IN1010 - Gruppe 11

Repetisjon

Agenda

- Arv og subklasser
- Beholdere
- Oppsummering
- Midtveiseksamen

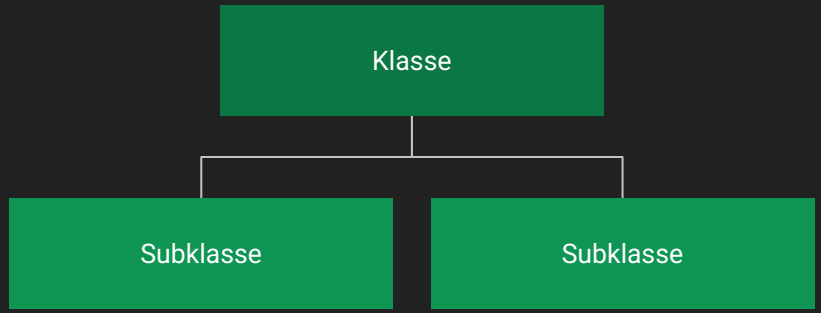


Noen spørsmål?

Arv og subklasser

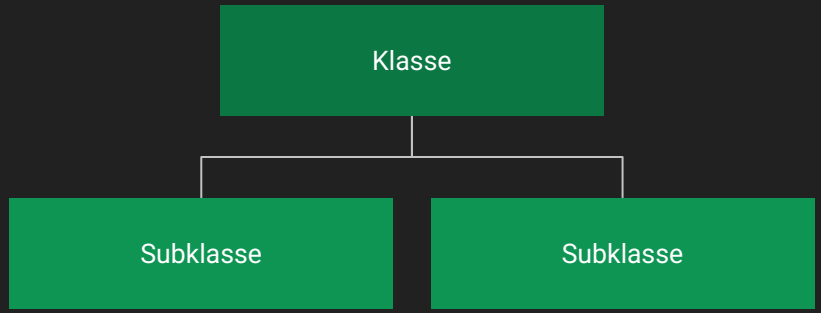
Arv og subklasser 1

- **Klasse**: beskriver objekter med visse egenskaper
- **Subklasse**: beskriver objekter med samme egenskaper som beskrevet av Klasse (superklassen)
  - Men har i tillegg noe mer / spesielle egenskaper



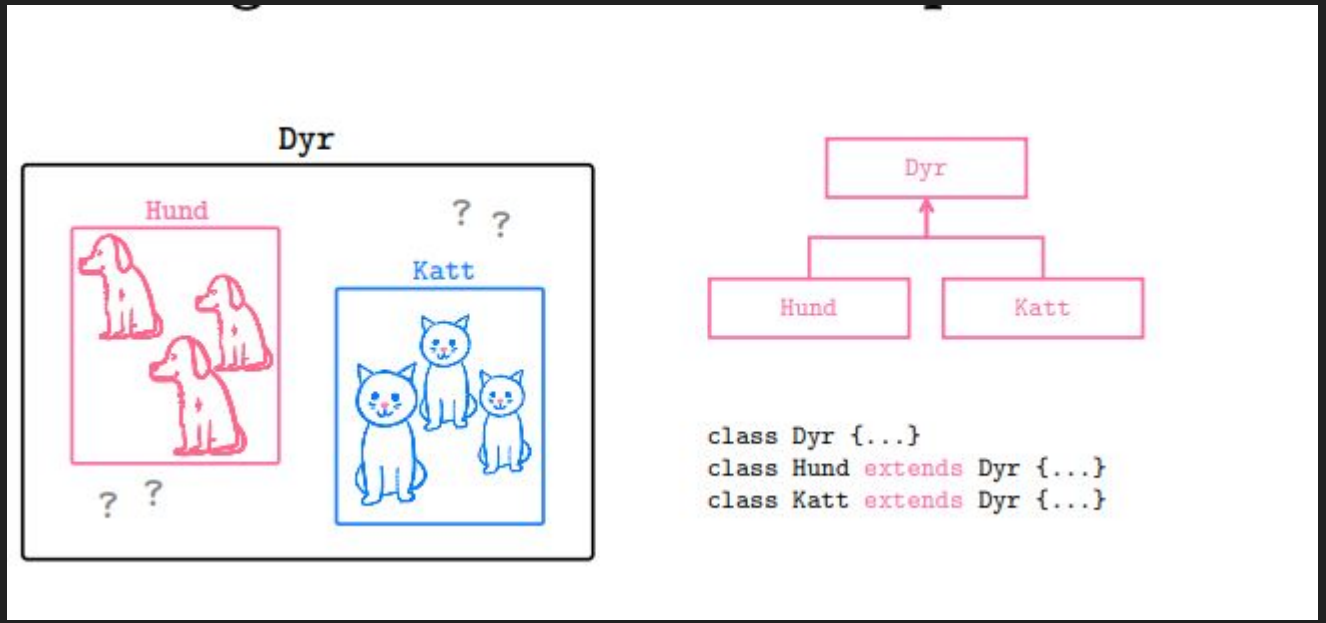
Arv og subklasser 1

- **Klasse**: beskriver objekter med visse egenskaper
- **Subklasse**: beskriver objekter med samme egenskaper som beskrevet av Klasse (superklassen)
  - Men har i tillegg noe mer / spesielle egenskaper
- Nøkkelord: **extends**
  - Arve fra en klasse
- Nøkkelord: **super**
  - Aksessere variabler / metoder i superklassen




```
class Klasse { … }
class Subklasse extends Klasse {...}
```

