IN1010

Repetisjon Uke 4

Arv, del 2

(Basert på Eyvind Axelsen og Stein Gjessings presentasjoner)

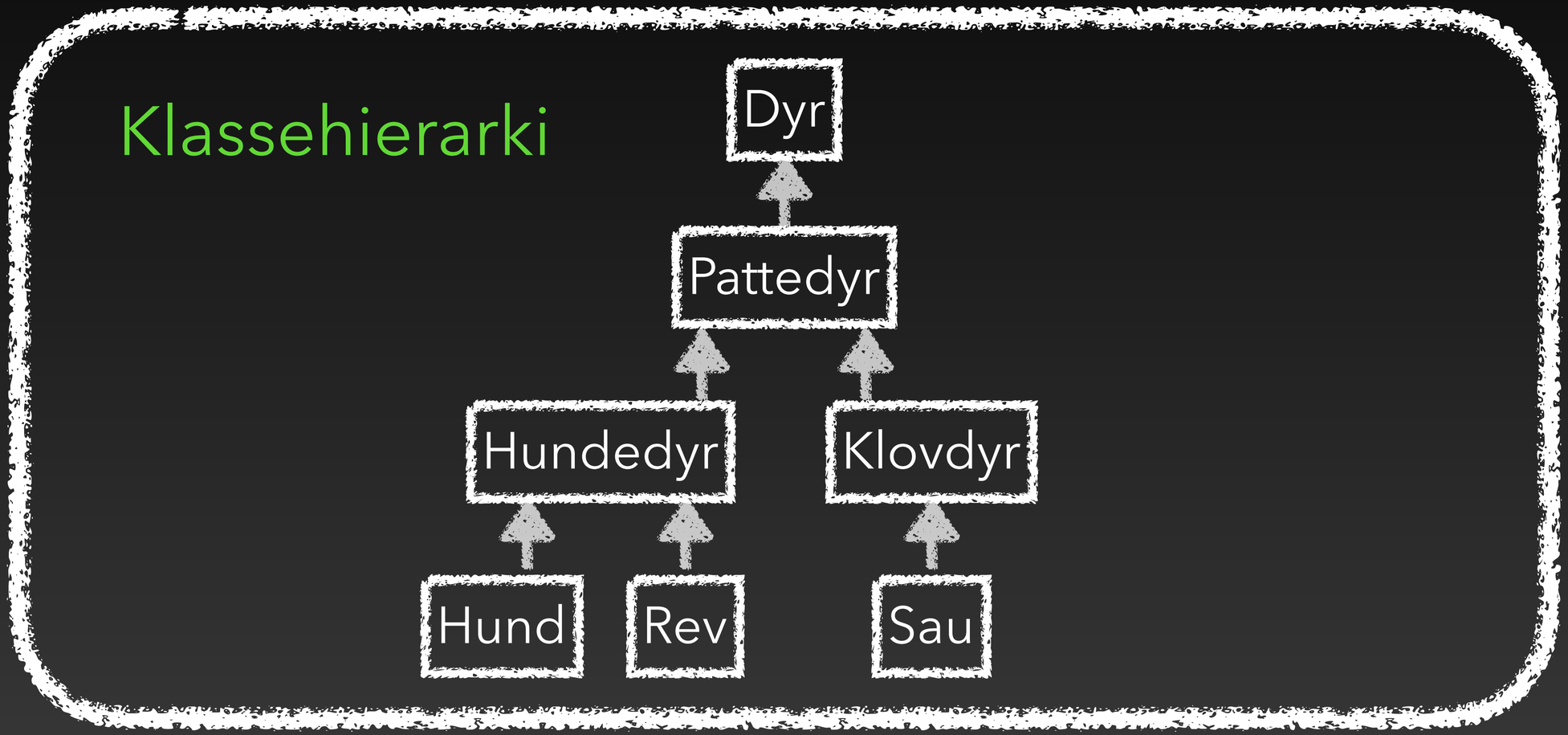
14. februar, 2025

Polymorfi

fra gresk: πολυ + μορφή: *mangeformet*

I objekt-orientert programmering betyr det at samme variabel/referanse kan bestå av flere typer

Fra siste uke



Nytt denne uken

- Override – overstyring
- Overload – overlasting
- Arv og komposisjon
- Mer om Object

