

Lister



Hva er en liste?

- Listen er en *lineær* datastruktur
- Hvert element har en *forgjenger*, unntatt *første* element i listen
- Hvert element har en *etterfølger*, unntatt *siste* element i listen
- I motsetning til stack og kø, har vi tilgang til *alle* elementer i en liste
- Vi kan sette inn, fjerne og søke etter elementer i *hele* listen

Tre ulike typer av lister

- **Ordnete lister**
 - Elementene ligger i *sortert* rekkefølge, basert på en innbyrdes sammenligning av elementenes verdi
- **Uordnete lister**
 - Elementene ligger i en eller annen rekkefølge, men denne avhenger *ikke* av elementenes verdi
- **Indekserte lister**
 - Hvert element har et entydig nummer ($0, 1, \dots, n - 1$) som kan brukes til direkte *oppslag* i listen

Vanlige operasjoner på en liste

removeFirst	Fjern første element i listen
removeLast	Fjern siste element
remove	Fjern et bestemt element
first, last	Se på første/siste element
contains	Søk etter et bestemt element
isEmpty	Sjekk om listen er tom
size	Antall elementer i listen
add...	Innsetting av ny verdi, løses på ulike måter for ordnet og uordnet liste

Grensesnitt for en liste-ADT i Java *

```
public interface ListADT<T> extends Iterable<T>
{
    public T removeFirst();
    public T removeLast();
    public T remove(T element);
    public T first();
    public T last();
    public boolean contains(T target);
    public boolean isEmpty();
    public int size();
    public Iterator<T> iterator();
}
```

* Som i læreboken

Iterator?

- Java-metode som returnerer et Iterator-objekt
- Brukes til å gå gjennom en hel datastruktur på *en eller annen* systematisk måte – typisk fra første til siste element for en liste – med *detaljene skjult*
- Viktigste Iterator - metoder:
 - hasNext: Sjekker om det er flere elementer å gå gjennom
 - next: Returnerer neste element i gjennomgangen
- De fleste Java-collections har en eller flere iterator-metoder implementert – og de *kan* være litt *skumle*
- Se kapittel 7 og *koden i læreboka* for mer om iteratorer

Typisk bruk av en iterator

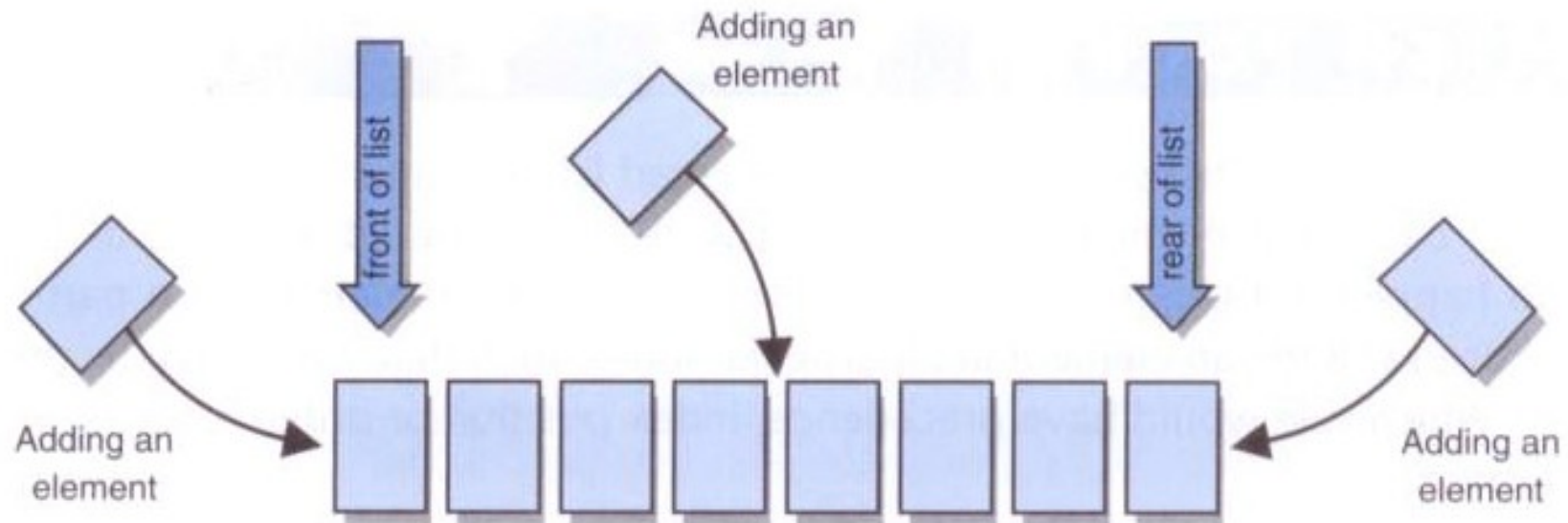
```
ArrayList<String> L = new ArrayList<String>();  
[ ... ]  
Iterator itr = L.iterator();  
  
while( itr.hasNext() )  
{  
    String S = (String) itr.next();  
    System.out.println(S);  
}
```

Innsetting av nytt element i *uordnet* liste

- Uordnet:
 - Kan sette inn elementer hvor som helst i listen
- Læreboken implementerer tre metoder for innsetting:
 - addToFront Legger til nytt element først i listen
 - addToRear Legger til nytt element sist i listen
 - addAfter Legger til nytt element etter et
bestemt element i listen

ADT for unordered list

```
public interface UnorderedListADT<T>
    extends ListADT<T>
{
    public void addToFront(T element);
    public void addToRear(T element);
    public void addAfter(T element, T target);
}
```

