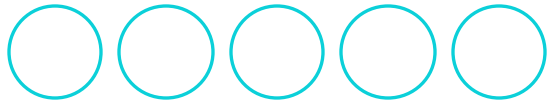
REPETISJON: ARV OG SUBKLASSER MED POLYMORFI

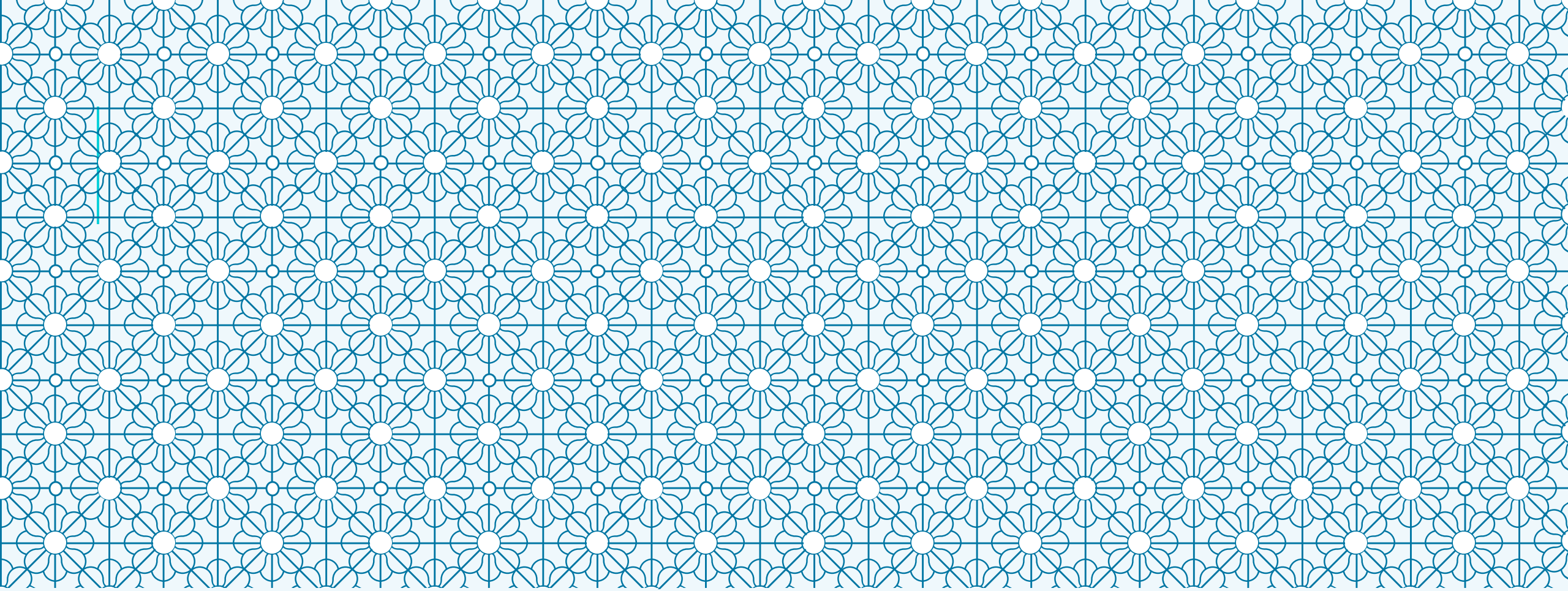
IN1010 gruppe 3

agneshb@uio.no

I DAG:

1. Arv og subklasser
2. Abstrakte klasser
3. Referansetyper
4. Arv og komposisjon – når skal hva brukes
5. Polymorfi





1. ARV OG SUBKLASSER

SUBKLASSE OG SUPERKLASSE

En **subklasse** er en klasse som bygger på en allerede spesifisert klasse (= **superklasse**)

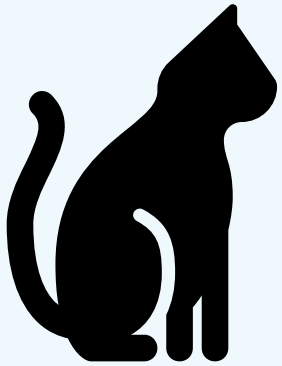
Subklassen **arver** egenskaper fra superklassen

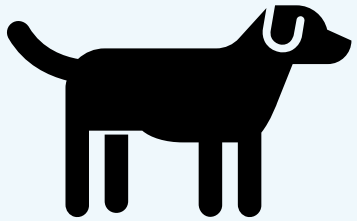
Subklassen er en «utvidelse» av superklassen («extends»)

Skrives:

```
Class <subklasse> extends <superklasse>
```

EKSEMPEL





En *superklasse* **Dyr**

To *subklasser* **Katt** og **Hund**

Alle dyr har noe til felles.

- Eks: spiser de noe, eller de lager en lyd.

Ulike typer dyr kan ha ulike egenskaper i tillegg til det som er felles.

- Eks: maler en katt, og en hund bjeffer.

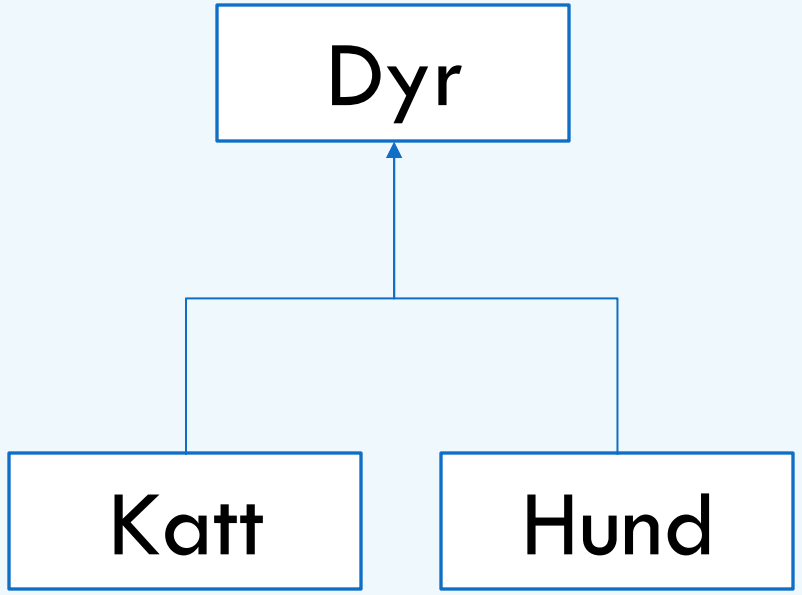
KLASSEHIERARKI

Superklasser og subklasser danner et hierarki

- Som kategorier og underkategorier
- Underkategorier, kan igjen ha underkategorier.

