# IN1010 - Seminar 7

- Beholdere og generiske klasser II

Laget av: Marlen Jarholt ( marleja@ifi.uio.no ) og Johanna Haarseth ( johannph@ifi.uio.no )

# Praktisk

Husk å sjekke emnesiden regelmessig

Undervisningstilbud:

- https://www.uio.no/studier/emner/matnat/ifi/IN1010/v21/undervisningstilbud/
- Jeg har konkrete spørsmål/problemer med min kode -> Labtime!
- Jeg vil ha mer liveprogrammering -> Plenumstime!
- Jeg vil jobbe med andre (og kanskje en kjapp recap av forelesning) -> Gruppetime!
- Jeg vil ha en recap av de vanskeligste konseptene fra forelesning -> Repetisjonsgruppe!

Oblig 3 har frist mandag 1. mars

# Repetisjon forrige uke

# Beholder

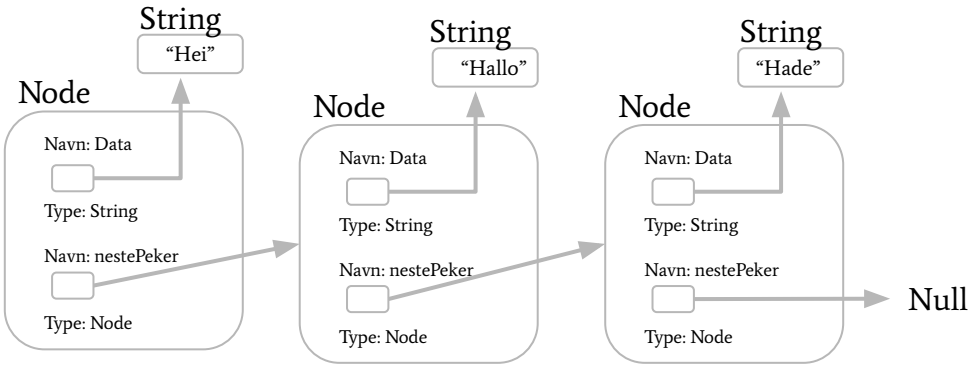
Et objekt som representerer en gruppe objekter av samme type.

Ofte metoder for å legge til, hente ut og finne størrelse.

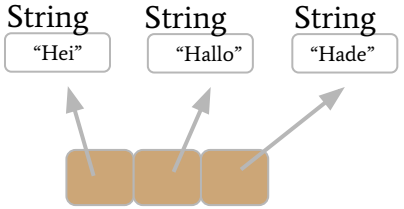
Eksempler på beholdere: array(har ingen metoder), ArrayList, HashMap, Lenkeliste

```
34
35      NavnKlasseEllerInterface[] navnVariabel = new NavnKlasseEllerInterface[lengdeArray];
36      ArrayList<NavnKlasseEllerInterface> navnVariabel = new ArrayList();
37      HashMap<NavnKlasseEllerInterfaceNokkel, NavnKlasseEllerInterfaceVerdi> navnVariabel = new HashMap();
38
```

# Lenkeliste




Lenkeliste




ArrayList eller Array f.eks.

# Lenkeliste

```java
//String versjon av Node:
class Node{
    private String data;
    private Node nestePeker;

    public Node(String data){
        this.data = data;
    }
    public void settNestePeker(Node nestePeker){
        this.nestePeker = nestePeker;
    }
    public Node hentNestePeker(){
        return nestePeker;
    }
    public String hentData(){
        return data;
    }
}
```

```java
class Lenkeliste{

    // Main for Node med data type String:
    public static void main(String[] args) {
        Node node1 = new Node("Hei");
        Node node2 = new Node("Hallo");
        Node node3 = new Node("Hade");
        node1.settNestePeker(node2);
        node2.settNestePeker(node3);
    }
```

# Generisk klasse

Hva betyr generisk klasse? Send meg direktemelding i chatten

```java
//String versjon av Node:
class Node{
    private String data;
    private Node nestePeker;

    public Node(String data){
        this.data = data;
    }
    public void settNestePeker(Node nestePeker){
        this.nestePeker = nestePeker;
    }
    public Node hentNestePeker(){
        return nestePeker;
    }
    public String hentData(){
        return data;
    }
}
```

```java
//Generisk klasse Node:
class Node<T>{
    private T data;
    private Node nestePeker;

    public Node(T data){
        this.data = data;
    }
    public void settNestePeker(Node nestePeker){
        this.nestePeker = nestePeker;
    }
    public Node hentNestePeker(){
        return nestePeker;
    }
    public T hentData(){
        return data;
    }
}
```

# Generisk klasse Lenkeliste

Jeg hadde main i lenkeliste-klassen i forrige uke og i forrige eksempel.

Men vi kan flytte main ut i en egen klasse og gjøre Lenkeliste til en generisk klasse.

Vi kan nå også gjøre Node til en indre klasse i Lenkeliste, da slipper vi å sende inn Noder men kun verdien.

```
class Lenkeliste<T>{

    private Node start;

    private class Node{
        T data;
        Node neste;
    }

    public void leggTil(T nyData){
        //legget til ny data
    }

    public void slettData(T slettData){
        //sletter data
    }

    public T hentData(T data){
        //henter node med gitt data
    }
}
```

# Repetisjon denne uken

# Send Marlen en direktemelding i chatten

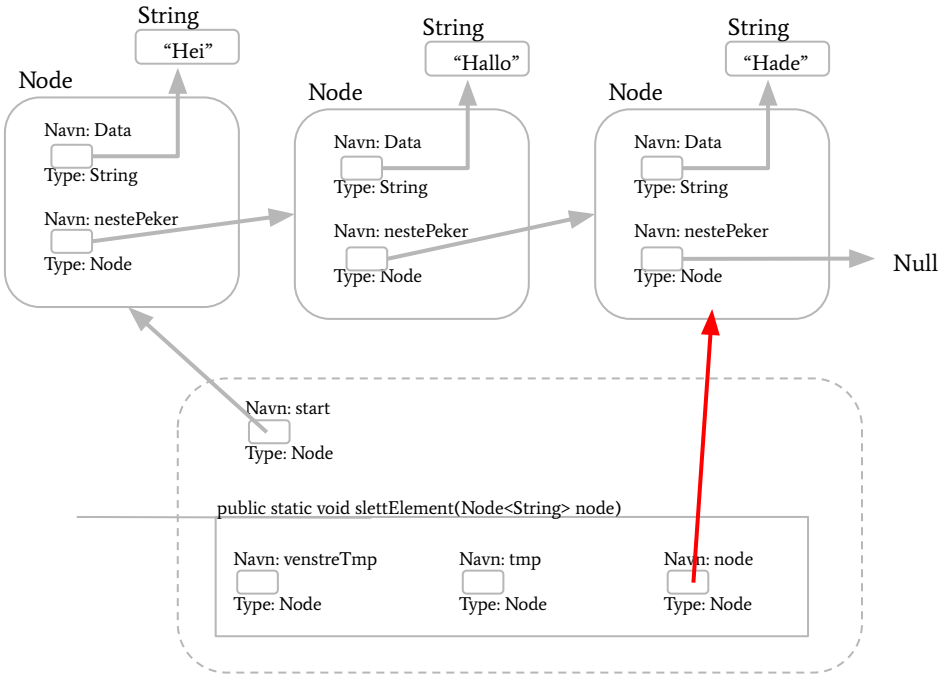
Vil du jobbe sammen med noen andre ? (ja /nei)

Hvis du har noen ønsker på hvem du vil jobbe med, så send det i samme melding

Svar gjerne også om svaret skulle være nei

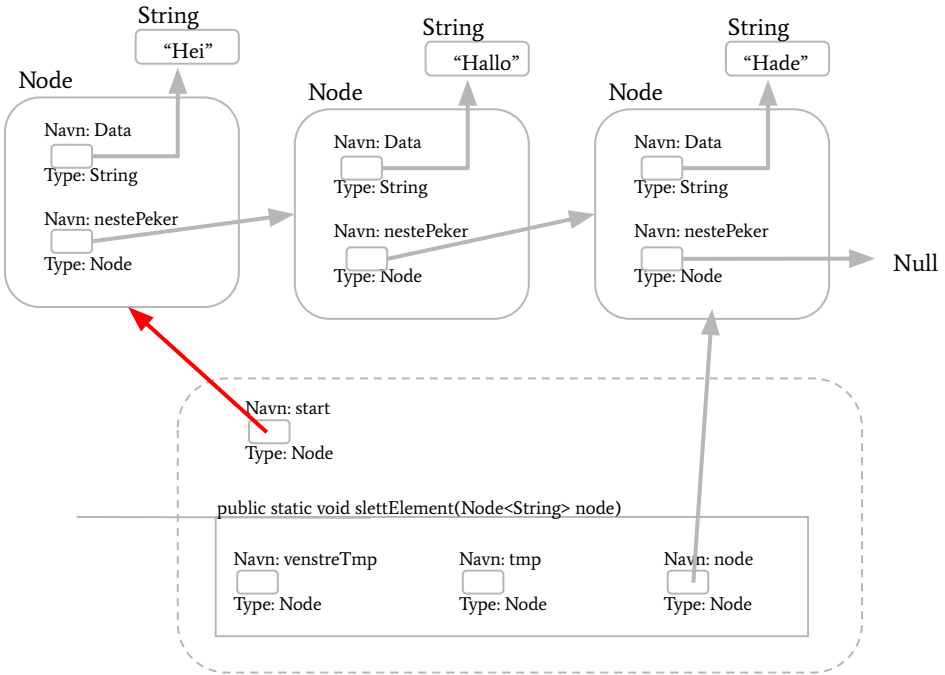
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