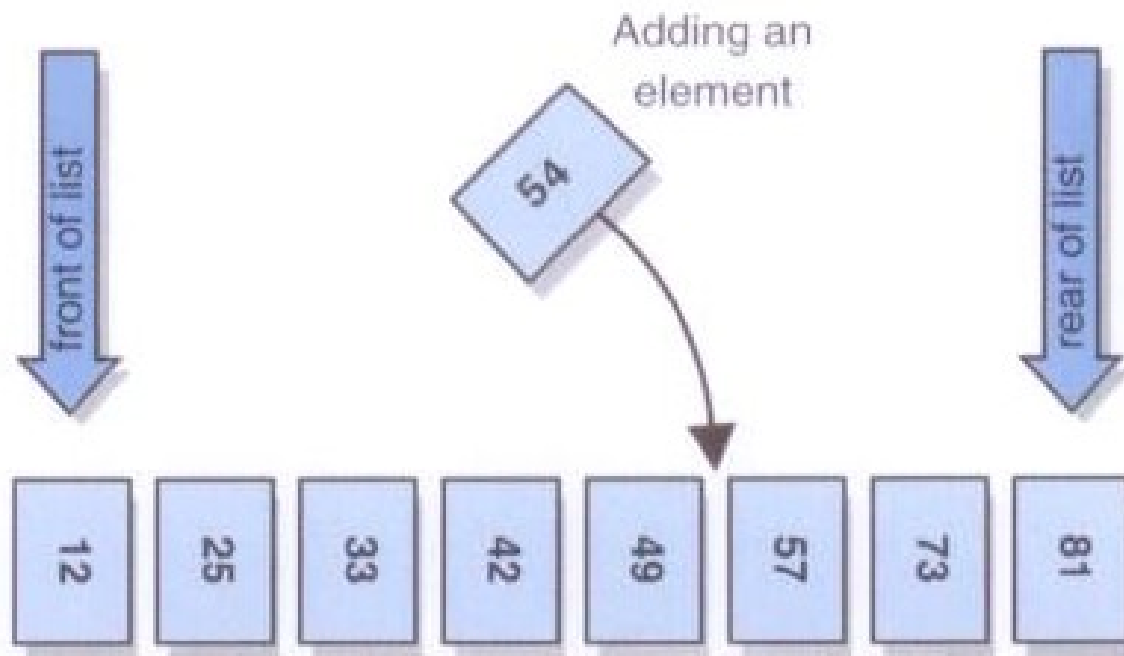


Innsetting nytt element i *ordnet* liste

- Ordnet liste:
 - Kan ikke sette inn slik at sorteringen ødelegges
 - Listen må selv sørge for dette
 - Tilbyr derfor kun én add - metode for å sette inn elementer
- Elementene i en ordnet liste må kunne sammenlignes med hverandre for å bestemme sorteringen:
 - I Java løses dette ved å kreve at alle elementer i listen tilhører subklasser av datatypen Comparable
 - Dette ligger *ikke* i ADT'en, men sjekkes i stedet av add – metoden i de ulike implementasjonene av ordnet liste

ADT for ordnet liste

```
public interface OrderedListADT<T>  
    extends ListADT<T>  
{  
    public void add(T element);  
}
```



Eksempel på bruk av ordnet liste:

Oppsett av en turnering

- Et antall lag skal spille en cup/utslags-turnering
- Sett opp *første runde* i turneringen slik at:
 - Beste lag møter dårligste lag
 - Nest beste lag møter nest dårligste lag, etc.
- Input:
 - Antall lag, må være en potens av 2: 4, 8, 16, 32, ...
 - Navn på hvert lag
 - En rangering av hvert lag, i form av antall kamper vunnet