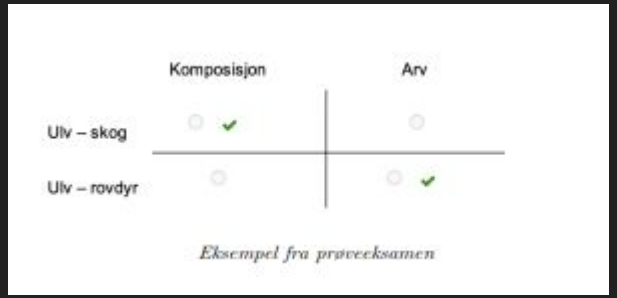
Arv og subklasser 1 eksempel



Arv og subklasser 2

- Arv vs. komposisjon
  - Arv: en klasse utvider en annen
  - Komposisjon: refererer til objekter av andre klasser

- Eksempel
  - Ulv er en type rovdyr
    - men IKKE en type skog
  - Ulv har felles egenskaper med andre rovdyr
    - men ikke samme egenskaper som en skog



Arv og subklasser 2

- Polymorfi = “flere former”
  - Overskriver (@Override)
    - samme metode har flere former

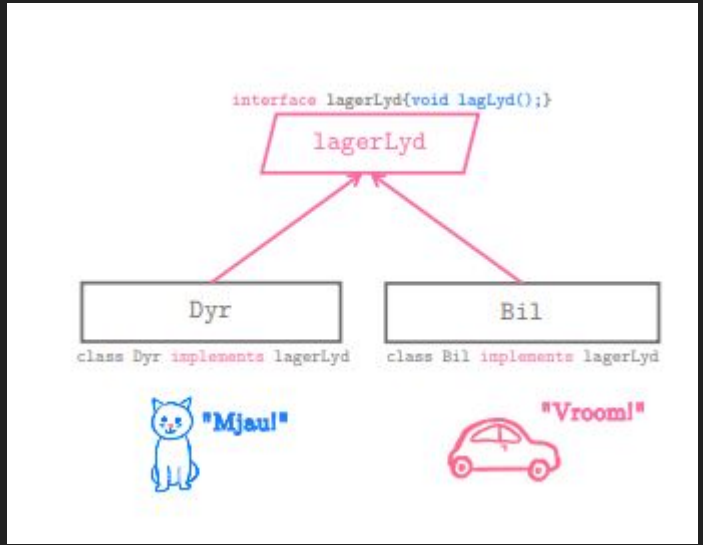
```
class Dyr {
    public void lagLyd(){System.out.println("Pip!");}
}

class Hund extends Dyr {
    public void lagLyd(){System.out.println("Voff!");}
}

class Katt extends Dyr {
    public void lagLyd(){System.out.println("Mjau!");}
}
```

Arv og subklasser 3

- Interface / grensesnitt
  - Samling felles egenskaper
    - Ulike ting av ulike typer kan dele egenskaper
      - f.eks lagerLyd
- Nøkkelord:
  - interface
  - implements

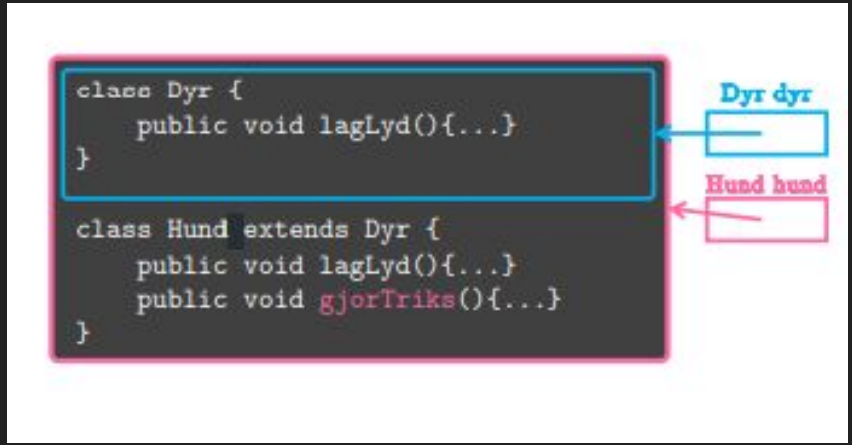


Casting - eksempel

```java
public static void main(String[] args) {
    Dyr dyr = new Hund();
    dyr.gjorTriks(); // går ikke

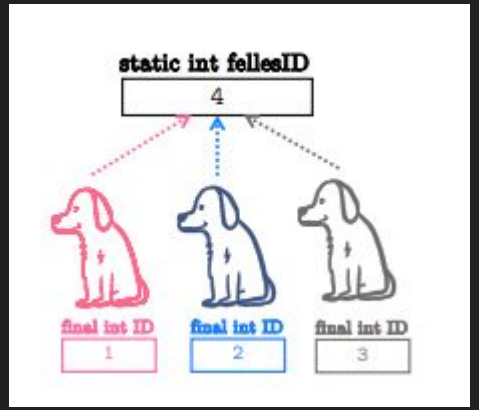
    Hund hund = (Hund) dyr;
    hund.gjorTriks();

    System.out.println(dyr.equals(hund));
    // --> true
}
```



