Lineære Datastrukturar

- ArrayList
- Pekerkjede
- LinkedList
- Iterere - Iterator
- Stakk
- Kø
- Sammenligning - Comparable

Repetisjon – ArrayList

- Har en tabell internt, «utvides» når den er full
- Holder denne tabellen pakket
- Derfor raskt å legge inn nytt element «bakerst», høyeste indeks
- Og raskt å ta ut element bakerst
- Stakk ble effektiv basert på ArrayList
- Men tung jobb å sette inn «foran», i tabellindeks 0. Må flytte alle de andre elementene for å få plass!
- Like tungt å ta ut foran, må flytte de andre elementene for å holde tabellen pakket.

Kø

- En «kø» er en lineær datastruktur der vi legger inn objekter i den ene enden, og tar ut i den andre.
- Tenk drosjekø (enten kø av drosjer, eller kø av mennesker som venter på drosje!)
- Man stiller seg bakerst i køen. Når alle foran er borte, er det din tur
- La oss prøve å bruke ArrayList til å lage dette

Kø basert på ArrayList

MyQueue
-list: ArrayList<Object>
+MyQueue() +MyQueue(capasity: int) +isEmpty(): boolean +getSize(): int +retrieve(): Object // returnere, ikke ta ut +remove(): Object // ta ut og returnere +add(o: Object): void // legge inn

Vi må velge om «add» skal legge inn bakerst (høyeste indeks) eller foran (indeks 0) i ArrayList'a (list)

add legger inn bakerst

- add blir da effektiv: list.add(o);
- retrieve må se på første – effektivt: list.get(0);
- remove må ta ut første – **tungt!** list.remove(0);

Vi prøver å la add legge inn foran

- Men da blir add tung: list.add(0, o);
- retrieve blir grei, se på bakerste: list.get(list.size()-1);
- og remove blir grei, ta ut bakerste: list.remove(list.size()-1);

Ingen av løsningene var gode!

Pekerkjede

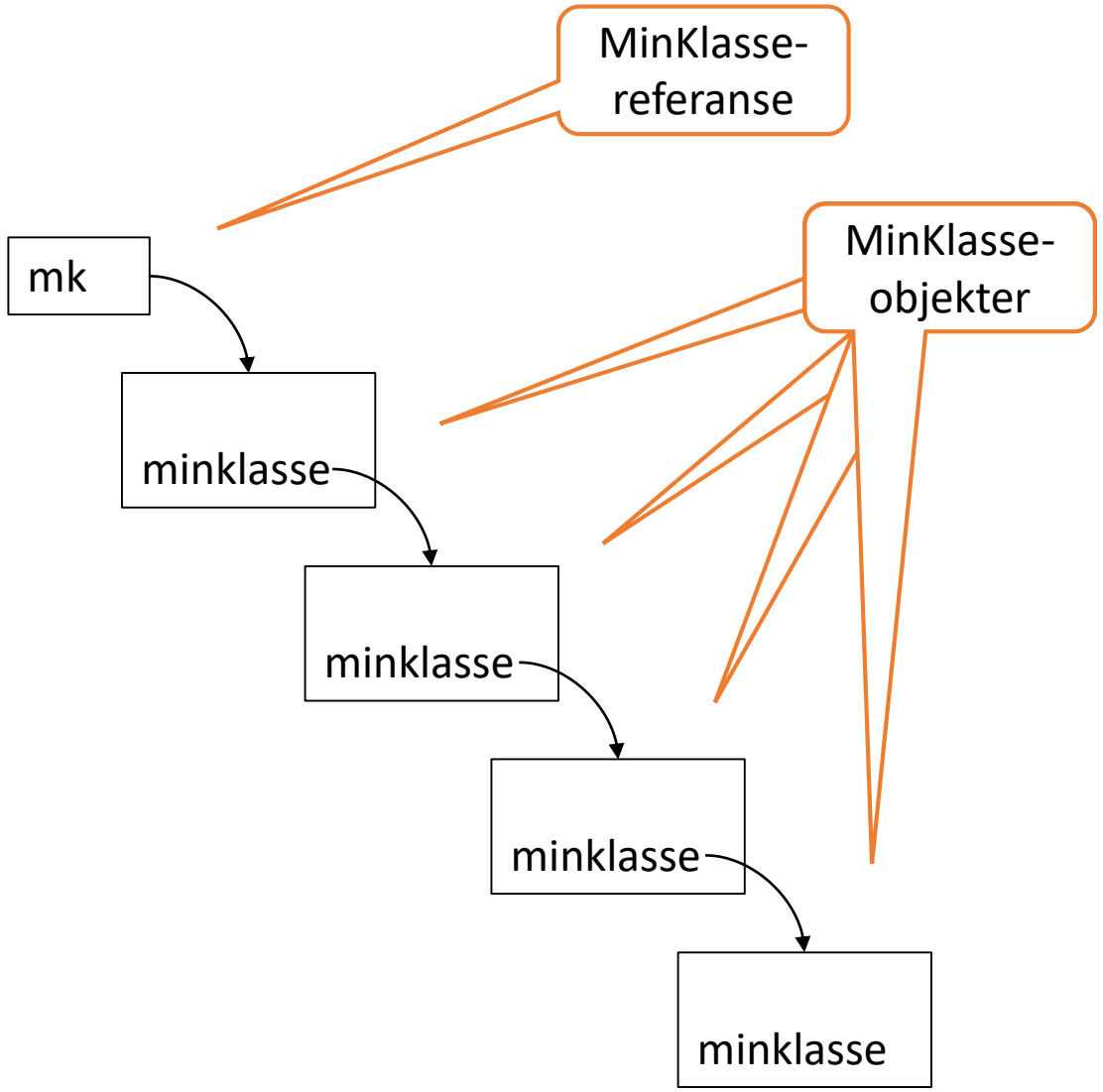
Tenk deg at vi lager en klasse som har som instansvariabel en referanse av klassen selv:

```
class MinKlasse{
  // diverse data og metoder
  MinKlasse minklasse;
}
```

