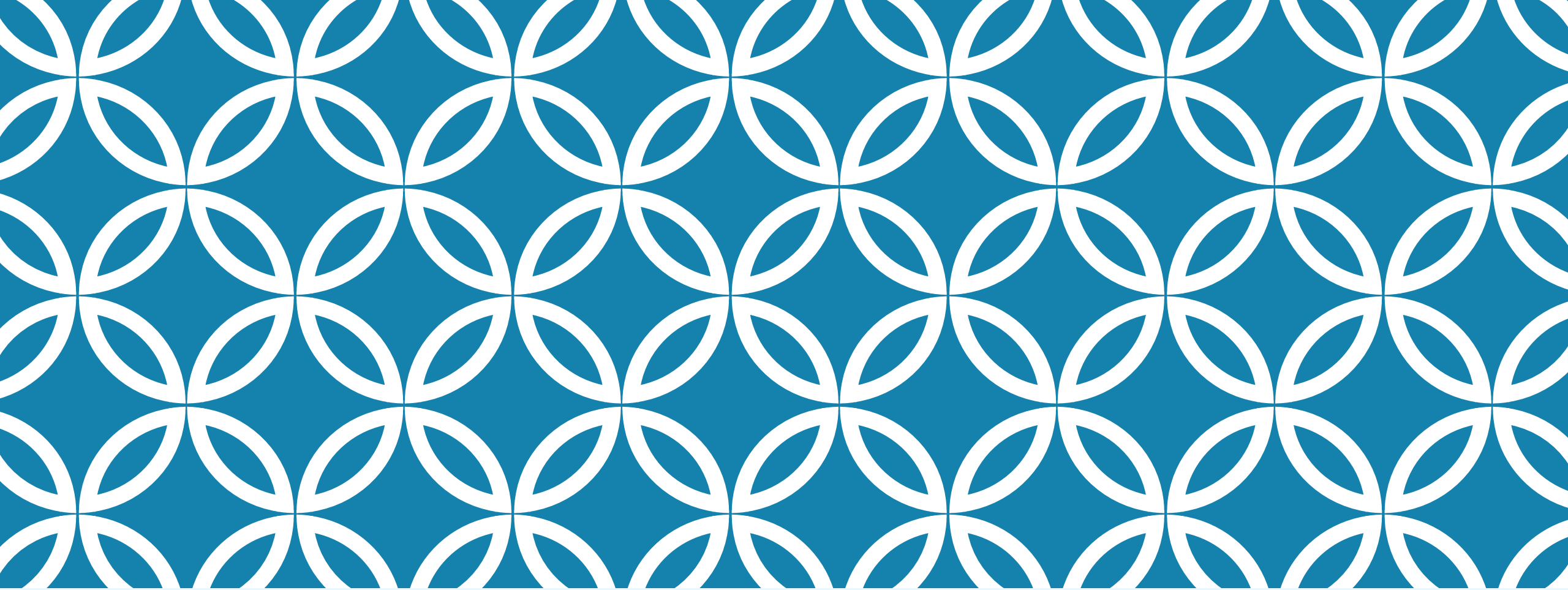


UKE 8 – BEHOLDERE 2

agneshb@uio.no

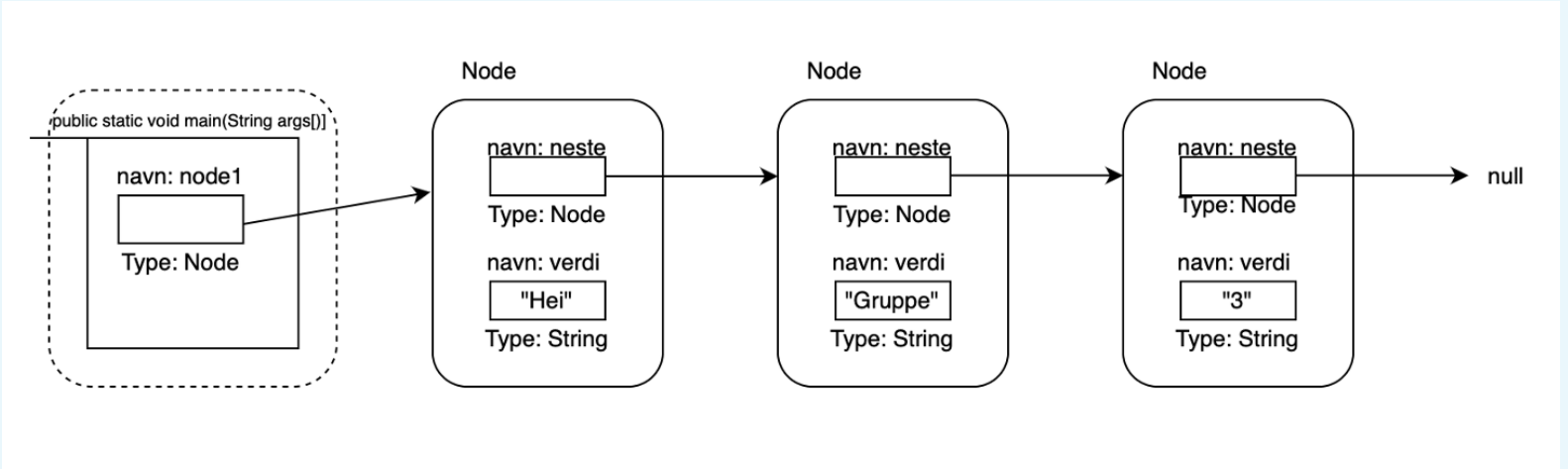
FORRIGE GANG

Generiske klasser

- class Klasse<T> {...}

Lenkelister

Node-klassen



GENERISKE KLASSER

Beholderen kan ha et parameter i klassedeklarasjonen.

- `class Beholder<Type> {..}`

T er en placeholder for en type.

Må spesifisere typen **T** i det vi oppretter en instans av den generiske klassen Boks.

E (Element), **K** (Key), **T** (Type), **V** (Value)

```java
public class Boks<T> {
    private T t;

    public void fyll(T t) {
        this.t = t;
    }

    public T hent() {
        return t;
    }
}
```

```java
Run | Debug
public static void main(String[] args) {

    Boks<String> b = new Boks<String>();
    String tekst = "hallo!";

    b.fyll(tekst);
    System.out.println(b.hent()); //"hallo!"

}
```