Override / overstyring

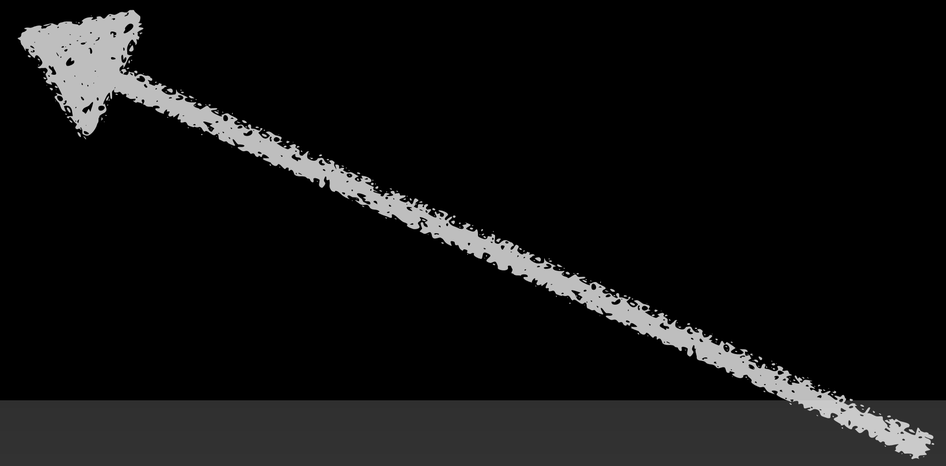
Vi kan lage metoder høyt i klassehierarkiet, som er generelle eller uten egentlig funksjon

Disse kan vi så overstyre lengre ned i hierarkiet, slik at vi kan ha metoder som er mer spesialisert

De må ha samme *signatur*

Signatur

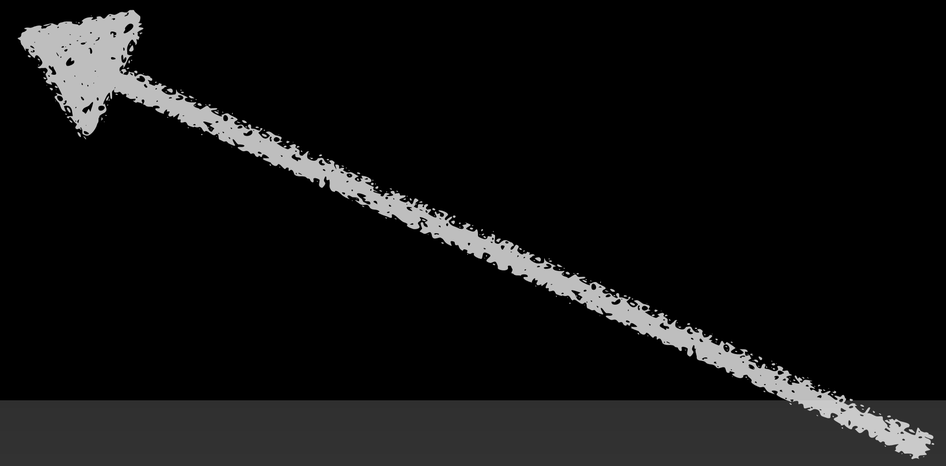
```
void lagerLyd(int varighet) {
   …
}
```