Om vi så oppretter et slikt objekt:

```
MinKlasse mk = new MinKlasse();
```

-så vil den inneholde en referanse av type MinKlasse. Den kan brukes til å holde på et nytt objekt av samme type. Det nye objektet vil inneholde ny referanse, som kan settes til å peke på nytt objekt osv.!



Pekerkjede av personer

- Kunne kjede sammen (f.eks.) Person-objekter desom class Person inneholdt Person-referanse.

```
class Person{
  Person p;
  // og mye annet
}
```

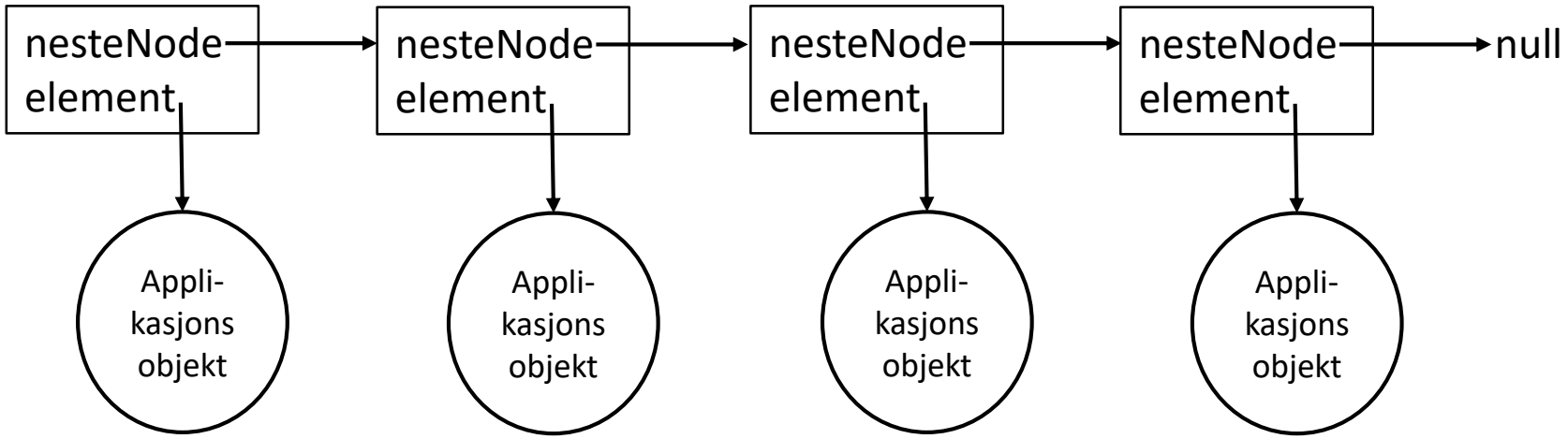
-og slik ble det ofte gjort for 30-40 år siden!

