Dagens tema

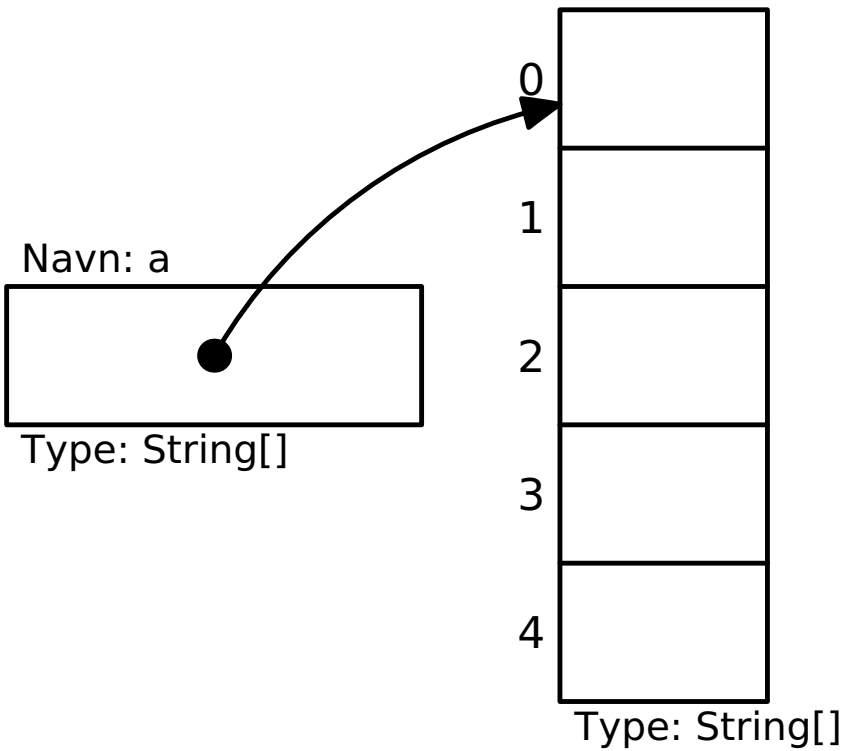
Lister og generiske klasser, del I

- Array-er og ArrayList (Big Java 6.1 & 6.8)
- Ulike lagringsformer (Collection) i Java (Big Java 15.1)
- Klasser med typeparametre («generiske klasser») (Big Java 18)
- Lage vår egen ArrayList (Big Java 16.2)
- Lenkelister (Big Java 15.2)

Hvorfor er dette pensum i IN1010?

- Det er nyttige verktøy å kjenne til.
- Som eksperter bør dere kjenne til hvordan de er bygget opp og fungerer i detalj.
- Dere skal være i stand til å skrive lignende selv.
- Dette er ypperlige eksempler på oo-programmering.

Array-er



En **array** er en sammenhengende gruppe celler i minnet.

Hva er bra og mindre bra med arrayer?

- En god notasjon: kompakt og lettforståelig:

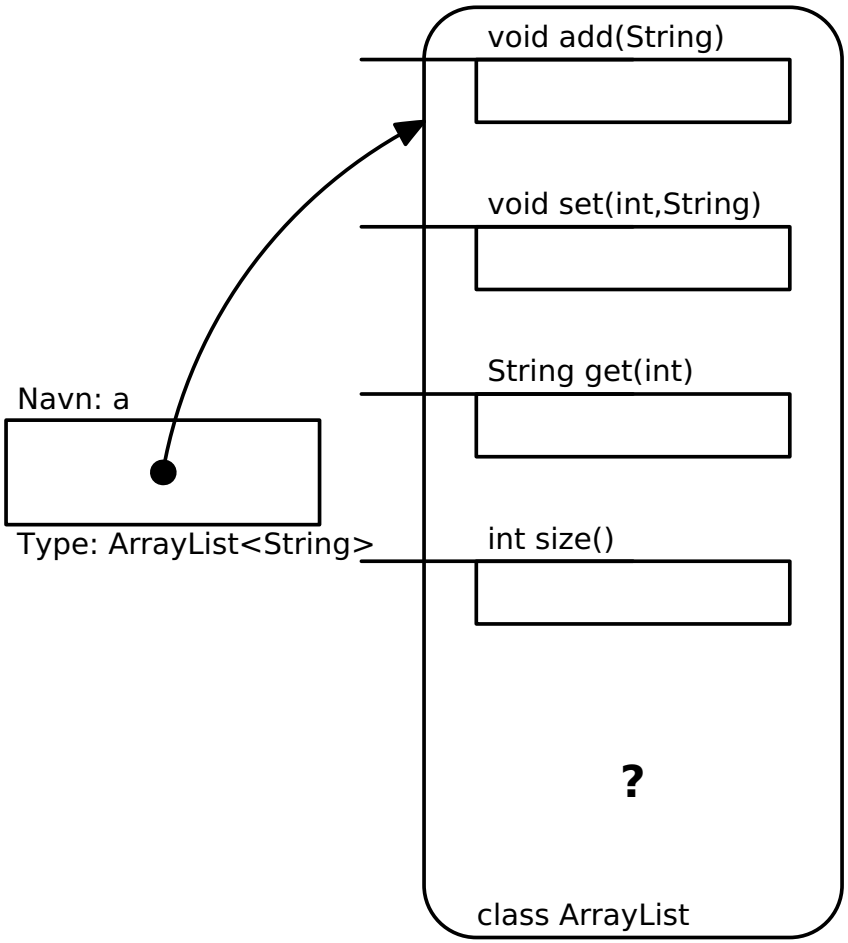
```
a[i] = a[i+1] + "*";
```

- Tar liten plass
- Raske

Men . . .

- Vi må vite størrelsen når arrayen opprettes.
- Størrelsen er uforanderlig.
- Kronglete å legge til nye verdier midt i arrayen.

