IN1010

Repetisjon Uke 3

Arv

(Basert på Eyvind Axelsen og Stein Gjessings presentasjoner)

7. februar, 2025

Subklasser

Når vi lager klasser for et formål, så oppdager vi ofte at mye er felles, og at forskjellene ikke alltid er så store.

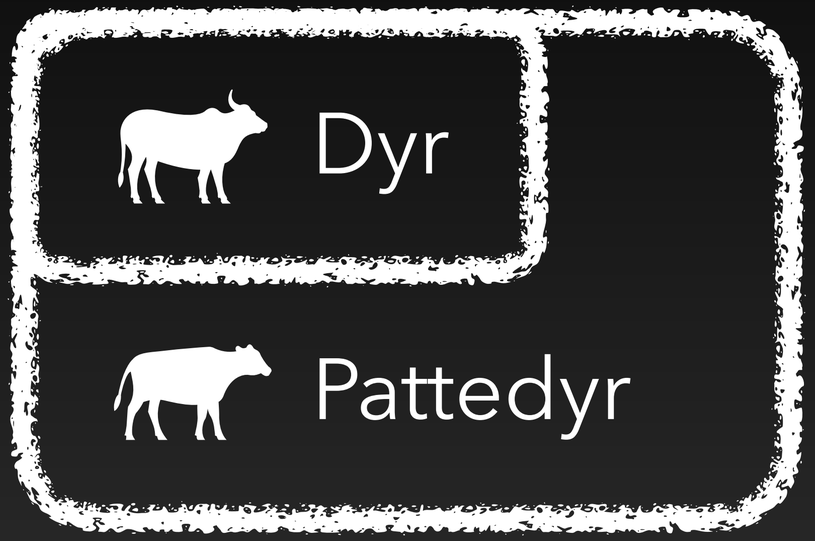
Da er det fint at vi kan lage en klasse for alt felles, som vi kan utvide til ulike lignende klasser.

Vi kaller det arv og subklasser.

