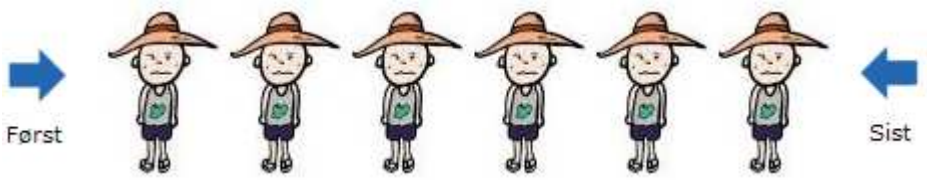
Algoritmer og datastrukturer
Kapittel 4 – Delkapittel 4.3

4.3 En toveiskø (deque)

4.3.1 Grensesnittet Toveiskø

En stakk kan godt ses på som en kø der vi kun legger inn i og tar ut fra den ene enden av køen. I en vanlig kø derimot legger vi inn i den ene enden (bakerst) og tar ut fra den andre enden (forrest). Begge disse ideene kan forenes i én type kø, dvs. en kø der vi både kan legge inn i begge ender og ta ut fra begge ender. En slik kø skal vi her kalle en *toveiskø*. Det engelsk navnet for dette er *deque* og ordet er en forkortelse for **d**ouble **e**nded **que**ue.

En *toveiskø* må ha metoder med navn som klart indikerer i hvilken ende av køen vi opererer. Vi skal her bruke ordene *først* og *sist* til dette formålet.



Figur 4.3.1: I en toveiskø kan innlegging og uttak skje i begge ender

Flg. grensesnitt inneholder to versjoner av hver av metodene for innlegging, kikking og uttak. Dvs. en versjon for hver ende av køen:

```
public interface Toveiskø<T>          // eng: Deque
{
  public void leggInnFørst(T verdi);  // legger inn først i køen
  public void leggInnSist(T verdi);   // legger inn sist i køen
  public T kikkFørst();               // ser på den første
  public T kikkSist();                // ser på den siste
  public T taUtFørst();               // tar ut den første
  public T taUtSist();                // tar ut den siste
  public boolean tom();               // er køen tom?
  public int antall();                // antall i køen
  public void nullstill();            // nullstiller køen

} // interface Toveiskø
```

***Programkode* 4.3.1**

Stakk: En toveiskø kan fungere som en stakk ved at metodene *leggInnFørst*, *kikkFørst* og *taUtFørst* brukes som *push*, *peek* og *pop*. Eller en kan bruke *leggInnSist*, *kikkSist* og *taUtSist*. Begge ender av toveiskøen skal fungere på samme måte.

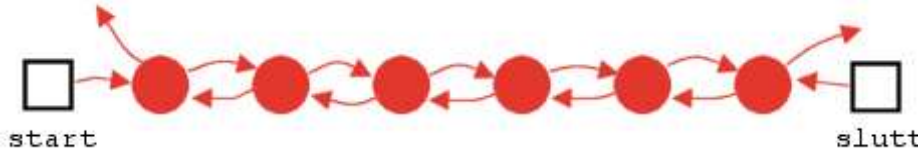
Kø: En toveiskø kan fungere som en vanlig kø ved at *leggInnSist*, *kikkFørst* og *taUtFørst* brukes som *push*, *peek* og *pop*. Eller eventuelt *leggInnFørst*, *kikkSist* og *taUtSist*.

Oppgaver til Avsnitt 4.3.1

1. Sjekk hvilke metoder grensesnittet *Deque* i java.util har.

4.3.2 Lenket toveiskø

En enkel måte å konstruere en toveiskø på er å bruke en toveis pekerkjede (dobbeltlenket liste) som intern datastruktur. Der brukes vanligvis hode og hale som navn på starten og slutten av listen. Her velger vi imidletid *start* og *slutt* siden det passer bedre i en toveiskø.



