IN1010

Repetisjon Uke 6
Beholdere, del 1

(Basert på Lars Tveitos presentasjon)

28. februar, 2025

Beholdere (Collections)

En beholder er en datastruktur for en samling elementer

Datastruktur er hvordan vi strukturerer data i f.eks. minne

Arrays, lister, mengder, ordbøker er eksempler på beholdere

Pekere og referanser

I datastrukturer er peker noe som peker på et element

Referanser er en variabel som peker på et objekt

Verdier er variabler med en gitt verdi (primitive typer)

Peker (pointer) brukes også som begrep for minneadresser, men i Java så er det noe underliggende som vi normalt ikke trenger å tenke på.

Det er likevel ofte at både datastrukturpeker, referanser og variabler tilsvarer en minnepeker.

Beholdere i Java

Vi har noen beholdere som kan inneholde primitive typer (arrays)

Men de fleste er generiske, som da er samling av objekter av en gitt type. Det er nærmere forklart neste uke

Vi kan også lage egne beholdere

Beholdere i Java

Beholdere er klasser, men også med et interface.

Interfacet er det vil helst bruker når vi skal lage referanse, og har de metodene vi trenger

En beholder har normalt gjennom interface en måte å samle operasjoner vi bruker på beholderen.

Som kan være metoder for å legge til, fjerne og hente ut elementer, nullstille og finne størrelsen til beholderen

Array

Array er en sammenhengende samling av elementer

Siden den er sammenhengende så må den opprettes med en gitt størrelse

ArrayList

Men vi kan også ha en dynamisk array, som betyr at vi må ha en tilleggsoperasjon som kan utvide arrayet, samtidig som det fortsatt skal være sammenhengende.

ArrayList er Java sin versjon av dynamisk array

Lister

Lister er en samling av elementer i en bestemt rekkefølge

ArrayList er en type liste

Vi kan bruke posisjon for de fleste operasjonene

Stack

Stack er en type liste hvor operasjoner gjøres på enden av listen

Vi setter inn (push) et element, det vil da bli lagt til til slutt

Når vi henter ut (pop) så hentes det siste elementet som ble lagt til

Kø

Vi kan også ha en struktur hvor vi setter inn fra den ene enden, og henter ut fra den andre.

Vi har da offer (også kalt enqueue) for å sette inn,

og poll (dequeue) for å ta ut

Lenkede Lister

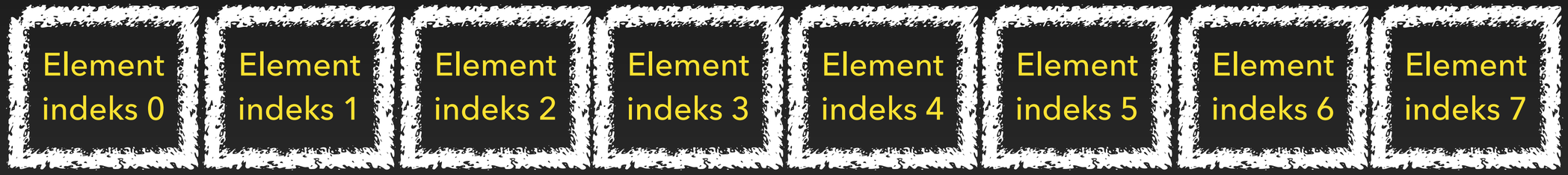
Dette er lister som består av en lenke av elementer, imotsetning til en array. Elementene lagres i det vi kalle noder, som tilsvarer en lenke. En node består av elementet og pekere til neste lenke.

Det er mulig å bruke posisjon, men da må vi gå gjennom lenkene for å komme til riktig node

Enkeltlenkede lister har en peker til neste node i listen, men vi kan også ha flere pekere, f.eks. dobbeltlenkede har pekere til forrige