Utfordring: final og static

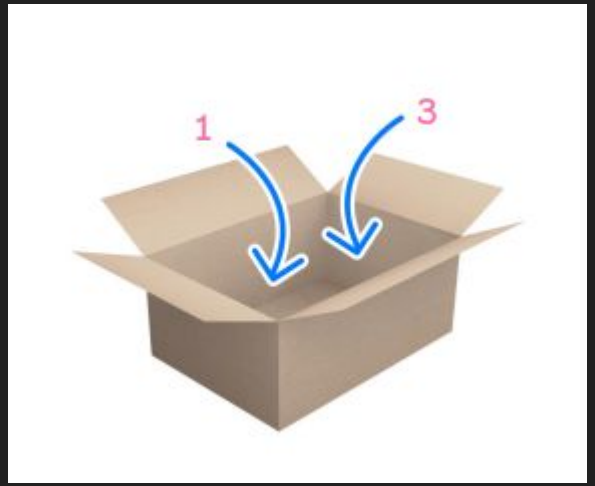
- final: nøkkelord for **konstanter**
  - kan ikke endres etter initialisering
- static: nøkkelord for **felles** for alle objekter av en klasse
  - Statiske metoder og felter
  - kan ikke bruke this
  - kan kalles uten objekt



Beholdere

Beholdere

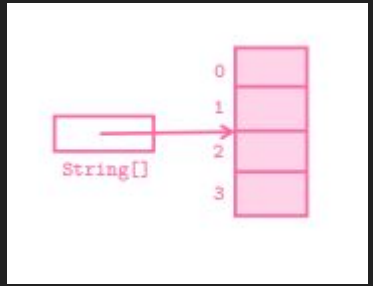
- Array og ArrayList
- Generiske klasser
- Lenkelister

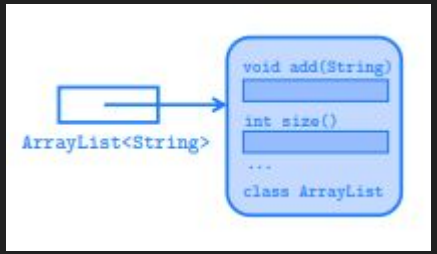


Array og ArrayList

- **Array:** sammenhengende gruppe i minnet som holder på et gitt antall verdier av samme type
  - Konstant størrelse
  - Tilordner og henter verdier med indekser (plassering i arrayet)

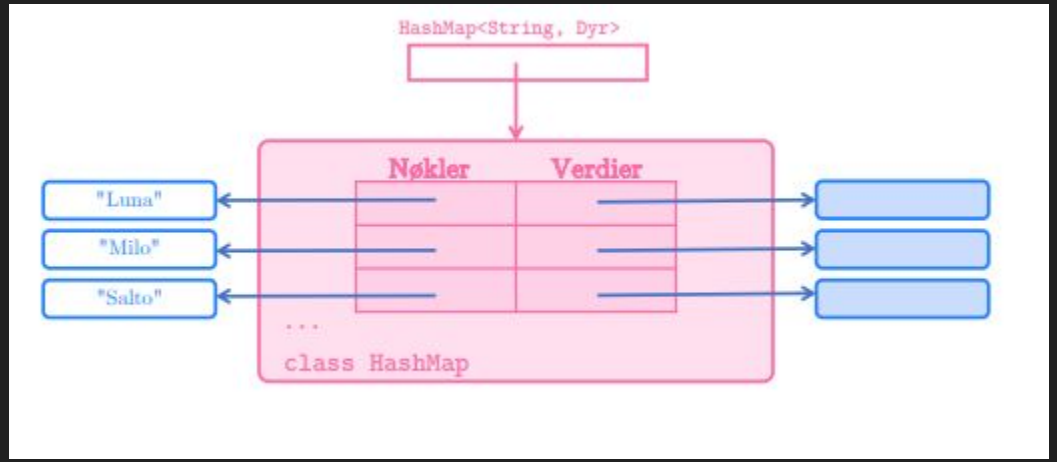
- **ArrayList**: fleksibel listeklasse i Java-biblioteket
  - Kan utvides/trekkes sammen etter behov
  - Men tar mer plass





HashMap

- Samling hvor elementer identifiseres med en nøkkel



Generiske klasser

- **Ønsker ofte at beholdere skal fungere til ulike typer/klasser**
  - Tungvint å lage en type beholder for integers og en annen type for String
- Beholderen kan ha parameter i klasse deklarasjonen
  - class Beholder { … }
- Bestemmer typen/klassen beholderen skal ha i det man oppretter en instans av beholderen

```
ArrayList<Integer> tall = new ArrayList<>();
ArrayList<String> strings = new ArrayList<>();
```



