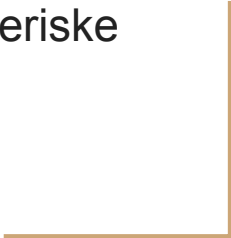


# IN1010 - Seminar 7

- Beholdere og generiske klasser II
- 



# Oppmøteregistrering:

<https://nettskjema.no/a/188717>



# Ris, ros, forslag ?

<https://nettskjema.no/a/181775>

# 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

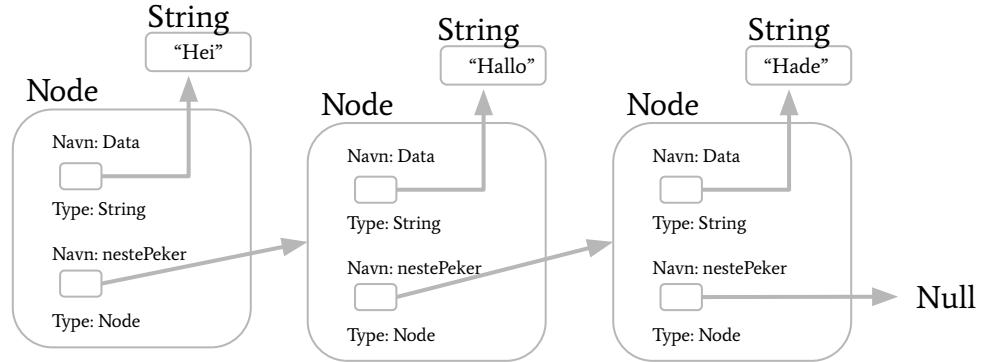
Et objekt som representerer en gruppe objekter av samme type.

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

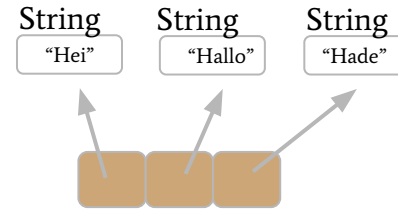
Eksempler på beholdere: array(har ingen metoder), ArrayList, HashMap, Lenkeliste

```
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

```
2 //String versjon av Node:-
3 class Node{-
4   · private String data;-
5   · private Node nestePeker;-
6   -
7   · public Node(String data){-
8   ·   · · this.data = data;-
9   · }-
10  · public void settNestePeker(Node nestePeker){-
11  ·   · · this.nestePeker = nestePeker;-
12  · }-
13  · public Node hentNestePeker(){-
14  ·   · · return nestePeker;-
15  · }-
16  · public String hentData(){-
17  ·   · · return data;-
18  · }-
19  }-
```

```
1 class Lenkeliste{-
2   -
3   · · // Main for Node med data type String:-
4   · public static void main(String[] args) {-
5   ·   · · Node node1 = new Node("Hei");-
6   ·   · · Node node2 = new Node("Hallo");-
7   ·   · · Node node3 = new Node("Hade");-
8   ·   · · node1.settNestePeker(node2);-
9   ·   · · node2.settNestePeker(node3);-
10  · }-
```



# 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.

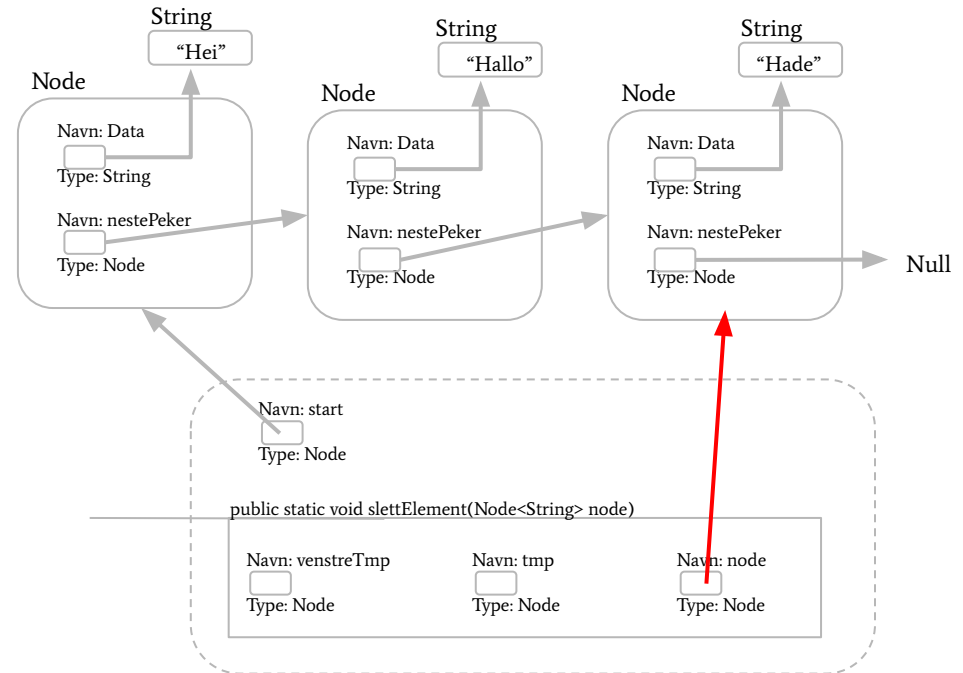
Vi kan nå også gjøre Node til en indre klasse i Lenkeliste, da slipper vi å sende inn Noder men kun verdien.

```
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

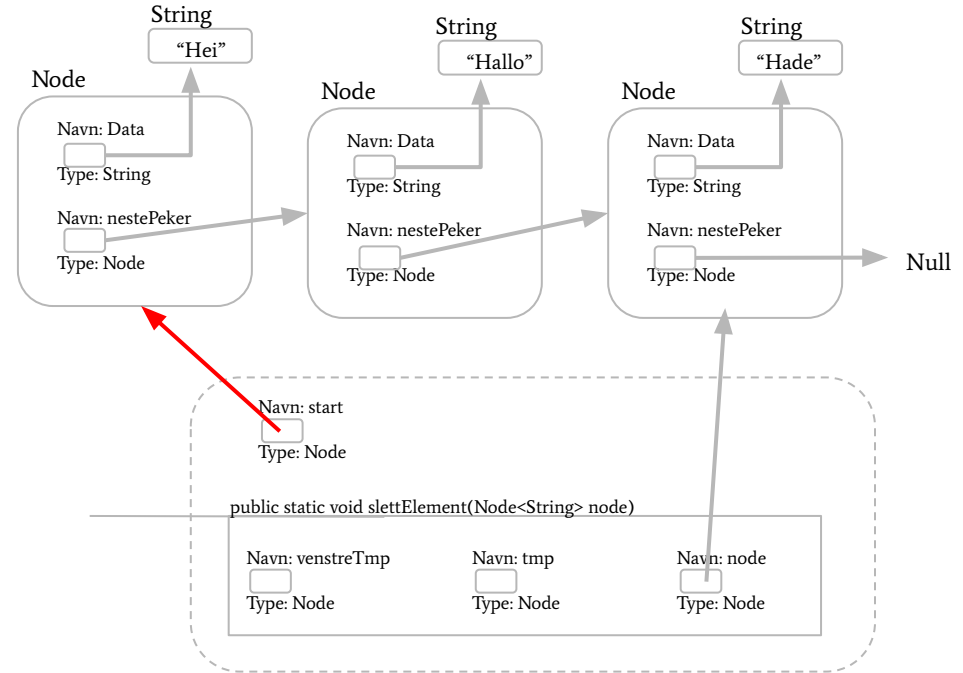
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



