Sekvenser

Litt om ADT-hierarki i JDSL 1.0 (Java Data Structures Library)

Container ADT

Iterator – Enumeration ADT (fra java.util)

RankedSequence ADT (Også kallt ADT Vektor)

Position ADT

PositionalSequence ADT (Også kallt ADT Liste)

Sequence ADT

Sammenlikning av implementasjoner vha array/lenket liste/2-veis lenket liste

Generisk Sortering av Sekvenser

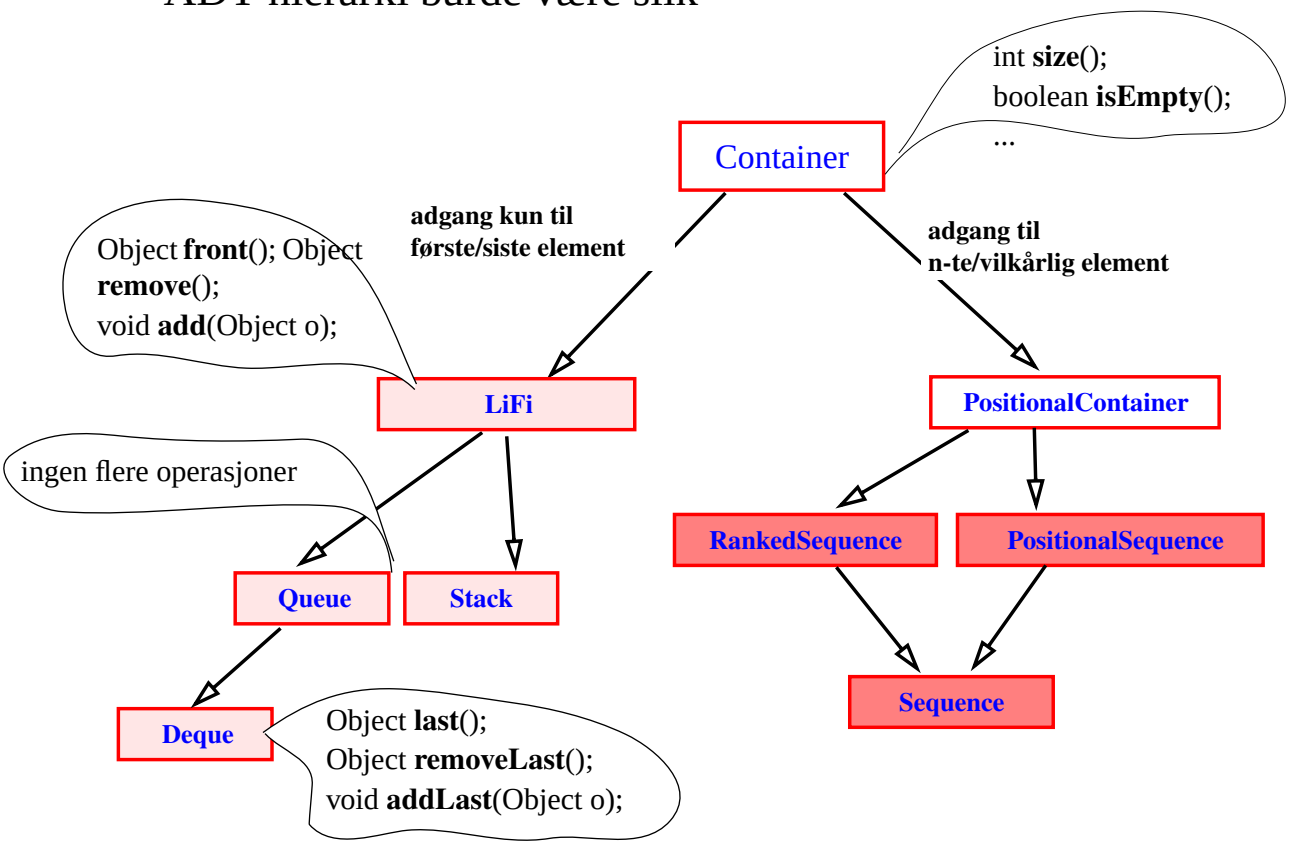
Kap	unntatt	kursorisk
3	3.2.4, 3.5.3	3.1.3, 3.2.3
hele 4		

