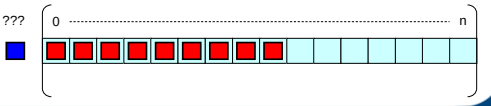
Algoritmer och data strukturer
-Länkade listor

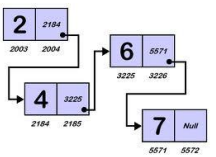
Statisk minnesallokering

- Statiska minnesstrukturer (Arrayer)
  - För stora / För små
  - ”Jobbiga” *add(index i) / remove(index i)* operationer

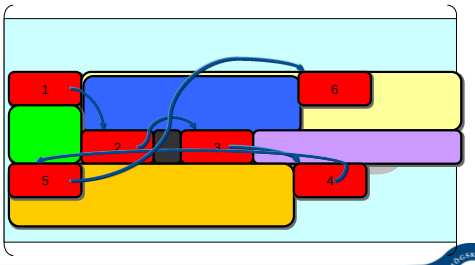


Dynamisk minnesallokering

- Separata *”noder”*
  - Allokeras vid behov
  - Avallokeras när de inte längre behövs
  - Håller information om:
    - Nodens data
    - Adress till nästa nod

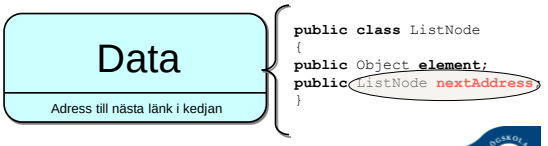


En kedja av data



En länk (nod) i kedjan

- Olika sorters listor
  - *Enkellänkade* listor
  - Dubbellänkade listor
  - Cirkulära länkade listor



Klassen ListNode / av Object

```
class ListNode
{
    public Object   element;
    public ListNode nextAddress;

    public ListNode( Object theElement )
    {
        element= theElement;
        nextAddress= null;
    }

    public ListNode( Object theElement, ListNode n )
    {
        element = theElement;
        nextAddress     = n;
    }
}
```

En nod INNEHÅLL

Konstruerare 1, skapar en ”ej länkad nod ”

Konstruerare 2 , skapar en länkad nod

Klassen LinkedList

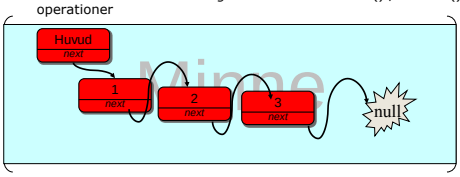
```
class LinkedList
{
    private ListNode head;

    public LinkedList ( )
    {
        head = new ListNode( null );
    }

// metoder som : add(), remove(), find()……
```

Listans huvud

Konstruerare , skapar en ”tom” nod som head ”känner till”

Enkellänkade listor

- Varje länk har en referens till nästa länk i listan
- Kräver kännedom om *”föregående länk”* vid insert() / remove() operationer



Huvud / Svans

- När är listan slut?
  - När sista länkens nextAddress = `null`
  - Testa på `null` under traverseringen

```
public void print()
{
    ListNode temp=head.nextAddress;
    while( temp!=null ){
    System.out.println(temp.element);
    temp=temp.nextAddress;
}
```

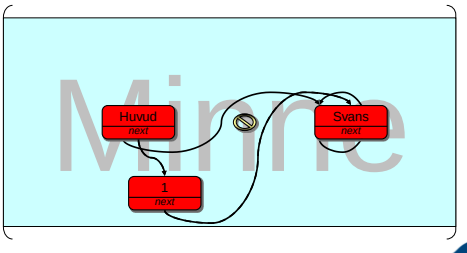
- Bättre sätt – Huvud / Svans!
  - *"Dummy"*-länkar som aldrig tas bort eller bär värden

List traversering

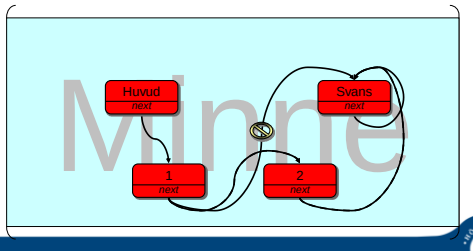
- Ingen möjlighet till indexering som hos statiska arrayer!
- *"Länk för länk"* med start på första länken tills rätt position hittad!

```
public ListNode getNode(Object x)
{
  ListNode temp = head.next;

  while(temp.element != x)
    temp = temp.nextAddress;

  return temp;
}
```


Huvud / Svans



OBS! Först länka mot svansen sedan bryt länken från huvud



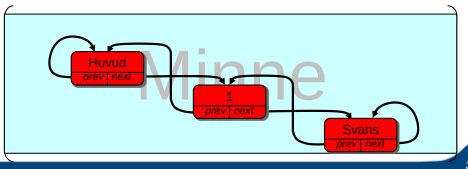
Vanliga operationer på länkade listor

- *add*()
- add(int index)
- *remove*()
- *find*()
- *get*(int index)
- *iterator*()
- contains()
- size()

I java gäller operationerna från List interface

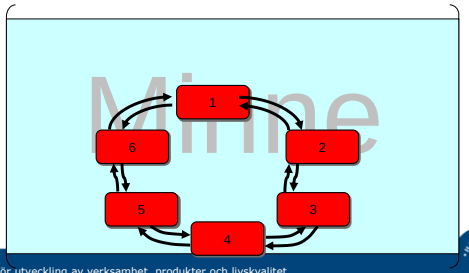
Dubbellänkade listor

- Varje länk har en referens till nästa länk OCH en referens till *föregående* länk i listan.
- Förenklar listmanipulationer
  - *add*(int index)
  - *remove*()



Cirkulära listor

- Ta bort svansen och huvudet och knyt ihop liständarna



Fördelar/ Nackdelar

Effektiv när man lägger till och tar bort element

Effektiv att hålla listan sorterad

Betydligt mer effektivt vid stora datamängder genom att dirigera om pekarna istället för själva länkinnehållet

Effektiv användning av minne.

Något långsammare, pga. Många minnesallokering

Många referensmanipulationer, lite svårare vid implementera och debbug