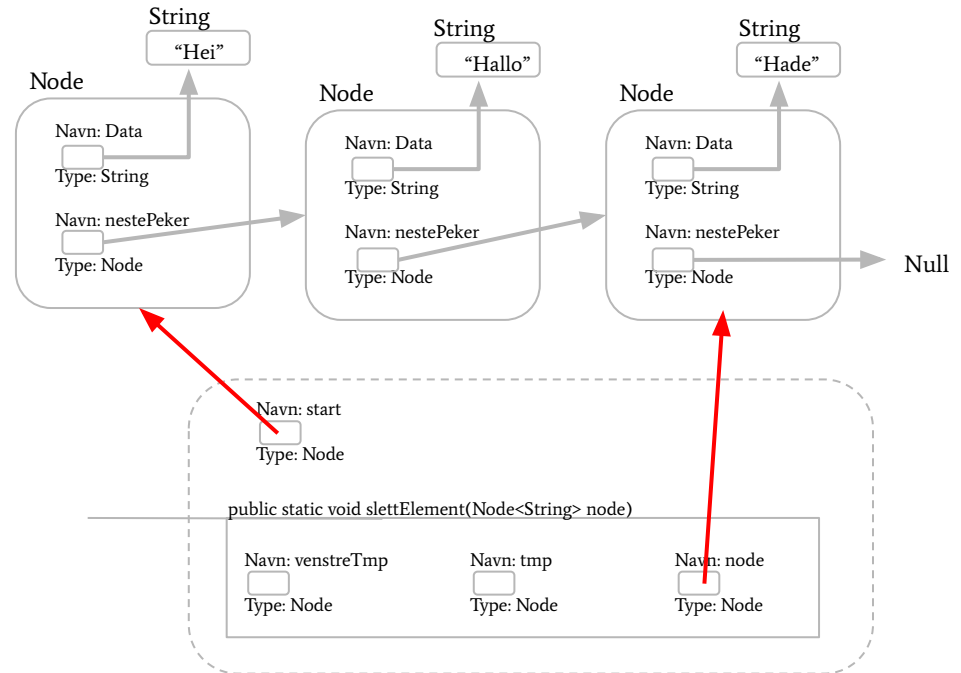
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



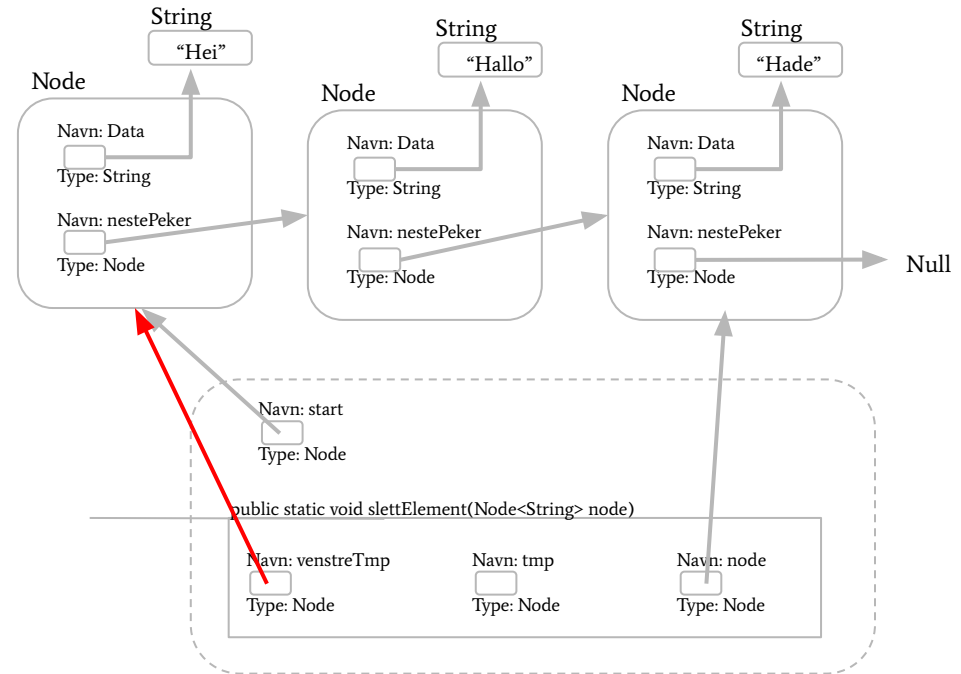
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