Dette kan tegnes med **klassehierarki**

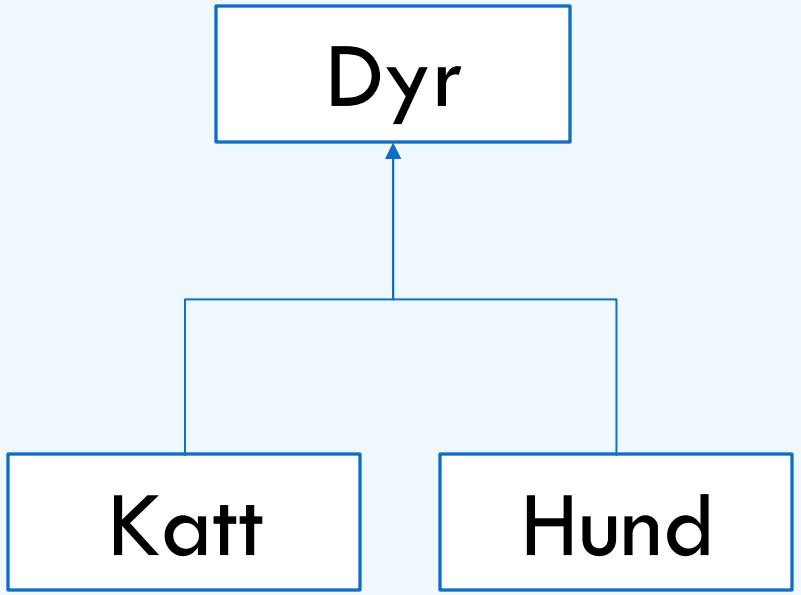
- Nyttig for å holde oversikt



```
public class Dyr {…
```

```
public class Hund extends Dyr{…
```

```
public class Katt extends Dyr{…
```

