Beholdere og generiske klasser

IN1010 uke 6

Tirsdag 16. februar 2021

Beholdere og generiske klasser - I

- Hvorfor og hvordan velge og bruke beholdere?
- Java Collections Framework
- Lage våre egne beholdere
- Interface Liste
  - Implementert med array som datastruktur
  - Implementert med lenkeliste som datastruktur
- Nye Java mekanismer
  - Klasseparametere (typeparametere) og generiske klasser (generics)
  - Indre klasser
  - Egne Exceptions: Deklarasjon, opprettelse og behandling

Dagens forelesning dekker det meste av oblig 3. Neste uke gir mer input til oppgave C og D

Mer detaljer og eksempler i Big Java (også digitalt)

- ArrayList og HashMap som alternativer til array (Big Java 6.1 & 6.8)
- Java Collections Framework (Big Java 15.1)
- Klasser med typeparametre - «generiske klasser» (Notatet «Enkle generiske klasser» og Big Java 18)
- Lage vår egen ArrayList (Big Java 16.2)
- Lenkelister (Big Java 15.2)

- Dessuten: Java 8 API og Java tutorials.

Hva er en beholder?
(collection/ container)

Java doc: A *collection* is an object that represents a group of objects

=> Et verktøy for å lagre/ organisere elementer av "samme" type

1. Espen
2. Per
3. Paal

- Kjent/ ukjent / variabelt antall
- Legge til, hente ut, finne antall/ størrelse
- Ulike måter å legge til/ hente ut
- Tilleggs-operasjoner
- Ulike måter å representere på – har betydning for plass- og tidsbruk

Beholdere i ulike språk

- Alle høynivåspråk tilbyr verktøy som
  - kan lagre en samling av elementer
  - utføre operasjoner på denne samlingen

- Dere kjenner lister, mengder og ordbøker fra Python - ArrayList og HashMap fra Java

- I objektorienterte språk implementeres en beholder typisk i form av en klasse med
  - et grensesnitt som tilbyr operasjoner på samlingen
  - en datastruktur for å lagre elementene
  - metoder som implementerer grensesnittet ved å operere på datastrukturen

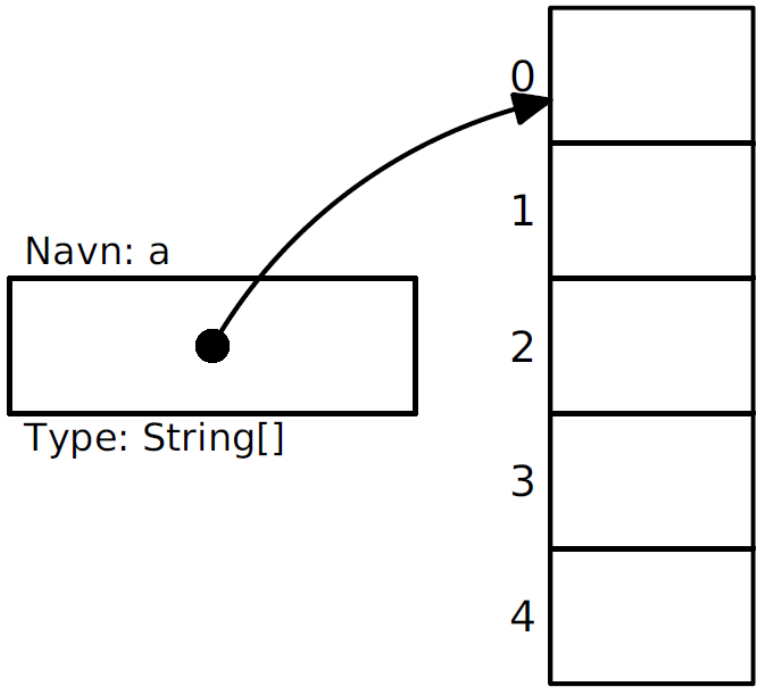
- Samme grensesnitt kan implementeres på ulikt vis av forskjellige klasser

En array kan brukes som beholder – men er ikke en klasse og tilbyr ikke metoder!

Hvorfor er beholdere ("collections") pensum i IN1010?

- Det er nyttige verktøy for svært mange programmer (det har dere allerede sett i IN1000/ IN1900).
- For å velge optimale verktøy bør dere kjenne til hvordan de er bygget opp og fungerer.
- Dere kan få behov for å skrive lignende selv.
- Dette er ypperlige eksempler på og trening i objektorientert programmering.

Array



- En array er en sammenhengende gruppe celler i minnet.
  - array er *ikke en klasse* med metoder (ikke å forveksle med klassene Array og Arrays.
  - Disse er verktøy-kasser i Java API med statiske metoder for aksess og manipulering av arrayer)

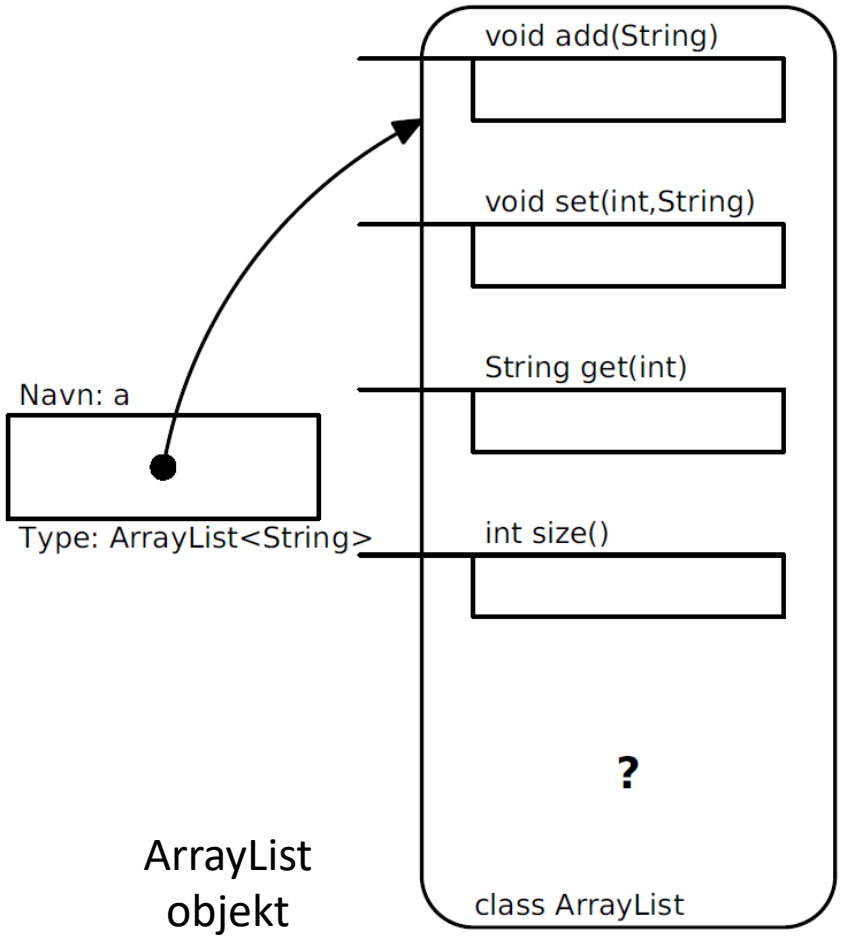
Hva er bra og mindre bra med arrayer?