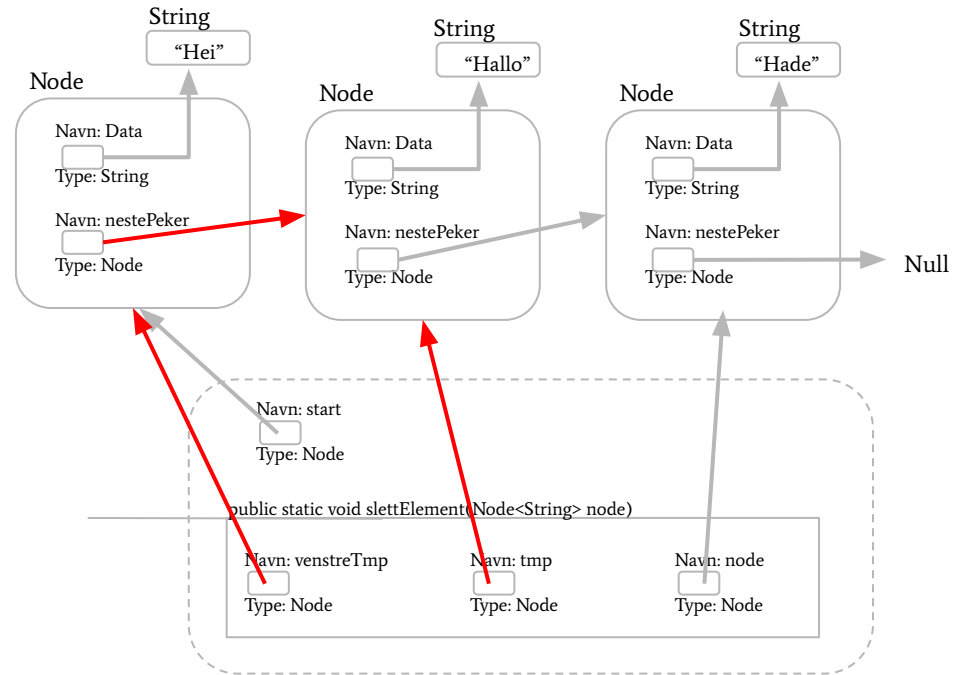
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



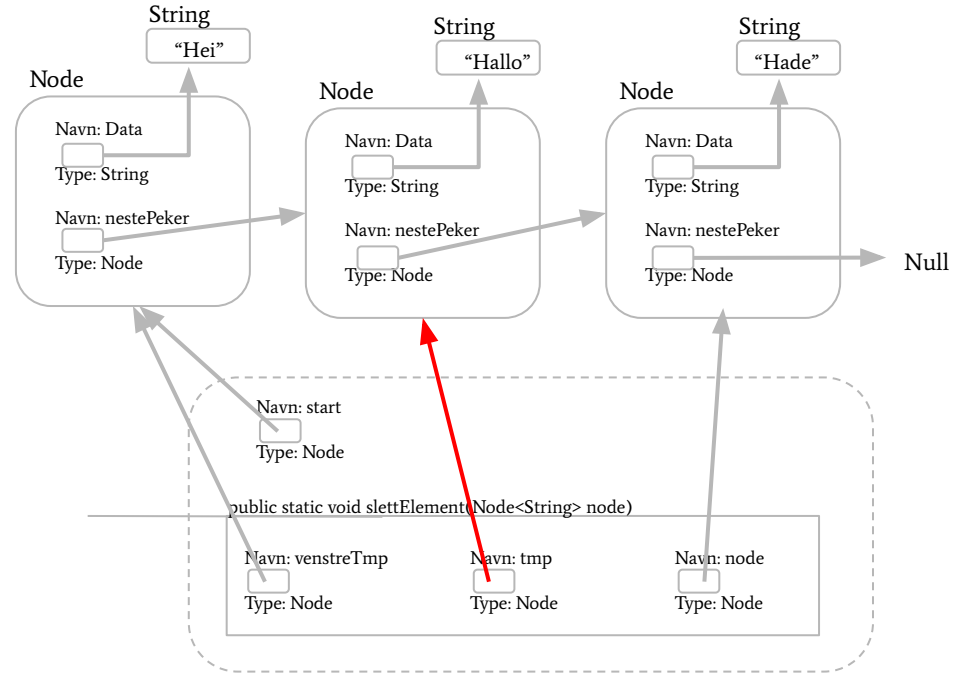
# Slette element fra enkeltlenket liste

```
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

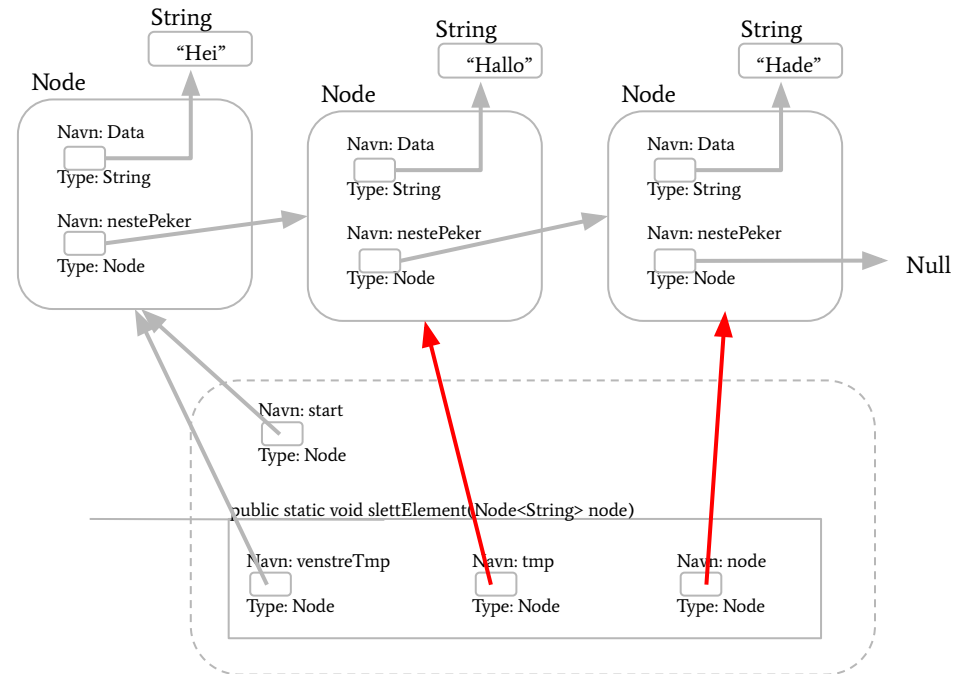
```
public static void slettElement(Node<String> node) {  
    if(start == null){  
        return;  
    }  
    if(start.equals(node)){  
        start = start.hentNeste();  
        return;  
    }  
    Node<String> venstreTmp = start;  
    Node<String> tmp = start.hentNeste();  
    while(tmp !=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





