Uke 4,
Forelesning 1

HUSK – *Hittil…*

- Essensen i faget: Effektive algoritmer
- Matematiske forutsetninger
- Introduksjon til metodekall og rekursjon
- Introduksjon til kombinatoriske søk, kombinasjoner og permutasjoner
- Introduksjon til analyse av algoritmer

Uke 1 og Uke 2

- Introduksjon til ADT'er
- Introduksjon til Lister

Uke 3.1

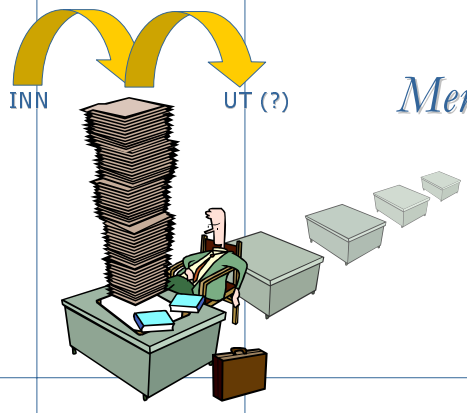
- Mer om lister…
- Introduksjon til Stabler
- Introduksjon til Køer

Uke 3.2

OVERSIKT – *Uke 4, Forelesning 1 (W4.L1)*

- Vil fortsetter fra forrige gang
  - TEMA #3: Stabler (Engelsk: Stacks)
  - TEMA #4: Køer
  - Radix-sort (Java eksempel)
- NB!
  Stabel ADT: Kap. 3.3 i boka (Mark Allen Weiss),
  Kø ADT: Kap. 3.4 i boka,

TEMA #3: STABLER – *Fortsetter…*

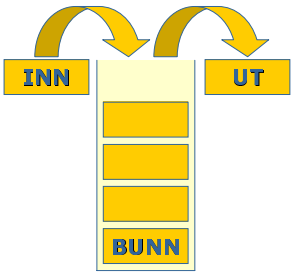


Mer om stabler...

STABEL ADT – *Definisjon, eksempler...*

- ADT'en stabel modellerer det vi tenker på som en stabel av et eller annet, for eksempel tallerkener (tenk Fredrikke).
- En stabel er en variant av en liste, men i en stabel har vi bare lov til å sette inn og slette elementer fra en bestemt ende (og samme ende) av listen.
- Det kalles også for en Last-In-First-Out (LIFO) liste.

- Eksempler:
  - Metodekall i Java og Java kall-stabelen
  - Enkle kalkulatorer (HP)
  - En full buss, der sjåføren nekter å tillate avstigning forfra



STABEL ADT – *ADT grensesnittet*

```
public interface StabelInterface
{
  // Legge et element på toppen av stabelen
  void push(Object x);
  // Fjerne et element fra toppen av stabelen
  void pop();
  // Returnere elementet på toppen av stabelen
  Object top();
  // Lage en ny stabel/tømme stabelen
  void create();
  // Sjekke om stabelen er tom
  boolean isEmpty();
}
```

```
public interface StabelInterface-2
{
    // Legge et element på toppen av stabelen
    void push(Object x);
    // Popp ut (fjern) et element fra toppen og returner det
    Object pop();       // Også returner elementet som fjernes.
    // Returner indeks for toppen av stabelen
    int top();                          // Fortell hvor toppen er nå.
    // Lage en ny stabel/tømme stabelen
    void create();
    // Sjekke om stabelen er tom
    boolean isEmpty();
}
```

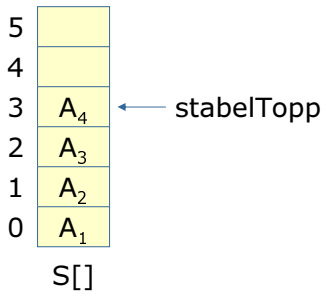

Array-implementasjon av en stabel:

Brukes ofte hvis antall elementer på stabelen alltid er begrenset.

```
public void push(Object x)
{       stabelTopp++;
        S[stabelTopp] = x;
}
```

