Repetisjon: beholdere

IN1010

20 mai 2025

Hva er en beholder?

- En beholder er en **datastruktur** som brukes til å **oppbevare elementer**.
  - f. eks variabler, objekter etc
- F.eks Array, ArrayList, Hashmaps
- En beholder vil ha operasjoner/metoder
  - legger til
  - fjerner,
  - henter elementer

Array

- Trenger ikke å importere bibliotek
- Fast størrelse
- Kan holde på primitiver(char, int, double
- Begrenset funksjonalitet

```
int[] array = new int[3];
int[] tall = {1,2,6};
```

ArrayList

- Må imortere
- Dynamisk størrelse
- Må være referansetyper/objekter for å kunne blir lagret i beholderen
- Har mange innebygde metoder

```
import java.util.ArrayList;
```

```
ArrayList<Integer> liste = new ArrayList<>();
liste.add(e:10);
```

Noen av metodene til ArrayList

Modifier and Type	Method	Description
boolean	`add(E e)`	Appends the specified element to the end of this list.
void	`add(int index, E element)`	Inserts the specified element at the specified position in this list.
boolean	`addAll(Collection<? extends E> c)`	Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	`addAll(int index, Collection<? extends E> c)`	Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	`clear()`	Removes all of the elements from this list.
`Object`	`clone()`	Returns a shallow copy of this `ArrayList` instance.
boolean	`contains(Object o)`	Returns `true` if this list contains the specified element.
void	`ensureCapacity(int minCapacity)`	Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
void	`forEach(Consumer<? super E> action)`	Performs the given action for each element of the `Iterable` until all elements have been processed or the action throws an exception.

Generiske beholder

- Ønsker at beholder skal kunne brukes på **flere forskjellige typer.**
- Bruker da **typeparameter** som en “placeholder”
- Lager da en **generisk klasse** ved å <T> eller <E> i klassen
- Når en liste skal opprettes må man spesifisere hviken type man skal bruke
  - ArrayList<String> liste = new ArrayList<>()

Konvensjon typeparametere

- E - element
- K - key
- V - value
- N - number
- **T** - type

Generisk beholder eksempel

Beholder.java > ...

```java
public class Beholder<T> {
    private T innhold;

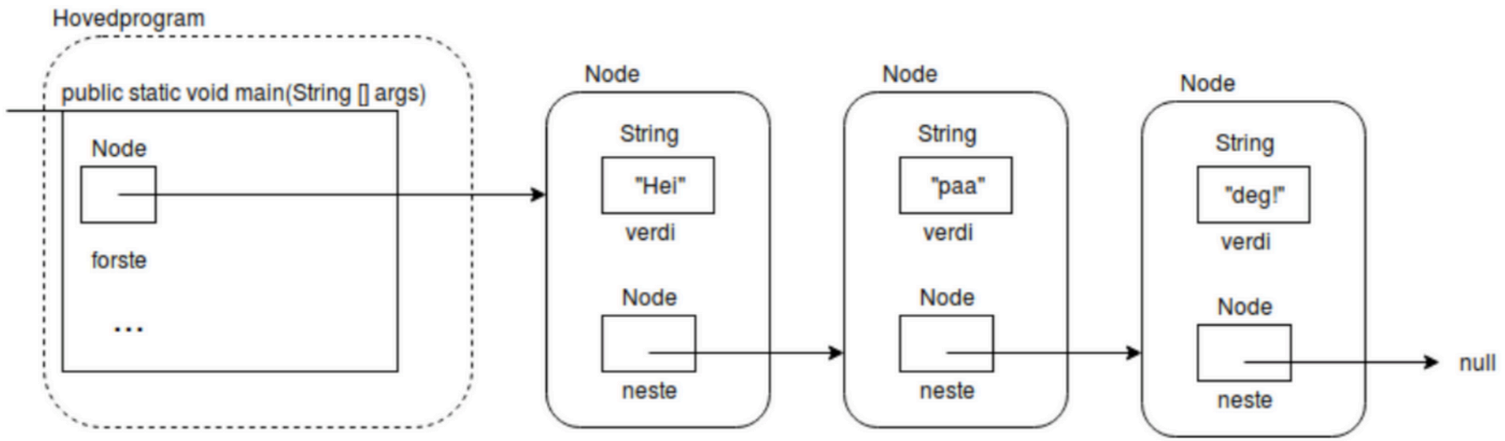
    public Beholder(T innhold){
        this.innhold = innhold;
    }

    public T hentInnhold(){
        return innhold;
    }
```