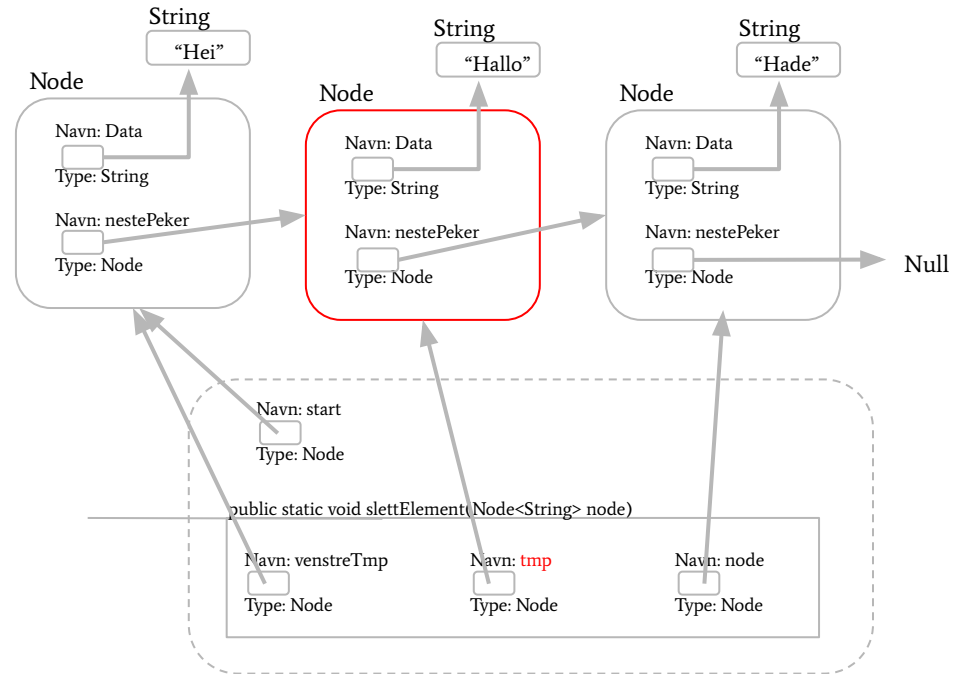
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



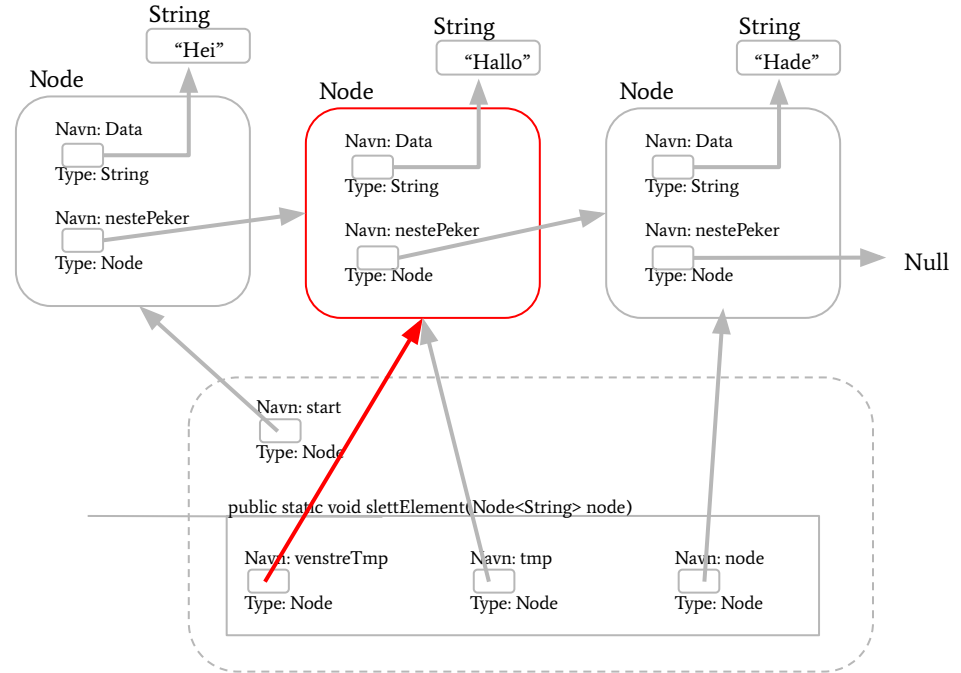
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



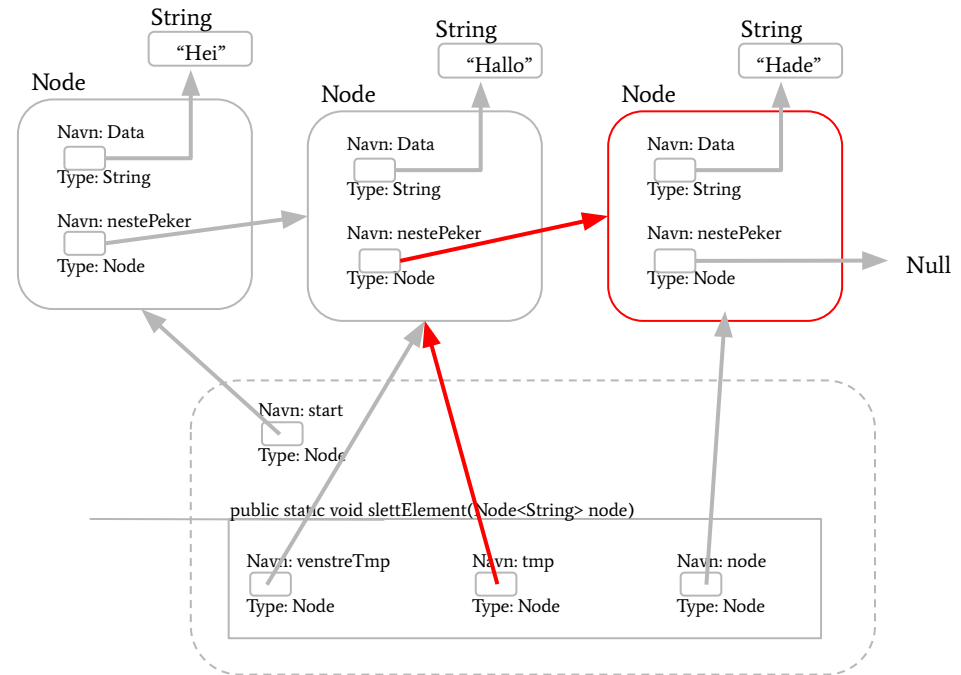
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