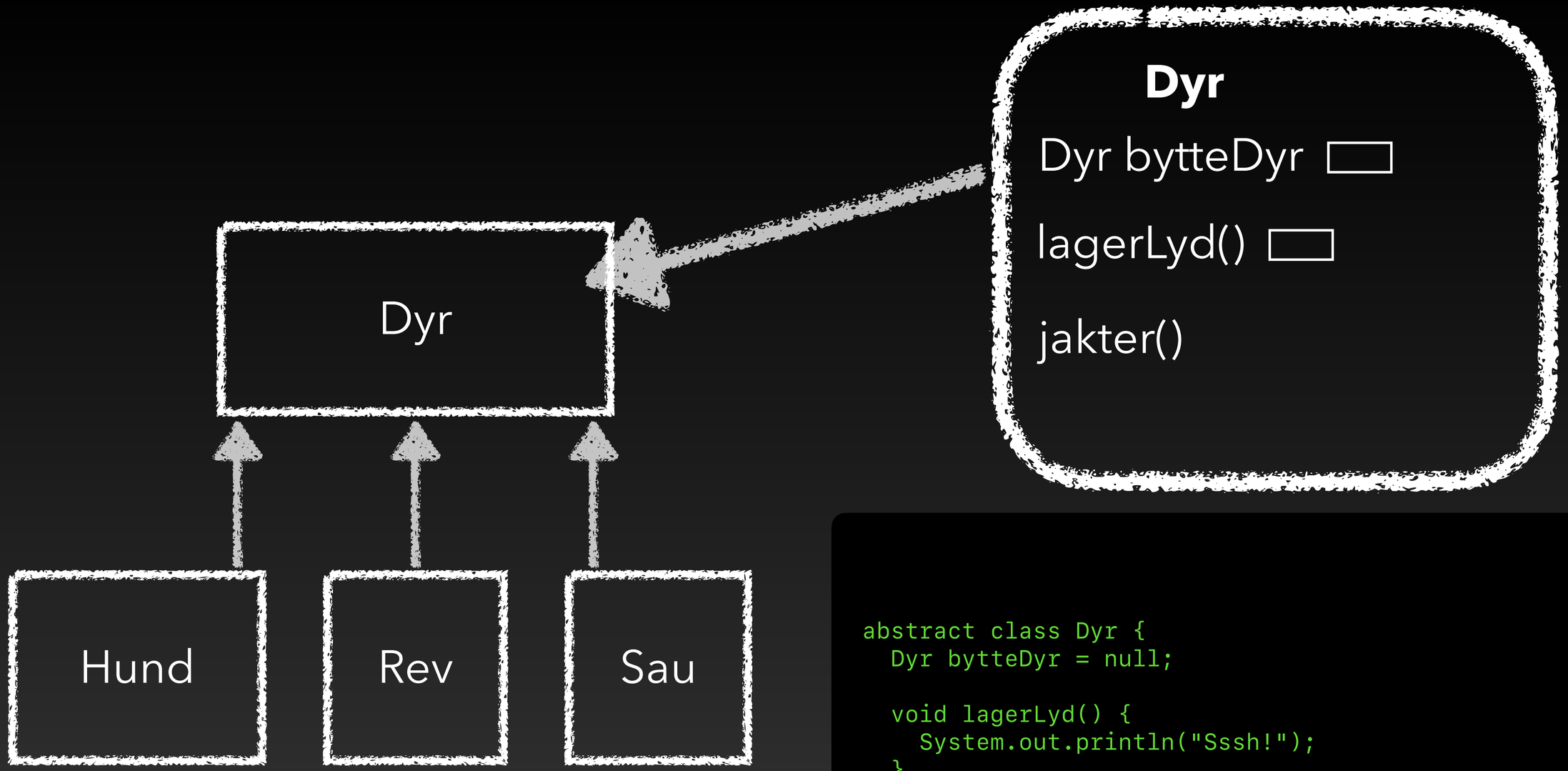
Signaturen er en del av "hodet" til metoden, altså det foran krøllparentesene