I DAG

Fortsetter med lenkelister

Enkeltlenket vs dobbeltlenket

Ulike typer lister

Interfacet Iterable

Interfacet Comparable

Livekoding av en kø/FIFO-lenkeliste med iterable

LENKELISTE MED NODER

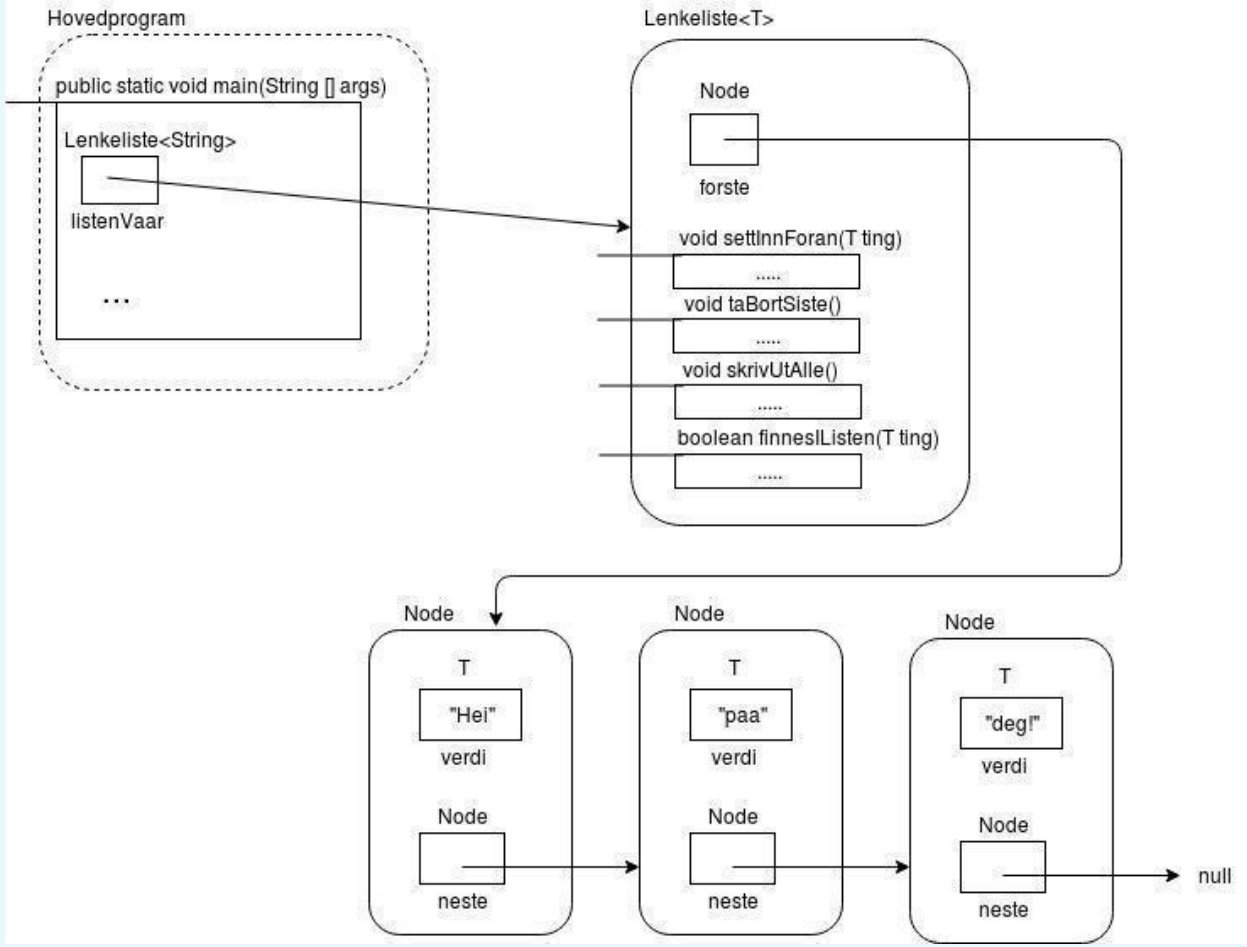
Noder er koblet sammen i en *lenkestruktur* inni en lenkeliste.

Lenkelisten håndterer innsetting, sletting og andre operasjoner.

Referanse til lenkelisten i main.

- Referanse til første (evt. siste) node ligger inni lenkelisten.

Gjennom referansen får vi tilgang til listemetoder og dermed data i nodene.