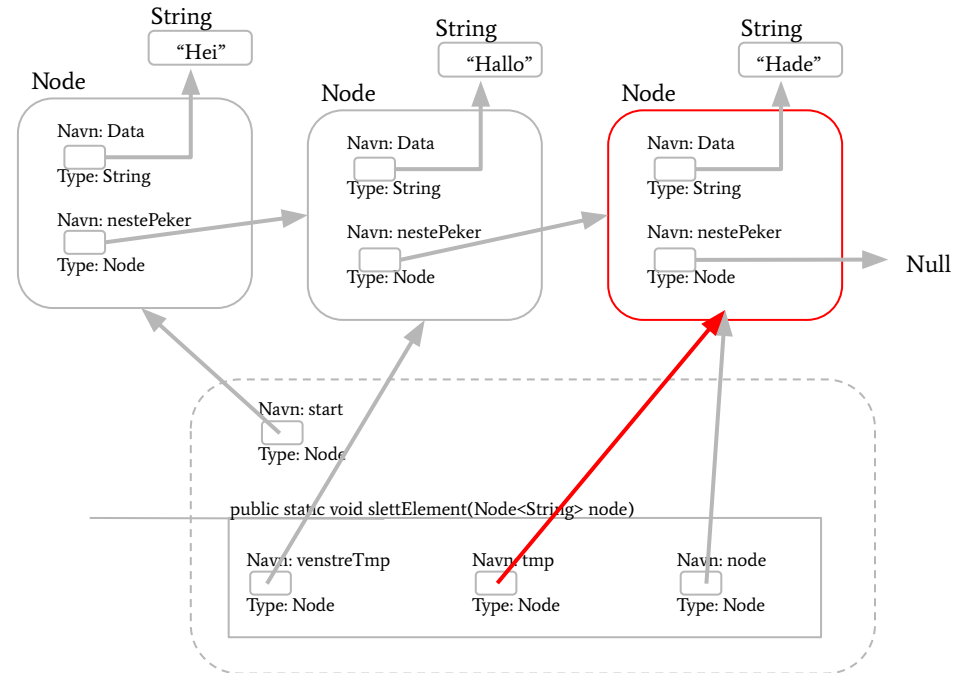
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



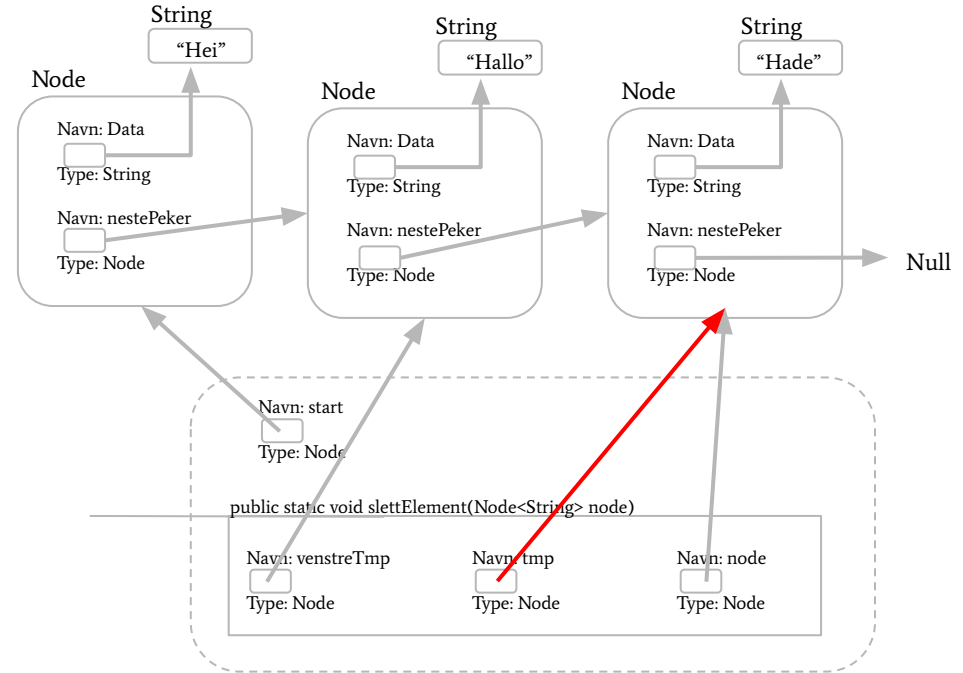
# Slette element fra enkeltlenket liste

```
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.setNeste(tmp.hentNeste());
            return;
        }
        venstreTmp = tmp;
        tmp = tmp.hentNeste();
    }
}
```



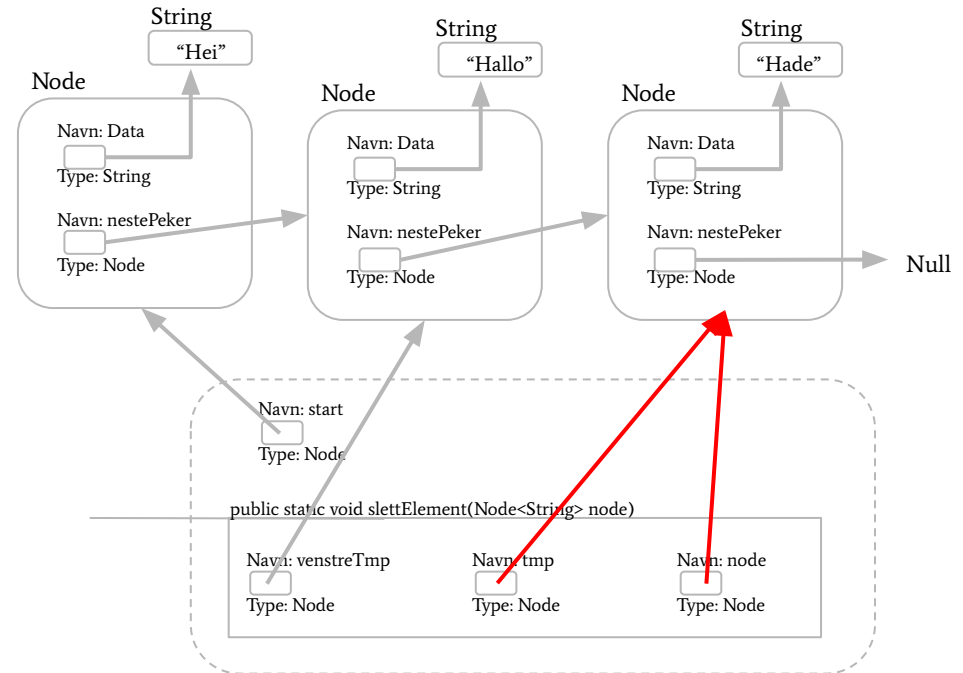
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



