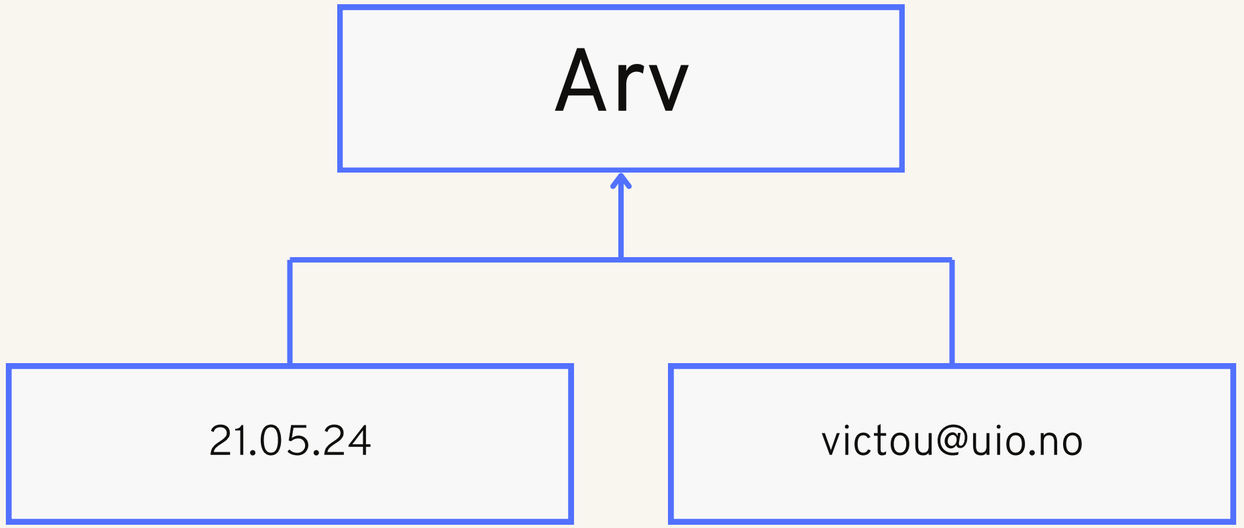
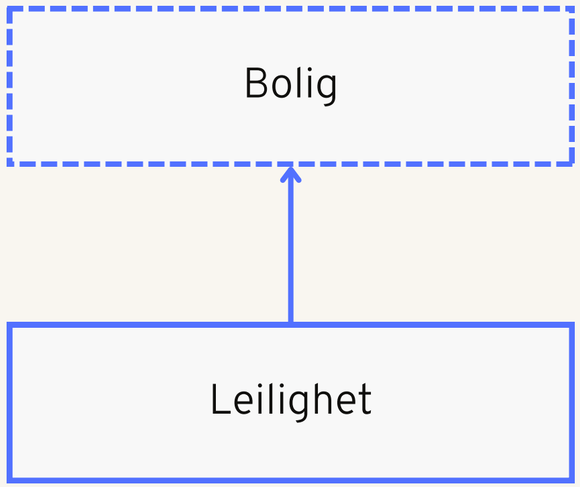


Menti: 6260 2064

Arv eller komposisjon?

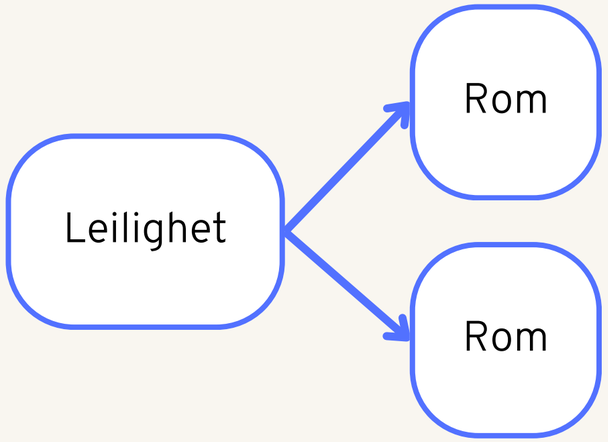
Arv

... er ....