Container ADT

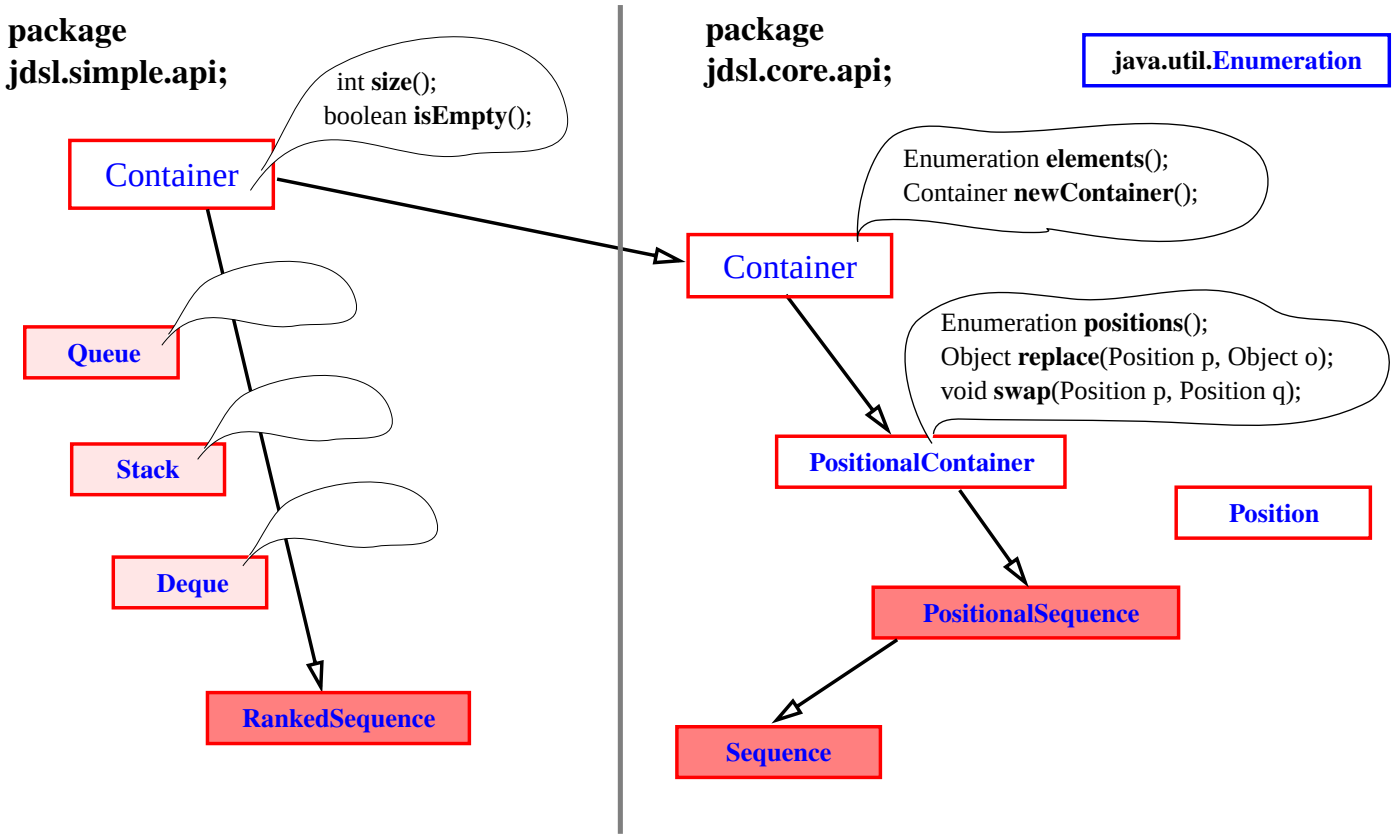
package jdsl.simple.api;

```
/**  generisk samling av Objekter
 *   supertype for alle samlinger
 */

public interface Container {

/**  @return      true hviss samlingen er tom */
     boolean isEmpty();

/**  @return      antall elementer i samlingen */
     int size();
}
```

