Uke 3

Arv og subklasser del 2

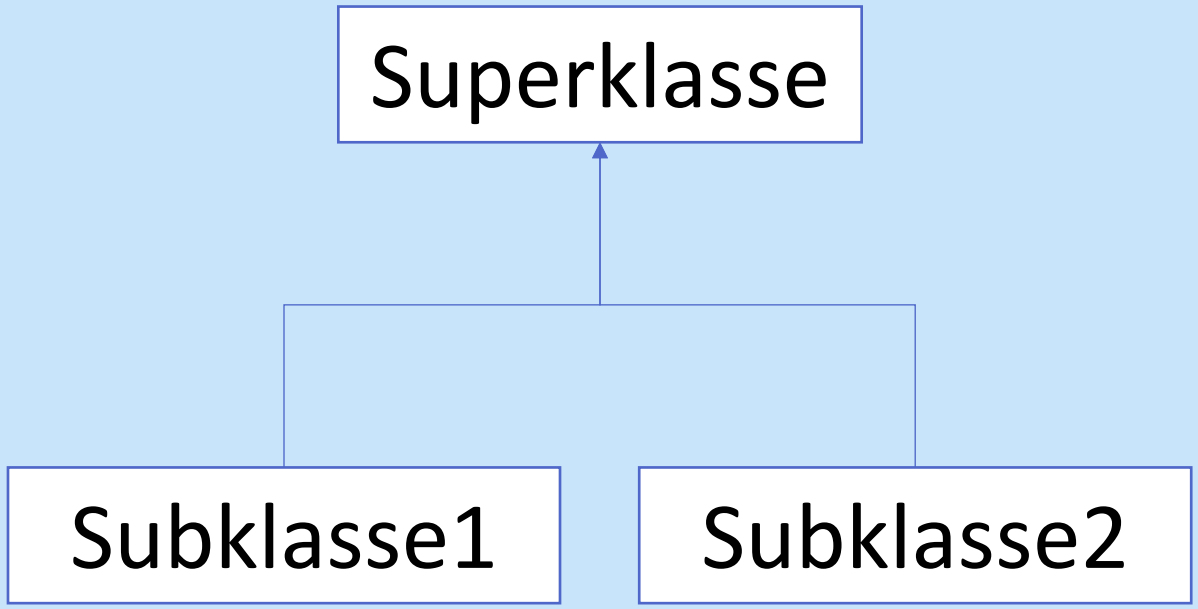
I dag

- Litt repetisjon
- Arv og komposisjon – når skal hva brukes
- Polymorfi
- Forksjell på == og .equals()

Repetisjon

Superklasse og subklasse

- En subklasse arver fra en superklasse
- En subklasse er en spesialisering av superklassen den arver fra
- Subklassen arver egenskaper fra superklassen, og kan i tillegg legge på mer
- Bruker nøkkelorder **extends**
- Eksempel: Dyr (superklasse) og Hund (subklasse av Dyr)
- **class** Hund **extends** Dyr {...}



Abstrakte klasser

- Kan ikke implementeres
- Subklasser kan arve fra en abstrakt klasse
- Eksempler på klasser som passer til å være abstrakte:
  - Dyr
    - Eks subklasser: Hund, Katt, Kanin, Hest
  - Rom
    - Eks subklasser: Soverom, Stue, Kjøkken, bad
  - Rett
    - Eks subklasser: forrett, hovedrett, desser

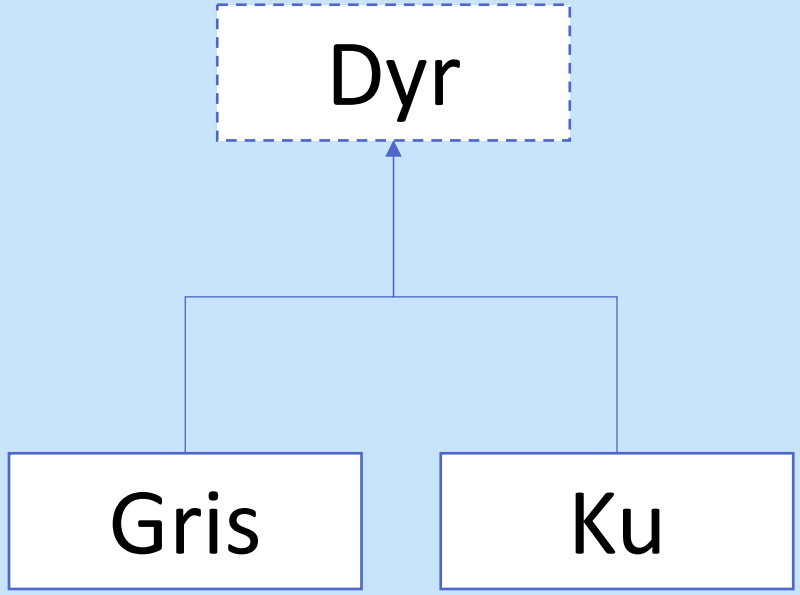
```
public abstract class Dyr {
```

instanceof

- Brukes for å sjekke om et objekt er av en type/klasse
- Hund hund = **new** Hund("Pluto")
- hund **instanceof** Hund -> ?
- hund **instanceof** Dyr -> ?