Generiske klasser

- **Ønsker ofte at beholdere skal fungere til ulike typer/klasser**
  - Tungvint å lage en type beholder for integers og en annen type for String
- Beholderen kan ha parameter i klasse deklarasjonen
  - class Beholder { … }
- Bestemmer typen/klassen beholderen skal ha i det man oppretter en instans av beholderen

```
ArrayList<Integer> tall = new ArrayList<>();
ArrayList<String> strings = new ArrayList<>();
```



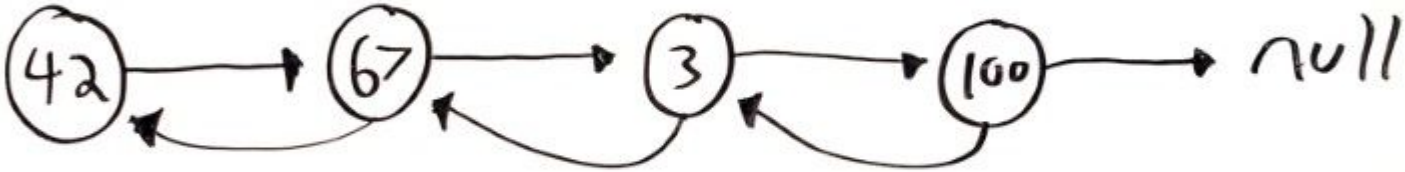
Lenkelister

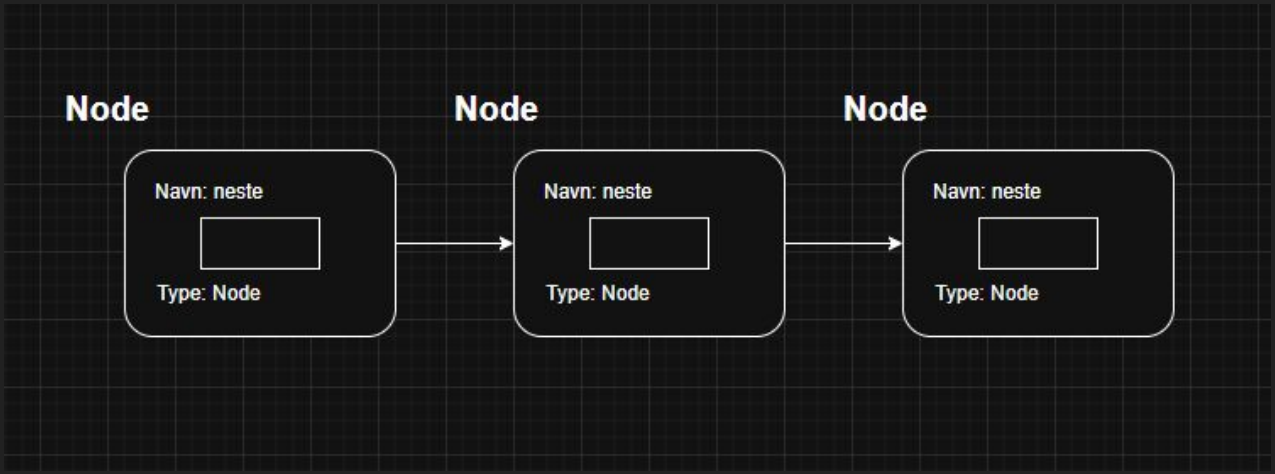
- Kan implementere egne beholdere.
- En hjelpeklasse Node.
  - Kan hete noe annet, men det er vanlig å kalle den Node.
- Node har en referanse til neste node i lenkelisten.
- Alle noder har dermed en referanse til neste node
  - **Unntak**: siste node

```
class Node {
    Node neste; //= null hvis ingen neste

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```


42
67
3
100
null




```
class Node {
    Node neste; //= null hvis ingen neste

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```

Lenkelister

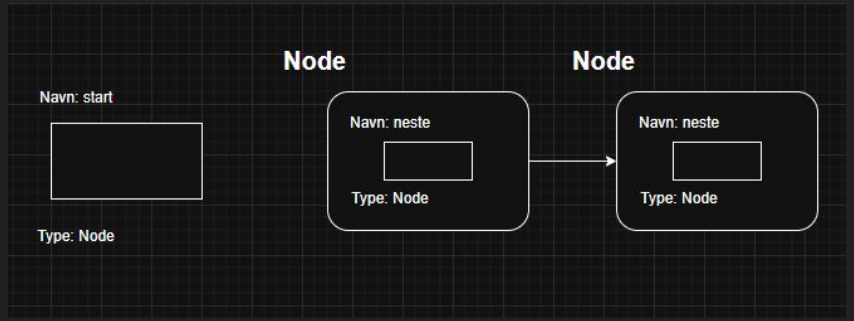
- Må ha en referanse til første noden i lenkelisten for å kunne nå alle elementene i lenkelisten.

