# Arv, subklasser og interface

IN1010 - Repetisjon

Tobias og Ahmed

Vår 2022

# Arv

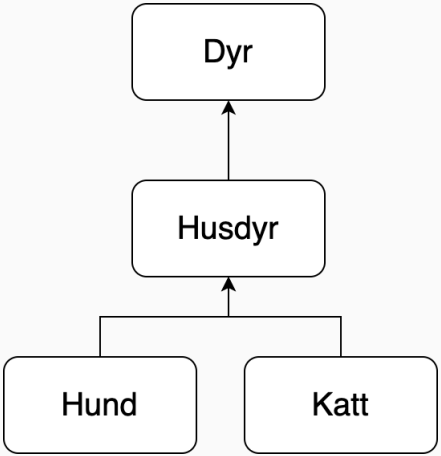
- Klasser i objektorientert programmering representerer noe som deler et sett med egenskaper
- En **subklasse** har egenskapene til en klasse + noen nye mer sesialiserte egenskaper
- Subklassen **arver** egenskapene til **superklassen**

## Hvorfor subklasser?

- Ønsker å modellere virkeligheten
- Gir struktur til systemet vi lager
- Gjenbruk

# Klassehierarki

- Kan ha flere nivåer med arv
- Alle hunder er både dyr og husdyr
- Alle katter er både dyr og husdyr
- Ingen hunder er katter
- Ingen katter er hunder



# Nøkkelord - arv

```
class Hund extends Husdyr{...}
```

- extends: for å lage en subklasse

```
protected int alder;
```

- protected: alle av klassens subklasser kan se egenskapen

```
abstract class Dyr {...}
```

- abstract: hvis en klasse er abstrakt kan man ikke opprette en instans av denne klassen, men subklassene arver egenskaper

```
public abstract void spis();
```

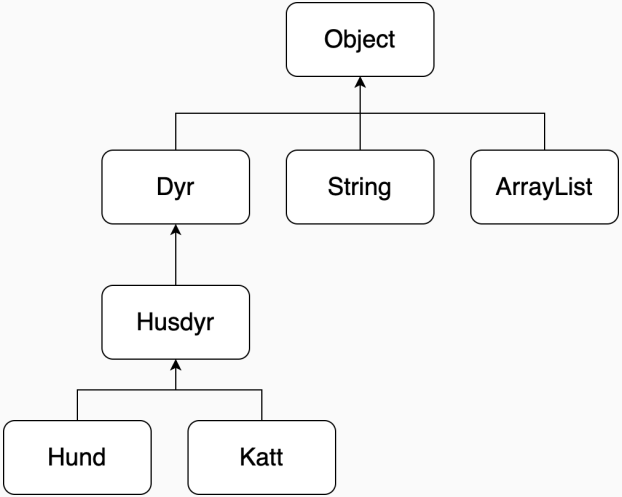
- i en abstrakt klasse så kan man også ha abstrakte metoder som man ikke trenger å implementere
- alle subklassene må da implementere metoden

# super i konstruktør

- `super(variabel);`, i konstruktøren til en subklasse brukes super for å sende verdier opp til superklassens konstruktør
- Et kall på super må legges først i konstruktøren
- Hvis man ikke kaller på super så legger java inn et tomt kall på super når programmet kompileres: `super();`
- Hvis en klasse ikke har konstruktør så legger også java inn et tomt kall på super
- Man må derfor ha med super hvis konstruktøren i superklassen tar inn noen verdier (**Ikke redefiner** superklassens variabler i subklassens konstruktør)

# Klassen Object

- Alle klasser arver fra klassen Object (automatisk)
- Metoder som clone(), equals() og toString()
  - Derfor man kan kalle på toString() før den er implementert



# Referanser

```
class A {}

class B extends A {}
```

Variabel av type A kan referere til objekt av B, men ikke omvendt

Kan derfor skrive:

- `A var1 = new A();`
- `A var2 = new B();`
- `B var4 = (B) var2;`

men ikke:

- `B var5 = new A();`
- `B var6 = (B) new A();`

# Polymorfi

- Spesialisering
- I stedet for å bruke **`instanceof`**
- Hvis en metode ikke er **`final`** kan metoden overskrives (`@Override`)
- Ulike subklasser kan da ha den samme metode-signaturen, men forskjellige implementasjon
- Hvis man ønsker å bruke superklassen sin implementasjon kan man skrive: **`super`**`.metodenavn();`

```java
  // I klassen Hund
@Override
public void lagLyd() {
    System.out.println("Voff");
}
// I klassen Katt
@Override
public void lagLyd() {
    System.out.println("Mjau");
}
```

# Interface

- Interface sikrer at en klasse har implementert noen metoder
- Kun abstrakte metoder, ligner på en abstrakt klasse uten implementasjoner og variabler (Kan ha konstanter)
- I java kan en klasse kun arve fra én annen klasse, men fra flere interfaces
- En klasse deler muligens også egenskaper med andre klasser som ikke har samme superklasse, kan da bruke interface
- Eksempler: `List`, `Comparable`, `Iterable`

# Interface

```java
interface Example {
    public void method1();
}

class A {}
```

```java
class B extends A implements Example {

    @Override
    public void method1() {
        System.out.println("Ex.");
    }
}
```

- Kan bruke interface som typen til en peker
- `Example var1 = new B();`
- Har da kun tilgang til de metodene deklarert i interfacet

# Oppgave

Du har blitt ansatt av en møbelforhandler for å modellere ulike typer møbler. I dette programmet bryr vi oss kun om senger og sofaer. Alle senger er enten **Enkelseng**, **Vannseng** eller **Dobbeltseng**. For **Sofa** har vi vanlig sofa (klassen skal da hete **Sofa**), **Vannsofa** og **Hjørnesofa**. I og med at både **Vannseng** og **Vannsofa** inneholder vann, så skal de også implementere interfacet **Vannholdig** som har metoden `hentVanninnhold`.

Tegn klasssehierarkiet og implementer klassene beskrevet over.