Signatur

```
void lagerLyd(int varighet) {
   …
}
```



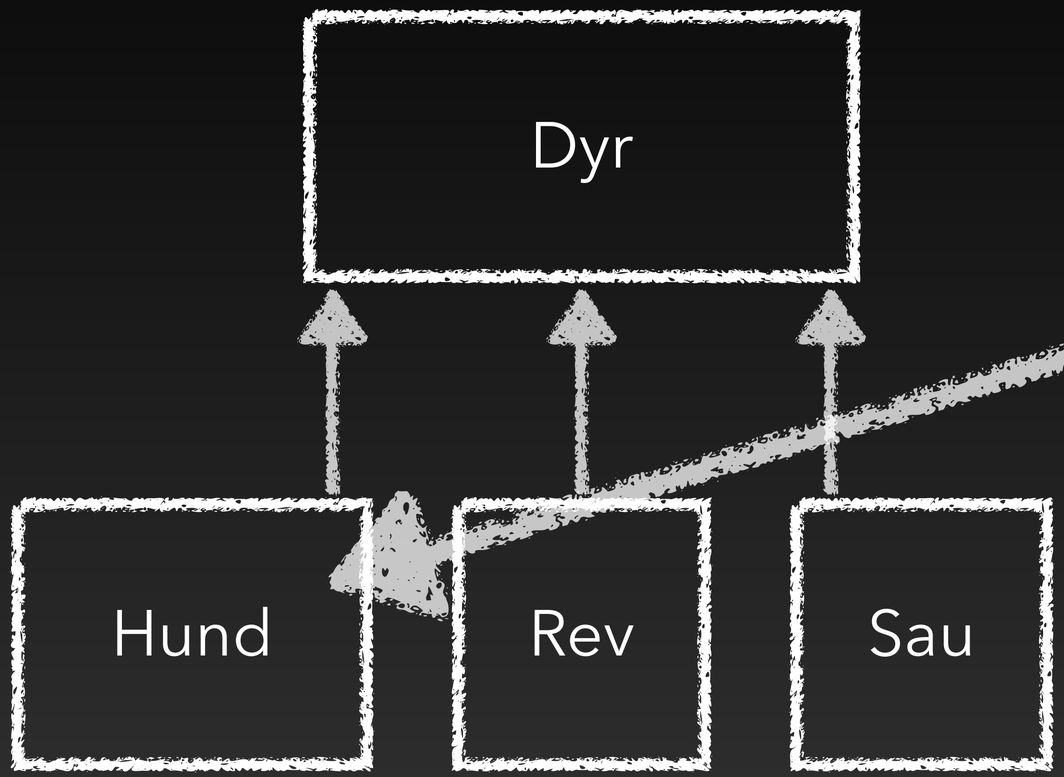
Består av navnet *(lagerLyd)* – og – parametrene *(int varighet)*