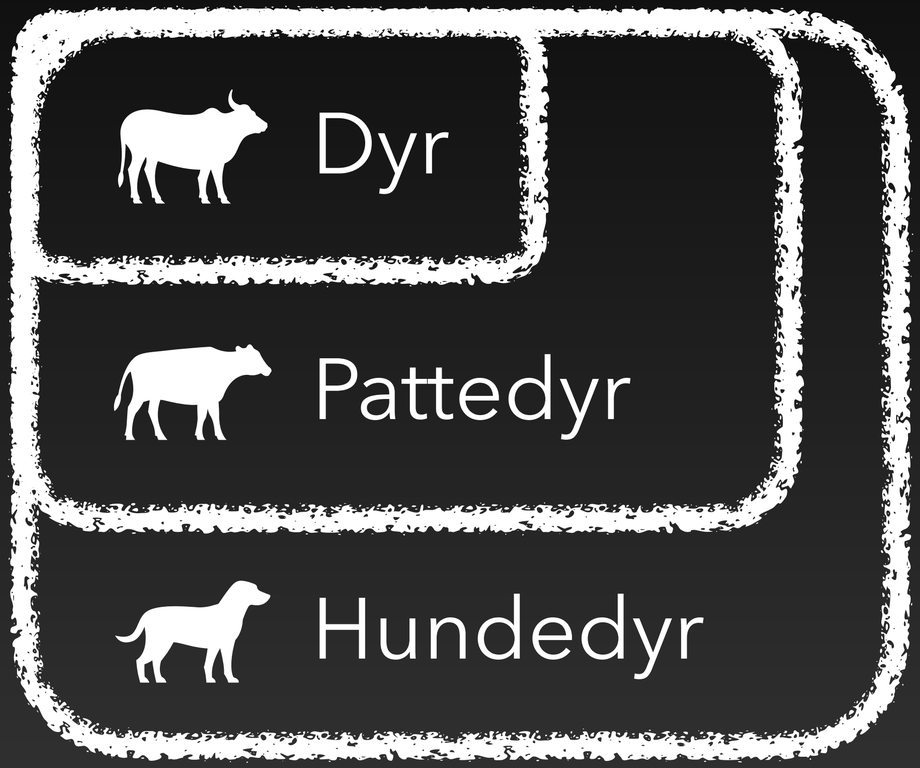
Subklasser



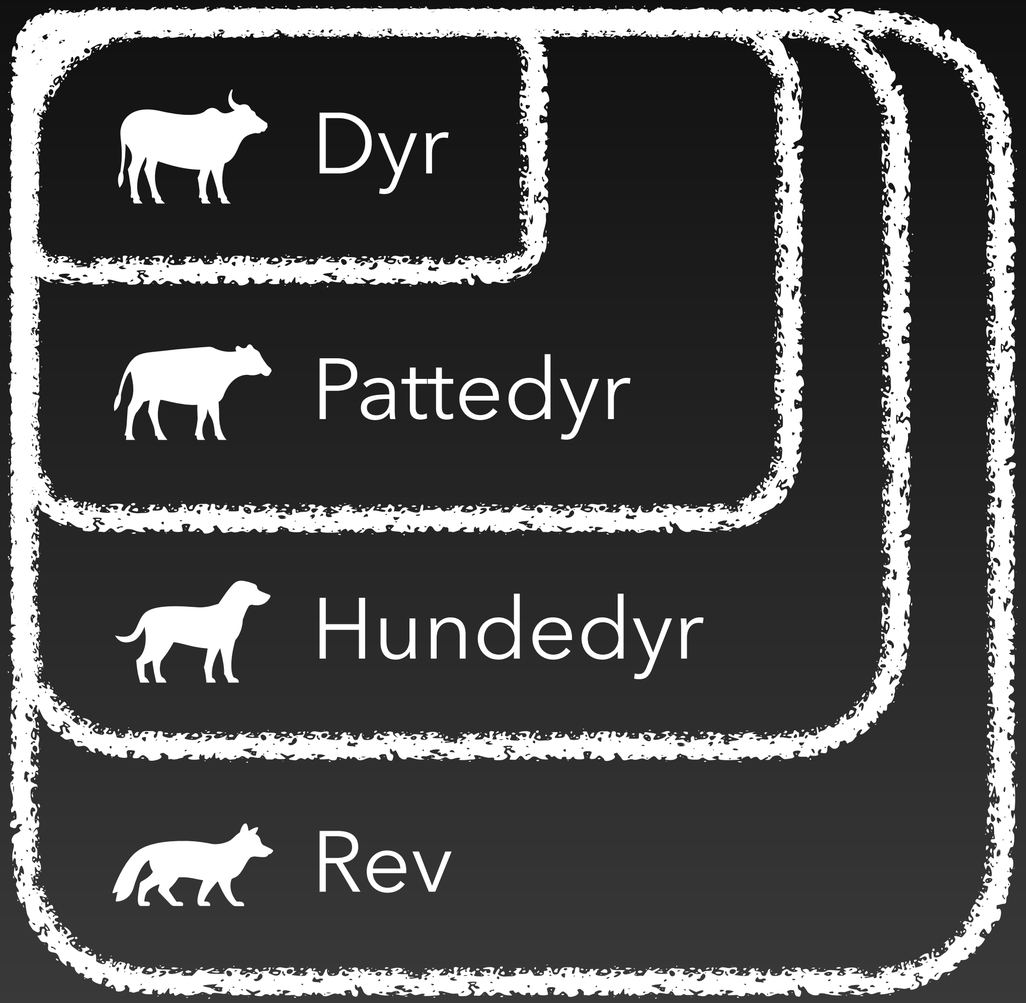
Subklasser



Subklasser



Subklasser



Subklasser

```
class Klasse { … }

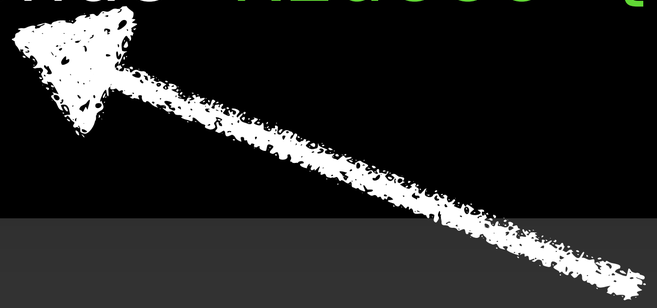
class Subklasse extends Klasse { … }
```

Subklasser brukes til å utvide klasser

Subklasser

```
class Klasse { … }

class Subklasse extends Klasse { … }
```



Når vi utvider en annen klasse så bruker vi *extends*

Subklasser

```
class Klasse { … }

class Subklasse extends Klasse { … }
```



Subklassen får alle instansvariabler og metoder fra klassen over

Vi kaller denne klassen for *super*

Subklasser

```
class Dyr {
  boolean erLevende = true;
  String navn;
  Dyr[] foreldre;
  void giNyttNavn(String navn) {
    this.navn = navn;
  }
}
