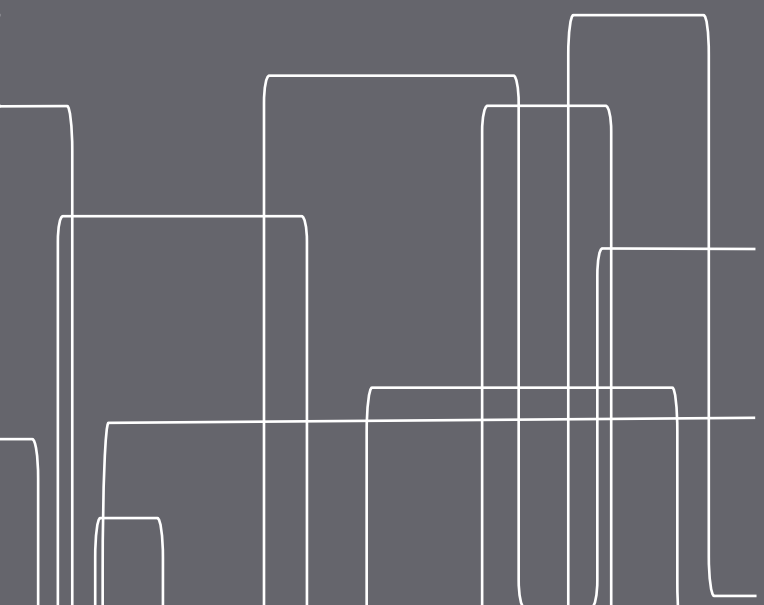
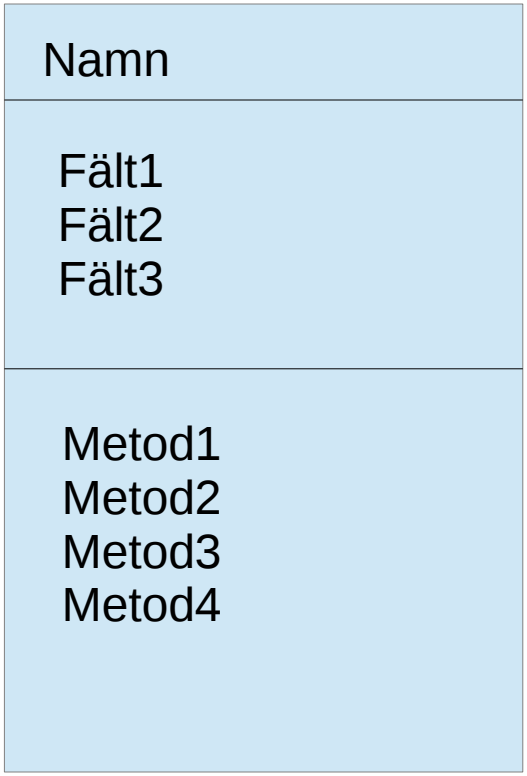
Objektorienterad Programkonstruktion