ArrayList-er



ArrayList er en klasse i Java-biblioteket.

- Grensesnittet er gitt
- men implementasjonen er ukjent.

Er ArrayList noen ganger bedre enn arrayer?

- Vi trenger ikke vite størrelsen initielt.
- Størrelsen kan endres underveis.
- Enkelt å legge til nye elementer hvor som helst.

Men . . .

- Notasjonen er litt kronglete:

```
a.set(i, a.get(i+1) + "*");
```

- Ikke for primitive typer som int, char etc.[1]
- Tar mye mer plass.
- Er langsommere i bruk.

[1] Men *innkapsling* er en løsning. Vi kommer til det neste uke.

Hva gjør man i Python?

I Python har man valgt å bruke *lister* som er en mellomløsning:

- Enkel notasjon (som Javas arrayer)
- Fleksibel størrelse (som Javas ArrayList)
- Ikke så rask

Er plassforbruket viktig?

Stort sett er programmets plassforbruk helt uvesentlig, men det finnes unntak:

- Hvis man trenger ekstremt mange objekter
- Hvis datamaskinen har veldig lite minne (f eks «*Internet of things*»)
- Hvis hastigheten er avgjørende

Konklusjon: Dette må vurderes når man starter et prosjekt.

Betyr hastighet noe?

En sammenligning basert på et program som bruker Javas arrayer og ArrayList og Pythons lister aktivt:

Java array	Java ArrayList	Python liste
1,77 s	5,66 s	155,32 s

Oftest er programmets kjøretid ikke spesielt viktig, men det finnes unntak:

- Når jobben er spesielt stor og tung
- Når man trenger øyeblikkelig respons

Konklusjon: Dette må vurderes når man starter et prosjekt.

Hva bør jeg velge i Java?

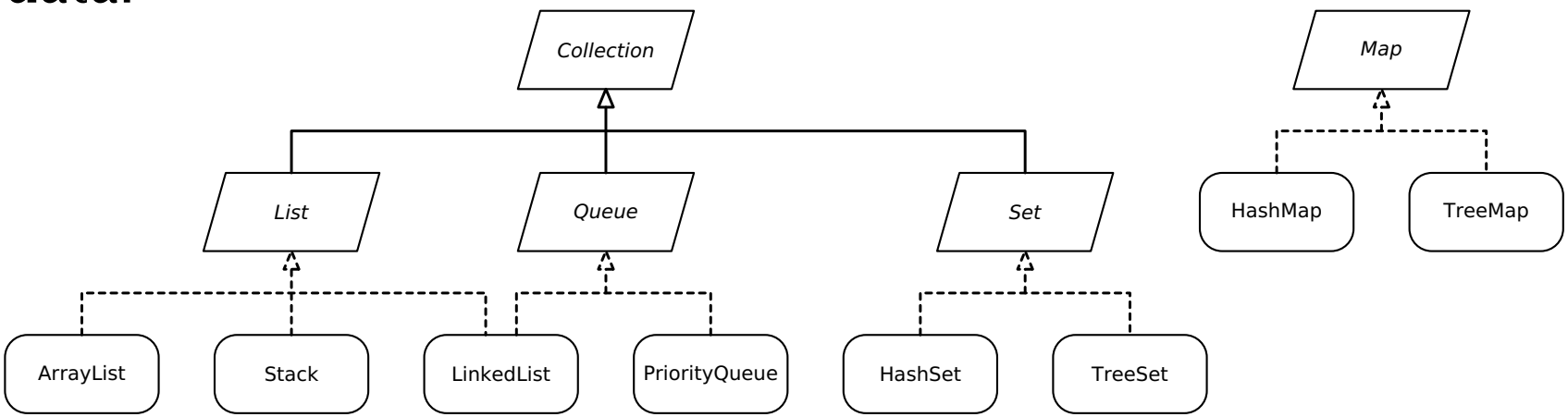
Velg array hvis

- du vet størrelsen på arrayen når den skal opprettes eller
- hastighet er viktig eller
- plass er viktig.

Velg ArrayList ellers.

Datalagring i Java

Java-biblioteket tilbyr mange ulike former for lagring av data:



Java støtter mange ulike former for datalagring

Collection er en samling av alle lagringsformene.

List er en sekvensiell samling.

ArrayList er en List implementert med en array.

Stack ser på elementene som en *stakk*.

LinkedList er en List implementert som en *lenkeliste*.

Queue lagrer i køordning.

PriorityQueue lagrer i en sortert kø.

Set lagrer uten noen rekkefølge.

HashSet lagrer et Set ved å bruke *hash-verdier*.

TreeSet lagrer et Set i et *tre*.

