Westerdals

Oslo School of Arts,
Communication and Technology

PGR 101 Objektorientert Programmering 2
Vår 2017

Forelesning 7.3.17

(Stein Marthinsen – marste@westerdals.no)

Dagens tema

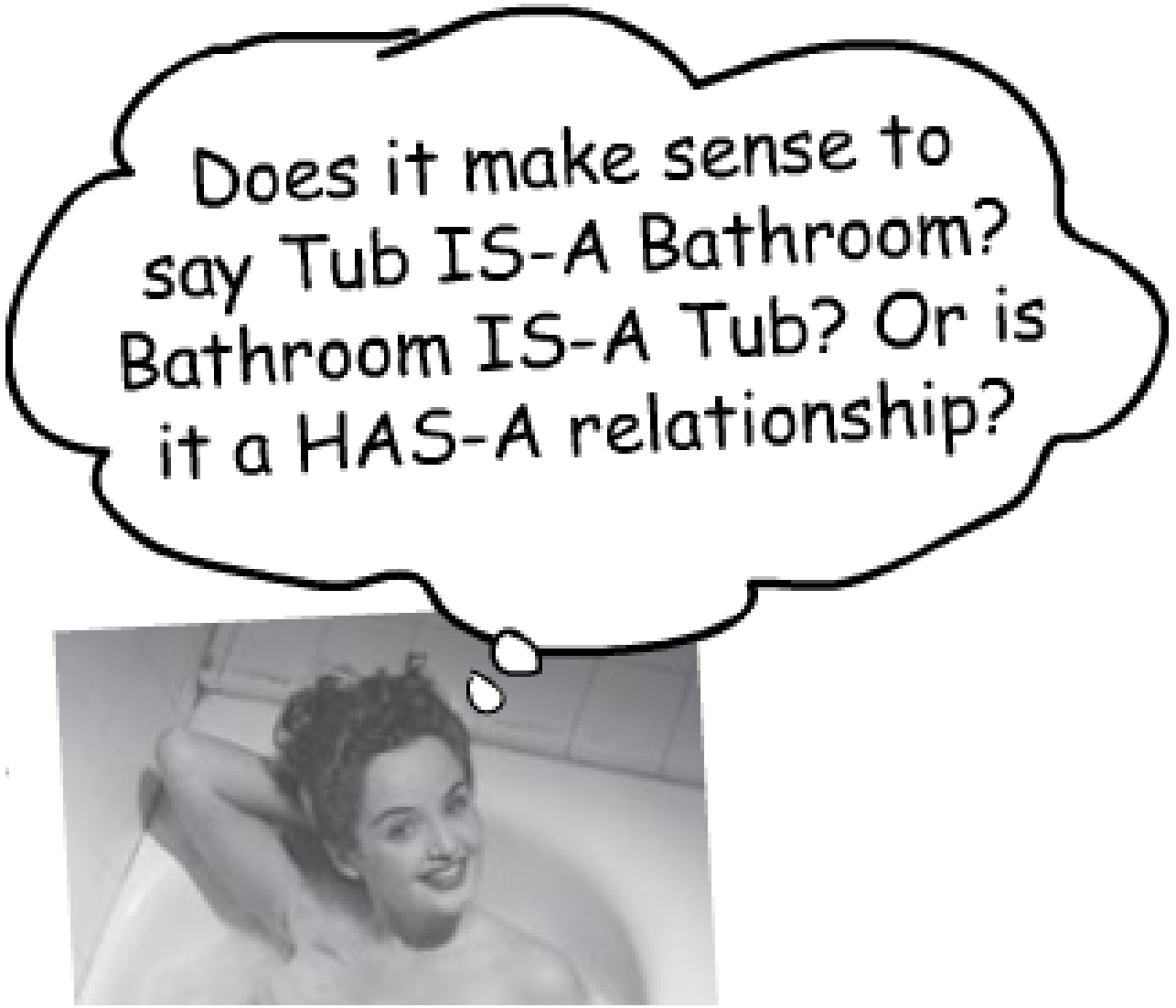
Eksamensoppgave

Klassen `Object`

- `equals`
- `toString`

Mer ARV

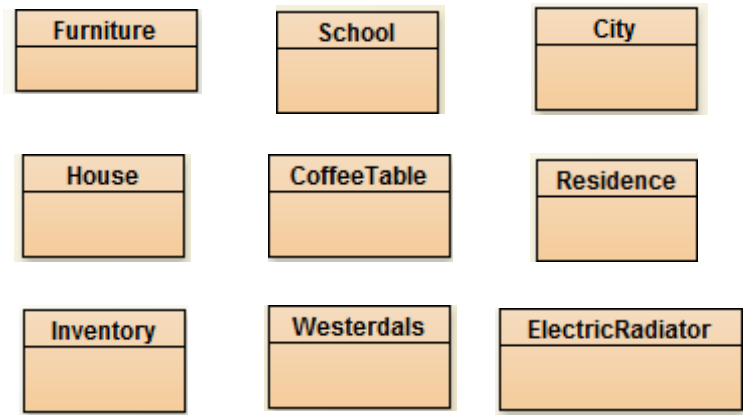
- `abstract`



Eksamensoppgave

Tenk deg følgende klasser:

`School`	(Studiested)
`CoffeeTable`	(Stuebord)
`Furniture`	(Møbel)
`House`	(Hus)
`Inventory`	(Inventar)
`City`	(By)
`Westerdals`	
`Residence`	(Bosted)
`ElectricRadiator`	(Panelovn)

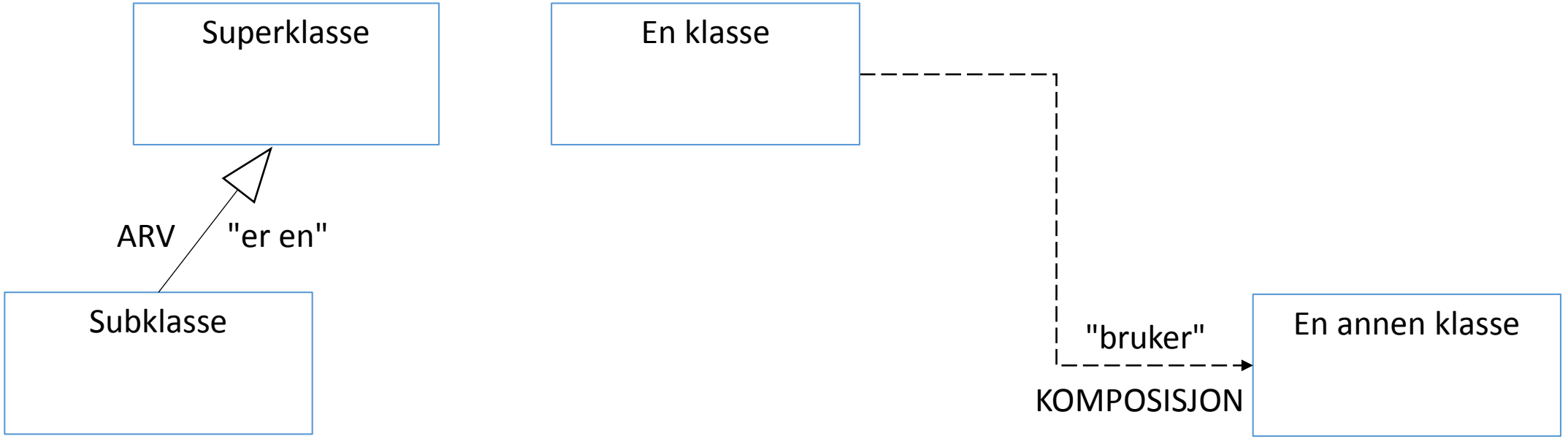


Hvilke klasser er det naturlig å knytte sammen ved *arv*?
Hvorfor er dette naturlig?
Hvordan kan klassene ellers knyttes sammen på en naturlig måte?

Eksamensoppgave

Tegn en figur som viser sammenhenger!

Bruk følgende notasjon:



Eksamensoppgave

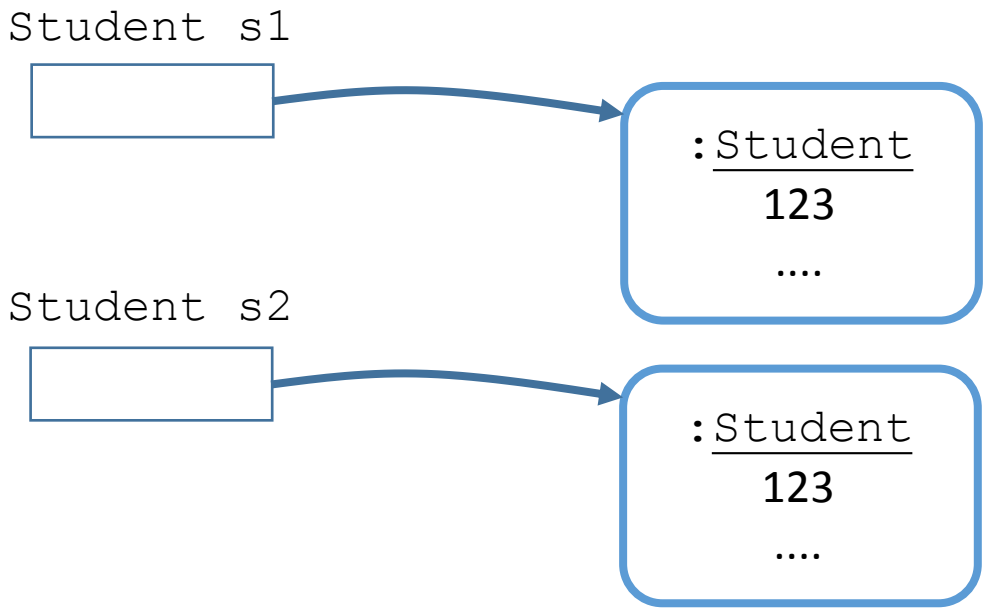
Et løsningsforslag (møt opp på forelesningen med et eget forslag til løsning!):

Sammenligning av objekter

```
Student s1 =new Student("123", "987", "Albin", "Albinsen", 20);
Student s2 =new Student("123", "987", "Albin", "Albinsen", 20);
```

```
if (s1 == s2) System.out.println("Like!");
else System.out.println("Ulike!");
```