LENKELISTE

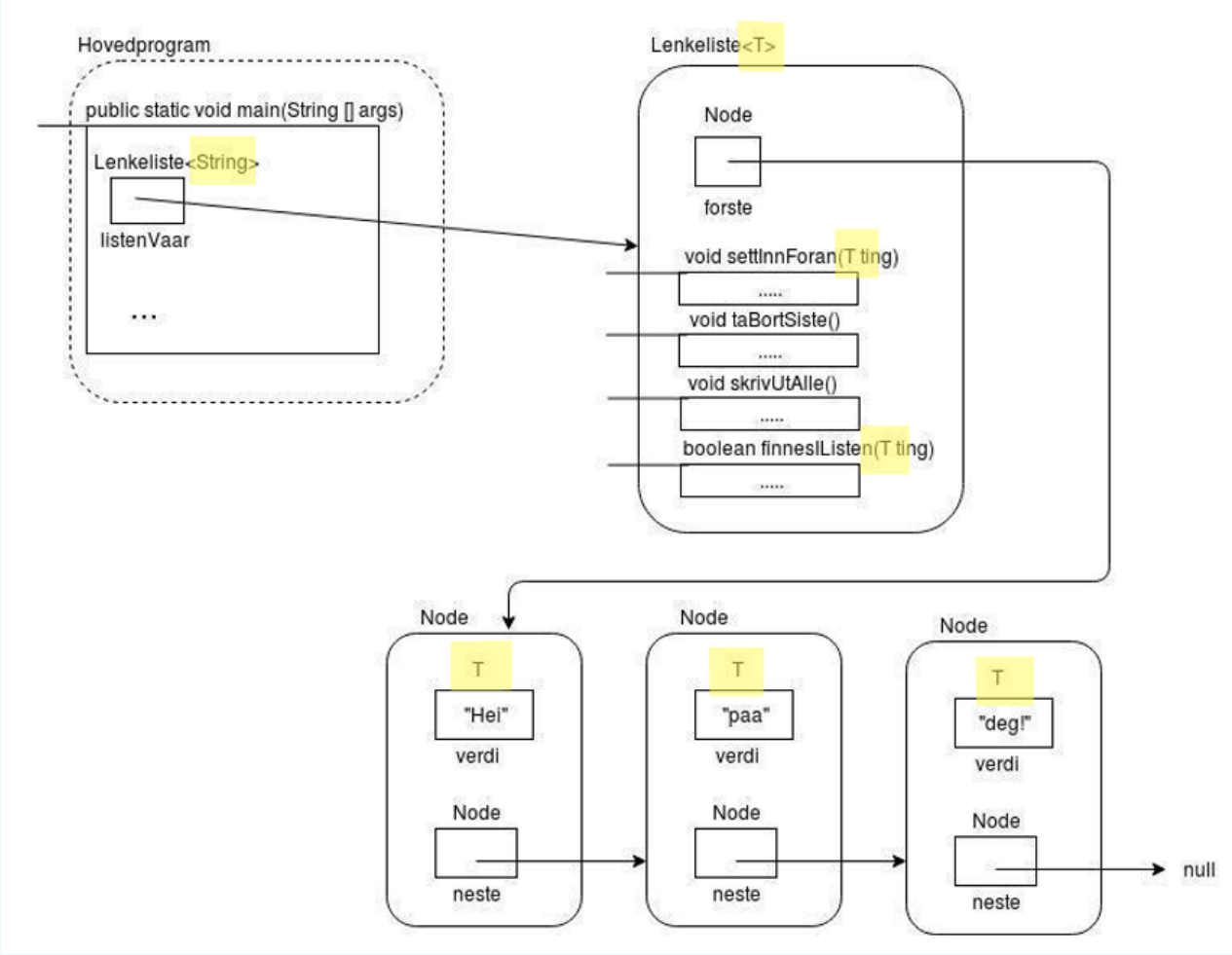
Lenkelister kan være generiske

- Klassen tar inn et typeparameter

Lenkelisten tar inn en generisk type, mens den indre Node-klassen bruker typen.

Wrapper-klasser:

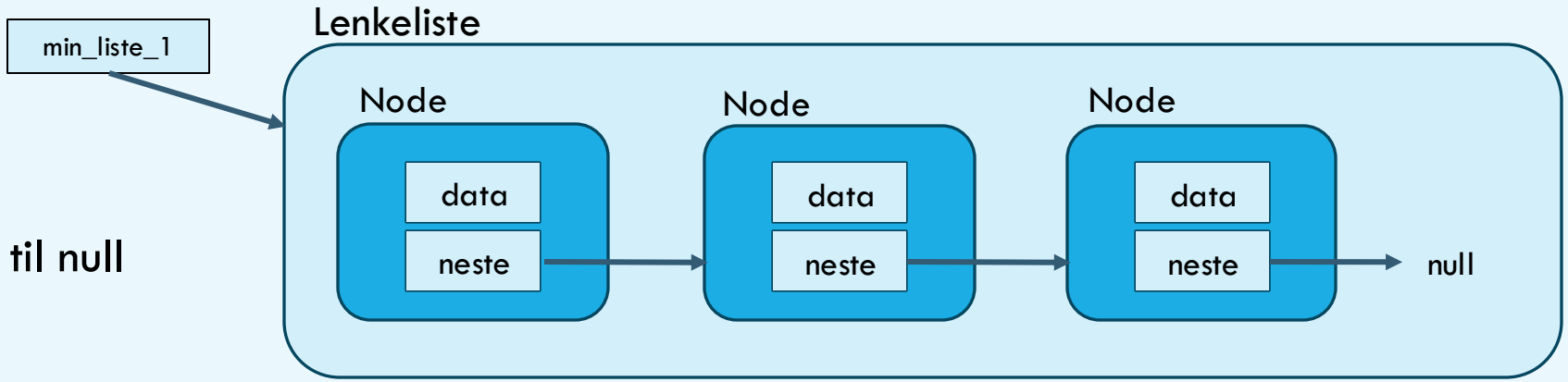
- Integer, Double, Boolean



ENKELTLENKET LISTE VS. DOBBELTLENKET LISTE

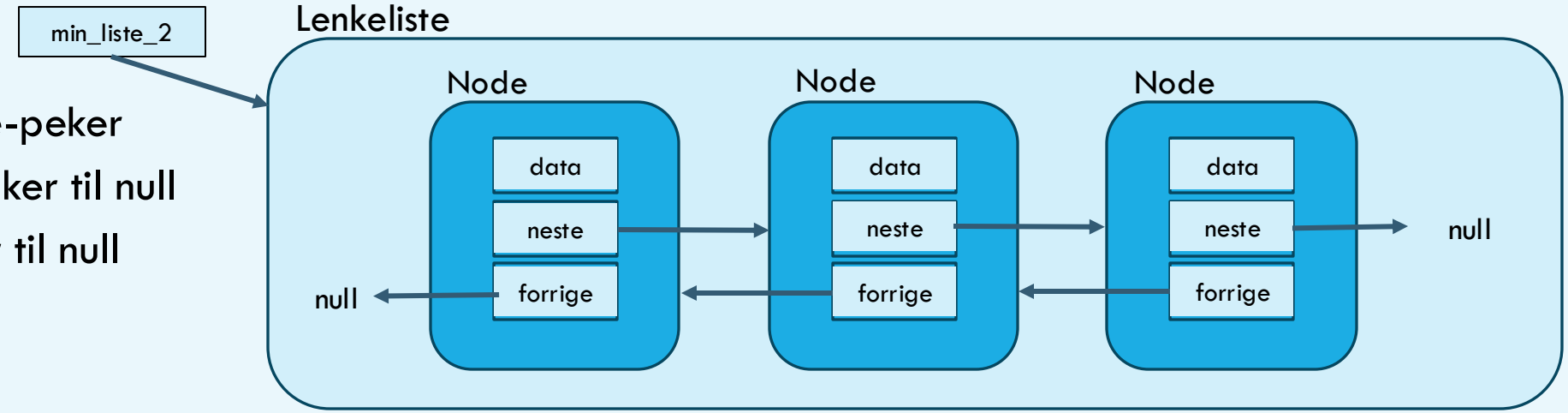
Enkeltlenket

- Kun neste-peker
- Siste node: neste peker til null



Dobbellenket

- Neste-peker og forrige-peker
- Første node: forrige peker til null
- Siste node: neste peker til null



(Ikke komplett datastrukturtegning, se feks slide 6)

Enkeltlenket

```
public class Enkeltlenket<T> {
    Node forste, siste;

    class Node {
        T data;
        Node neste;
        Node(T data) {
            this.data = data;
        }
    }
}
```