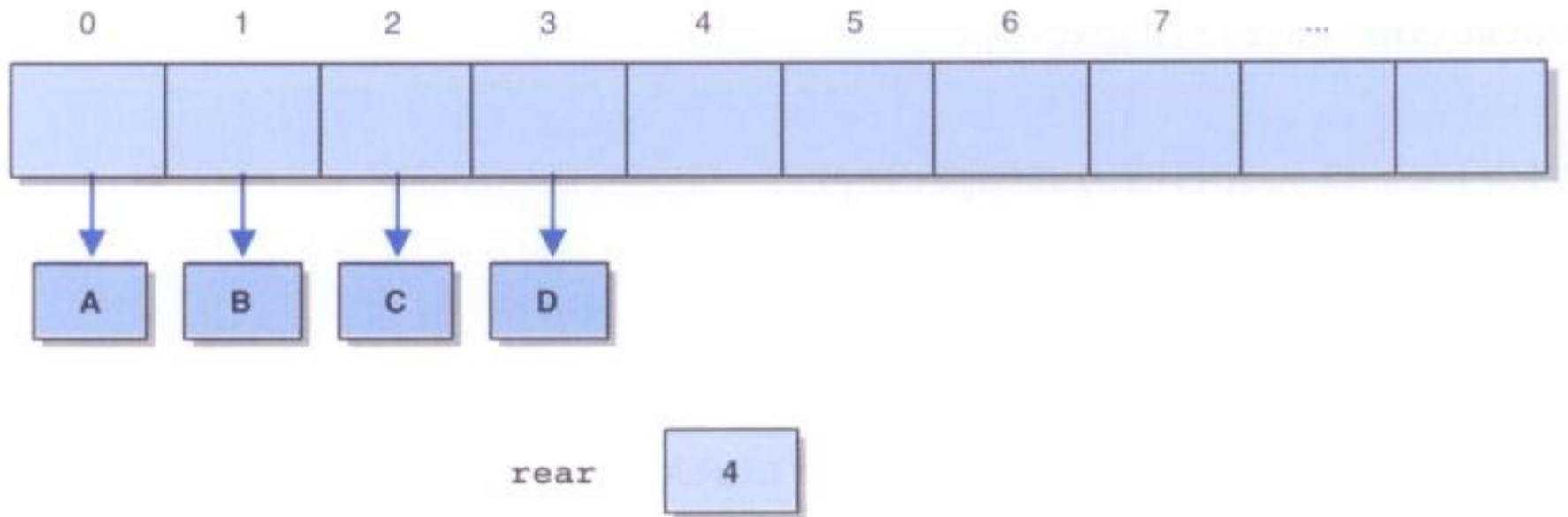
Turnering: Implementasjon

- Egen **klasse for lagene** som implementerer Comparable:
 - Inneholder lagets navn og antall kamper vunnet
- Leser inn antall lag og data for hvert lag fra bruker
- Legger alle lag i en liste sortert på antall kamper vunnet
- Gå i en løkke inntil listen er tom:
 - Ta ut første og siste element/lag i listen
 - Skriv ut at disse to lagene skal spille mot hverandre
- Java-program: **TournamentMaker.java**

Lister implementert med array

- Bruker en array av referanser til objekter
- Fast initiell lengde på array, utvider når nødvendig
- Begynnelsen på listen er alltid i indeks 0 (null)
 - Unødvendig å bruke sirkulær array som for køer
 - Vi må *uansett* flytte elementer, fordi fjerning og innsetting *ikke* bare kan skje i endene av listen
- Trenger bare å lagre antall elementer i listen i tillegg til selve arrayen
- Kode fra læreboka: [ArrayList.java](#)

Lister implementert med array



Liste med array: Fjerning av element

- Tre metoder:

```
public T removeFirst();  
public T removeLast();  
public T remove(T element);
```

- Fjerning av siste element er trivielt, $O(1)$
- Fjerning av første element eller et element inne i listen krever at alle elementer *etter* det som er fjernet flyttes en plass til venstre for å unngå et “tomt hull” i arrayen
- `remove` krever i tillegg et søk (intern metode `find`)
- Fjerning blir derfor generelt en $O(n)$ -operasjon for en array-liste med n elementer

Liste med array: Søke etter element

- Metode:

```
public boolean contains(T target);
```

- Læreboka bruker her samme interne metode `find` som for fjerning av et bestemt element i listen
- Søking er en $O(n)$ -operasjon for *uordnet* array-liste med n elementer, fordi vi i verste fall må gå gjennom hele listen
- Søking kan implementeres som en $O(\log n)$ operasjon for *sorterte* array-lister, hvis vi bruker *binærsøk**

*: Beskrevet i kapittel 9 i læreboka

Liste med array:

Innsetting av element i *uordnet* liste

- Tre metoder:

```
public void addToFront(T element);  
public void addToRear(T element);  
public void addAfter(T element, T target);
```

- Innsetting sist i listen er trivielt, $O(1)$
- Innsetting først eller inne i listen krever at alle elementer etter det nye først flyttes en plass til høyre
- `addAfter` krever i tillegg et søk for å finne posisjonen
- Innsetting blir generelt en $O(n)$ -operasjon
- Kode fra læreboka: [ArrayUnorderedList.java](#)

Liste med array:

Innsetting av element i *ordnet* liste

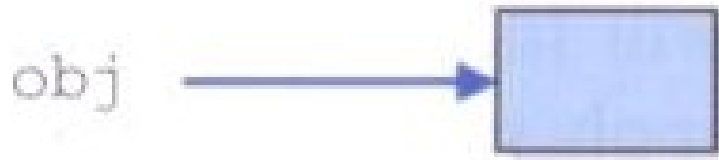
- Metode:

```
public void add(T element);
```

- Må gå gjennom array fra begynnelsen inntil vi finner posisjonen/indeksen der nytt element skal inn
- Deretter må alle elementer etter det nye flyttes en plass til høyre, før nytt element settes inn riktig
- Innsetting er en $O(n)$ -operasjon for en ordnet array-liste med n elementer
- Kode fra læreboka: [ArrayOrderedList.java](#)

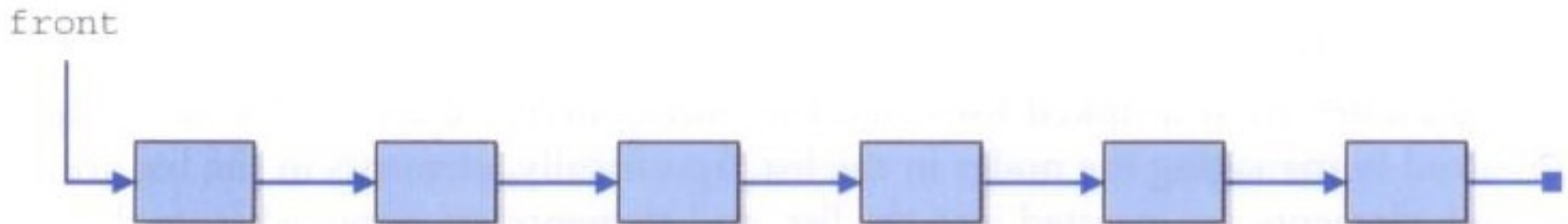
Lenkede datastrukturer

- Et alternativ til array-baserte implementasjoner er å bruke lenkede datastrukturer
- En lenket datastruktur i Java bruker *objektreferanser* til å lage lenker (eller forbindelser) mellom objektene
- Variabler i Java som lagrer referanser til objekter inneholder egentlig *minneadressen* til dataene som utgjør objektet (kalles *pekere* i språk som f.eks. C)