Ulempe: Må inneholde en slik ekstra referanse, enten vi bruker den eller ikke.

«Moderne» pekerkjede

- Vi lager et lite ekstra objekt som bare har til oppgave å holde pekerkjeden samlet! Kan se slik ut:

```
class Node{
  Node nesteNode;  // referanse til neste node-objekt
  Object element;  // applikasjonsobjektet – det som add'es
}
```



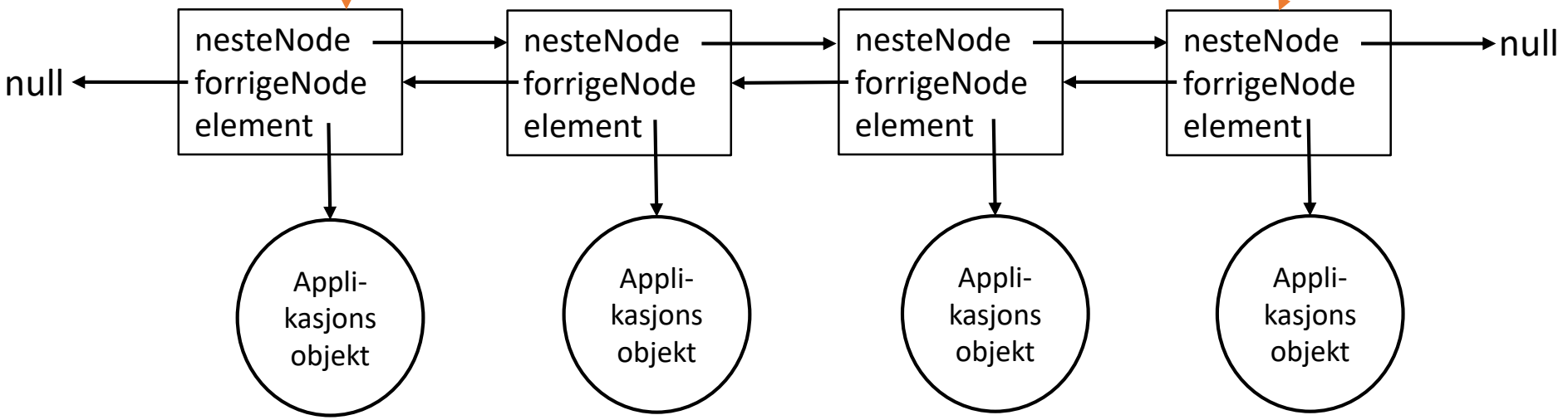
Vilkårlig størrelse! Bare begrenset av minne

class LinkedList

- Har et litt større Node-objekt:

```
class Node{
  Node nesteNode;   // referanse til neste node
  Node forrigeNode; // referanse til forrige node
  Object element;   // applikasjonsobjektet
}
```

Alt LinkedList trenger av instansvariable, er referanse til første og siste node!



Kø basert på LinkedList

MyQueue
-list: LinkedList<Object>
+MyQueue() // +MyQueue(capasity: int) er uaktuell +isEmpty(): boolean +getSize(): int +retrieve(): Object // returnere, ikke ta ut +remove(): Object // ta ut og returnere +add(o: Object): void // legge inn

```
public class MyQueue{

  LinkedList<Object> list;

  public MyQueue(){
    list = new LinkedList<>();
  }

  public boolean isEmpty(){
    return list.isEmpty();
  }

  public int getSize(){
    return list.size();
  }
```

Hvordan implementere retrieve, remove og add? Vi må studere API'et. Hva har LinkedList å tilby?