Live koding av enkel node-klasse

Oppgave

- Skriv et hovedprogram, hvor du opretter Noder og lenker dem sammen slik du får følgende datastruktur:

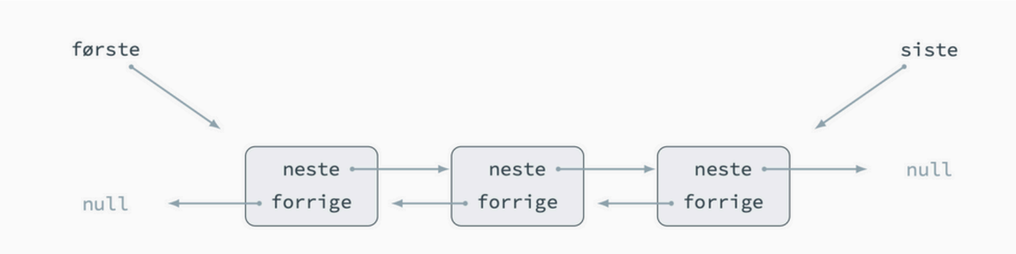


Enkeltlenket liste

- Første peker
- Siste peker
- Hver node har en peker til neste node
  - Ingen peker til noden før

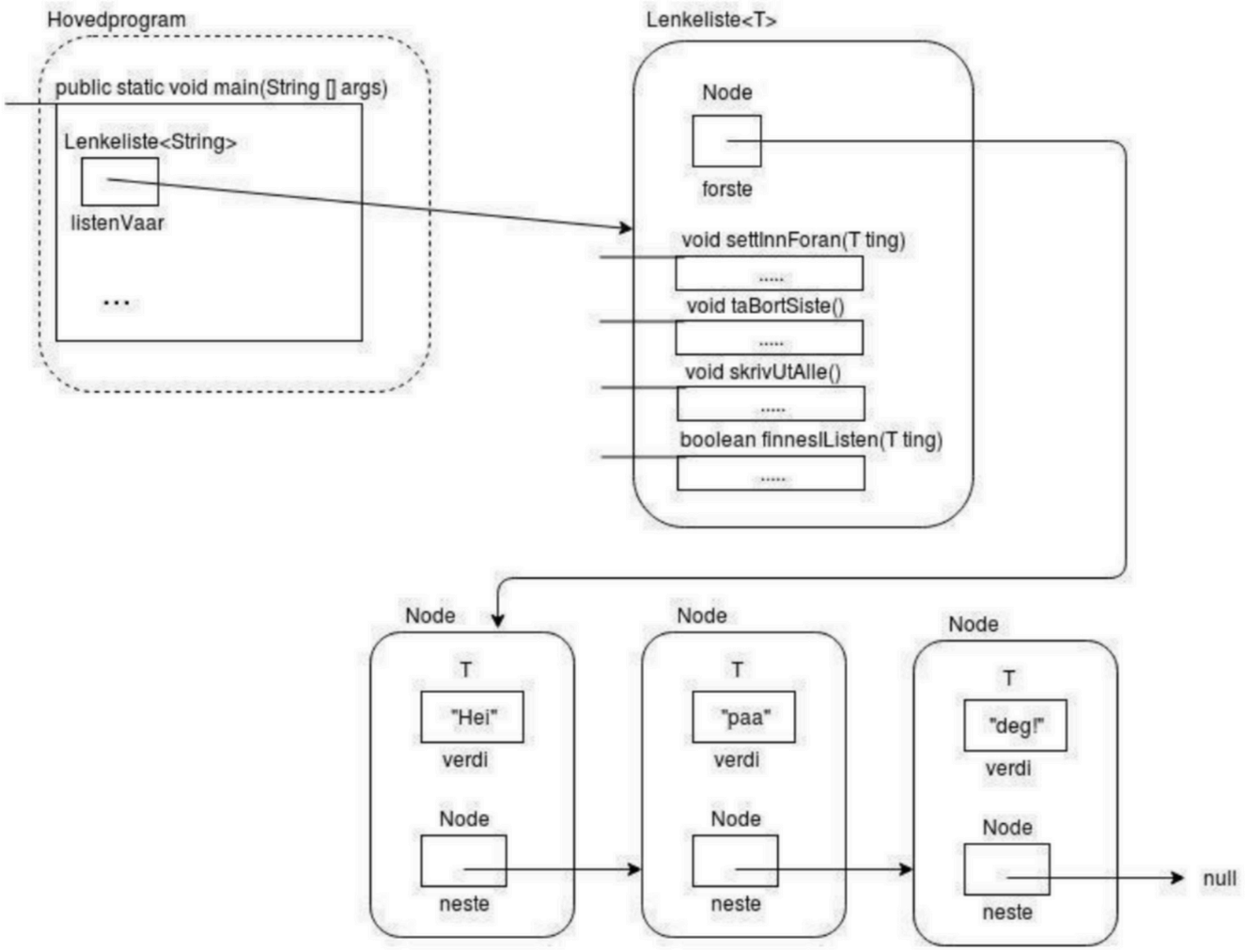
Dobbeltlenket liste

- Første peker
- Siste peker
- Hver node har en peker til neste node
  - Har peker til noden før



Lenkeliste klassen

- Lenkeliste klassen har en **indre node klasse**
  - **Referansen** til nodene er **gjennom** lenkelisten
- Lenkelisteklassen **håndterer innsetting**, **sletting** og andre **operasjoner**



Med lenkelister kan vi implementere :

- Stabel (Last In First Out, LIFO)
- Kø (First In First Out)
- Prioritetskø (elementer settes og fjernes etter prioritet, f.eks høyest tall)

- Når man skal implementere, lurt å tegne når man skal sette inn, og fjerne elementer

Comparable grensesnittet

- Når vi har lyst til kan sammenligne objekter
  - Sortere objekter
- Må implementere compareTo()
  - returnerer
    - <0 hvis obj som kalte på metoden er mindre
    - >0 hvis obj som kalte på er større
    - = 0 hvis obj er like

```java
public class Person implements Comparable<Person> {
    private int hoyde, alder;
    Person(int hoyde, int alder) {
    this.hoyde = hoyde;
    this.alder = alder;
    }
    @Override
    public int compareTo(Person annen) {
    int hoydeforskjell = this.hoyde - annen.hoyde;