ADT hierarki burde være slik



... men i JDSL 1.0 er det altså slik



Container og Enumeration ADT

```
package jdsl.core.api;

import java.util.Enumeration
/**  generisk samling av Objekter  */
public interface Container extends jdsl.simple.Container {
/**  @return      enumerering av alle elementene fra samlingen */
     Enumeration elements();
/**  @return      ny samling
                  av samme type */
     Container newContainer();
}
```

f.eks. for å skrive ut alle elementene fra en instans/objekt av type Container C;

```
Enumeration en = C.elements();
while (en.hasMoreElements())
      System.out.println(en.nextElement());
```

```
package java.util;
/**  for å traversere en samling en gang */
public interface Enumeration {
/**   @return true hviss flere elementer */
      boolean hasMoreElements();
/**   @return   neste element
 *                første kall gir første elementet
 *    @exception NoSuchElementException
 *                hvis ikke flere elementer  */
      Object nextElement();       }
```

Det nyere 'interface Iterator' tillater også å fjerne et objekt.

ADT vektor= interface RankedSequence i JDSL

```
package jdsl.simple.api;

/** adgang til/fjerning/innsetting av elementer
 *   med vilkårlig rank (ulik Stack og Queue)
 *   Rank – angis med et tall 0 <= r < size()  */
public interface RankedSequence
      extends (jdsl.simple.)Container {

/**  setter inn Objekt o med rank r, andres rank kan endres
 *    @param       r  et tall: 0 <= r <= size()
 *    @param       o  Objektet som skal innsettes
 *    @exception   BoundaryViolationEx om r < 0 eller r > size()
 */  void insertElemAtRank(int r, Object o);
```

A	B	C	...	O	V	...	Z
0	1	2	...	r	r+1	...	N+1

```
/**  returnerer Objektet med rank r
 *    @param       r  et tall: 0 <= r < size()
 *    @return      Objektet  i posisjon r
 *    @exception   BoundaryViolationEx om r < 0 eller r > size()-1
 */  Object elemAtRank(int r);
```

A	B	C	...	V	...	Z
0	1	2	...	r	...	N

```
/**  erstatter Objektet med rank r med et nytt Objekt
 *    @param       r  et tall: 0 <= r < size()
 *    @param       o Objektet som skal innsettes
 *    @return      Objektet som ble erstattet
 *    @exception    BoundaryViolationEx
 *                  om r < 0 eller r > size()-1
 */  Object replaceElemAtRank(int r, Object o);
```

A	B	C	...	X	V	...	Z
0	1	2	...	r	r+1	...	N+1

```
/**  fjerner og returnerer Objektet med rank r
 *    @param       r  et tall: 0 <= r < size()
 *    @return      Objektet  som ble fjernet
 *    @exception   BoundaryViolationEx
 *                 hvis r < 0 eller r > size()-1
 */  Object removeElemAtRank(int r);
}
```

A	B	C	...	V	...	Z
0	1	2	...	r	...	N

Implementasjon av JDSL RankedSequence vha java.util.Vector

*med **java.util.Vector***

```
package java.util;
public class Vector {
...
void ensureCapacity(
      int minCapacity) { ... }
void insertElementAt(
      Object o, int i) { ... }
void setElementAt(
      Object o, int i) { ... }
void removeElementAt(
      int i) { ... }
Object elementAt(int i) { ... }
Object lastElement() { ... }
Object firstElement() { ... }
int capacity() { ... }
int size() { ... }
}
```

