

Beholdere og generiske klasser

IN1010 uke 6
Onsdag 19. februar 2020

IN1010 - vår 2020
Siri Moe Jensen

1

Beholdere og generiske klasser - I

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

IN1010 - vår 2020
Siri Moe Jensen

2

Mer detaljer - i tillegg til Java 8 docs

- Array-er, ArrayList og HashMap (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)

(Dagens forelesning dekker det meste av oblig 3. Neste uke gir litt mer input på oppgave C og spesielt D).

IN1010 - vår 2020
Siri Moe Jensen

3

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

IN1010 - vår 2020
Siri Moe Jensen

4

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 - Array, 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 opererer på datastrukturen
- Samme grensesnitt kan implementeres på ulikt vis av forskjellige klasser

IN1010 - vår 2020
Siri Moe Jensen

5

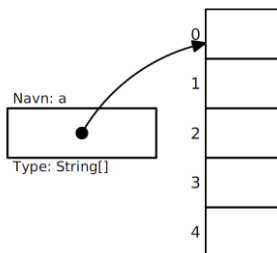
Hvorfor er beholdere ("collections") pensum i IN1010?

- 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 objektorientert programmering.

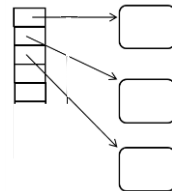
IN1010 - vår 2020
Siri Moe Jensen

6

Array-er



En array er en sammenhengende gruppe celler i minnet.



IN1010 - vår 2020
Siri Moe Jensen

7

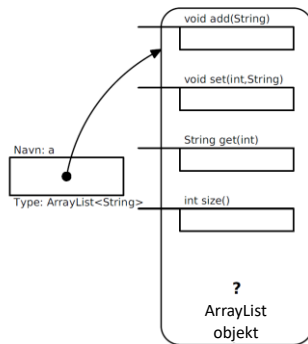
Hva er bra og mindre bra med arrayer?

- + 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.
- Udefinert hva som skjer ved fjerning av verdier.
- Ingen innebygde metoder som i en klasse.

IN1010 - vår 2020
Siri Moe Jensen

8

Hva er en ArrayList?



- ArrayList er en klasse i Java-biblioteket.
- Gitt grensesnitt
- Ukjent implementasjon

IN1010 - vår 2020
Siri Moe Jensen

9

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 (finnes en omvei)
- Tar mye mer plass.
- Er langsommere i bruk.

IN1010 - vår 2020
Siri Moe Jensen

10

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
- Ikke så raskt

IN1010 - vår 2020
Siri Moe Jensen

11

Er plassbruk viktig?



Oftest ikke, 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 (NB – kan være avining mellom fart og plass)

Konklusjon: Dette må vurderes når man starter et prosjekt.

IN1010 - vår 2020
Siri Moe Jensen

12

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

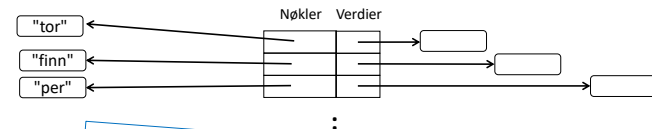
Oftest er programmets kjøretid ikke spesielt viktig, men det finnes unntak:

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

Konklusjon: Dette må vurderes når man starter et prosjekt.

HashMap: En "sekk" objekter

- Klassen `HashMap` lar oss lagre (pekere 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 *nøkkel* (typ. 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 liste, 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<String, DVD> ();
```

- Legg et objekt i en HashMap

```
DVD ny = new DVD ("Hobbiten");  
dvdArkiv.put (ny.toString(), 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 array/ ArrayList/ HashMap

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 et heltall – og der rekkefølgen ikke betyr noe?
 - HashMap

Java Collection Framework

Verktøy for lagring og organisering av objekter

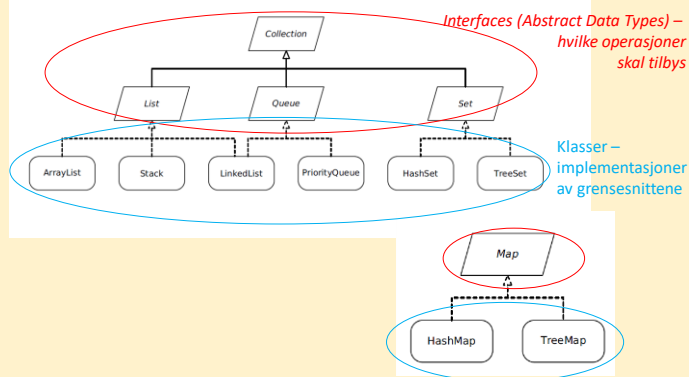
Hierarki av Interface- og klasstyper for beholdere.