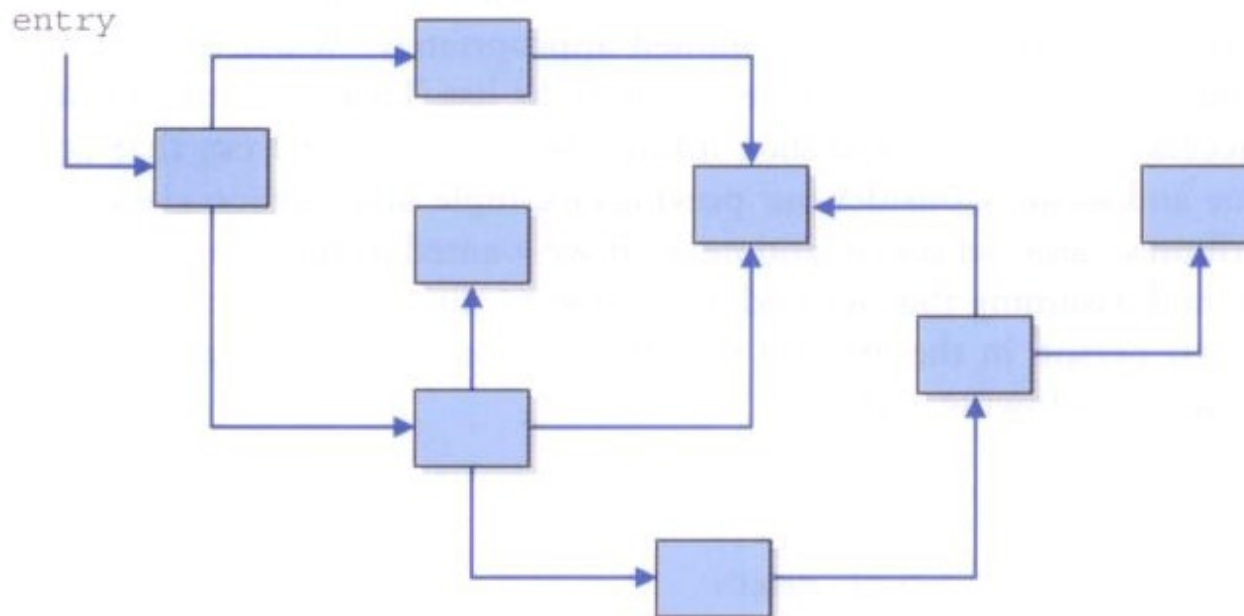
Eksempel: Lenket datastrukturer

- Data om personer lagres i objekter av en Java-klasse `Person`
- Hvert objekt inneholder også en referanse til et annet `Person`-objekt
- En sekvens av `Person`-objekter kan danne en lineær datastruktur som kalles en *lenket liste*
- Objektene i en slik datastruktur kalles *noder*



Ikke-lineære lenkede datastrukturer

- Hvis hver node inneholder *flere* referanser til andre noder, kan de lenkes sammen til å danne mer kompliserte *ikke-lineære* datastrukturer
- I dette kurset skal vi senere se på de ikke-lineære datstrukturene *trær* og *grafer*



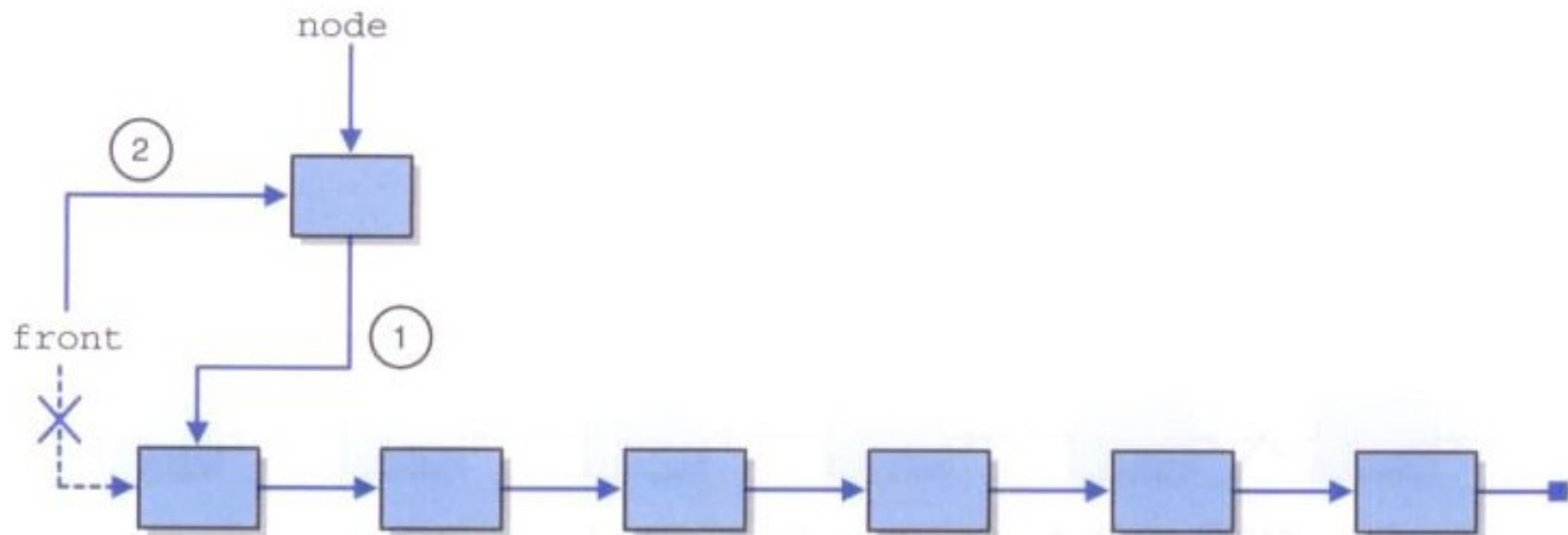
Lenket liste

- Lenkede lister har ingen indekser for direkte oppslag
- Det er bare en referanse til noden i *starten* på listen
- For å aksessere hvert element i listen må vi følge referansene fra en node til neste node:

```
Person current = first;
while (current != null)
{
    System.out.println(current);
    current = current.next;
}
```