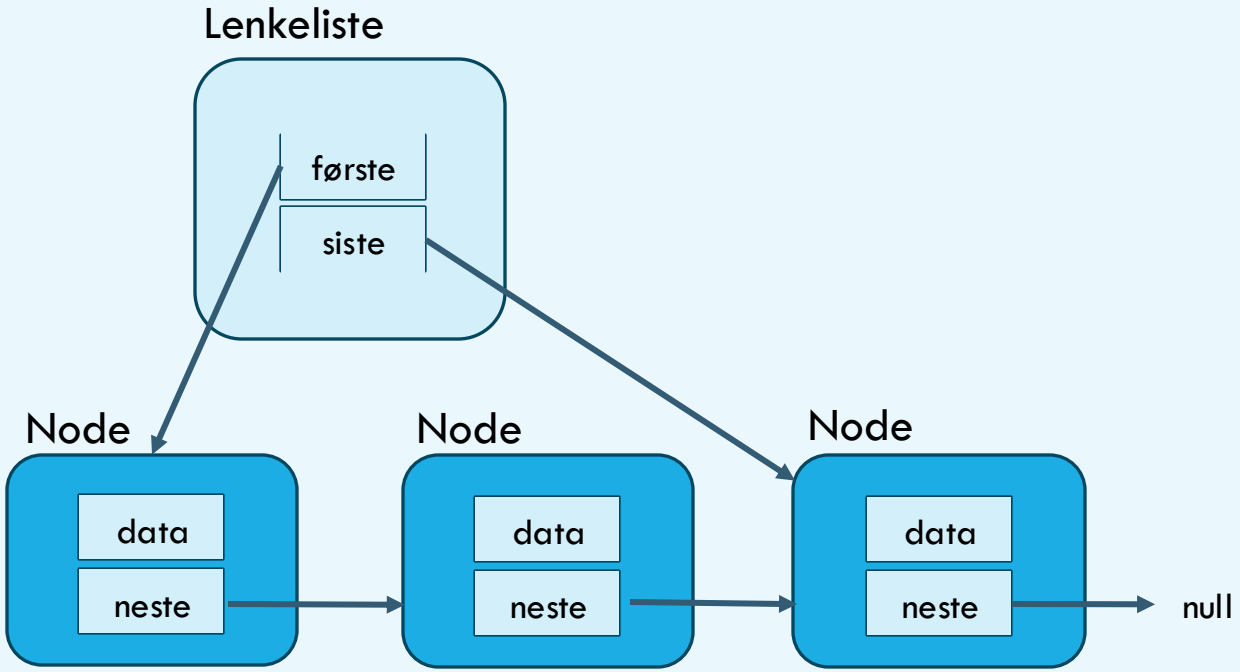
Dobbeltlenket

```
public class Dobbeltlenket<T> {
    Node forste, siste;

    class Node {
        T data;
        Node forrige
        Node neste;
        Node(T data) {
            this.data = data;
        }
    }
}
```

LENKELISTE

- Ofte lurt å ha en peker til første **og** siste node.
- Hvis man skal ha tak i siste element/legge til noe bakerst, men kun har peker til første node må man traversere hele lenkelisten.



ULIKE TYPER LISTER

Stabel/stack (LIFO)
- elementer settes inn og tas ut foran
- (f.eks. tallerkenstabel)

Kø (FIFO)
- elementer settes inn bakerst og tas ut foran
- (f.eks. kassekø i butikken)

Prioritetskø
- elementer settes inn og tas ut etter ønsket prioritet
- (f.eks. eldre først)
- Elementene er sortert etter det de prioriteres på, og man henter ut det første elementet.

Metode for å legge til og fjerne elementer fra listen er implementert annerledes ut i fra hvilken type

INTERFACET ITERABLE

Når vi vil iterere over elementer i lister og andre beholdere med en **for each-løkke**.

```
Lenkeliste<T> implements Iterable<T>
```

```
public interface Iterable<T> {
        …
    Iterator<T> iterator();
```

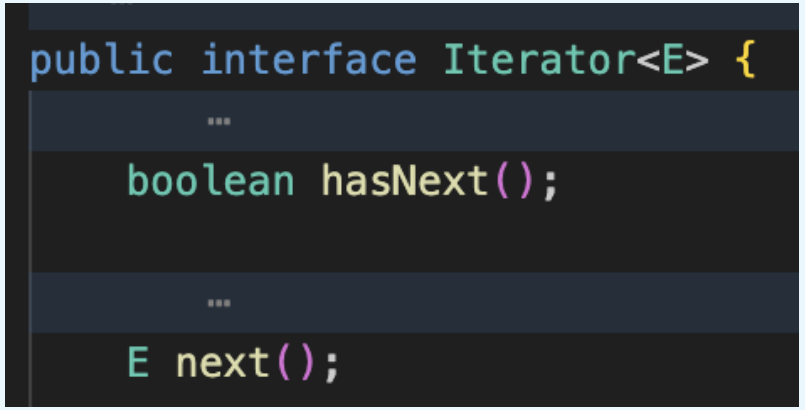
Må implementere metoden iterator(), som returnerer et objekt av typen Iterator<T>.

