# IN1010

## Repetisjon Uke 5

## Arv, del 3

(Basert på Eyvind Axelsen og Stein Gjessings presentasjoner)

21. februar, 2025

# Grensesnitt / Interface

Definerer en implementasjon av

en rolle, oppførsel, tjenste, kontrakt

Til nå så har vi arvet nedover fra én klasse over, men hvis vi trenger noe som går på tvers av (ulike) klasser, så kan vi bruke *interface*

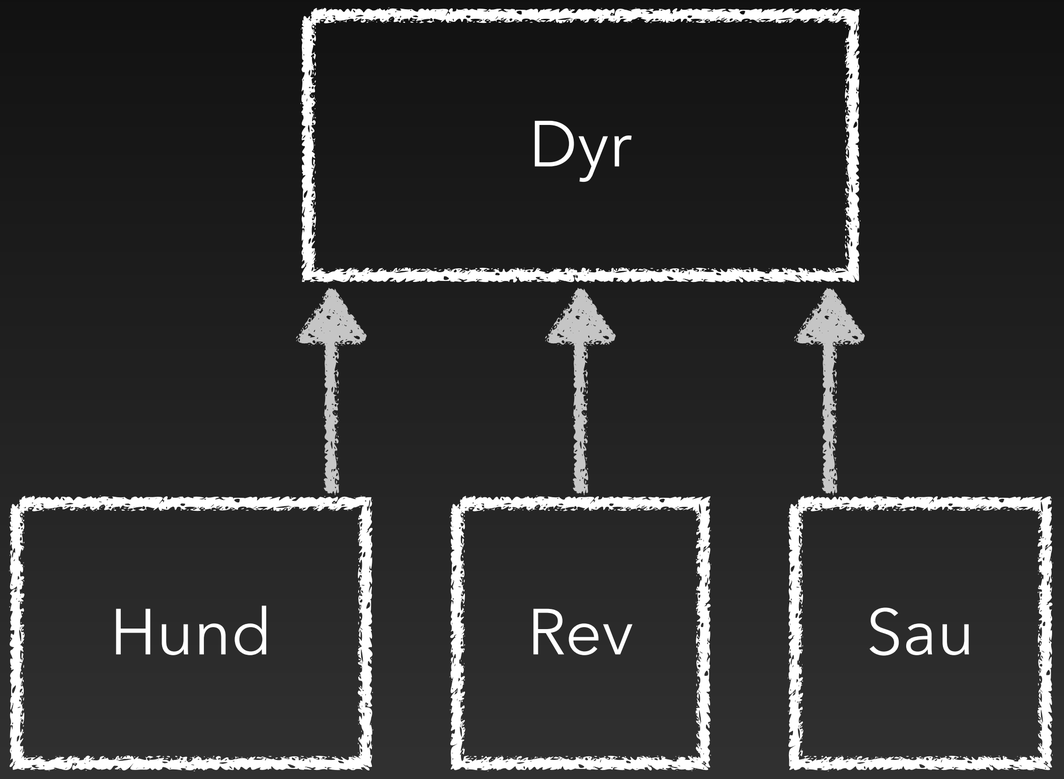
Alt vi trenger er en egenskap som er felles