Föreläsning 2

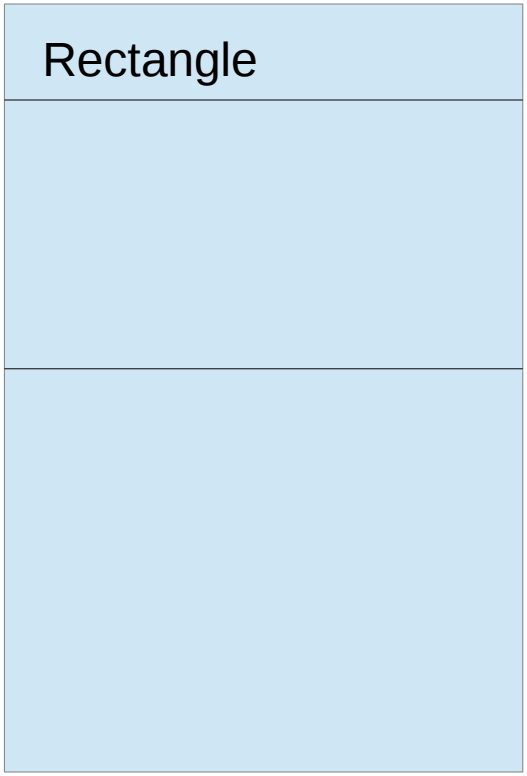
2 nov 2016



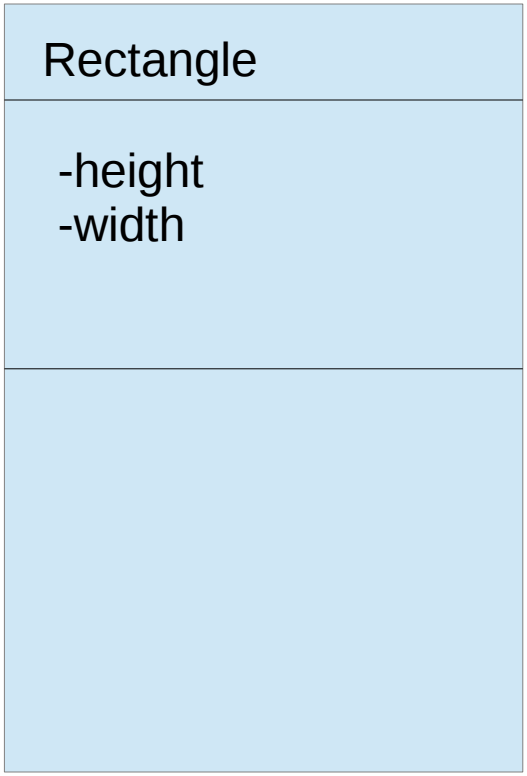
Objekt - klass



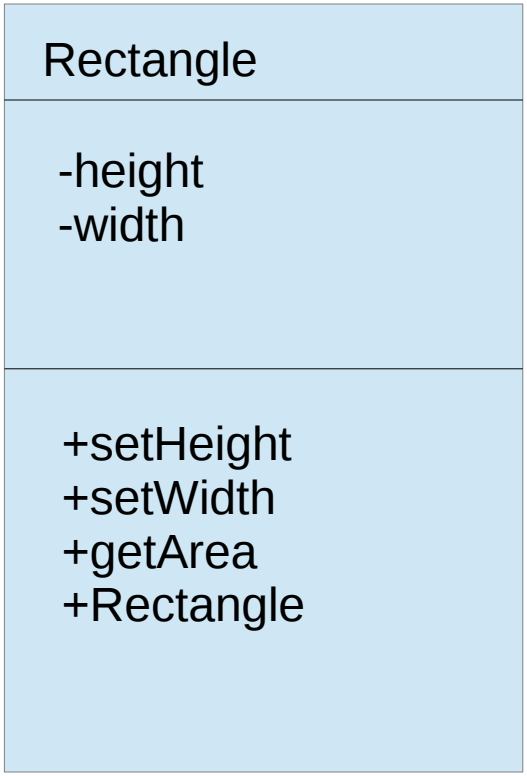
Objekt - klass



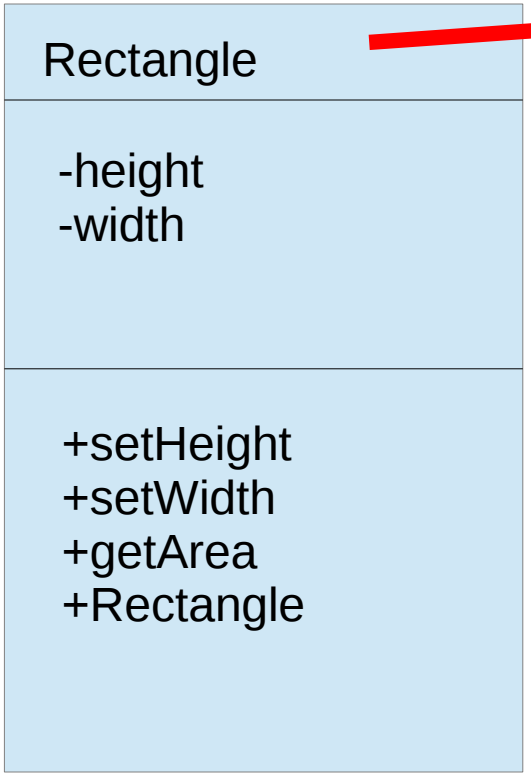
Objekt - klass



Objekt - klass

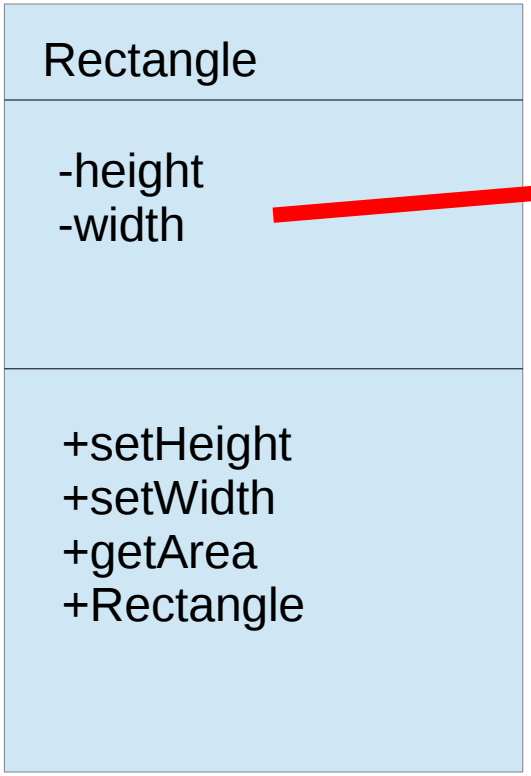


Objekt - klass



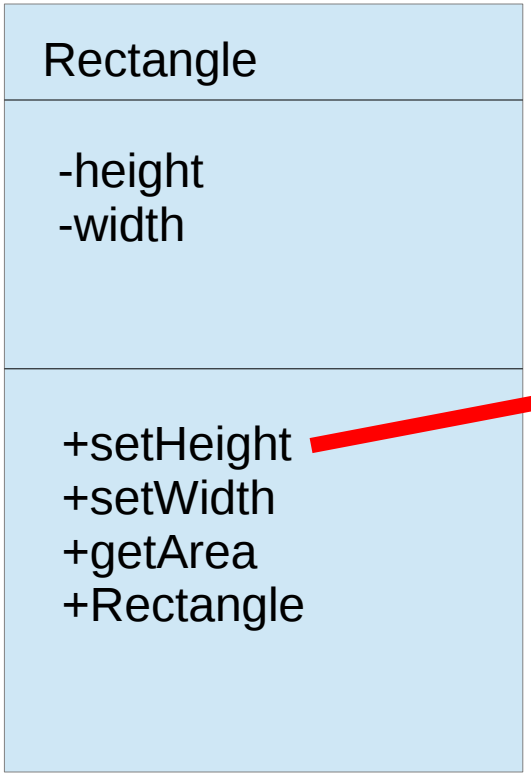

```
public class Rectangle{

}
```

Objekt - klass



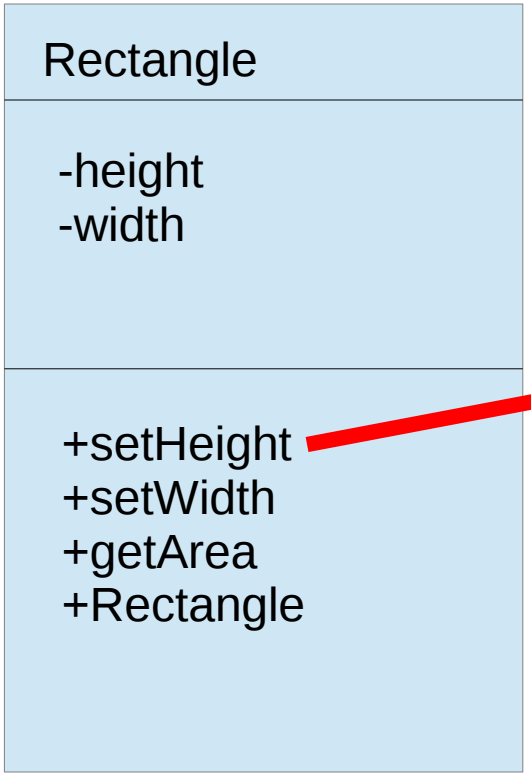

```
public class Rectangle{

    private double height;
    private double width;

}
```