```

Vi finner egenskaper og grensesnitt som alle dyr kan ha

Subklasser

```
class PatteDyr extends Dyr {
  String pelsfarge;
  void settPelsfarge(String farge) {
    pelsfarge = farge;
  }
}
```

Så kan vi gå et nivå ned, og legge til egenskaper unikt for denne subklassen

Subklasser

```
class HundeDyr extends PatteDyr {
}
```

Vi kan ha subklasser av subklasser

Subklasser

```
class Rev extends HundeDyr {
}
```

Vi kan ha subklasser av subklasser

Objekter og typer

```
Rev rev = new Rev();
rev.giNyttNavn("Reven");
rev.settPelsFarge("rød");
```

Vi oppretter objekter av subklasser som en hvilken som helst klasse

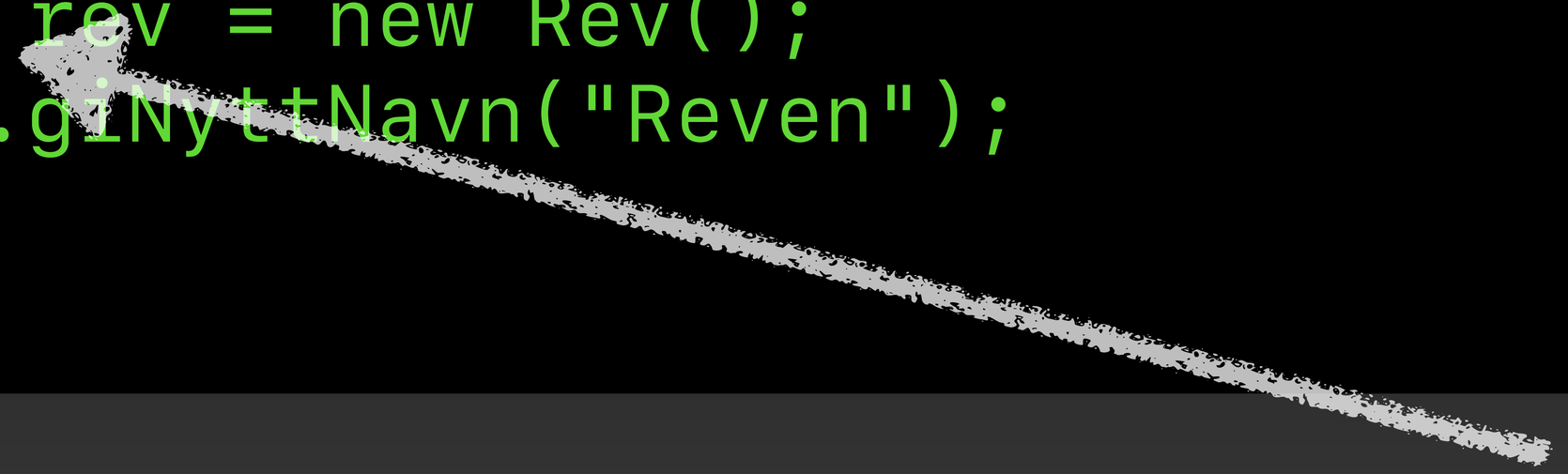
Objekter og typer

```
Rev rev = new Rev();
rev.giNyttNavn("Reven");
rev.settPelsFarge("rød");
```