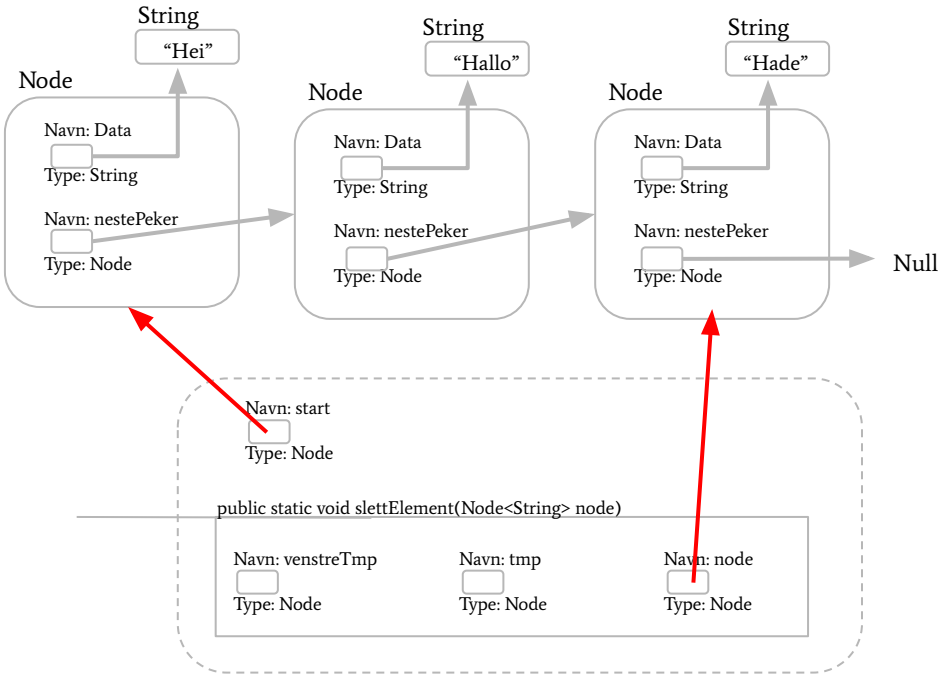
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



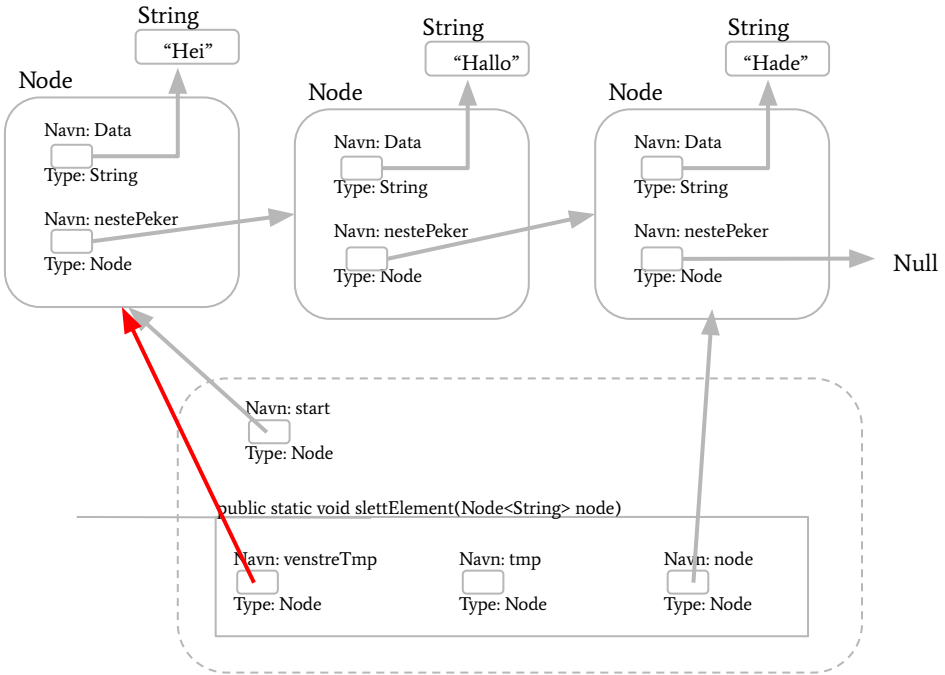
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



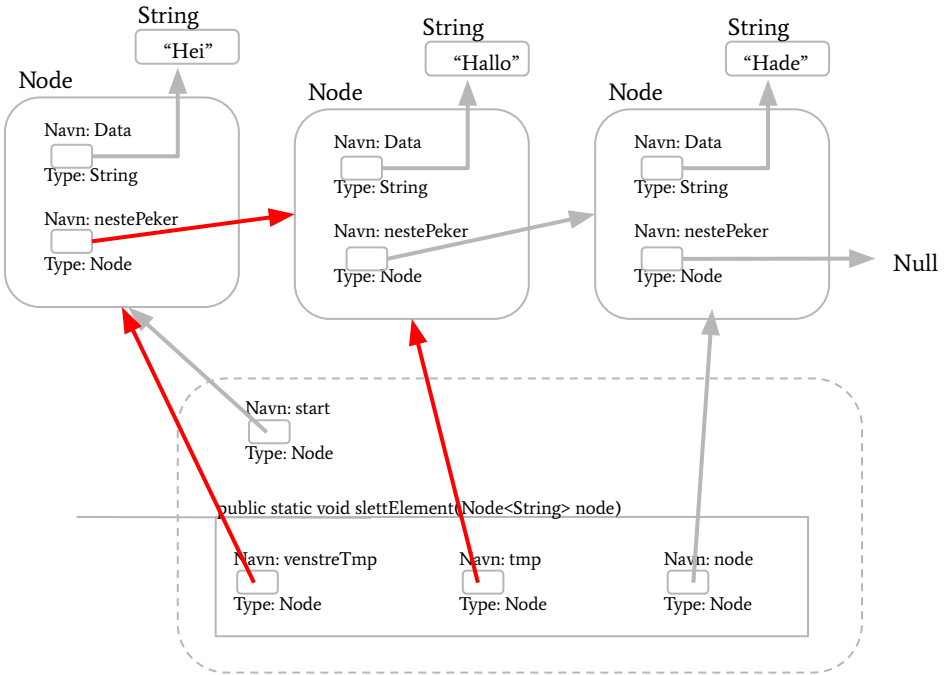
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



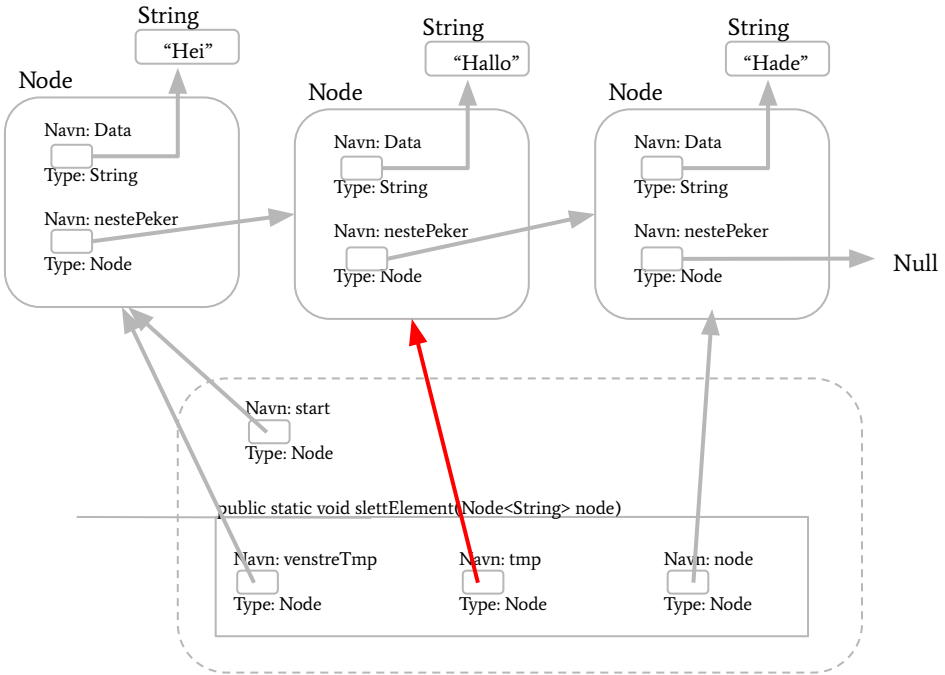
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



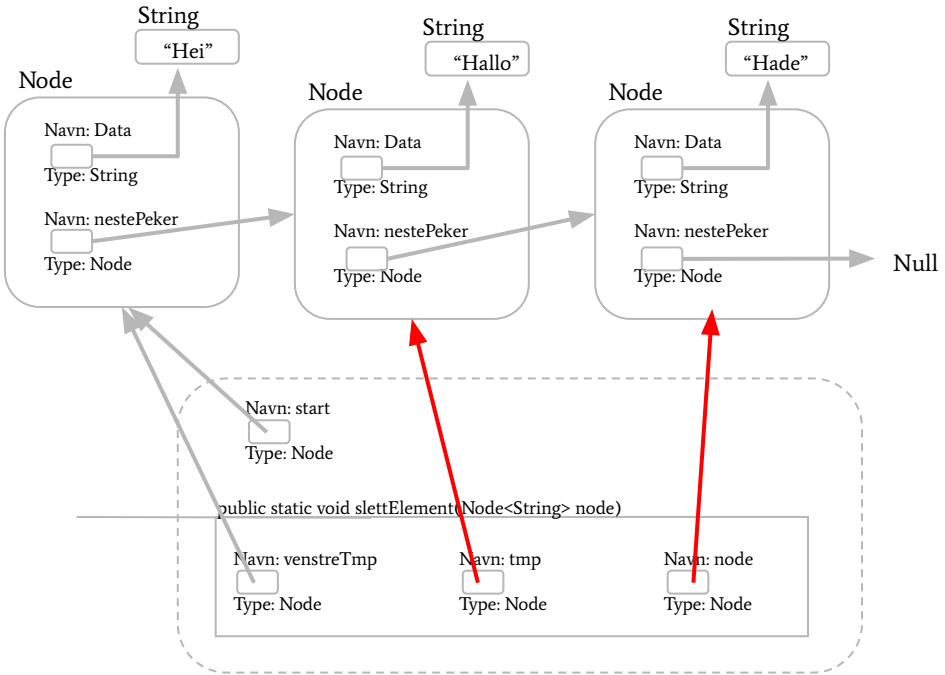
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