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

Map er interface 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 (fra Big Java)



Klasser som implementerer beholdere for samlinger av kjent type

- Dere har skrevet klasser som refererte til objekter av andre klasser
- Eksempel: **class Spilleliste** med **Sang**-objekter. Spillelisteobjekter kunne bare organisere Sang-objekter, og var skreddersydd for disse (*tett koplede 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 definerer 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

```
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);
```

- Dette virker – men krever typekonvertering når vi henter ut elementer som skal brukes videre
- Må selv passe på at vi kun legger inn riktige typer, dvs usikker løsning

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<String>();
```

Deklarasjon og bruk av generisk klasse

- En parameter i en klassedeklarasjon (*formell parameter*) angir at klassen kan arbeide med ulike typer
- Når vi lager en instans av klassen bestemmer vi hvilken type denne instansen (objektet) skal jobbe med (*aktuell parameter/ argument*)
- Dette er nyttig for beholdere!

```
public class ArrayList<E> ....{
    .....
    public E remove(int index) {...}
    .....
}
ArrayList<String> minListe = new ArrayList<String>();
```

IN1010 - vår 2020
Siri Moe Jensen

25

Generelt: Typeparametere

- Interface kan *ha* og *være* parameter(e) på samme måte som klasser
- Bruker ofte begrepet *typeparameter*
- 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*

IN1010 - vår 2020
Siri Moe Jensen

26

Liste-interface med typeparameter

- Klassen som implementerer dette interfacet kan ha en klasseparameter som representerer typen til objektene i listen.

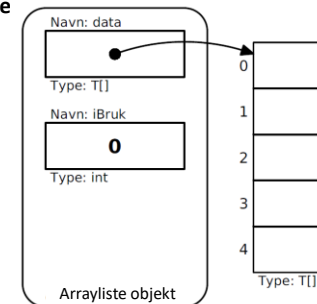
```
interface Liste<T> {
    int size();
    void add(T x);
    void set(int pos, T x);
    T get(int pos);
    T remove(int pos);
}
```

IN1010 - vår 2020
Siri Moe Jensen

27

Implementasjon med array

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



IN1010 - vår 2020
Siri Moe Jensen

28

Datastruktur – og en feil i Java

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

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

Fungerer ikke?!

IN1010 - vår 2020
Siri Moe Jensen

29

En "fix" – som fungerer

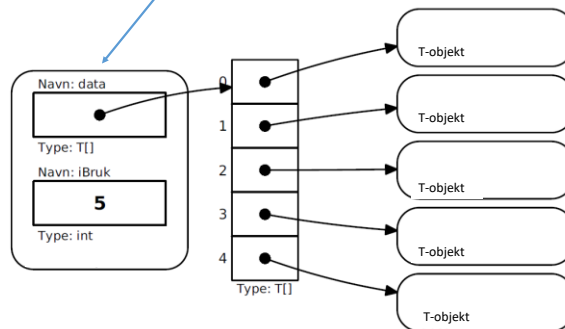
```
class Arrayliste<T> implements Liste<T> {
    @SuppressWarnings("unchecked")
    private T[] data = (T[])new Object[10];
    private int iBruk = 0;
    // metode-deklarasjoner
}
```

- 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

IN1010 - vår 2020
Siri Moe Jensen

30

Arrayliste objekt



IN1010 - vår 2020
Siri Moe Jensen

31

Klassen Arrayliste: size, set, get

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

    public int size() {
        return iBruk;
    }

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

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

IN1010 - vår 2020
Siri Moe Jensen

32

Klassen Arrayliste II (remove)

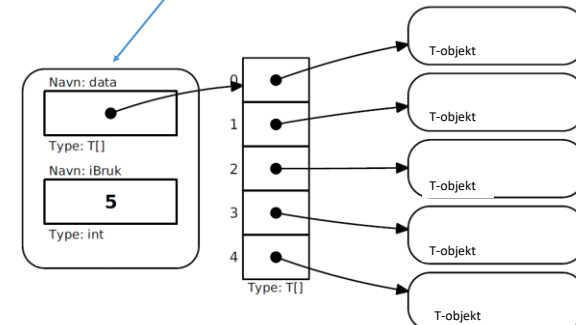
```
class Arrayliste<T> implements Liste<T> {
    @SuppressWarnings("unchecked")
    private T[] data = (T[])new Object[10];
    private int iBruk = 0;
    // metoder set, get, size utelatt

    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;
    }
}
```

IN1010 - vår 2020
Siri Moe Jensen

33

Arrayliste objekt med full array



IN1010 - vår 2020
Siri Moe Jensen

34

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*)
 - flytter eksisterende elementer over
 - legger til det nye på første ledige plass

IN1010 - vår 2020
Siri Moe Jensen

35

Klassen Arrayliste III (add)