Objekt - klass



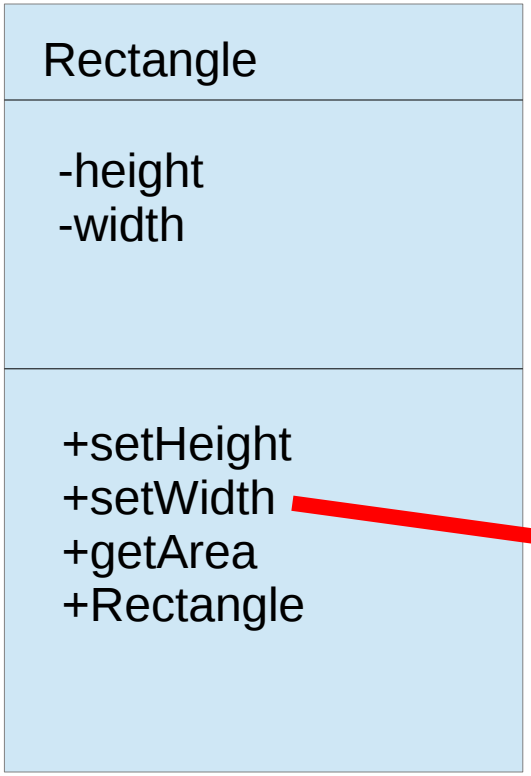

```
public class Rectangle{

    private double height;
    private double width;

    public void setHeight(double h){
        height = h;
    }

}
```

Objekt - klass



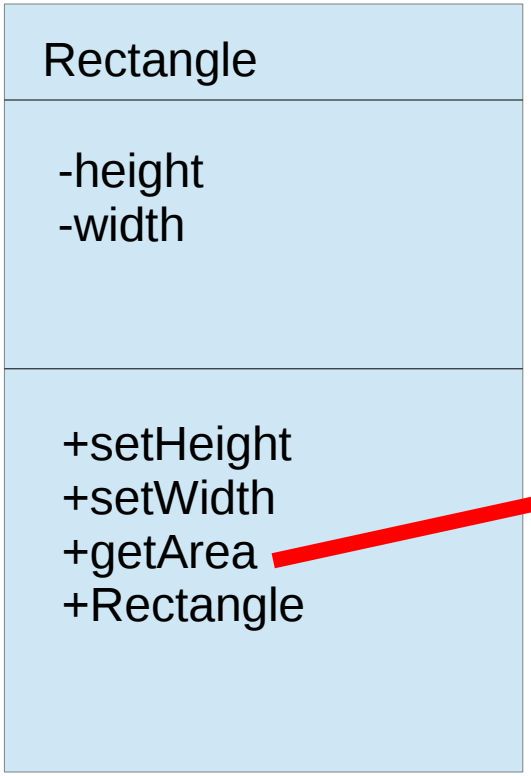

```
public class Rectangle{

    private double height;
    private double width;

    public void setHeight(double h){
        if(h >= 0){
            height = h;
        }else{
            height = 0;
        }
    }

}
```

Objekt - klass



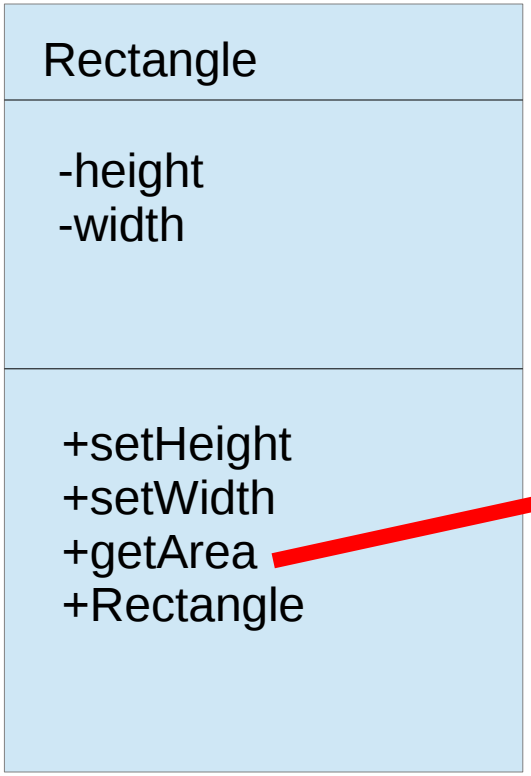

```
public class Rectangle{

    private double height;
    private double width;

    public void setHeight(double h){
        height = h;
    }

    public void setWidth(double w){
        width = w;
    }

}
```

Objekt - klass



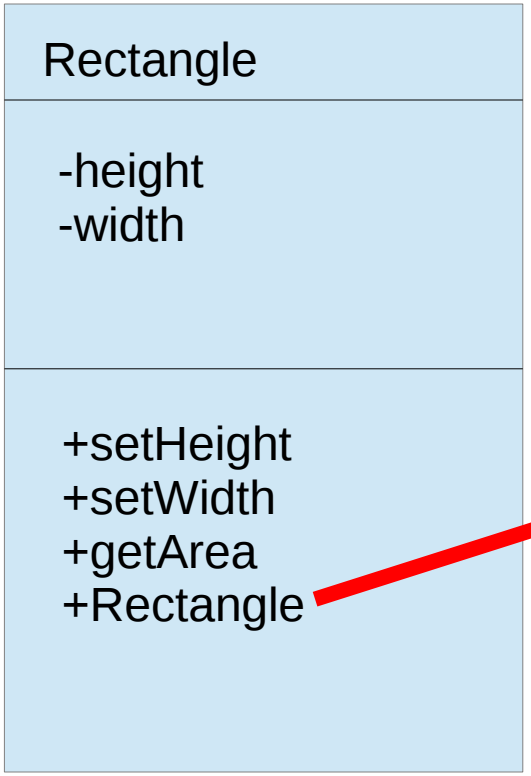

```
public class Rectangle{

    private double height;
    private double width;

    . . .

    public double getArea(){
        return height*width;
    }

}
```

Objekt - klass




```
public class Rectangle{

    private double height;
    private double width;

    . . .

    public double getArea(){
        double a;
        a = height*width;
        return a;
    }

}
```

Objekt - klass




```
public class Rectangle{

    private double height;
    private double width;

    . . .

    public Rectangle(){
        height = 1;
        width = 1;
    }
}
```

Att skapa objekt

Vi kan skapa nya objekt med nyckelordet **new**. Då anropas konstruktorn, och ett nytt objekt returneras.