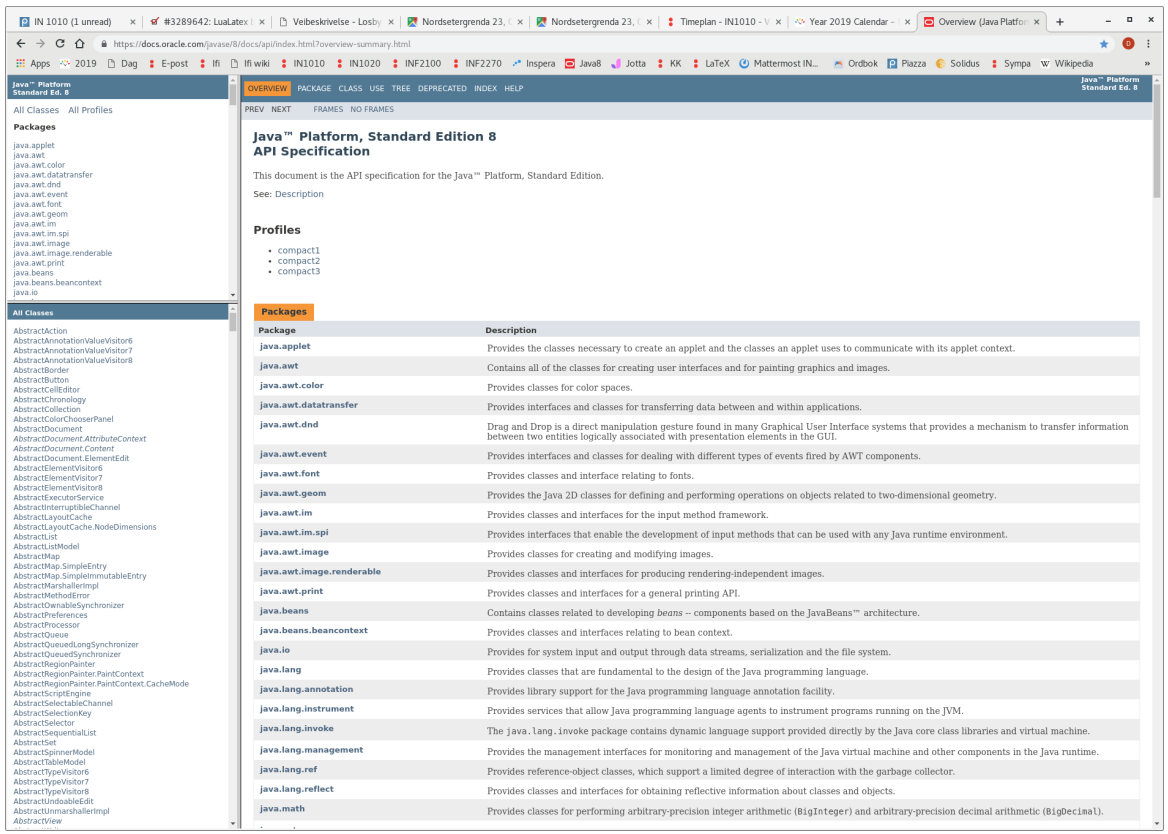
Map lagrer par av *nøkkel+verdi*.

HashMap lagrer en Map ved hjelp av *hash-verdier*.

TreeMap lagrer en Map i et *tre*.

Dokumentasjon av Java-biblioteket

https://docs.oracle.com/javase/8/docs/api/index.html?overview-summary.html er en ypperlig dokumentasjon:



Java™ Platform, Standard Edition 8
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing *beans* -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.instrument	Provides services that allow Java programming language agents to instrument programs running on the JVM.
java.lang.invoke	The `java.lang.invoke` package contains dynamic language support provided directly by the Java core class libraries and virtual machine.
java.lang.management	Provides the management interfaces for monitoring and management of the Java virtual machine and other components in the Java runtime.
java.lang.ref	Provides reference-object classes, which support a limited degree of interaction with the garbage collector.
java.lang.reflect	Provides classes and interfaces for obtaining reflective information about classes and objects.
java.math	Provides classes for performing arbitrary-precision integer arithmetic (`BigInteger`) and arbitrary-precision decimal arithmetic (`BigDecimal`).

La oss lage en liste

Før vi starter programmeringen, må vi bestemme oss for hvilke operasjoner vi ønsker oss. Hva trenger vi i en lineær lagringsstruktur?

get henter et element fra en gitt posisjon.

add utvider listen med et nytt element.

set erstatter elementet i et gitt posisjon med et nytt.

remove fjerner elementet i en gitt posisjon.

size forteller hvor lang listen vår er nå.

Start med grensesnittet

Det er ofte nyttig å starte med å definere grensesnittet i form av et interface:

Liste.java

```
interface Liste {
    int size();
    void add(??? x);
    void set(int pos, ??? x);
    ??? get(int pos);
    ??? remove(int pos);
}
```

Problem: hva skal vi lagre?

Vi ønsker å lagre objekter av mange ulike klasser i listen vår. Hvordan kan vi angi det?

Én mulig løsning: Object-er

I Java er klassen Object superklassen til alle klasser, enten direkte eller via andre klasser. Følgelig kan en Object-referanse peke på absolutt alle objekter.

En mulig løsning:

Liste.java

```
interface Liste {
    int size();
    void add(Object x);
    void set(int pos, Object x);
    Object get(int pos);
    Object remove(int pos);
}
```

En halvgod løsning

Den virker.

Men . . .

Kronglete notasjon siden vi må typekonvertere:

```
lx.set(10, ((String)lx.get(10))+"*");
```

Klasseparametre

Vi ønsker å lage for eksempel

- lister av Lege-er
- lister av Resept-er
- lister av String-er
- . . .

Da hadde det vært fint om vi kunne ha Lege, Resept, String etc som parameter til Liste-ne våre, på samme måte som verdier kan være parametre til metoder.

En løsning

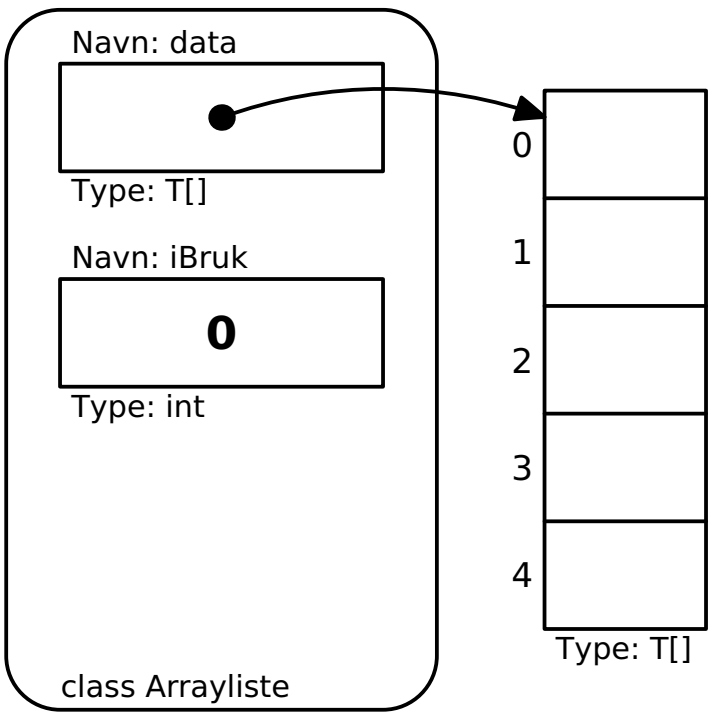
Java har klasseparametre (også kalt *generiske klasser* eller «generics»):