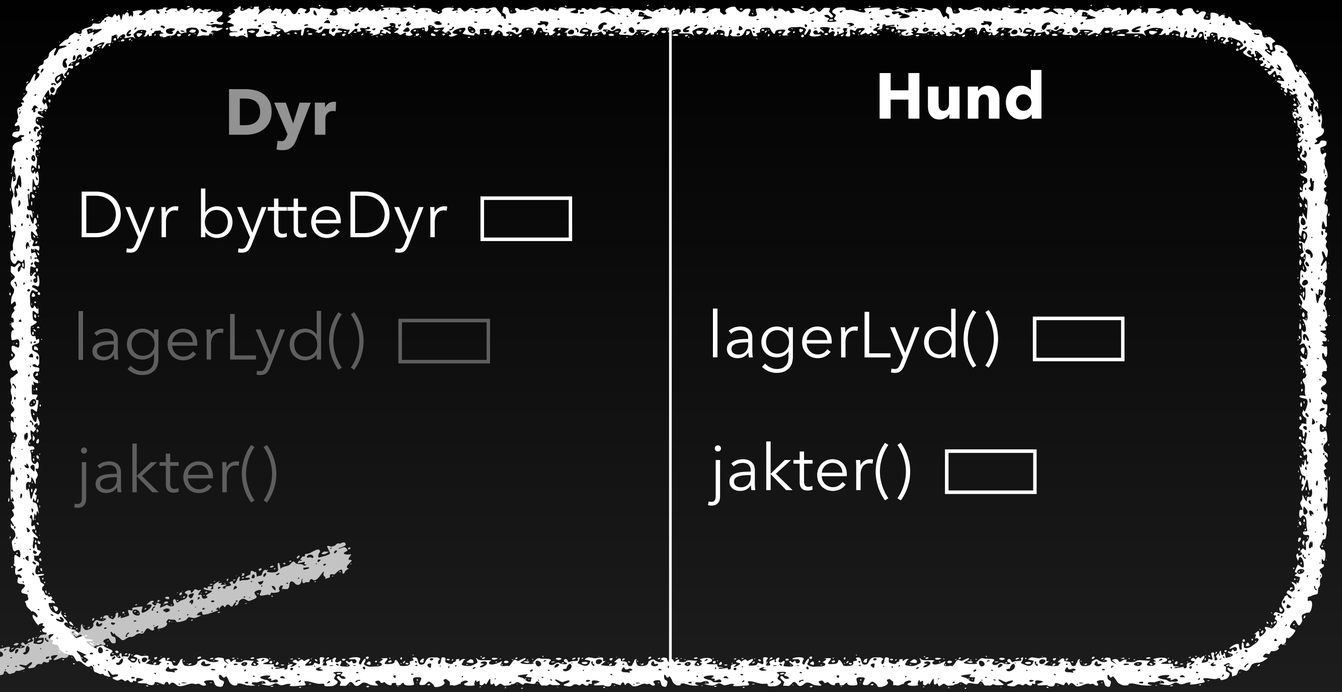



```
abstract class Dyr {
  Dyr bytteDyr = null;

  void lagerLyd() {
    System.out.println("Sssh!");
  }

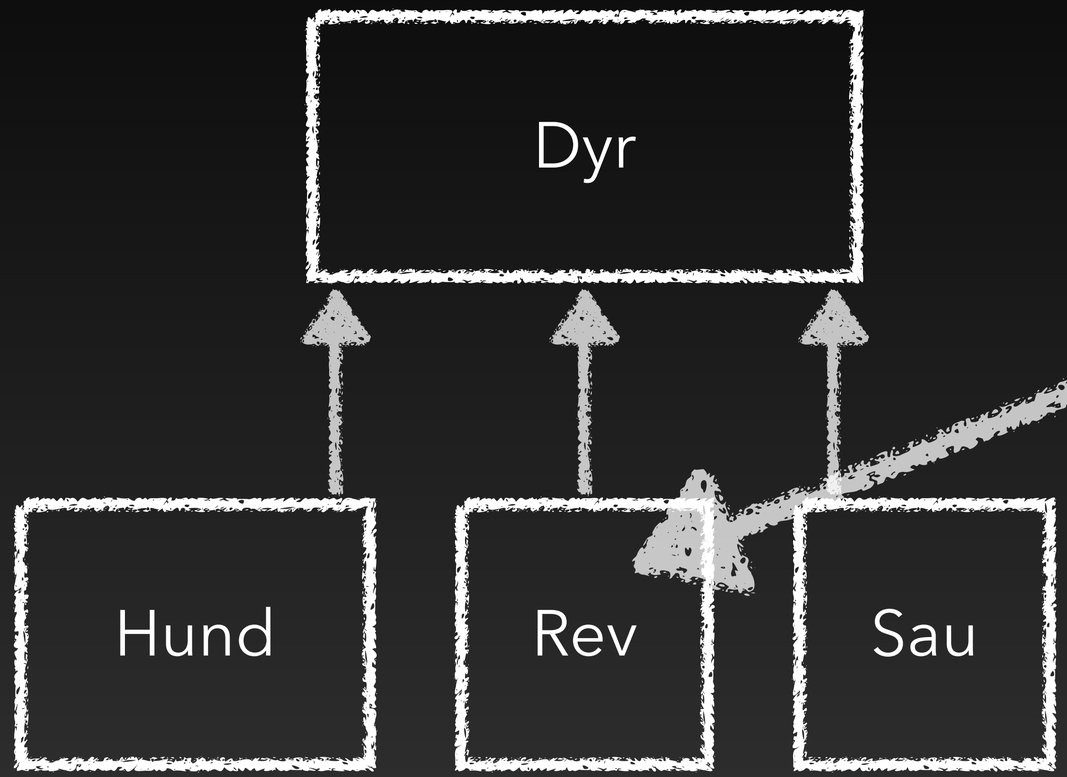
  abstract Dyr jakter(Dyr dyr);
}
```