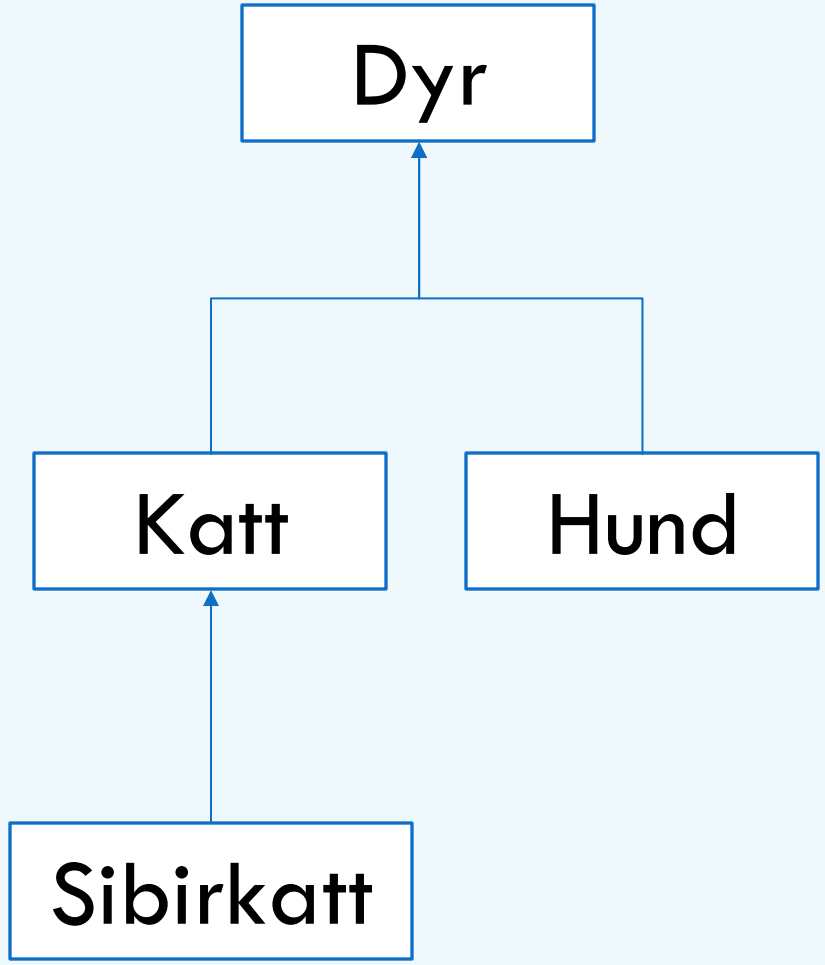


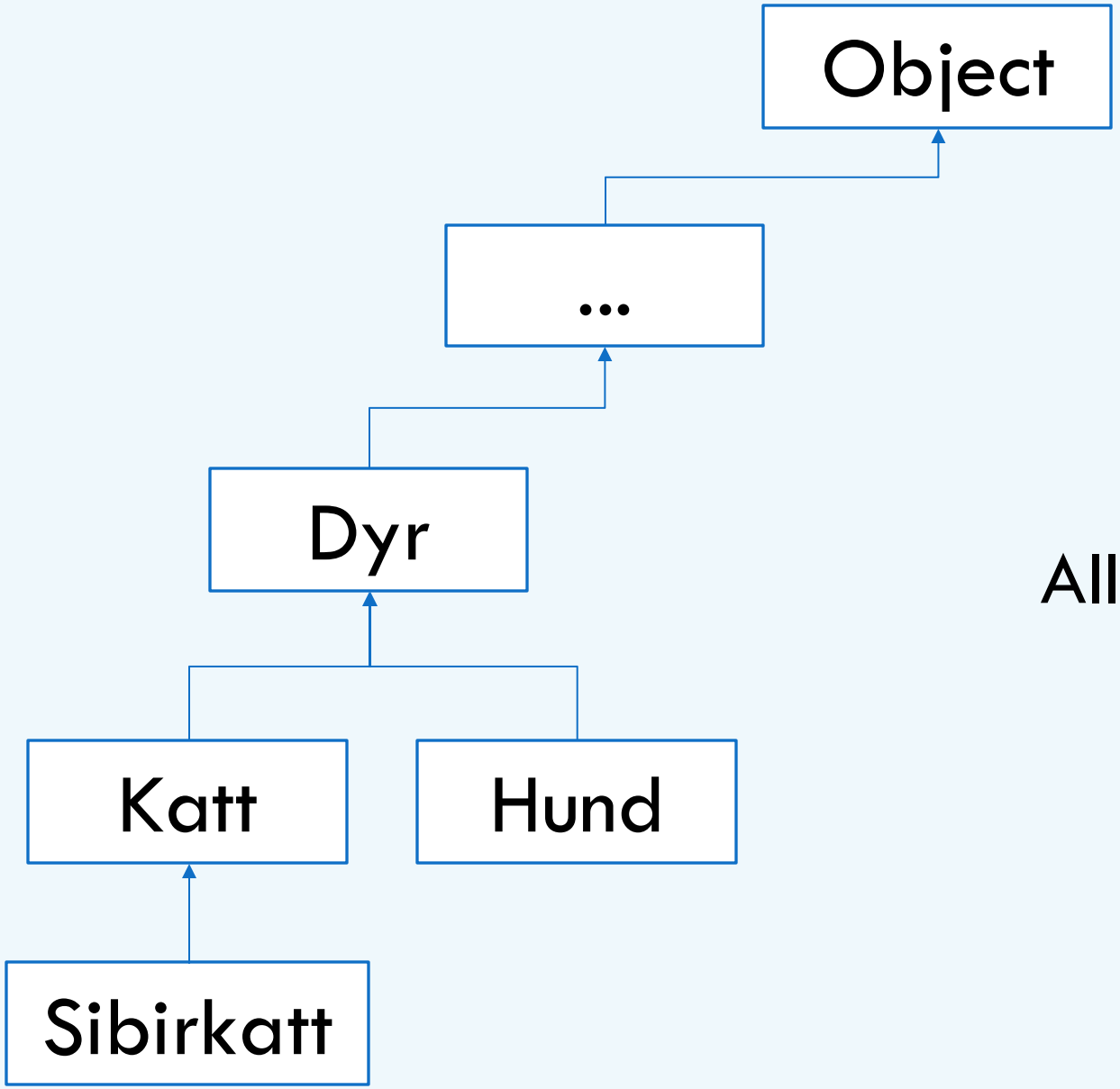
```
public class Dyr {…
```

```
public class Hund extends Dyr{…
```

```
public class Katt extends Dyr{…
```

```
public class Sibirkatt extends Katt{…
```





Alle klasser arver fra Object

SUPERKLASSEN DYR

Alle dyr må ha et navn (i dette eksempelet)

- Instansvariabelen navn

Alle dyr kan spise

- Metoden lagLyd()

Egenskaper som er spesielle for spesifikke typer dyr legges til i subklassene de tilhører.

```
public class Dyr {

    public String navn;


    public Dyr(String navn){
        this.navn = navn;
    }

    public void lagLyd(){
        System.out.println(x:"dyrelyd");
    }
```



SUBKLASSEN KATT

Katt arver fra Dyr

Alle objekter av klassen Katt må få et navn når de opprettes

Instansvariabelen «navn» arves fra Dyr
→ bruker super(navn) i konstruktøren.

```java
public class Dyr {

    public String navn;


    public Dyr(String navn){
        this.navn = navn;
    }

    public void lagLyd(){
        System.out.println(x:"dyrelyd");
    }
```

```java
public class Katt extends Dyr{

    Katt(String navn) {
        super(navn);
    }

}
```

SUPER() I KONSTRUKTØREN

I konstruktøren til en subklasse

Initialere instansvariabler fra superklassen.

Kaller superklassen sin konstruktør med subklassen sin(e) verdier for variablene.

