F8 - Arv

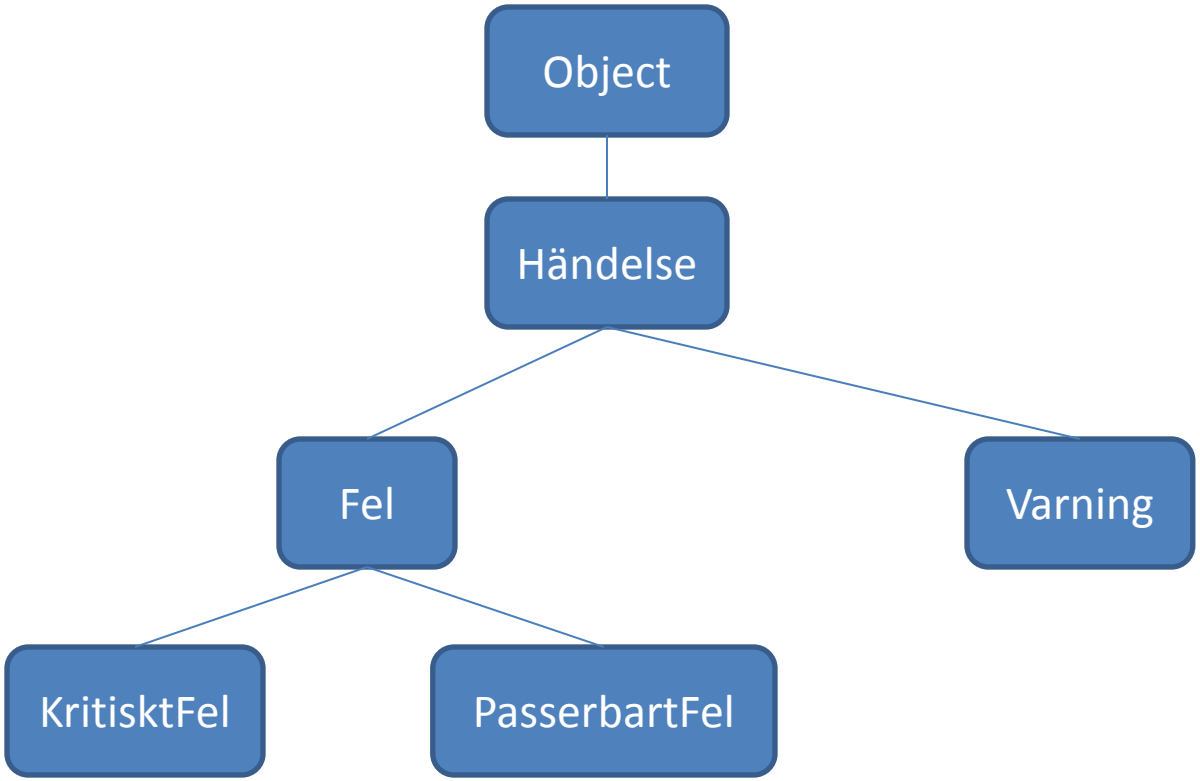
ID1004 Objektorienterad programmering

Fredrik Kilander fki@kth.se

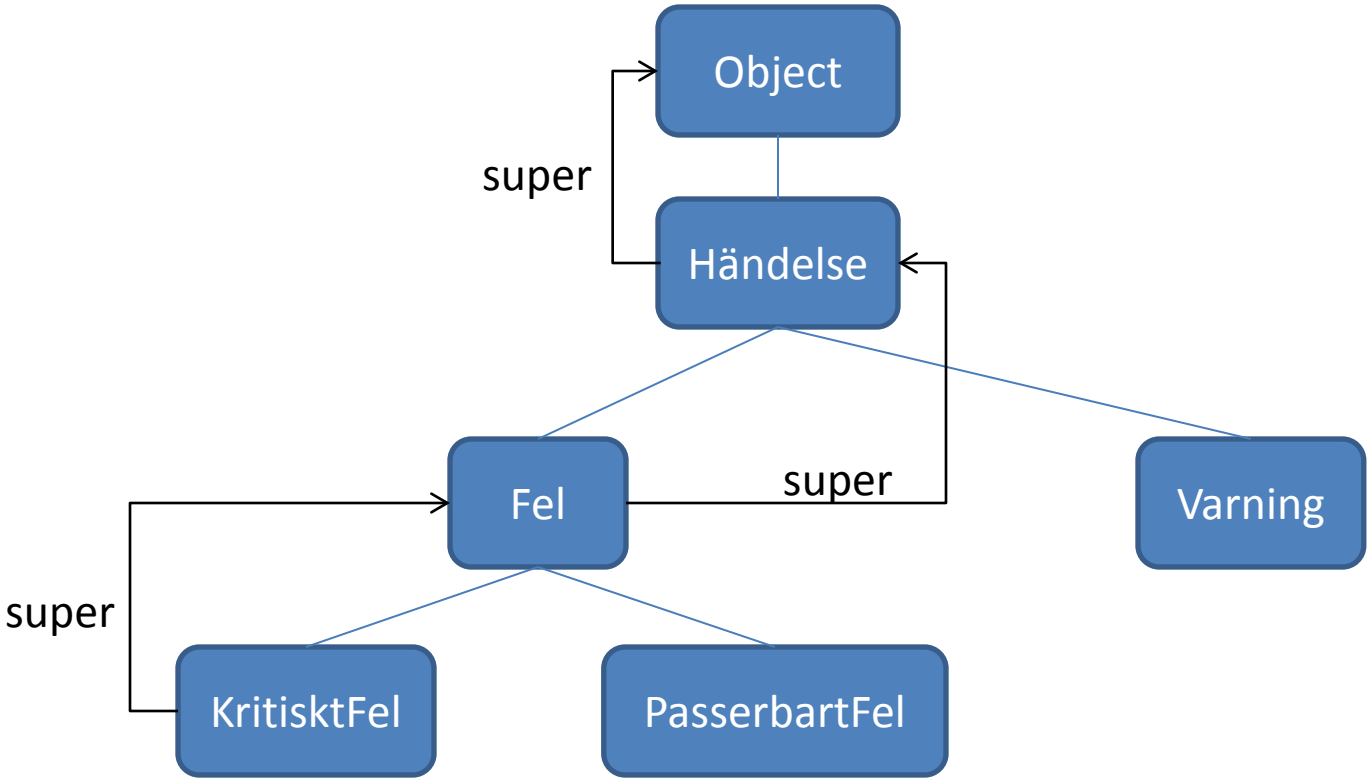
Arv och subklasser

- Klasser innehåller attribut och beteenden
- En subklass ärver dessa från föräldern
- Detta ger:
  - Återanvänd kod
  - Specialisering