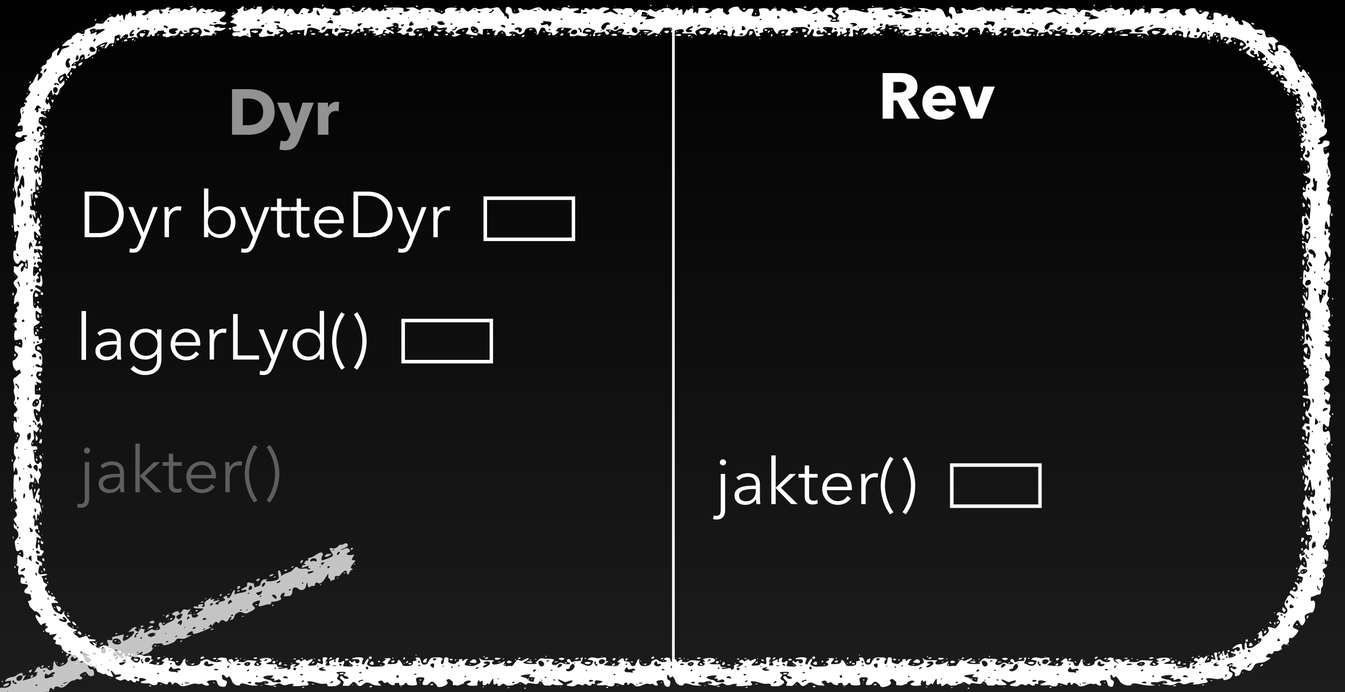




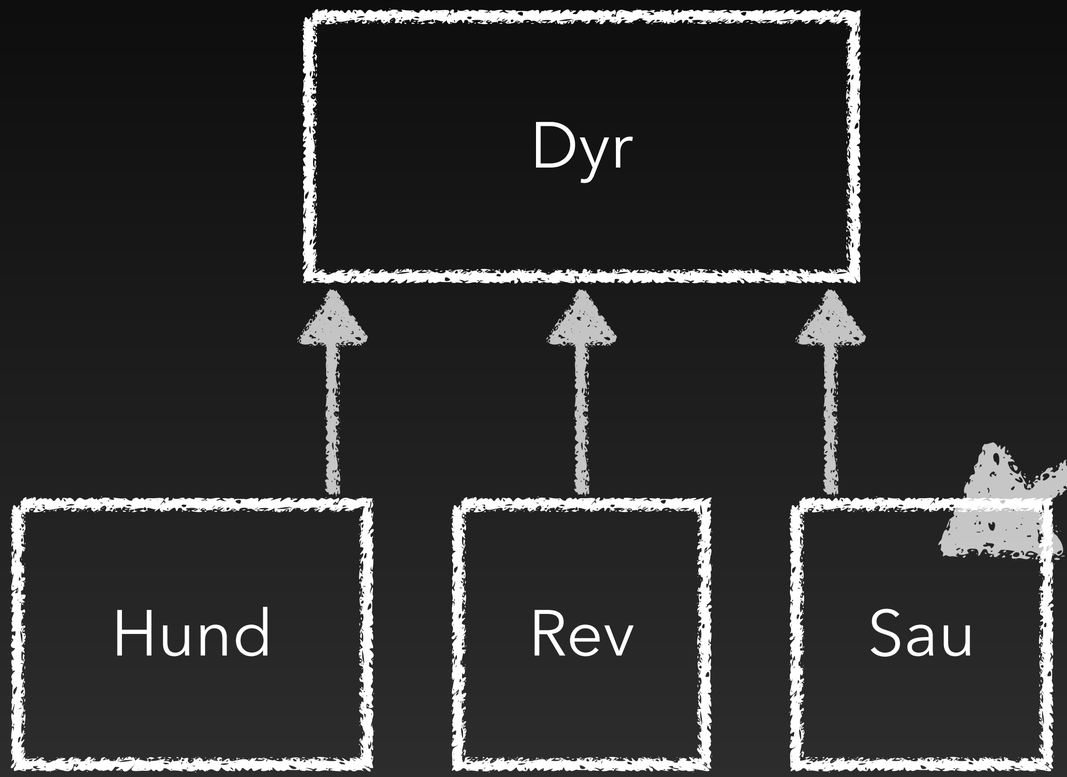

```
class Hund extends Dyr {
  @Override
  void lagerLyd() {
    System.out.println("Voff!");
  }

  @Override
  Dyr jakter(Dyr dyr) {
    if (dyr instanceof Katt) {
      bytteDyr = dyr;
      return dyr;
    }
    return null;
  }
```






```
class Rev extends Dyr {
  @Override
  Dyr jakter(Dyr dyr) {
    if (dyr instanceof Sau) {
      bytteDyr = dyr;
      return dyr;
    }
    return null;
  }
}
```