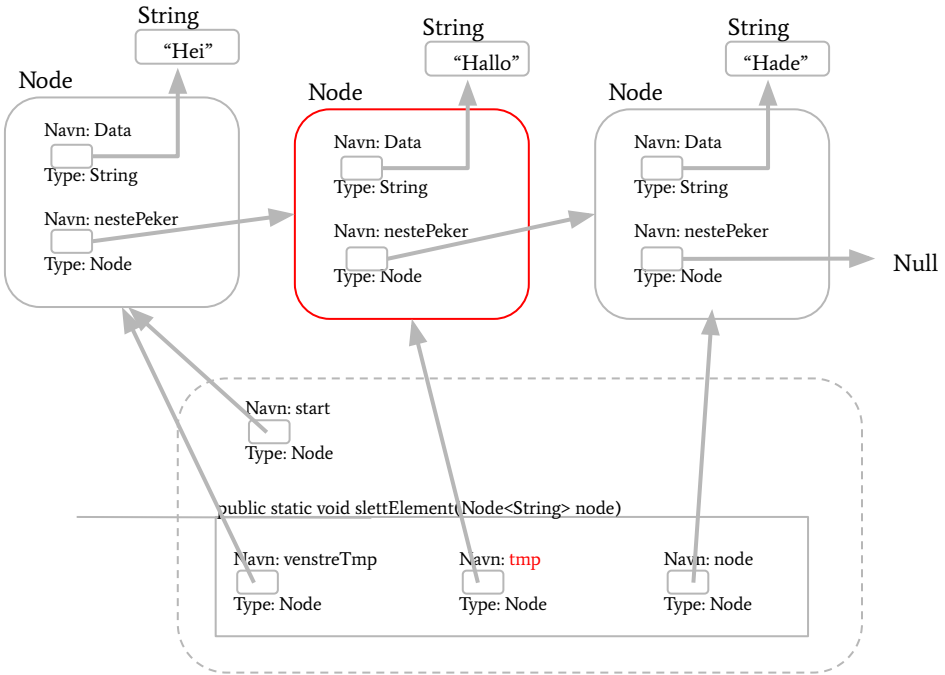
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



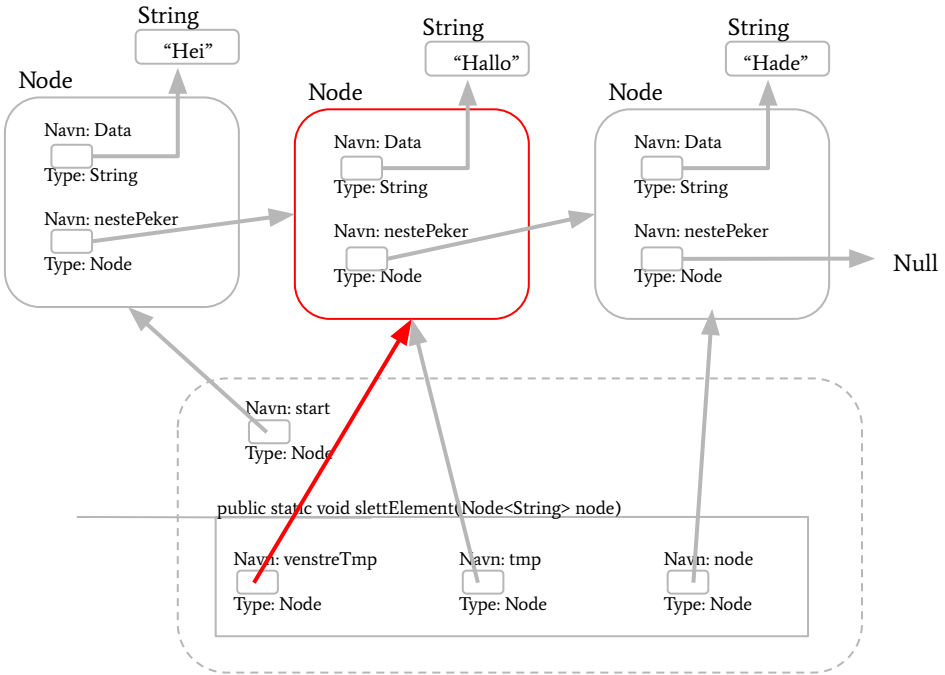
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



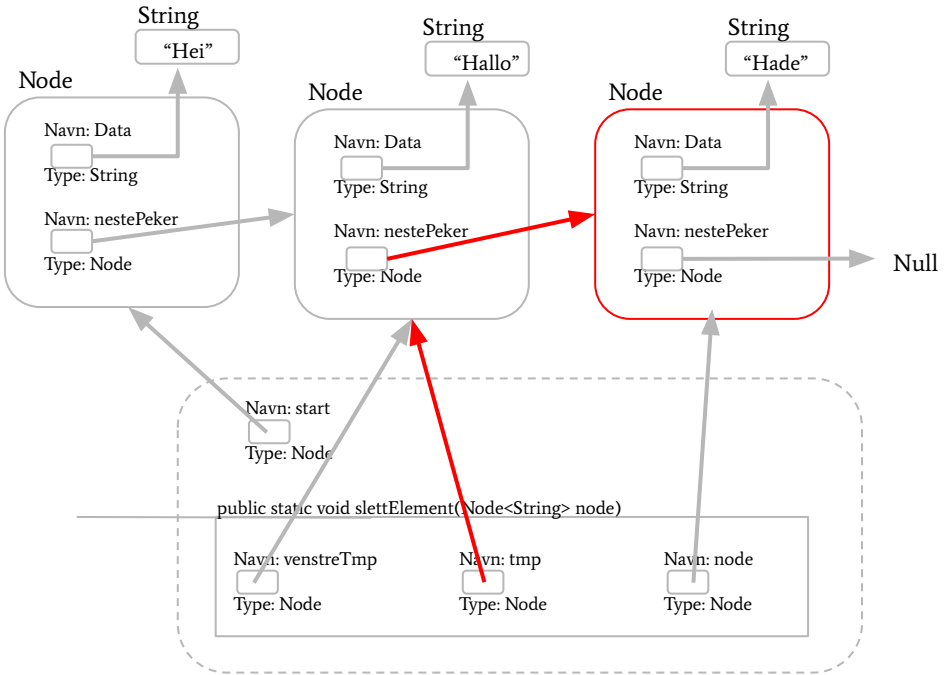
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



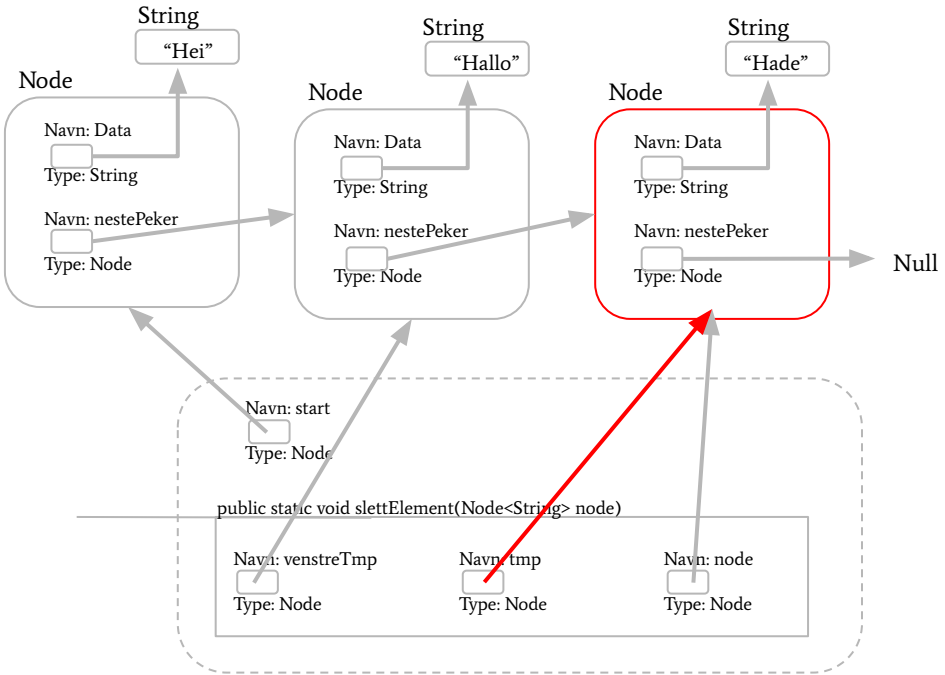
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



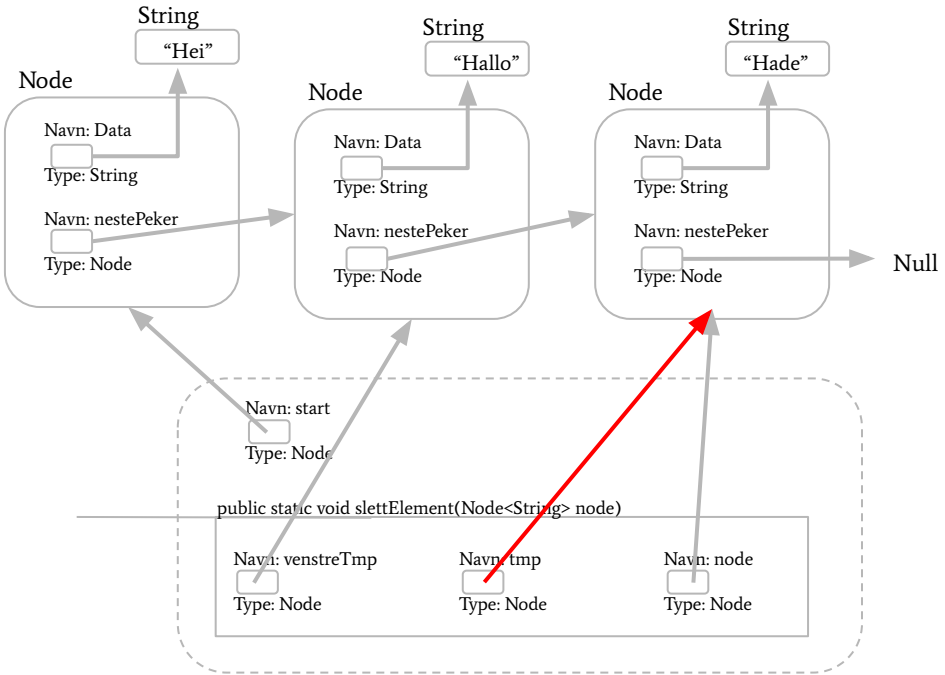
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