Og kan dermed bruke metodene som er arvet fra klassene over

Objekter og typer

```
Dyr rev = new Rev();
rev.giNyttNavn("Reven");
```



Det er også mulig å lage en *referanse* til et objekt, men som en *type* av klassene over

Objekter og typer

```
Dyr rev = new Rev();
rev.giNyttNavn("Reven");

rev.settPelsFarge("rød");
```

Men da mister vi tilgang til variablene og metodene i subklassene

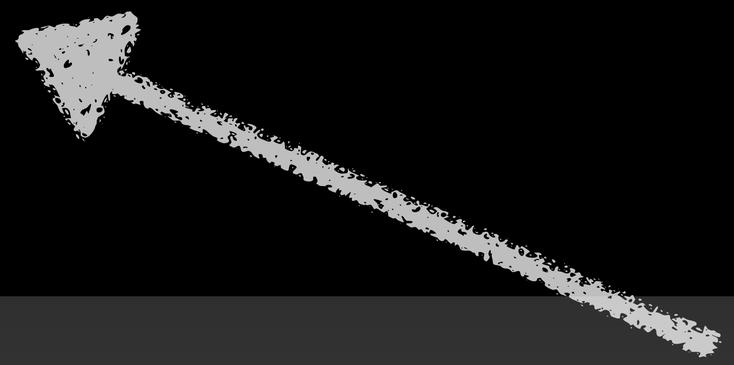
Objekter og typer

```
Dyr rev = new Rev();
Dyr ku = new Sau();
Dyr[] dyrene = { rev, sau };
```

Vi gjør det om vi vil sammenligne, hvis vi lager felles metoder, i andre klasser, eller f.eks. lage en array av objekter som kun er felles i et høyere nivå i hierarkiet

Objekter og typer

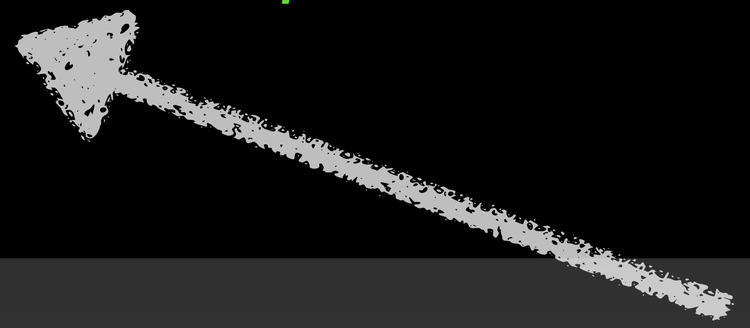
```
Dyr rev = new Rev();
```



Men objektet er fortsatt av typen som vi konstruerer det med

Objekter og typer

```
Dyr rev = new Rev();
Rev reven = (Rev) rev;
```



For å få tilbake de fulle egenskapene, så kan vi konvertere (casting) til en ny referanse