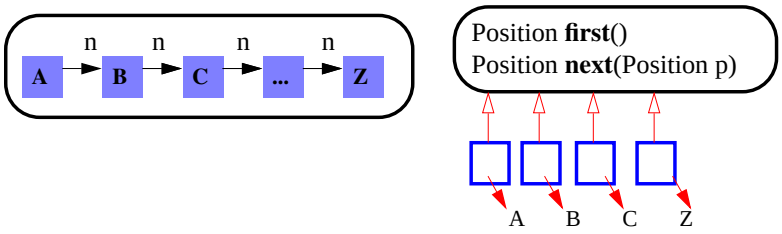
```
import java.util.Vector;
public class RSvector implements RankedSequence {
  private Vector v;
  public RSvector() { v= new Vector(); }
  public void insertElemAtRank(int r, Object o)
        throws BoundaryViolationException  {
    try{ v.insertElementAt(o, r); }  // sjekker rank r
    catch(ArrayIndexOutOfBoundsException e) {
        throw new BoundaryViolationException ; } }
  public void elemAtRank(int r) {  // sjekk rank r
    return v.elementAt(r);  }
  public Object removeElemAtRank(int r) { //sjekk rank r
    Object o = v.elementAt(r);
    v.removeElementAt(r); return o;  }
  public Object replaceElemAtRank(int r) { //sjekk rank r
    Object o = v.elementAt(r);
    v.setElementAt(r); return o;  }
  public int size() { return v.size();  }
  public boolean isEmpty() { return v.size() == 0; }
}
```


Implementasjoner av RankedSequence

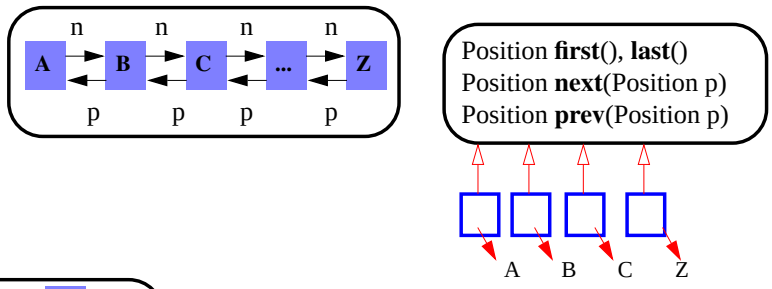
kompleksitet operasjon	Vector *relativt!!!*	Array	en-veis liste	to-veis liste
Container				
size	*1 Vector-op*	1	1	1
isEmpty	*1 Vector-op*	1	1	1
RankedSequence				
elemAtRank	*1 Vector-op*	**1**	n	n
insertElemAtRank	*1 Vector-op*	n	n	n
removeElemAtRank	*1 Vector-op*	n	n	n
replaceElemAtRank	*1 Vector-op*	**1**	n	n

Position ADT

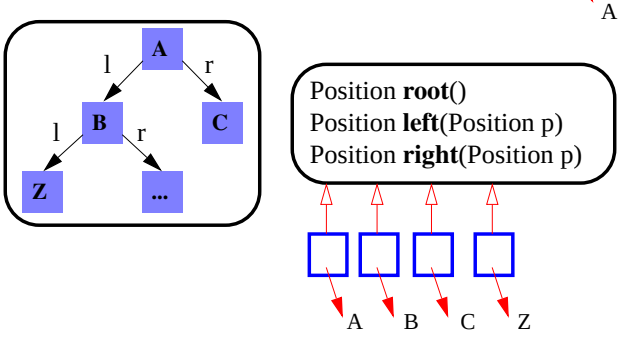
= ***abstraksjon av "et sted"***
som kan lagre noe




```
package jdsl.core.api;
public interface Position {
  Object element();
  Container container();
  void setElement(Object o);
}
```



"Rå struktur"
= *aksess-metoder til*
en samling av Position*'er*



interface PositionalContainer i JDSL

```
package jdsl.core.api;
 import java.util.Enumeration
 /** generisk samling av Objekter som aksesseres vha posisjon */
 public interface PositionalContainer extends jdsl.api.Container {
 /** @return      enumerering av posisjonene fra samlingen */
      Enumeration positions();

 /** bytt Objektene lagret ved argumentposisjoner
  *   @param      p, q posisjoner i samlingen
  *   @exception  InvalidPositionException hvis p eller q ikke er i denne samlingen */
      void swap(Position p, Position q) throws InvalidPositionException;

 /** erstatt Objektet ved posisjon p med Objektet o
  *   @param      p posisjon der Objektet skal erstattes
  *   @param      o det nye Objektet som skal plasseres ved p
  *   @return     det gamle Objektet lagret opprinnelig ved p
  *   @exception  InvalidPositionException hvis p ikke er i denne samlingen */
      Object replace(Position p, Object o) throws InvalidPositionException;
 }
```