Ex:

```
Rectangle myRect;          // Deklarerar en variabel
                           // som kan lagra objekt
                           // av typen Rectangle


myRect = new Rectangle(); // Ett nytt objekt av
                          // Rectangle skapas och

                          // och lagras i myRect
```

Terminologi

Vi hänvisar till fält i ett objekt med punktnotation:

Ex: `myRect.height`

Metoder anropas på liknande vis, men tar (eventuellt tomma) argument också:

Ex: `myRect.setHeight(10)`

Ett fält i ett objekt, kan i sig vara ett objekt, då kan vi skriva längre uttryck för att hänvisa till något.

Ex:
```
myRect.color.getBlue();  // ger B-värdet från färgens
                         // RGB-värde
```

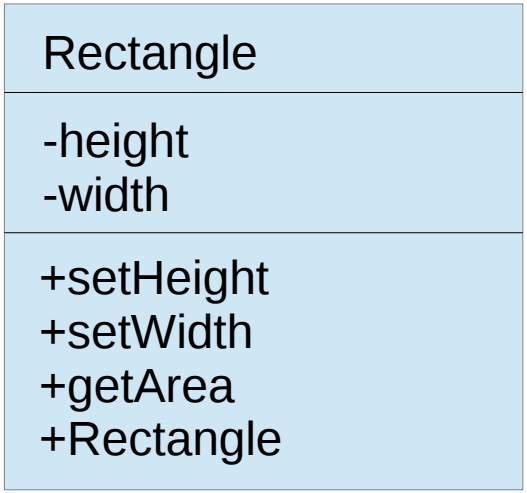
Att referera sig själv

Alla objekt kan explicit referera sig själva med hjälp av nyckelordet `this`. Ett metodanrop av typen `this()` blir ett anrop av den egna konstruktorn. Detta kan användas om man vill skriva en konstruktor som anropar en annan:

```
public class Exempel{
    private int a;
    private double b;

    public Exempel(int a, double b){
       this.a = a;
       this.b = b;
    }

    public Exempel(int a){
        this(a, (double)a);
    }
}
```

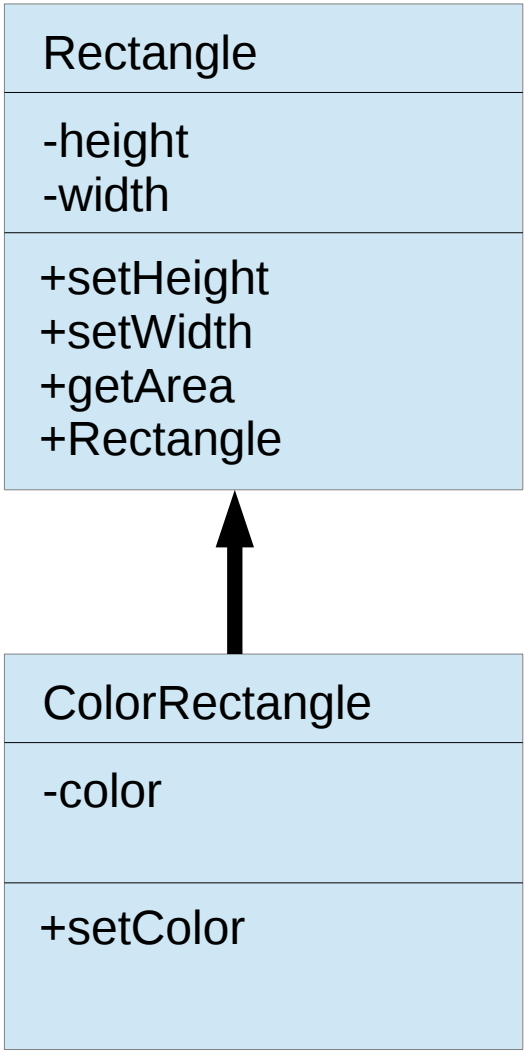

KTH
VETENSKAP
OCH KONST

Arv


Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle


KTH
VETENSKAP
OCH KONST

Arv


Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle
ColorRectangle
-color
+setColor

Arv

En klass **B** som ärver från en annan klass **A** får automatiskt samma fält och metoder som **A**, om man inte deklarerar om dessa i **B**. I **B** kan man lägga till ytterligare fält och metoder, eller definiera om de gamla, sk överskuggning

- Om man i **B** vill använda en metod så som den definierades i **A**, kan man ange detta explicit med hjälp av nyckelordet super
  - Ex: `super.doSomething()`
- I **B**:s konstruktor(er) kommer automatiskt **A**:s argumentlösa konstruktor att anropas om man inte explicit anger något annat.
- Kom ihåg att om **A** har fält som är privata, så kommer **B** inte att kunna anropa dessa direkt!

Arv från Object

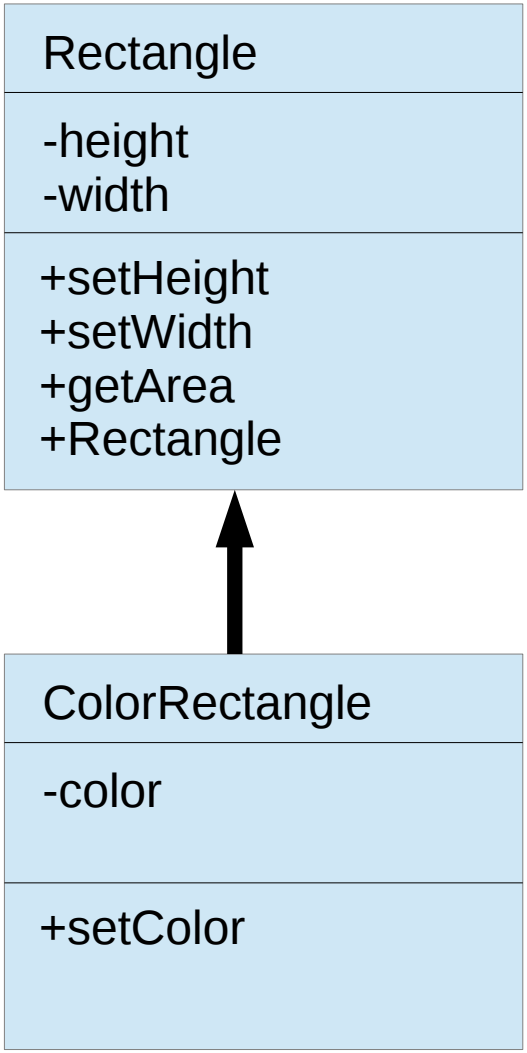
Om inget annat deklareras ärver en klass från superklassen Object