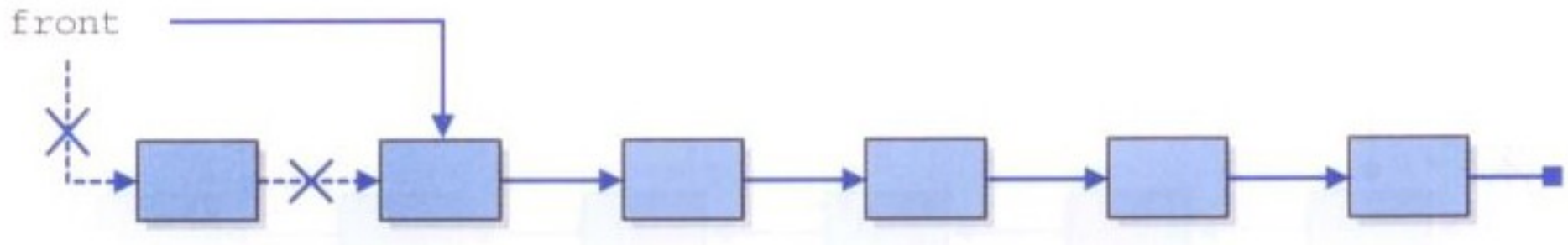
Håndtering av lenker

- Operasjoner på lenkede lister må være nøyaktige, og sikre at referanser/pekere er korrekte og konsistente
- For f.eks. å sette inn en ny node først i listen:
 1. Sett ny node til å peke på nåværende første node
 2. Sett starten på listen (front) til å peke på den nye noden



Håndtering av lenker forts.

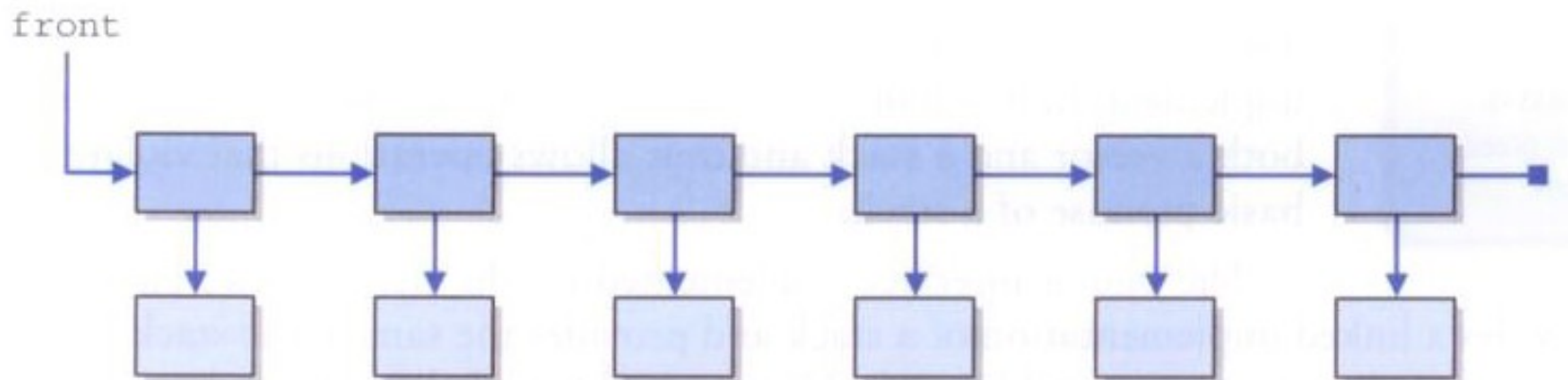
- Tilsvarende, for å fjerne første node:
 - Flytt pekeren til starten på listen (front) til å referere til nåværende første node sin etterfølger / neste.
 - Hvis noden som fjernes skal brukes senere, må det tas vare på en referanse til noden *før* “front”-pekeren flyttes, hvis ikke er den å regne som fjernet fra hukommelsen



I læreboka: Generiske lenkede lister

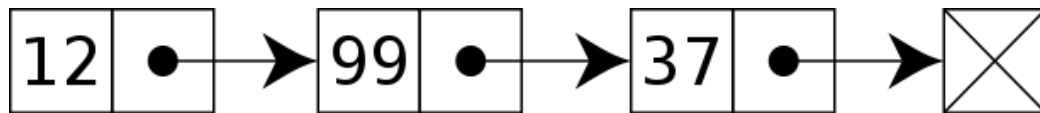
- En generell datastruktur der koblingene/ referansene mellom nodene er uavhengige av dataene som skal lagres i strukturen
- Løsning: Bruk en egen node-klasse som bare inneholder:
 - En referanse til neste node i listen (null for siste node)
 - En generisk referanse til objektet som er lagret i listen
- Java-kode fra læreboka:

`LinearNode.java` og `LinkedList.java`



En enklere implementasjon av uordnede lenkede lister

- For å fokusere på metodene og ikke på Java, implementer vi i stedet en uordnet lenket liste som bare kan lagre enkle heltall:



- Klassen som representerer en slik lenket liste inneholder:
 - En indre klasse for nodene: `ListNode`
 - Referanse til første node i listen: `head`
 - Antall elementer i listen: `size`
- Implementasjon av lenket liste med heltall: [intList.java](#)