Liste.java

```
interface Liste<T> {
    int size();
    void add(T x);
    void set(int pos, T x);
    T get(int pos);
    T remove(int pos);
}
```

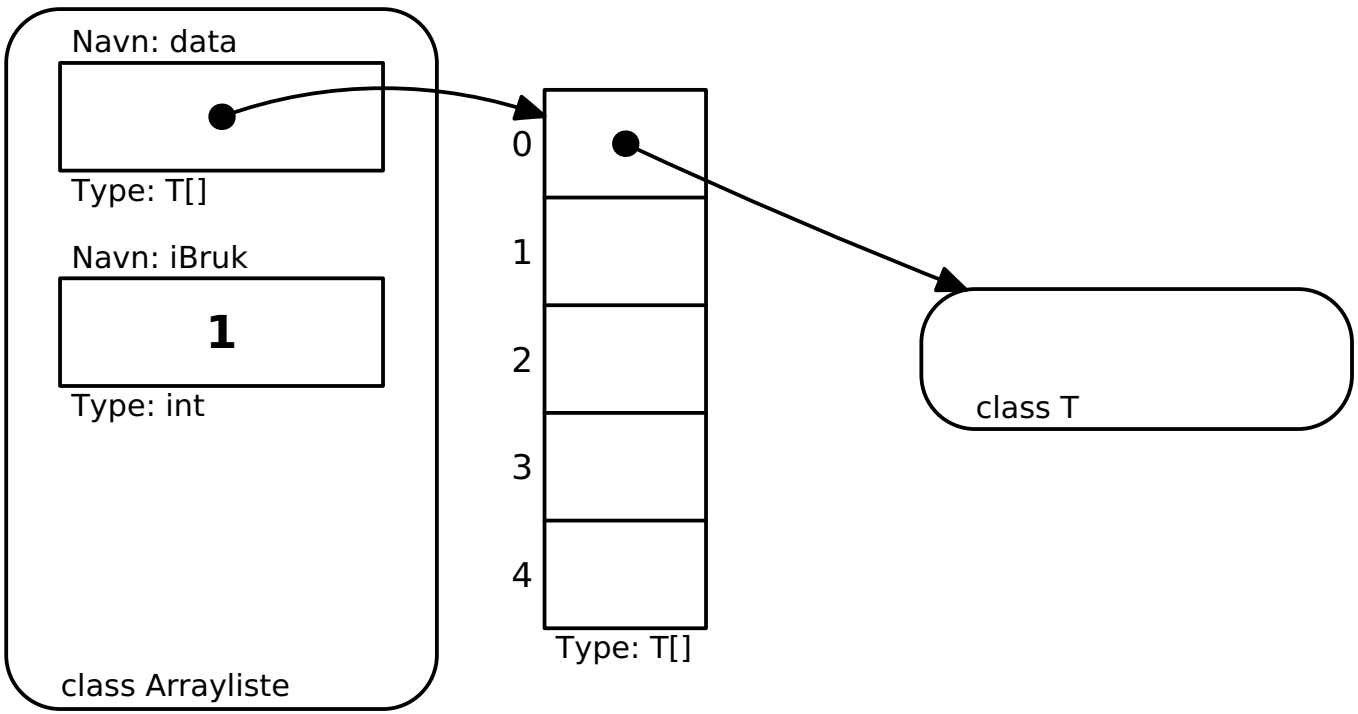
En løsning med array

Vi skal lagre referansene våre i en array:



La oss lagre referansene i en array

Så kan vi legge nye elementer inn i arrayen:



Klassen Arrayliste

Programkoden starter slik:

Arrayliste.java

```
class Arrayliste<T> implements Liste<T> {
    private T[] data = new T[10];
    private int iBruk = 0;
```

Men dette fungerer ikke?!

```
$ javac Arrayliste.java
Arrayliste.java:3: error: generic array creation
    private T[] data = new T[10];
                       ^
1 error
```

En feil i Java!

Denne feilen er egentlig en feil i Java selv!

En løsning som fungerer

Heldigvis finnes det en nødutvei:

Arrayliste.java

```
class Arrayliste<T> implements Liste<T> {
    @SuppressWarnings("unchecked")
    private T[] data = (T[])new Object[10];
    private int iBruk = 0;
```

(Løsningen ville normalt gitt en advarsel, men vi slår av den.)