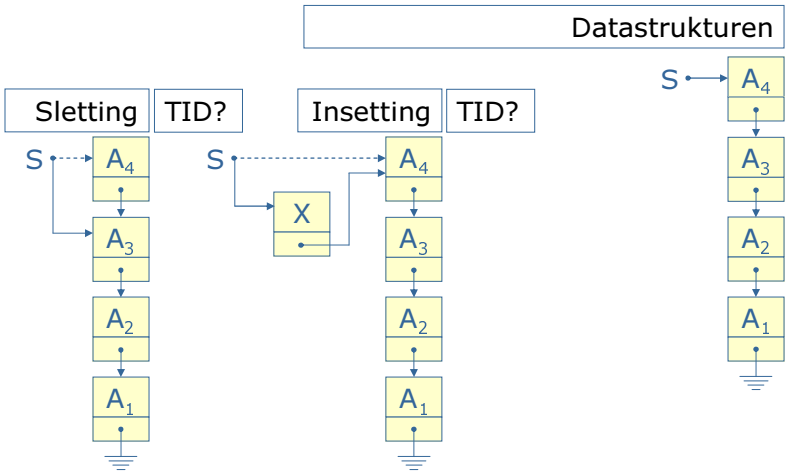
```
public void pop()
{       stabelTopp--;
}
```

```
public Object top()
{       return S[stabelTopp];
}
```



I tillegg: Eventuell feilhåndtering

Pekerkjede-implementasjon av en stabel:




```
public class PekerStabel implements StabelInterface
{
   private class Node
   {  Object element;
      Node neste = null;
   }

   private Node topp = null;

   public void push(Object x)
   {  Node ny = new Node;
      ny.element = x;
      ny.neste = topp;
      topp = ny;
   }

   public void pop()
   {  if (topp != null) topp = topp.neste;
   }

   public Object top()
   {  if (topp != null) return topp.element;
      else return null;
   }

   public void create()
   {  topp = null;
   }

   public Boolean isEmpty()
   {  return topp == null;
   }
}
```

STABEL OPERASJONER – *Tidsforbruk...*

- Uansett hvor mange elementer vi har på stabelen, er vi garantert konstant tidsforbruk, det vil si $O(1)$ for alle operasjonene.

  Dette gjelder uansett om implementasjonen bruker en array eller en pekerkjede!

  Bedre er det ikke mulig å få det, derfor er stabelen en veldig populær ADT.

- I tillegg kan mange "naturlige" problemer løses ved hjelp av en stabel.
- Typisk bruksmønster er mange stabel-operasjoner, men få elementer på stabelen om gangen.
- På mange maskiner kan disse oversettes til kun to-tre instruksjoner i maskinkode.

STABEL OPERASJONER – *Tidsforbruk, eksempel*

- **Eksempel: Beregning av postfiks uttrykk**

- På de fleste kalkulatorer skriver man inn regneuttrykkene på såkalt **infiks** form, med operatorene **mellom** argumentene:

  4 + 2 = 6
  3 + 5 * 2 = 13
  (3 + 5) * 2 = 16

- Et alternativ er å skrive uttrykkene på postfiks form, der operatorene står **etter** argumentene. Med denne notasjonen slipper man å bruke parenteser:

  4 2 + = 6
  3 5 2 * + = 13
  3 5 + 2 * = 16

STABELBRUK – *Eksempel (fortsetter)*

Ved hjelp av en stabel er det lett å beregne et postfiks uttrykk på følgende måte:

- For hvert symbol i input:
  - Hvis symbolet er et tall, legges det på stabelen.
  - Hvis symbolet er en operator, "popper" vi to tall fra stabelen, anvender operatoren på disse to tallene og dytter svaret tilbake på stabelen.
- Hvis input var et ekte postfiks uttrykk, vil nå svaret ligge som det eneste elementet på stabelen.
- Eksempel:
  6 5 2 3 + 8 * + 3 + *

TEMA #4: KØER - *Fortsetter…*

Mer om køer…

KØ ADT – *Definisjon*

- ADT'en kø modellerer stort sett det vi tenker på som en kø...

  Som kø foran billett-luka, kø i posten (styrt ofte av kø-lapper i moderne tider), buss-kø ...

  Og om du ikke vet hva en kø er eller hva disse køer er, bør du sjekke med folkeregisteret hvor du egentlig bor...

- En kø er en variant av en liste, men hvor vi setter inn elementer i den ene enden og tar ut elementer fra den andre enden:

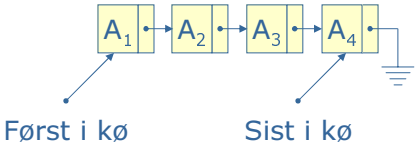
  Det er en First-In-First-Out (FIFO) liste.