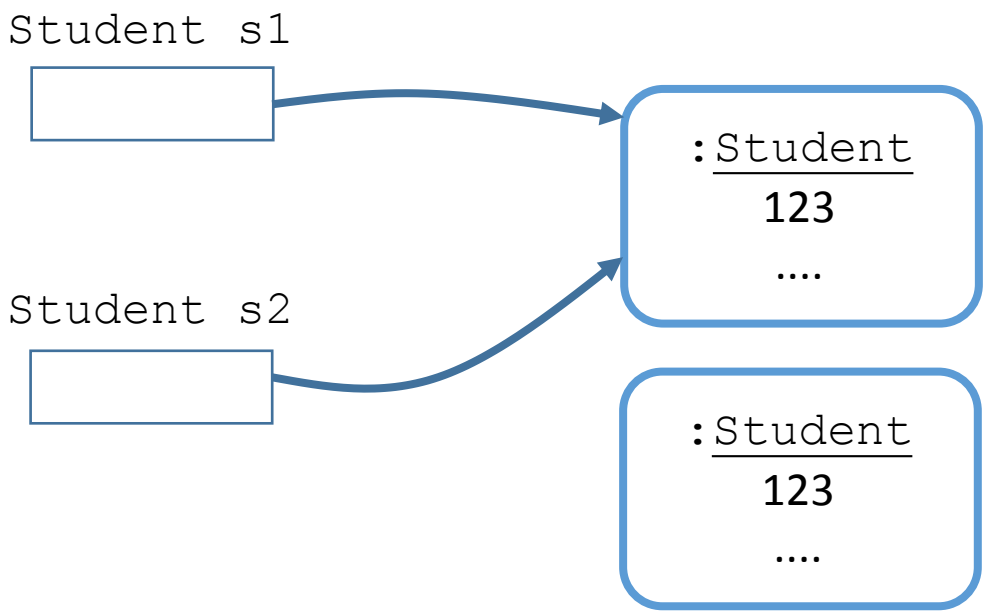
```
Ulike!
```



Sammenligning av objekter

```
s2 = s1;
if (s1 == s2) System.out.println("Like!");
else System.out.println("Ulike!");
```

```
Like!
```



Sammenligning av objekter

Operatoren == virker ikke (som forventet) med objekter.

== sammenligner *objekt-referansene* (adressene hvor i minnet de ligger), ikke hva objektene *inneholder* (deres tilstand).

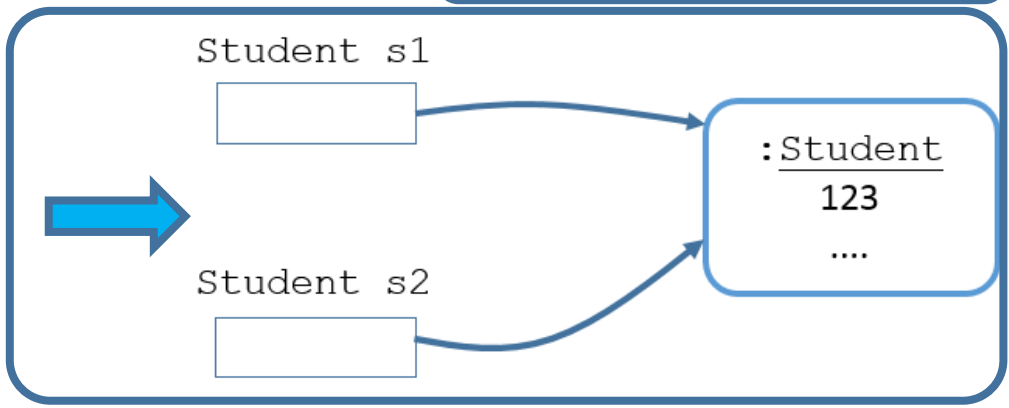
Det betyr at metoden bare returnerer true

når en objekt-referanse sammenlignes med "**seg selv**"

if (s1 == s1)

eller

når to referanser **refererer samme objekt**.



Sammenligning av objekter

Må bruke metoden `equals` for å sammenligne objekter!

Klassen `Object`

Alle klasser har en superklasse kalt **`Object`**.

`Object`–klassen definerer flere metoder:

`public String toString()`
Skal returnere en tekst-representasjon av objektet (tilstanden), slik at den f.eks. kan skrives ut.

`public boolean equals(Object other)`
Skal sammenligne objektet med et annet, for likhet.
Returnerer **`true`** hvis objektene har samme *innhold* (tilstand).

Klassen `Object`

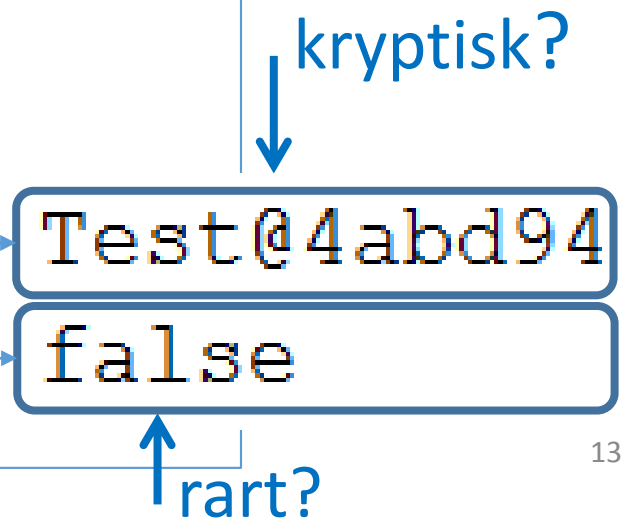
Merk:
Hvis din klasse *ikke* definerer en **`toString`** eller **`equals`**, vil den arvede versjonen bli brukt.