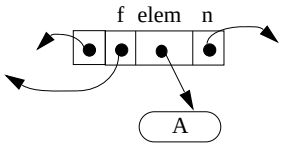
ADT List = interface PositionalSequence i JDSL

```
package jdsl.core.api;

public interface PositionalSequence
        extends PositionalContainer {

// Position aksess _____________________          implem.

/**   @return posisjonen til første elementet i sekvensen
 *    første elem fåes ved first().element() */
  Position first();                                  O(1)

/**   @return posisjon til siste elementet i sekvensen */
  Position last();                                   O(1)

/**   @param v en posisjon i sekvensen
 *    @return posisjon til elementet rett før v i sekvensen
 *    @exception BoundaryViolationException om v==first() */
  Position before(Position v);                       O(1)

/**   @param v en posisjon i sekvensen
 *    @return posisjon til elementet rett etter v i sekvensen
 *    @exception BoundaryViolationException om v==last() */
  Position after(Position v);                        O(1)

// element håndtering _____________________
/**   @return posisjon til det innsatte elementet */
  Position insertFirst(Object e);                    O(1)

/**   @return posisjon av det innsatte elementet */
  Position insertLast(Object e);                     O(1)

/**   sett inn elt o i sekvensen med rank rett foran det i v */
  Position insertBefore(Position v, Object e);       O(1)

/**   sett inn elt o i sekvensen med rank rett etter det i v */
  Position insertAfter(Position v, Object e);        O(1)

/**   @return Objektet som ble fjernet */
  Object remove(Position v);                         O(1)

// int size();  boolean isEmpty();                   O(1)

// Enumeration elements();                           O(n)
// Container newContainer();                         O(1)

// Enumeration positions();                          O(n)
// void swap(Position v, Position w);                O(1)
// Object replace(Position v, Object e);             O(1)
}
```

implementasjon med	*krever at*
en-/to-veis liste	Node implements Position
array	*"indeks"* implements Position

PositionalSequence *med* ***to-veis liste***

```
public class DLNode implements Position
{  private DLNode f, n;
   private Object elem;
   private Container sam;
   public DLNode(Object o, DLNode ff, DLNode nn, Container s)
   {   elem= o; f= ff; n= nn; sam= s; }
   public Object element() { return elem; }
   public void setElement(Object o)
       { elem= o; }
   public container() { return sam; }
   public DLNode getNext() { return n; }
   public DLNode getPrev() { return f; }
   public void setNext(DLNode d) { n= d; }
   public void setPrev(DLNode d) { f= d; }
}
```



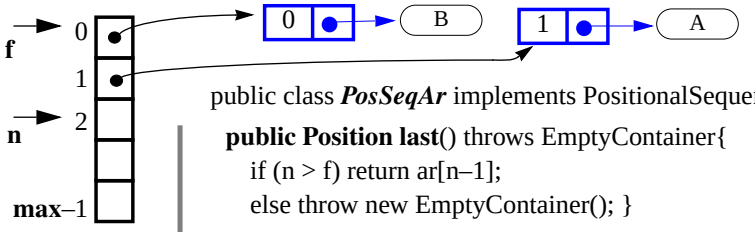
DataStruktur

```
        private DLNode top, sist;
   // antall elementer
        private int n;
```

```
public class PSdl implements PositionalSequence {
   public Position last() { return sist; }
   public Position before(Position p) {
      return ((DLNode)p).getPrev() ; }
   public Position insertFirst(Object o) {
      top= new DLNode(o, null, top, this) ;
      n++;
      return top ; }
   public Position insertBefore(Position p, Object o) {
      DLNode pp = (DLNode)p;
      DLNode ny = new DLNode(o, pp.getPrev(), pp, this);
      n++;
      return ny; }
   . . .
}
```