+ Kompakt og enkel notasjon:

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

+ Bygger på datamaskinens arkitektur og instruksjoner
  + Tar liten plass
  + Raske
- Vi må vite størrelsen når arrayen opprettes.
  - Størrelsen er uforanderlig.
- Kronglete å legge til nye verdier midt i arrayen.
- Udefinert hva som skjer ved fjerning av verdier.
- Ingen innebygde metoder som i en klasse.

Hva er en ArrayList?



- ArrayList er en klasse i Java-biblioteket.
- Gitt grensesnitt: En liste (dvs rekkefølgen er gitt)
- Ukjent implementasjon (hvordan objektene lagres og rekkefølgen vedlike-holdes)

ArrayList sammenlignet med array

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

– / + Metodekall i stedet for egen syntaks – må huske disse:

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

– Ikke for primitive typer som int, char etc
(men kan bruke en *wrapper* klasse, for eksempel **`Integer(tall)`**)
– Tar mye mer plass.
– Er langsommere i bruk.

Sammenligning med Python

I Python har man *lister* som en mellomløsning:

+ Enkel (egen) notasjon (som Javas arrayer)
+ Fleksibel størrelse (som Javas ArrayList)
+ Stort tilbud av innebygde metoder
+ Mange syntaktiske "snarveier"/ kortformer for operasjoner

– Ikke så raskt

Er plassbruk viktig?

- Oftest ikke, men det finnes unntak:

- Hvis man trenger ekstremt mange verdier
- Hvis datamaskinen har veldig lite minne (f eks «Internet of things»)
- Hvis hastigheten er avgjørende (NB – kan være avveining mellom fart og plass)

- Konklusjon: Dette må vurderes når man starter et prosjekt.

Betyr hastighet noe?

En sammenligning av programmer 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:

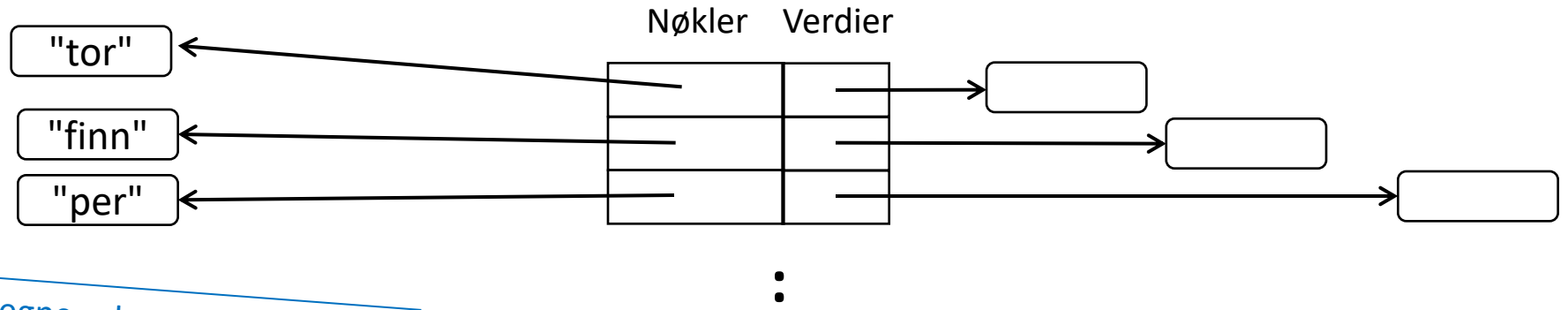
- Når jobben er spesielt stor og tung
- Når man trenger øyeblikkelig respons

Konklusjon: Dette må vurderes når man starter et prosjekt.

HashMap

- Klassen HashMap lar oss lagre (referanser til) objekter uten noen bestemt indre rekkefølge eller nummerering. Den er effektiv og lett å bruke til oppslag.

- Hvert objekt i en HashMap må ha en unik (og immutable) *nøkkel* (ofte en String) som oppgis når vi legger det inn, og brukes for oppslag når noe skal hentes ut



Selv om vi gjerne tegner elementene i en rekkefølge, er de ikke nummerert og vi vet ikke hvilken rekkefølge de ligger i!

Bruk av HashMap I

- Importer klassen

```
import java.util.HashMap;
```

- Deklarer og opprett en HashMap

```
HashMap<String, DVD> dvdArkiv = new HashMap<> ();
```

- Legg et objekt i en HashMap

```
tittel = "Hobbiten";
DVD ny = new DVD (tittel);
dvdArkiv.put (tittel, ny);
```

Bruk av HashMap II

- Hent (peker til) et objekt med en gitt nøkkel – NB sjekk resultatet før du prøver å bruke objektet!

```
DVD denne = dvdArkiv.get(tittel);
if (denne != null) {..}   // fant et objekt med rett tittel
```

- Sjekk størrelsen (antall elementer)

```
int antall = dvdArkiv.size();
```

- Fjern et objekt med en gitt nøkkel fra en HashMap

```
dvdArkiv.remove ("Hobbiten");
```

Valg av beholder – noen indikasjoner

HashMap/ Map: Javas versjon av dictionary i Python.

- Skal du lagre et (kjent) antall verdier av en primitiv type (int, boolean, char,..) og plass eller hastighet er viktig?
  - array
- Har elementene en implisitt rekkefølge?
  - array eller ArrayList
- Skal du lagre et ukjent/ varierende antall objekter?
  - ArrayList eller Hashmap
- Skal du lagre objekter som det er naturlig å slå opp med noe annet enn indeks – og der rekkefølgen ikke betyr noe?
  - HashMap

Java Collection Framework

Verktøy for lagring og organisering av objekter

Java dokumentasjonen: *A collections framework is a unified architecture for representing and manipulating collections*