Klassen Arrayliste

Arrayliste.java

```
class Arrayliste<T> implements Liste<T> {
    @SuppressWarnings("unchecked")
    private T[] data = (T[])new Object[10];
    private int iBruk = 0;
```

Metoden set:

```
    public void set(int pos, T x) {
        data[pos] = x;
    }
```

Metoden get:

```
    public T get(int pos) {
        return data[pos];
    }
```

Metoden size:

```
public int size() {
    return iBruk;
}
```

Metoden remove:

For å fjerne et element, må vi dytte elementene lenger opp i arrayen ett hakk ned.

```
public T remove(int pos) {
    T res = data[pos];
    for (int i = pos+1;  i < iBruk;  i++)
        data[i-1] = data[i];
    iBruk--;
    return res;
}
```

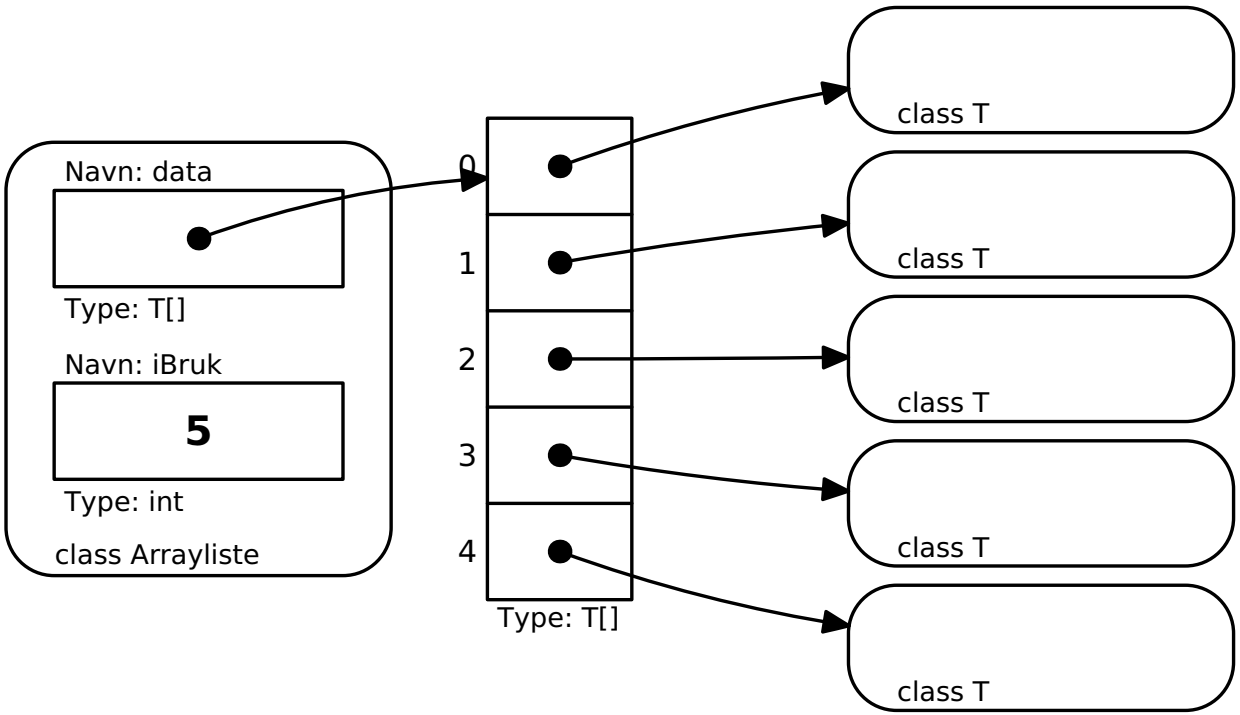
Metoden add:

Det er enkelt å legge til et element i arrayen, med ett unntak:

- Hvis arrayen er full, må vi
  1. opprette en ny og større array og
  2. kopiere innholdet over dit.

```
public void add(T x) {
    if (iBruk == data.length) {
        @SuppressWarnings("unchecked")
        T[] ny = (T[])new Object[2*iBruk];

        for (int i = 0;  i < iBruk;  i++)
            ny[i] = data[i];
        data = ny;
    }
    data[iBruk] = x;
    iBruk++;
}
```

Da setter vi i gang



Intet program er komplett uten testing!

Testing

TestListe.java

```
class TestListe {
    public static void main(String[] args) {
        Liste<String> lx = new Arrayliste<>();

        // Sett inn 13 elementer:
        for (int i = 0; i <= 12;  i++)
            lx.add("A"+i);

        // Sjekk størrelsen:
        System.out.println("Listen har " + lx.size() + " elementer");

        // Marker element nr 10:
        lx.set(10, lx.get(10)+"*");

        // Fjern det første elementet:
        String s = lx.remove(0);
        System.out.println("Fjernet " + s);

        // Skriv ut innholdet:
        for (int i = 0;  i < lx.size();  i++)
            System.out.println("Element " + i + ": " + lx.get(i));

        // Lag en feil:
        lx.remove(999);
    }
}
```

Intet program er komplett uten testing!

Resultatet av testen er:

```
java TestListe
Listen har 13 elementer
Fjernet A0
Element 0: A1
Element 1: A2
Element 2: A3
Element 3: A4
Element 4: A5
Element 5: A6
Element 6: A7
Element 7: A8
Element 8: A9
Element 9: A10*
Element 10: A11
Element 11: A12
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 999
        at Arrayliste.remove(Arrayliste.java:30)
        at TestListe.main(TestListe.java:25)
```

Det meste går bra, men

1. Gal parameter til remove gir en uforståelig feilmelding. (Det gjelder også get og set.)

Egne feilmeldinger

En feilmeldinger bør være en subklasse av RuntimeException. Vi bør ta vare på relevant informasjon.