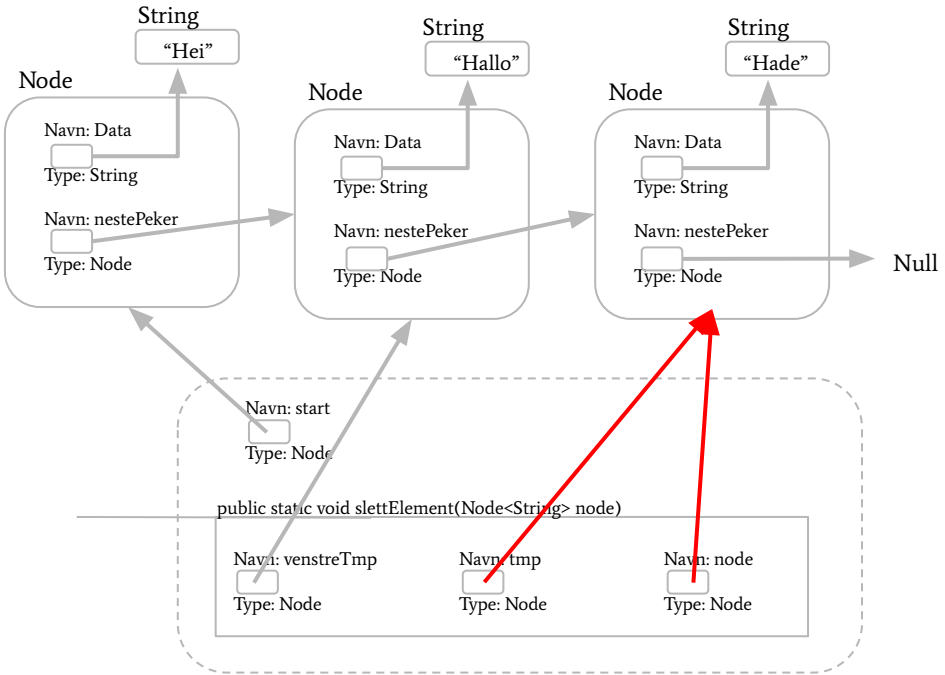
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



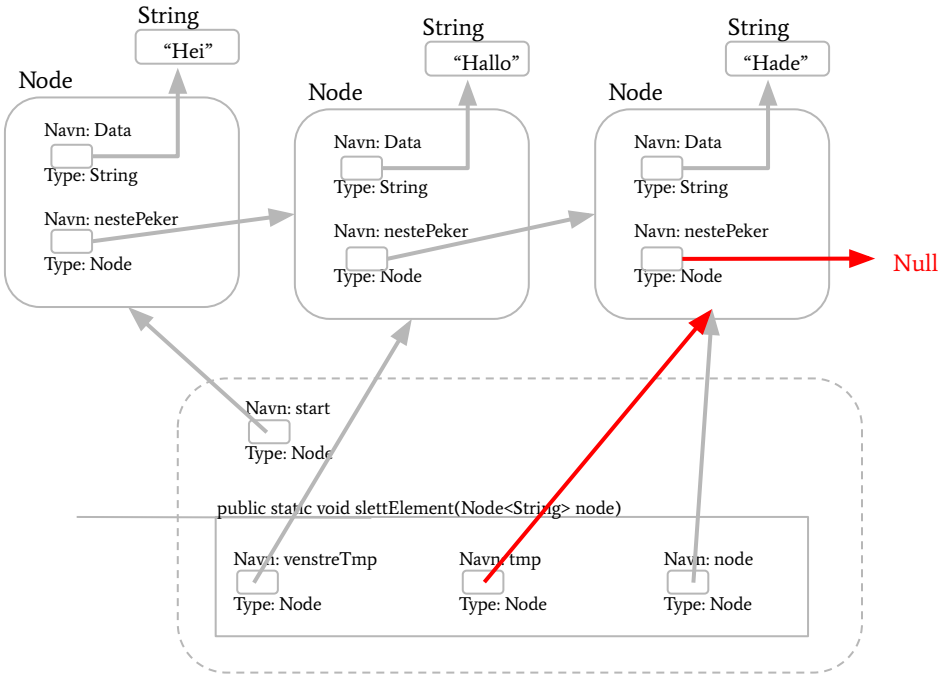
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



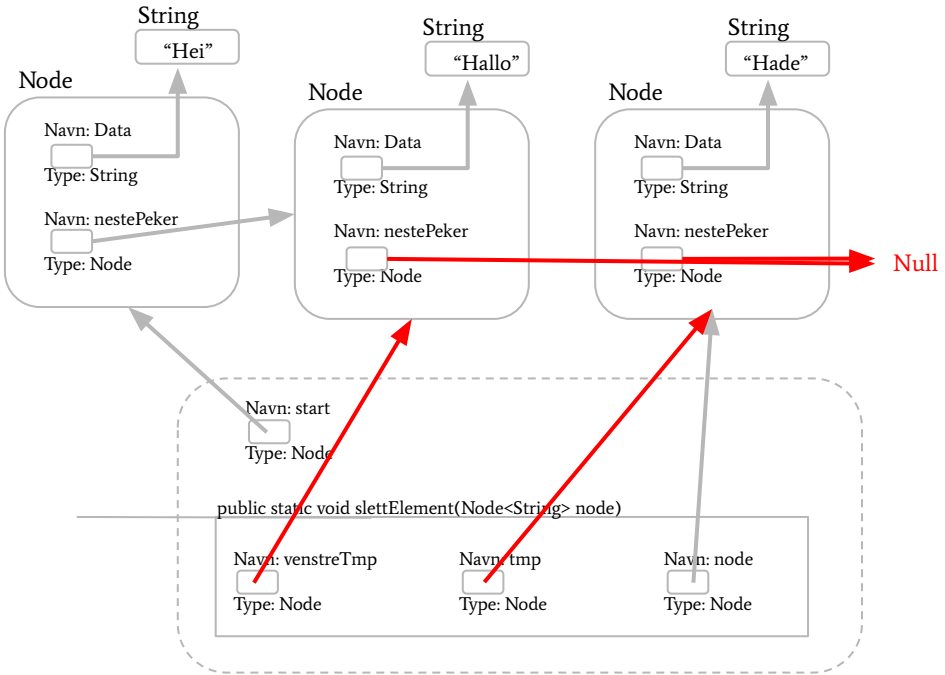
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



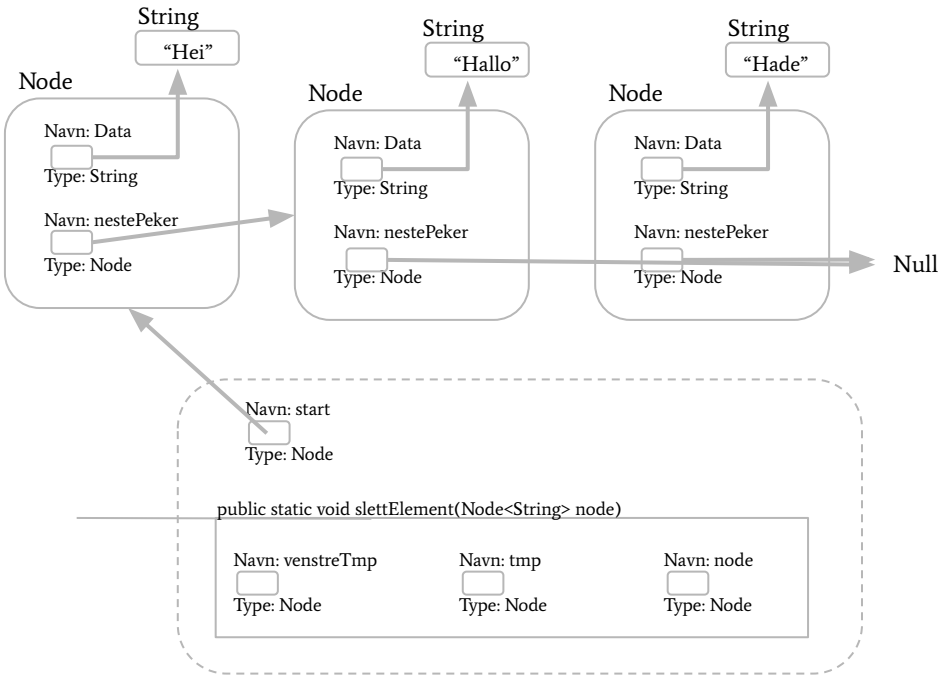
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