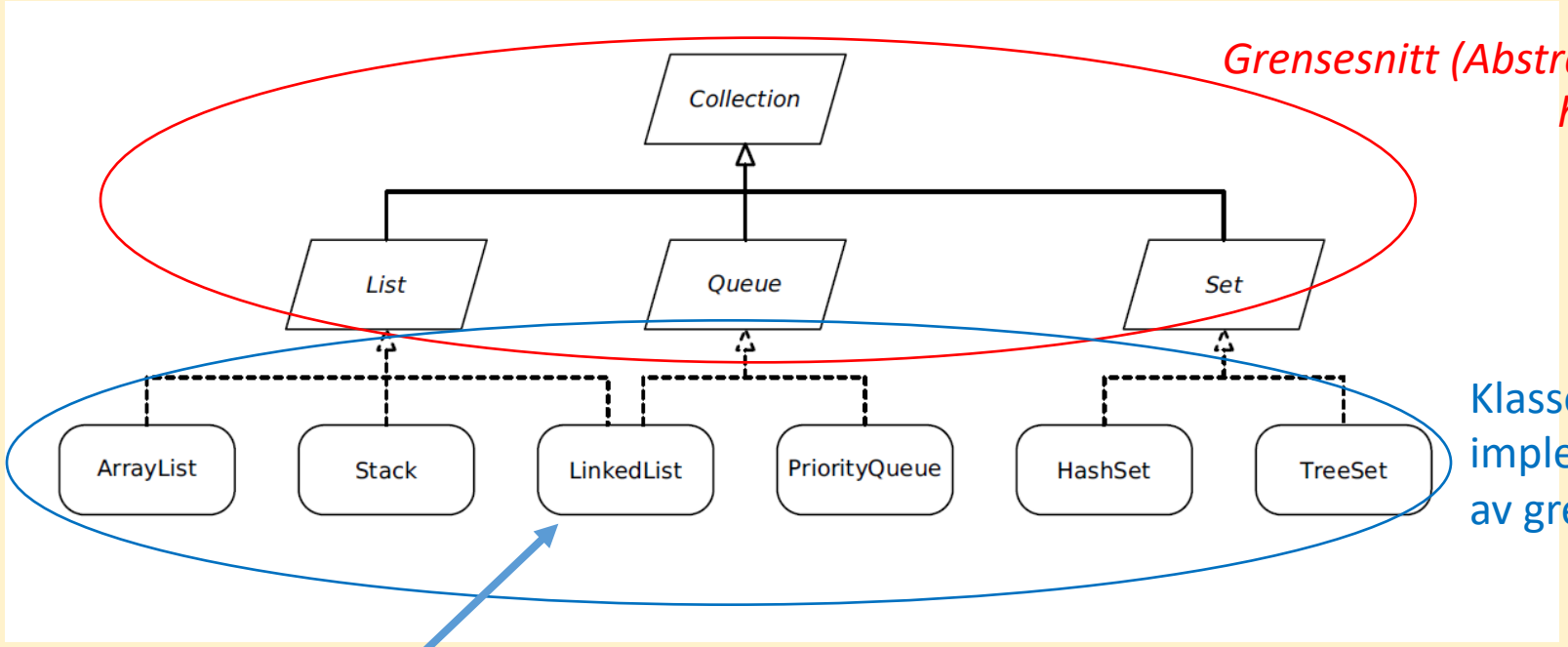
Hierarki av grensesnitt- og klassetyper for beholdere.

Collection Interface er et felles grensesnitt for lister (med rekkefølge) og mengder (uten rekkefølge)

Map er grensesnitt for organisering av nøkkel+verdi par.

- En rekke klasser som implementerer et eller flere grensesnitt
- En rekke grensesnitt som er implementert av en eller flere klasser

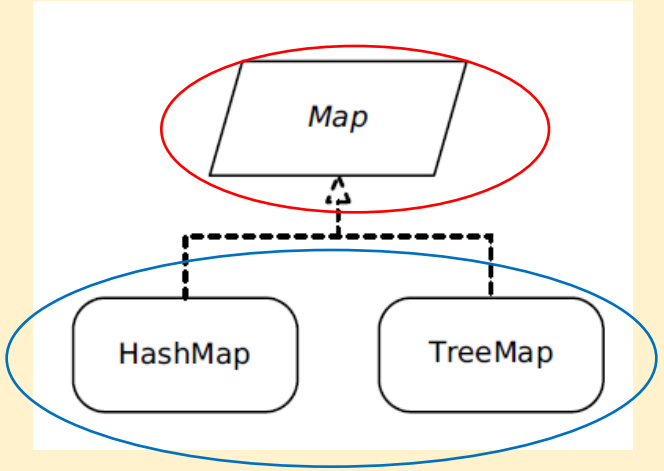
Java Collections Framework (utdrag! fra Big Java)



Grensesnitt (Abstract Data Types) – hvilke operasjoner skal tilbys

Klasser – implementasjoner av grensesnittene

Klassen LinkedList implementerer (her) 2 grensesnitt



Klasser som implementerer beholdere for samlinger av ukjent type

- Dere har skrevet klasser som refererte til objekter av andre klasser

- Eksempel: **class Spilleliste** med **Sang**-objekter. Spilleliste-objekter kunne bare organisere Sang-objekter, og var skreddersydd for disse (*tett koplete klasser)*

- Men vi ønsker å kunne bruke samme verktøy til f.eks:
  - lister av Resept-objekter
  - lister av Lege-objekter
  - lister av String-objekter
  - (kaniner, biler, oster, ...)

Implementasjon av egen Liste

Hvilke operasjoner ønsker vi i en lineær (har en rekkefølge) lagringsstruktur?

- **add** utvider listen med et nytt element.
- **size** forteller hvor lang listen vår er nå.

- **remove** fjerner elementet i en gitt posisjon.
- **get** henter et element fra en gitt posisjon.
- **set** erstatter elementet i gitt posisjon med et nytt.

Implementasjon av Liste: Grensesnitt

- Starter med å definere et Java interface klassen skal implementere
- Da trenger vi returtyper og parametere for metodene

  size, add, set, get, remove

- Men hva slags elementer skal vi lagre og hente ut?

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

Object som type for elementene

- Dette virker – men krever typekonvertering når vi henter ut elementer som skal brukes videre

- Bruker av klassen må selv passe på at listen kun inneholder riktige typer
  => usikker løsning

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

```
:
String element = (String) minListe.get(10);
```

Klasseparametere

- Vi har brukt parametere for å få en metode til å bruke en ny verdi (av samme type) for hver gang den blir kalt – i stedet for å skrive en egen metode for hver tenkelige verdi (ikke gjennomførbart!)

- Klasser i Java kan ha parametere som angir en type (klasse) som skal brukes (inne) i en bestemt instans av klassen Dette kaller vi generiske klasser med klasseparametere

- Brukes f.eks. i ArrayList:

```
ArrayList<String> minListe = new ArrayList<>();
```

Deklarasjon og bruk av generisk klasse

- En parameter i en klassedeklarasjon (formell parameter) angir at klassen kan arbeide med ulike klasser
- Kalles *klasseparameter* eller *typeparameter*

- Når vi lager en instans av klassen bestemmer vi hvilken type denne instansen (objektet) skal jobbe med
- Kalles aktuell parameter/ argument

- Veldig nyttig for beholdere!

```
class Beholder<E> {
    E element1;
    E element2;

    public void settInn(E ny1, E ny2) {
        element1 = ny1;
        element2 = ny2;
    }

    public E taUt1() {
        return element1;
    }

    public static void main(String[] args) {
        Beholder<Integer> b = new Beholder<>();
        b.settInn(5, 17);
        System.out.println(b.taUt1());
    }
}
```

Om typeparametere

- Interface kan ha og være parameter på samme måte som klasse
- Bruker ofte begrepet typeparameter (siden den kan angi klasse *eller* interface)

- Navnekonvensjon for typeparametere (fra Java doc):
  - E - Element (used extensively by the Java Collections Framework)
  - K - Key
  - N - Number
  - T - Type
  - V - Value
  - S,U,V etc. - 2nd, 3rd, 4th types
- Bruk av typeparametere i Java: Generics

Et generisk Liste-interface (grensesnitt)

- Grensesnittet kan ha en typeparameter som representerer typen til objektene i listen.