Minnestruktur

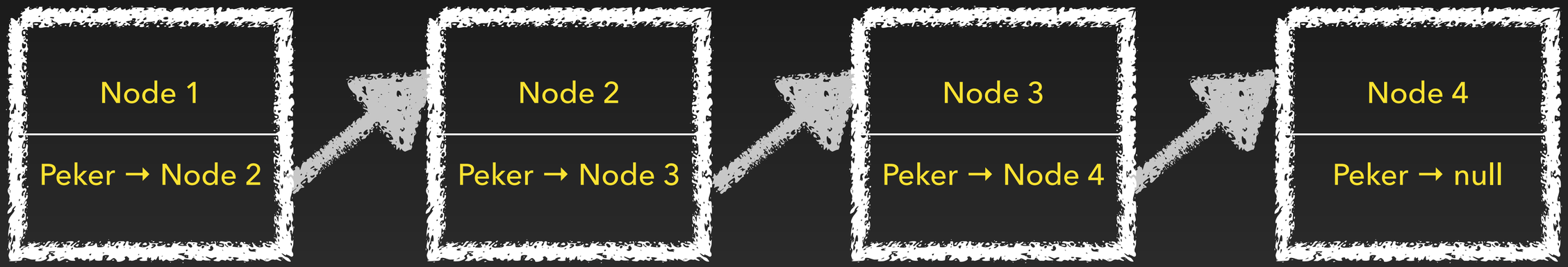
Vi har flere typer fordi vi kan ha ulike behov for lagring av strukturene



Array

Minnestruktur

Vi har flere typer fordi vi kan ha ulike behov for lagring av strukturene



Lenkeliste

Eksempel på to ulike lister

```
interface StrengListe {
  public int antall();
  public void settInnFørst(String element);
  public void settInnSist(String element);
  public void fjernFørste();
  public void fjernSiste();
  public String hentFørste();
  public String hentSiste();
  public String hent(int indeks);
}
```

Vi starter med et interface som kan brukes for begge

Metodene

```
interface StrengListe {
  public int antall();
  public void settInnFørst(String element);
  public void settInnSist(String element);
  public void fjernFørste();
  public void fjernSiste();
  public String hentFørste();
  public String hentSiste();
  public String hent(int indeks);
}
```

Vi trenger en metode for å hente ut antall elementer

Metodene

```
interface StrengListe {
  public int antall();
  public void settInnFørst(String element);
  public void settInnSist(String element);
  public void fjernFørste();
  public void fjernSiste();
  public String hentFørste();
  public String hentSiste();
  public String hent(int indeks);
}
```

Så noen metoder for å sette inn et element

Metodene

```
interface StrengListe {
  public int antall();
  public void settInnFørst(String element);
  public void settInnSist(String element);
  public void fjernFørste();
  public void fjernSiste();
  public String hentFørste();
  public String hentSiste();
  public String hent(int indeks);
}
```

Så kan vi ha noen for å fjerne elementer

Metodene

```
interface StrengListe {
  public int antall();
  public void settInnFørst(String element);
  public void settInnSist(String element);
  public void fjernFørste();
  public void fjernSiste();
  public String hentFørste();
  public String hentSiste();
  public String hent(int indeks);
}
```

Og også hente ut elementer

Liste som dynamisk array

```
class DynamiskStrengListe implements StrengListe {
  private int antall;
  private String[] elementer;
  DynamiskStrengListe() {
    antall = 0;
    elementer = new String[1];
  }
}
```

Vi lager en klasse for listen som implementerer interfacet

Liste som dynamisk array

```
class DynamiskStrengListe implements StrengListe {
  private int antall;
  private String[] elementer;
  DynamiskStrengListe() {
    antall = 0;
    elementer = new String[1];
  }
}
```

Som en start så lager vi en array for elementene

Liste som dynamisk array

```
class DynamiskStrengListe implements StrengListe {
  private int antall;
  private String[] elementer;
  DynamiskStrengListe() {
    antall = 0;
    elementer = new String[1];
  }
}
```

Vi sette antallet til 0, siden vi har ingen elementer

Liste som dynamisk array

```
class DynamiskStrengListe implements StrengListe {
  private int antall;
  private String[] elementer;
  DynamiskStrengListe() {
    antall = 0;
    elementer = new String[1];
  }
}
```

Men arrayet oppretter vi i konstruktøren med størrelse 1

Finne antall

```
public int antall() {
  return antall;
}
```

Denne metoden returnerer bare antallet, så det betyr at den variabelen må følge det faktiske antallet

Sette inn elementer

```
public void settInnSist(String element) {
  sikreKapasitet();
  elementer[antall++] = element;
}
```

Først lager vi en metode som setter inn element sist i listen

Sette inn elementer

```
public void settInnSist(String element) {
  sikreKapasitet();
  elementer[antall++] = element;
}
```