Iterator<T> er et grensesnitt, og du lager selv en iterator-klasse som implementerer dette grensesnittet.

INTERFACET ITERATOR

Lager en indre klasse i lenkelisten som implementerer Iterator:

```
class LenkeListeIterator implements Iterator<T>
```



hashNext() – sjekker om det er et til element å iterere over.

next() – går til neste element.

hasNext-metoden og next-metoden kan tilpasses slik vi ønsker.

LENKELISTE MED ITERABLE SER OMTRENT SLIK UT

```
class Lenkeliste<T> implements Iterable<T> {

        @Override
        public Iterator<T> iterator() {
                return new LenkeListeIterator();
    }
        class LenkeListeIterator implements Iterator<T>{
                @Override
                public boolean hasNext() {..}

                @Override
                public boolean next() {..}
        }
}
```

1. Impelemterer Iterable
2. En metode iterator som returnerer et objekt av typen Iterator, som er klassen i punkt 3.
3. En indre klasse som implementerer Iterator-grensesnittet. Denne må ha metodene hasNext og next.

INTERFACET COMPARABLE

Brukes for å sammenligne objekter på ønsket måte.

Praktisk i prioritetskøer.

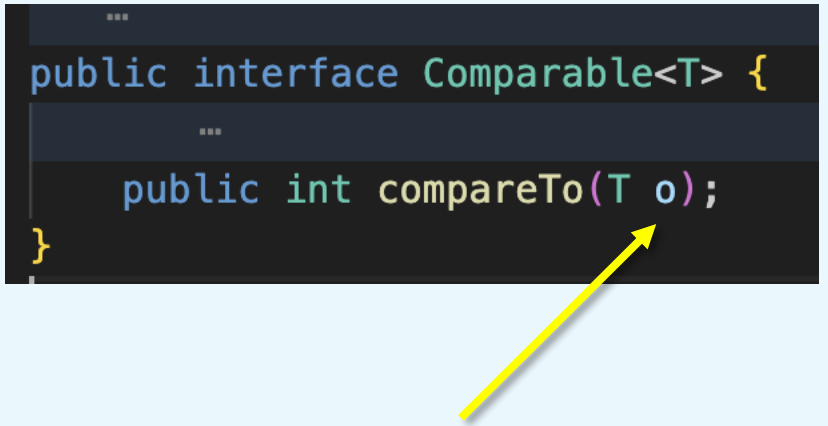
- Klassen Pasient med en variabel for hvor syk den er, og dermed prioritet.
- Klassen Person som prioriteres ut i fra alderen.

Comparable er generisk.

```
...
public interface Comparable<T> {
        ...
    public int compareTo(T o);
}
```

COMPARETO

compareTo-metoden må implementeres i klasser som implementerer Comparable



Leksikografisk (alfabetisk) sammenligning.

Returnerer et positivt heltall når dette objektet er større enn det andre.

Returnerer et negativt heltall når dette objektet er mindre enn det andre.

Returnerer 0 når objektene er like

Objektet vi sammenligner med er det som sendes inn i compareTo()

```
dette.compareTo(annet_objekt)
```

EKSEMPEL

Sammenligne personer baser på alder

Hvis dette objektet (this.alder) er større enn det andre objektet (annen.alder) returneres et positivt tall.

Hvis dette objektet er mindre enn det andre objektet returneres et negativt tall.

```java
public class Person implements Comparable<Person> {
    private int alder;

    Person(int alder) {
        this.alder = alder;
    }

    @Override
    public int compareTo(Person annen) {
        int aldersforskjell = this.alder - annen.alder;
        return aldersforskjell;
    }
}
```

PRIORITETSKØ

For å sikre at objekter kan sammenlignes og sorteres etter prioritet i en prioritetskø, skriver vi slik:

```
public class Lenkeliste<T extends Comparable<T>> {
```

Gjør at T må være av en type som implementerer Comparable<T> og dermed kan sammenlignes med typen T.