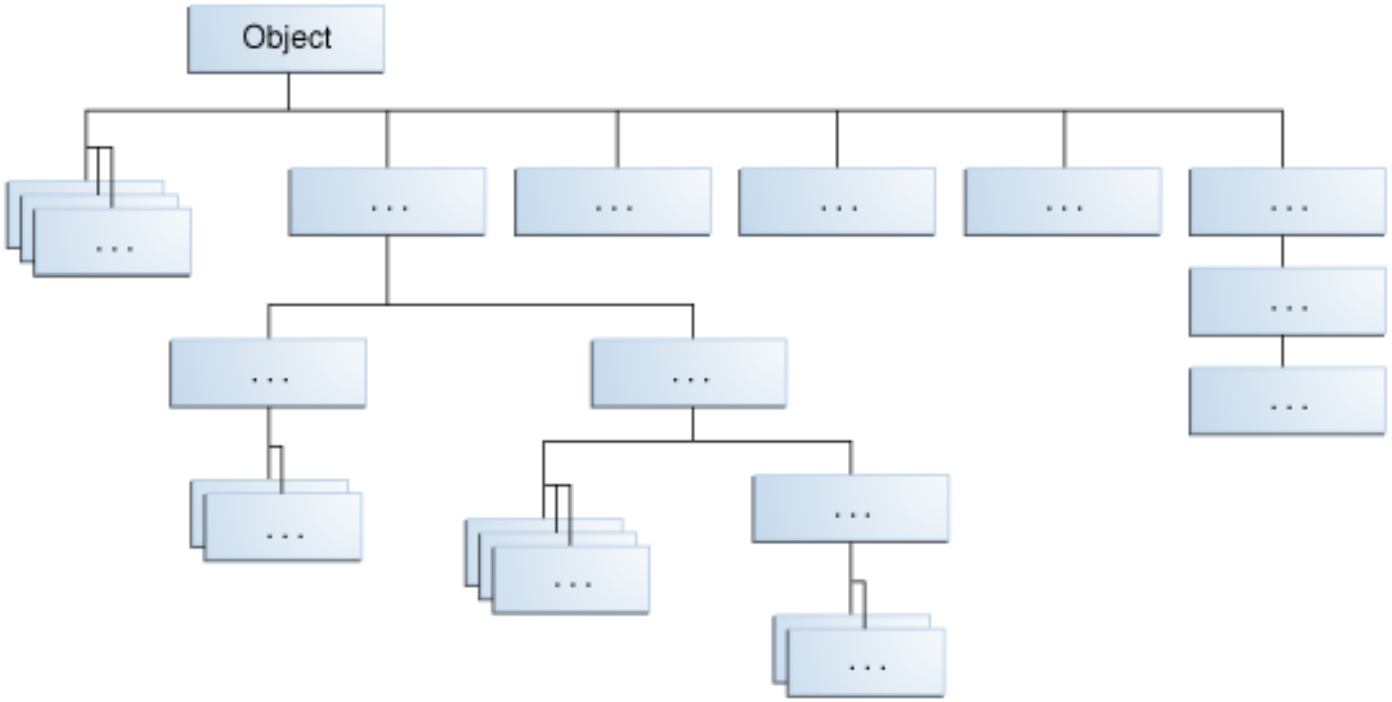
Nasleđivanje

Drvo nasleđivanja

- Ako klasa ne koristi extends u definiciji koga nasleđuje?
- Koja klasa ne nasleđuje nikoga?



Šta se može sa nasleđenom klasom (podklasom)?

- Mogu se koristiti nasleđena polja
- Može se definisati polje sa istim imenom kao u nadklasi, i na taj način **prikriti** polje tako da će polje iz nadklase neće biti vidljivo **(nije preporučljivo)**
- Mogu se definisati nova polja
- Mogu se koristiti nasleđene metode

Šta se može sa nasleđenom klasom (podklasom)?

- Mogu se napisati objektne metode sa istim potpisom i na taj način redefinisati (@Override)
- Može se napisati statička metoda sa istim potpisom. Pozivanje je: Nadklasa.statickaMetoda() ili Podklasa.statickaMetoda()
- Mogu se deklarisati nove metode
- Može se napraviti novi konstruktor koji koristi poziv ka starom (ključna reč **super**)

Kastovanje objekata (gore i dole po drvetu)

```
Bicikl b = new Bicikl();

// Kastovanje ka gore (uopštavanje)
Object o = b;

// Kastovanje ka dole (specijalizacija)
Bicikl b1 = (Bicikl) o;

// Šta je sa ovim?
Kamion k = (Kamion) o;

//Operator za proveru
if(o instanceof Kamion)
    k = (Kamion) o;
```

Prikrivanje i redefinsanje

```
public class Zivotinja {
    public static void testirajStaticku() {
        System.out.println(“Staticka metoda u Zivotinji");
    }
    public void testirajObjektnu() {
        System.out.println(“Objektna metoda u Zivotinji");
    }
}
```

Prikrivanje i redefinisanje

```
public class Macka extends Zivotinja {
    public static void testirajStaticku() {
        System.out.println(“Staticka metoda u Macka");
    }
    public void testirajObjektnu() {
        System.out.println(“Objektna metoda u Macka");
    }
    public static void main(String[] args) {
        Macka m= new Macka();
        Zivotinja z = m;
        Zivotinja.testirajStaticku();
        z.testirajObjektnu();
    }
}
```

Polimorfizam

- Princip po kome podklase zadržavaju svojstva nadklasa i pritom dodaju svoje sopstvena svojstva

```
class Bicikl{
    int brojPogona;

    @Override
    public String toString(){
        return “Bicikl sa “+brojPogona+” pogona”;
    }
}
```

Polimorfizam

```
class BrdskiBicikl extends Bicikl{
    int brojPogona;
    int nagib;

    @Override
    public String toString(){
        return super.toString()+” na nagibu ”+nagib;
    }
}
```

Prikrivanje polja

```
class Bicikl{
    int brojPogona;
}

class NekiBicikl extends{
    float brojPogona; //ovo prekriva polje iz
                                //nadklase
}
//u okviru podklase, polju iz nadklase se ipak moze
//pristupiti sa super.brojPogona
```

Korišćenje konstruktora nadklase

```
class BrdskiBicikl extends Bicikl{
   int nagib;

   public BrdskiBicikl(int pogona,int nagib){
        super(pogona);
        this.nagib=nagib;
   }
}
```

Ključna reč **final**

- Služi da se neki metod proglasi konačnim, tj. da se njegova implementacija ne može redefinisati kroz podklase

```
class SahAlgoritam {
    final String prviIgra() {
        return “Beli”;
    }
}
```

Abstraktni metodi i klase

- Abstraktan metod je onaj koji nema implementaciju

```
abstract class Platno{
    private int sirina, visina; //isto za sva platna
    private boolean[][] matrica;

    abstract void prikazi(); //abstraktni metod, zavisi od
                                //konkretnog platna

    void resetuj(){
        … //uvek isto, sve se postavlja na false
    }
}
```

- Kada koristiti abstraktne klase, a kada interfejse?

Zadatak

- Ispraviti sve greške u paketu koji se nalazi na adresi:
  htttp://www.math.rs/~kartelj/nastava/MGOOP2014/5/ispraviti_greske.zip