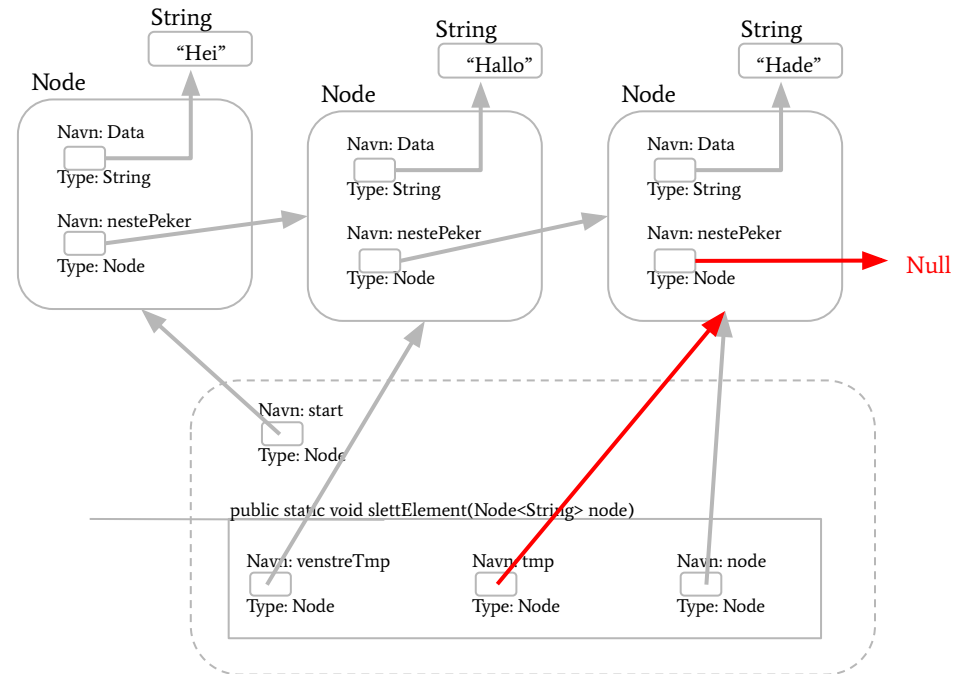
# Slette element fra enkeltlenket liste

```
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

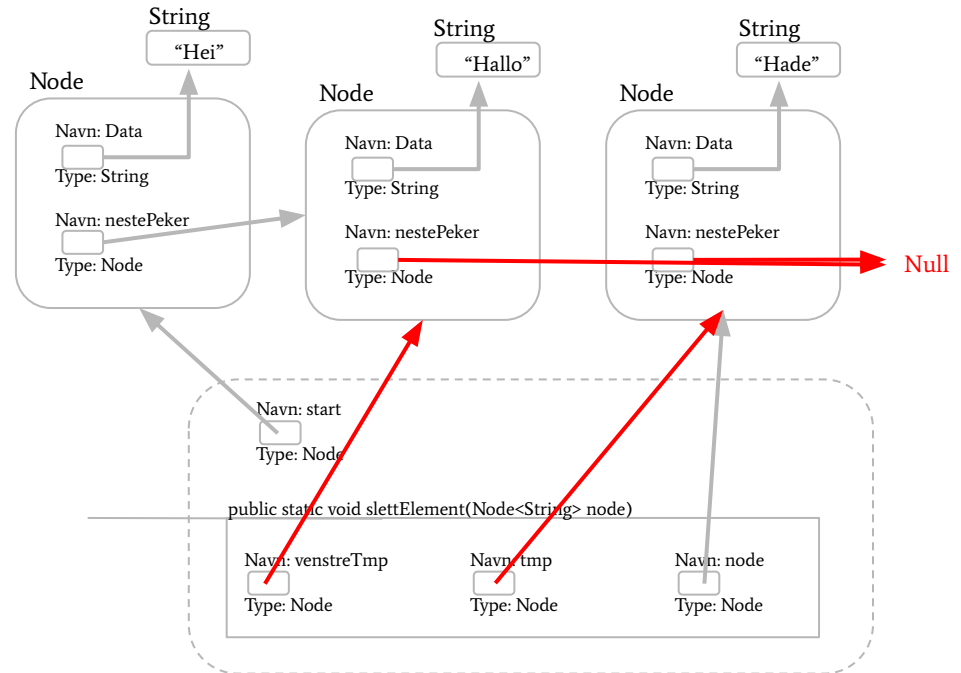
```
public static void slettElement(Node<String> node) {  
    if(start == null){  
        return;  
    }  
    if(start.equals(node)){  
        start = start.hentNeste();  
        return;  
    }  
    Node<String> venstreTmp = start;  
    Node<String> tmp = start.hentNeste();  
    while(tmp !=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





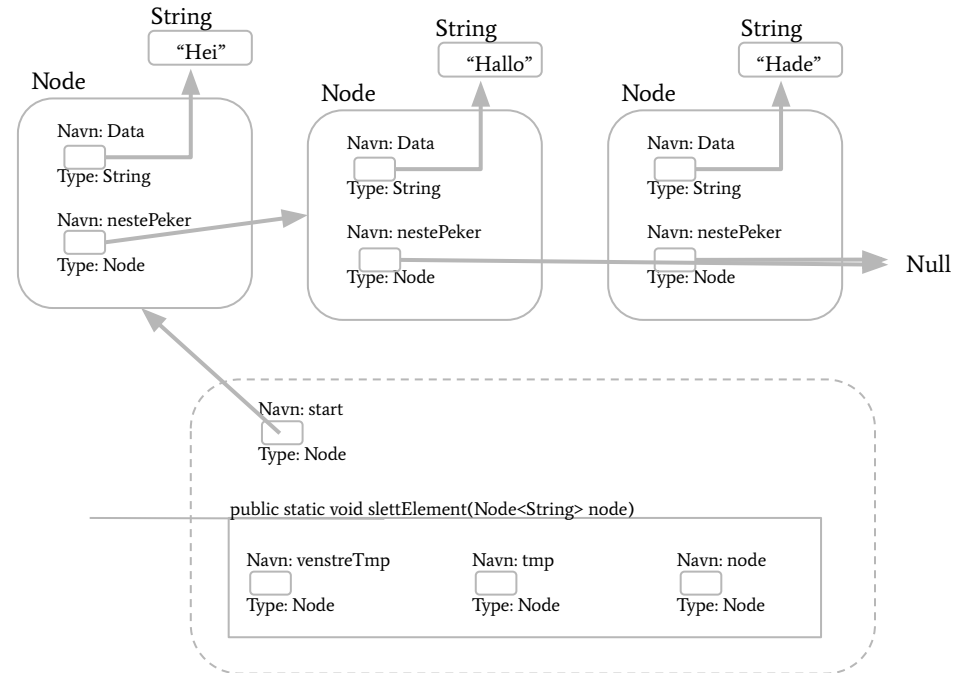
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