Objekter og typer

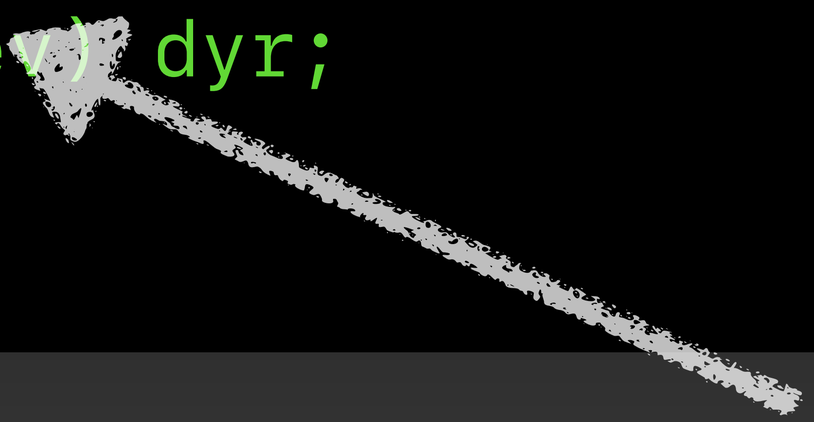
```
Dyr[] dyrene = { rev, sau };
for (Dyr dyr : dyrene) {
    Rev reven = (Rev) dyr;
}
```

Men det er ikke alltid vi vet hva slags objekt som det opprinnelig ble konstruert som

Her vil det bli feil hvis vi prøver å gjøre en sau til en rev

Objekter og typer

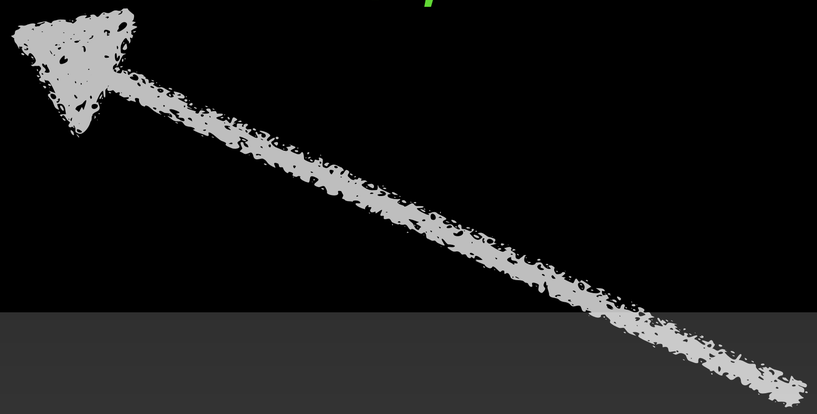
```
Dyr[] dyrene = { rev, sau };
for (Dyr dyr : dyrene) {
   if (dyr instanceof Rev) {
      Rev reven = (Rev) dyr;
   }
}
```



Da kan vi sjekke med *instanceof* slik at bare reven blir konvertert til en ordentlig rev

Konstruktører og super

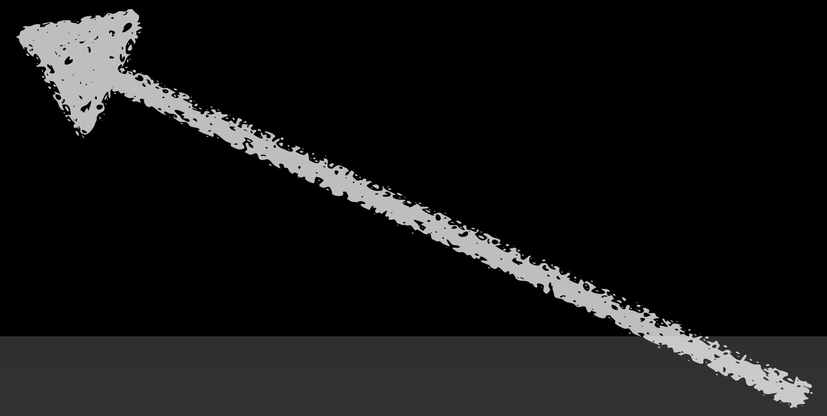
```
class Rev extends Dyr {
   Rev (String navn) {
      super(navn);
   }
}
```



Konstruktøren kan kalle på super-klassen (klassen over) slik

Konstruktører og super

```
class Dyr {
   String navn;
   Dyr (String navn) {
      this.navn = navn;
   }
}
```