Kan initialisere subklasse-spesifikke instansvariabler under.

```java
public class Person {

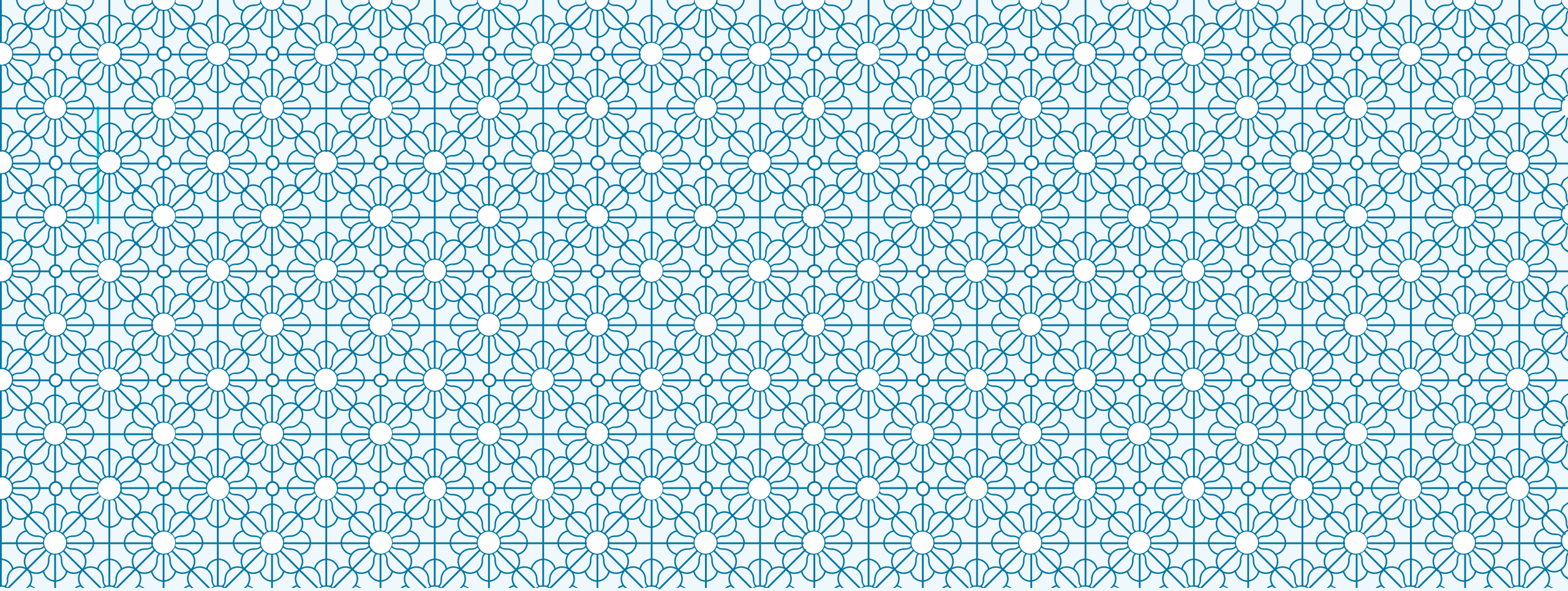
    //instansvariabler:
    String navn;
    int alder;

    //konstruktør:
    public Person(String navn, int alder){
        this.navn = navn;
        this.alder = alder;
    }
}
```

```java
public class Student extends Person{

    //instansvariabler for studenter, men ikke alle personer:
    String linje;
    int trinn;

    public Student(String navn, int alder, String linje, int trinn) {
        super(navn, alder);
        this.linje = linje;
        this.trinn = trinn;
    }
}
```



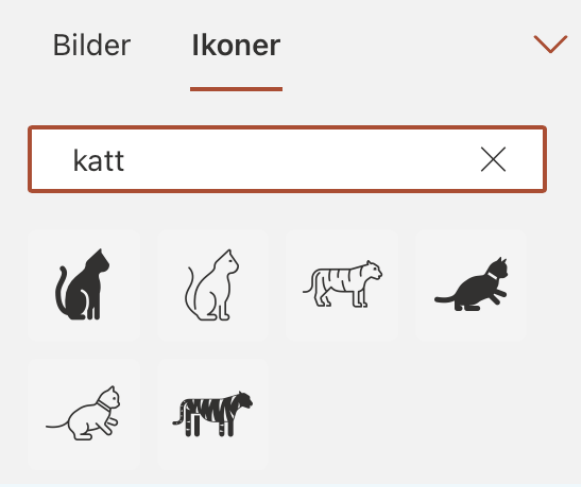
2. ABSTRAKTE KLASSER

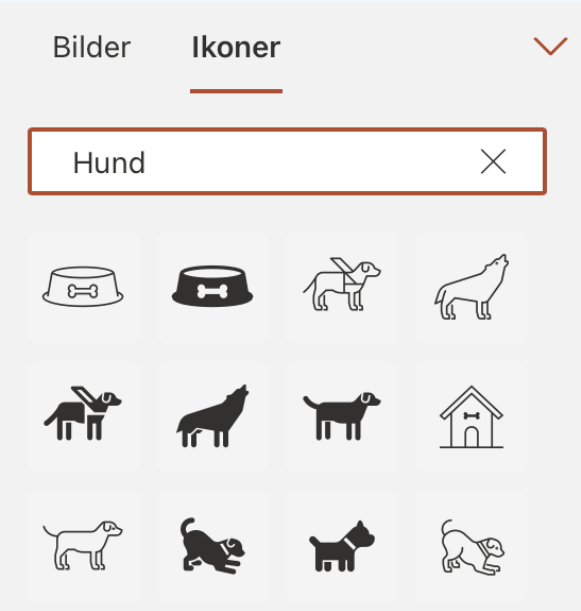
ABSTRAKTE KLASSER

Gir det egentlig mening å implementere et objekt av klassen Dyr?

Noen ganger ønsker man en superklasse som inneholder de instansvariablene og metodene som er felles for subklassene, men uten at denne superklassen kan implementeres.







ABSTRAKTE KLASSER FORTS.

`abstract`

Nå er det **ikke mulig å opprette et dyreobjekt:**

- ~~`Dyr d = new Dyr(«Jack»)`~~

Man kan fortsatt opprette katte- og hundeobjekter:

- `Katt k = new Katt(«Jack»)`
- `Hund h = new Hund(«Jackie»)`

```java
public abstract class Dyr {

    public String navn;

    public Dyr(String navn){
        this.navn = navn;
    }

    public void lagLyd(){
        System.out.println(x:"dyrelyd");
    }
}
```

3. REFERANSETYPER

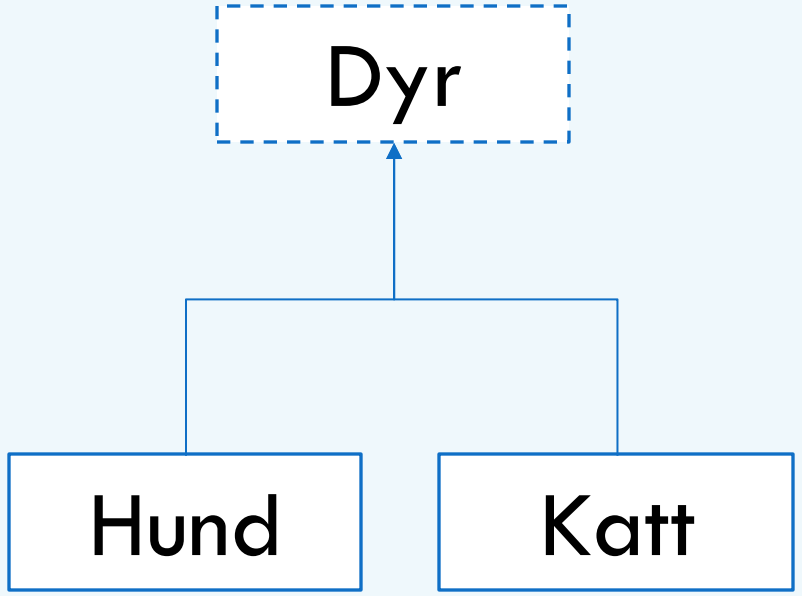


INSTANCEOF

Brukes for å sjekke om et objekt er av en type/klasse.