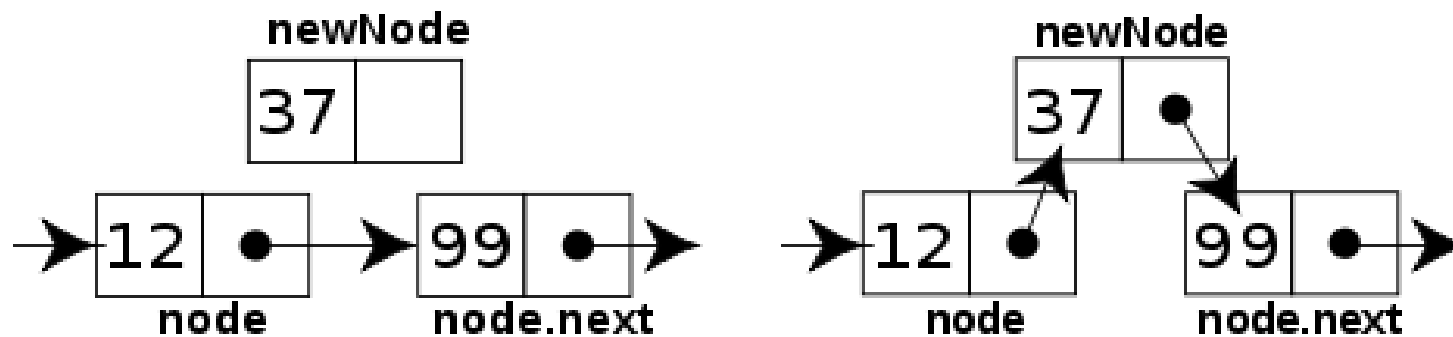
Innsetting av element i lenket liste

- Tre metoder:

```
void    addToFront(int data);  
void    addToRear (int data);  
boolean addAfter   (int target, int data);
```

- Innsetting først er trivielt, `addToFront` blir $O(1)$
- `addAfter` og `addToRear`:
 - Krever at listen traverseres fra begynnelsen
 - Innsettingen er litt “fiklete”, må oppdatere forgjenger
 - Begge metodene har worst-case som er $O(n)$

Innsetting av element i lenket liste



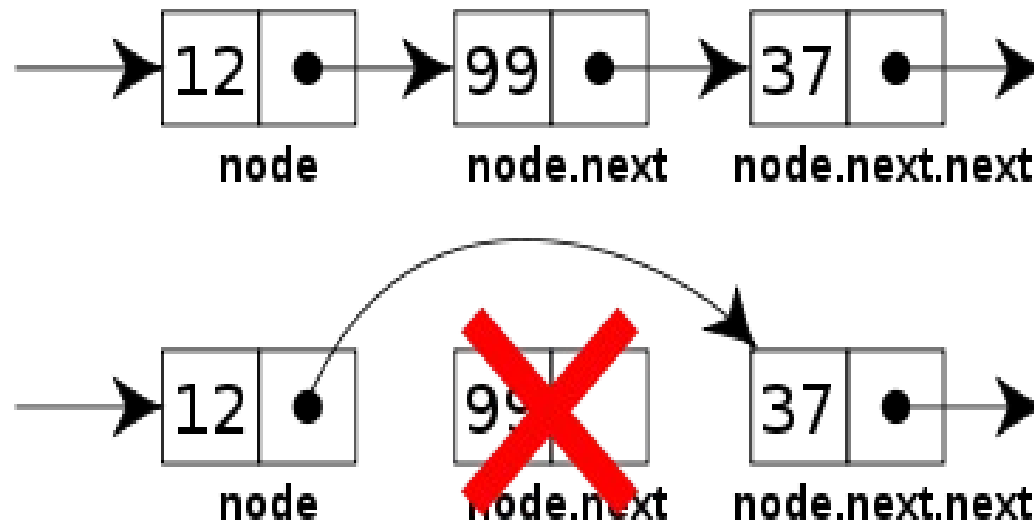
Fjerning av element i lenket liste

- Tre metoder:

```
int  removeFirst();  
int  removeLast ();  
void remove      (int data);
```

- Fjerning av første element er enkelt, removeFirst blir $O(1)$
- Fjerning av siste element eller et element inne i listen:
 - Traversér listen for å finne noden *før* den som skal fjernes
 - Fjern elementet ved å sette neste-pekeren til *forgjenger*-noden til å peke på fjernet node sin *etterfølger*
 - Implementasjon er “fiklete”, må håndtere spesialtilfeller
 - Begge metodene har worst-case som er $O(n)$

Fjerning av element i lenket liste



Lenket liste: Søke etter element

- Metode: `boolean contains(int data);`
- Må alltid gå fra begynnelsen av listen og sammenligne med en og en node inntil vi evt. finner verdien
- Verste tilfelle er n steg, gjennomsnittlig $n/2$
- Søking i en lenket liste med n elementer er $O(n)$
- Kan ikke effektiviseres, selv om listen er sortert
- Men: Hvis vi innfører to “neste-pekere” i hver node, kan vi lage en ordnet datastruktur som har $O(\log n)$ oppførsel for innsetting, fjerning og søking(!) – et binært søketre*

