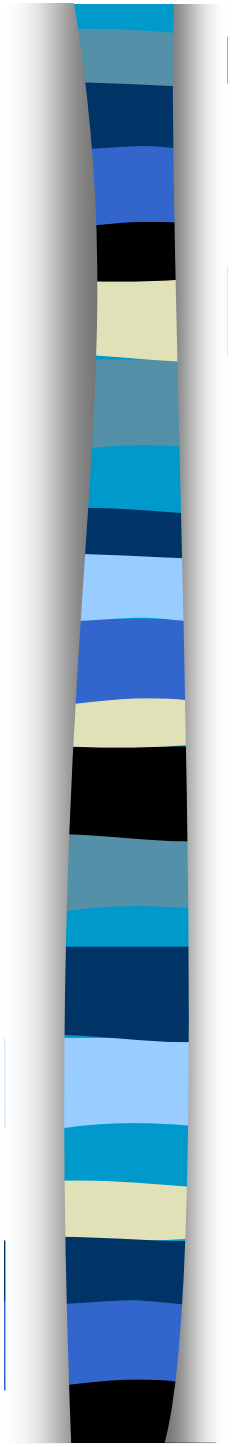


Nasleđivanje u Javi

Nasleđivanje

- Pomoću nasleđivanja, postojeće klase se mogu koristiti za pravljenje novih klasa
- Nova klasa koja nasleđuje neku postojeću klasu zadržava sve članove postojeće klase osim privatnih članova
- Novu klasu zovemo ***direktna podklasa*** postojeće klase, a postojeću klasu zovemo ***direktna nadklasa*** nove klase
- Klasa može da dobije i nove članove, koje njena direktna nadklasa ne sadrži, a postojeći metodi diretne nadklase mogu biti promenjeni (redefinisani - overriden) u novoj klasi
- Pored termina direktna podklasa i direktna nadklasa, uvešćemo i termine ***podklasa*** i ***nadklasa*** (tranzitivna zatvorenja relacija direktna podklasa, odnosno direktna nadklasa)
- Pomoću ključne reči **extends** se deklariše da klasa nasleđuje neku postojeću klasu

Nasleđivanje - primer

Klasa Tacka

```
class Tacka {
  private float x, y;
  Tacka( float X, float Y) {
    x = X;
    y = Y;
  }

  void transliraj(float promenaX, float promenaY) {
    x = x + promenaX;
    y = y + promenaY;
  }
  String opis() {
    return "tacka sa Dekartovim koordinatama (" + x + ", " + y + ")";
  }
  float getX() {  //vraca x koordinatu
    return x;
  }
  float getY() {  //vraca y koordinatu
    return y;
  }
  void setX(float x) {
    this.x = x;  //koristimo this jer je x i ime parametra i ime polja
  }
  void setY(float y) {
    this.y = y;
  }
}
```

Nasleđivanje - primer

- Tačka se u ovoj klasi Tacka predstavlja Dekartovim koordinatama.
- Pretpostavimo dalje da hoćemo da napravimo novu klasu pod imenom ObojenaTacka, pomoću koje će se predstavljati obojene tačke u ravni.
- Pošto klasa ObojenaTacka treba da ima mnoge sličnosti sa klasom Tacka, iskoristićemo postojeću klasu Tacka da bismo napravili klasu ObojenaTacka.

Nasleđivanje - primer

Proširenje klase Tacka

```
class ObojenaTacka extends Tacka {

  private String boja;

  ObojenaTacka(float x, float y, String boja) {
    super(x, y);
    this.boja = boja;
  }

  void oboji(String novaBoja) {
    boja = novaBoja;
  }

  String getBoja() {
    return boja;
  }

  String opis() {
    return "obojena " + super.opis() + " boje " + boja;
  }
}
```

Nasleđivanje

- Kod nasleđivanja klasa često imamo situaciju da metod koji se navodi u podklasi (a koji ima **istu signaturu**, tj. isto ime, broj i tip parametara, kao i tip vrednosti koju vraća nakon izrvšavanja) redefiniše (eng.**overrides**) odgovarajući metod iz nadklase.

- Na ovaj način se postiže da podklasa nasledi iz nadklase slično ponašanja ali da može i da promeni metode koji nisu adekvatni za novu podklasu.

Nasleđivanje

- Redefinisani metod može da vrati vrednost koja je podtip vrednosti koju vraća metod iz nadklase što može dodatno da stvori konfuziju prilikom programiranja i intenzivnog nasleđivanja klasa.
- Od novijih verzija Jave (Jave 6) preporučuje se upotreba posebne anotacije (informacije koje nisu deo samog programa) `@override`
- Anotacija `@override` jasno ukazuje na to da se metod redefiniše ali je dodatno olakšica i za kompajler jer on može lakše da otkrije probleme u nasleđivanju i javi adekvatne greške (npr. ako je ime redefinisanog metoda pogrešno navedeno ili nije adekvatan broj i tip parametara)