Figur 4.3.2 a): En toveis pekerkjede mes start og slutt

Dette gir oss flg. datastruktur for klassen *LenketToveiskø*:

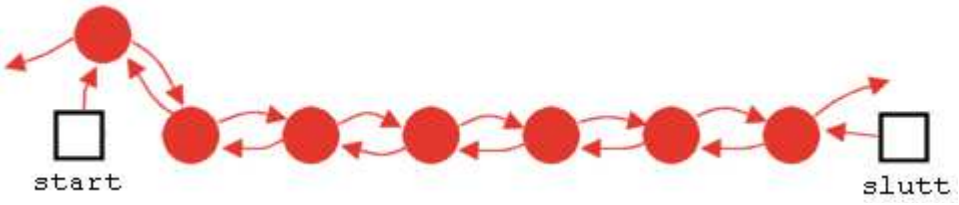
```
import java.util.*;

public class LenketToveiskø<T> implements Toveiskø<T>
{
  private static final class Node<T>     // en indre nodeklasse
  {
    T verdi;                             // nodens verdi
    Node<T> forrige;                     // peker til forrige node
    Node<T> neste;                       // peker til neste node

    Node(T verdi, Node<T> forrige, Node<T> neste)  // konstruktør
    {
      this.verdi = verdi;
      this.forrige = forrige;
      this.neste = neste;
    }
  } // class Node

  private Node<T> start;                 // køens start
  private Node<T> slutt;                 // køens slutt
  private int antall;                    // antall i køen

  public LenketToveiskø()                // standardkonstruktør
  {
    start = slutt = null;
    antall = 0;
  }

  // her skal resten av metodene inn

} // class LenketToveiskø
```

Programkode 4.3.2 a)

Innlegging i en tom kø gir en kø med kun én node og både *start* og *slutt* må dermed peke på denne noden. Nodens forrige- og nestepeker settes til *null*:

Figur 4.3.2 b): Til venstre en tom kø - til høyre en ny node

En ny node først i en ikke-tom kø får vi til ved å la forrigepeker i den første noden peke til den nye. Nestepeker i den nye må så peke til det som var den første og forrigepeker i den nye må peke til *null*. Det hele blir korrekt når pekeren *start* settes til å peke på den nye:



Figur 4.3.2 c): Innlegging først i en toveiskø som ikke er tom

Koden for innlegging først i en lenket toveiskø blir dermed slik:

```
public void leggInnFørst(T verdi)
{
  if (antall == 0)   // køen er tom
    start = slutt = new Node<T>(verdi,null,null);
  else
    start = start.forrige = new Node<T>(verdi,null,start);

  antall++;
}
```

Programkode 4.3.2 b)

Metoden *kikkFørst* er rett frem. I en ikke-tom kø skal verdien i første node returneres. Se *Oppgave* 3. Metoden *taUtFørst* er også mer eller mindre rett frem. Hvis køen ikke er tom, skal første node fjernes. Men først må verdien tas vare på. Dernest flyttes *start* til neste node. Hvis køen har nøyaktig én node fra før, vil den etterpå bli tom. Dermed må *slutt* settes til *null*. Hvis ikke, må forrigepeker i noden som nå blir første node, settes til *null*:

```
public T taUtFørst()
{
  if (antall == 0)  // køen er tom
    throw new NoSuchElementException("Køen er tom!");

  T temp = start.verdi;
  start.verdi = null;
  start = start.neste;

  if (antall == 1) slutt = null;
  else start.forrige = null;

  antall--;
  return temp;
}
```

Programkode 4.3.2 c)

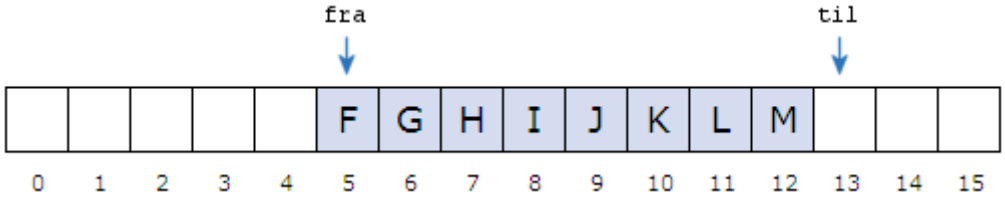
Oppgaver til Avsnitt 4.3.2

1. Legg grensesnittet `Toveiskø` på filen `Toveiskø.java` f.eks. under `hjelpeklasser`.
2. Legg klassen `LenketToveiskø` på filen `LenketToveiskø.java` og kopier inn metodene `leggInnFørst` og `taUtFørst`.
3. Kod resten av metodene i grensesnittet `Toveiskø`. Hvis «først»-metodene «speilvendes», får vi «sist»-metodene. Lag også en *toString*-metode.