- Starter med å opprette en startnode (start)
- Oppretter en node som skal komme etter startnoden (nrTo)
- Setter nrTo til å være start sin neste

```
class Node {
    Node neste; //= null hvis ingen neste

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }
```



Lenkelister

- Node-klassen vi har gått gjennom hittil er ganske ubrukelig da den ikke holder på noe data
  - Kun en peker til neste
- Ofte ønsker vi at en Node inneholder:
  - En peker til neste node (eller null)
  - Data

```
class Node {
    Node neste; //= null hvis ingen neste
    String verdi;

    //Konstruktør:
public Node(String verdi){
        this.verdi = verdi;
    }

    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }

    public String hentData(){
        return verdi;
    }
}
```

Ulike typer lister / lenkelister

- **Kø / queue**– first in first out (**FIFO**)
  - Som en vanlig kø i butikken.
- **Stabel/stack** – last in first out (**LIFO**)
  - Som tallerkener som man stabler oppå hverandre.
- **Prioritetskø** – Den med høyest prioritet først
  - Feks den sykeste pasienten på legevakten

```
class Node {
    Node neste; //= null hvis ingen neste
    String verdi;

    //Konstruktør:
public Node(String verdi){
        this.verdi = verdi;
    }

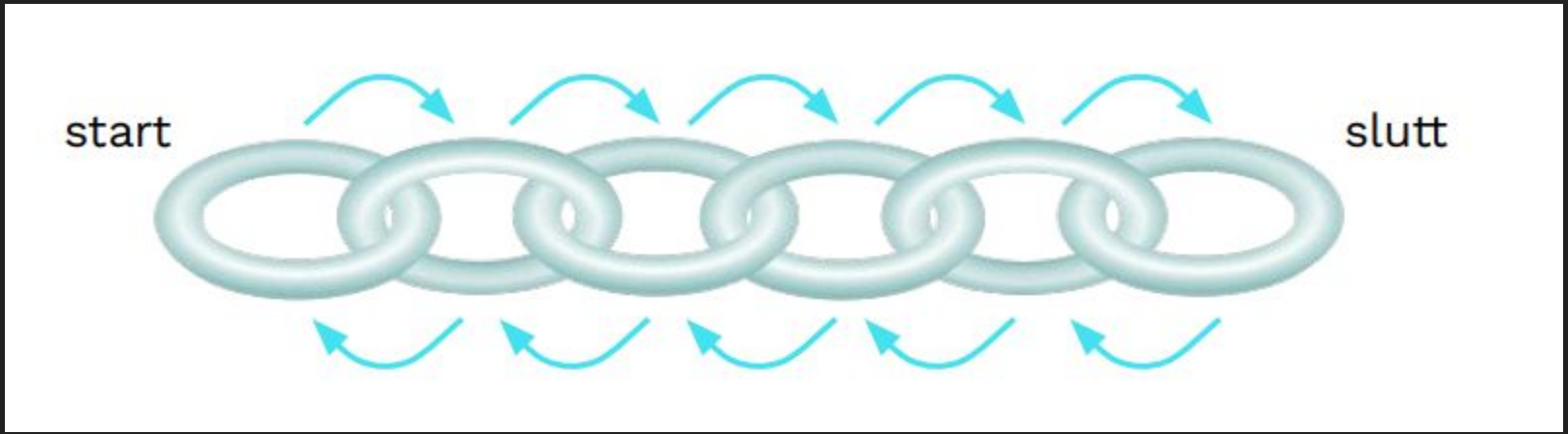
    public void settNeste(Node n){
        neste = n;
    }

    public Node hentNeste(){
        return neste;
    }

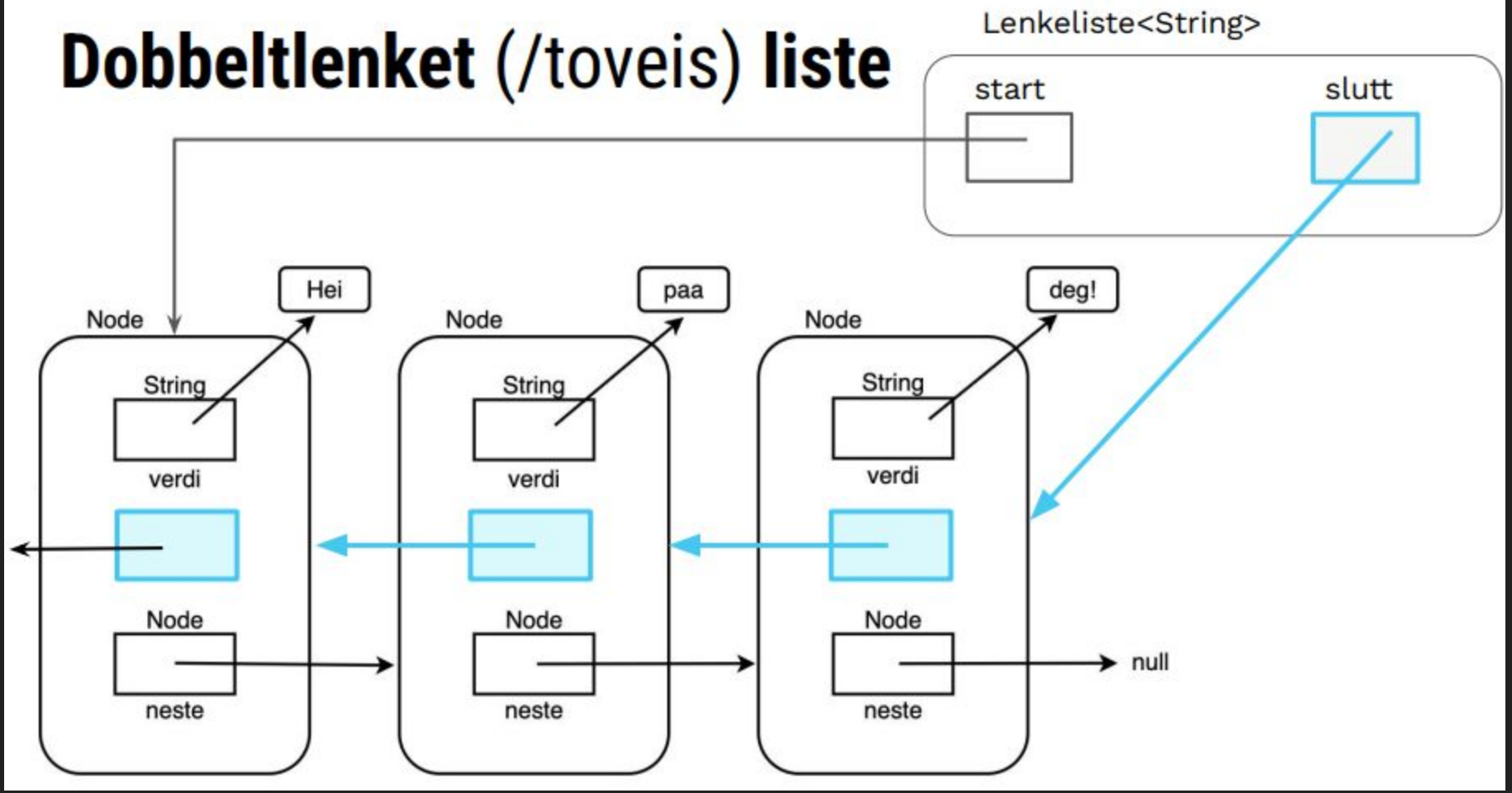
    public String hentData(){
        return verdi;
    }
}
```