* Lærebokas kapittel 11

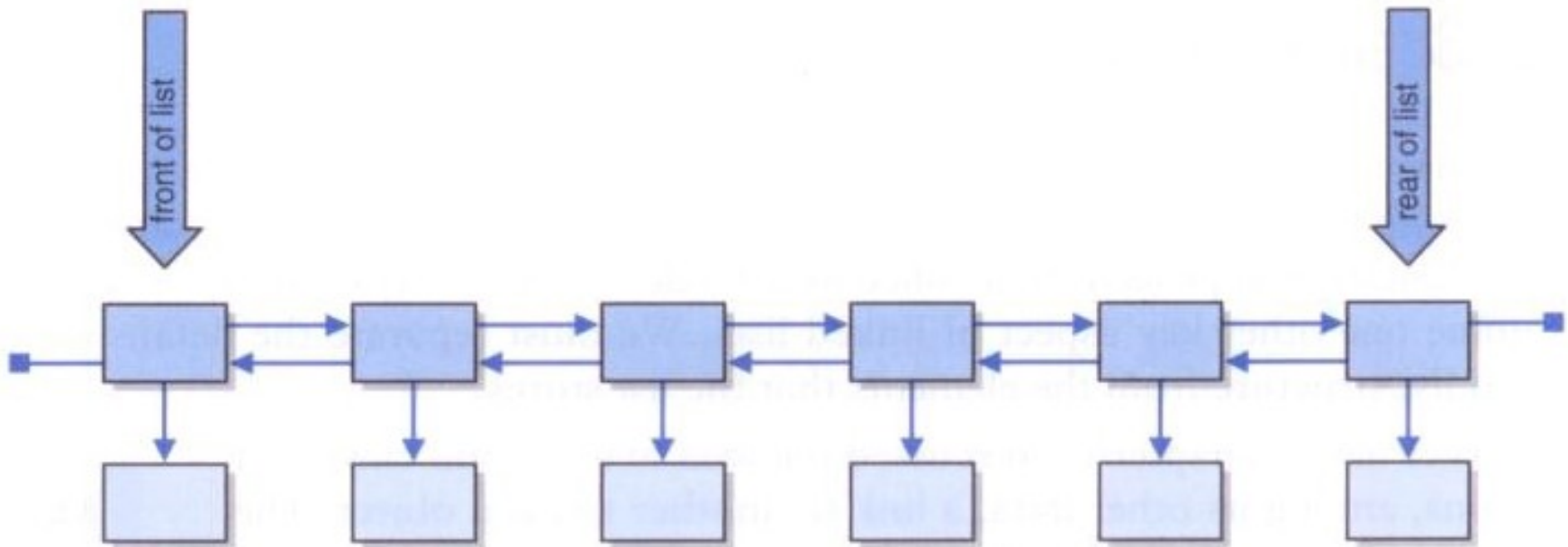
Vanlige varianter av lenket liste

- Liste med “hode” (og “hale”):
 - Egen “dummy” (aka “sentinel” / ”vaktpost”) node for begynnelsen (og evt. slutten av listen)
 - Slipper unna noen spesialtilfeller ved innsetting og fjerning, fordi listen aldri er tom
- Sirkulære lister:
 - Siste node peker til første

Dobbeltlenkede lister

- Hver node har peker både til neste og til forrige node
- Enklere koding av innsetting og fjerning
- Kan traverseres “forlengs og baklengs”
- Typisk anvendelse: Redigering av en linje med tekst
- Implementasjon:
 - `DoubleNode.java`
 - `DoubleList.java`

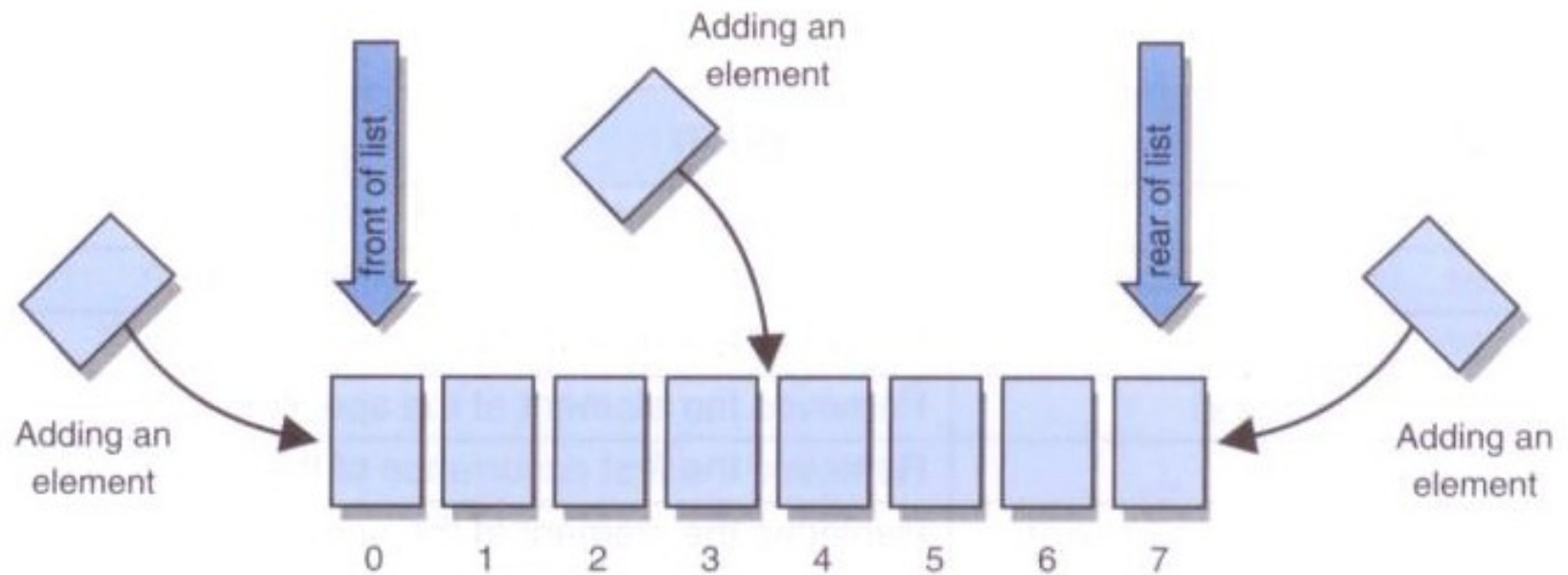
Dobbeltlenket liste



Indekserte lister

- Har samme metoder som en vanlig liste
- Tilbyr i tillegg metoder for å se på, sette inn og fjerne elementer på en bestemt *posisjon* (eller *indeks*) i listen
- Indeksene til elementer i listen vil oftest endres ved innsetting og fjerning
- En indeksert liste er vanligvis ikke ordnet (hvorfor?)
- Indekserte lister er implementert i Java Collections