Klassehierarki for dagens eksempel

- **Dyr er en abstrakt superklasse**
  - Navn
  - void lagLyd()
- **Gris og Ku er subklasser av Dyr**
  - Arver fra Dyr
  - Har dermed også navn
  - Har også lagLyd()



Arv og komposisjon

Arv vs. Komposisjon

- Komposisjon
  - Lager pekere til objekter av eksisterende klasser.
  - Lager objekter og kaller på metoder av disse klassene
- Arv
  - Oppretter en ny (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
- Kan dermed ha 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");
    }
```

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");
    }
```

```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);
    }
```

En Gård består av
kuer og griser

InstanceOf

Bruk av instanceof

- 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

```java
//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();
        }
    }
}
```

```java
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
```

Polymorfi

Metoder med samme signatur på tvers av superklasser og subklasser får ulik oppførsel/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

```java
public class Dyr {

    String navn;

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

    void spis(){
        System.out.println(x:"Jeg spiser.");
    }
}
```

```java
public class Katt extends Dyr{

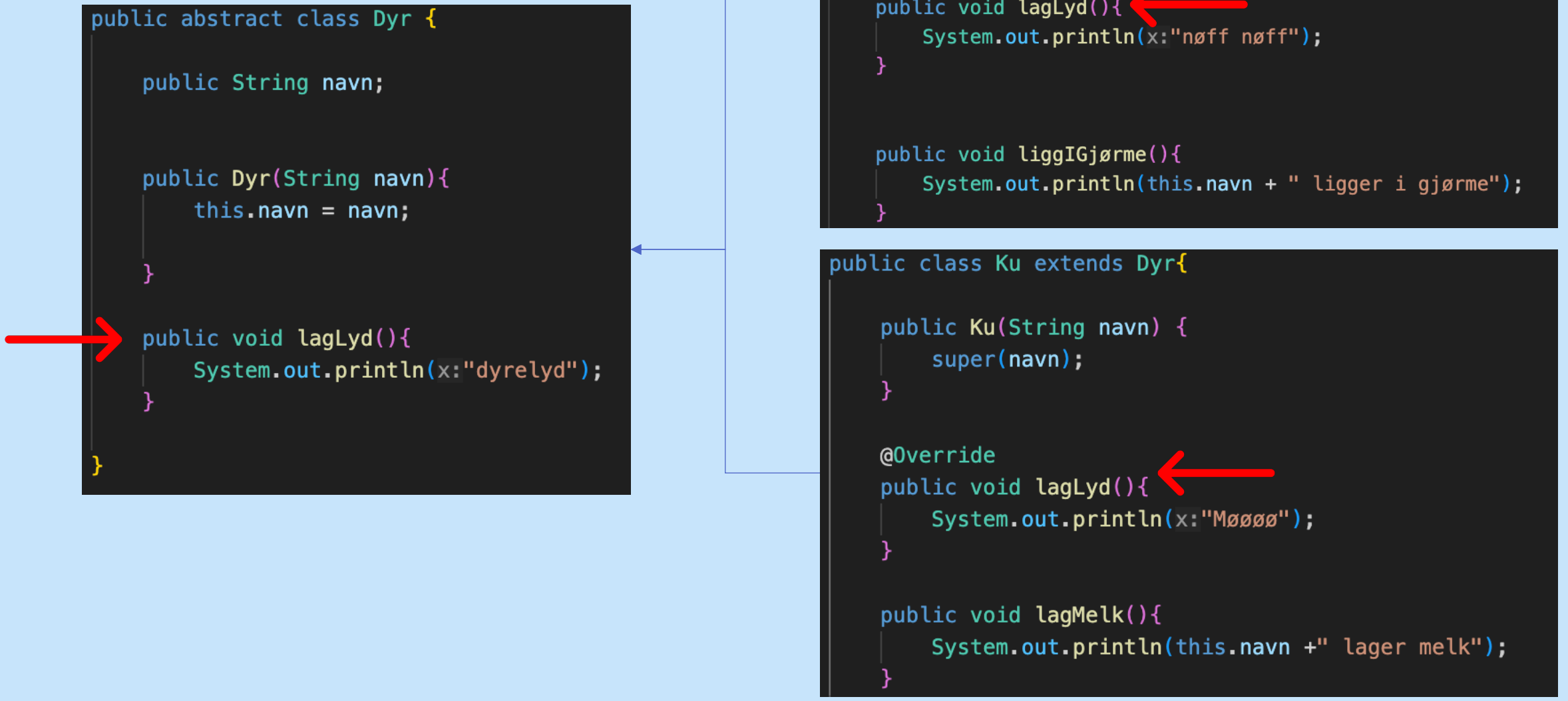
    Katt(String navn) {
        super(navn);
    }

    @Override
    void spis() {
        System.out.println(x:"jeg spiser kattemat");
    }
```

Eksempel fra forrige gruppetime brukte polymorfi

Polymorfi forts.

- Brukes **@Override** for redefinering av metoder i subklassene
- 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



```java
public abstract class Dyr {

    public String navn;


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

    }

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

}
```

```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 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");
    }
```

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

Noen metoder i Object

- Alle klasser arver fra Object, og har dermed tilgang til metoer i Object.
- String toString()
  - returnerer en String-representasjon av objektet
- boolean equals(Object o)
  - sjekker om to objekter er like (i Object det samme som pekerlikhet)
- Kan redefinere disse med @Override

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

```java
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);
}
```

Oppgaveløsing🌼

- Ukesoppgaver
- Seminaroppgaver på gruppe 3 på emnesiden