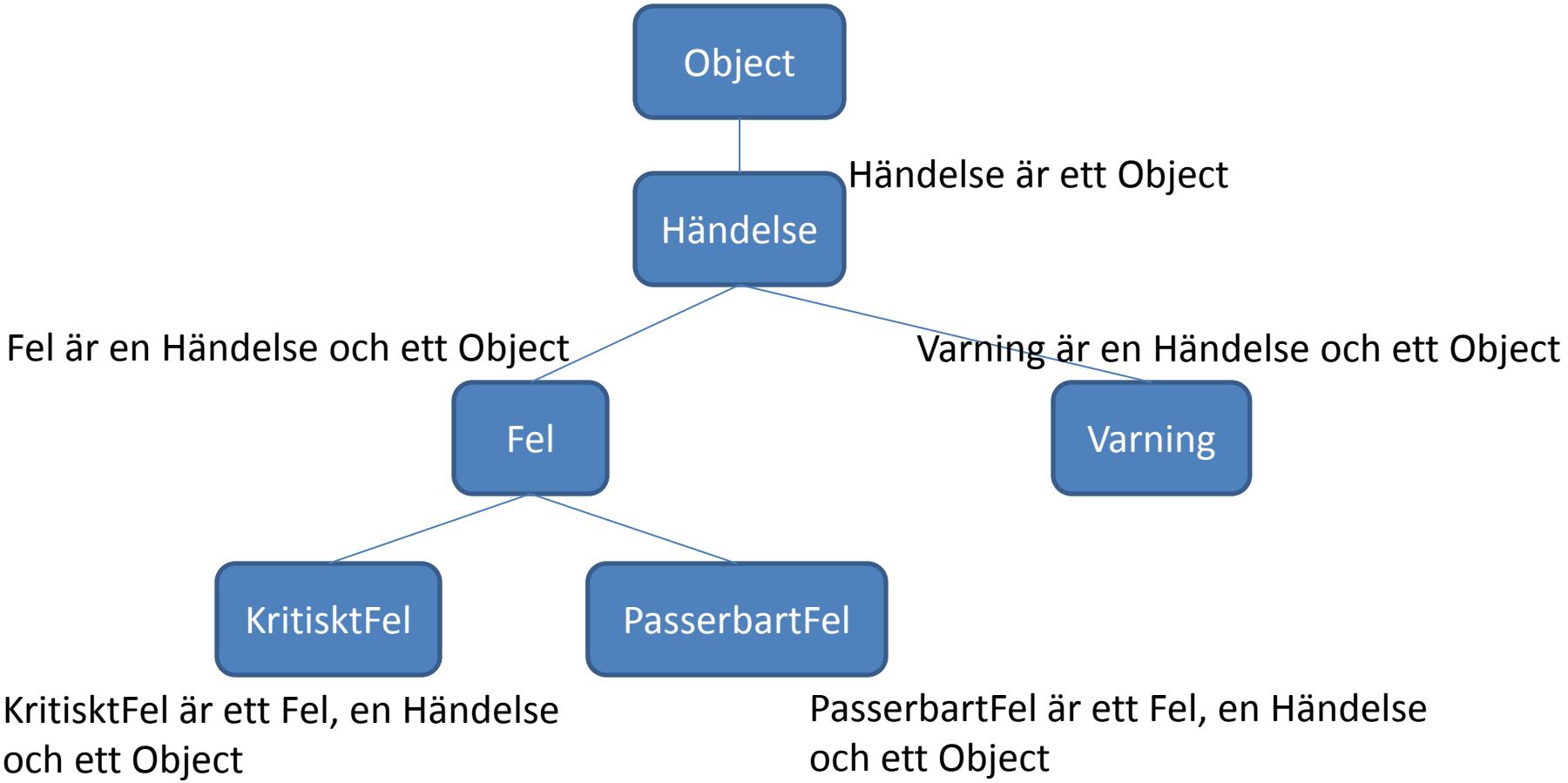
Arv och subklasser



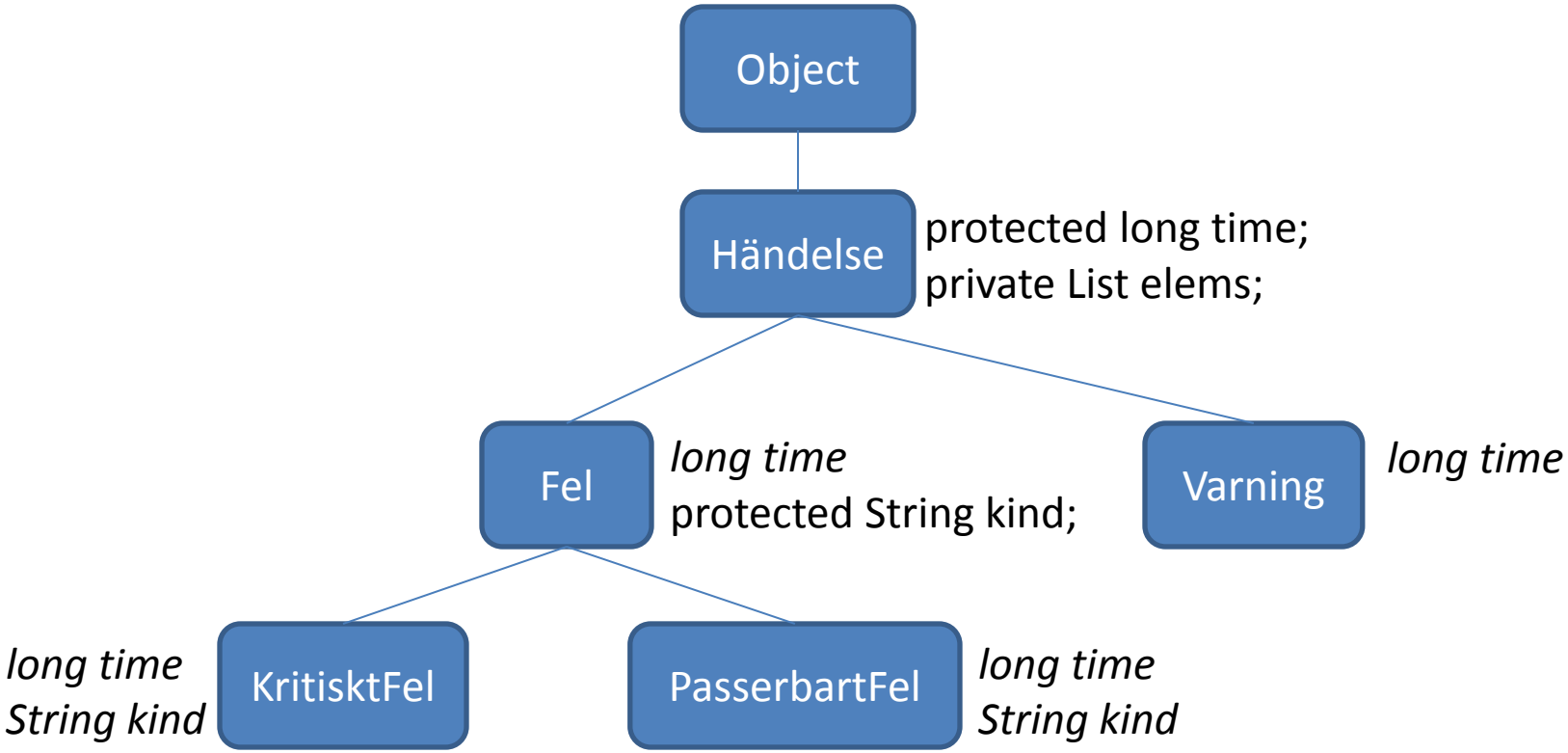
Arv och subklasser



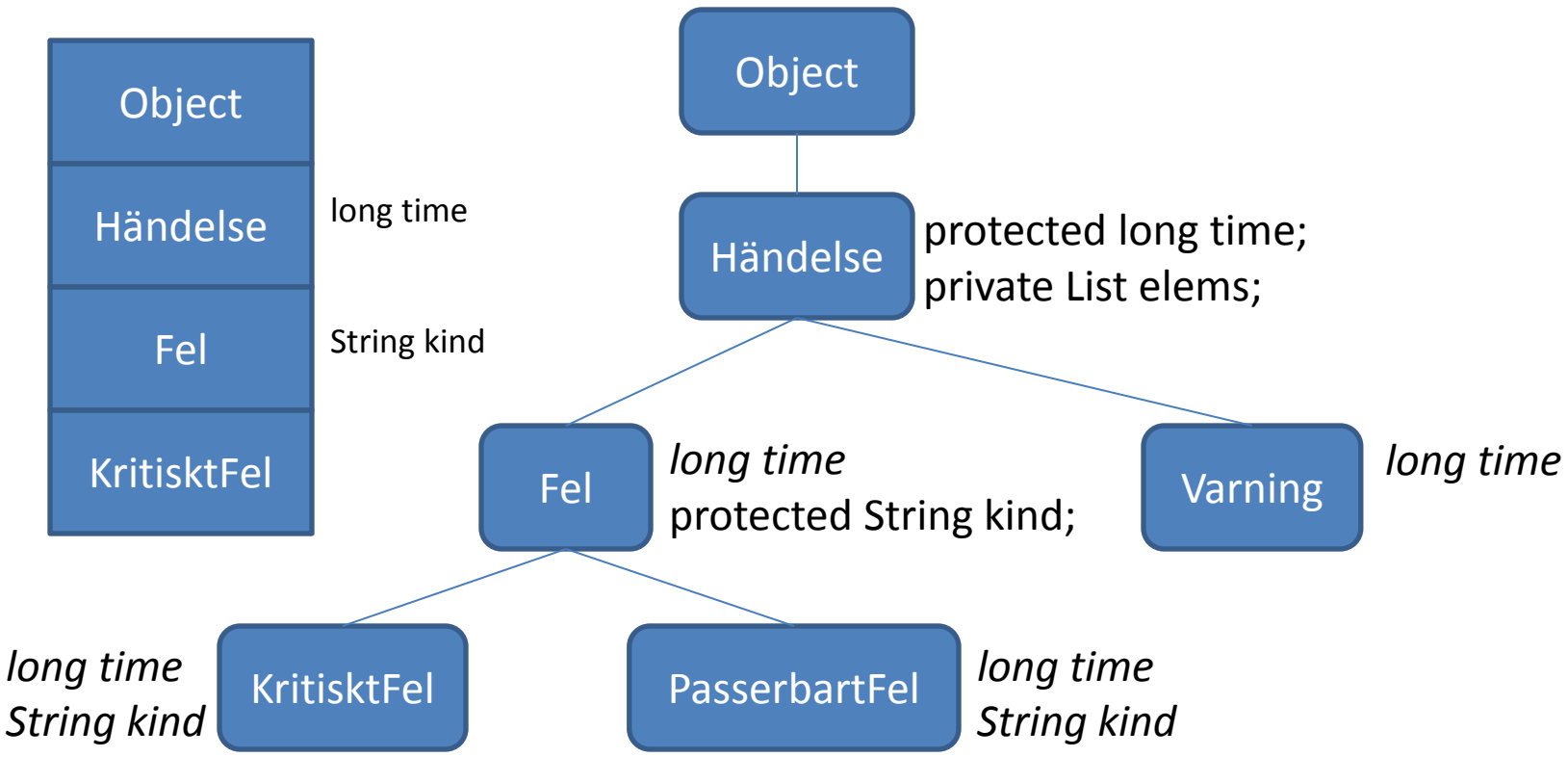
Arv och subklasser – relationen is-a



Arv och subklasser – protected vs private



Arv och subklasser – protected vs private



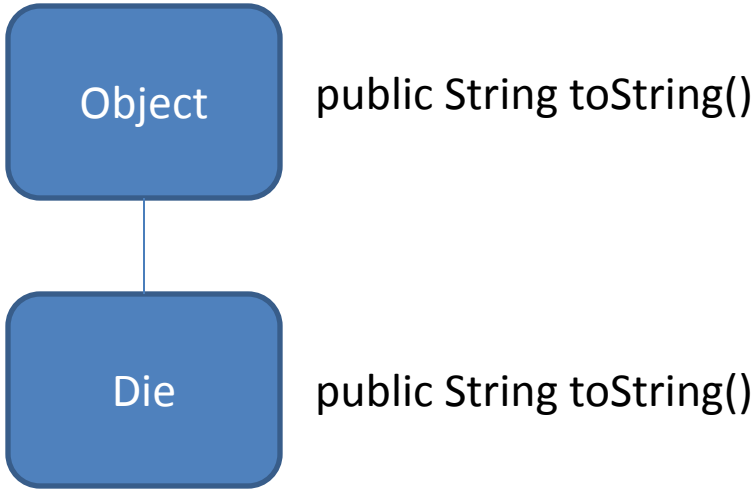
Arv, härkomst och beteende

- Varje klass kan bara ärva från en förälder
  - Strikt hierarki

- En klass kan implementera flera interface
  - Flera separate beteenden

Åsidosatta metoder

- eng *Overriding Methods*
- En subklass kan ha en egen version av en metod definierad av någon förälder
- Subklassen åsidosätter förälderns version



Klass Die har en egen version av toString(). Metoden är åsidosatt av klass Die.

Åsidosatta metoder

```
public class Hund {
  public String toString(){
    return ”En hund”;
}
```

```
System.out.println(new Hund());
```
En hund

```
public class Tax extends Hund{
}
```

```