Nasleđivanje

Proširenje klase Tacka

```
class ObojenaTacka extends Tacka {

  private String boja;

  ObojenaTacka(float x, float y, String boja) {
    super(x, y);
    this.boja = boja;
  }

  void oboji(String novaBoja) {
    boja = novaBoja;
  }

  String getBoja() {
    return boja;
  }

  @Override
   String opis() {
    return "obojena " + super.opis() + " boje " + boja;
  }
}
```

Nasleđivanje - privatni članovi

- Podklasa ne nasleđuje privatne članove svoje nadklase.
- U podklasi se ne može direktno pristupati članovima koji su u nadklasi deklarisani kao privatni.
- Međutim, ti članovi postoje i u podklasi, samo nisu vidljivi.
- Podklasa pomoću nasleđenih metoda nadklase pristupa privatnim poljima.
- Npr. objekat klase **ObojenaTacka**, može da pozove metod **getX** i tako pristupi polju **x** deklarisanom u klasi **Tacka,** iako to polje nije vidljivo u klasi **ObojenaTacka.**

Nasleđivanje - dinamičko vezivanje (1)

- Promenljiva čiji je tip neka klasa može kao svoju vrednost dobiti referencu bilo kog objekta čiji je tip ta klasa ili neka njena podklasa.

Primer – i obojena tačka je tačka

```
class program {
  public static void main(String[] args) {
    ObojenaTacka ot = new ObojenaTacka(2, 3, "crvena");
    Tacka t = ot;
    System.out.println( t.opis() );
  }
}
```

- Dinamičko vezivanje je tehnika koja se koristi u modernim objektno-orijentisanim programskim jezicima da bi se tek u toku izvršavanja programa odredili tipovi objekata.

- Za razliku od dinamičkog vezivanja, *statičko vezivanje* se vrši još u toku prevođenja programa, tako da u programskim jezicima koji ga koriste tipovi svih vrednosti koji se u programu pojavljuju moraju biti određeni još u toku prevođenja.

Nasleđivanje - dinamičko vezivanje (2)

Primer – nije svaka tačka obojena tačka casting

```
class program {
  public static void main(String[] args) {
    ObojenaTacka ot = new ObojenaTacka(5, 2, "ruzicasta");
    Tacka t = ot;
    ObojenaTacka ot1 = (ObojenaTacka) t;
    System.out.println( ot1.opis() );
  }
}
```

Važi i obrnuto ali,

- promenljiva tipa **ObojenaTacka** može kao svoju vrednost dobiti vrednost promenljive tipa **Tacka**,
  - ta vrednost je referenca objekta koji je tipa **ObojenaTacka** (ili neke njene podklase).
  - obavezna je primena eksplicitne konverzije (**cast** operatora).

Nasleđivanje - dinamičko vezivanje (3)

`Instaceof` operatorom proverava se da li je promenljiva odgovarajućeg tipa

Primer – `instaceof` operator

```
class program {
  public static void main(String[] args) {
    ObojenaTacka ot = new ObojenaTacka(1, 2, "sarena");
    Tacka t = ot;
    //...
    // deo programa gde promenljiva t moze dobiti novu vrednost
    //...
    if (t instanceof ObojenaTacka) {
      ObojenaTacka ot1 = (ObojenaTacka) t;
      System.out.println( ot1.opis() );
    } else {
      System.out.println("Tacka nije obojena, dodela se ne moze izvrsiti");
    }
  }
}
```

Nasleđivanje - polimorfizam

- Polimorfizmom se predstavlja osobina jezika koja omogućava da se poziv nekog metoda promenljive referencijalnog tipa može izvršiti na više načina, u zavisnosti od tipa objekta čija referenca je vrednost te promenljive.

Primer – poziv metoda opis()

```
class Tacka {
...
  String opis() {
    return "tacka sa koordinatama (" + x + ", " + y + ")";
  }
}

class ObojenaTacka extends Tacka {
...
  String opis() {
    return "obojena " +
super.opis() + " boje " + boja;
  }
}
```

Nasleđivanje - ključna reč `super`

- Razlikujemo dva načina upotrebe:
  - Ako se nakon ključne reči **super** u zagradama navedu neke vrednosti, tada se poziva konstruktor direktne nadklase tekuće klase. Ovakva primena ključne reči **super** je moguća samo kao prva naredba u telu konstruktora klase. U tom slučaju omogućena je inicijalizacija polja, nasleđenih iz nadklase.

```
ObojenaTacka(float x, float y, String boja) {
    super(x, y);
    this.boja = boja;
}
```

  - Ključna reč **super** se može koristiti i ako hoćemo da pristupimo nekom metodu direktne nadklase koji je u podklasi redefinisan.

```
 String opis() {
    return "obojena " + super.opis() + " boje " + boja;
}
```

Nasleđivanje - ključna reč `final`

- Ključnu reč **final** možemo koristiti pri deklaraciji
  - **klasa,**
  - **metoda ili**
  - **promenljivih**
- Ako hoćemo da zabranimo nasleđivanje neke ***klase***, tada je deklarišemo pomoću ključne reči **final.**
- Ako hoćemo da dozvolimo nasleđivanje neke klase, ali bismo da sprečimo redefinisanje nekog njenog ***metoda*** u podklasama, tada taj metod deklarišemo sa **final.**