Siden listen ved opprettelse bare har plass til ett element, så må sørge for at vi kan gi den mer plass ved behov

Sette inn elementer

```
public void settInnSist(String element) {
  sikreKapasitet();
  elementer[antall++] = element;
}
```

Så setter vi inn elementet i posisjonen "antall" og øker antallet med 1

Husk at antall er det samme som siste indeks + 1 (siden vi indekserer på 0)

Gi plass

```
private void sikreKapasitet() {
  if (antall < elementer.length) {
    return;
  }
  String[] ny = new String[elementer.length * 2];
  for (int i = 0; i < antall; i++) {
    ny[i] = elementer[i];
  }
  elementer = ny;
}
```

For å sørge for at vi har nok plass, så har vi en privat metode i klassen

Gi plass

```
private void sikreKapasitet() {
  if (antall < elementer.length) {
    return;
  }
  String[] ny = new String[elementer.length * 2];
  for (int i = 0; i < antall; i++) {
    ny[i] = elementer[i];
  }
  elementer = ny;
}
```

Hvis vi har nok plass, så gjør vi ingenting

Men vi kunne også krympet den hvis vi vil

Gi plass

```
private void sikreKapasitet() {
  if (antall < elementer.length) {
    return;
  }
  String[] ny = new String[elementer.length * 2];
  for (int i = 0; i < antall; i++) {
    ny[i] = elementer[i];
  }
  elementer = ny;
}
```

Så oppretter vi ett nytt array, her så dobbler vi størrelsen

Gi plass

```
private void sikreKapasitet() {
  if (antall < elementer.length) {
    return;
  }
  String[] ny = new String[elementer.length * 2];
  for (int i = 0; i < antall; i++) {
    ny[i] = elementer[i];
  }
  elementer = ny;
}
```

Siden vi lager et nytt array, så må vi kopiere den gamle til den nye

Gi plass

```
private void sikreKapasitet() {
  if (antall < elementer.length) {
    return;
  }
  String[] ny = new String[elementer.length * 2];
  for (int i = 0; i < antall; i++) {
    ny[i] = elementer[i];
  }
  elementer = ny;
}
```

Og til slutt så endrer vi referansen til listen vår til det nye arrayet

Sette inn elementer

```
public void settInnFørst(String element) {
  sikreKapasitet();
  for (int i = antall; 0 < i; i--) {
    elementer[i] = elementer[i - 1];
  }
  elementer[0] = element;
  antall++;
}
```

For å sette inn først, så er vi nødt til å flytte alle elementene ett hakk for å gi plass

Fjerne et element

```
public void fjernSiste() {
  if (antall == 0) {
    return;
  }
  elementer[antall--] = null;
}
```

Vi fjerner elementer ved å sette de til *null*

Vi må også huske å endre antallet med en mindre

Fjerne et element

```
public void fjernSiste() {
  if (antall == 0) {
    return;
  }
  elementer[antall--] = null;
}
```

Hvis vi ikke har elementer, så gjør vi ikke noe

Fjerne et element

```
public void fjernFørste() {
  if (antall == 0) {
    return;
  }
  for (int i = 0; i < antall - 1; i++) {
    elementer[i] = elementer[i + 1];
  }
  antall--;
  elementer[antall] = null;
}
```

For å fjerne første må vi nå flytte alle elementer et hakk tilbake

Siste elementet bør settes til *null*

Hente ut element

```
public String hentFørste() {
  if (antall == 0) {
    return null;
  }
  return elementer[0];
}
```

For å hente ut første element er det bare å returnere elementet på indeks 0

Hente ut element

```
public String hentFørste() {
  if (antall == 0) {
    return null;
  }
  return elementer[0];
}
```

Men vi sjekker også at vi har elementer, hvis ikke returnerer vi *null*

Hente ut element

```
public String hentSiste() {
  if (antall == 0) {
    return null;
  }
  return elementer[antall-1];
}
```

Siste elementet har samme indeks som "antall" og trukket fra 1

Hente ut element

```
public String hent(int indeks) {
  if (indeks < 0 || antall() <= indeks) {
    throw new IndexOutOfBoundsException();
  }
  return elementer[indeks];
}
```

Hvis indekset ikke er gyldig, så kaster vi et unntak