System.out.println(new Tax());
```
En hund

```
public class Boxer extends Hund{
  public String toString(){
    return ”En boxer”;
  }
}
```

```
System.out.println(new Boxer());
```
En boxer

Åsidosatta metoder

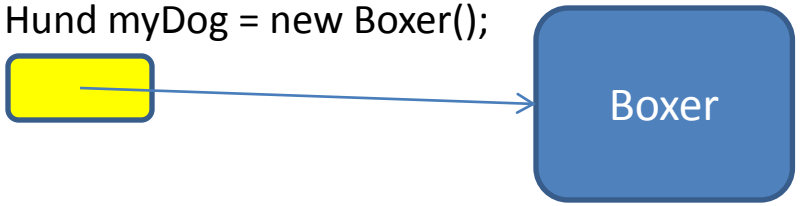
toString()

toString()

toString()

Den mest specifika (lägsta) metoden anropas

Hund myDog = new Boxer();




```
myDog.toString()
```

Åsidosatta metoder - super

toString()

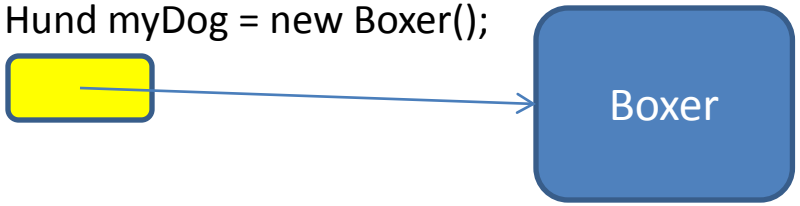
toString()

toString()

Den mest specifika (lägsta) metoden anropas. Men subklassen kan referera till sin förälder med **super**

```
String toString() {
  return ”En boxer, men även ” +
    super.toString();
}
```

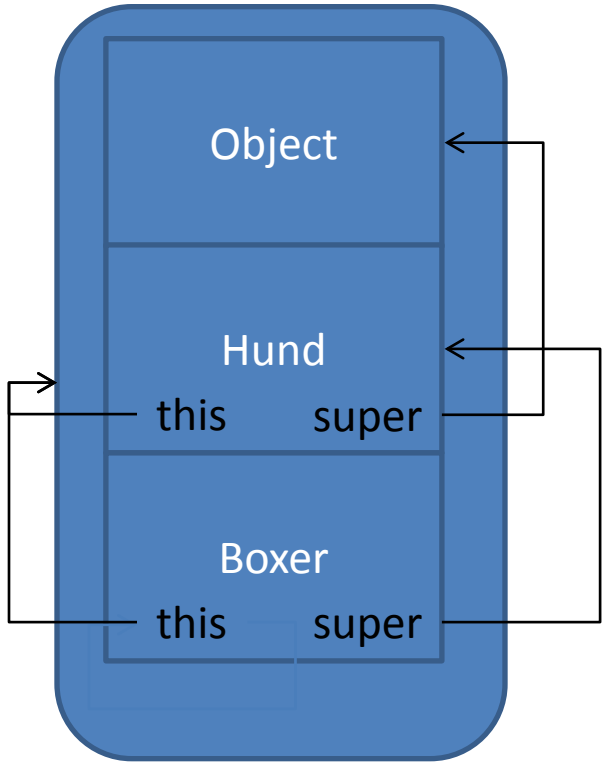
Hund myDog = new Boxer();




```
myDog.toString()
```

En boxer, men även En hund

Åsidosatta metoder – super och this



- **`super`** låter subklassen anropa förälderns version av metoden
- bara en nivå; **`super.super`** går inte
- **`this`** refererar alltid till instansen
- **`super`** används bl a för att köra förälderns konstruktor

JFrame(String title) // konstruktor

…

```
public Chart extends JFrame {
   ...
   public Chart(String title) {
     super(title);
     ...
}
```

Förhindra åsidosättning – final

- Metoder kan skyddas från åsidosättning
- Metoden deklareras då **`final`**

```
public final String getVotingResult() {
   ...
}
```

Skuggvariabler

- En klass har en variabel
- Subklasser kan använda den
- Subklasser kan deklarera en egen variabel med samma namn och typ
- Denna kallas då *skuggvariabel*

- Leder till förvirring och bör undvikas

Skuggvariabler

```
public class Hund{
  protected String name;
  protected float hunger=22;
  ...

}
```

skuggning

```
public class Boxer extends Hund{
  private float hunger=0.1;
  ...
  ...println(hunger);
}
```

skriver ut 0.1

Kompilatorn kan lätt hålla isär de två variablerna.
Människor blir lätt förvirrade – undvik skuggvariabler.