Nasleđivanje - ključna reč `final`

- Deklarisanje ***promenljivih*** pomoću **final** modifikatora se ne odnosi na nasleđivanje, već ovim modifikatorom određujemo promenljive koje neće menjati svoju vrednost – one će se ponašati kao konstante:
  - ***Statičko final polje*** – dodela vrednosti odmah kod deklaracije ili u statičkom inicijalizatoru klase
  - ***Nestatičko final polje*** – dodela vrednosti odmah kod deklaracije ili u inicijalizatoru objekta ili u svim konstruktorima
  - ***Lokalne promenljive*** – dodela vrednosti se može realizovati i kasnije, ali se posle dodele ne može više menjati
  - ***Parametri metoda i konstruktora*** - ako se njihova vrednost ne menja u telu metoda odnosno konstruktora

Nasleđivanje - `final` klase

Primer – `final` klasa

```
final class Pravougaonik {

  // ova klasa sluzi samo za pravougaonike cije su stranice paralelne
  // sa koordinatnim osama
  private float gore, dole, levo, desno;   //koordinate stranica

  //konstruktor
  Pravougaonik(float gore, float dole, float levo, float desno) {
    this.gore = gore;
    this.dole = dole;
    this.levo = levo;
    this.desno = desno;
  }

  float povrsina() {
    return (gore - dole) * (desno - levo);
  }

  // ostali clanovi klase
  // ...

}
```

Nasleđivanje - `final` metodi

Primer – `final` metodi

```
class Autoput {

  private int brojTraka = 3;

  final void postaviBrojTraka(int noviBrojTraka) {
    brojTraka = noviBrojTraka;
  }

  final int citajBrojTraka() {
    return brojTraka;
  }
}
```

- Ako hoćemo da dozvolimo nasleđivanje klase, ali bismo da sprečimo redefinisanje nekog metoda – taj metod deklarišemo sa **final.**

Final promenljive

Primer – final promenljive

```
class Automobil {

  static final int brojTockova = 4;  // final polje
  private String marka;

  void upisiMarku(final String marka) {  // final parametar
    this.marka = marka;
  }

  String citajMarku() {
    return marka;
  }

  String opis() {
    return "Automobil marke " + marka;
  }
}

class program {
   public static void main(String[] args) {
    final Automobil a = new Automobil(); // final lokalna promenljiva
    a.upisiMarku("BMW");
    System.out.println(a.opis());
  }
}
```

Nasleđivanje - ključna reč `abstract`

- **Apstraktna klasa** - klasa čija je jedina svrha to da je neke druge klase nasleđuju.
- Pravljenje instanci apstraktne klase **`new`** operatorom nije moguće. Ako bismo to pokušali, Java prevodilac bi prijavio grešku.
- **Najčešće sadrži u sebi implementaciju zajedničkih delova za te klase**, dok se implementacija specifičnosti pojedinih klasa naslednica ostavlja za kasnije.
- Apstraktna klasa može da sadrži apstraktne metode, ali i ne mora. Ako neka klasa sadrži bar jedan apstraktni metod, tada ona takođe mora biti apstraktna
- **Deklaracija apstraktnog metoda** se sastoji samo od reči **abstract**, zaglavlja metoda i tačke-zareza.
- Svaka neapstraktna klasa koja je podklasa apstraktne klase sa apstraktnim metodima mora sadržati punu deklaraciju tih metoda.

Nasleđivanje - ključna reč `abstract` - primer(1/6)

Primer – klasa PrevoznoSredstvo

```
abstract class PrevoznoSredstvo {

  private int maxBrzina;

  void staviMaxBrzinu(int maxBrzina) {
    this.maxBrzina = maxBrzina;
  }

  int uzmiMaxBrzinu() {
    return maxBrzina;
  }

  abstract String uString();
}
```

Nasleđivanje - ključna reč `abstract` - primer(2/6)

Primer – klasa Bicikl

```
class Bicikl extends PrevoznoSredstvo {

  private int brojStepenaPrenosa = 1;  //default vrednost

  void staviBrojStepenaPrenosa(int br) {
    brojStepenaPrenosa = br;
  }

  int uzmiBrojStepenaPrenosa() {
    return brojStepenaPrenosa;
  }

  String uString() {
    return "Bicikl sa maksimalnom brzinom " + uzmiMaxBrzinu() +
" i " + brojStepenaPrenosa + " stepena prenosa";
  }
}
```

Nasleđivanje - ključna reč `abstract` - primer(3/6)