Om en klass explicit sägs ärva från en annan klass kommer denna andra klass att direkt eller indirekt ärva från Object (Det går inte att skapa cirkulära arvskedjor)

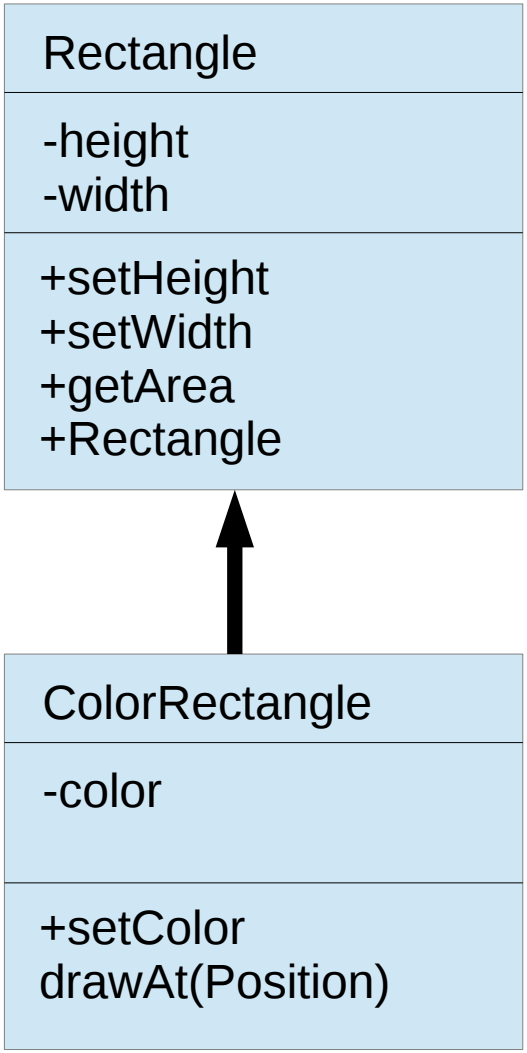
Arv från Object

- `Object clone()` - Creates and returns a copy of this object.
- `boolean equals(Object obj)` - Indicates whether some other object is "equal to" this one.
- `String toString()` - Returns a string representation of the object.
- `protected void finalize()` - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- `Class<?> getClass()` - Returns the runtime class of this Object.
- `int hashCode()` - Returns a hash code value for the object.
- `void notify()` - Wakes up a single thread that is waiting on this object's monitor.
- `void notifyAll()` - Wakes up all threads that are waiting on this object's monitor.
- `void wait()` - Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` - method for this object.
- `void wait(long timeout)` - Causes the current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
- `void wait(long timeout, int nanos)` - Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