Komposisjon

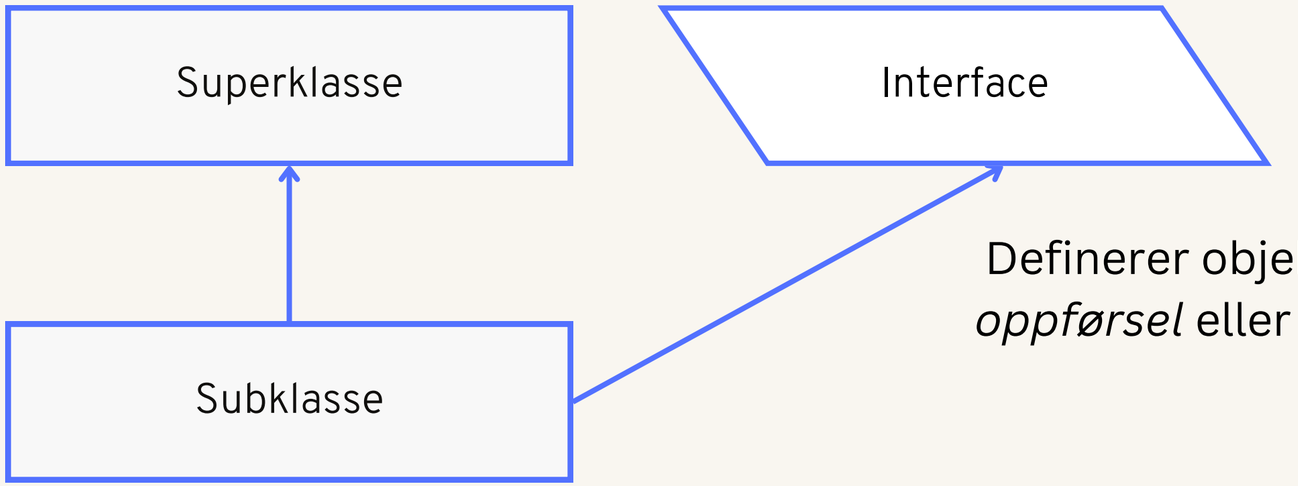
... har ...



Super- og subklasser

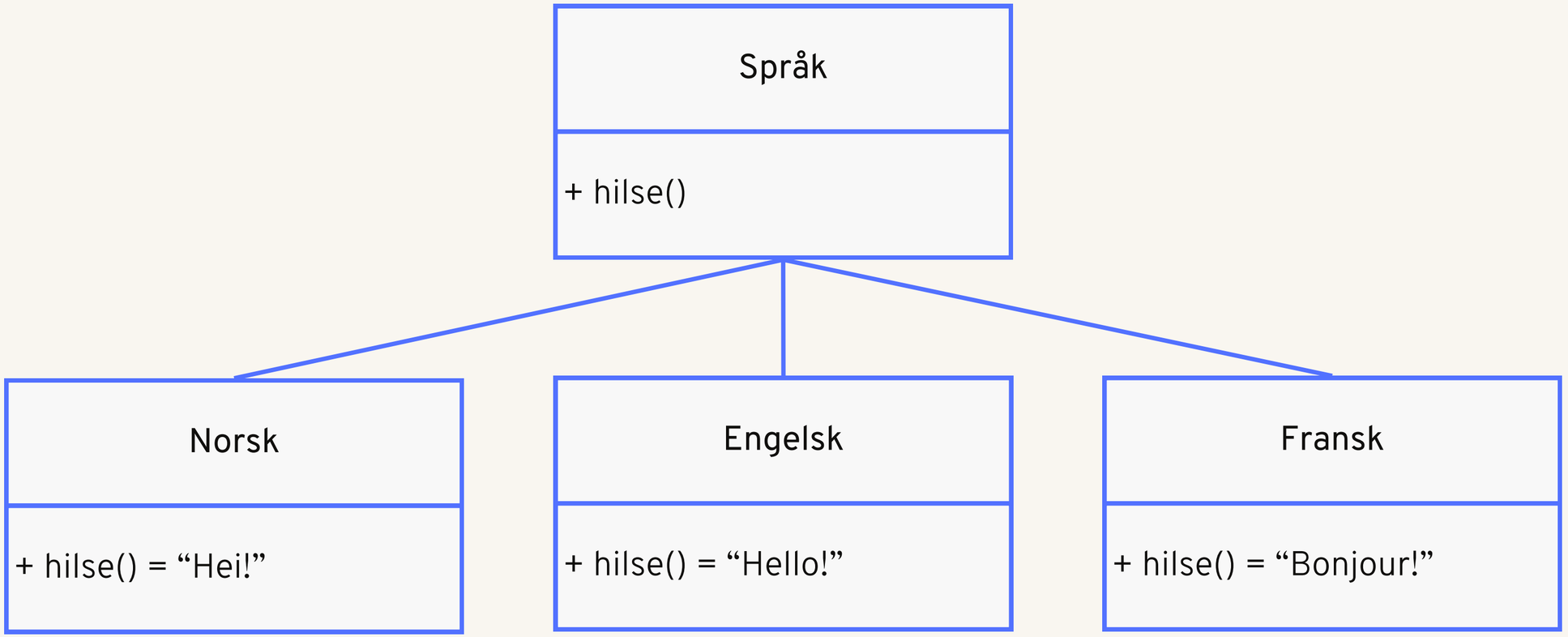
Beskriver et objekt med “egenskaper”

Har i tillegg *flere* egenskaper.