PositionalSequence *med* ***array (ikke sirkulær)***

Data Representasjon




```
public class ArPos
implements Position
{  private int r;
   private Object obj;
   private Container sam;
   public ArPos(
       int i, Object o, Container s)
   {   r= i; obj= o; sam= s; }
   public element() { return obj; }
   public rank() { return r; }
   public setElement(Object o) { obj= o; }
   public container() { return sam; }
   public setRank(int i) { r= i ; }
}
```

Data Struktur og Invariant

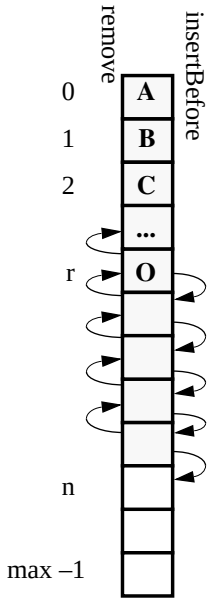
private ArPos[N] ar;
private int max; // *størrelse til ar*
private int f, n; // *første–neste ledig*

f = 0, f <= n

isEmpty hviss f = n ; size = n

ArPos[f...n] er fylt med alle elementene

```
public class PosSeqAr implements PositionalSequence {
  public Position last() throws EmptyContainer{
     if (n > f) return ar[n–1];
     else throw new EmptyContainer(); }
  public Position before(Position p) throws InvalidPosExc
  {  if ( check(p).rank()==0) throw new  ...Exception();
     return ar[ check(p).rank()–1 ] ; }
  public Position insertLast(Object o) {
     ar[n] = new ArPos(n,o,this); n++; }
  public Position insertBefore(Position p, Object o)
                throws InvalidPositionExc {
     int r = check(p).rank();
     for (int k= n; k>r; k--) {
          ar[k]= ar[k–1]; ar[k].setRank(k); }
     ar[r]= new ArPos(r,o,this);
     n++ ;  return ar[r]; }    }
  private ArPos check(Position p) throws InvalidPositionExc {
     if (! p instanceOf ArPos) throw new InvalidPositionExc();
     ArPos a = (ArPos)p;
     if (p.rank() >= 0 && p.rank() <= n) return a;
     else throw new InvalidPositionExc();    }
```



Implementasjoner av PositionalSequence

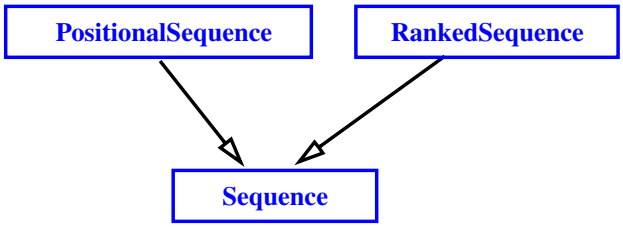
kompleksitet operasjon	Array	en-veis liste	to-veis liste
size, isEmpty	1	1	1
newContainer	1	1	1
elements, positions	n	n	n
replace, swap	1	1	1
first	1	1	1
last	1	n	1
before	**1**	**n**	**1**
after	1	1	1
insertFirst	n/1(sirkulær)	1	1
insertLast	1	n	1
insertBefore	**n**	**n**	**1**
insertAfter	**n**	**1**	**1**
remove	**n**	**n**	**1**

Sequence ADT

size, isEmpty
elements, positions
newContainer
replace, swap

first, last
before, after
insertFirst/-Last
insertBefore/-After
remove

elemAtRank
replaceElemAtRank
insertElemAtRank
removeElemAtRank



Position atRank(int r)
int rankOf(Position p)

```
package jdsl.core.api ;
public interface Sequence
        extends PositionalSequence {
/** @param r rank
    @return posisjon til element med rank r
    @exception BoundaryViolationException
  Position atRank(int r)
/** @param p posisjon
    @return rank til element i posisjonen p
    @exception InvalidPosition
  int rankOf(Position p)
/** @param r rank
    @param o objektet som skal settes inn
    @return posisjonen til det innsatte objektet
    @exception BoundaryViolationException
  Position insertAtRank(int r, Object o)
/** @param r rank
    @return objektet som ble fjernet
    @exception BoundaryViolationException
  Object removeAtRank(int rank)
}
```