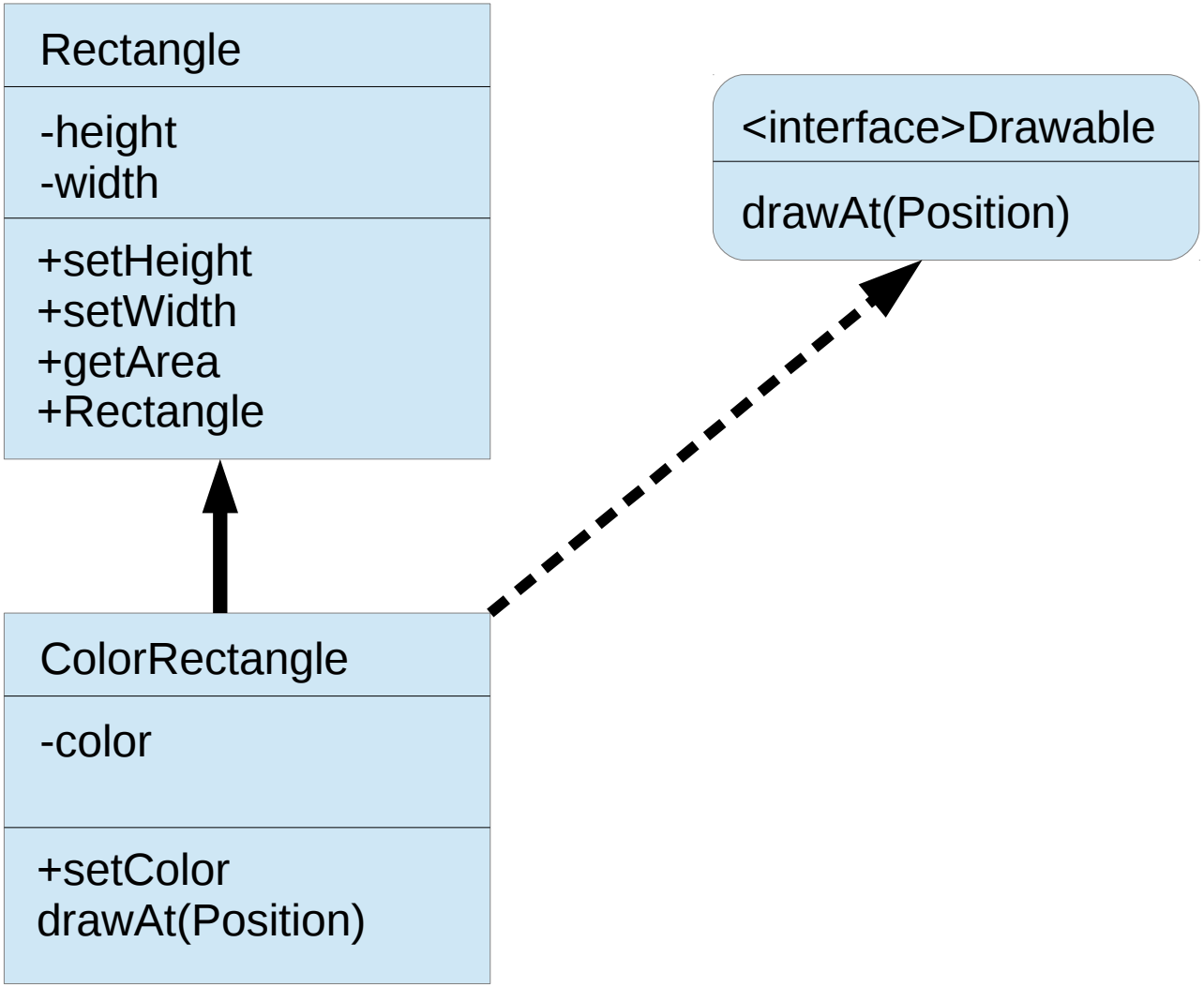
Gränssnitt


Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle
ColorRectangle
-color
+setColor

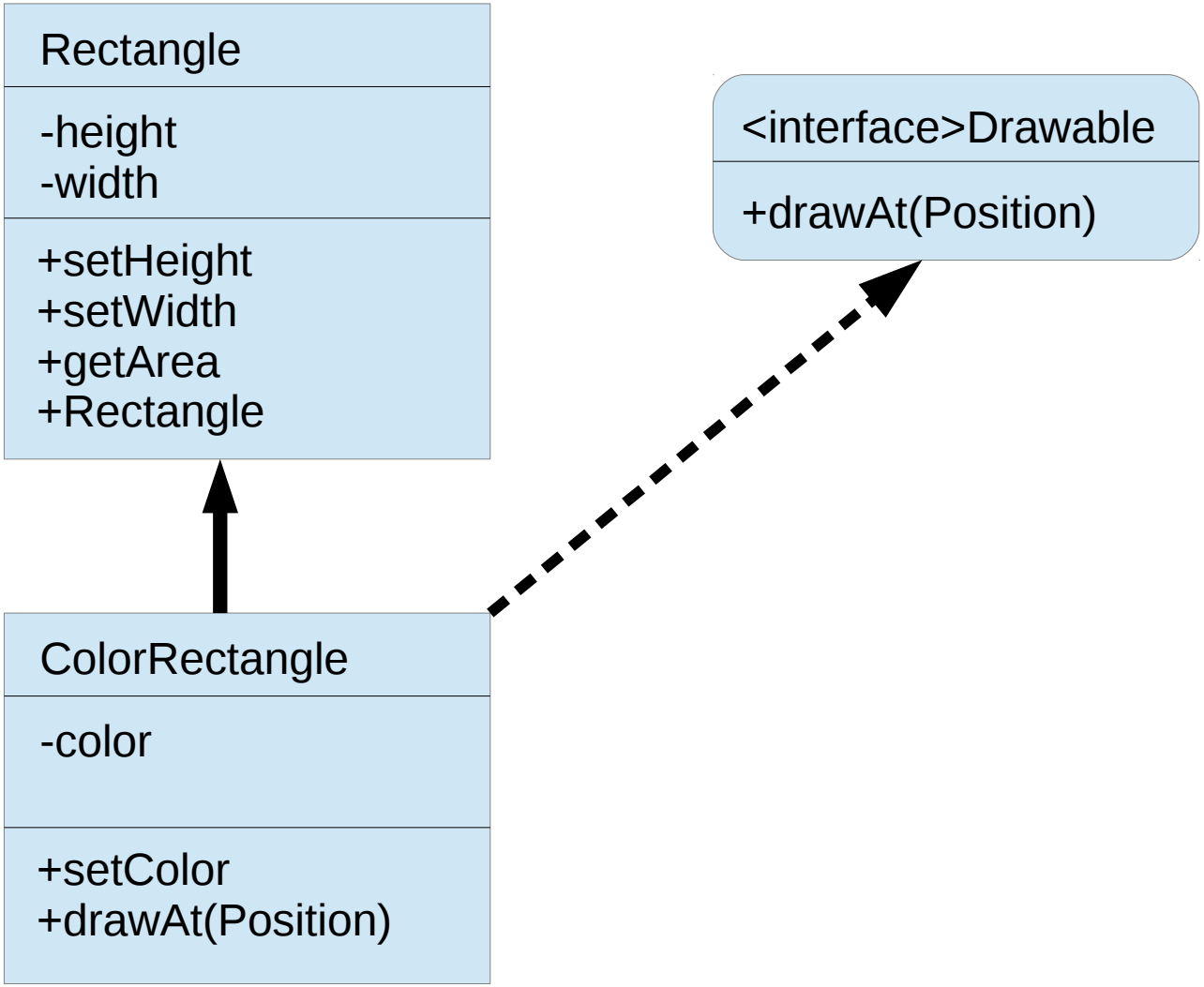
Gränssnitt


Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle
ColorRectangle
-color
+setColor
drawAt(Position)