# Grensesnitt / Interface

Ved vanlig arv hvor vi bruker *extends*, så kan vi bare arve fra én klasse over

Med *interface* kan vi legge til så mange vi ønsker

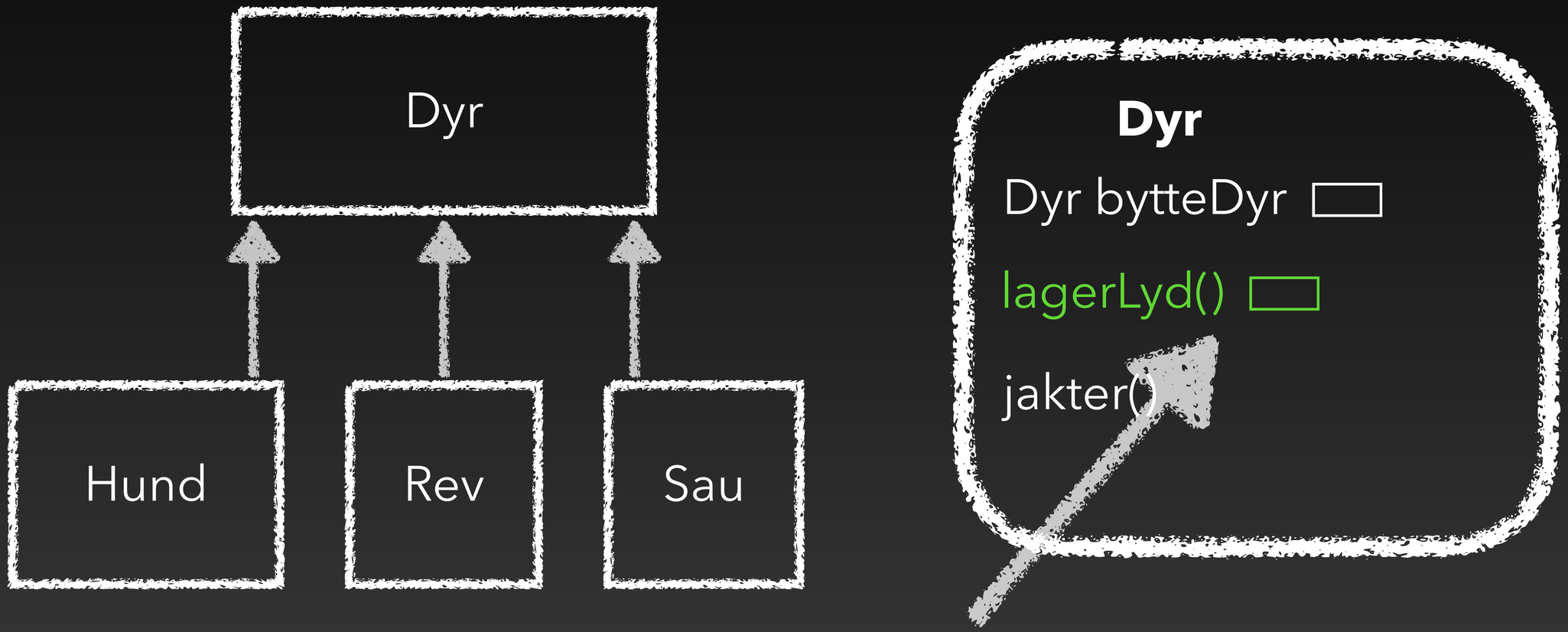
# Sist gang:



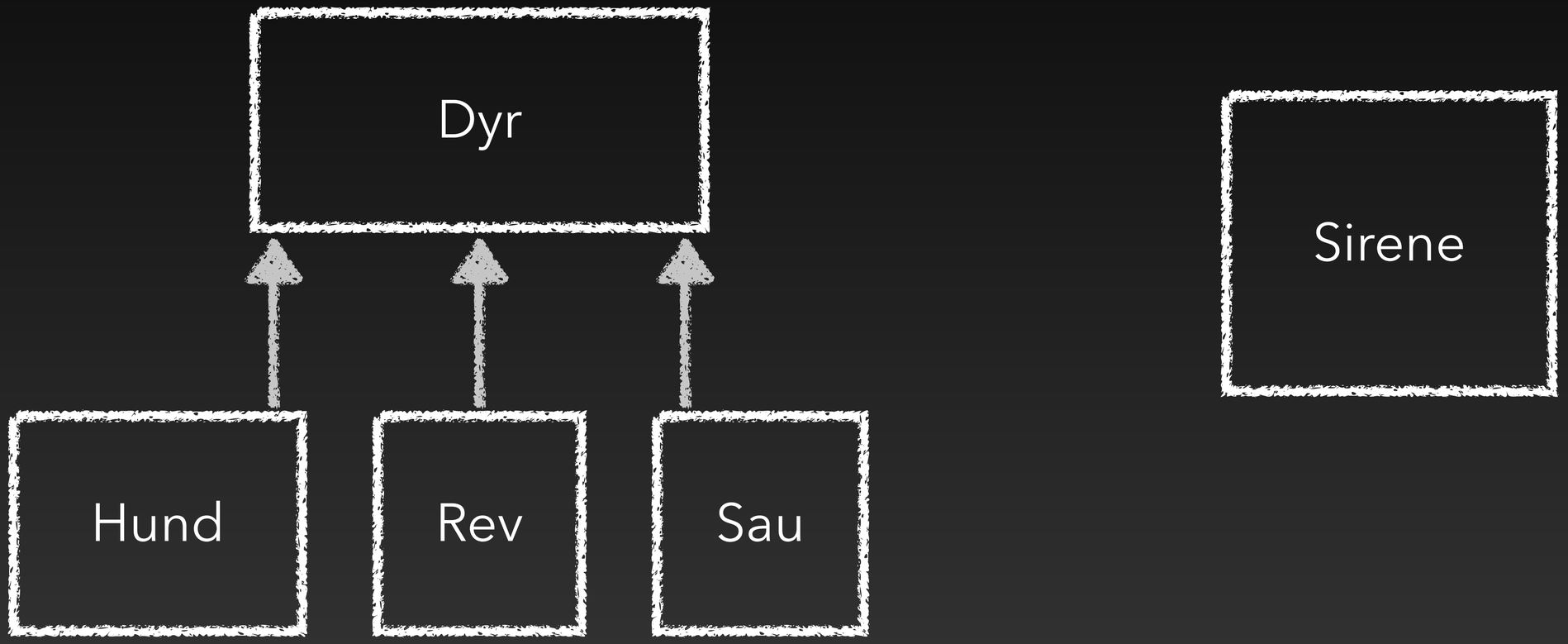




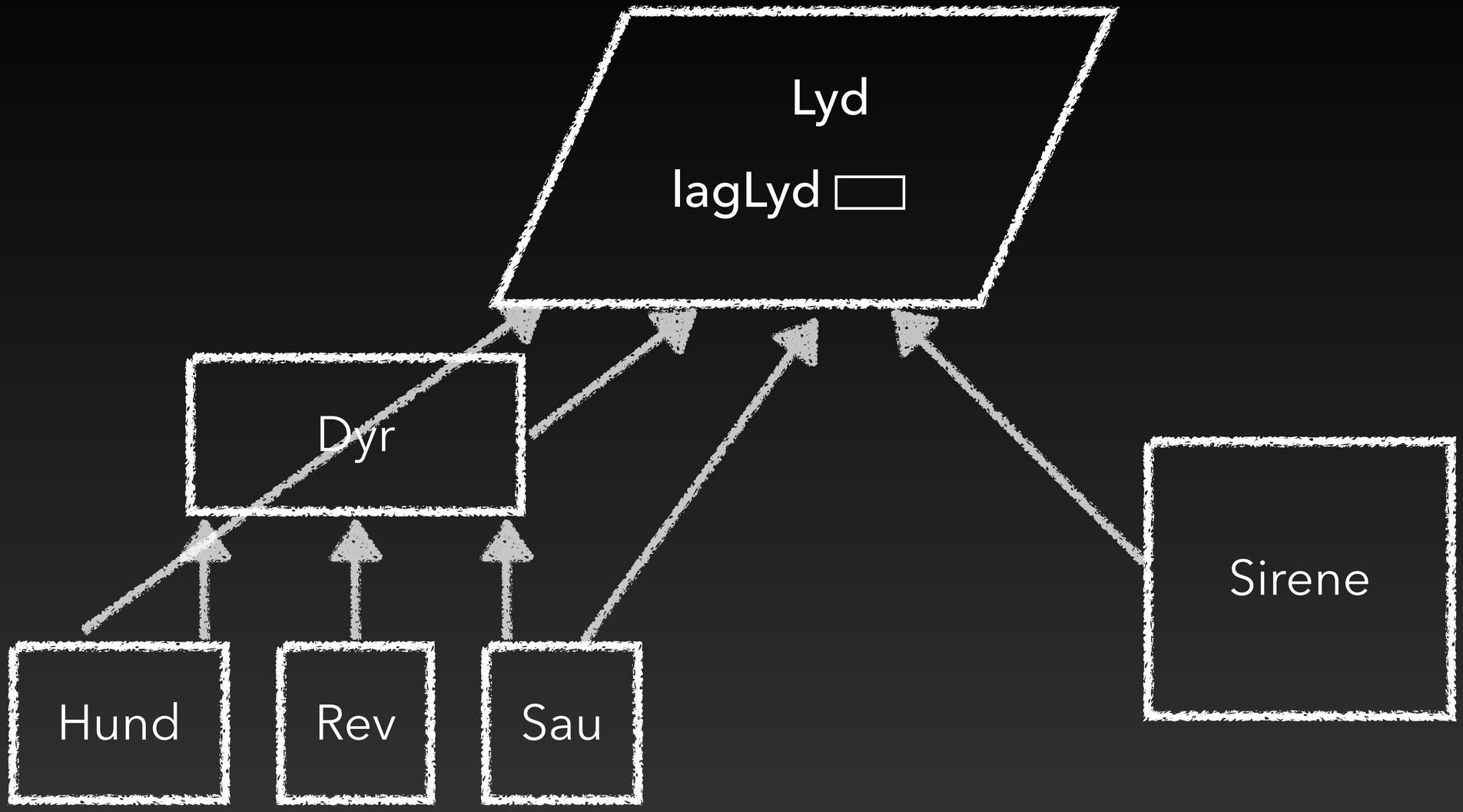
Hva hvis vi ønsker å registrere lyd fra ulike kilder, ikke bare dyrene?