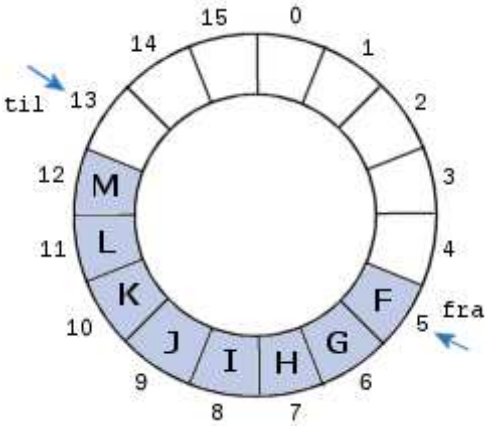
4.3.3 Sirkulær toveiskø

En sirkulær toveiskø bruker samme idé som i den sirkulære køen i *Avsnitt* 4.2.2. Vi kan derfor starte med klassen `TabellKø` og så bytte ut navnet `TabellKø` med navnet `TabellToveiskø` alle aktuelle steder. Videre bytter vi ut navnene på metodene `leggInn()`, `kikk()` og `tauT()` til henholdsvis `leggInnSist()`, `kikkFørst()` og `taUtFørst()`. Metodene `utvidTabell()`, `antall()`, `tom()`, `nullstill()` og `toString()` kan brukes som de er.

Det som må kodes er metodene `leggInnFørst()`, `kikkSist()` og `taUtSist()`.



Figur 4.3.3 a) : To piler markerer starten og slutten på køen



Figur 4.3.3 b) : En sirkulær tabell

Det å legge inn en ny verdi først betyr at den skal legges på plassen til venstre for `fra` eller hvis vi ser på sirkelen, på plassen vi får ved å gå én enhet mot klokken. På begge figurene blir det posisjon 4.

Her må vi imidlertid passe på tilfellet at `fra` er 0. Én enhet mot klokken blir da posisjon 15, eller generelt lik `a.length - 1` hvis tabellen heter a.

Vi må tenke på samme måte når vi skal se på eller ta ut den siste verdien. Den siste verdien ligger i posisjon `til - 1` eller posisjon `a.length - 1` hvis `til` er 0. Det er tomt hvis `fra` og `til` er like.

```
public void leggInnFørst(T verdi)
{
  if (fra == 0) fra = a.length - 1; else fra--;
  a[fra] = verdi;
  if (fra == til) a = utvidTabell(2*a.length);  // dobler tabellen
}

public T kikkSist()
{
  if (fra == til) throw new NoSuchElementException("Køen er tom!");
  if (til == 0) return a[a.length - 1];
  else return a[til - 1];
}

public T taUtSist()
{
  if (fra == til) throw new NoSuchElementException("Køen er tom!");
  if (til == 0) til = a.length - 1; else til--;
  T temp = a[til];
  a[til] = null;
  return temp;
}
```

Programkode 4.3.3 a)

Oppgaver til Avsnitt 4.3.3

1. Lag klassen TabellToveiskø.

4.3.4 Deque i java.util

Java har flg. grensesnitt for en toveiskø:

```
public interface Deque<T> extends Queue<T>
{
  public void addFirst(E e);       // legger inn forrest i køen
  public void addLast(E e);        // legger inn bakerst i køen
  public boolean offerFirst(T t);  // legger inn forrest i køen
  public boolean offerLast(T t);   // legger inn bakerst i køen
  public T peekFirst();            // ser på den første (null hvis tomt)
  public T peekLast();             // ser på den siste (null hvis tomt)
  public T getFirst();             // ser på den første (unntak hvis tomt)
  public T getLast();              // ser på den siste (unntak hvis tomt)
  public T pollFirst();            // tar ut den første (null hvis tomt)
  public T pollLast();             // tar ut den siste (null hvis tomt)
  public T removeFirst();          // tar ut den første (unntak hvis tomt)
  public T removeLast();           // tar ut den siste (unntak hvis tomt)

  // + metoder som arves fra Queue<T> (og Collection<T>)

} // interface Deque
```

Programkode 4.3.4 a)

Både ArrayDeque og LinkedList kan brukes som en toveiskø. Her er et eksempel der ArrayDeque brukes:

```
Deque<Integer> toveiskø = new ArrayDeque<>();

toveiskø.offerFirst(3);
toveiskø.offerLast(4);
toveiskø.offerFirst(2);
toveiskø.offerLast(5);
toveiskø.offerFirst(1);
toveiskø.offerLast(6);

while (!toveiskø.isEmpty())
{
  System.out.print(toveiskø.pollFirst() + " ");
}

// Utskrift 1 2 3 4 5 6
```

Programkode 4.3.4 b)