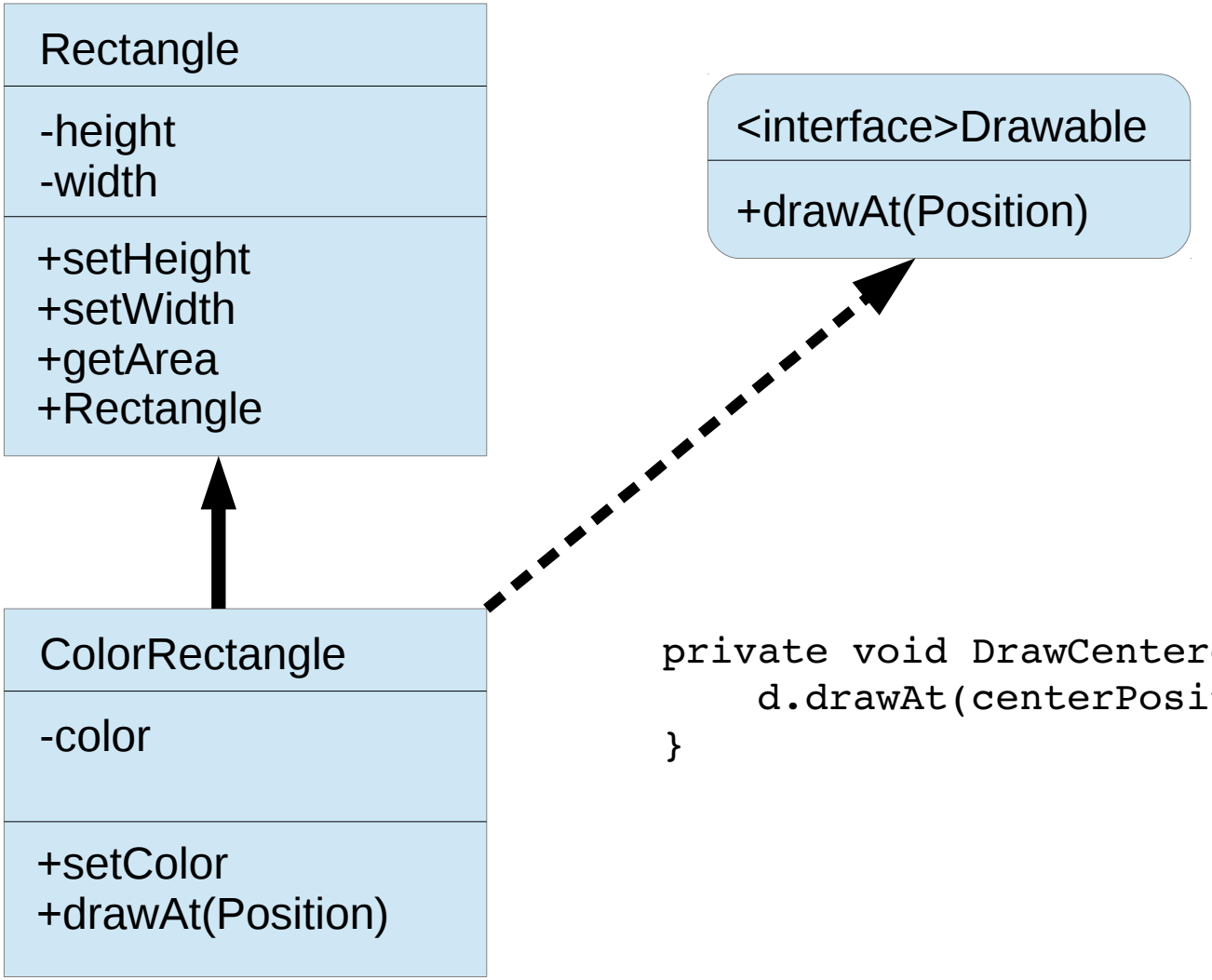
Gränssnitt


Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle
<interface>Drawable
drawAt(Position)
ColorRectangle
-color
+setColor
drawAt(Position)

Gränssnitt

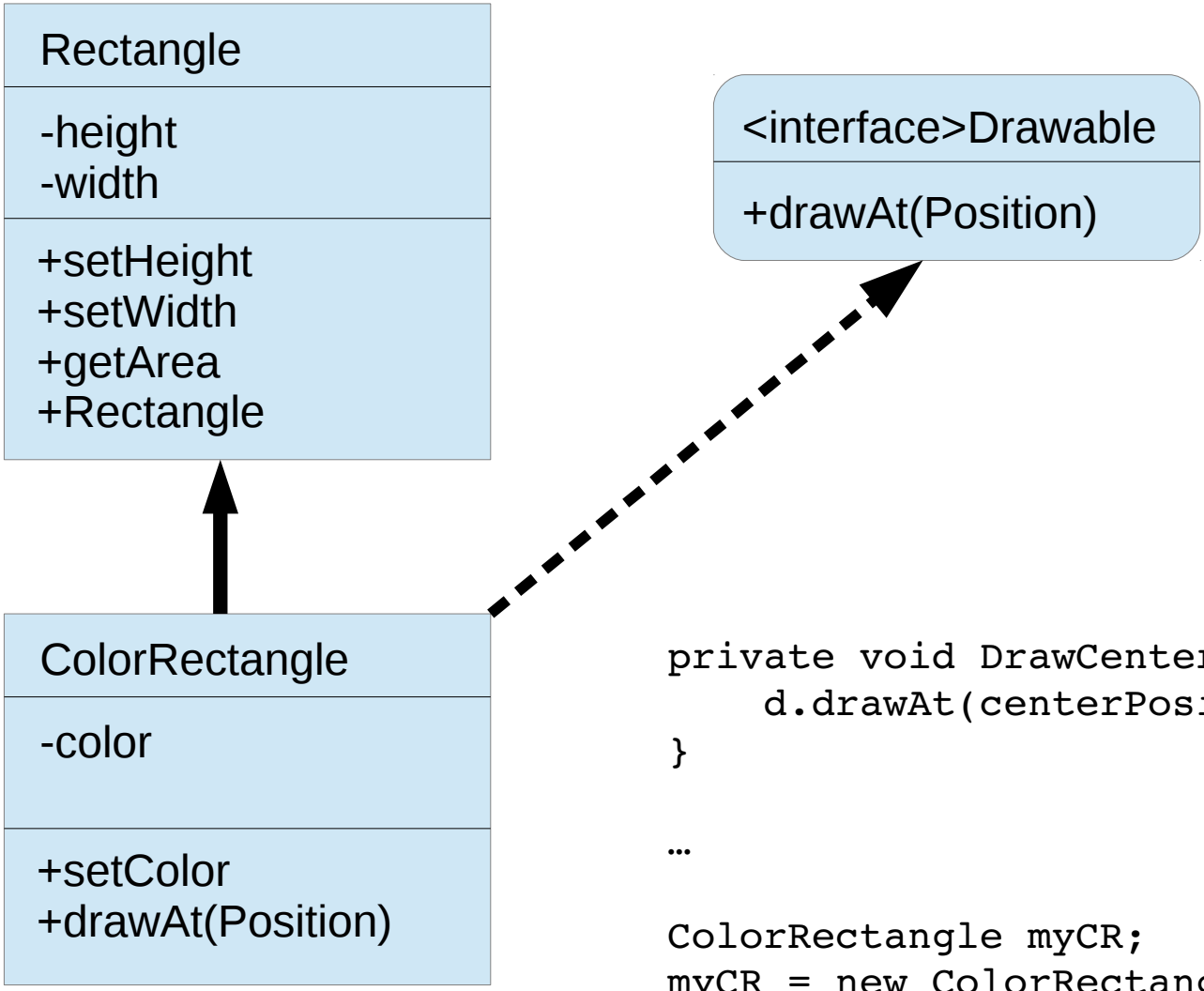

Rectangle
-height
-width
+setHeight
+setWidth
+getArea
+Rectangle
<interface>Drawable
+drawAt(Position)
ColorRectangle
-color
+setColor
+drawAt(Position)