Dobbeltlenke liste (toveis)

- Kan gå igjennom listen begge retningene
- Lettere sletting av noder som ikke ligger først i listen
- Mer å holde orden på når man skriver metodene



Dobbeltlenket (/toveis) **liste**



Comparable, compareTo

- Comparable: interface med metoden

```
int compareTo(T otherObj);
```

- Implementer hvis vil sammenligne objekter av vår klasse
- **a**.compareTo(**b**) returnerer en int…
  - = 0 hvis lik
  - **positivt** tall hvis **a** er **større** enn **b**
  - **negativt** tall hvis **a** er **mindre** enn **b**

Hva burde skrives ut? (0 / positivt tall /negativt tall)?

- compareTo er implementert i Vare basert på prisen

```
public static void main(String[] args) {
    Vare agurk = new Vare(pris: 20);
    Vare tomat = new Vare(pris: 10);
    Vare melk = new Vare(pris: 20);

    System.out.println(agurk.compareTo(tomat));
    System.out.println(agurk.compareTo(agurk));
    System.out.println(agurk.compareTo(melk));
    System.out.println(tomat.compareTo(melk));
```

Hvordan kan vi implementere compareTo i Vare basert på pris?

```java
class Vare implements Comparable<Vare> {
    int pris;

    public Vare(int pris) {
        this.pris = pris;
    }

    @Override
    public int compareTo(Vare o) {
        return pris - o.pris;
    }
```

```java
class Vare implements Comparable<Vare> {
    int pris;

    public Vare(int pris) {
        this.pris = pris;
    }

    @Override
    public int compareTo(Vare o) {
        if (pris > o.pris) return 1;
        if (pris < o.pris) return -1;
        // pris == o.pris
        return 0;
    }
```