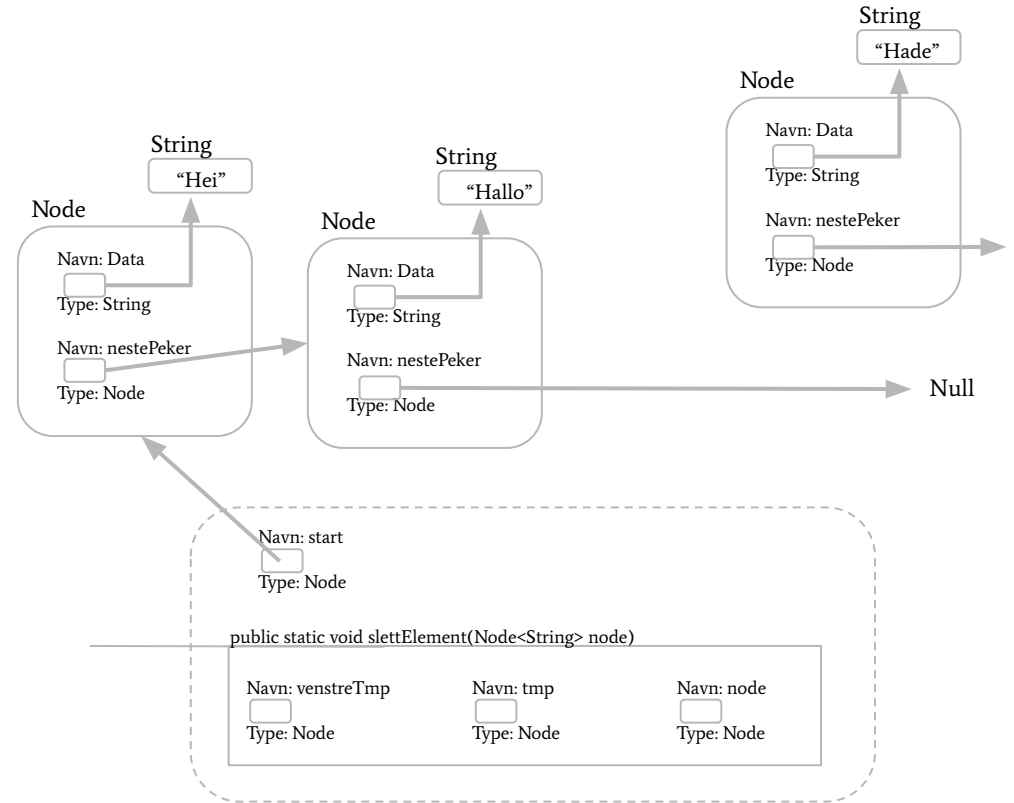
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



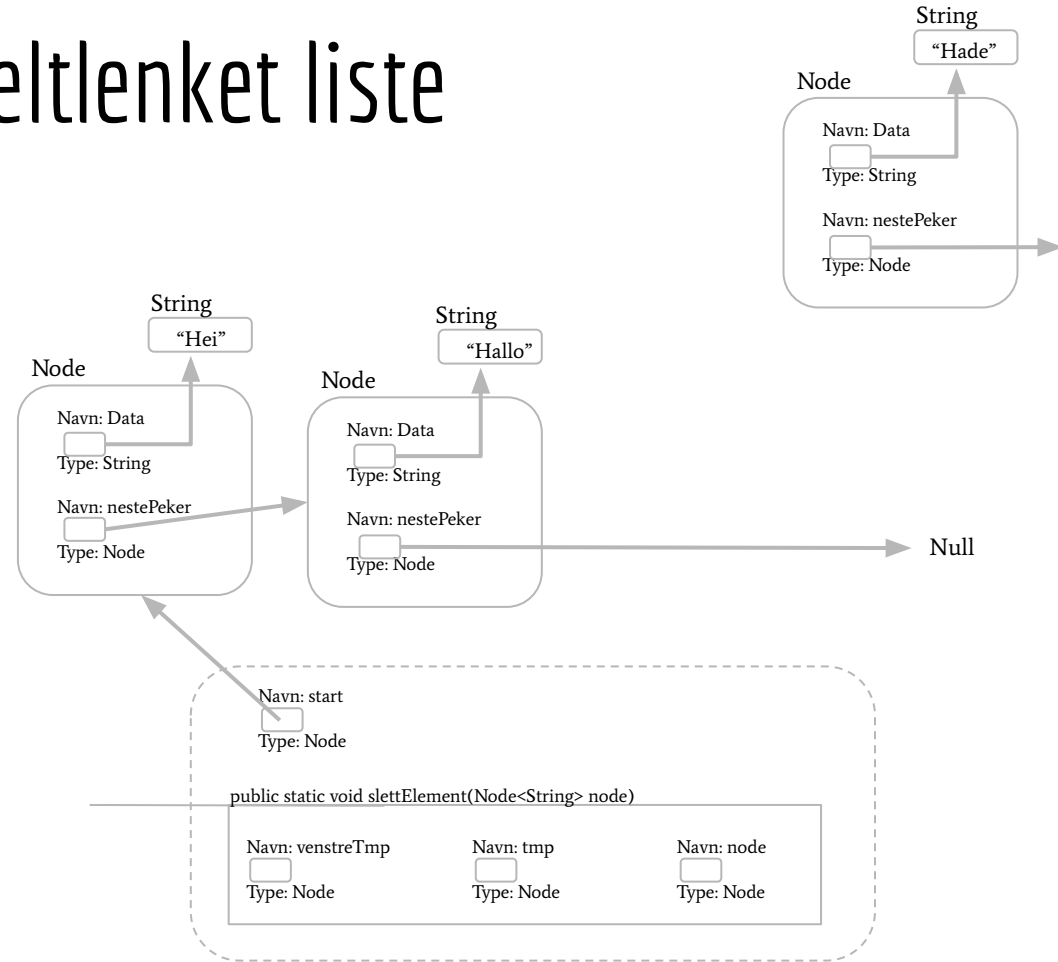
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