Gränssnitt




```
private void DrawCentered(Drawable d){
    d.drawAt(centerPosition);
}
```

Gränssnitt




```
private void DrawCentered(Drawable d){
    d.drawAt(centerPosition);
}

…

ColorRectangle myCR;
myCR = new ColorRectangle();
DrawCentered(myCR);
```

Gränsnitt (Interface)

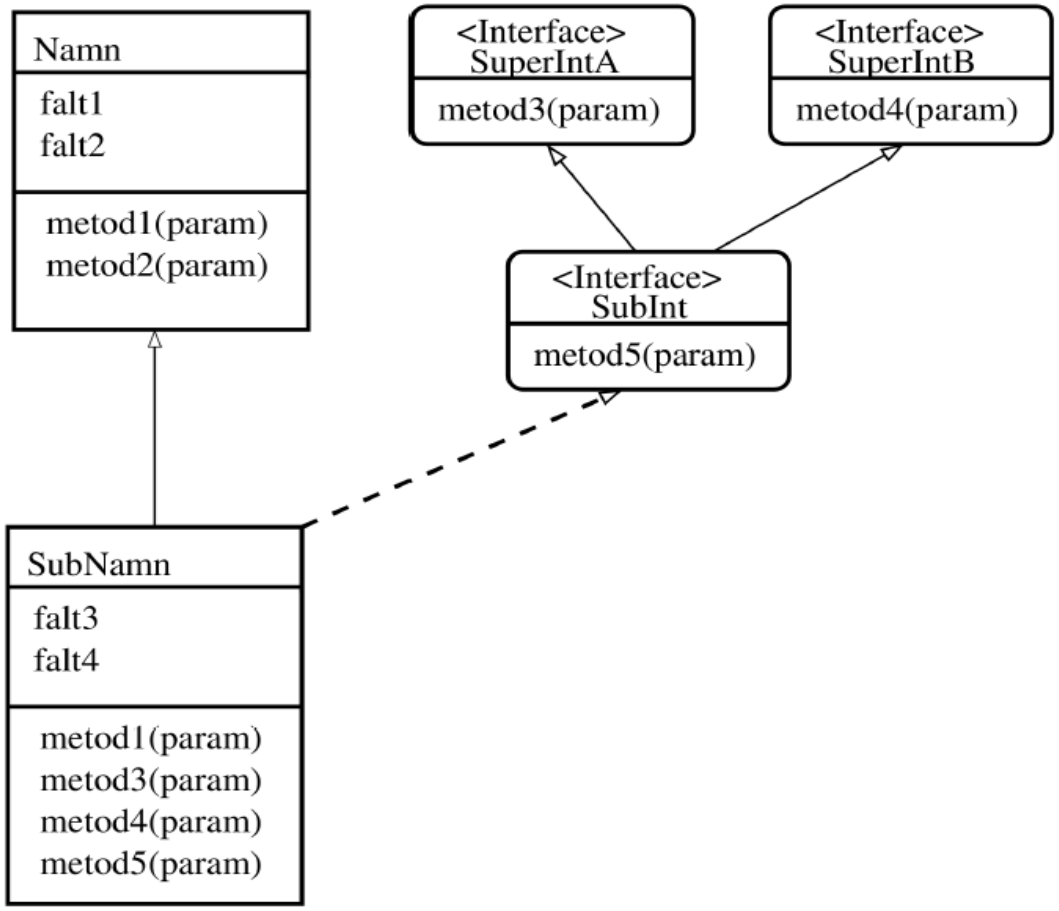
Gränssnitt kan liknas vid en superklass som inte innehåller några konkreta metoder eller instansvariabler

- Gränsnitt kan ärva från flera andra gränsnitt
- Alla metoder i ett gränsnitt är automatiskt publika

exempel (från Java Tutorial):

```
public interface GroupedIntfce extends Intfce1,Intfce2,Intfce3 {
    // constant declarations
    public static final double E = 2.718282; // base of natural logarithms

    // method signatures
    public void doSomething (int i, double x);

    public int doSomethingElse(String s);
}
```

Exempel: Gränssnitt och Arv



Klass eller instans?

Med hjälp av nyckelordet static kan vi bestämma att en metod eller ett fält ska tillhöra själva klassen i stället för en specifik instans av klassen (ett objekt)

- Instansmetoder kan komma åt instansvariabler och -metoder direkt
- Instansmetoder kan komma åt klassvariabler och -metoder direkt
- Klassmetoder kan komma åt klassvariabler och -metoder direkt

men

- Klassmetoder kan inte komma åt instansvariabler och -metoder direkt

Klass/instans, exempel

```
public class Exempel{

    private int a = 5;
    static int b = 3;

    public int getA(){
        return a;
    }

    public int getB(){
        return b;
    }

    public static int getStatB(){
        return b;
    }
}
```

```
public class FelaktigtExempel{

    // OBS! denna kod är
    // felaktig !!
    private int a = 5;
    static int b = 3;

    public static int getA(){
        return a;
    }
}
```

non-static variable a cannot be referenced from a static context

När bör man använda klassmetoder/-fält?

- Egenskaper som är gemensamma för alla objekt
  - ex: räknare för antalet instanser
- Konstanter (kan deklareras static final för att förhindra att de ändras av misstag)
  - ex: `Math.PI, Math.E`
- Metoder som utför operationer med primitiva variabler
  - ex: `Math.cos(double a), Math.round(double a)`
- Metoder som fungerar oberoende av om objekt av en viss klass har skapats
  - ex: `Map.convertKmToMiles(double km)`