Primer – klasa MotornoVozilo

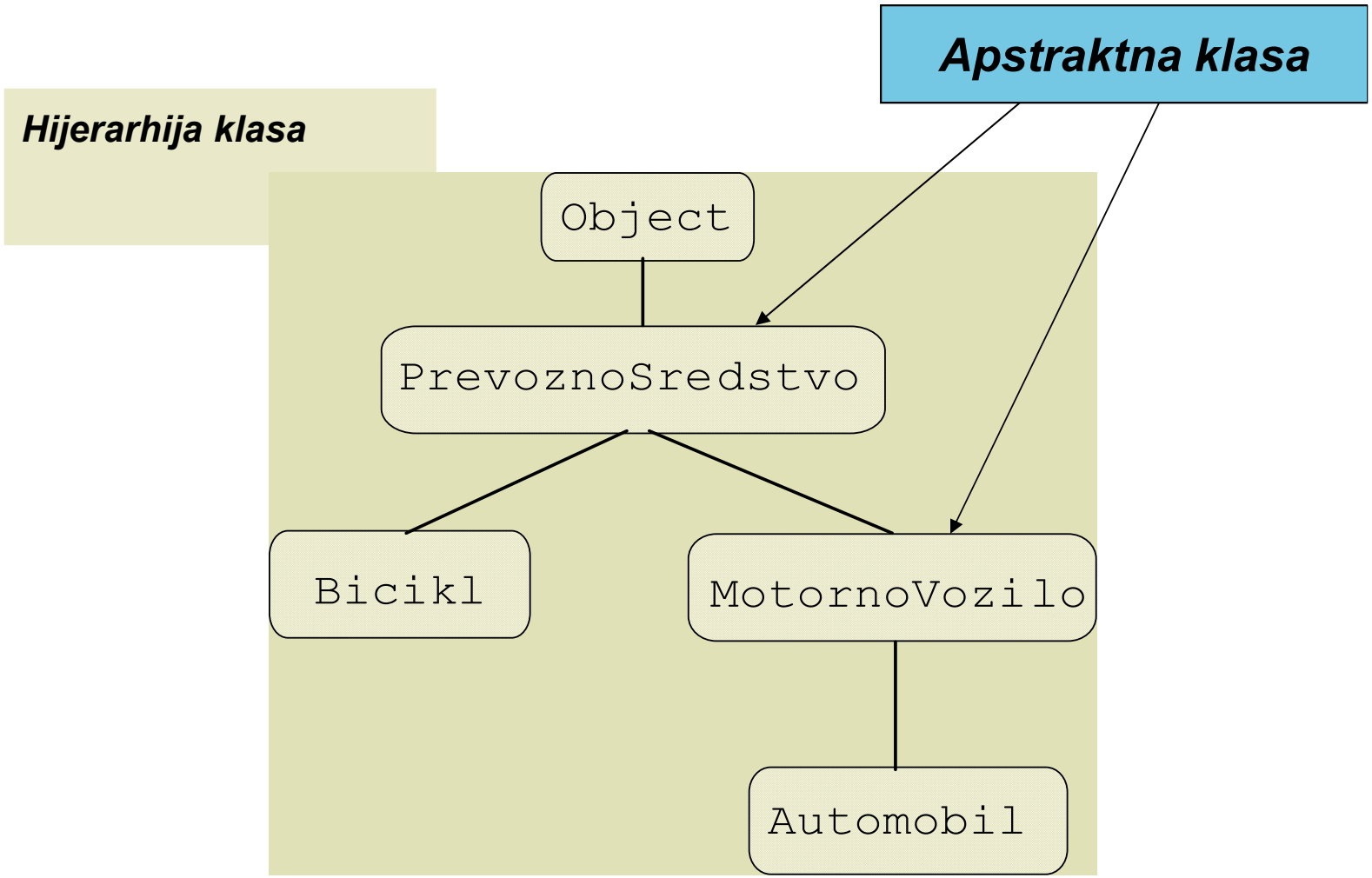
```
abstract class MotornoVozilo extends PrevoznoSredstvo {

  private float potrosnja;

  void staviPotrosnju(float potrosnja) {
    this.potrosnja = potrosnja;
  }

  float citajPotrosnju() {
    return potrosnja;
  }
}
```

Nasleđivanje - ključna reč `abstract` - primer(4/6)

Primer – klasa Automobil

```
class Automobil extends MotornoVozilo {

  private int brojSedista;

  void staviBrojSedista(int br) {
    brojSedista = br;
  }

  int uzmiBrojSedista() {
    return brojSedista;
  }

  float potrosnjaPoSedistu() {
    return citajPotrosnju() / brojSedista;
  }

  String uString() {
    return "Automobil sa maksimalnom brzinom " + uzmiMaxBrzinu() +
           ", potrosnjom " +  citajPotrosnju() + " i brojem sedista "
           + brojSedista;
  }
}
```

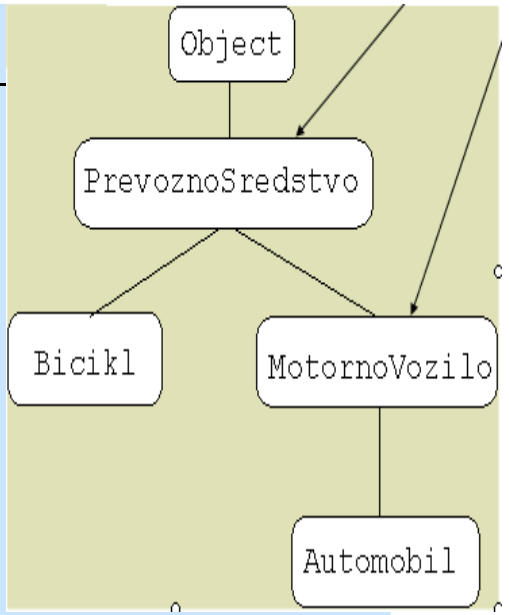
Nasleđivanje - ključna reč `abstract` - primer(5/6)