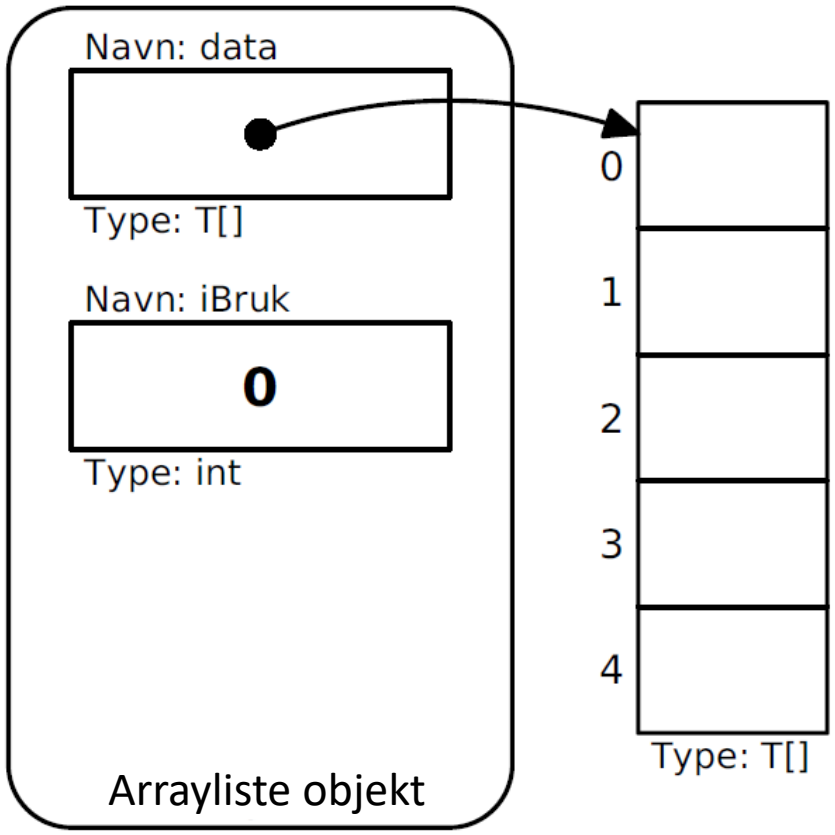
```
interface Liste<T> {
    int size();
    void add(T x);
    void set(int pos, T x);
    T get(int pos);
    T remove(int pos);
}
```

Liste-interface implementert med array

Full implementasjon av Arrayliste. Koden med testprogram ligger på uke6-siden.

Implementasjon med array

- Skriver en egen klasse **Arrayliste**
- Implementerer interface **Liste**
- Lagrer elementene i en array



Datastruktur – og en mangel i Java

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

```
Siri>javac .\Arrayliste.java
.\Arrayliste.java:2: error: generic array creation
    private T[] data = new T[10];
                       ^
1 error
Siri>
```

- Fungerer ikke?!

⇒Java håndterer ikke arrayer med typeparametere

- Kan ha generiske klasser og interface – men ikke generiske arrayer

En "fix" – som fungerer

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

```
Siri>javac .\Arrayliste.java
Note: .\Arrayliste.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
Siri>javac .\Arrayliste.java -Xlint
.\Arrayliste.java:2: warning: [unchecked] unchecked cast
    private T[] data = (T[]) new Object[10];
                            ^
  required: T[]
  found:    Object[]
  where T is a type-variable:
    T extends Object declared in class Arrayliste
1 warning
Siri>
```

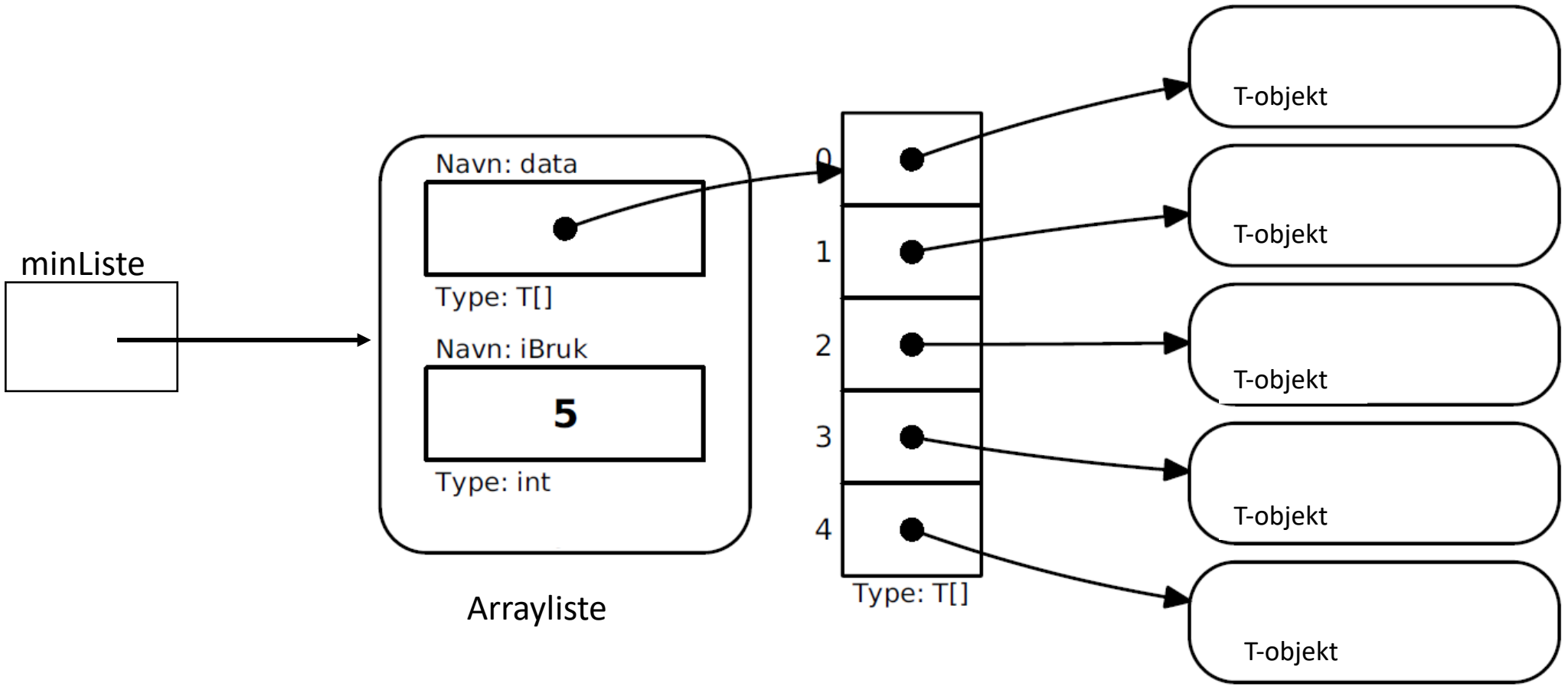
En "fix" – som fungerer II

- Må fortsatt bruke Object som type når vi oppretter arrayen
- MEN har flyttet typekonvertering inn i vår egen klasse, der vi har mer kontroll – og slår derfor av advarsler
- Ved bruk av liste-klassen vår, vil brukeren få feilmelding hvis det er feil type objekt som legges inn eller hentes ut

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

Arrayliste objekt

Det finnes ikke T-objekter, under kjøring vil de ha en annen, eksisterende type (for eksempel String) som vi opprettet listen med