Definerer objektets *oppførsel* eller *rolle*.

Polymorfi


Språk
+ hilse()
Norsk
+ hilse() = “Hei!”
Engelsk
+ hilse() = “Hello!”
Fransk
+ hilse() = “Bonjour!”

Polymorfi i Java

```
class Spraak{
    String hilse(){return "?";}
}

class Norsk extends Spraak{
    @Override
    String hilse(){return "Hei!";}
}

class Engelsk extends Spraak{
    @Override
    String hilse(){return "Hello!";}
}
```

Lik *signatur*, flere *implementasjoner*.

@Override

```
class Spraak{
    String hilse(){return "?";}
}

class Norsk extends Spraak{
    @Override
    String hilse(){return "Hei!";}
}

class Engelsk extends Spraak{
    @Override
    String hilse(){return "Hello!";}
}
```

@Override overskriver en metode fra en superklasse / interface.

Gir Java beskjed om at metoden overskrives. :)

Super-nøkkelordet

```java
class Forelder{
    String navn;
    Forelder(String navn){
        this.navn = navn;
    }
    String navn(){return navn;}
}

class Barn extends Forelder{
    Barn(String navn){
        super(navn);
    }

    @Override String navn(){
        return super.navn() + " er et barn";
    }
}
```

"Super" referer enkelt og greit til *noe* i superklassen.

Kaller super-konstruktør

Kaller metode fra superklassen

Test: Super

```
class A{
    void metode(){System.out.println(x:"A");}
}

class B extends A{
    @Override void metode(){System.out.println(x:"B");}
    void m(){super.metode();}
}

class C extends B{
    @Override void metode(){
        super.metode();  // ??
        super.m(); // ??
    }
}
```

Hva skrives ut?

Test: Super

```
class A{
    void metode(){System.out.println(x:"A");}
}

class B extends A{
    @Override void metode(){System.out.println(x:"B");}
    void m(){super.metode();}
}

class C extends B{
    @Override void metode(){
        super.metode();  // B
        super.m(); // A
    }
}
```

Referansetyper

Avgjør hvordan objektet oppfører seg!

```
String eksamen = "jeg er stressa!";
Integer lykketall = 1010;
Object hva = new String(original:"Hvaaa?");
```

Referansetyper: Casting

Å caste er å bytte referansetype!

```
abstract class Emne{}
class IN1010 extends Emne{}
```

```
IN1010 in1010 = new IN1010();
Emne e = (Emne) in1010;
```

Gjøre et 1010-objekt til et Emne-objekt

Referansetyper: Casting

… å caste “nedover” er ikke lov, vet du hvorfor?

```
interface Ansatt{
    void jobb(int timer);
}
class Person{
    int personNummer = 1234;
}
class Gruppelaerer extends Person implements Ansatt{
    int konto = 0;
    @Override
    public void jobb(int timer){konto += timer * 200;}
}
class IN1010Grl extends Gruppelaerer{
    void repetisjon(){}
}
```

```
Gruppelaerer v = new Gruppelaerer(); // Grl-egenskaper
Ansatt a = (Ansatt) v; // Ansatt-egenskaper
Person p = (Person) v; // Person-egenskaper

p = new Person(); // Person-objekt
Gruppelaerer ny = (Gruppelaerer) p; // Ulovlig

IN1010Grl in1010 = (IN1010Grl) v; // Ulovlig
```

Interface som referansetype??

Ja, det går an!

```
interface Ansatt{
    void jobb(int timer);
    void betal(int sum);
}
```

Kun tilgang på Ansatt-metodene

```
static void lonning(Ansatt... ansatte){
    for(Ansatt a : ansatte){
        a.betal(sum:5_000);
    }
}
```

Statiske egenskaper

```java
class Student{
    static int antall;
    int kortNummer;
    Student(int kortNummer){
        this.kortNummer = kortNummer;
        antall++;
    }
}
```

Statiske egenskaper hører til *klasser*, ikke *instanser*.

```java
Student a = new Student(kortNummer:123);
Student b = new Student(kortNummer:234);
System.out.println(a.kortNummer);
System.out.println(Student.antall);
```

Kalles på med “Klassenavn.egenskap”