```
class Arrayliste<T> implements Liste<T> {
    // Datastruktur og metoder set, get, size, remove utelatt
    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++;
    }
}
```

IN1010 - vår 2020
Siri Moe Jensen

36

Testprogram I

```

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)+"*");
}

```

IN1010 - vår 2020
Siri Moe Jensen

37

Testprogram I (kommentert kopi)

```

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("

    // Marker element nr
    lx.set(10, lx.get(10)

```

Kommentar til spørsmål fra forelesning ("Hvorfor er lx deklartert som Liste og ikke ArrayListe"):

Tanken her var egentlig å deklare referansen som type ArrayListe – men dette er jo en grei illustrasjon på at begge deler fungerer for referansen.

Så lenge ArrayListe klassen implementerer interfacet Liste så gjør det ingen forskjell om vi ser objektet gjennom linsen Liste eller ArrayListe.

IN1010 - vår 2020
Siri Moe Jensen

38

Testprogram II

```

main i class TestListe

// ... fortsetter

// 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);

```

IN1010 - vår 2020
Siri Moe Jensen

39

Kjøring av test

Resultatet av testen:

```

$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
- gal parameter til remove gir en uforståelig feilmelding.
- (Det gjelder også get og set.)

IN1010 - vår 2020
Siri Moe Jensen

40

Egne feilmeldinger

- Feilmeldinger bør være en subclasse av passende Exception
- Her: RuntimeException (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);
    }
}
```

IN1010 - vår 2020
Siri Moe Jensen

41

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 Exceptionen-objektet vi "kaster" med **throw**

```
public T remove(int pos) {
    if (pos<0 || pos>=iBruk)
        throw new UlovligListeindeks(pos,iBruk-1);
    ...
}
```

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

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

IN1010 - vår 2020
Siri Moe Jensen

42

Oppdage at noe er feil (kopi med kommentar)

- Vi tar vare på relevant informasjon i metoden **remove**) og sender den med til Exceptionen-objektet vi "kaster" med **throw**

```
public T remove(int pos) {
    if (pos<0 || pos>=iBruk)
        throw new UlovligListeindeks(pos,iBruk-1);
    ...
}
```

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

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

Kommentar til spørsmål fra forelesning ("Stopper programmet når vi har utført try-catch blokken?"):

Svar: Ja – men det er fordi det ikke er flere linjer i testprogrammet. Om vi legger inn en utskrift nedenfor catch-blokken vil denne bli utført. Se nytt testprogram m/kjøring nederst i presentasjonen der unntakshåndtering er inkludert.

Om vi *ikke* fanger unntaket stopper kjøringen.

IN1010 - vår 2020
Siri Moe Jensen

43

Arrayliste med egen feilmelding

```
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 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;
    }
}
```

IN1010 - vår 2020
Siri Moe Jensen

44

Liste-interface implementert med lenkeliste i stedet for array

IN1010 - vår 2020
Siri Moe Jensen

45

Kan vi implementere Liste på en annen måte?

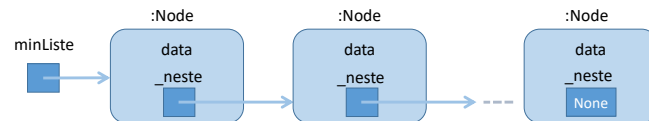
- I forrige eksempel implementerte vi interface Liste ved hjelp av klassen Arrayliste
- Arrayliste bruker en array som datastruktur for objektene – krevde håndtering av fullt array
- Kan vi lage en beholder som lagrer objekter på en mer dynamisk måte – der vi alltid kan ta inn *ett til*?
- Det vi skal lagre 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 nytt hjelpeobjekt!

IN1010 - vår 2020
Siri Moe Jensen

46

Lenkeliste (NB: figur fra IN1000)

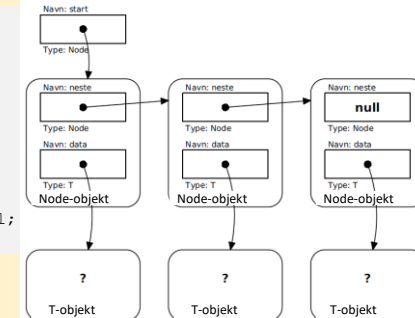


- Poenget med denne strukturen er at for hvert nye objekt vi lager – så lager vi samtidig en referansevariabel som kan referere til et nytt objekt
- dvs hvert objekt må kunne referere til et annet objekt
- dermed får vi en lenket liste av objekter – og trenger bare ha én referanse til det første objektet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

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

```

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



IN1010 - vår 2020
Siri Moe Jensen

48

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**

IN1010 - vår 2020
Siri Moe Jensen

49

Indre klasser

- Klasser kan deklarerer inne i metoder eller andre klasser – om de kun skal brukes der
- En klasse deklartert 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, men med et spesielt navn

IN1010 - vår 2020
Siri Moe Jensen

50

Klassen Lenkeliste: Datastruktur og grensesnitt

```
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) {}
}
```

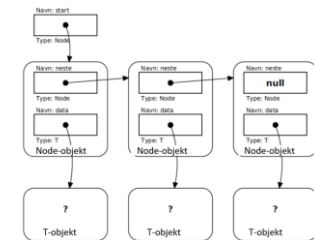
IN1010 - vår 2020
Siri Moe Jensen

51

Hvordan finne størrelsen?

