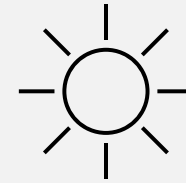


GRUPPE 8 INI010 – UKE 7

Sivert Fjeldstad Madsen

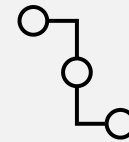
I DAG

- Lenkeliste
- Ulike listevarianter
- Comparable og compareTo
- Iterable
- Oppgaver



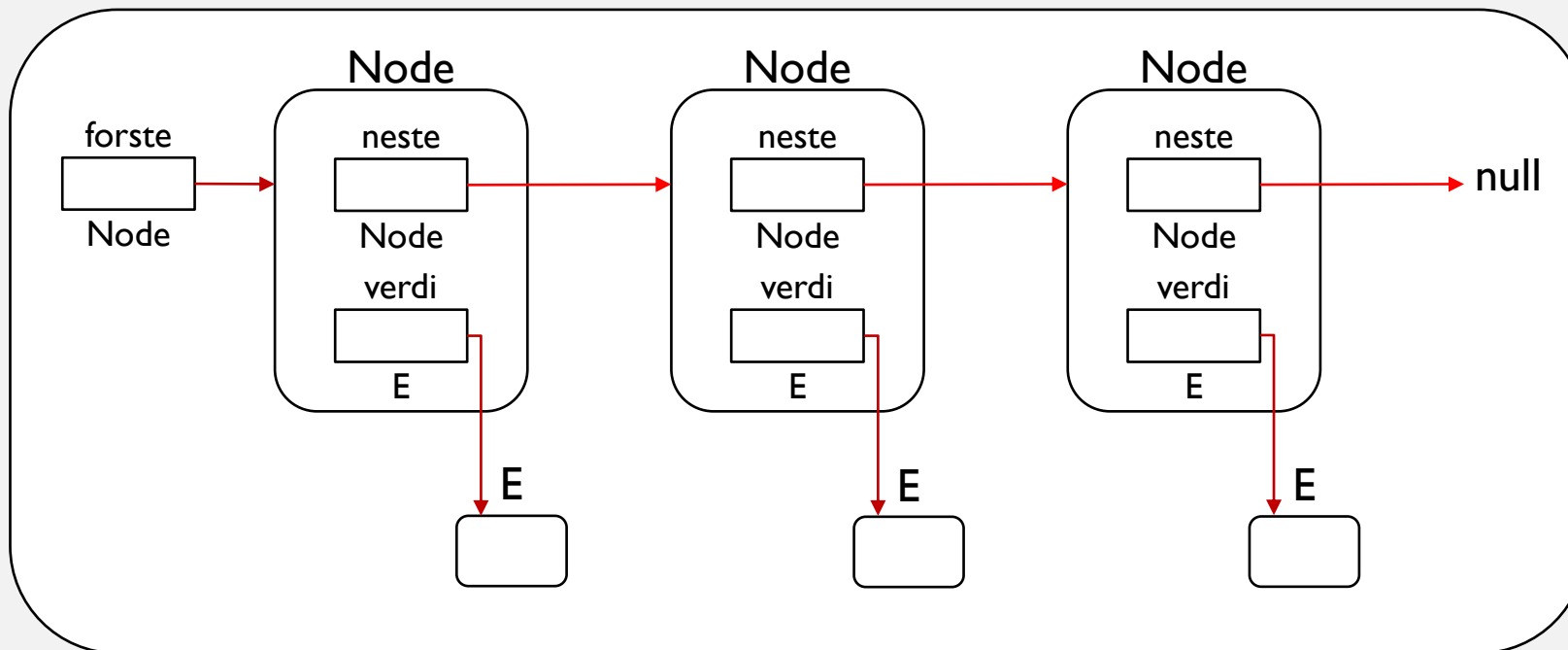
LENKELISTE

- Det er lurt å slippe å hekte sammen masse noder i main
- Vi lager derfor et eget objekt som håndterer nodene
 - Trenger bare referanse til første og eventuelt siste node
- Tilbyr de metodene vi behøver for å hente ut og sette inn
 - Og ting som størrelse på listen
- Kan være **enkeltlenket** eller **dobbeltlenket**
 - Begge variantene har fordeler og ulemper



ENKELTLENKET LENKELISTE - FIGUR

Lenkeliste



LISTEVARIANTER

- Når man designer en liste (eller annen beholder), må man bestemme seg for hvordan den skal oppføre seg ved henting og innsetting
- **FIFO – first in, first out**
 - Typisk hvordan en kø fungerer
- **LIFO – last in, first out**
 - En stabel, hvor man alltid legger inn og henter ut det øverste objektet
- **Prioritetskø**
 - Elementene blir sortert basert på *noe*, og det med høyeste prioritet blir hentet først



FLERE LISTEVARIANTER?