UlovligListeindeks.java

```
class UlovligListeindeks extends RuntimeException {
    public UlovligListeindeks(int pos, int max) {
        super("Listeindeks " + pos + " ikke i intervallet 0-" + max);
    }
}
```

Da kan vi fange feilen og gi en god feilmelding:

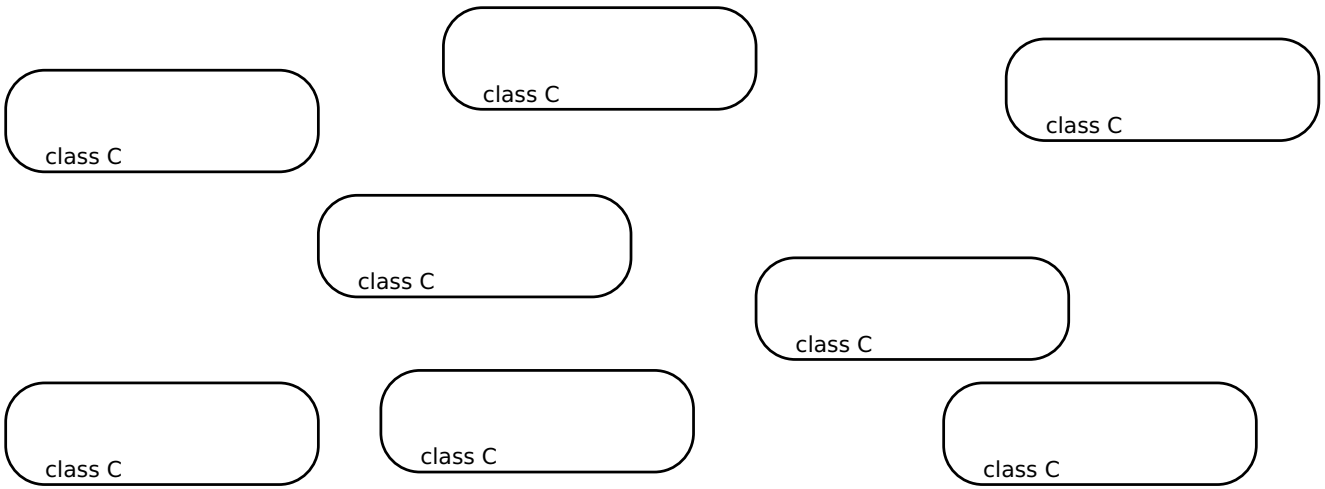
```
try {
    lx.remove(999);
} catch (UlovligListeindeks u) {
    System.out.println("Feil: "+u.getMessage());
}
```

En perfekt(!) ArrayList

Arrayliste.java

```
class Arrayliste<T> implements Liste<T> {
    @SuppressWarnings("unchecked")
    private T[] data = (T[])new Object[10];
    private int iBruk = 0;

    public void set(int pos, T x) {
        if (pos<0 || pos>=iBruk)
            throw new UlovligListeindeks(pos,iBruk-1);
        data[pos] = x;
    }

    public T get(int pos) {
        if (pos<0 || pos>=iBruk)
            throw new UlovligListeindeks(pos,iBruk-1);
        return data[pos];
    }

    public T remove(int pos) {
        if (pos<0 || pos>=iBruk)
            throw new UlovligListeindeks(pos,iBruk-1);
        T res = data[pos];
        for (int i = pos+1;  i < iBruk;  i++)
            data[i-1] = data[i];
        iBruk--;
        return res;
    }
}
```

Hvordan holde orden på objekter?



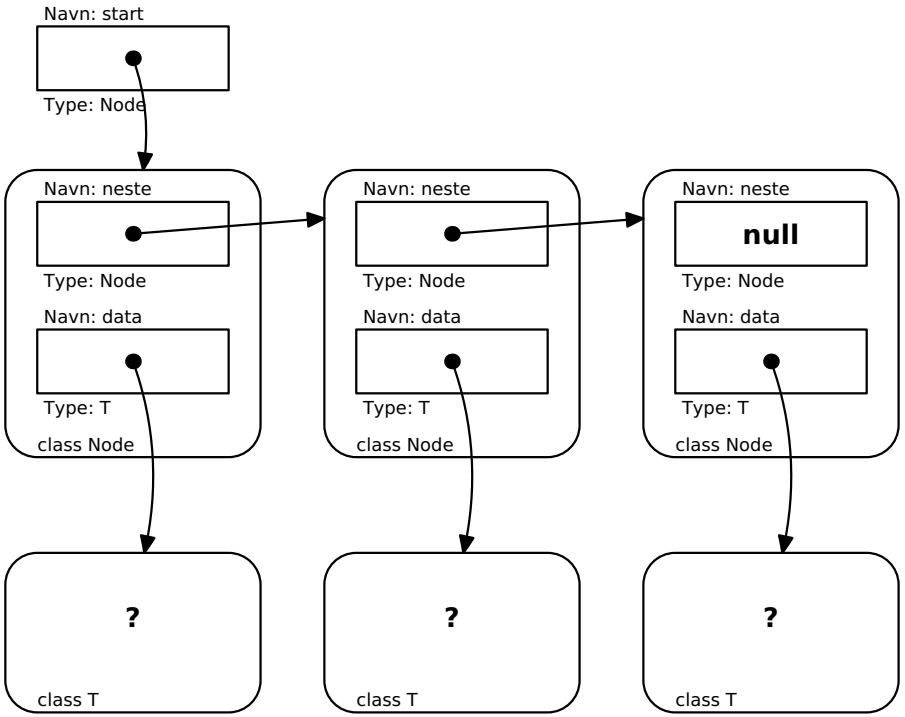
Vi har mange objekter, men vi vet ikke på forhånd hvor mange. Hvordan holder vi orden på dem? Vi må kunne

- finne frem til et gitt objekt (for eksempel nr 7)
- sette inn nye objekter
- fjerne objekter

Finnes det andre muligheter enn å bruke en array?

Vi kan kjede sammen pekerne til data

Løsningen er *lenkelister*




```
class Node {
    Node neste = null;
    T data;

    Node(T x) { data = x; }
}
private Node start = null;
```

1. La hver **Node** inneholde en peker til det neste objektet.
2. Hver **Node** peker også på sine data.
3. Vi har en peker start til det første objektet.