Et objekt er instanceof sin klasse, eller en klasse oppover i hierarkiet.

```
Hund hund = new Hund("Pluto")
hund instanceof Hund -> true
hund instanceof Dyr -> true
hund instanceof Katt -> false
```



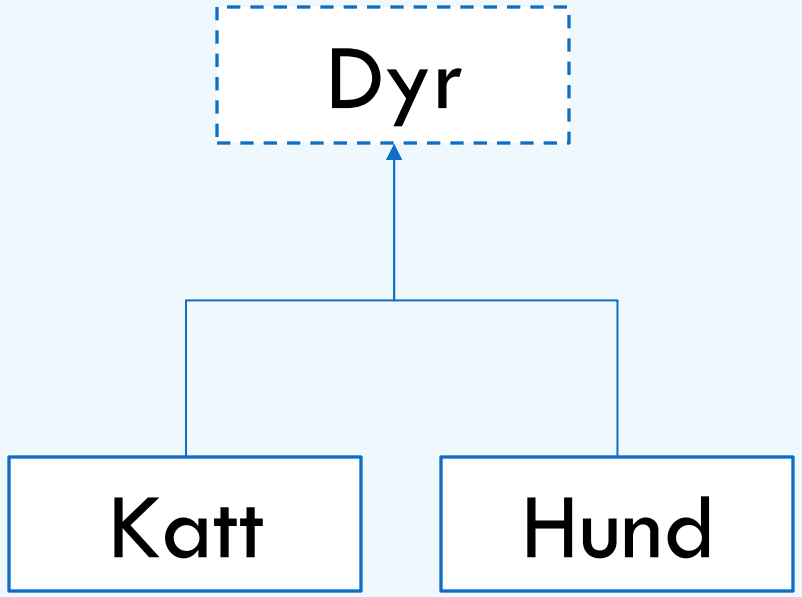
REFERANSETYPER

Referanser har lov til å peke nedover, ikke oppover.

Altså: klasser med samme egenskaper.

```
Dyr dyr = new Katt()
```

→ Katte-egenskapene er skjult.

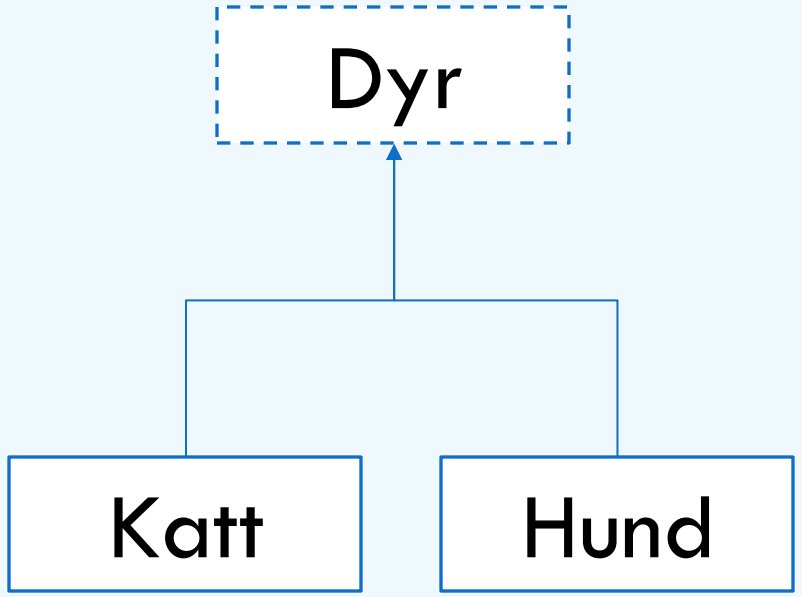


CASTING

Kan caste for å få tak i skjulte egenskaper hvis objektet er instance of klassen vi caster til.

```
Dyr dyr = new Katt()
Katt katt = (Katt) dyr;
```

Bruk instanceof for å unngå feilmelding.

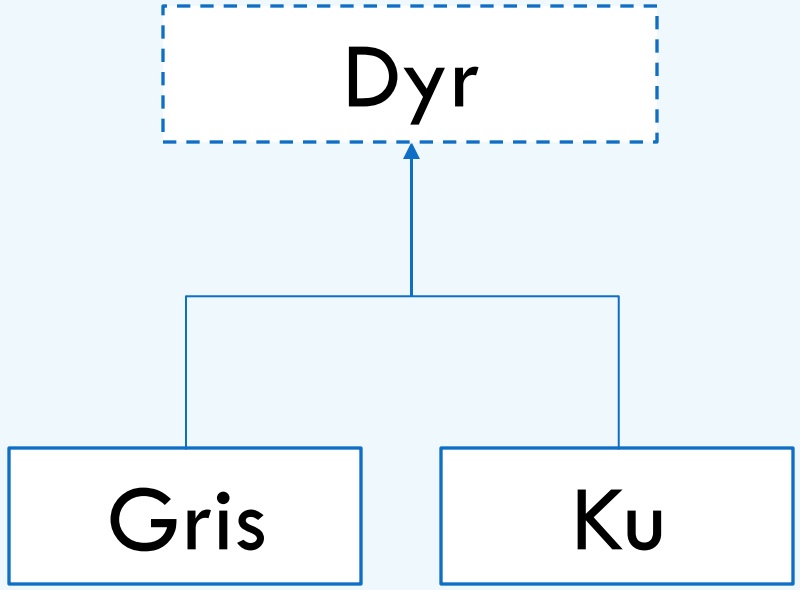


4. ARV OG KOMPOSISJON



NYE DYR, SAMME DYREKLASSE

- **Dyr er en abstrakt superklasse**
  - String Navn
  - void lagLyd()
- **Gris og Ku er subklasser av Dyr**
  - Arver fra Dyr
  - Har dermed også navn
  - Har også lagLyd()
- Gris har sin egen metode liggIGjørme().
- Ku har sin egen metode lagMelk().