Dyr

Dyr bytteDyr

lagerLyd()

jakter()

Sau

teller sauNummer

lagerLyd()

jakter()

```
class Sau extends Dyr {
  static int teller = 0;
  private int sauNummer = ++teller;

  @Override
  void lagerLyd() {
    System.out.println("Bææææ!");
  }

  @Override
  Dyr jakter(Dyr dyr) {
    return null;
  }
}
```

Metoder og super

```
class Figur {
  void tegn() {
    System.out.println("Nå skal vi tegne");
  }
}
```

Hvis vi vil kjøre kode både fra den egentlige klassen men også klassen har vi *super*

Metoder og super

```
class Sirkel extends Figur {
  @Override
  void tegn() {
    super.tegn();
    System.out.println("Her er en sirkel ○");
  }
}
```

Hvis vi vil kjøre kode både fra den egentlige klassen men også klassen har vi *super*

Overload / overlaste

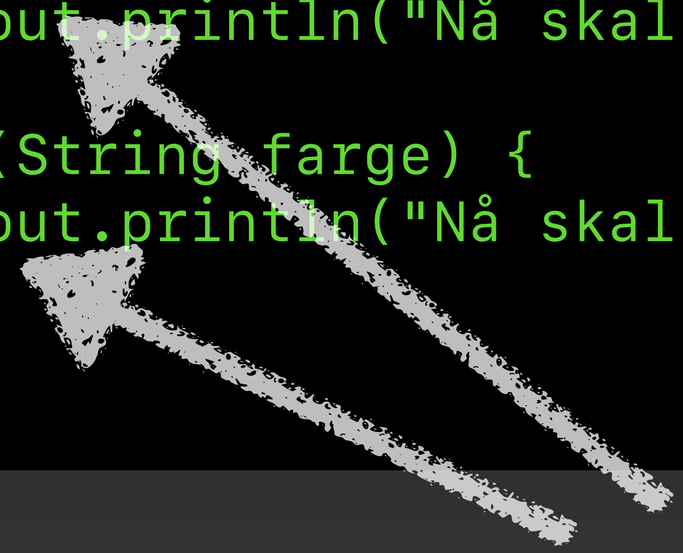
Vi kan lage metoder som har samme navn, men som har ulike parametere.

Det kalles å overloade/overlaste metodene.

Det kan være nyttig når metodene gjør det samme, men vi ønsker f.eks. en variant med og en uten parametere, eller ved bruk av ulike datatyper eller klasser.