En klasse `Test`

```
public class Test {
   private String attributt;
   public Test() { }
   public Test (String s) {
       attributt = s;
   }
}
```

I en mainMethod:

```
Test t1 = new Test("Hallo");
System.out.println(t1.toString());
Test t2 = new Test("Hallo");
System.out.println(t1.equals(t2));
```



Metoden `equals`

Altså: når du lager en klasse, så vil dens (arvede) **`equals`** metode oppføre seg som

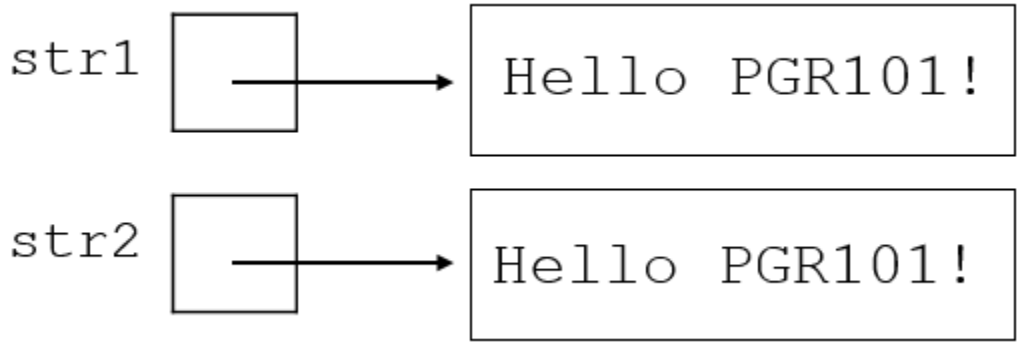
`==`

Metoden `equals`

En tilpasset **`equals`** *må* lages for å kunne sammenligne objekters tilstand (innholdet).

Klassen **`String`** for eksempel, har en som er tilpasset.

```
String str1 = new String("Hello PGR101!");
String str2 = new String("Hello PGR101!");
```




```
System.out.println(str1.equals(str2));    // true
System.out.println(str1 == str2);         // false
```

Klassen `Test` – standard oppsett av `equals`

```
public class Test {
    private String attributt;
    public Test() { }
    public Test (String s) {
        attributt = s;
    }

    public boolean equals(Object obj) {
        if (!(obj instanceof Test)) return false;
        if (obj == this) return true;
        Test test = (Test) obj;
        return test.attributt.equals(attributt);
    }
}
```

signaturen

feil type

seg selv

cast'er til riktig type

merk: direkte tilgang!

`String` sin equals

Klassen `Test` – `toString`

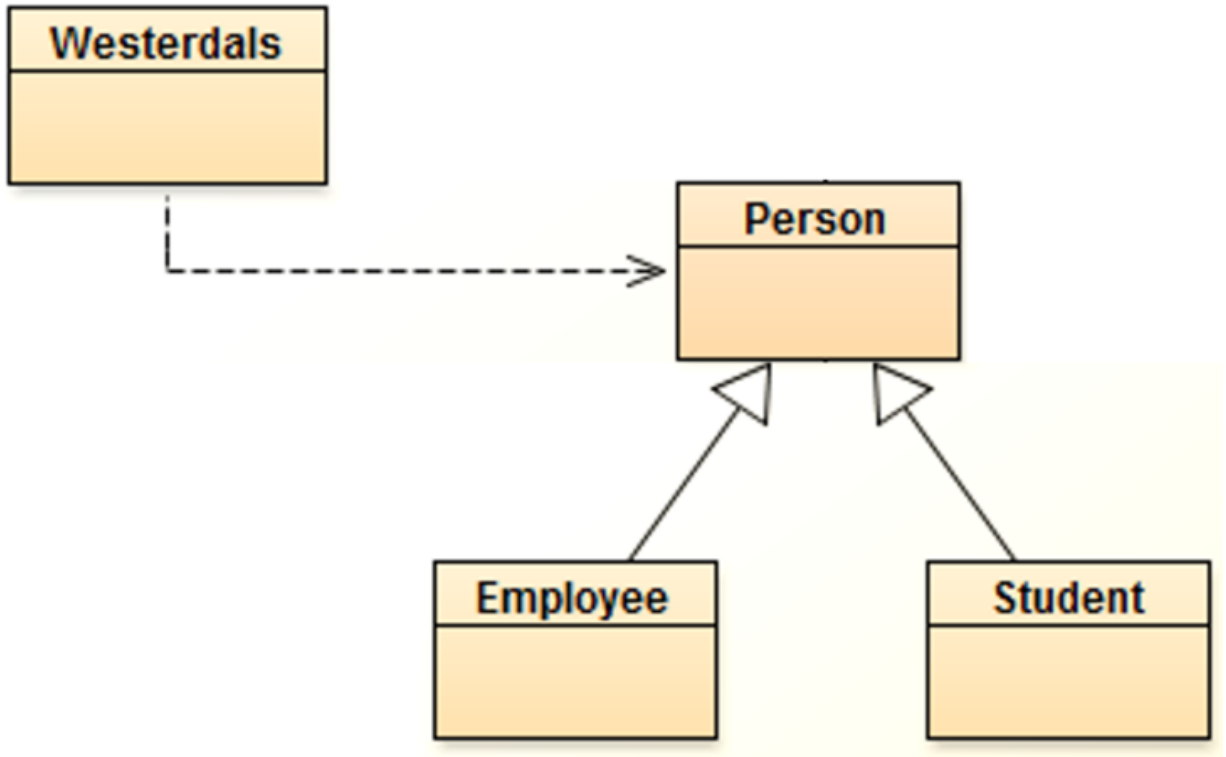
```
public String toString() {
    return "Tilstanden: " + attributt;
}
```