Nasleđivanje - ključna reč `abstract` - primer(6/6)

Primer – upotreba napravljenih klasa

```
class program {
  public static void main(String[] args) {
    Bicikl b = new Bicikl();
    b.staviMaxBrzinu(70);

    Automobil a = new Automobil();
    a.staviMaxBrzinu(210);
    a.staviPotrosnju(7);
    a.staviBrojSedista(4);

    PrevoznoSredstvo pz = b;
    System.out.println( pz.uString() );

    pz = a;
    System.out.println( pz.uString() );

    Automobil a1 = (Automobil) pz;
    a1.staviBrojSedista(2);
    System.out.println( a1.uString() );
  }
}
```



Ispis

```
Bicikl sa maksimalnom brzinom 70 i 1 stepena prenosa
Automobil sa maksimalnom brzinom 210, potrosnjom 7.0 i brojem sedista 4
Automobil sa maksimalnom brzinom 210, potrosnjom 7.0 i brojem sedista 2
```

Paketi

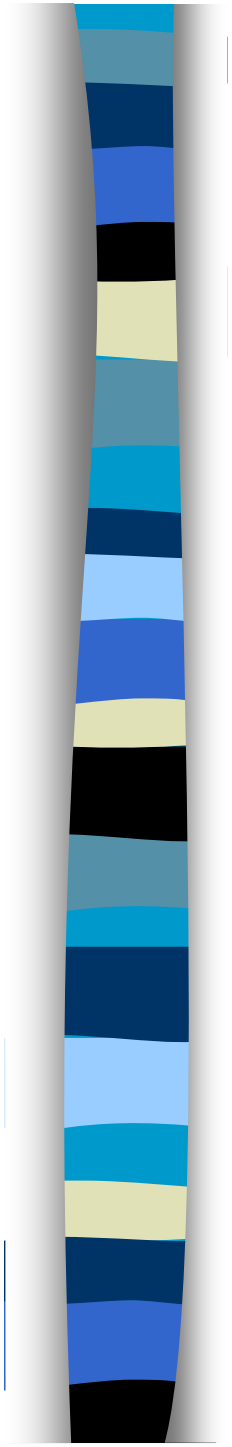
- Program u Javi je skup klasa i interfejsa od kojih bar jedna klasa sadrži metod

```
public static void main (String [] args).
```

- Obično se klase i interfejsi ne prave samo zbog jednog programa, već da bi se iskoristili i u drugim, budućim programima.
- Takve klase i interfejse smeštamo u posebne biblioteke, koje se u Javi zovu ***paketi*.**

Paketi

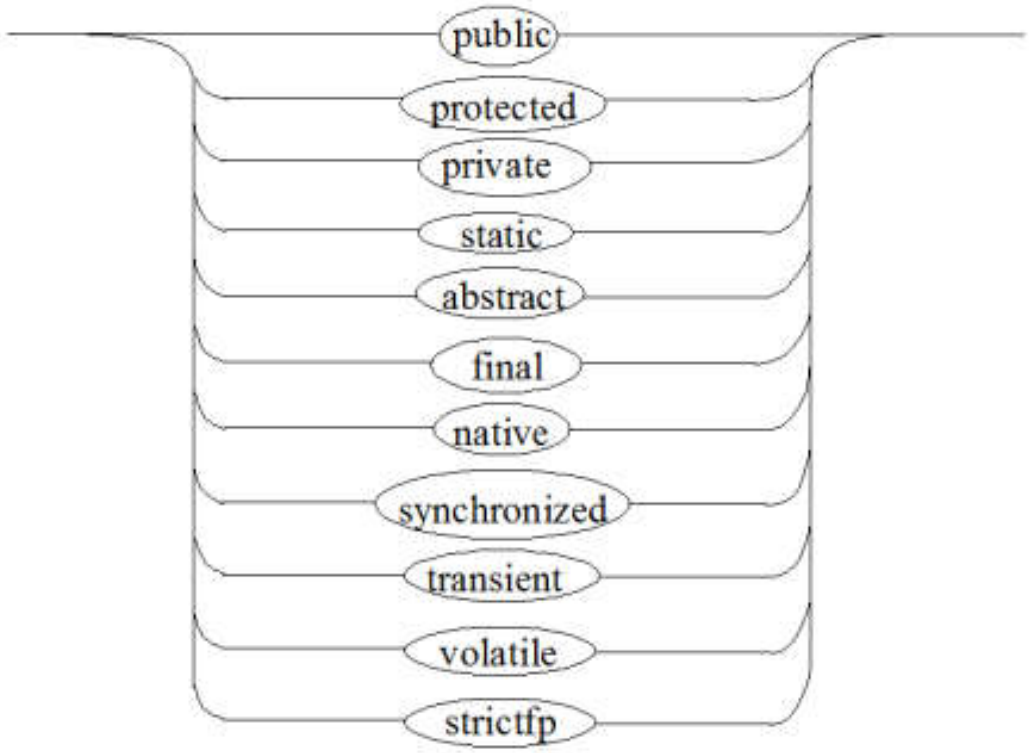
- Osim korisnički napravljenih paketa, postoje i standardni paketi koji se isporučuju zajedno sa Java prevodiocem - **Java API** (*Application Programming Interface*)
- Svaka napravljena klasa i svaki napravljen interfejs u Javi pripadaju nekom paketu. Ako ne navedemo ime paketa kom napravljena klasa treba da pripada, Java će je staviti u *anoniman paket*, kojeg čine sve klase i interfejsi napravljeni u tekućem direktorijumu računara
- U jedan paket najčešće stavljamo klase i interfejse koji su na neki logičan način povezani - **java.awt** (grafika), **java.io** (ulazno izlazne operacije), ...