Klassen Object

- Alla klasser ärver från Object bl a:
- **`boolean equals(Object o)`**
  - obj1 == obj2
- **`String toString()`**
  - Returnera en textrepresentation av objektet
- **`Object clone()`**
  - Skapa och returnera en kopia av objektet

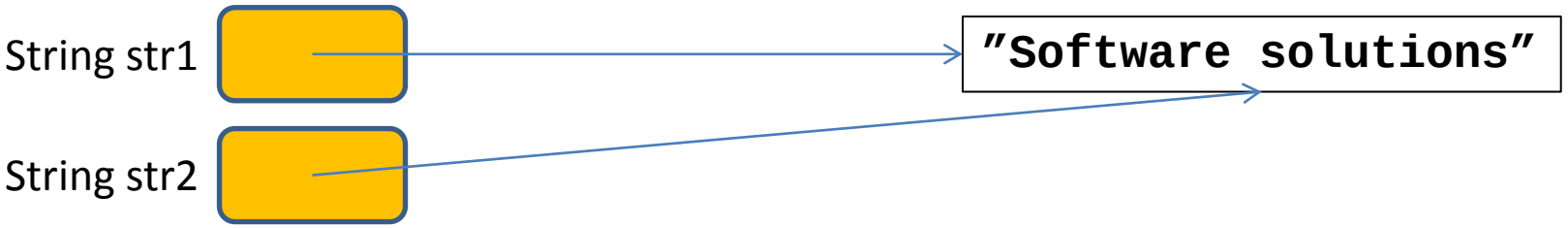
Klassen Object

- Alla klasser ärver från Object bl a:
- **`boolean equals(Object o)`**
  - obj1 == obj2

    - Definitionen av Object.equals() baseras på alias (samma referens)
    - Ofta vill man hellre ha en likhet baserad på objektets tillstånd.
    - Klassen måste då överlagra equals.

equals(Object o)

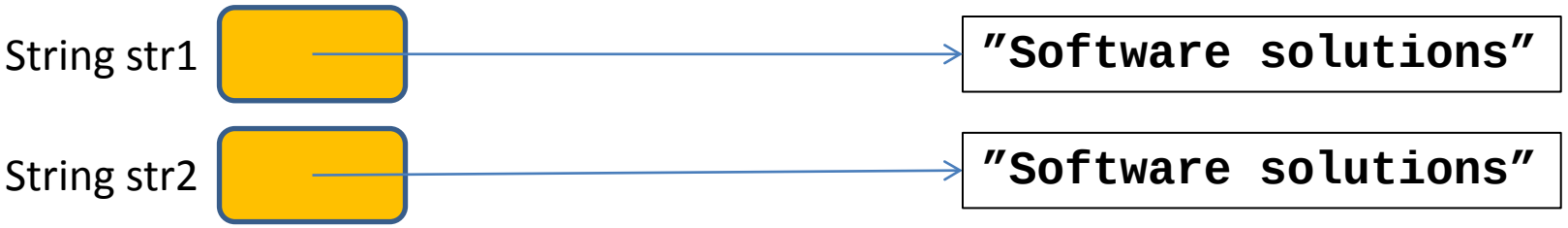
- Likhet råder om det är *samma* objekt eller om objekten *ser lika* ut



str1 och str2 refererar till samma strängobjekt

equals(Object o)

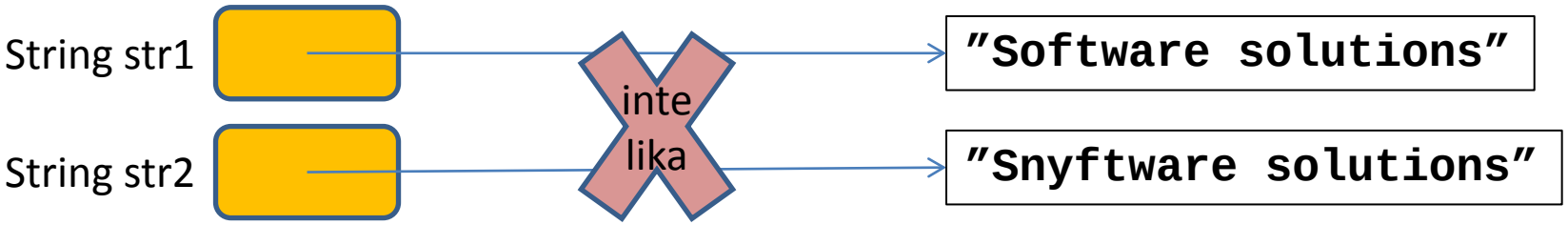
- Likhet råder om det är *samma* objekt eller om objekten *ser lika* ut



str1 och str2 refererar till strängobjekt som *ser lika* ut

equals(Object o)