Iterable

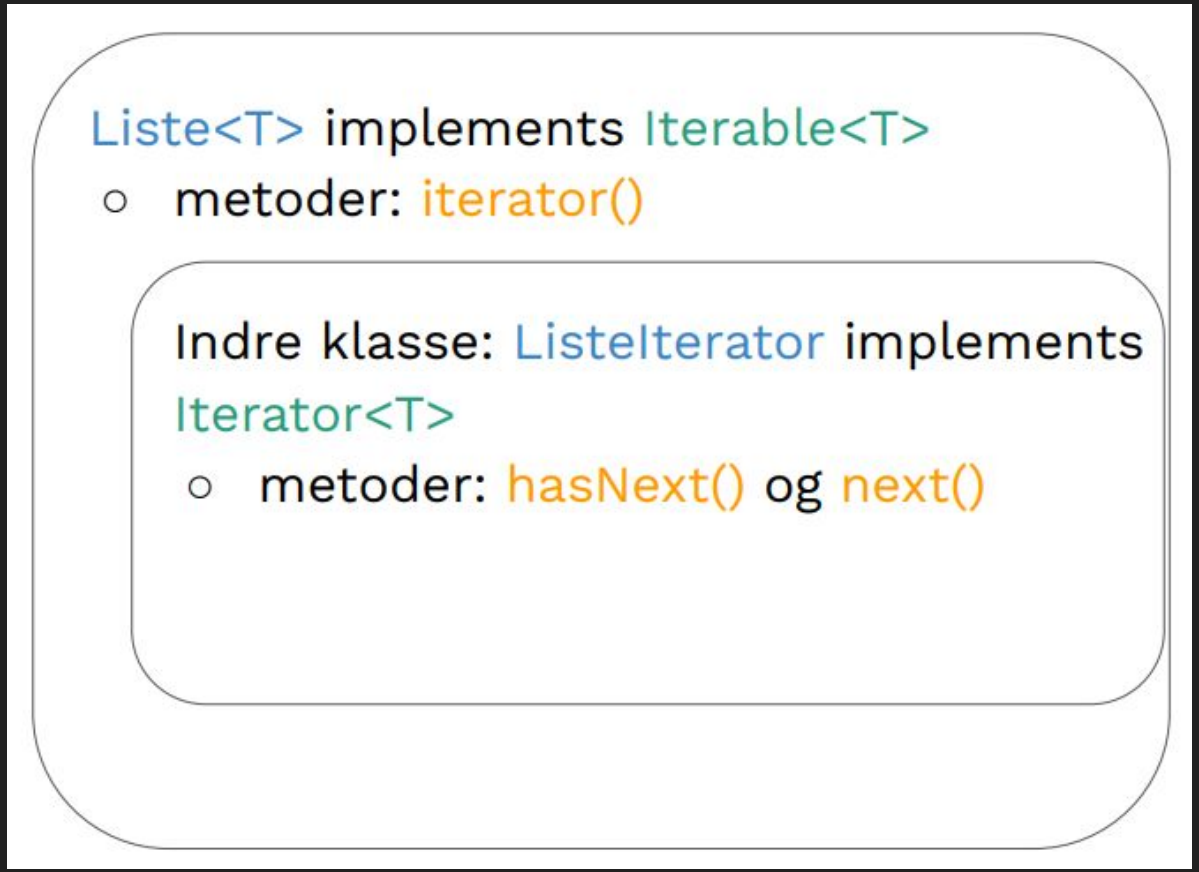
- **Iterable:** interface med metoden

```
public Iterator<T> iterator()
```

- For å kunne gå gjennom listen med en for-each loop

Iterable

- Trenger:

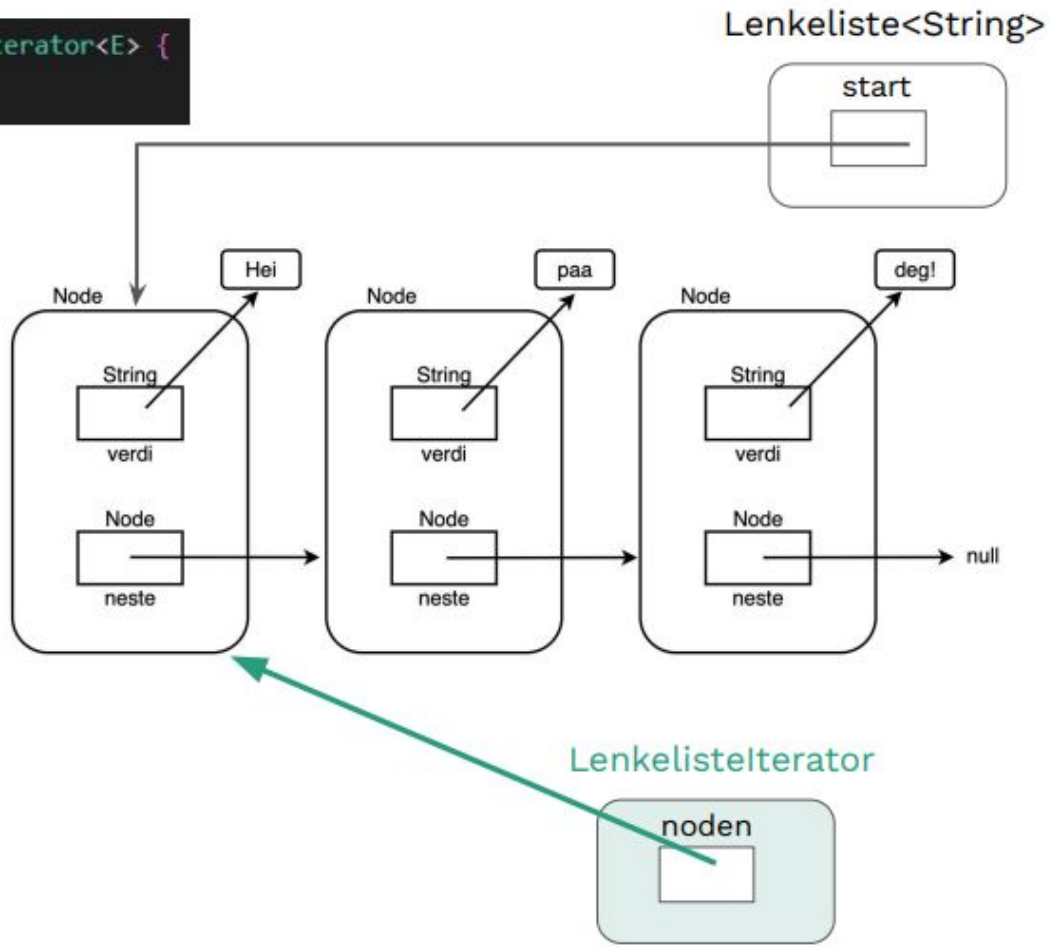


```java
private class LenkelisteIterator implements Iterator<E> {
    Node noden = start;
```

```java
@Override
public boolean hasNext() {
    return noden != null;
}
```

```java
@Override
public E next() {
    if (noden == null) {
        throw new NoSuchElementException(
            s: "next");
    }

    E returVerdi = noden.verdi;
    noden = noden.neste;
    return returVerdi;
}
```