Da er det denne konstruktøren som kjøres når vi bruke *super*

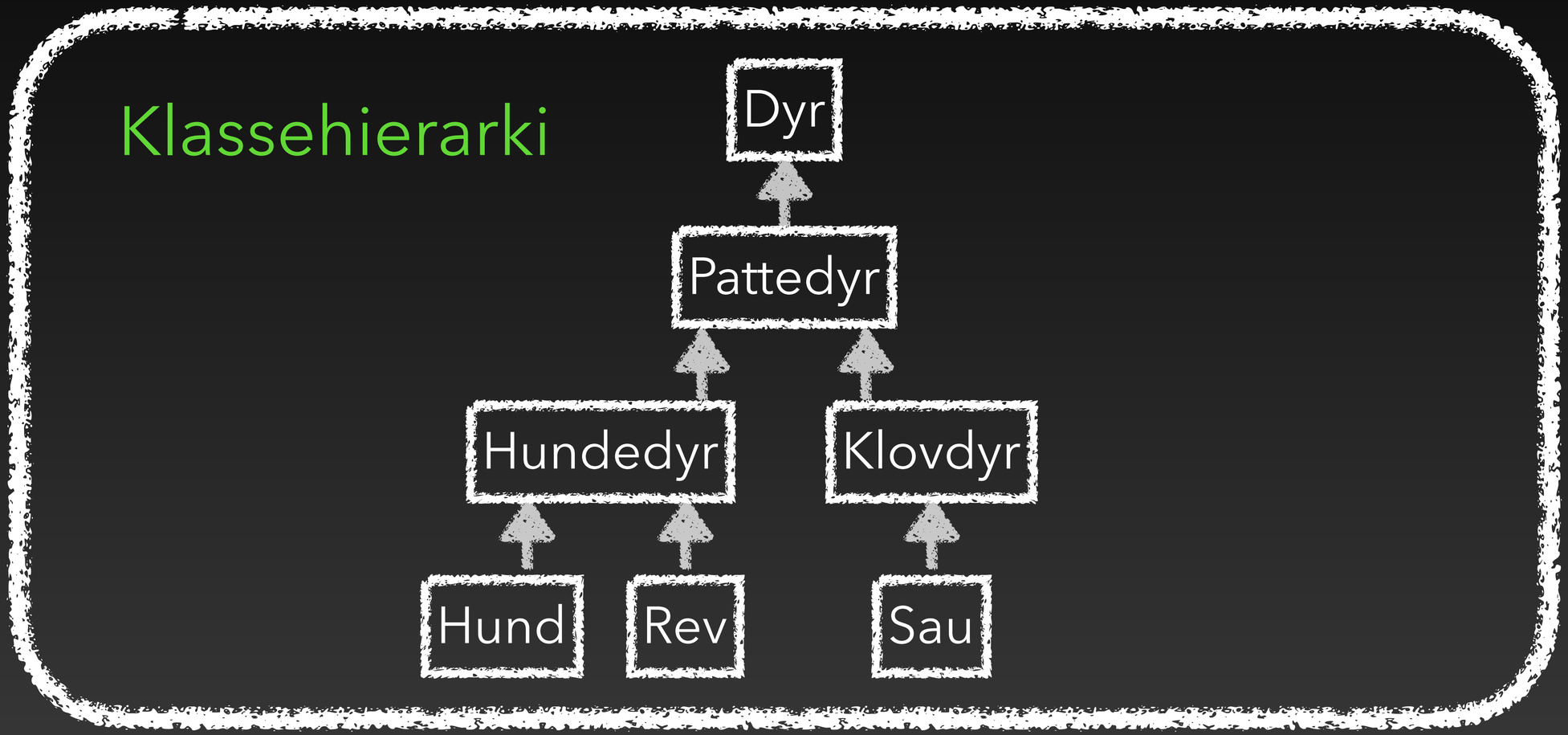
Konstruktører og super

```
class Rev extends Dyr {
   Rev (String navn) {
      super(navn);
   }
}
```