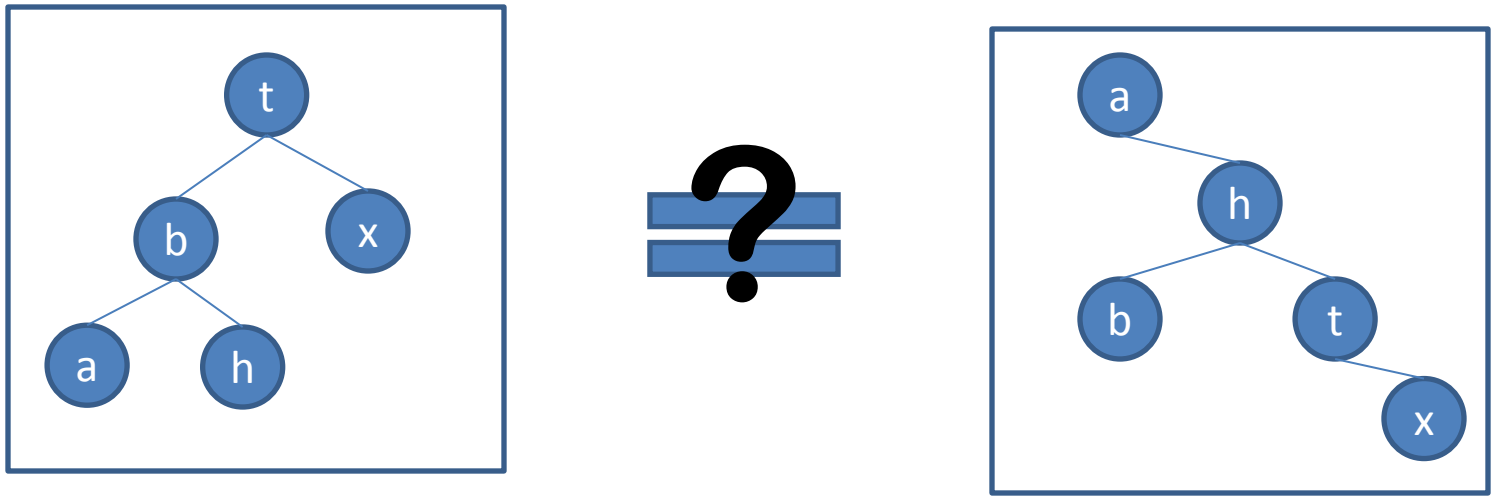
- Likhet råder om det är *samma* objekt eller om objekten *ser lika* ut



str1 och str2 refererar till strängobjekt som *ser olika* ut

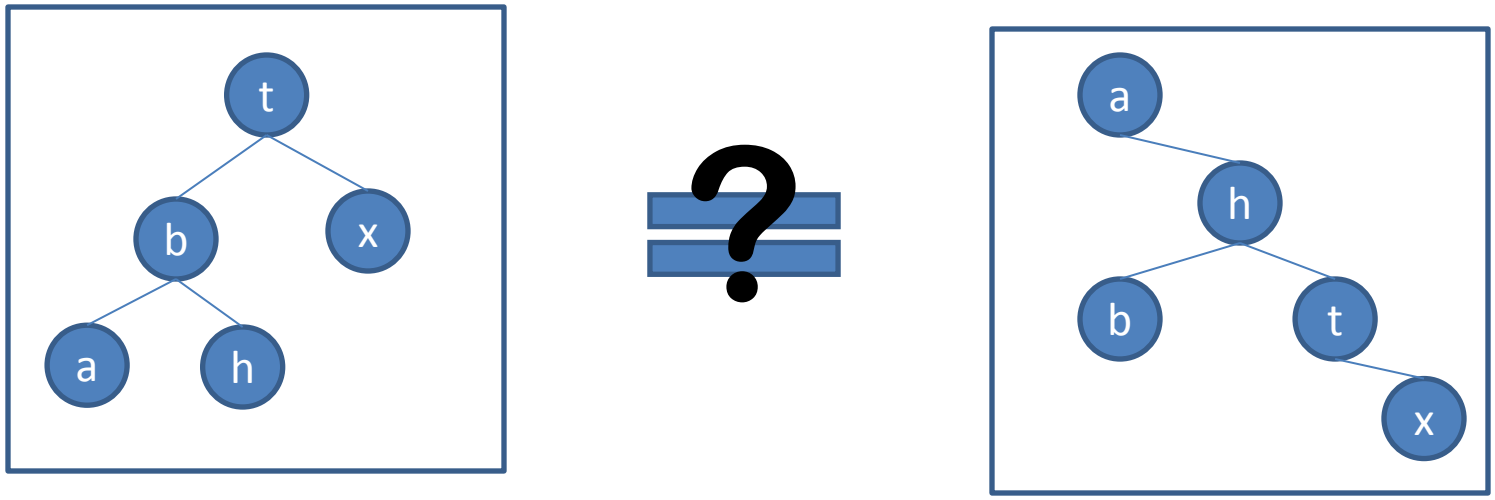
equals(Object o)

- Klassdesignern bestämmer när två objekt ur samma klass är lika
- Att jämföra tillståndet mellan två objekt kan vara kostsamt (ta tid)



Tree.equals(Tree t)?