Modifikatori

Modifikatori

- Modifikatori u Javi su ključne reči koje se koriste kod deklaracije klasa i interfejsa, kao i kod deklaracije konstruktora klasa, metoda i polja.



Modifikatori

- Modifikatori u Javi su ključne reči koje se koriste kod deklaracije klasa i interfejsa, kao i kod deklaracije konstruktora klasa, metoda i polja

- Najčešće korišćeni su modifikatori vidljivosti -
  public, private, protected

- Ostali modifikatori su vezani za:
  - nasleđivanje (**final, abstract**),
  - programiranje pomoću niti (**synchronized, volatile**),
  - serijalizaciju ili deserijalizaciju (**transient**),
  - poštovanje standarda (**strictfp**),
  - vrstu metoda ili polja (**static**),
  - korišćenje elemenata drugih programskih jezika (**native**)

Modifikatori - modifikatori vidljivosti

- Modifikatori koji utiču na vidljivosti članova klase su -
  - **public,**
  - **private,**
  - **protected**
- Član klase može biti deklarisan sa najviše jednim od ovih modifikatora tj. ovi modifikatori se isključuju.
- Najveću vidljivost članu klase pruža modifikator ***public,*** sledi modifikator ***protected***, malo manju vidljivost daje deklarisanje člana klase bez modifikatora, a najmanju vidljivost daje modifikator ***private***.

Modifikatori - modifikatori vidljivosti

Modifikator	Klasa / interfejs	Član klase
public	vidljivi u svim paketima	vidljivi za sve klase svih paketa
protected	/	vidljivi za sve klase svog paketa i za nasleđene klase ili interfejse iz bilo kog paketa
default ***(bez modifikatora)***	vidljivi samo u okviru svog paketa	vidljivi za sve klase u istom paketu
private	/	vidljivi samo u okviru svoje klase

Modifikatori - ostali modifikatori

- Modifikator **static** služi za deklarisanje statičkih delova klase tj. elemenata koji su vezani isključivo za klasu i ne javljaju se u instancama klasa.
- Modifikatore **abstract** i **final** smo već opisali.
- Ako hoćemo da napišemo program u Javi, ali hoćemo da iskoristimo i gotove programe koje smo pisali na nekom drugom programskom jeziku, tada možemo taj postojeći kod da uključimo u Java program pomoću **native** metoda.

Modifikatori - ostali modifikatori

- Modifikator metoda **synchronized** i modifikator polja **volatile** se koriste za sinhronizaciju niti koje se paralelno izvršavaju.
- Modifikatorom **transient** možemo označiti polja klase koja nisu bitna za predstavljanje stanja nekog objekta - ta polja se preskaču u postupcima serijalizacije i deserijalizacije objekta
  - Serijalizacija – predstavljanje objekta nizom bajtova
  - Deserijalizacija – od niza bajtova dobijenih serijalizacijom ponovo se dobija objekat
- Modifikator **strictfp** se koristi ako hoćemo da se sve operacije nad brojevima tipa **float** i **double** u datoj klasi, interfejsu ili metodu odvijaju striktno po standardu koji nalaže IEEE 754 format zapisa realnih brojeva.

Modifikatori - ostali modifikatori

- Standard IEEE 754 propisuje načine zapisa i interpretaciju realnih brojeva u pokretnom zarezu na računarima.
- Danas je najšire korišćen standard integrisan u mnoge procesore (engl. CPU, Central processing unit) i jedinice za obradu brojeva u pokretnom zarezu (engl. FPU, Floating point unit).
- Standard definiše formate raznih veličina i specijalne vrednosti (poput beskonačnosti i nemogućeg broja).
- Takođe određuje i četiri moda zaokruživanja brojeva i pet izuzetaka (specijalnih slučajeva).
- .

Teorijske vežbe 3

Objektno-orjentisano programiranje

Zadatak 6

- Klasom Kladionicar predstavljen je jedan igrač u kazinu.
- Ova klasa kao atribute ima:
  - ime igrača (String),
  - ime igre koju trenutno igra (String) - može imati vrednosti "poker", "ajnc" i "rulet" (set metoda ne dozvoljava neku drugu vrednost, setuju se samo ispravne vrednosti, ili "poker" ako vrednost nije ispravna), i
  - stanje na računu (int).
- Klasa jos sadrži i konstruktor koji inicijalizuje sve atribute, get i set metode za svaki atribut i toString() metod.