Objektets tilstand

Disse to metodene i klassen **`Test`**, vil nå *skjule* versjonene arvet fra klassen **`Object`**!

Oppgave på øvingen

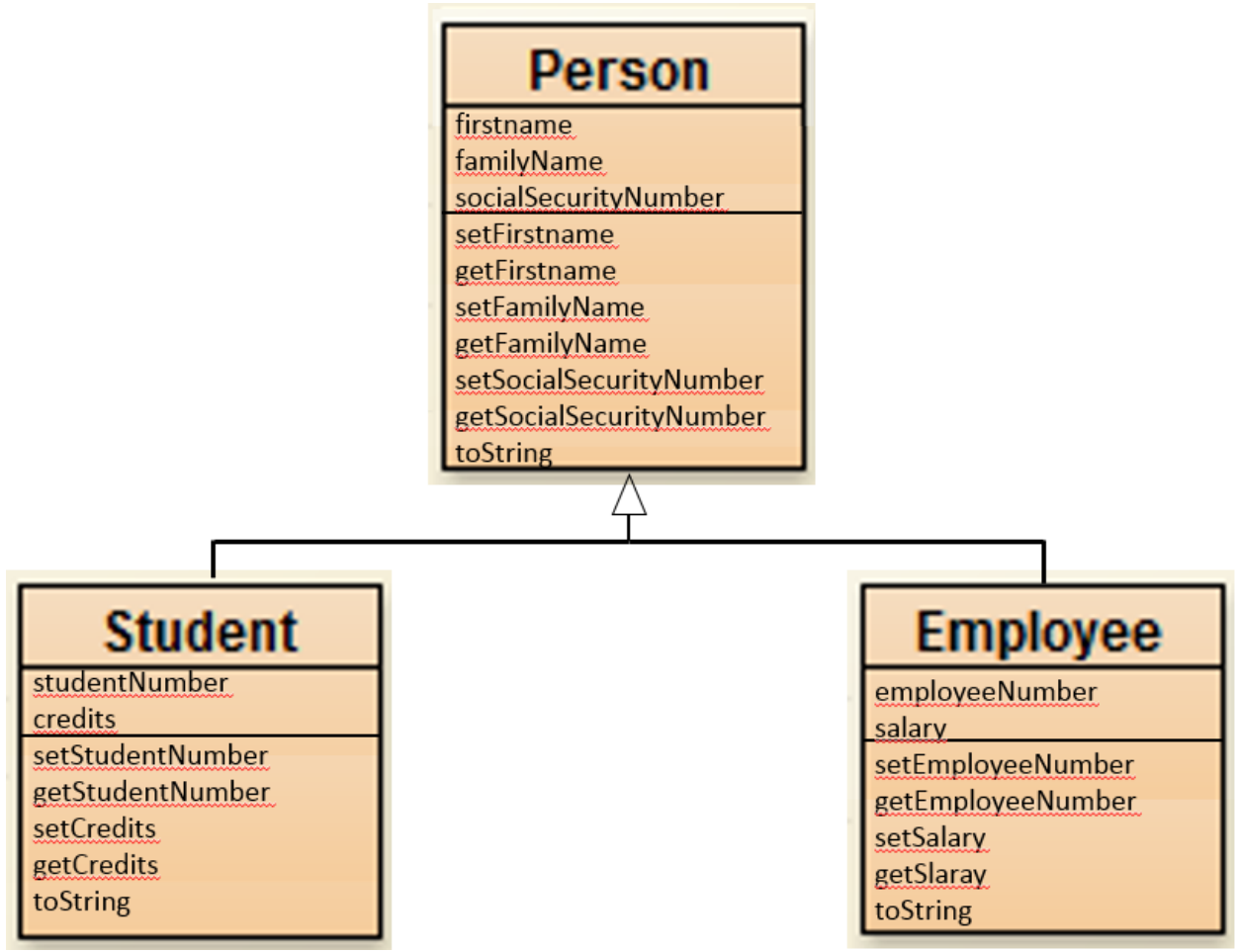
Legg til en *tilpasset* equals-metode i *passende* klasse(r) i prosjektet **`westerdals-v2`**.