ARV VS. KOMPOSISJON

Komposisjon	Arv
• Lager pekere til objekter av eksisterende klasser. • Lager objekter og kaller på metoder av disse klassene	• Oppretter en (sub)klasse som er en utvidelse av en eksisterende (super)klasse • Kan legge til ekstra variabler og metoder i subklassen

NÅR BRUKES ARV VS. KOMPOSISJON

Arv

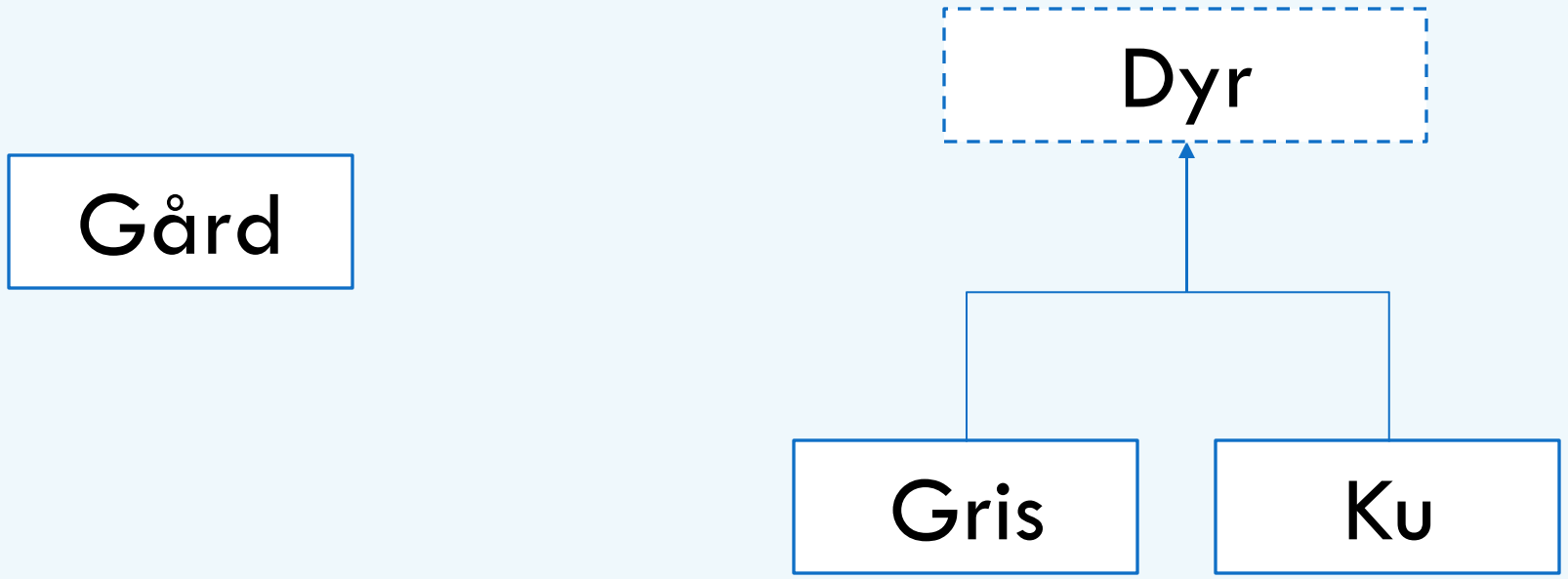
Er-en/et-relasjoner:

- En Ku er et Dyr
- En Gris er et Dyr

Komposisjon

Har-en/består-av-relasjoner:

- En Ku og en Gris er ikke en Gård, men kan være i en Gård
- En går har en/består av en Gris
- En Gård har en/består av en Ku



Pekere til objekter av typen Ku eller Gris i klassen Gård.
MEN Ku og Gris **arver ikke** fra Gård (eller motsatt)

```java
public abstract class Dyr {

    public String navn;


    public Dyr(String navn){
        this.navn = navn;

    }

    public void lagLyd(){
        System.out.println(x:"dyrelyd");
    }

}
```

Gris er et Dyr →
Gris arver fra Dyr

```java
public class Gris extends Dyr{

    public Gris(String navn) {
        super(navn);

    }

    @Override
    public void lagLyd(){
        System.out.println(x:"nøff nøff");
    }


    public void liggIGjørme(){
        System.out.println(this.navn + " ligger i gjørme");
    }
```

```java
public class Gård {

    String navn;
    ArrayList<Dyr> dyreliste = new ArrayList<>();

    public Gård(String navn){
        this.navn = navn;
    }

    void leggTilDyr(Dyr dyr){
        dyreliste.add(dyr);
    }
```

Ku er et Dyr →
Ku arver fra Dyr

```java
public class Ku extends Dyr{

    public Ku(String navn) {
        super(navn);
    }

    @Override
    public void lagLyd(){
        System.out.println(x:"Møøøø");
    }

    public void lagMelk(){
        System.out.println(this.navn +" lager melk");
    }
```

En Gård består
av
kuer og griser

INSTANCEOF I KOMPOSISJON

Kan gjøre ulike ting ut i fra hva slags type objekt det er.

Kalle ulike metoder ut i fra hva slags type objekt det er.