Kø basert på LinkedList (forts.)

```
// Må velge om vi skal legge inn foran eller bak.
// Begge løsninger blir effektive. Velger legge inn bak.
// Må da ta ta ut foran

  public Object retrieve(){
    return list.element(); // evt list.get(0); evt list.getFirst();
  }

  public Object remove(){
    return list.remove(); // evt list.remove(0); evt list.removeFirst();
  }

  public void add(Object o){
    list.add(o);
  }
}     // Slutt class MyQueue
```

ArrayList eller LinkedList?

- De kan stort sett utføre det samme
- Men noen operasjoner er mer effektive med ArrayList
- Andre operasjoner er mer effektive med LinkedList
- Vi må forstå hvordan de virker, for å forstå hva som er effektivt!

Hvordan iterere (gå gjennom) en ArrayList/LinkedList

1. Med for-løkke og indeksering. Anta at «liste» er ArrayList eller LinkedList

```
for (int i=0; i<liste.size(); i++)
  System.out.println(liste.get(i));
```

Fungerer greit for ArrayList. **Veldig tungt** for LinkedList! Hvorfor?

Hvordan iterere (gå gjennom) en ArrayList/LinkedList

2. Med kompakt løkkesyntaks

```
for (Object o : liste)
  System.out.println(o);
```

Effektiv løsning for både ArrayList og LinkedList! Hvorfor?

Én liten ulempe: Vi kjenner ikke indeksen.
Anta at vi skal finne posisjon at gitt objekt:

```
for (int i=0; i<liste.size(); i++)
  if (liste.get(i).equals(gittObjekt))
    return i;
```

-kan ikke lages med teknikk 2.

Hvordan iterere (gå gjennom) en ArrayList/LinkedList

3. Med Iterator

```
Iterator it = liste.iterator();
while (it.hasNext())
  System.out.println(it.next());
```

Effektiv løsning for både ArrayList og LinkedList – tilsvarer teknikk 2!

Kan også starte bakers og gå framover:

```
Iterator it = liste.iterator(list.size());
while (it.hasPrevious())
  System.out.println(it.previous());
```

Iterator er generisk, vi bør derfor spsifisere:

```
LinkedList<Person> llp = new LinkedList<>();        // legg inn objekter...
Iterator<Person> itp = LinkedList.iterator();
Person p = itp.next();                              // Uten <Person> blir det mye casting!
```

Sortere / sammenligne

- Ofte behov for å sortere objekter
- Da må vi kunne sammenligne – hvilket av disse to er størst?
- Må defineres slik:

Comparable er et *grensesnitt*, et *interface*. Sjekk API'et!

```
class Person implements Comparable<Person>{
  // navn, adresse, fødselsdato osv

  public int compareTo(Person p){
    // Skal returnere >0 desom this er større enn p,
    // Skal returnere 0 dersom this og p er like
    // Skal returnere <0 dersom p er større enn this
    // Men hva er «størst»? Det må vi definere! F.eks. basert på navn!
    return this.navn.compareTo(p.navn);
  }
}
```

interface Comparable

Method Detail

compareTo

`int compareTo(T o)`

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive.

Parameters:

`o` - the object to be compared.

Returns:

a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

Throws:

`NullPointerException` - if the specified object is null

`ClassCastException` - if the specified object's type prevents it from being compared to this object.

implements Comparable

- Betyr: jeg lover å ha alle de metoder som er spesifisert i grensesnittet.
- Comparable har bare én metode: compareTo. Den må lages.
- Mer om grensesnitt og abstrakte klasser neste uke.

- Hvordan bruke dette?

```
Person p1 = new Person("Atle Antonsen", ...);
Person p2 = new Person("Bernt Balken", ...)
if (p1.compareTo(p2) > 0) ...
  //Testen vil gi false her, A er ikke større enn B
```

Lister med objekter som er Comparable kan lett sorteres!

ArrayList<Person> alp = new ArrayList<>(); // Legg inn Person-objekter
Collections.sort(alp); // Kan sammenligne Person-objekter, kaller vår compareTo