Tenk på dette…

Er det *mulig* å lage objekter av klassen **`Person`**?

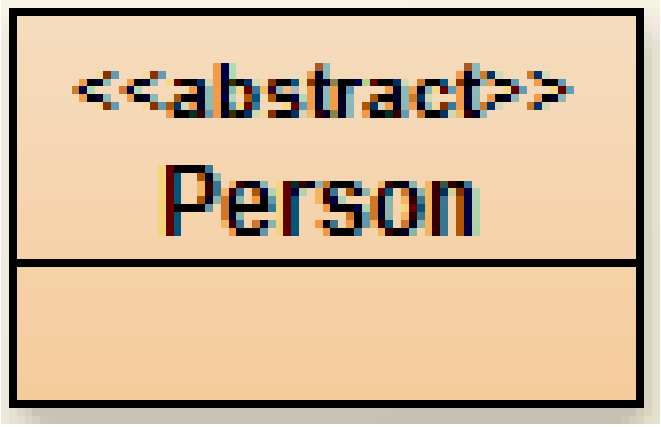
Er det *naturlig* (aktuelt) å lage objekter av klassen **`Person`**?



Klassen `Person`

Hvis vi vil gjøre det *umulig* å lage objekter av klassen **`Person`**

```
public abstract class Person
```



Nødvendig? *Nei*

Lurt?

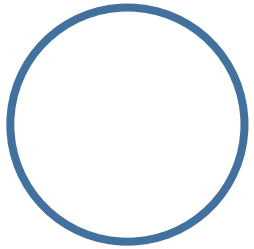
Klassen `Person`

Prøver nå å lage et objekt av klassen **`Person`**:

```
Person p = new Person("Anki", "Anka", "123");
```

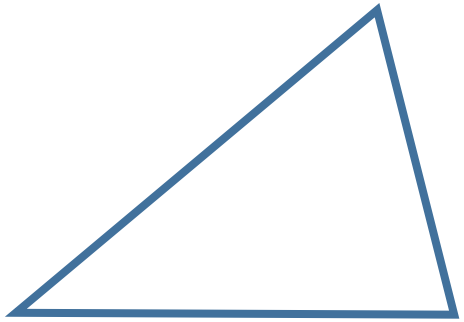
```
Person p = new Person("Anki", "Anka", "123");
```

Hva har disse klassene felles?