- Går gjennom liste og teller noder!

```
Node p = start;
int n = 0;
while (p != null) {
    n++;
    p = p.neste;
}
```



- Alternativ?

IN1010 - vår 2020
Siri Moe Jensen

52

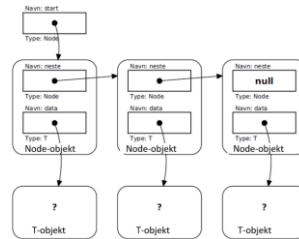
Hvordan hente et element?

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

- Går gjennom liste, teller oss frem til rett plass

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

- NB: Hva skal vi returnere?



IN1010 - vår 2020
Siri Moe Jensen

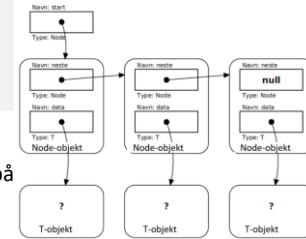
53

Hvordan fjerne et element fra listen?

- Teller oss frem til rett sted: Elementet *før* det som skal fjernes

```
Node p = start;
for (int i = 0; i < pos-1; i++)
    p = p.neste;
Node n = p.neste;
p.neste = n.neste;
```

1. Hvilket element må vi stoppe på
2. Hvilket spesialtilfelle må håndteres her?



IN1010 - vår 2020
Siri Moe Jensen

54

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

IN1010 - vår 2020
Siri Moe Jensen

55

Neste uke

- Andre måter å implementere lenkelister
- Varianter av lister:
 - stabel (stack, Last In First Out – LIFO)
 - k \emptyset (First In First Out – FIFO)
 - Prioritetsk \emptyset
- Mer Java
 - Innpakking ("boxing")
 - Å sammenligne objekter (Interface Comparable)
 - Å gå gjennom alle elementer i en samling (Iterator)

IN1010 - vår 2020
Siri Moe Jensen

56

Lagt til etter forelesning: Exception handling i testprogrammet

```

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

    for (int i = 0; i <= 12; i++)
        lk.add("A"+i);

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

    // Fjern element nr 10:
    lk.set(10, lk.get(10)+"");

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

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

    // Lag en feil:
    try {
        lk.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

```

57

Lagt til etter forelesning: "Hvor kom **data** fra og hva er det"

- I hver av klassene som implementerer interfacet `Liste` har vi en datastruktur for å lagre elementene i samlingen:
 - Arrayliste** lagrer (referanser til) elementene i en array – referansen til denne arrayen kaller vi **data**
 - Innholdet i arrayen **data** refererer, til får vi fra parameteren **x** når metoden **add** kalles for å legge til et nytt objekt i beholderen
- Lenkeliste** klassen lagrer (referanser til) elementene i objekter av den indre klassen **Node**. Hvert node-objekt har to instansvariable: En referanse til en ny node (**neste**) og en referanse til elementet noden tar vare på (**data**).
- Lenkeliste-klassen får også inn nye elementer fra parameteren **x** til metoden **add**, som lager en ny node og sender med **x** til denne som parameter til Node-konstruktøren – som setter sin instansvariable **data** til å referere til det samme som **x**.

IN1010 - vår 2020
Siri Moe Jensen

58

Lagt til etter forelesning: "burde ikke **start** variabelen være statisk?"

Nei, vi trenger en ny datastruktur med starten på de sammenlenkede nodene hver gang vi oppretter en ny Lenkeliste.

Hver **new Lenkeliste<T>** skal lage et nytt objekt av Lenkeliste (en egen beholder), uten tilgang til eller å vite noe om lenkelistene i andre beholdere.

Så hver beholder (Lenkeliste objekt) inneholder én referanse **start**, men kan ha mange node-objekter lenket sammen – som vi får tilgang til fra **start**

IN1010 - vår 2020
Siri Moe Jensen

59

Lagt til etter forelesning: HashMap

- Ikke sentralt i IN1010
- Men veldig praktisk i mange sammenhenger
- Tenk dictionary (ordbok) fra Python
- I Big Java står det mest om bruk av Map – dette er interfacet som HashMap implementerer, og forklarer det meste som trengs ☺.

IN1010 - vår 2020
Siri Moe Jensen

60