Indeksert liste



Lister i Java Collections API

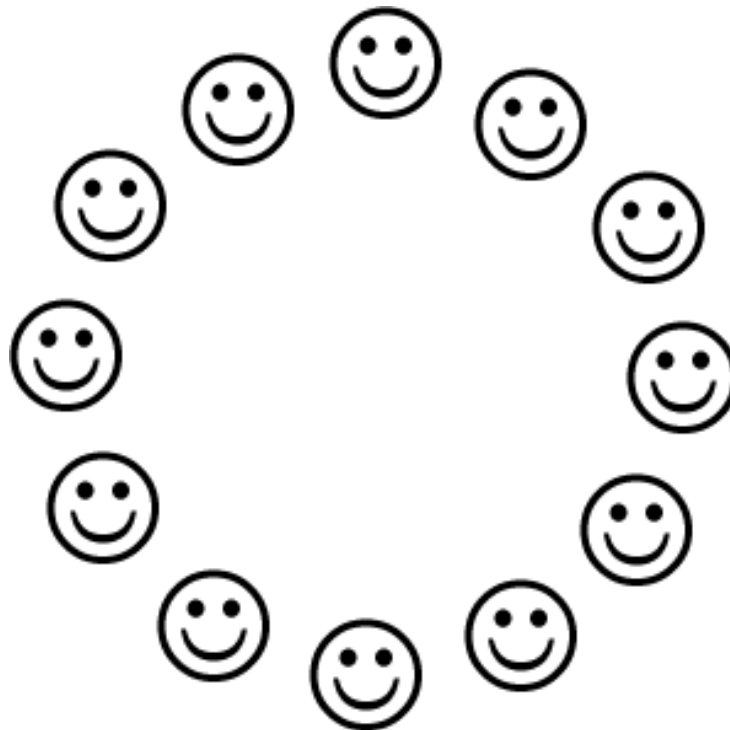
- Generell liste-ADT:
 - `java.util.List`
- Tre ulike implementasjoner av lister i Java:
 - Array-liste: `java.util.Vector` (synkronisert)
 - Array-liste: `java.util.ArrayList` (ikke synkronisert)
 - Lenket liste: `java.util.LinkedList` (ikke synkronisert)
- Alle tre implementerer indekserte lister
- Java-listene inneholder også metoder som implementerer stack og ulike typer køer

Noen metoder fra `java.util.List` som implementerer indekserte lister

Metode	Beskrivelse
<code>add(int index, E element)</code>	Setter inn element på gitt indeks
<code>get(int index)</code>	Returnerer element på gitt indeks
<code>remove(int index)</code>	Fjerner element på gitt indeks
<code>set(int index, E element)</code>	Erstatter element på gitt indeks

En anvendelse av indekserte lister: Josephus' problem

- 2000 år gammel historie, se [Wikipedia](#)
- [Animasjon:](#)



Josephus' problem som en liste

- Input: Liste med tallene 1, 2, 3, ..., n
- Ta ut (f.eks.) hvert tredje element fra en liste, inntil bare ett element står igjen
- Eksempel med $n = 7$:

1 2 **3** 4 5 6 7

1 4 5 **7**

4

1 2 4 5 **6** 7

1 4 **5**

1 **2** 4 5 7

1 4

Implementasjon av Josephus' problem

- Bruker en indeksert liste til å lagre tallene
- Listen behandles som om den var sirkulær – vi går til begynnelsen når vi kommer til slutten av listen ved å bruke enkel modulus-regning
- Vi må ta hensyn til at listen kollapse mens vi er i gang, og hele tiden holde rede på *indeksen* til *neste* element som skal ut

Algoritme for Josephus' problem*

Input: n Antall elementer

d Avstand mellom elementene som fjernes

f Første element som skal fjernes

- 1 Legg inn tallene $1, 2, 3, \dots, n$ i en indeksert liste L
- 2 Sett indeks til første element som fjernes: $p = f - 1$
- 3 Så lenge $n > 1$
 - 3.1 Fjern elementet på indeks p i listen
 - 3.2 $n = n - 1$
 - 3.3 $p = (p + d - 1) \% n$

-
- Implementasjon: [Josephus_2.java](#)

* Litt anderledes enn [koden i læreboka](#)

Lister: Oppsummering av effektivitet

Array-liste:

- Fordeler:
 - Enkel implementasjon
 - Innsetting og fjerning sist i listen er $O(1)$
 - Direkte oppslag, $O(1)$
 - Søking i ordnet liste kan være $O(\log n)$
- Ulemper:
 - Innsetting og fjerning først og inne i liste $O(n)$

Lenket liste:

- Fordeler:
 - Innsetting først og sist $O(1)$
 - Fleksibel, flere varianter
- Ulemper:
 - Fiklete implementasjon
 - Ikke direkte oppslag
 - Søking, innsetting og fjerning inne i liste $O(n)$