- Hva med:

- **FILO?**
- **LILO?**



- De får ikke være med, ettersom de betyr akkurat det samme som LIFO og FIFO
- Men det finnes mange andre muligheter!
 - Tilfeldig inn og ut
 - Oppførsel forandrer seg basert på størrelse
 - Osv.



SAMMENLIGNE OBJEKTER

- For å kunne implementere (f.eks.) en prioritetskø er vi nødt til å kunne sammenligne objekter
 - Hvilke verdier vi ønsker å sammenligne er helt opp til oss!
- Vi bruker metoden «**int compareTo(T o)**» fra interfacet «**Comparable<T>**»
- *forste.compareTo(andre)* skal evaluere til:
 - 0 dersom *forste* og *andre* er like*
 - <0 dersom *forste* er mindre* enn *andre*
 - >0 dersom *forste* er større* enn *andre*
- *Du bestemmer selv hva det vil si om to objekter er like, mindre enn, eller større enn hverandre



COMPARABLE

- En klasse som implementerer interfacet «Comparable<T>» må implementere metoden «int compareTo(T o)»
- Når vi designer egne beholdere kan vi «tvinge» elementene i beholderen til å *måtte* implementere «Comparable<T>»:



```
class Prioritetskoe<T extends Comparable<T>> {  
    public void settInn(T nyttElement) {  
        ...  
        if (nyttElement.compareTo(peker) > 0) { ... }  
        ...  
    }  
}
```

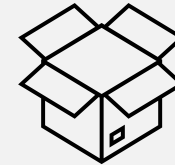

...MEN VENT NÅ LITT

- class Prioritetskoe<T **extends** Comparable<T>> ?
- Er ikke Comparable<T> et interface?
- Betyr det da at **T** også er nødt til å være et interface?
- ...
- **NEI!**
- «extends» her er en placeholder for *enten extends eller implements*.
- Java gidder ikke å lage et eget kodeord som kommer i tillegg til disse to, og bruker bare «extends»



ITERABLE

- I tillegg til å kunne sammenligne objekter i beholderen vi lager, kan vi også designe den slik at det går an å iterere gjennom den med en **for-each**-løkke

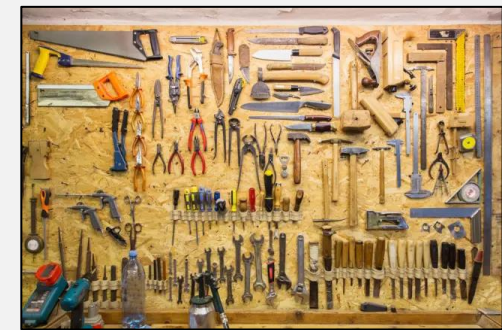


```
for (E element : beholder) { ... }
```

- For å gjøre dette må beholderen vår implementere interfacet **Iterable<T>**, som krever at vi skriver metoden «**public Iterator<T> iterator()**»
- **Iterator<T>** er også et interface, men vi skal *ikke* her returnere et interface!
- Vi skal returnere et objekt som *implementerer* interfacet **Iterator<T>**
- Vi må altså skrive en *indre klasse* i beholderen vår som vi kan returnere instanser av med en metode

ITERABLE - ANALOGI

- La oss gå helt tilbake til starten av objektorientering og se på dette med en analogi:
- Se for deg at du har en masse byggematerialer:
 - Skruer, spikere, planker, osv.
- I tillegg har vi en masse verktøy:
 - Sager, skrutrekkere, driller, hammere, osv.
- Vi kan bruke dette til å lage veldig mye forskjellig



ANALOGI – FORTS.