Vi kan kjede sammen pekerne til data

Klassen Lenkeliste

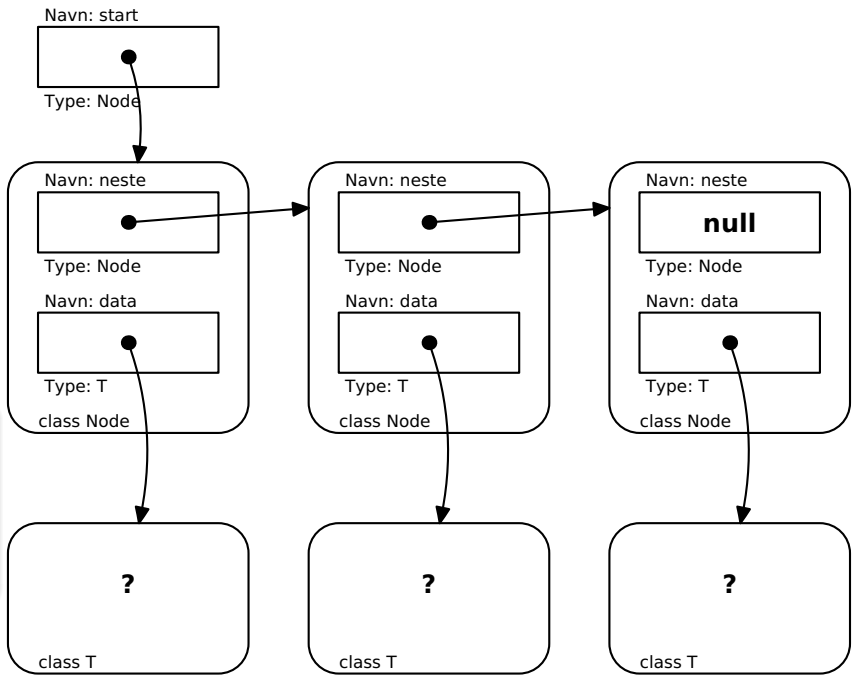
Lenkeliste.java

```
class Lenkeliste<T> implements Liste<T> {
    class Node {
        Node neste = null;
        T data;

        Node(T x) { data = x; }
    }
    private Node start = null;

    public int size() {

    public void add(T x) {

    public void set(int pos, T x) {

    public T get(int pos) {

    public T remove(int pos) {
```

Hvordan finne størrelsen?

Vi må gå gjennom listen og telle alle nodene:

```
Node p = start;
int n = 0;

while (p != null) { n++;  p = p.neste; }
```



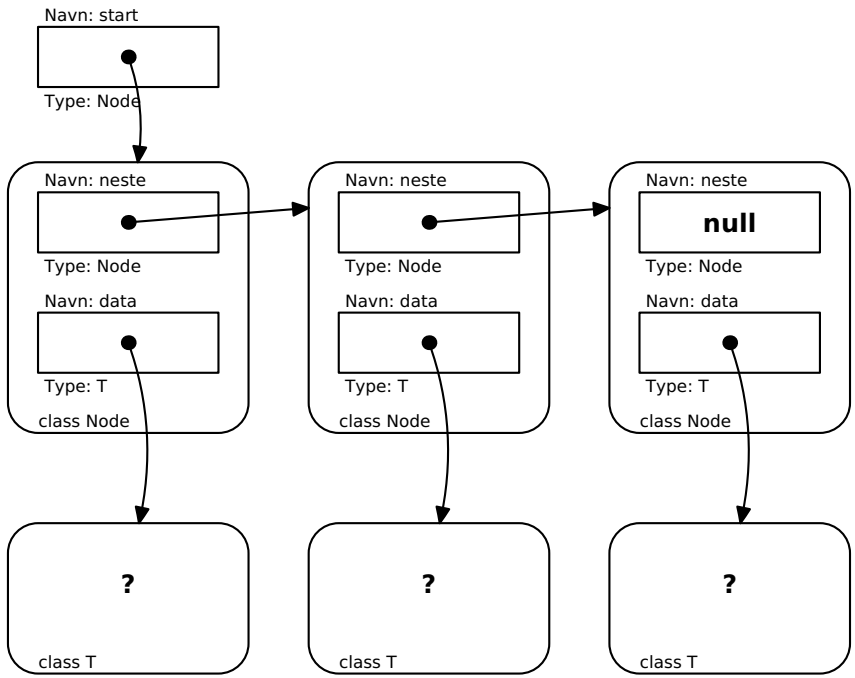
Hint:

Hvis man ofte skal finne lengden, er det lurt å ha lengden lagret i en egen variabel.

Hvordan hente et element?

Vi må telle oss frem til den riktige noden:

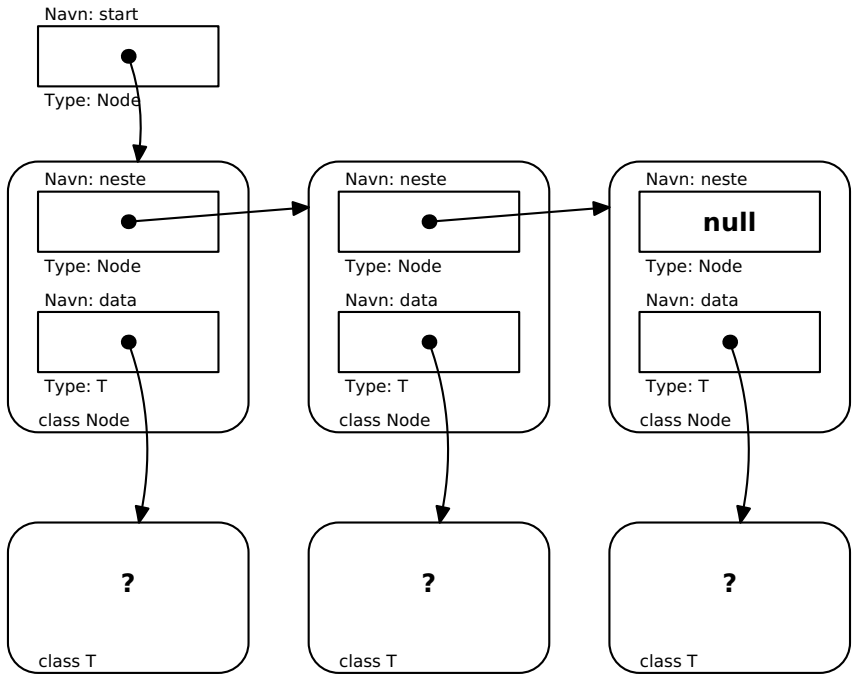
```
Node p = start;
for (int i = 0;  i < pos;  i++)
    p = p.neste;
```



Hvordan sette et vilkårlig element i listen?