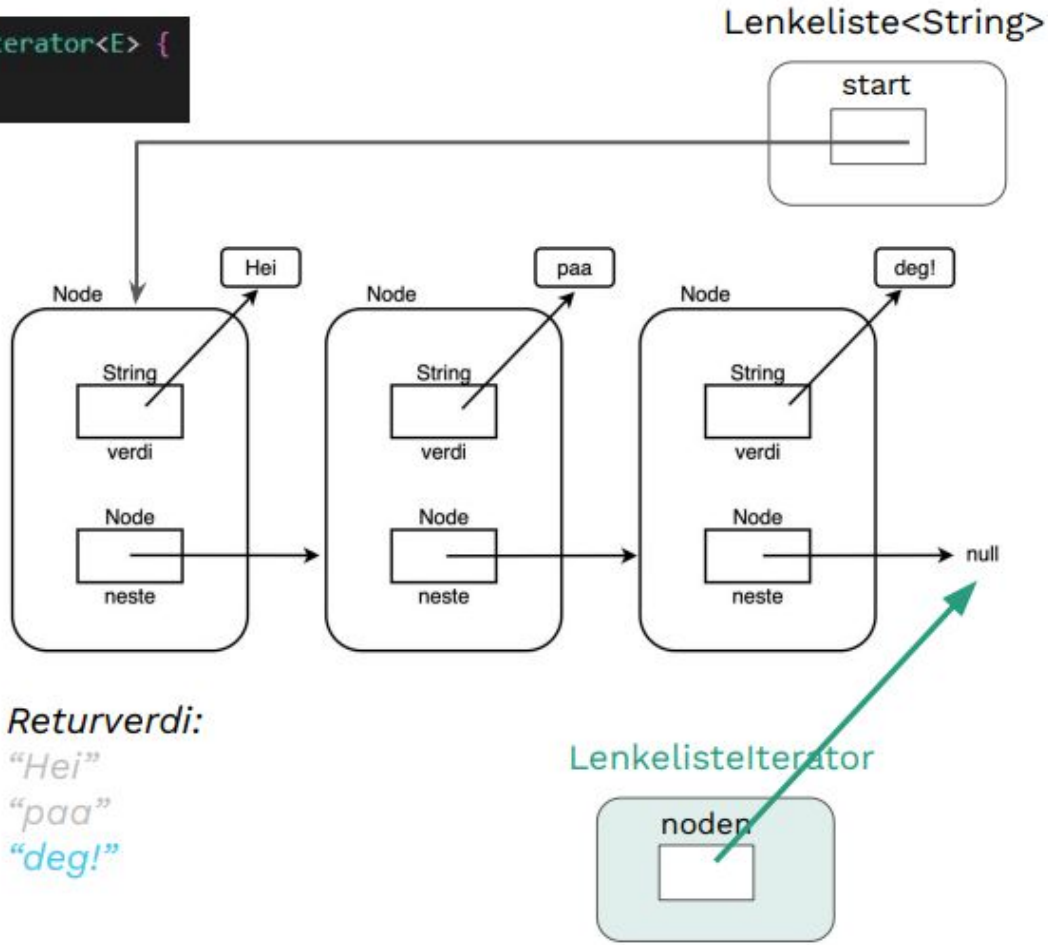


```java
private class LenkelisteIterator implements Iterator<E> {
    Node noden = start;
```

```java
@Override
public boolean hasNext() {
    return noden != null;
}
```

```java
@Override
public E next() {
    if (noden == null) {
        throw new NoSuchElementException(
            s: "next");
    }

    E returVerdi = noden.verdi;
    noden = noden.neste;
    return returVerdi;
}
```



Oppsummering

- **Arv og subklasser**
  - Arv
  - Komposisjon
  - Polymorfi
  - Interface
  - Casting
  - final og static
- **Beholdere**
  - array
  - ArrayList
  - Lenkeliste
  - Generiske klasser
  - Kø, stabel, prioritetskø
  - Comparable
  - Iterator / iterable

Oppsummering

- **Andre relevante temaer:**
  - Exceptions
  - Scanner
    - Fillesing/-skriving
    - Input
  - Datastrukturtegning
  - Overloading

Midtveiseksamen

- 20 Mars kl 15.00
  - 2 timer
  - Silurveien 2
- Ingen hjelpemidler
- Må bestås
  - Teller 25% av endelig karakter (dv.s du kan fortsatt få A hvis du får B nå!)

Lab!