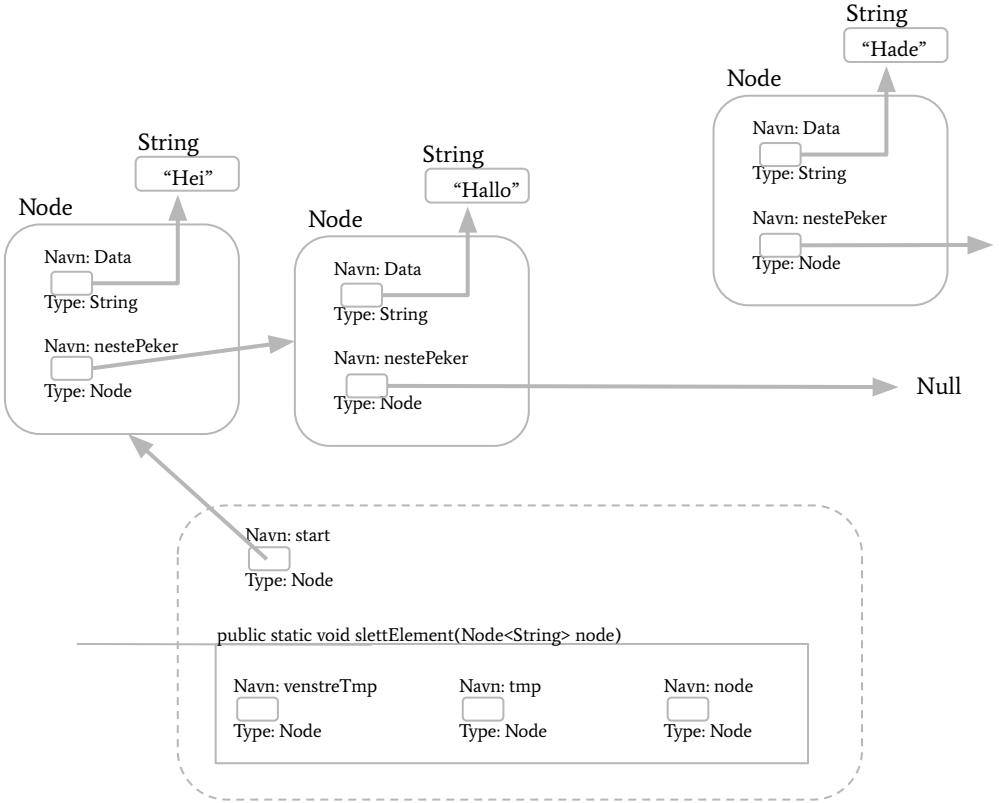
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



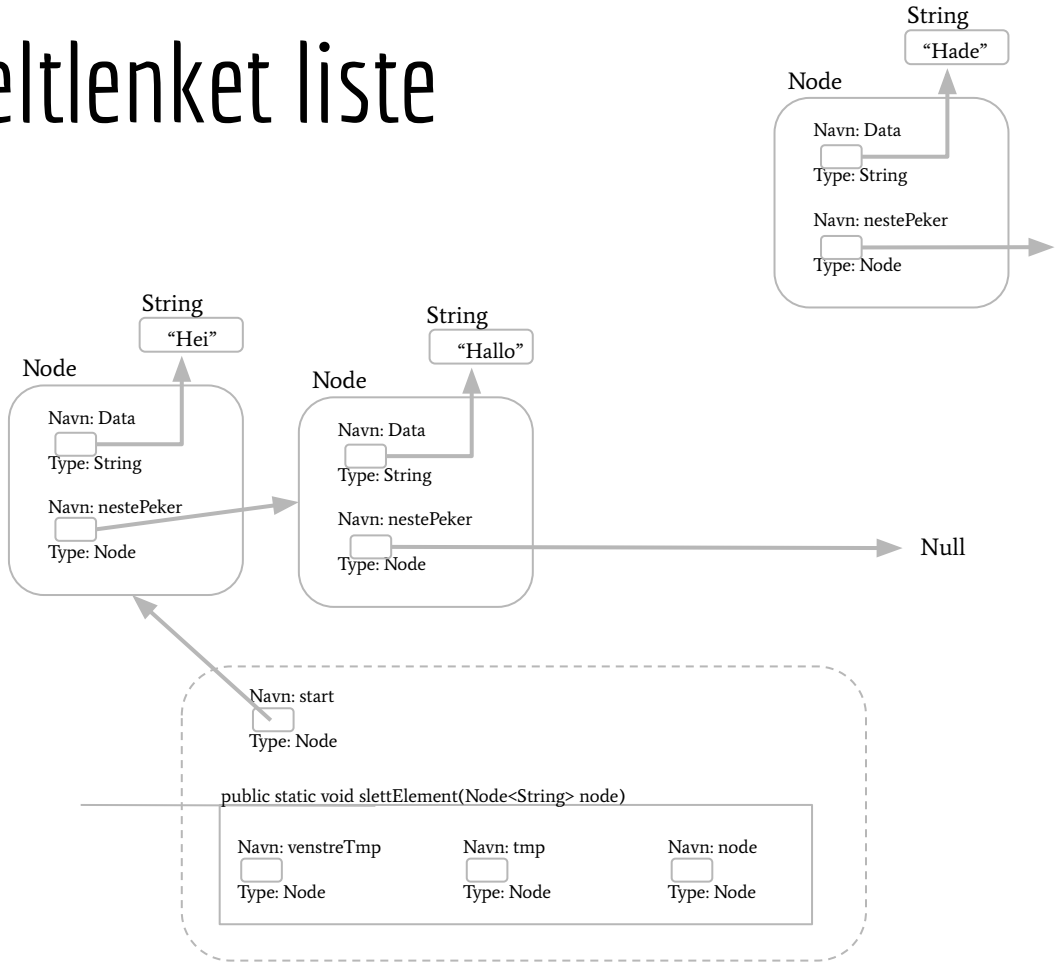
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



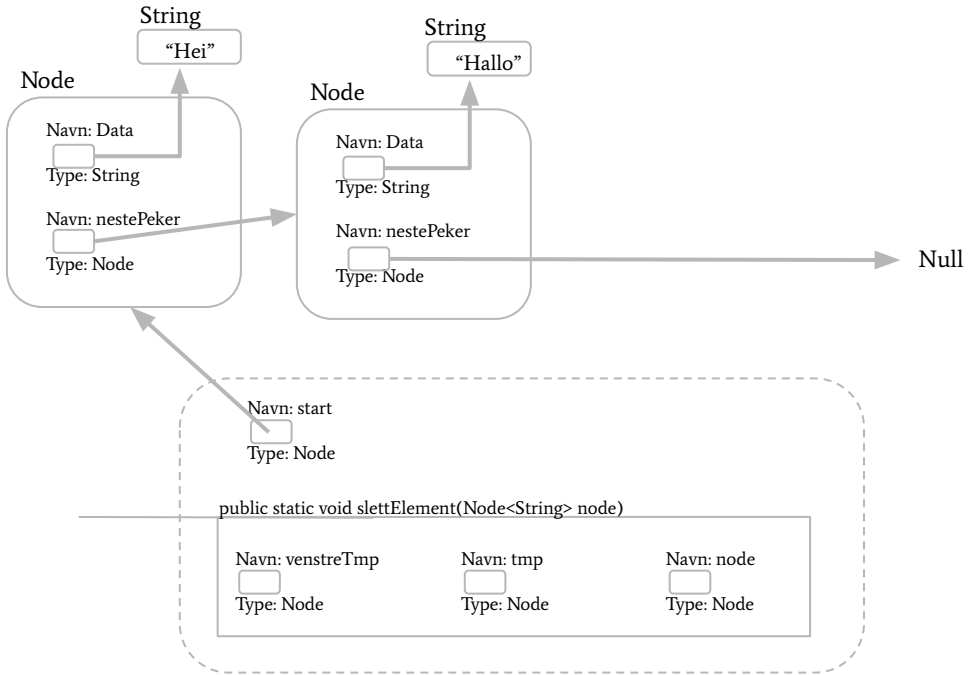
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



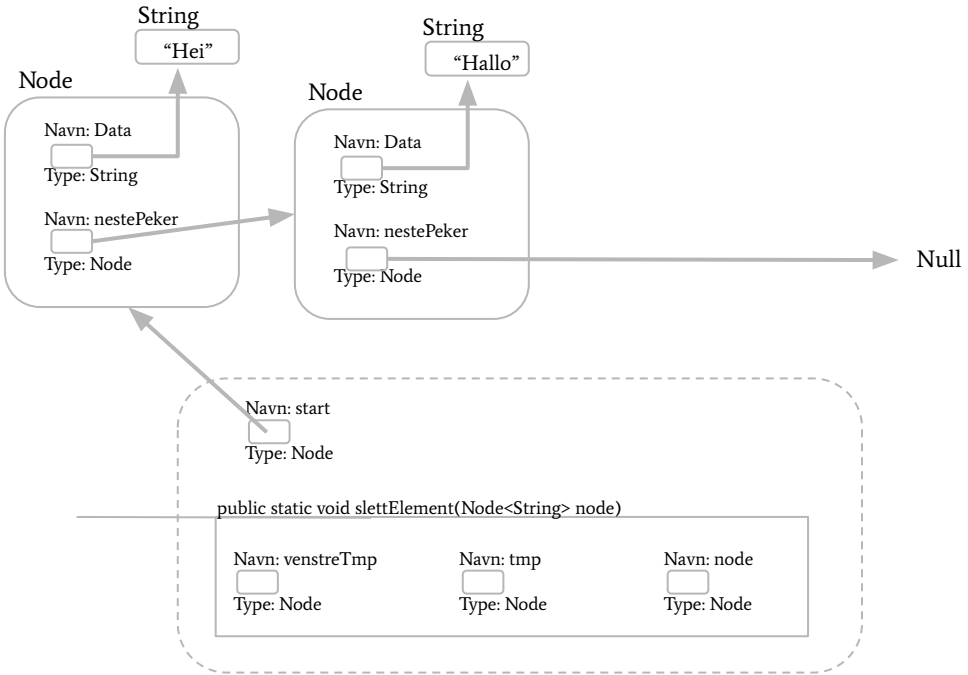
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```