Implementasjoner av Sequence

	kompleksitet operasjon	Array	en-veis liste	to-veis liste
Container	size, isEmpty	1	1	1
	newContainer	1	1	1
PositionalContainer	elements, positions	n	n	n
	replace, swap	1	1	1
PositionalSequence	first	1	1	1
	last	1	n	1
	before	1	n	1
	after	1	1	1
	insertFirst	n/1(sirkulær)	1	1
	insertLast	1	n	1
	insertBefore	**n**	**n**	**1**
	insertAfter	**n**	**1**	**1**
	remove	**n**	**n**	**1**
Sequence	rankOf	1	1	1
	atRank	**1**	**n**	**n**
	insertAtRank	n	(n)	(n)
	removeAtRank	n	(n)	(n)

Sortering

- SelectionSort, MergeSort, ...
- *er en operasjon på strukturer med rekkefølgen på Posisjoner*

total ordning: **for alle** Posisjoner p, q : p < q **eller** q < p **eller** q = p

array, liste, RankedSequence, ...

Sequence ADT *uttrykker denne totale ordningen med operasjoner before/after*

- *videre, må også elementene lagret i strukturen stå i en*

total ordning: **for alle** Elementer a, b : a < b **eller** b < a **eller** a = b

Denne uttrykkes abstrakt (generelt) med

forskjellige implementasjoner vil da definere forskjellige totale ordninger for forskjellige type objekter

```
public interface Comparator
{       /** @return true iff a < b; false otherwise */
boolean isLessThan (Object a, Object b)
        /** @return true iff a > b; false otherwise */
boolean isGreaterThan (Object a, Object b)
        /** @return true iff a == b; false otherwise */
boolean isEqualTo (Object a, Object b)
        /** @return true iff a <= b; false otherwise */
boolean isLessThanOrEqualTo (Object a, Object b)
        /** @return true iff a >= b; false otherwise */
boolean isGreaterThanOrEqualTo (Object a, Object b)
}
```


Generisk seleksjon-sort av **Sequence**

```
for (k=0,1,2...< n) {    // n= S.size()
   int m = k;
   for (j= k+1...< n)
      if (S[j] < S[m]) m = j;
   S.swap( m, k )
}
```

1+2+...+(n-1)+n = $\boldsymbol{O}(n^2)$

en algoritme er generisk hviss alle klasse-parametre er ADTer : interface eller Object

```
void SS1(Sequence S, Comparator C)  {
  int k, j, min;
  int n = S.size() ;
  for (k=0; k<n; k++)  {
    min = k ;
    em = S.atRank(min).element();
    for (j= k+1; j<n; j++)
      if (C.isLessThan(
        S.atRank(j).element(), em ))
      {  min = j;
        em = S.atRank(min).element(); }
    S.swap( S.atRank(min), S.atRank(k) ) ;
}
```

kan brukes kun når **atRank**(**r**) *er* $\boldsymbol{O}(1)$ *;*

er den $\boldsymbol{O}(n)$ *får vi SS1 =* $\boldsymbol{O}(n^3)$ *!!!*

```
void SS2(Sequence S, Comparator C)
  Position k, j, min;
  k= S.first();
  while ( k != S.last() )  {
    min = k;  j = k;
    while ( j != S.last() ) {
      j = S.after(j);
      if (C.isLessThan(j.element(), m.element()))
        min = j;
    }
    S.swap(min, k) ;
    k = S.after(k) ;
} }
```

first(), **last**(**r**), **after**(**p**) *kan alltid fåes* $\boldsymbol{O}(1)$ *;*