Super kan også brukes andre steder i klassen for å referere til superklassen

Eksempel

Diagrammer


Klassehierarki
Dyr
Pattedyr
Hundedyr
Klovdyr
Hund
Rev
Sau

Tilganger

```
public
protected
private
```

Java har tre typer tilgangs-modifikatorer

De kan brukes både for klasser og inne i klasser, vi ser på den siste nå

Tilganger

```
public
protected
private
```

Public betyr at vi har tilgang til metoden eller variabelen overalt

Tilganger

```
public
protected
private
```

Private gjør at variabelen eller metoden bare kan nås internt i klassen

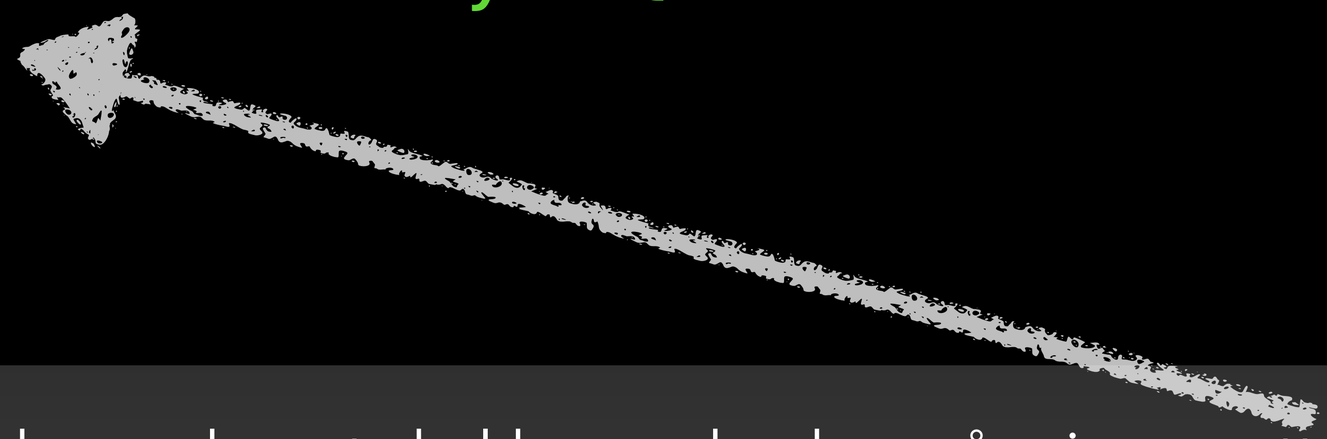
Tilganger

```
public
protected
private
```

Hvis variabelen eller metoden er markert med *protected*, så er den som *private* men også tilgjengelig for/i subklassene (og klasser i samme fil/pakke)

Abstrakte klasser

```
abstract class Dyr {
}
```



Klasser som vi ikke ønsker at skal kunne brukes når vi oppretter objekter, men hvor vi må bruke subklassene, kan vi markere som *abstract*

Abstrakte klasser

```
abstract class Dyr {
}
```

De kan likevel fint brukes i referanser, og kan innholde en konstruktør og statiske metoder som vi kan kjøre

Abstrakte klasser

```
abstract class PatteDyr {
}
```

En subklasse kan også være abstract, men det må være minst en subklasse under denne igjen

Den siste klasse

```
final class Labrador extends Hund {
}
```

Hvis vi ikke ønsker at en klasse skal kunne subklasses så kan vi markere den med *final*

Konstante variabler

```
final int PI = 3.142;
```

For å markere at en variabel er konstant, altså ikke skal kunne forandres, så markerer vi variabelen med *final*. Det er vanlig at disse variablene skrives med store bokstaver.

Java, Object og API

Alle klasser i Java arver fra en klasse som heter Object

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html

Java sitt egne API (innebygde klasser og metoder) er bygget opp av subklasser