Da kan vi lage et interface for lyd, og vi skiller ut metoden til det



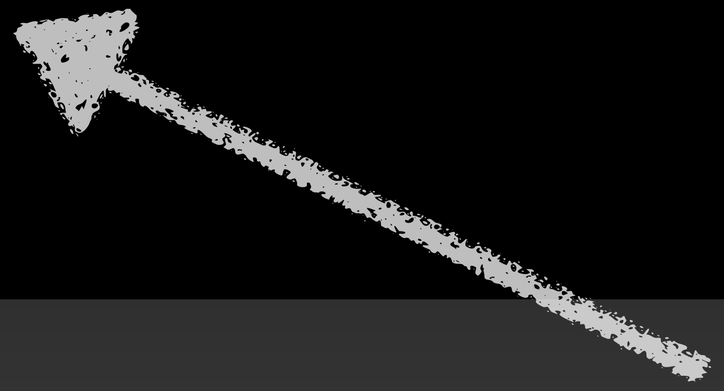

La oss si vi også har klassen Sirene




Vi lager da et interface Lyd som inneholder signaturen til metoden vi vil ha i alle klassene

# Interface

```
interface Lyd {
  String lagerLyd();
}
```



I et *interface* så legger vi bare inn metode-signaturen med retur-typen

# Interface

```
class Hund extends Dyr implements Lyd {
  @Override
  String lagerLyd() {
    return "Voff";
  }
}
```

I klassen vi ønsker å implementere metoden, så må vi legge til litt ekstra

# Interface

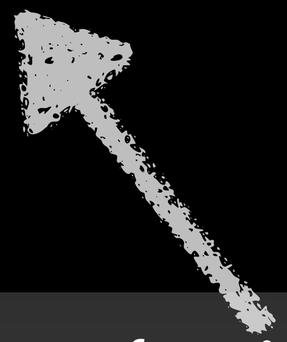
```
class Hund extends Dyr implements Lyd {
  @Override
  String lagerLyd() {
    return "Voff";
  }
}
```



I klasse-definisjonen så legger vi til at vi ønsker å legge til et *interface*

# Interface

```
class Hund extends Dyr implements Lyd {
  @Override
  String lagerLyd() {
    return "Voff";
  }
}
```



På samme måte som før så "overrider" vi metoden fra interfacet

Hvis vi ikke gjør det eller glemmer det, så vil vi få feil under kompilering

# Interface

```
class Sirene implements Lyd {
  @Override
  String lagerLyd() {
    return "Uuuuuul";
  }
}
```



Så gjør vi det samme i alle klassene som implementerer interfacet

# Interface

```
class Hund extends Dyr implements Lyd, Levende {
  @Override
  String lagerLyd() {
    return "Voff";
  }
  @Override
  boolean erLevende() {
    return true;
  }
```



Hvis vi ønsker det så kan vi bruke flere interface i samme klasse

# Interface

```
Lyd lyd_hund = new Hund("Labben");
Lyd lyd_sirene = new Sirene();
System.out.println(lyd_hund.lagerLyd());
System.out.println(lyd_sirene.lagerLyd());
```

Det som gjør interface ekstra nyttig er at vi kan bruke de som referanse-typer

De kan da referere til hvilket som helst objekt som implementerer interfacet

# Grensesnitt / Interface

- Inneholder bare metode-signaturen
- Kan brukes som referanse/type/peker (polymorfi)
- Du kan bruke mange interface i en klasse
- Defineres som en rolle, tjeneste eller kontrakt
- Nyttig for inkapsling (abstraksjon, skjule detaljer)
- Illustreres som et parallelogram i klassehierarkiet

# Comparable

```
class By implements Comparable<By> {
  int størrelse;
  @Override
  public int compareTo(By annen) {
    return this.størrelse.compareTo(annen.størrelse);
  }
}
```

Mange av Java sine avanserte metoder er implementert som interface, og et eksempel er Comparable som brukes til å sammenligne og sortere