Overload

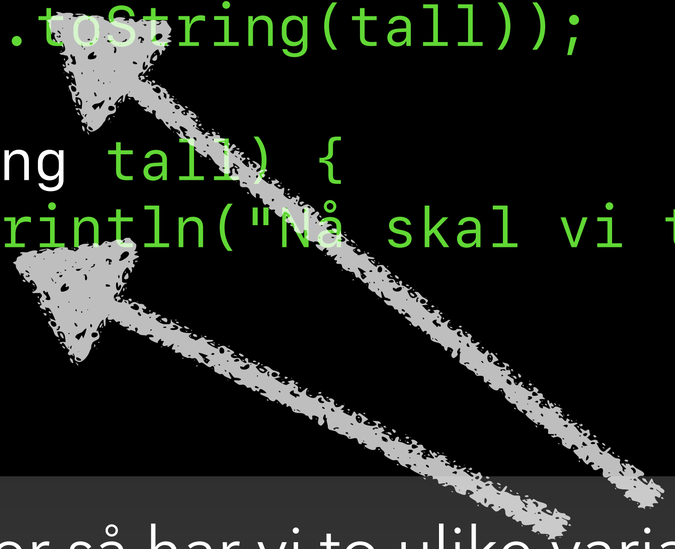
```
class Figur {
  void tegn() {
    System.out.println("Nå skal vi tegne");
  }
  void tegn(String farge) {
    System.out.println("Nå skal vi tegne en " + farge + " figur");
  }
}
```



Her har vi to metoder som heter tegn

Overload

```
class Figur {
  void tegn(int tall) {
    tegn(Integer.toString(tall));
  }
  void tegn(String tall) {
    System.out.println("Nå skal vi tegne tallet " + tall);
  }
}
```



Her så har vi to ulike varianter av parametere.

Overloading er ikke en type polymorfi, siden det er ikke samme metode

Samme variabelnavn

Vi kan bare override metoder, ikke instansvariabler

Hvis du forsøker å opprette samme instansvariabel i både super- og subklassene, så vil det være ulike avhengig av referansens type. Det vil forvirre.

Unngå det, opprett instansvariablene kun i superklassen om du vil ha tilgang i begge

Komposisjon eller arv

Komposisjon: objektet **har et** annet objekt

Arv: objekt **er en** type av annet objekt

Eksempler

Dyrehagen har en løve (komposisjon: Dyrehage og Løve)

Løven er en stor katt (arv: Løve og Katt)

Mer om Object-klassen

Alle klasser i Java arver fra en klasse som heter Object

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

Noen av de viktigste metodene er

equals(), hashCode() og toString()

equals()

```
Hest hest_a = new Hest();
Hest hest_b = new Hest();
if hest_a.equals(hest_b) {
  System.out.println("Hestene er like!");
}
```

Equals() har vi allerede vært borti flere ganger, og sammenligner to objekter for å se om de er like

equals()

```
class Hest {
  String farge;
  @Override
  public boolean equals(Object obj) {
    if (obj instanceof Hest) {
      if obj.farge.equals(farge) return true;
    }
    return false;
  }
}
```

Vi kan override equals() slik at vi kan bestemme hva som gjør et objekt lik

F.eks. hvis hestene har samme farge, så kan vi si at de er like

toString()

```
Hest hest = new Hest();
System.out.println(hest); // f.eks. Hest@511d50c0
```

toString() brukes av bl.a. System.out.println() for å skrive en representasjon av objektet

Hvis den ikke overrides så består det av klasse-navnet og hash-verdien til objektet

toString()

```
class Hest {
  String farge;
  @Override
  public String toString() {
    return "En " + farge + " hest";
  }
}
```

Det kan være hjelpsomt å gi mer leselig versjoner slik at det er lettere å finne ut om objektet ditt

Mer om Object-klassen

hashcode() – Gir deg en hash-verdi av objektet

getClass() – Gir deg en referanse til Class-klassen (som er subklassen til Object)

Class-klassen har noen nyttige metoder:

getName() – navnet på klassen som en String

Litt mer om konstruktøren

Ved arv så kalles først klassen vi vil instantiere, og så kalles konstruktørene til super-klassene, helt til toppen

Deretter opprettes objektet fra super-klassen ned til instans-klassen

Mangler konstruktøren så prøver Java å lage en for deg, det inkluderer også kall på super

Hvis antall parametere er ulike, så må det gjøres manuelt for å bli riktig

Kall på super() i konstruktøren må være på første linje

Litt mer om konstruktøren

```
class Figur {
  private int lengde;
  Figur() {
    this.lengde = 10;
  }
  Figur(int lengde) {
    this.lengde = lengde;
  }
}
```

Vi kan ha flere konstruktører med ulike parametere