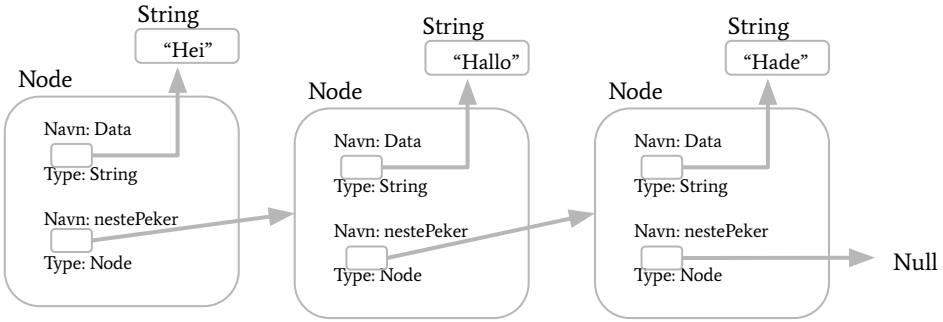


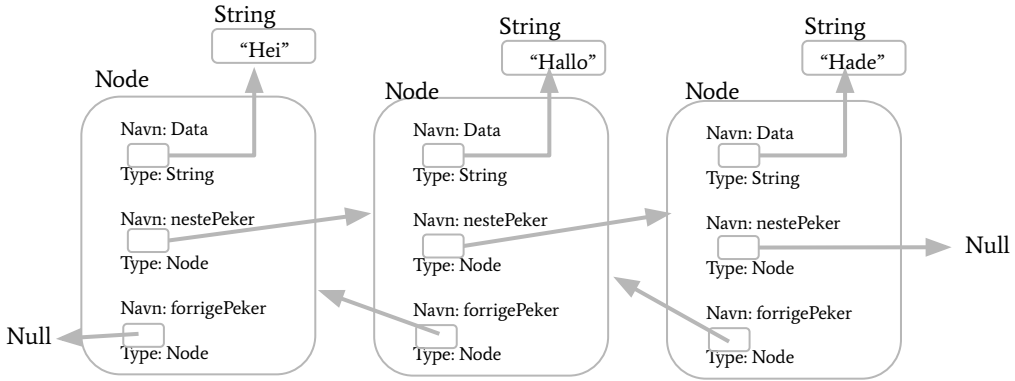
# Slette element fra enkeltlenket liste

```java
public static void slettElement(Node<String> node) {
    if(start == null){
        return;
    }
    if(start.equals(node)){
        start = start.hentNeste();
        return;
    }
    Node<String> venstreTmp = start;
    Node<String> tmp = start.hentNeste();
    while(tmp != null){
        if(tmp.equals(node)){
            venstreTmp.settNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



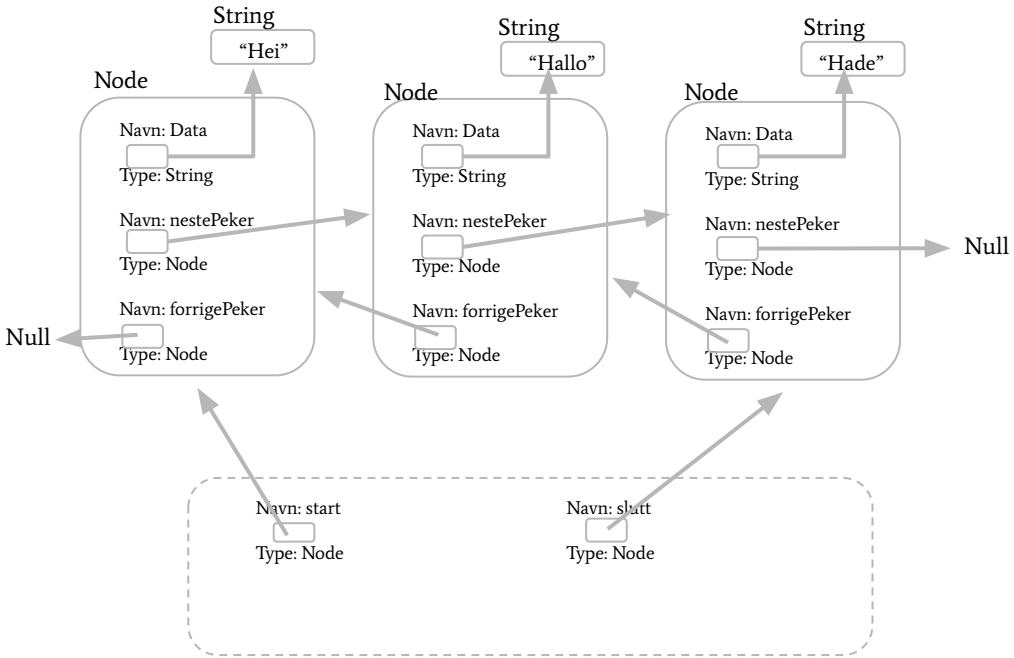
# Enkeltlenket vs dobbeltlenket lenkeliste



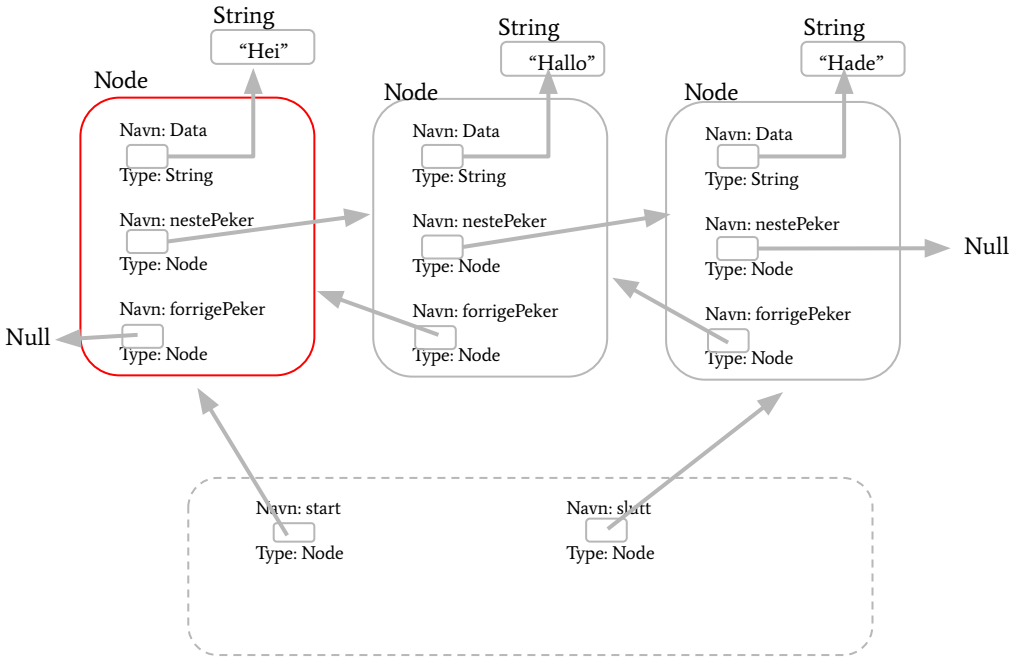
# Slette element fra dobbeltlenket liste

Se på alle ulike cases:



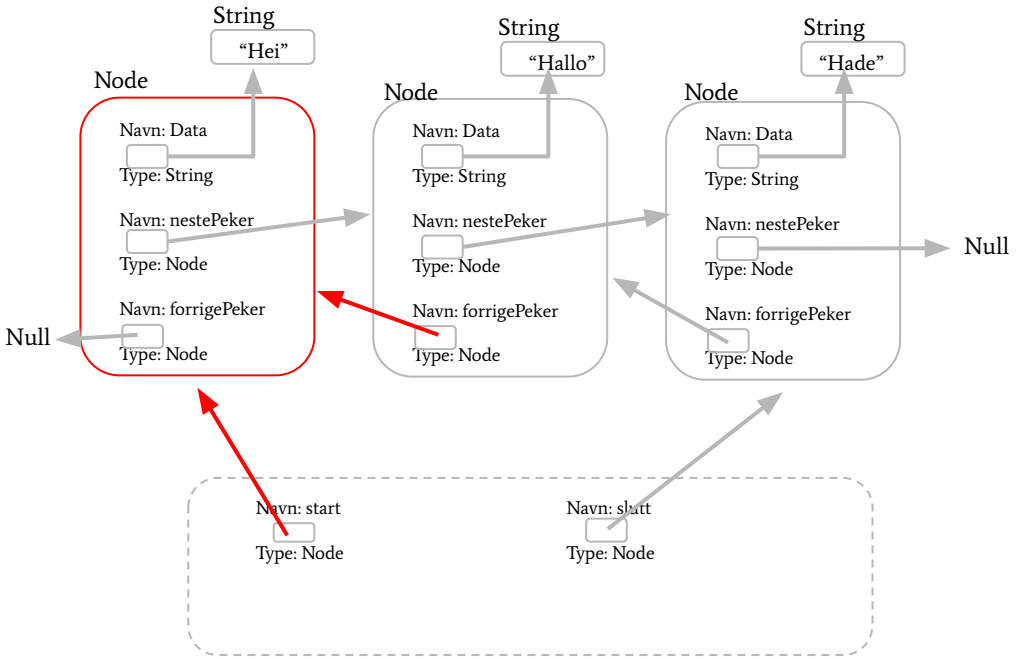
# Slette element fra dobbeltlenket liste

Slette første element



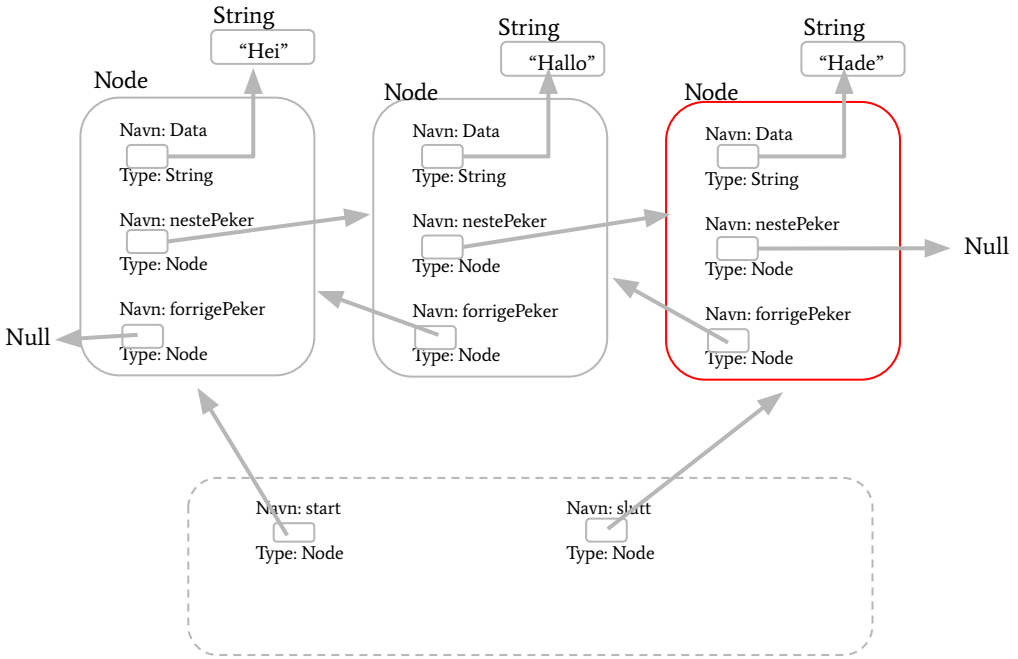
# Slette element fra dobbeltlenket liste

Slette første element



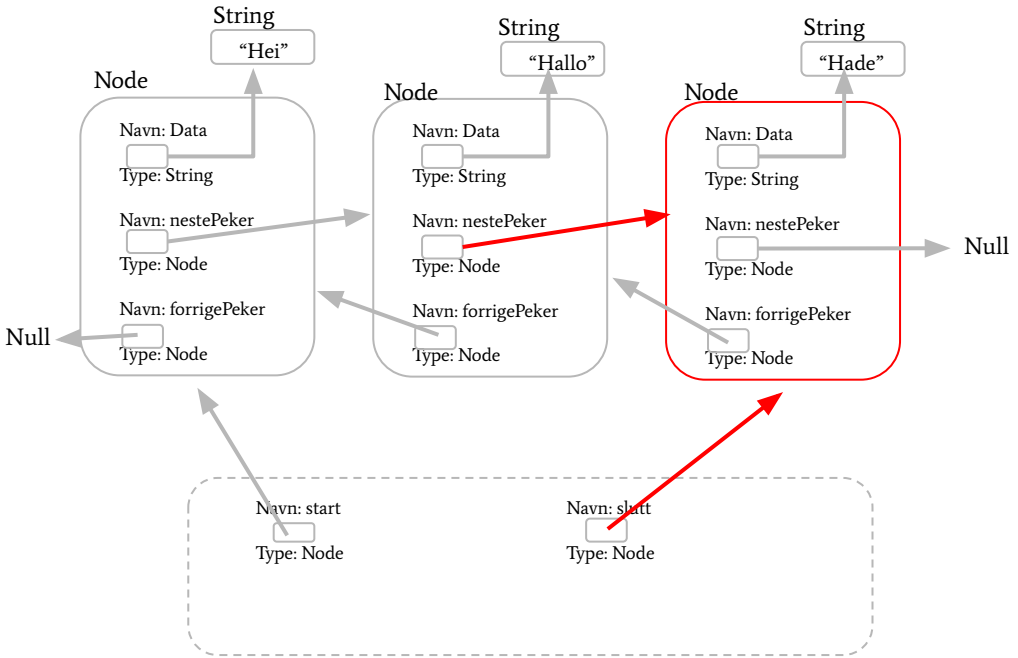
# Slette element fra dobbeltlenket liste

Slette siste element



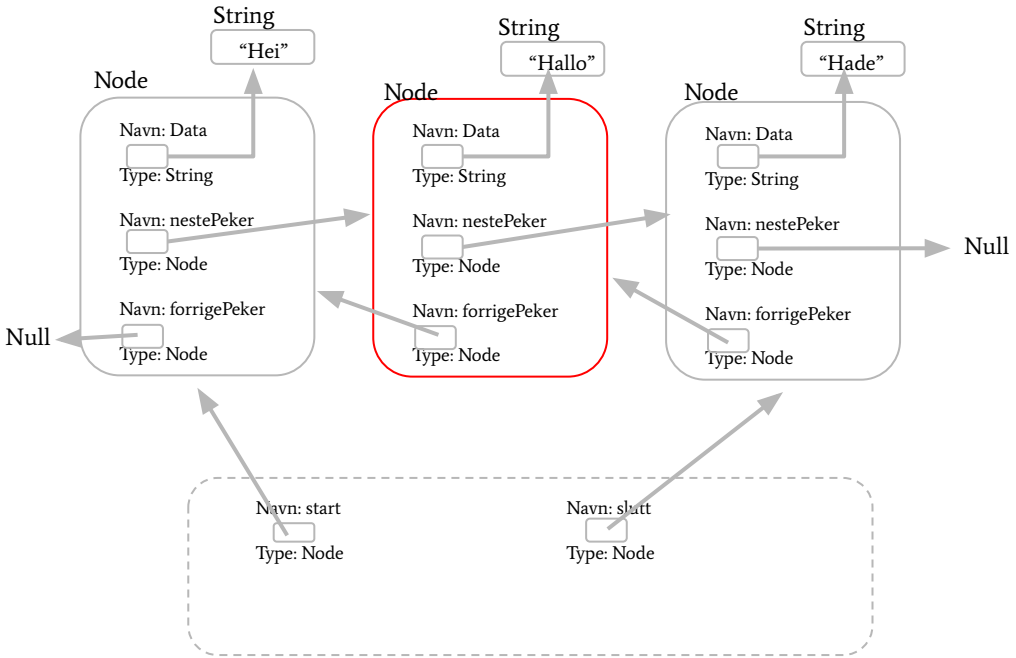
# Slette element fra dobbeltlenket liste

Slette siste element



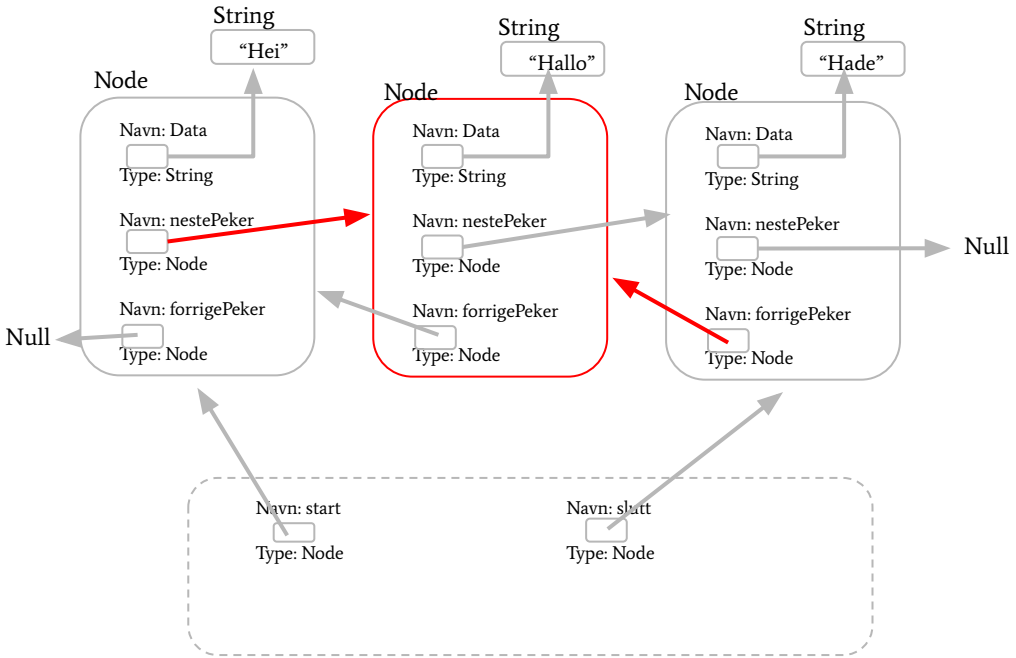
# Slette element fra dobbeltlenket liste

Slette element midt i listen



# Slette element fra dobbeltlenket liste

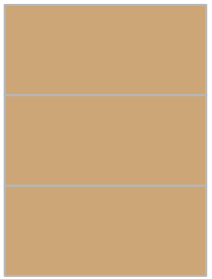
Slette element midt i listen

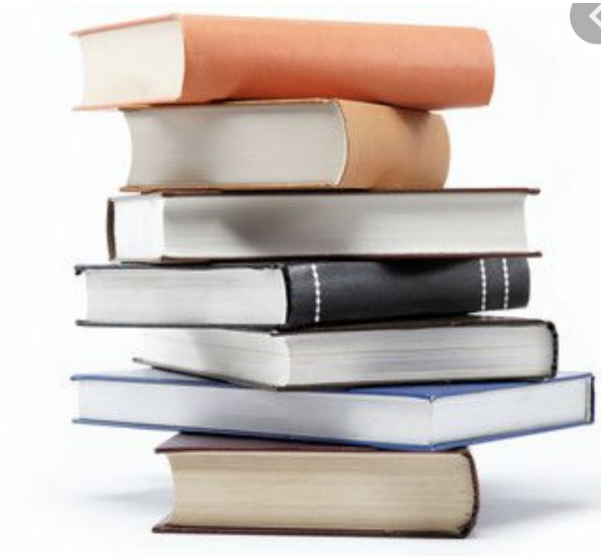


# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:

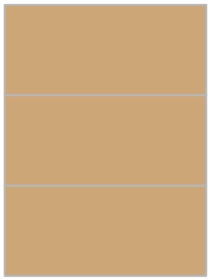


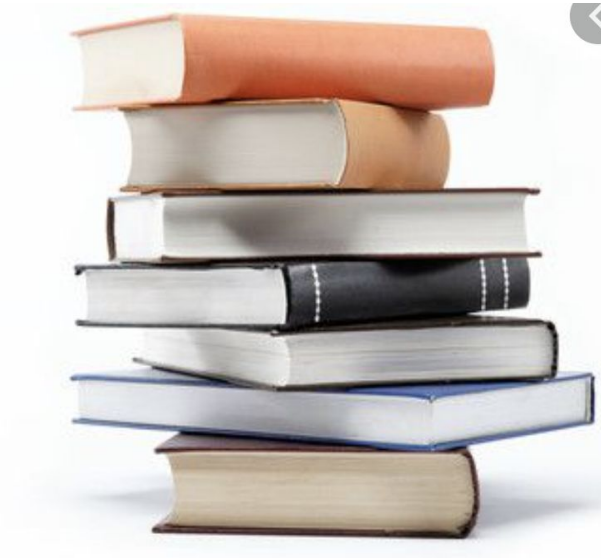


# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:

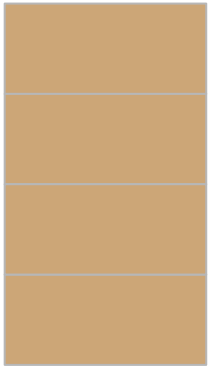


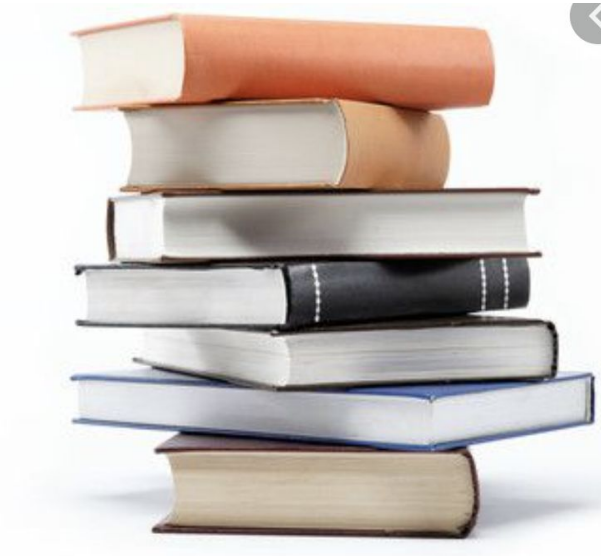


# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:



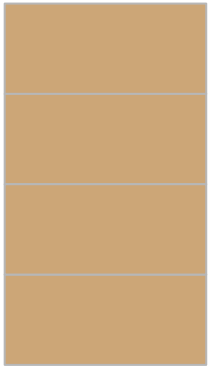


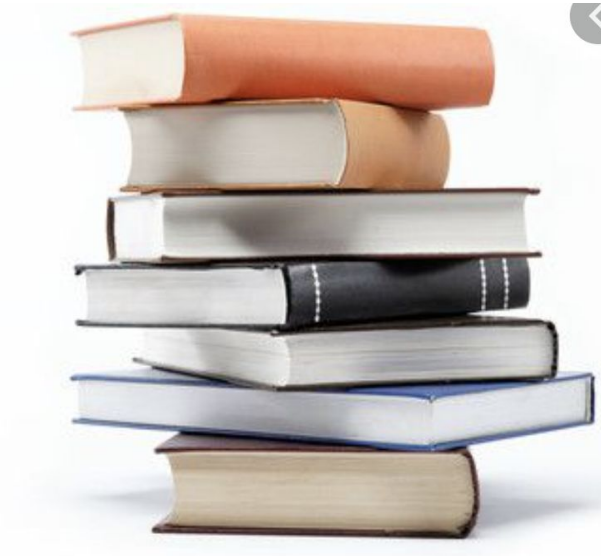
# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:

pop:



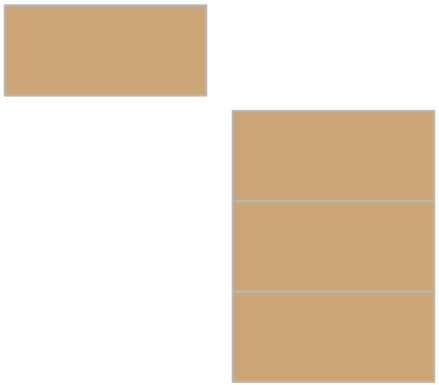


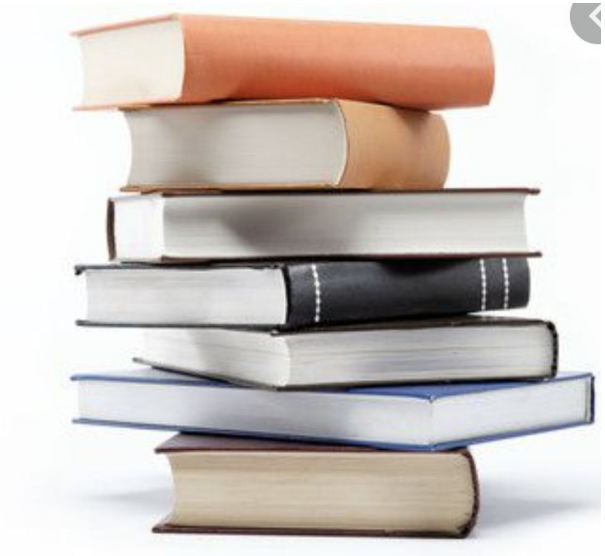
# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:

pop:



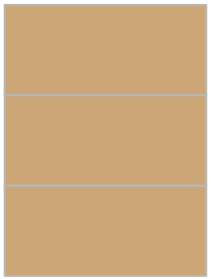


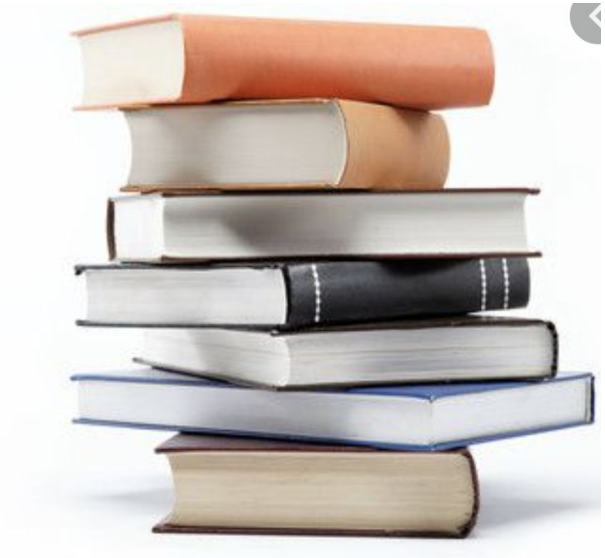
# Stabel/Stack: LIFO (Last In First Out)

Det siste vi legger inn er det første vi tar ut:

push:

pop:





# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add: 

# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add:

# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add:

# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add:

remove:

# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add:

remove:

# Kø/queue: FIFO (First In First Out)

Det første vi legger inn er det første vi tar ut:

add:

remove:

# Interfacet Comparable og Metoden CompareTo

Hvis en klasse implements Comparable betyr det at vi kan gjøre sånne operasjoner:

objekt1 < objekt2 objekt1 > objekt2 objekt1 == objekt2

f.eks. klassen Integer og String implementerer comparable:

1 < 2 1 > 2 1==2

“a” < “b” “a” > “b” “a” == “b”

# Interfacet Comparable og Metoden CompareTo

Vi har et interface Comparable, det kan vi implementere i node (legge til compareTo() metode), men det eneste vi ev. kan sammenligne i vår nodeklasse er dataen.