    int aldersforskjell = this.alder - annen.alder;
    return hoydeforskjell == 0 ? aldersforskjell : hoydeforskjell;
    }
    }
```

Iterable grensesnittet

- Gjør at vi kan iterere/ gå gjennom listen med en for løkke
- Må implementere iterator(), og Iterable klassens metoder:hasNext() og next()

```java
import java.util.Iterator;
public class Lenkeliste<T> implements Iterable<T> {
    @Override
    public Iterator<T> iterator() {
        return new LenkelisteIterator();
    }
```

```java
class LenkelisteIterator implements Iterator<T> {
    Node denne = forste;

    @Override
    public boolean hasNext() {
        return denne != null;
    }

    @Override
    public T next() {
        T data = denne.data;
        denne = denne.neste;
        return data;
```

- **Seminaroppgaver**
- **Trix**
- **Eksamensoppgaver**

Eksamen 2024

Oppgave 3. Klassen Flyformasjon (28 poeng)

I oppgave 3 skal du skrive en klasse **Flyformasjon** med variabler og metoder. En Flyformasjon er en enkelthektet lenket liste av Flyobjekter og du skal bruke instansvariablene i disse objektene til å lage denne listen. I denne oppgaven kan du aksessere instansvariabler i Fly-klassen og dens subklasser.

3a) 1 poeng

⇨ Skriv konstruktør og instansvariabler for klassen Flyformasjon med peker til første Fly-objekt i listen.

3b) 3 poeng

⇨ Skriv metoden **leggTil** i klassen Flyformasjon som legger til et Flyobjekt først i listen. Parameter til metoden er en referanse til objektet som skal legges til.