Zadatak 6

- Klasom Kazino predstavljen je kazino u kom se igrači klade i sadrži kao svoje polje niz kladionicara.
- toString() metod poziva toString() metod svih kladioničara u nizu.
- Klasa definiše i metod sa sledećim zaglavljem:

  - String najprofitnijaIgra() - vraća naziv igre koju igra najveći broj kladioničara sa računom u minusu. Ako ima više takvih igara, metod vraća prvu na koju je naišao.

Zadatak 6

- U glavnom programu potrebno je napraviti instancu klase Kazino, inicijalizovati je sa nizom kladioničara i ispisati rezultate izvršavanja metoda toString() i najprofitnijaIgra().
- Rezultat izvršavanja treba da bude sledeći:
  - Pera igra poker i ima 15000 na racunu.
  - Mika igra ajnc i ima 17000 na racunu.
  - Zika igra rulet i ima -1500 na racunu.
  - Bora igra ajnc i ima -11000 na racunu.
  - Dora igra rulet i ima 5000 na racunu.
  - Zora igra ajnc i ima -3000 na racunu.
  - Lara igra poker i ima 22000 na racunu.
  - Mara igra poker i ima 7000 na racunu.

  - Najprofitnija igra trenutno je ajnc.

Zadatak 6 – klasa Kladionicar

```
public class Kladionicar {

    private String ime;
    private String igra;
    private int stanje;

    public Kladionicar(String ime, String igra, int stanje) {
        this.ime = ime;
        setIgra(igra);
        this.stanje = stanje;
    }

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getIgra() {
        return igra;
    }
```

Zadatak 6 – klasa Kladionicar

```
public class Kladionicar {

    private String ime;
    private String igra;
    private int stanje;

    public Kladionicar(String ime, String igra, int stanje) {
        this.ime = ime;
        setIgra(igra);
        this.stanje = stanje;
    }

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getIgra() {
        return igra;
    }
```

Zadatak 6 – klasa Kladionicar

```
    public void setIgra(String igra) {
        if (igra.equals("poker") || igra.equals("ajnc") || igra.equals("rulet")) {
            this.igra = igra;
        } else {
            this.igra = "poker";
        }
    }

    public int getStanje() {
        return stanje;
    }

    public void setStanje(int stanje) {
        this.stanje = stanje;
    }

    @Override
    public String toString() {
        return ime + " igra " + igra + " i ima " + stanje + " na racunu";
    }

}
```

Zadatak 6 – klasa Kazino

```
public class Kazino {

    private Kladionicar[] kladionicari;

    public Kazino(Kladionicar[] kladionicari) {
        this.kladionicari = kladionicari;
    }

    @Override
    public String toString() {
        String toString = "";
        for (int i = 0; i < kladionicari.length; i++) {
            toString += kladionicari[i].toString() + "\n";
        }
        return toString;
    }
```

Zadatak 6 – klasa Kazino

```
    String najprofitnijaIgra() {
        int ajnc = 0;
        int poker = 0;
        int rulet = 0;
        for (int i = 0; i < kladionicari.length; i++) {
            if (kladionicari[i].getIgra().equals("ajnc") && kladionicari[i].getStanje() < 0) {
                ajnc++;
            } else if (kladionicari[i].getIgra().equals("poker") && kladionicari[i].getStanje() < 0) {
                poker++;
            } else if (kladionicari[i].getStanje() < 0) {
                rulet++;
            }
        }
        if (ajnc >= poker && ajnc >= rulet) {
            return "ajnc";
        }
        if (poker >= ajnc && poker >= rulet) {
            return "poker";
        }
        return "rulet";
    }
}
```

Zadatak 6 – klasa Glavna

```
public class Glavna {

    public static void main(String[] args) {
        Kazino kazino = new Kazino (
                new Kladionicar[] {
                        new Kladionicar("Pera", "poker", 15000),
                        new Kladionicar("Mika", "ajnc", 17000),
                        new Kladionicar("Zika", "rulet", -1500),
                        new Kladionicar("Bora", "ajnc", -11000),
                        new Kladionicar("Dora", "rulet", 5000),
                        new Kladionicar("Zora", "ajnc", -3000),
                        new Kladionicar("Lara", "poker", 22000),
                        new Kladionicar("Mara", "poker", -7000) });
        System.out.println(kazino);
        System.out.println("Najprofitnija igra trenutno je " +
        kazino.najprofitnijaIgra() + ".");
    }
}
```

Zadatak 7

- Klasom Radnik se opisuje jedan radnik radne organizacije.
- Klasa kao atribute sadrži
  - Identifikator radnika (String)
  - Radni staž u godinama (int)
  - Platu (double)
  - Referencu na nadređenog radnika
- Klasa takođe definiše konstruktor koji postavlja vrednosti atributa klase, get metode i toString() metod

Zadatak 7

- Klasa Direktor nasleđuje klasu Radnik. Direktor je radnik koji nema nadređenih.
- Klasa Direktor se ne može dalje nasleđivati.
- Klasa redefiniše toString() metod kako bi se jasno istaklo da je instanca te klase direktor.