KØ ADT – *Grensenittet*

```
public interface KoeInterface
{

   // Legge til et element sist i køen (enqueue)
   void settInn(Object x);

   // Fjerne elementet først i køen (dequeue)
   Object taUt();

   // Lage en ny kø/tømme køen
   void makeEmpty();

   // Sjekke om køen er tom
   boolean isEmpty();

}
```

Pekerkjede-implementasjon av en kø:

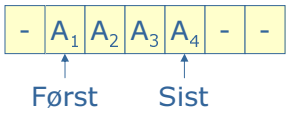
Datastrukturen:
En liste med både første- og siste-peker



TIDSFORBRUK:

- Hvordan blir det å sette inn i køen? Hva blir tidsforbruket for det?
- Hvordan blir det å ta ut fra køen? Hva blir tidsforbruket for det?

Array-implementasjon av en kø:



- Vi har to heltallsvariable som "peker" på første og siste element i køen.
- PROBLEM: Hvis køen lever lenge og det er mange inn/ut-operasjoner, kan vi nå enden av array'en selv om det bare er noen få elementer i køen!
- LØSNING: Bruk en såkalt **sirkulær** array, hvor "pekerne" flyttes tilbake til begynnelsen etter at de har nådd enden av arrayen.
- Det kan bli litt fiklete kode, men operasjonene er veldig tidseffektive.

Eksempel: Palindromer

- Palindromer er ord/uttrykk/setninger som er like forlengs og baklengs.
  - Kjente eksempler:
    Otto
    ABBA
    Radar
  - Svensk: Ni talar bra latin
    Dansk: En af dem der red med fane
    Engelsk: If I had a Hi-fi
- Hvordan sjekke om en gitt tekst-streng er et palindrom?

En lekende løsning

1. Les tekst-strengen inn i både en stabel og en (FIFO-)kø.
2. Sammenlign innholdet i stabelen og køen tegn for tegn.

```
public void sjekkPalindrom( String setning, StabelInterface stabel,
                                    KoeInterface koe )
{  while (!setning.equals(""))                  // Lag både en kø og en stabel
   {  char bokstav = setning.charAt(0);         // av setningen...
      setning = setning.substring(1);
      if (!Character.isWhitespace(bokstav))
      {  koe.settInn(new Character(bokstav));
         stabel.push(new Character(bokstav));
      }
   }
   while (!koe.isEmpty())                       // Og test om de er like!
   {  if (!koe.taUt().equals(stabel.pop()))
      {  System.out.println("Dette er ikke et palindrom.");
         System.exit(1);
      }
   }
   System.out.println("Gratulerer, dette er et palindrom");
}
```

LISTER (fra forrige uke) – *Radix i bøtter og spann #1*

```
public class Radix
{  // Bøtter implementert som "array av Liste"...
   static Liste[] botter;

   public static void main (String[] args)
   { int[] input = {64, 8, 216, 512, 27, 729, 0, 1,343, 125};
     // Initialiserer bøtte-listene
     botter = new Liste[10];
     for (int i = 0; i < 10; i++)
     {   botter[i] = new Liste();
     }

     // Plasserer input i bøttene etter siste siffer
     for (int i = 0; i < input.length; i++)
     {   int tall = input[i];
         int indeks = tall%10;
         botter[indeks].insert(tall);
     }

     // Fortsetter... (klassen avslutter på neste side).
```


LISTER – *Radix i bøtter og spann #2*

```
public class Radix
{  //Fortsetter fra forrige foil...

   public static void main (String[] args)
   { // Fortsetter fra forrige foil...

     // ***********************************
     // Her skal selve sorteringen inn - se neste foil
     // ***********************************

     for (int i = 0; i < 10; i++)
     {  System.out.print("Bøtte " + i + ": ");
        botter[i].print();
     }
   }
}
```

LISTER – *Radix i bøtter og spann #3*

```
// Selve sorteringen... 3 Pass... Skal inn i klassen Radix
for (int pass = 2; pass <= 3; pass++)
{  //Går gjennom hver bøtte og sorterer i henhold til sifferet
   //på plassen pass bakfra.
   for (int i = 0; i < 10; i++)
   { IntNode node = botter[i].forste();
     while (node != null)
     {  int tall = node.element;
        // Finner de pass bakerste sifferne
        int siffer = tall%(int)Math.pow(10,pass);
        // Plukker ut det første av disse
        siffer = siffer/(int)Math.pow(10, pass-1);
        if (siffer != i)
        {  botter[i].remove(tall);
           botter[siffer].insert(tall);
        }
        node = node.neste;
     }
   }
}
```


NESTE GANG – *Uke 4, Forelesning 2 (W4.L2)*

- Neste gang, dvs. uke 4, forelesning 2 (W4.L2): Trær!
- Binære trær...
  Senere binære søketrær...
  og så litt annet (som hashing o.l.) og til slutt B-trær

  Trær i ca. 5 x 2 timer (2,5 uker)...