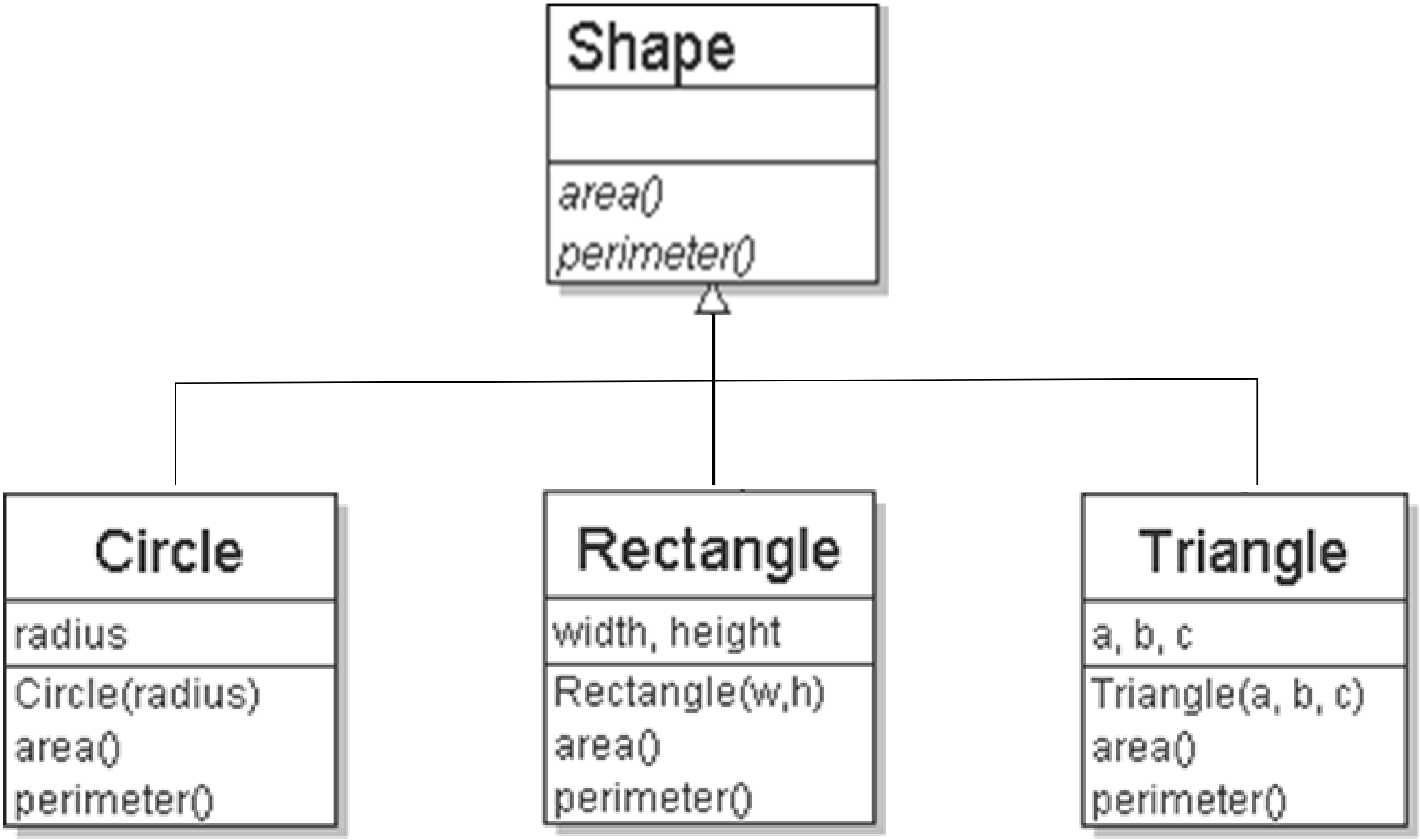


Circle
radius
Circle(radius) area() perimeter()

Rectangle
width, height
Rectangle(w,h) area() perimeter()



Triangle
a, b, c
Triangle(a, b, c) area() perimeter()


Shape
area()
perimeter()
Circle
radius
Circle(radius)
area()
perimeter()
Rectangle
width, height
Rectangle(w,h)
area()
perimeter()
Triangle
a, b, c
Triangle(a, b, c)
area()
perimeter()

Klassen `Shape`

Alle typer **`Shape`**'s har en omkrets (**`perimeter`**) og et areal (**`area`**).

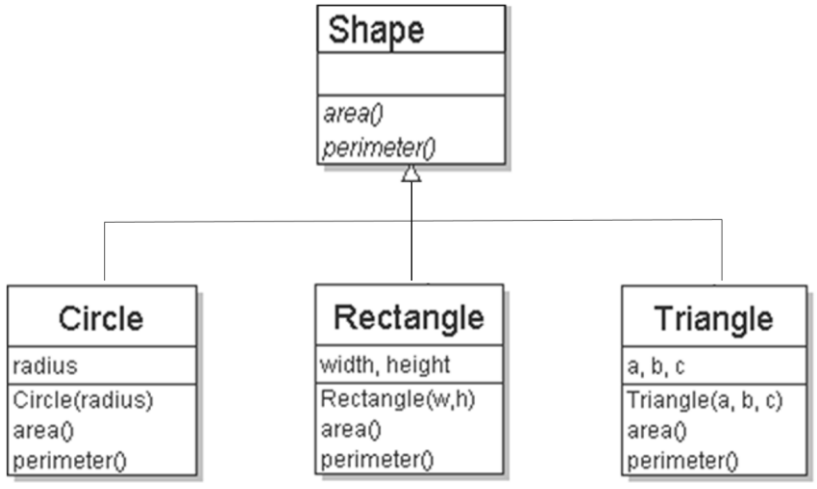
Og dette regnes ut på forskjellig måte for de ulike typene.

Klassen **`Shape`** kan derfor ikke definere disse metodene, siden disse altså er forskjellig for de ulike sub-klassene.

```
public class Shape {
  public Shape() { }

  public double getArea() {
    return ???;
  }
  public double getPerimeter() {
    return ???;
  }
}
```

Hvordan løses dette?



Klassen Shape

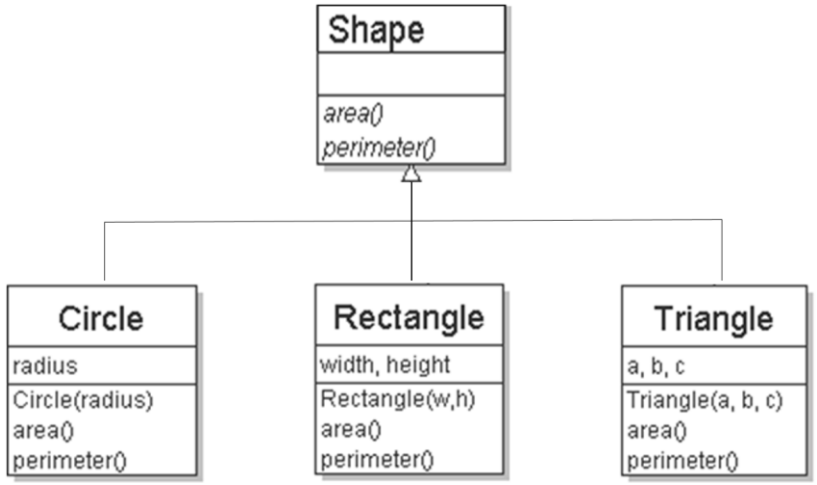
Én løsning:

```
public class Shape {
  public Shape() { }

  public double getArea() {
    return 0;
  }
  public double getPerimeter() {
    return 0;
  }
}
```

Ikke så meningsfullt, kanskje???

Men – fungerer det???