Zadatak 7

- Klasa RadnaOrganizacija kao atribut sadrži niz radnika.
- Radna organizacija se učitava iz fajla koji je organizovan na sledeći način
  - Prva linija – broj radnika
  - U svakoj sledećoj liniji informacije o jednom radniku i to
    - ID, radni staž, plata i id nadređenog (razdvojeni zarezima) – za obične radnike
    - ID, radni staž, plata (razdvojeni zarezima) – za direktora
- **Za nekog radnika informacije o nadređenom radniku su uvek date pre informacija o samom tom radniku u ulaznom fajlu.**

Zadatak 7

- Klasa RadnaOrganizacija takođe definiše dve metode
  - **void stampajSveNadređene(String id)** – koja štampa sve nadređene radnike za nekog radnika
  - **void plataNadredjenogManja()** – koja štampa sve radnike koji imaju veću platu od njihovog direktnog nadređenog

Primer ulaznog fajla

7

Mika, 10, 12000.34

Pera, 7, 8000.5, Mika

Zika, 9, 9000, Mika

Tika, 5, 9500.57, Mika

Cale, 12, 9300, Pera

Sale, 3, 1000, Pera

Brale, 8, 9600, Tika

```java
public class Radnik {
	private String id;
	private int radniStaz;
	private double plata;
	private Radnik nadredjeni;

	public Radnik(
		String id, int radniStaz, double plata, Radnik nadredjeni
	) {
		this.id = id;
		this.radniStaz = radniStaz;
		this.plata = plata;
		this.nadredjeni = nadredjeni;
	}

	public String getId()          { return id;          }
	public int getRadniStaz()      { return radniStaz;   }
	public double getPlata()       { return plata;       }
	public Radnik getNadredjeni() { return nadredjeni; }

	public String toString() {
		return id + ", " + radniStaz + ", " + plata;
	}
}
```

```java
public final class Direktor extends Radnik {

	public Direktor(
		String id,
		int radniStaz,
		double plata)
	{
		super(id, radniStaz, plata, null);
	}

	public String toString() {
		return "Direktor " + super.toString();
	}
}
```

```
public class RadnaOrganizacija {
	private Radnik[] radnici;

	public RadnaOrganizacija(String spisakRadnika)
		throws IOException
	{
		BufferedReader br = new BufferedReader(
			new FileReader(spisakRadnika));

		int brRadnika = Integer.parseInt(br.readLine());
		radnici = new Radnik[brRadnika];

		for (int i = 0; i < brRadnika; i++) {
			String[] tokeni = br.readLine().split(",");

			// instanciranje objekata
			...
		}

		br.close();
	}

	// to be continued
}
```

```
String[] tokeni = br.readLine().split(",");
String id = tokeni[0].trim();
int radniStaz = Integer.parseInt(tokeni[1].trim());
double plata = Double.parseDouble(tokeni[2].trim());

if (tokeni.length == 4) {
	String idNadredjeni = tokeni[3].trim();
	Radnik nadredjeni = pronadjiNadredjenog(idNadredjeni, i);
	if (nadredjeni == null) {
		br.close();
		throw new IOException("Greska u ulaznom fajlu");
	}

	radnici[i] = new Radnik(id, radniStaz, plata, nadredjeni);
} else {
	radnici[i] = new Direktor(id, radniStaz, plata);
}
```

```
private Radnik pronadjiNadredjenog(
        String idNadr, int brojDodatihRadnika
) {
        for (int i = 0; i < brojDodatihRadnika; i++) {
                if (radnici[i].getId().compareTo(idNadr) == 0) {
                        return radnici[i];
                }
        }

        return null;
}
```

```
public void plataNadredjenogManja() {
	for (int i = 0; i < radnici.length; i++) {
		Radnik r = radnici[i];

		if (!(r instanceof Direktor)) {
			Radnik nadr = r.getNadredjeni();

			if (nadr.getPlata() < r.getPlata()) {
				System.out.println(
					r.getId() + " ima platu " + r.getPlata() +
					", a njegov nadredjeni " + nadr.getId() +
					" ima platu " + nadr.getPlata()
				);
			}
		}
	}
}
```

```
public void stampajSveNadredjene(String id) {
	Radnik r = null;
	for (int i = 0; i < radnici.length; i++) {
		if (radnici[i].getId().compareTo(id) == 0) {
			r = radnici[i];
			break;
		}
	}

	if (r == null) {
		System.out.println("Radnik " + id + " ne postoji");
	} else if (r instanceof Direktor) {
		System.out.println("Radnik " + id + " je direktor");
	} else {
		System.out.println("Nadredjeni radnici za " + id + " su: ");

		Radnik nadr = r.getNadredjeni();
		while (nadr != null) {
			System.out.println(nadr.getId());
			nadr = nadr.getNadredjeni();
		}
	}
}
```

```java
public static void main(String[] args)
        throws IOException
{
        RadnaOrganizacija ro = new RadnaOrganizacija("SpisakRadnika.txt");
        ro.stampajSveNadredjene("Sale");
        ro.plataNadredjenogManja();
}
```

```
Nadredjeni radnici za Sale su:
Pera
Mika
Cale ima platu 9300.0, a njegov nadredjeni Pera ima platu 8000.5
Brale ima platu 9600.0, a njegov nadredjeni Tika ima platu 9500.57
```