Eksempel med ulik **print** for ulike klasser ved hjelp av instanceof

```java
public class Gård {

    String navn;
    ArrayList<Dyr> dyreliste = new ArrayList<>();

    public Gård(String navn){
        this.navn = navn;
    }

    void leggTilDyr(Dyr dyr){
        dyreliste.add(dyr);
    }

    void hilsPåDyrene(){
        for (Dyr dyr : dyreliste) {
            if (dyr instanceof Gris){
                System.out.println(x:"nøff nøff");
            }
            else if (dyr instanceof Ku){
                System.out.println(x:"Møøøø");
            }
        }
    }
```

```java
Run | Debug
public static void main(String[] args) {
    Gård g = new Gård(navn:"Gruppe 3 gård");

    Ku ku1 = new Ku(navn:"Ferdinand");
    Gris gris1 = new Gris(navn:"Nasse nøff");
    Ku ku2 = new Ku(navn:"Heidi");

    g.leggTilDyr(ku1);
    g.leggTilDyr(gris1);
    g.leggTilDyr(ku2);

    g.hilsPåDyrene();

}
```

```
Møøøø
nøff nøff
Møøøø
```

Eksempel med ulike **metodekall** for ulike klasser ved hjelp av instanceof

```
//hvis dyret er en gris: kaller liggIGjørme()
//hvis dyret er en ku: kalles lagMelk()
//Må caste til hhv Gris og Ku for å få tilgang
//til metoder spesifikk fir Gris eller Ku
void dyreaktiviteter(){
    for (Dyr dyr : dyreliste) {
        if (dyr instanceof Gris){
            ((Gris) dyr).liggIGjørme();
        }
        else if (dyr instanceof Ku){
            ((Ku) dyr).lagMelk();
        }
    }
}
```

```
public static void main(String[] args) {
    Gård g = new Gård(navn:"Gruppe 3 gård");

    Dyr ku1 = new Ku(navn:"Ferdinand");
    Dyr gris1 = new Gris(navn:"Nasse nøff");
    Dyr ku2 = new Ku(navn:"Heidi");

    g.leggTilDyr(ku1);
    g.leggTilDyr(gris1);
    g.leggTilDyr(ku2);

    g.dyreaktiviteter();
```

```
Ferdinand lager melk
Nasse nøff ligger i gjørme
Heidi lager melk
```

5. POLYMORFI



POLYMORFI

Metoder med **samme signatur** på tvers av superklasser og subklasser får **ulik funksjonalitet**

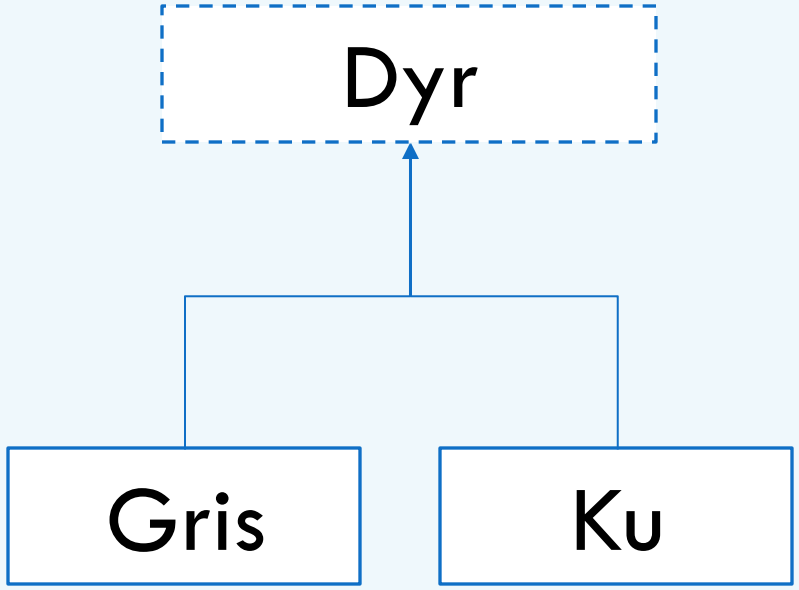
POLYMORFI

Kan redefinere en metode fra superklassen slik at den passer til den enkelte subklassen.

Metoden må ha samme signatur som i superklassen

Kalles polymorfe metoder

Bruker **@Override** for redefinering av metoder i subklassene

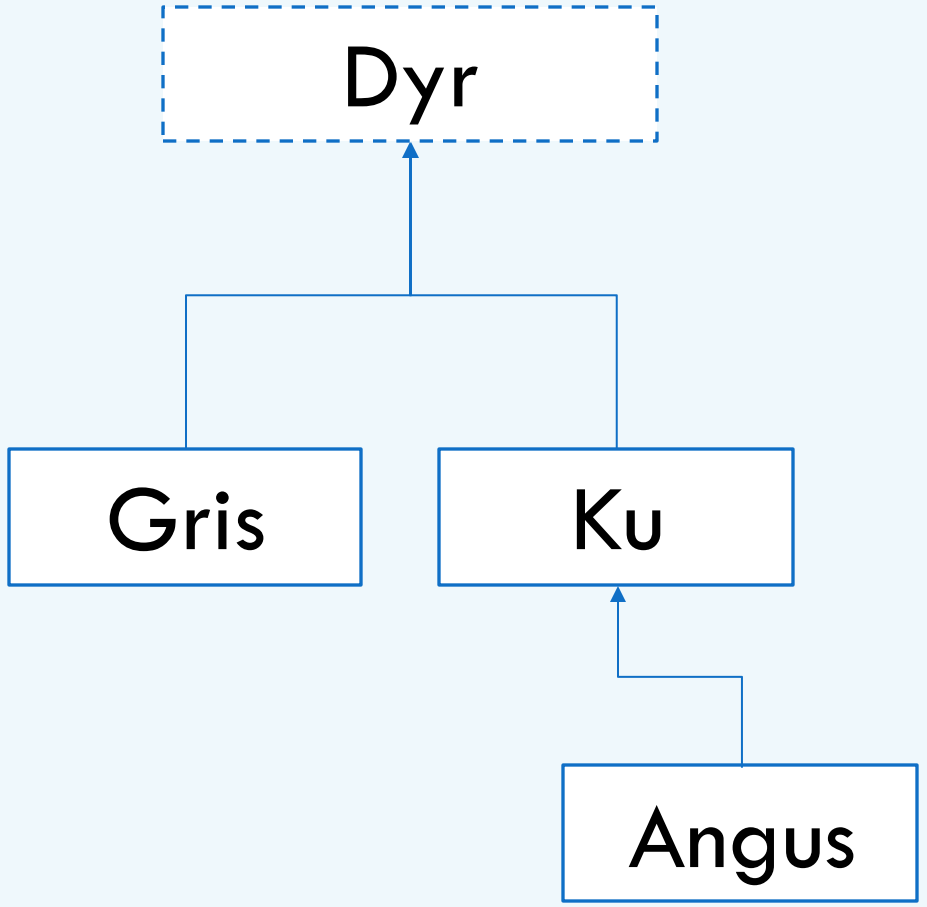


POLYMORFI FORTS.