SS2 = $\boldsymbol{O}(n^2)$ *!!!*

Generisk **boble-sortering** av Sequence

```
// n= S.size()
for (k=0,1...< n) {     S[0...k-1] er sortert og
                        har k minste elem. fra S
   int m = k;
   for (j= k+1...< n) {   S[m] er minst i S[k...j-1]
      if (S[j] < S[m])
         m = j;
   }                      S[m] er minst i S[k...n–1]
   S.swap( m, k )
}                       S[0...n–1] er sortert
                        //og har n minste elem. fra S
```

```
// n= S.size() – 1
for (k=n...> 0) {     S[k+1...n] er sortert og har
                      n–k største elem. fra S

   for (j= 0...<k) {          S[j] er størst i S[0...j]
      if (S[j+1] < S[j])
         S.swap( j, j+1 )
   }                          S[k] er størst i S[0...k]

}                       S[1...n] er sortert
                        og har n største elem. fra S
```

```
void BubbleSort(Sequence S, Comparator C)  {
   Position p1, p2 ;
   for ( k = S.size()–1; k>0; k--)   {
      p1 = S.first() ;
      for ( j = 0; j<k; j++) {
         p2 = S.after(p);
         if (C.isLessThan(p2.element(), p1.element()))
            S.swap(p1,p2);
         p = a;
} } }
```


Korrekthet av boble-sortering

BubbleSort(int[] A) { // n er største indeks i tabellen

INN1: k=n : A[n+1..n] ...0 <= r < s <= n & s > n ...ingen slik s

1:***for (k = n, k > 0, k--)*** {

LI1: 0 <= r < s <= n & s > k → A[r] <= A[s]

dvs. A[k+1...n] er sortert og har n-k største elementer

INN2: j=0 : A[0] er størst i A[0...0]

2: ***for (j = 0, j < k, j++)*** {

LI2: 0 <= t < j <= n → A[t] <= A[j]

dvs. A[j] er størst i A[0...j]

if (A[j] > A[j+1]) swap(A[j], A[j+1]);

j'=j+1: er A[j'] størst i A[0...j'] ?

hvis A[j] <= A[j+1] & LI2 → LI2'

ellers A[j'] = A[j] > A[j+1] = A[j'–1] & LI2 → LI2'

LI2': j'=j+1: 0 <= t < j' <= n → A[t] <= A[j']

}

UTG2: LI2 & ***j=k*** gir: 0 <= t < k <= n → A[t] <= A[k]

dvs. A[k] er størst blant A[0...k]

UTG2 & LI1 gir: A[0...k–1] <= A[k] <= A[k+1...n] →

LI1': k'= k–1: 1 <= r < s <= n, s > k' → A[r] <= A[s]

dvs. A[k'...n] er sortert og har n-k' største elementer

}

UTG1: **LI1 & *k=0*** gir **0 <= r < s <= n & s > 0 → A[r] <= A[s]**

dvs. : A[1...n] er sortert og har n-1 største elementer,

men da har A[0] minste element <= A[r] for 0 < r <= n

(spesielt, for r=1 : A[0] <= A[1])

Oppsummering

- *jdsl.simple.api.Container*
- *jdsl.core.api.Container*
- *java.util.Enumeration*
- *jdsl.simple.api*
  - *Stack*
  - *Queue*
  - *Deque*
  - *RankedSequence*
- *jdsl.core.api*
  - *Position*
  - *PositionalContainer*
  - *PositionalSequence*
  - *Sequence – implementert med*
    - *– java.util.Vector*
    - *– array*
    - *– en-veis liste*
    - *– to-veis liste*

- *organisering av ADTer*
  - *hierarki av interface burde avspeile alle relasjoner mellom begrepene*
- *Adaptor pattern: ADT-2 kan brukes for en "generisk" implementasjon av en annen ADT-1*
  - *relativ kompleksitet, dvs kompleksitet til ADT-1 avhengig av implementasjon av ADT-2*
- *riktige abstraksjoner kan være vanskelig å finne*
  - *Position*
- *forskjellige implementasjoner av samme ADT*
  - *vurdering av implementasjoner opp mot hverandre*
- *"generisk" algoritme bruker kun ADTer*
  - *grensesnitt informasjon om parametre*
  - *og derfor kan brukes med vilkårlige implementasjoner*
- *sorterings algoritmer*
  - *seleksjon–, merge–, bobble–sortering*