```
public interface Comparable<T> {
    public int compareTo(T otherObj);
}
```

```
25  class Node<T> implements Comparable<Node<T>>{
26  
27      public int compareTo(Node<T> node){
28          //Returnere -1, 0 eller 1
29      }
```

# Interfacet Comparable og Metoden CompareTo

Vi har et interface Comparable, det kan vi implementere i node (legge til compareTo() metode), men det eneste vi ev. kan sammenligne i vår nodeklasse er dataen.

Så isteden er det bedre å si at det er T som extends Comparable<T>. Da må vi legge inn data som kan sammenlignes: f.eks. String, tall osv. og vi trenger ikke skrive comparteTo() metode i Node-klassen.

```
public interface Comparable<T> {
    public int compareTo(T otherObj);
}
```

```
25  class Node<T extends Comparable<T>>{
26      private T data;
```

# Interfacet Comparable og Metoden CompareTo

En annen mulighet i Lenkeliste er å bare si at T i Lenkeliste extends Comparable<T> og bruke en indre klasse Node.

```
1  class Lenkeliste<T extends Comparable<T>>{
2      private static Node start;
3
4      class Node{
```

# Interface Iterable og interfacet Iterator

Disse finnes i java biblioteket!! Vi skal ikke skrive dem!

```
public interface Iterator<T>{
  boolean hasNext();
  T next();
}
public interface Iterable{
  Iterator iterator();
}
```

Hvis vi ønsker å lage en beholder der man f.eks. kan bruke en for-each-loop må beholderen vår extend interface Iterable altså implementere metoden iterator(). Metoden skal returnere et objekt av en klasse som extends interfacet Iterator. Den klassen må vi også skrive!

# Ris, ros, forslag ?

https://nettskjema.no/a/180345

# Breakoutrooms

1. Slå på kamera og ha en presentasjonsrunde
2. Diskuter:
    a. Hva er forskjellen på interface og superklasse?
    b. Hva er fordelene på hashmap(ordbok)? Når burde man bruke det?
3. Jobb sammen med ukesoppgavene, de ligger på emnesiden -> grupper
    a. Enten ved at én deler skjerm med codecollab.io/
    b. OBS: codecollab er gratistjeneseter som UiO ikke har avtale med, sannsynligvis vil de samle data om dere. Dere kan fint løse oppgavene uten å bruke disse tjenestene!
4. Bruk “ask for help”-knappen for å få hjelp 😊
5. Vi møtes her igjen for å gå gjennom oppgavene til slutt (dere bestemmer hvilke)

# Jobbe med oppgaver