Dersom subklassen ikke har redefinert metoden, vil superklassens metode kjøres

Kall på `super.<metodenavn>()` om du heller ønsker superklassens metode

Metoden i objektet som er definert dypest i klassehierarkiet som utføres




public abstract class Dyr {
public String navn;
public Dyr(String navn){
this.navn = navn;
}
public void lagLyd(){
System.out.println(x:"dyrelyd");
}
}
public class Gris extends Dyr{
public Gris(String navn) {
super(navn);
}
@Override
public void lagLyd(){
System.out.println(x:"nøff nøff");
}
public void liggIGjørme(){
System.out.println(this.navn + " ligger i gjørme");
}
public class Ku extends Dyr{
public Ku(String navn) {
super(navn);
}
@Override
public void lagLyd(){
System.out.println(x:"Møøøø");
}
public void lagMelk(){
System.out.println(this.navn +" lager melk");
}

EKSEMPEL POLYMORFI ISTEDENFOR INSTANCEOF

Kan bruke polymorfi fremfor instanceof for hilsPåDyrene()

```java
//med instancof:
void hilsPåDyrene(){
    for (Dyr dyr : dyreliste) {
        if (dyr instanceof Gris){
            System.out.println(x:"nøff nøff");
        }
        else if (dyr instanceof Ku){
            System.out.println(x:"Møøøø");
        }
    }
}

//med polymorfi:
void hilsPåDyrene2(){
    for (Dyr dyr : dyreliste) {
        dyr.lagLyd();
    }
}
```

```java
public static void main(String[] args) {
    Gård g = new Gård(navn:"Gruppe 3 gård");

    Ku ku1 = new Ku(navn:"Ferdinand");
    Gris gris1 = new Gris(navn:"Nasse nøff");
    Ku ku2 = new Ku(navn:"Heidi");

    g.leggTilDyr(ku1);
    g.leggTilDyr(gris1);
    g.leggTilDyr(ku2);

    g.hilsPåDyrene2();

}
```

```
Møøøø
nøff nøff
Møøøø
```

OVERRIDE OG OVERLOADE

Override:

- Bruker samme signatur som superklassen sin metode, og overskygger metoden med ny funksjonalitet.
- Polymorfe metoder

Overload:

- Man kan ha flere definisjoner av en metode med samme navn, men med forskjellige parametre/forskjellige signaturer.
- Ikke polymorfe metoder

FORSKJELL PÅ == OG .EQUALS()

equals() sjekker om de to elementene som sammenlignes har samme verdi

== sjekker om de to elementene peker til samme plass i minnet

- NB! Noen ganger vil man få forventet resultat med == for sammenligning, men det er ikke alltid tilfelle

```
public static void main(String[] args) {

    String a = "hei";
    String b = "hei";

    //sjekker likhet i verdi/innhold:
    System.out.println(a.equals(b));

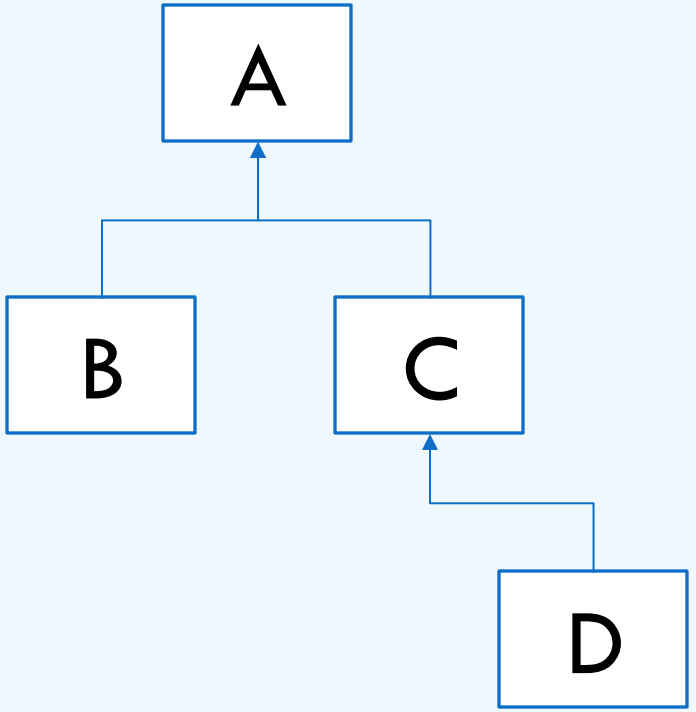
    //sjekker likhet i minneområde:
    System.out.println(a == b);
}
```

OPPGAVE 2

```
class A {}
class B extends A {}
class C extends A {}
class D extends C {}

class TrueOrFalse {
        public static void main(String[] args) {
                A a = new A();
                B b = new B();
                C c = new C();
                D d = new D();
                A e = new D();
                A f = new B();
                System.out.println("a) " + (b instanceof A));
                System.out.println("b) " + (c instanceof D));
                System.out.println("c) " + (d instanceof A));
                System.out.println("d) " + (e instanceof C));
                System.out.println("e) " + (f instanceof C));
        }
}
```

Tips: Tegn et klassehierarki



OPPGAVE 2 FORTS.

```
//Tilfelle 1
B ab = (B) a;

//Tilfelle 2
if( e instanceof B ){
   B eb = (B) e;
}

//Tilfelle 3
D ed = (D) e;

//Tilfelle 4
if( e instanceof D ){
   D ed = (D) e;
}

//Tilfelle 5
B fb = (B) f;

//Tilfelle 6
C ec = (C) e;
```

```
A a = new A();
B b = new B();
C c = new C();
D d = new D();
A e = new D();
A f = new B();
```

Tips : kan kun caste oppover