Klassen Rectangle

$$area = width * height$$
$$perimeter = 2 * (width + height)$$

```
public class Rectangle extends Shape {
  private double width;
  private double heigth;

  public Rectangle(double w, double h) {
    setWidth(w);
    setHeigth(h);
  }

  public void setWidth(double w){width = w;}
  public void setHeigth(double h){heigth = h; }
  public double getWidth() { return width; }
  public double getHeigth() { return width; }

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

  public double getPerimeter() {
    return 2.0 * (width + heigth);
  }
  ...
}
```

Her blir de arvede versjonene overskrevet

Oppretter objekt av klassen `Rectangle`

```
public class Client {
    public void mainMethod() {
        Rectangle r = new Rectangle(5, 10);
        System.out.println("Areal: " + r.getArea());
        System.out.println("Omkrets: " + r.getPerimeter());
    }
}
```

```
Areal: 50.0
Omkrets: 30.0
```

Oppretter objekt av klassen `Rectangle`

```
public class Client {
    public void mainMethod() {
        Shape s = new Rectangle(5, 10);
        System.out.println("Areal: " + s.getArea());
        System.out.println("Omkrets: " + s.getPerimeter());
    }
}
```

```
Areal: 50.0
Omkrets: 30.0
```

Klassen `Circle`

$$area = \pi r^2$$
$$perimeter = 2\pi r$$

```
public class Circle extends Shape {
  private double radius;

  public Circle(Point p, double radius) {
    super (p);
    setRadius(radius);
  }

  public void setRadius(double radius) {
    this.radius = radius;
  }

  public double getRadius() {
    return radius;
  }

  public double getArea() {
    return Math.PI * radius * radius;
  }

  public double getPerimeter() {
    return 2.0 * Math.PI * radius;
  }
}
```

Her blir de arvede versjonene overskrevet

Klassen `Triangle`

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$
$$perimeter = a + b + c$$
$$s = \frac{a+b+c}{2}$$

```
public class Triangle extends Shape {
   private double a;
   private double b;
   private double c;

   public Triangle(double a, double b, double c) {
      setA(a); setB(b); setC(c);
   }
   public void setA(double a) { this.a = a; }
   public void setB(double b) { this.b = b; }
   public void setC(double c) { this.c = c; }
   public double getA() { return a; }
   public double getB() { return b; }
   public double getC() { return c; }

   public double getArea() {
      double s = getPerimeter() / 2;
      return Math.sqrt(s * (s - a) * (s - b) * (s - c));
   }
   public double getPerimeter() {return a + b + c;}
}
```

Her blir de arvede versjonene overskrevet

Klassen `Shape` – en viktig betraktning

Klassen kan ikke definere de to metodene på fornuftig måte.
Det er riktigere å deklarere klassen og de to metodene som **`abstract`**.

```
public class Shape {
  public Shape() { }

  public double getArea() {
    return 0;
  }
  public double getPerimeter() {
    return 0;
  }
}
```

```
public abstract class Shape {
   private Point p;

   public Shape(Point p) {setPoint(p); }

   public void setPoint(Point p) {this.p = p; }
   public Point getPoint() {return p; }

   public abstract double getArea();

   public abstract double getPerimeter();

}
```

Merk!

Med dette blir det samtidig umulig å lage objekter av klassen **`Shape`**.
Det er heller ikke naturlig å lage objekter av denne klassen.

Klassen `Shape`

På samme måte som med klassen **`Person`**, kan vi nå behandle de tre typene **`Circle`**, **`Rectangle`** og **`Triangle`** som en **`Shape`**.

Vi kan altså "si":

```
Shape s1 = new Circle(...);
Shape s2 = new Rectangle(...);
Shape s3 = new Triangle(...);
```

Eller:

```
import java.util.ArrayList;
public class Client {
    public void mainMethod() {
        ArrayList<Shape> shapes = new ArrayList<Shape>();
        shapes.add(new Rectangle(18, 18));
        shapes.add(new Triangle(30, 30, 30));
        shapes.add(new Circle(12));

        for (Shape s : shapes) {
            String type = "";
            if (s instanceof Circle) type = "Sirkel: ";
            else if (s instanceof Rectangle) type = "Rektangel: ";
            else if (s instanceof Triangle) type = "Trekant: ";
            System.out.printf(
                type + "area = %6.2f, perimeter = %6.2f\n",
                s.getArea(),
                s.getPerimeter());
        }
    }
}
```

```java
import java.util.ArrayList;
public class Client {
    public void mainMethod() {
        ArrayList<Shape> shapes = new ArrayList<Shape>();
        shapes.add(new Rectangle(18, 18));
        shapes.add(new Triangle(30, 30, 30));
        shapes.add(new Circle(12));

        System.out.printf(
            type + "area = %6.2f, perimeter = %6.2f\n",
            s.getArea(),
            s.getPerimeter());
    }
}
}
```

```
Rektangel: area = 324,00, perimeter =  72,00
Trekant: area = 389,71, perimeter =  90,00
Sirkel: area = 452,39, perimeter =  75,40
```

abstract

- Klasser som det ikke er naturlig å lage objekter av, kan deklareres **`abstract`**.
- Metoder i en slik superklasse, som ikke kan "spesifiseres" (implementeres), men som *bør* ligge der, deklareres da også **`abstract`**.
- Merk: slike metoder får en "amputert" implementasjon:

```
public abstract double getArea();

public abstract double getPerimeter();
```

Oppgaver på øvingen

Se itsLearning