Klassen Arrayliste: size, set, get

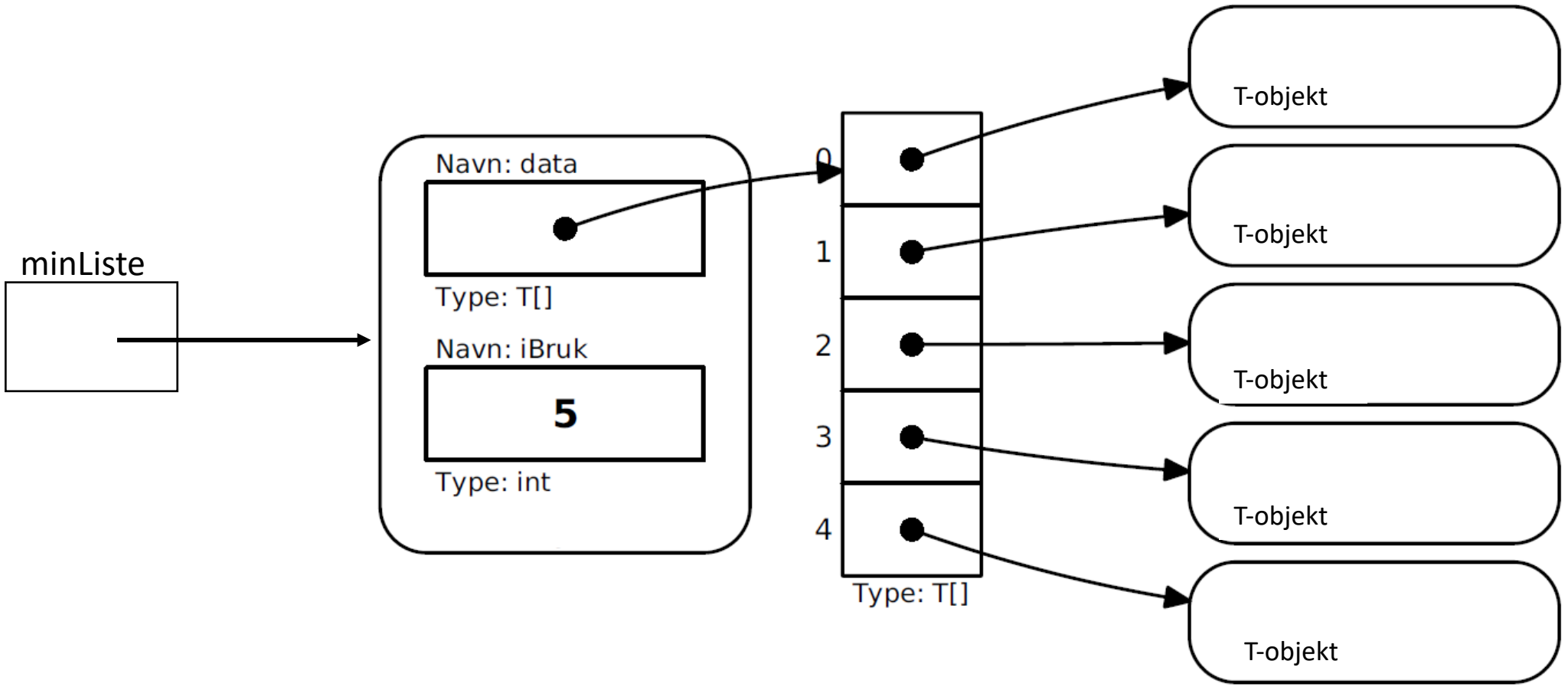
- Kan noe skape problemer her? Kommer tilbake til dette

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

    @Override
    public int size() {
        return iBruk;
    }

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

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

Klassen Arrayliste II: remove

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

    // andre metoder

    @Override
    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;
        }
    }
}
```

Arrayliste objekt med full array



Klassen Arrayliste:
Lage plass til flere elementer

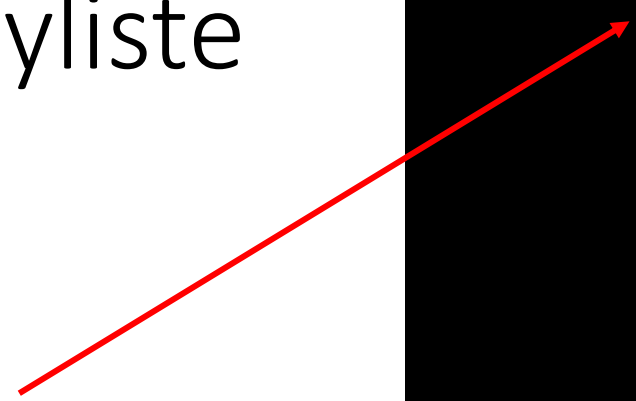
- Spesialtilfelle ved tillegg nytt element i listen:
  - arrayen som holder dataene kan være full!

- Må da allokere mer plass =>
  - oppretter ny array med flere plasser (2* den vi har)
  - flytter eksisterende elementer over
  - legger til det nye på første ledige plass

Klassen Arrayliste III (add)

```
@Override
public void add(T x){
    if (iBruk == data.length) {
        @SuppressWarnings("unchecked")
        T[] ny = (T[]) new Object[2*iBruk];
        for (int i=0; i<data.length;i++) {
            ny[i] = data[i]; }
        data = ny;
        }
    data[iBruk] = x;
    iBruk++;
    }
}
```

Testprogram Arrayliste



```
public class TestArrayliste<T> {
    public static void main(String[] args) {
        Liste<String> lx = new Arrayliste<>();
        // Sett inn 13 elementer:
        for (int i = 0; i < 13; i++) {
            lx.add("A" + i);
        }
        // sjekk antall
        System.out.println("Listen har " + lx.size() + " elementer");
        // Marker element nr 10:
        lx.set(10, lx.get(10) + "*");
        // Fjern første element
        lx.remove(0);
        System.out.println("Fjernet element 0");
        // Skriv ut listen
        for (int i = 0; i < lx.size(); i++) {
            System.out.println("Element " + i + ": " + lx.get(i));
        }
        // Lag en feil
        lx.remove(999); }
}
```

Kjøring av test