- Til slutt har du bygget deg et stort fint hus i skogen
- Det funker fint å fyre i peisen og hente vann fra brønnen
 - ...men kanskje du vil ha innlagt vann og strøm?
- Da må du snakke med kommunen!
 - De trenger en del informasjon om huset for å kunne legge inn strøm og vann
- Du kan ikke bare sende dem en enkel skrue, spiker eller planke – **det gir ikke kommunen nok informasjon**
- Du må sette deg ned og lage en ordentlig søknad som inneholder flere forskjellige ting
- Merk at du ikke kan gjenbruke den samme søknaden flere ganger



ANALOGIEN OVERSATT

- La oss gå helt tilbake til starten av objektorientering og se på dette med en analogi:

- Se for deg at du har en masse byggematerialer:

- `int`, `char`, `String` osv.

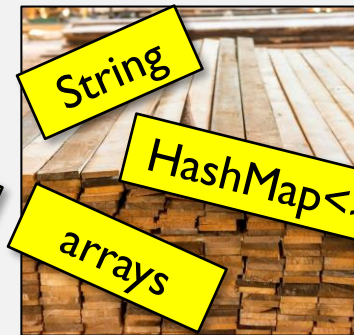
primitive typer



`int`

`boolean`

`char`



`String`

`HashMap<>`

`arrays`

- I tillegg har vi en masse verktøy:

- `if/else`, `try/catch`, `for/while`, `+, -, *, /` osv.

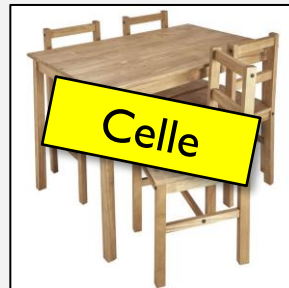
- Vi kan bruke dette til å lage veldig mye forskjellig



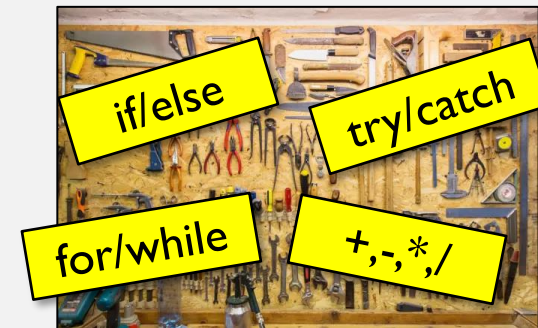
`Person`



`Legemiddel`



`Celle`



`if/else`

`try/catch`

`for/while`

`+, -, *, /`

OVERSATT ANALOGI – FORTS.

- Til slutt har du bygget deg en **en kul beholder** en
- Det funker fint å **bruke en vanlig for-løkke** en
- ...men kanskje du vil **bruke en for-each-løkke?**
- Da må du **implementere Iterable<T>!**

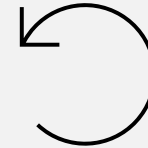


Java trenger en del informasjon om **beholderen** for å kunne **la deg bruke en for-each-løkke**

- Du kan ikke bare sende dem en **int, boolean eller metode** – **det gir ikke nok informasjon**
- Du må sette deg ned og lage **et iterator-objekt** som inneholder flere forskjellige ting
- Merk at du ikke kan gjenbruke den samme **iterator-instansen** flere ganger

ITERATOR - EKSEMPEL

```
import java.util.iterator;  
class Beholder<T> implements Iterable<T> {  
    public Beholder() { ... }  
    class BeholderIterator<T> implements Iterator<T> {  
        @Override  
        public T next() { ... }  
  
        @Override  
        public boolean hasNext() { ... }  
    }  
    public Iterator<T> iterator() {  
        return new BeholderIterator();  
    }  
}
```



OPPSUMMERING

- Lister kan oppføre seg på mange forskjellige måter med hensyn til innsetting og henting
- For å sammenligne objekter må vi implementere interfacet **Comparable<T>**
- For å kunne bruke en for-each-løkke på beholderene våre må vi implementere interfacet **Iterable<T>**, og definere en indre klasse som implementerer interfacet **Iterator<T>**
- Ukas tips:
 - Når vi skal putte variabler inn i strenger kan det lønne seg å bruke `String.format()` eller `printf()`
 - Syntaks:
 - **`String.format(«Hunden %s er %d år gammel.», navn, alder);`**