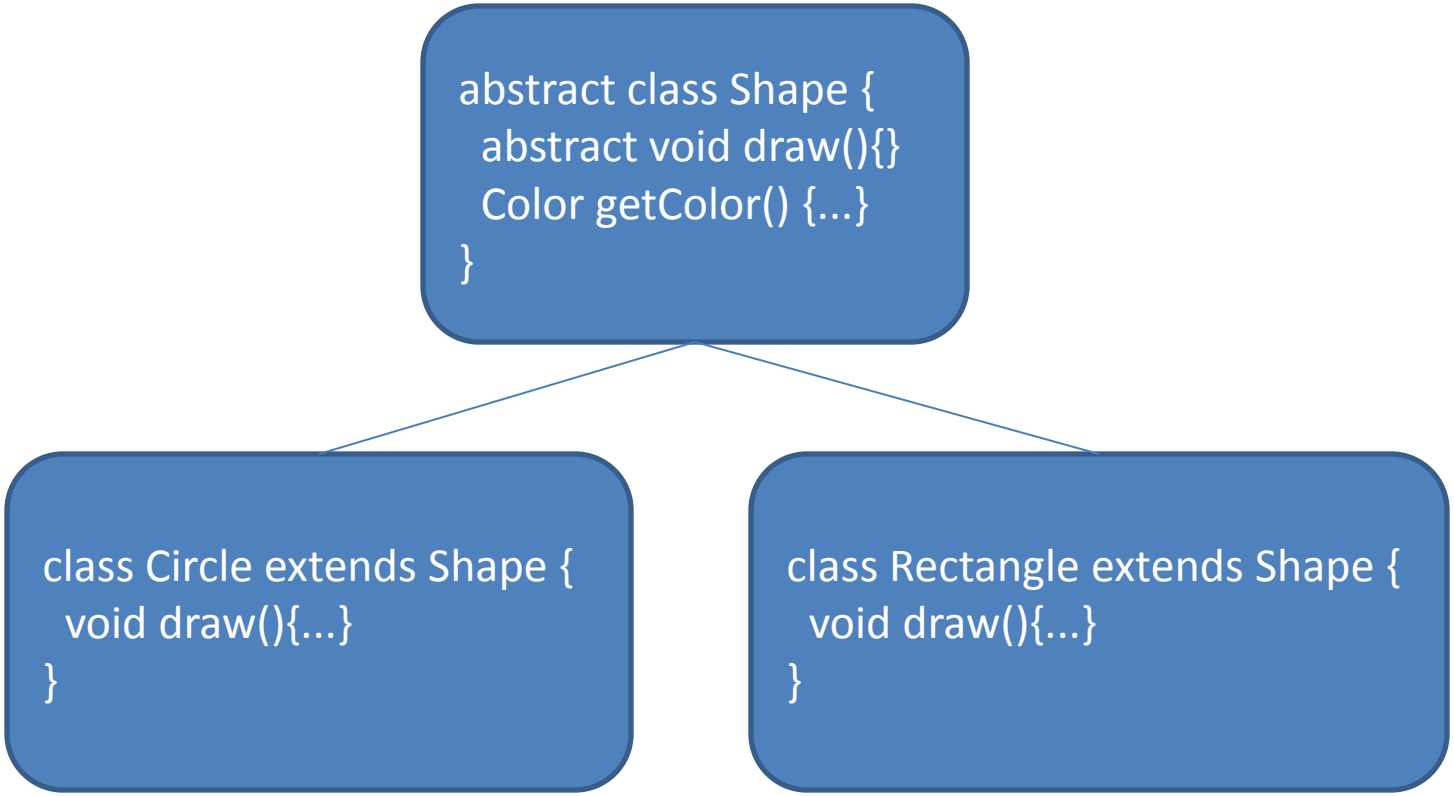
- Nej, träden har olika arrangemang (ordning)
- Ja, träden innehåller samma data (mängd)



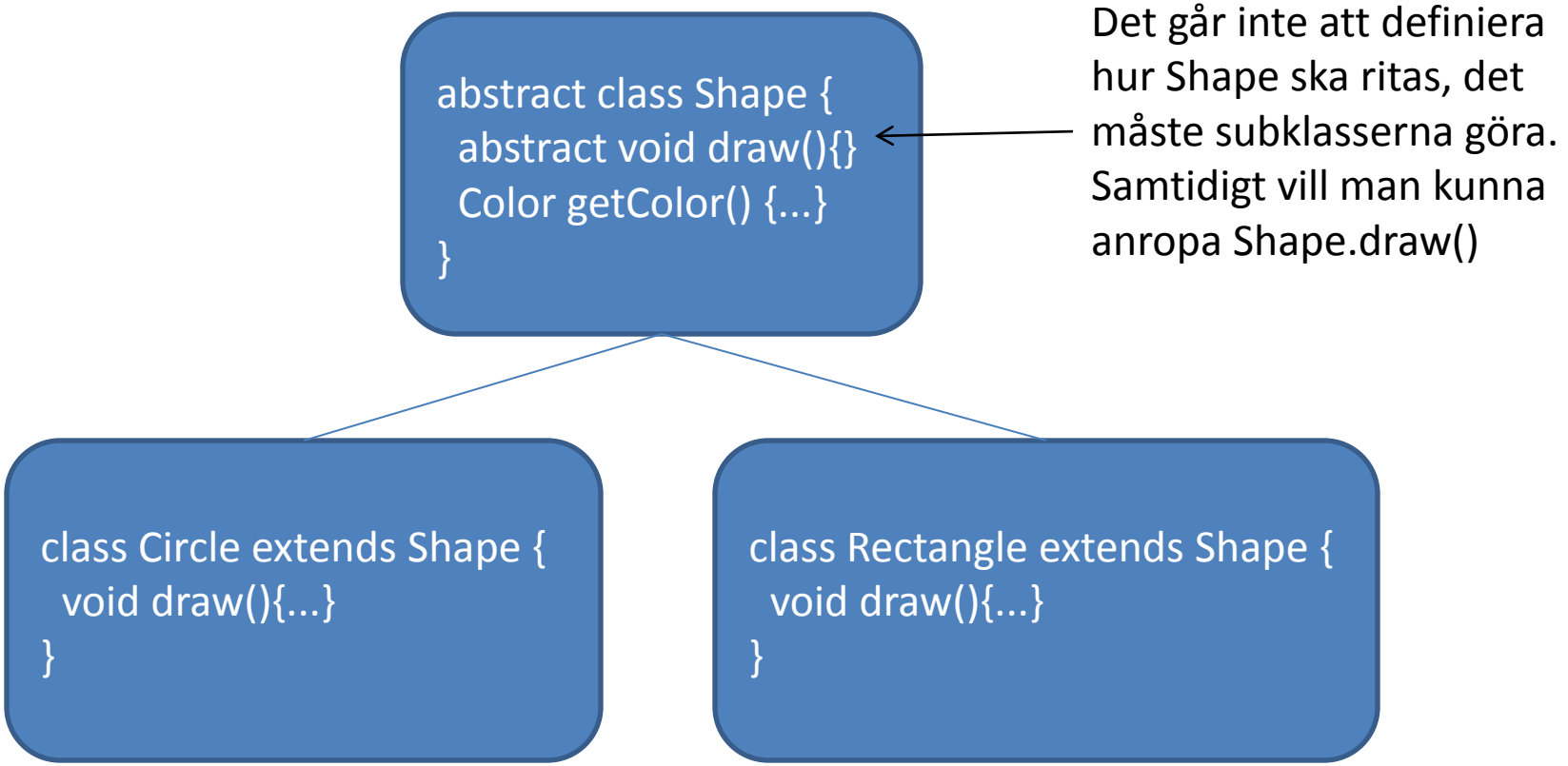
Abstrakta klasser

- En abstrakt klass innehåller en eller fler abstrakta metoder
- En abstrakt klass kan inte instantieras
- Men det går att ärva från den
- En abstrakt klass innehåller en ofullständig beskrivning av en klass
- Subklasser måste komplettera den
- **En abstrakt klass fungerar som *mall* för subklasser**