Vi må først telle oss frem til det aktuelle elementet i listen:

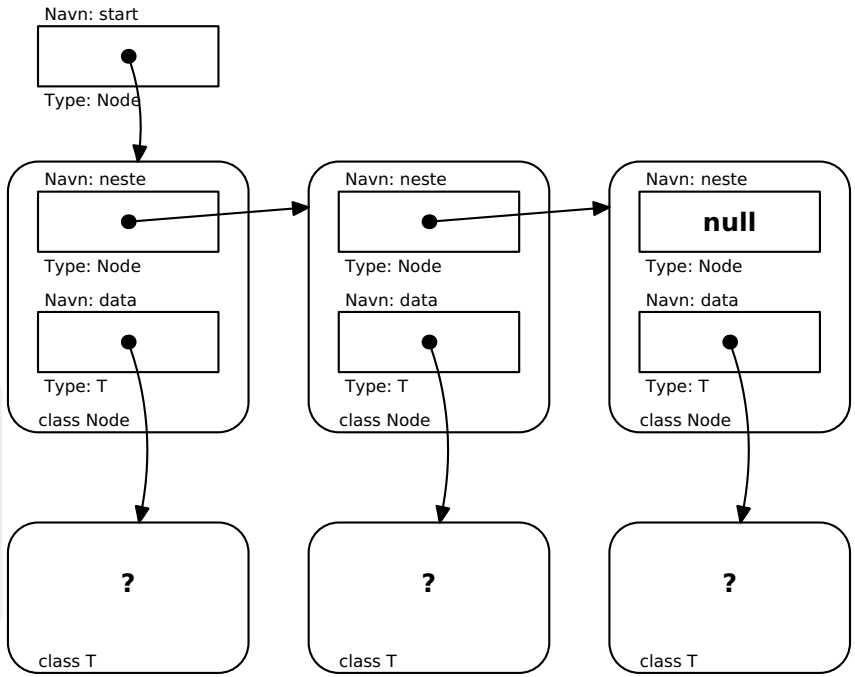
```
Node p = start;
for (int i = 0;  i < pos;  i++) {
    p = p.neste;
}
p.data = x;
```



Hvordan utvide listen med et element?

Vi må først telle oss frem til det siste elementet i listen:

```
Node p = start;
while (p.neste != null)
    p = p.neste;
p.neste = new Node(x);
```



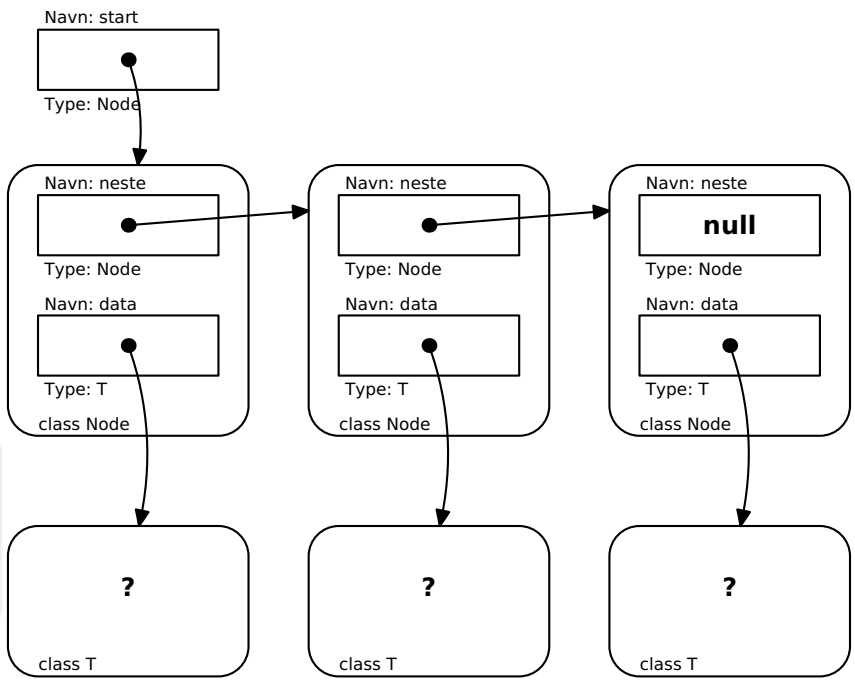
NB!

Det er ett spesialtilfelle vi må kunne håndtere: Hvis listen er tom, hva må gjør vi da?

Hvordan fjerne et element fra listen?

Vi må først telle oss frem til elementet *før* det som skal fjernes:

```
Node p = start;
for (int i = 1;  i < pos;  i++)
    p = p.neste;
Node n = p.neste;
p.neste = n.neste;
```



NB!

Igjen: Hva gjør vi for å fjerne det første elementet i listen?

Neste uke

Neste uke skal vi se nærmere på lenkelister og introdusere andre lignende strukturer.

Java-bibliotekets ArrayList

Hvis noen har lyst, kan de se på hvordan ArrayList er implementert i Java-biblioteket, se her: https://www-adm.uio.no/studier/emner/matnat/ifi/IN1010/v19/java/ArrayList.java