```
Siri>java TestArrayliste
Listen har 13 elementer
Fjernet element 0
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:
 Index 999 out of bounds for length 20
        at Arrayliste.remove(Arrayliste.java:32)
        at TestArrayliste.main(TestArrayliste.java:20)
Siri>
```

- Det meste går bra, men:

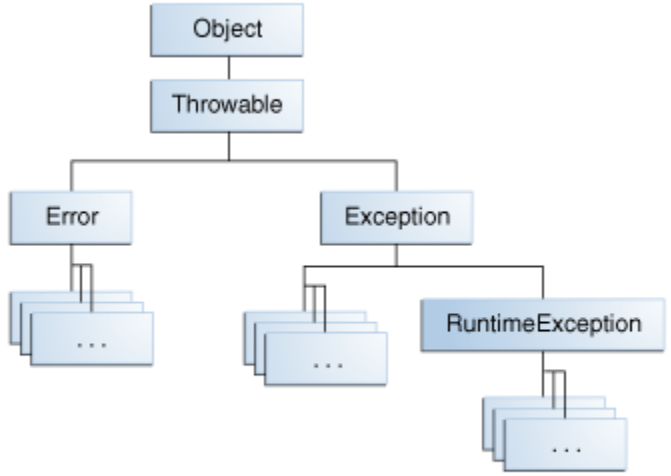
  => gal parameter til remove gir en uforståelig feilmelding.

- (Det gjelder også get og set.)

Feilhåndtering

- Hva kan gå galt?
  - Feil i Java => Error, trenger ikke tenke på/ håndtere
  - Feil som oppstår i Java API eller andre biblioteker du bruker

  => Feil i input (argumenter eller terminalbruker)
  - Feil (bugs) i din kode



The Throwable class.

- Strategi: Oppdag tidlig (1) – håndter på passende nivå(2)!

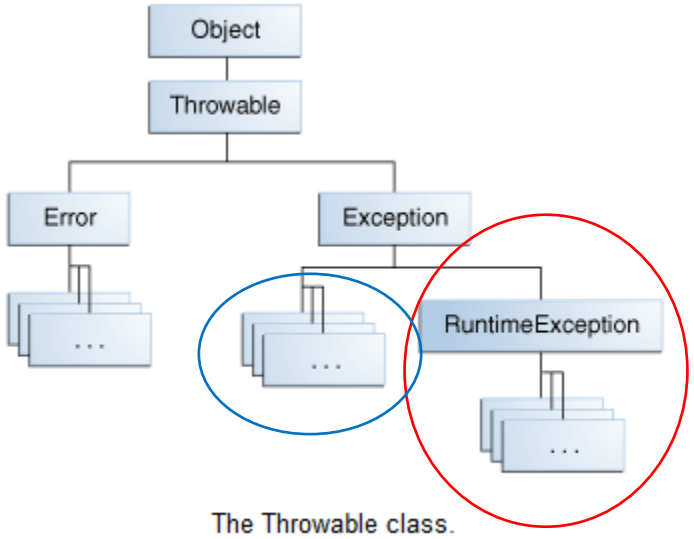
  (1) Mest mulig informasjon om feilsituasjon

  (2) Mest mulig informasjon om hva som er beste håndtering
  - Jo større avstand mellom 1) og 2), desto nyttigere å bruke Exception mekanismer

Feilhåndtering med Exceptions

- Checked eller unchecked
  - checked (..io... må håndteres, ref fil-håndtering)

  => unchecked (dine valg, trenger ikke importeres)
- Der feilen oppdages:
  - Håndter eller kast passende Exception (**throw**)
    - evt egendefinert subklasse av passende Exception
  - Ta med i dokumentasjon og signatur (**throws**)



The Throwable class.

- Teste og fange (**try** – **catch**) Exceptions ved kall på metode som kaster
  - Kan velge hvordan de håndteres
  - Kan be om ny verdi fra bruker, avslutte en operasjon eller avslutte programmet
  - Om du ikke fanger en Exception som oppstår, håndterer Java den "på sin måte"

Egne Exceptions (kan også bruke innebygde)

- Feilmeldinger bør være en subklasse av passende Exception
- Her: RuntimeException eller en subklasse(se Exception klasse-hierarki med forklaringer i Big Java).
- Konstruktøren tar parametere med nyttig informasjon om feilen (her: hvilken indeks ble brukt, og hvilke er lovlige)

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

Oppdage at noe er feil

- Vi tar vare på relevant informasjon der feil kan oppstå (for eksempel i metoden remove)
- ... og sender den med til Exception-objektet vi oppretter
- ... som vi så "kaster" tilbake til kallstedet med **throw**

```
    @Override
    public T remove(int pos) throws UlovligListeIndeks {
        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; }
```

Håndtere feil som oppstår i en metode

- Når vi bruker metoder som kan kaste unntak (som remove) skriver vi en try - catch blokk for å håndtere dem

```
// Lag en feil - og håndter den
try {
    lx.remove(999);
} catch (UlovligListeIndeks u) {
    System.out.println("Feil: " + u.getMessage()); }
```

- Hvordan vet vi om andres metoder kaster unntak?

Arrayliste med egen feilmelding

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

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

    @Override
    public T remove(int pos)
       throws UlovligListeIndeks {
        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; }
}
```

Feilhåndtering i testprogrammet

```java
class TestArrayliste {
    public static void main(String[] args) {
        Liste<String> lx = new Arrayliste<>();
        // ....Sett inn 13 elementer, andre tester....

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

        System.out.println("Fortsetter etter catch-blokken");

    }
}
```

```
>java TestArrayliste
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
Feil: Listeindeks 999 ikke i intervallet 0-11
Fortsetter etter catch-blokken
```

Liste-interface implementert med lenkeliste i stedet for array

Noen tips for implementasjon med lenkeliste

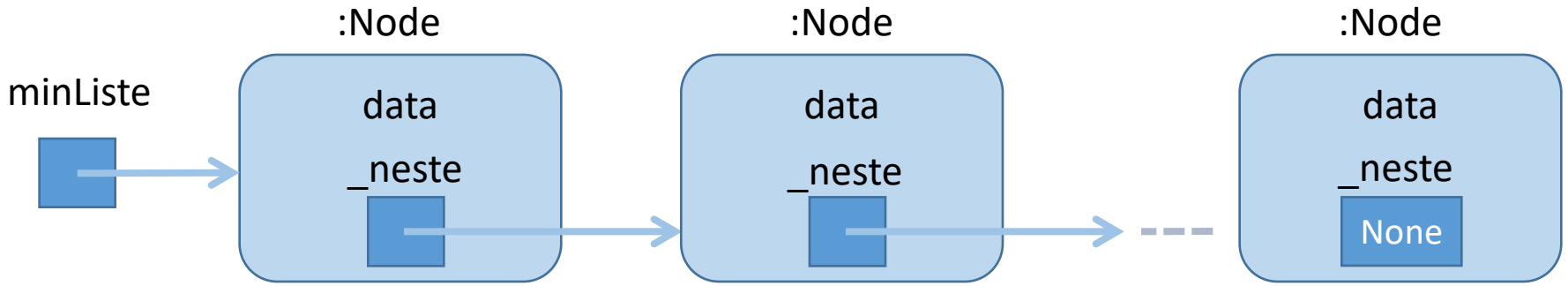
Kan vi implementere Liste på en annen måte?

- I forrige eksempel implementerte vi interface Liste ved hjelp av klassen Arrayliste
- Arrayliste bruker en array som datastruktur for objektene – krevde håndtering av fullt array

- Kan vi lage en beholder som lagrer objekter på en mer dynamisk måte – der vi alltid kan ta inn *ett til* uten å "bygge om" datastrukturen vår?
- Det vi skal lagre (objektene) opprettes utenfor beholder-klassen, det vi trenger er en datastruktur der det alltid er *en ledig referanse* til det nye elementet

=> for hvert element, oppretter vi et hjelpe-objekt (node) som skal referere til det nye elementet – OG kan referere til et enda et nytt hjelpeobjekt!