Abstrakta klasser



Abstrakta klasser

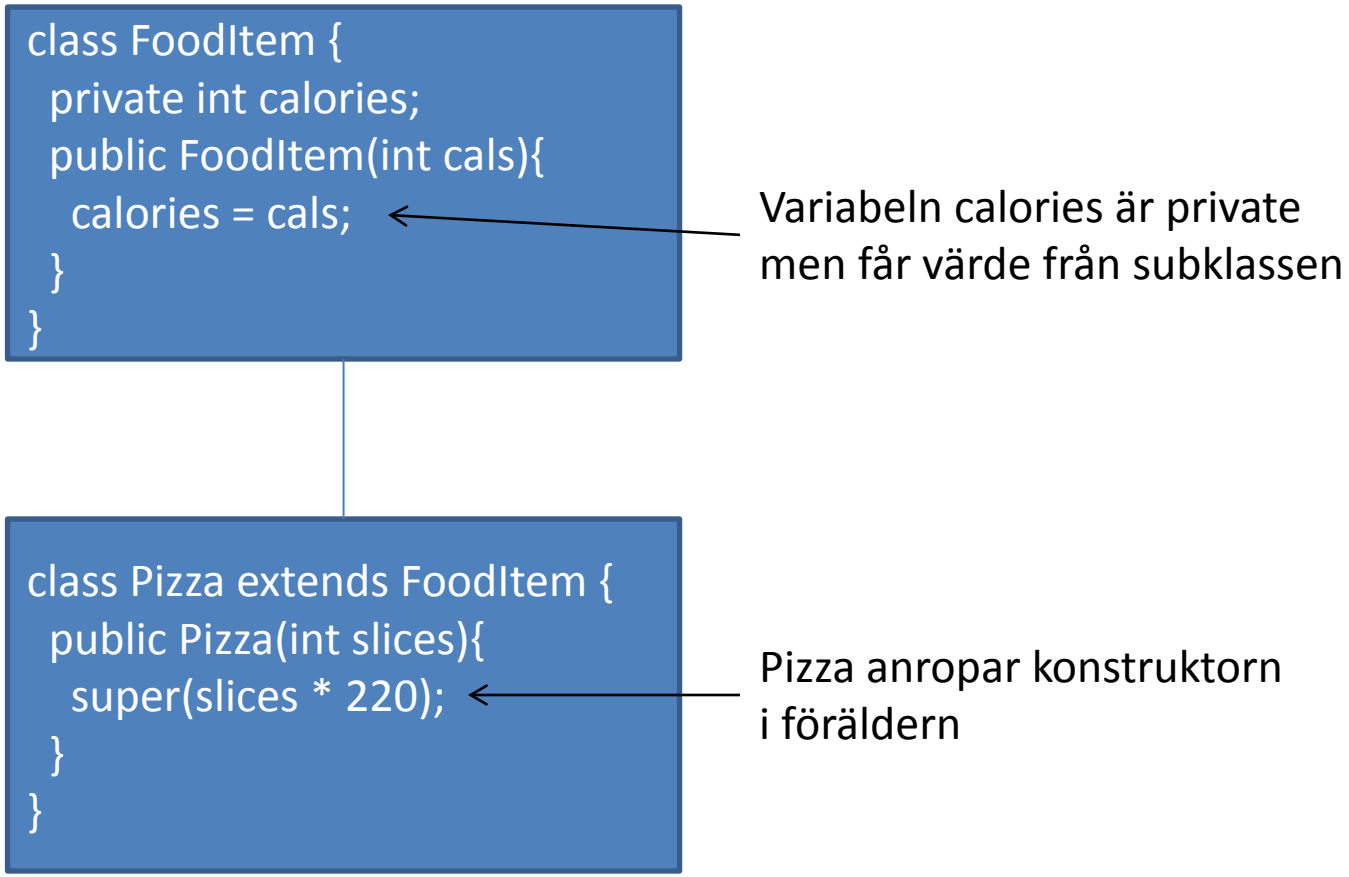


En subklass som ärver från en abstrakt klass måste implementera (åsidosätta) alla abstrakta metoder. Om subklassen inte gör det, blir den själv abstrakt.

Synlighet - private

- En klass kan skydda variabler och metoder genom att deklarera dem private
- Subklasser kan inte ’se’ dem direkt
- Men de finns i varje instans, och ingår i tillståndet
- De nås indirekt genom förälderns tillgängliga metoder (protected eller public)

Synlighet - private



Designa för arv

- En subklass ska vara en mer specifik version av sin förälder
- Fokusera på återanvändning (kod, funktionalitet)
- Placera gemensamma särdrag så högt upp i klasshierarkin som det är praktiskt

Designa för arv

- Åsidosätt metoder för att anpassa beteendet till subklassens särdrag
- Lägg till nya variabler i subklassen om det behövs, men undvik skuggvariabler
- Låt varje klass själv hantera sina interna data

Designa för arv

- Använd **`super`** för att anropa förälderns konstruktor när så behövs
- Använd **`super`** för att anropa förälderns version av en åsidosatt metod
- Använd interface för att skapa en klass med flera beteenden/roller

Designa för arv

- Designa klasshierarkin för programmet men tänk även framåt
- Åsidosätt toString() och equals() så att de ger rimliga resultat
- Abstrakta klasser tydliggör vertikal struktur och delfunktionalitet

Designa för arv

- Använd åtkomstskydd (private/protected) för att skapa vertikal inkapsling
- Använd final för att blockera arv och åsidosättning *när så är motiverat*

```
public final class Standards {
  // This stuff stays this way
}
```

blink blink

BILDSPEL UPPHÖR