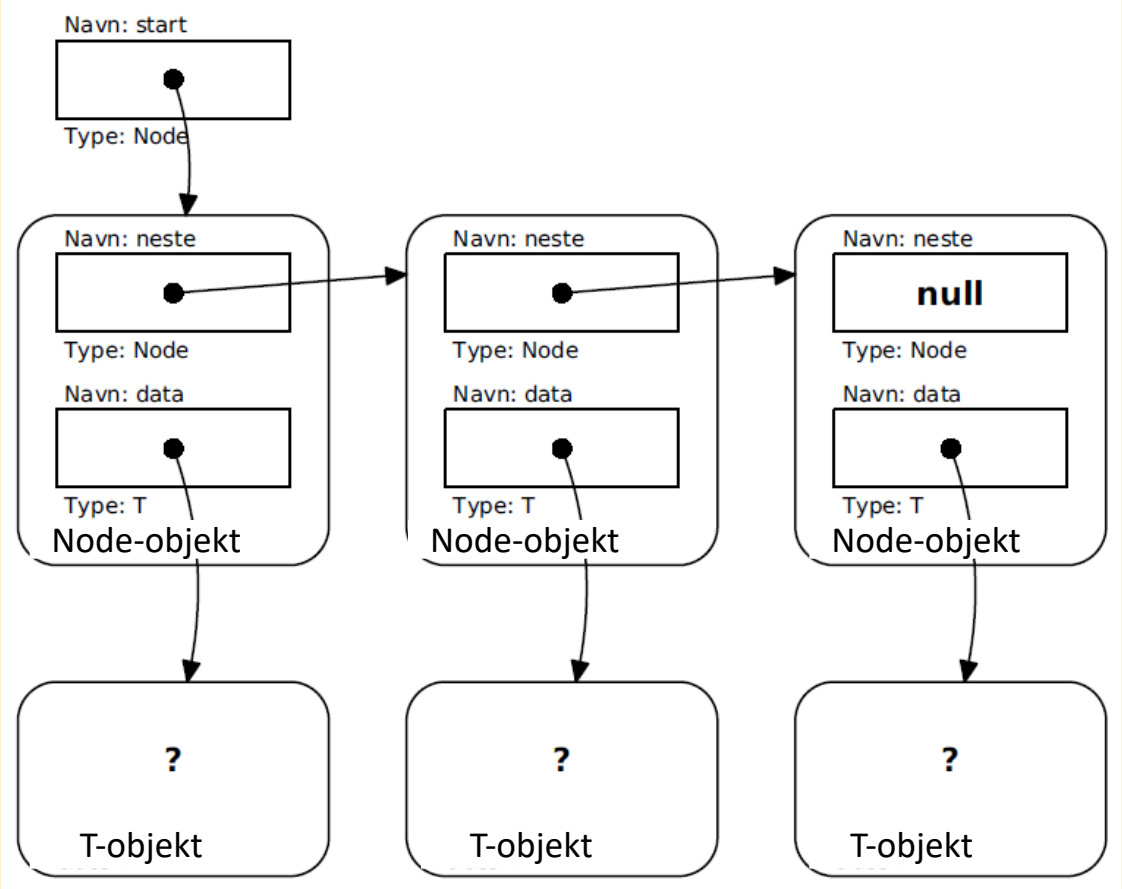
Lenkeliste (NB: figur fra IN1000)



- Poenget med denne strukturen er at for hvert nye objekt vi lager – så lager vi samtidig en referansevariabel som kan referere til et nytt objekt
- dvs hvert objekt må kunne referere til et annet objekt
- dermed får vi en lenket liste av objekter – og trenger bare ha én referanse til det første objektet

Datastruktur inne i en Lenkeliste
(erstatter arrayen vi brukte i Arrayliste)

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



Klassen Lenkeliste

- Vi implementerer samme interface **Liste** som **Arrayliste** implementerte
- Vi har bestemt datastruktur: En sammenlenket kjede av **Node**-objekter, og en referanse **start** til første Node-objekt
- Hvordan legge dette inn i klassen **Lenkeliste**?
- Vi deklarerer en *indre klasse* **Node** inne i klassen **Lenkeliste**

Indre klasser

- Klasser kan deklareres inne i metoder eller andre klasser – om de kun skal brukes der (nøstede klasser)
- En klasse deklarert i en annen klasse er tilgjengelig for den ytre klassens metoder, men ikke utenfor
- Tydeliggjør at den kun brukes internt, og hindrer aksess fra utsiden. Fjerner behovet for innkapsling og forenkler bruk!

- Den indre klassen får en egen .class-fil ved kompilering (**Lenkeliste$Node.class**)

Klassen Lenkeliste: Datastruktur og grensesnitt

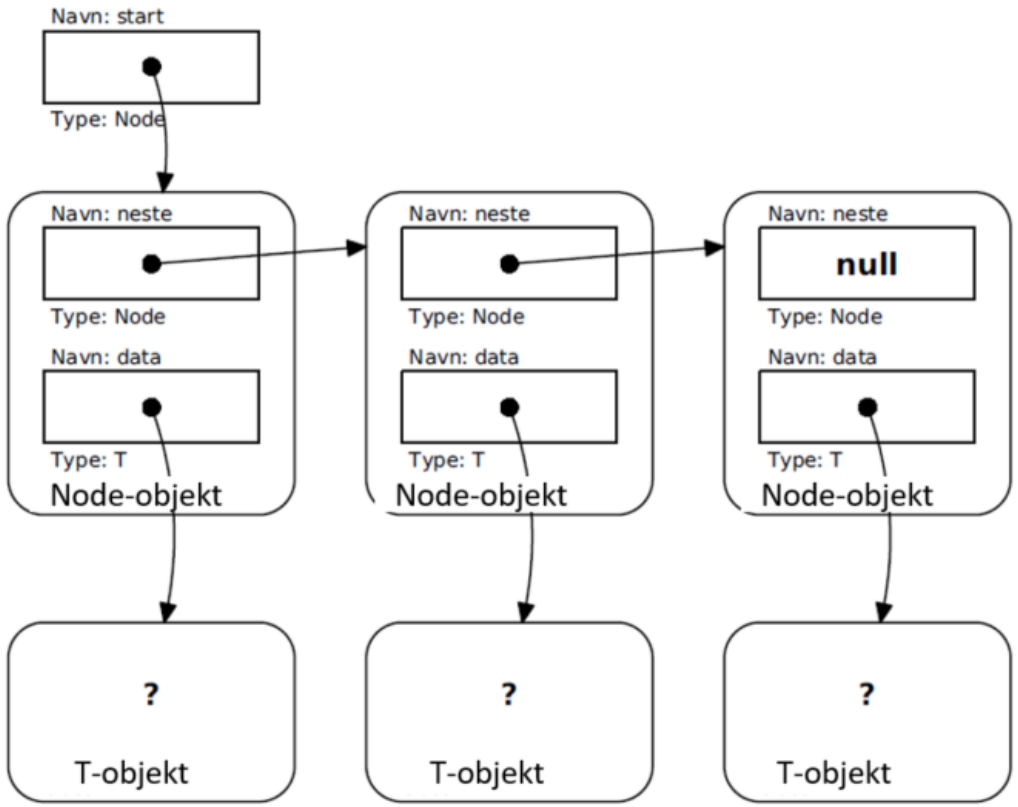
```
class Lenkelist<T> implements Liste<T> {
    class Node {
        Node neste = null;
        T data;
        Node (T x) {
            data = x; }
    }
    private Node start = null;

   // Resten av grensesnittet
```

Hvordan finne størrelsen?

- Går gjennom liste og teller noder!

```
public int size() {
    int teller = 0;
    Node peker = start;
    while (peker != null) {
        teller++;
        peker = peker.neste;
    }
    return teller;
}
```

- Alternativ?



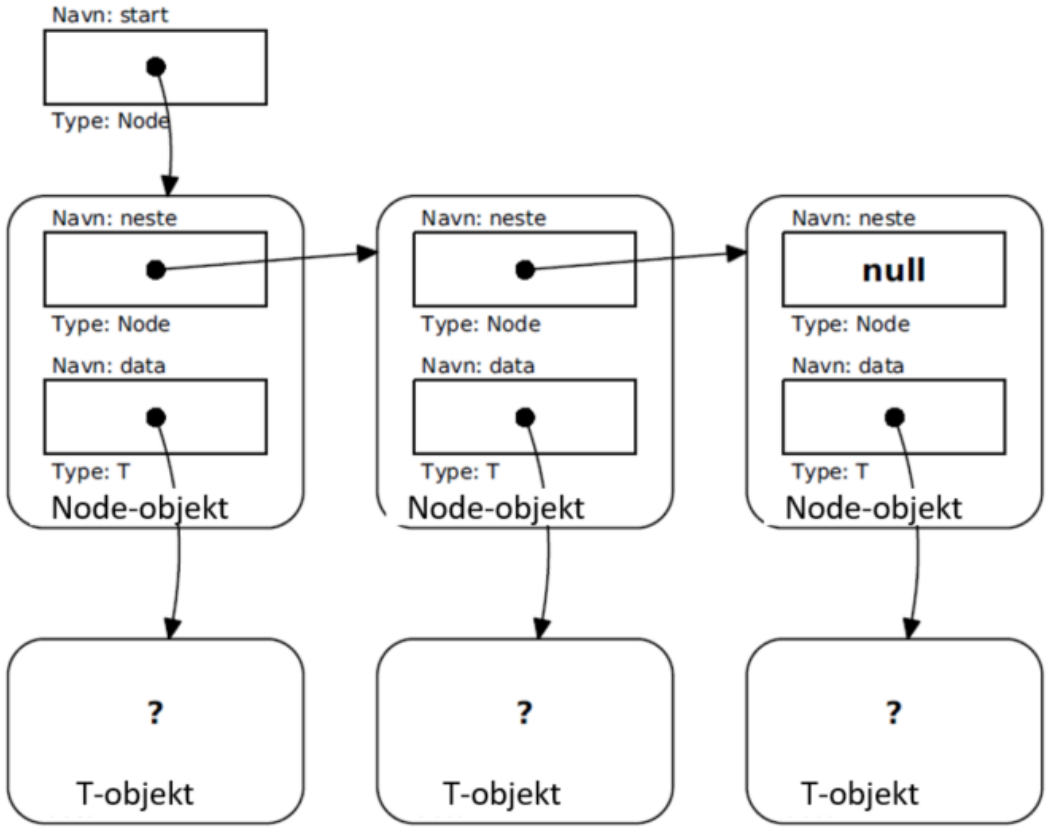
Hvordan hente et element?

```
public T get(int pos) {
```

- Går gjennom liste, teller oss frem til rett plass

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

- NB: Hva skal vi returnere?

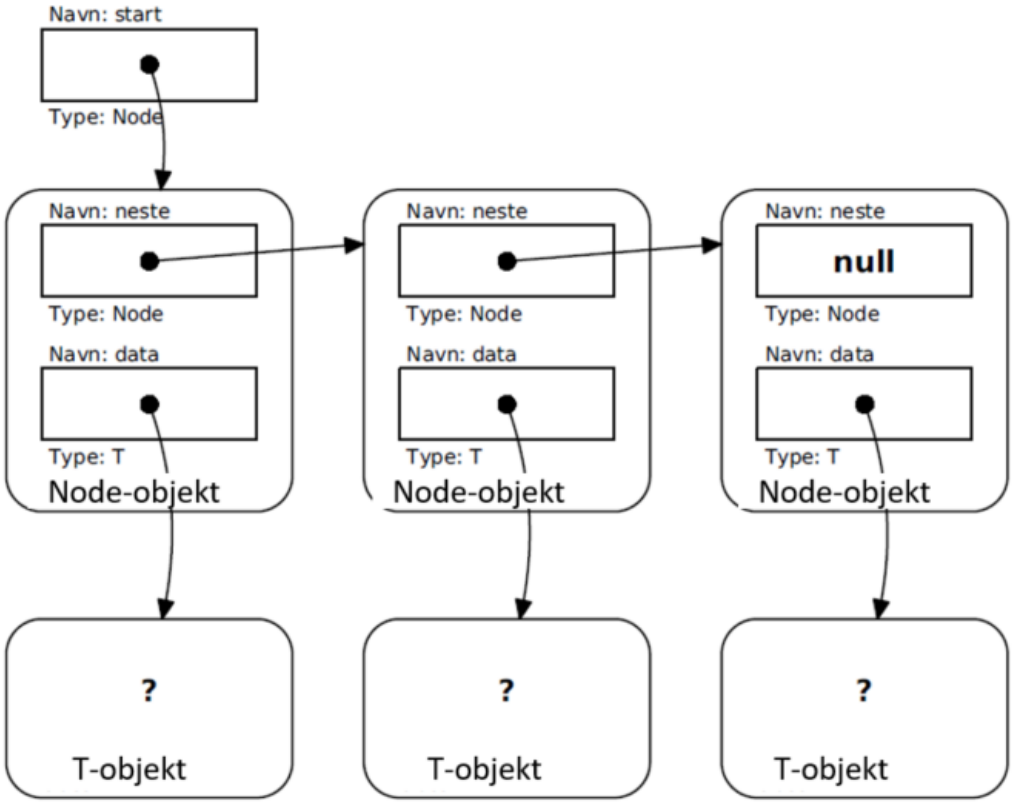


Hvordan fjerne et element fra listen?

- Teller oss frem til rett sted:

```
Node peker = start;
for (int i=0; i<pos-1; i++) {
    peker = peker.neste;
}
res = peker.neste.data;
peker.neste = peker.neste.neste;
```

Elementet *før* det som skal fjernes



- Hvilket element må vi stoppe på?
- Hvilket spesialtilfelle må håndteres her?

Bruk av lenkelisten

?

Svar i direkte chat til Stein ☺

Hvordan skiller et testprogram for lenkeliste-klassen seg fra testprogrammet vi skrev for arrayliste-klassen?

```
class TestArrayliste {
    public static void main(String[] args) {
        Liste<String> lx = new Arrayliste<>();
        // ....Sett inn 13 elementer, andre tester....

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

        System.out.println("Fortsetter etter catch-blokken");

    }
}
```

Oppsummering

- Beholder: Hva og hvordan
  - Liste-interface
  - Implementering av Liste med array eller lenkeliste (sentrale deler av koden)

- Nytt i Java
  - Klasseparametere (typeparametere) og "generics"
  - Indre klasser
  - Egne Exceptions: Deklarasjon, opprettelse og behandling

Neste uke

- Andre måter å implementere lenkelister
- Varianter av liste-grensesnitt:
  - stabel (stack, Last In First Out – LIFO)
  - kø (First In First Out – FIFO)
  - Prioritetskø

- Mer Java
  - Innpakking ("autoboxing/ unboxing")
  - Å sammenligne objekter (Interface Comparable)
  - Å gå gjennom alle elementer i en samling (Iterator)