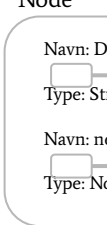
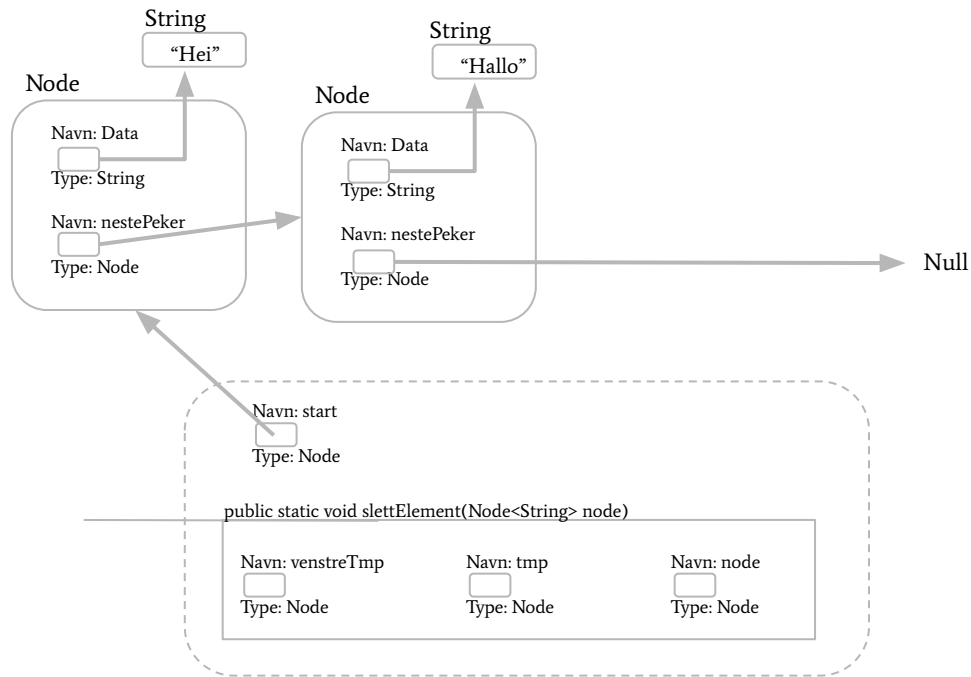
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```



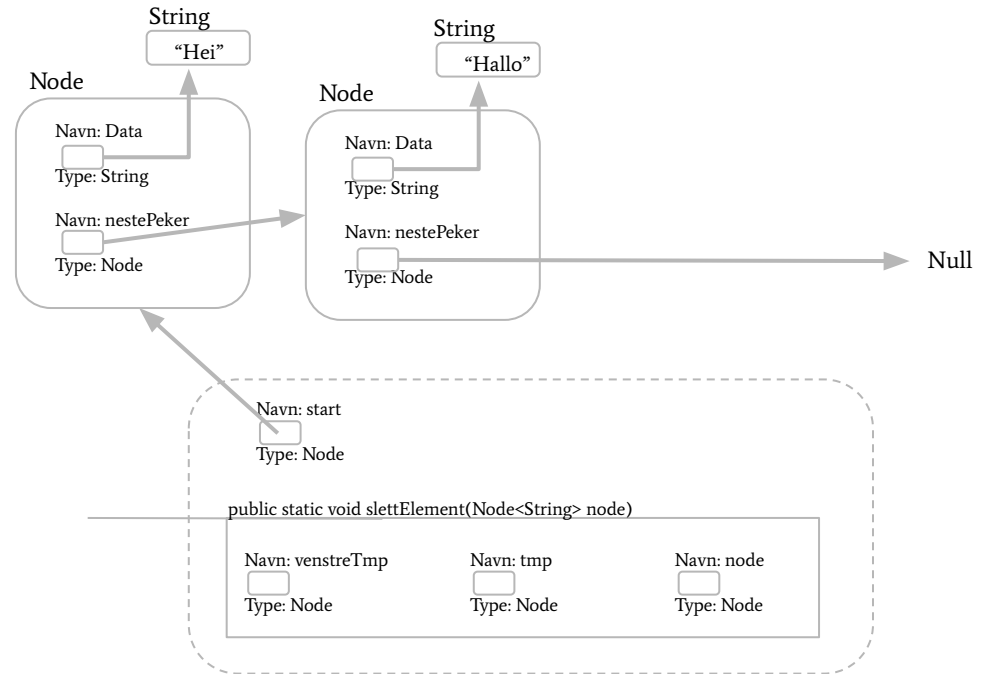
# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```

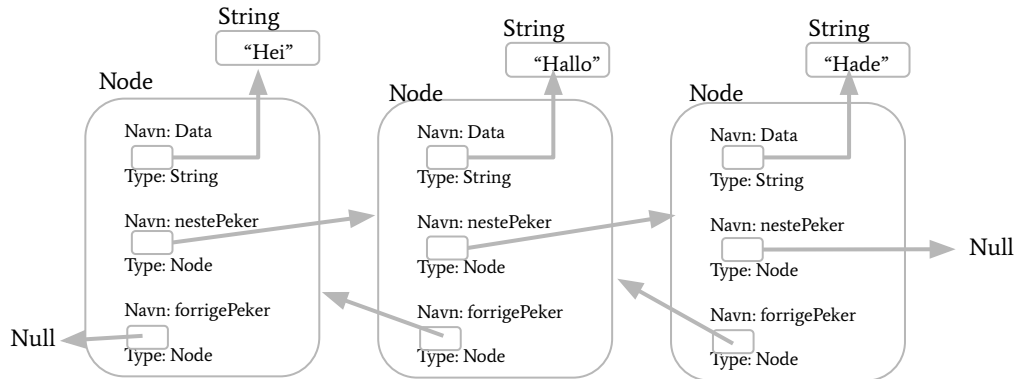
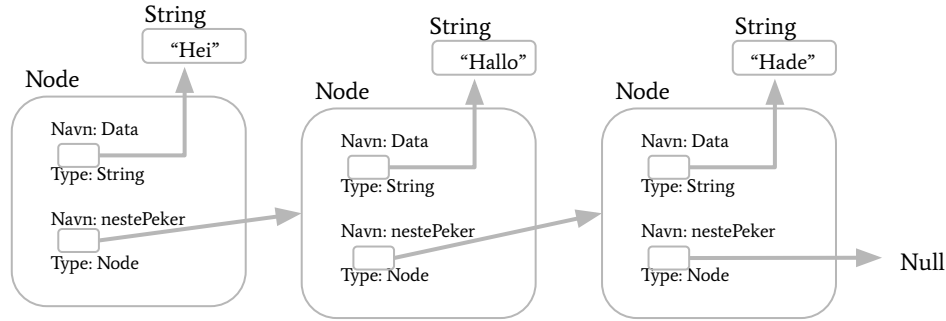


# Slette element fra enkeltlenket liste

```
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();  
    }  
}
```

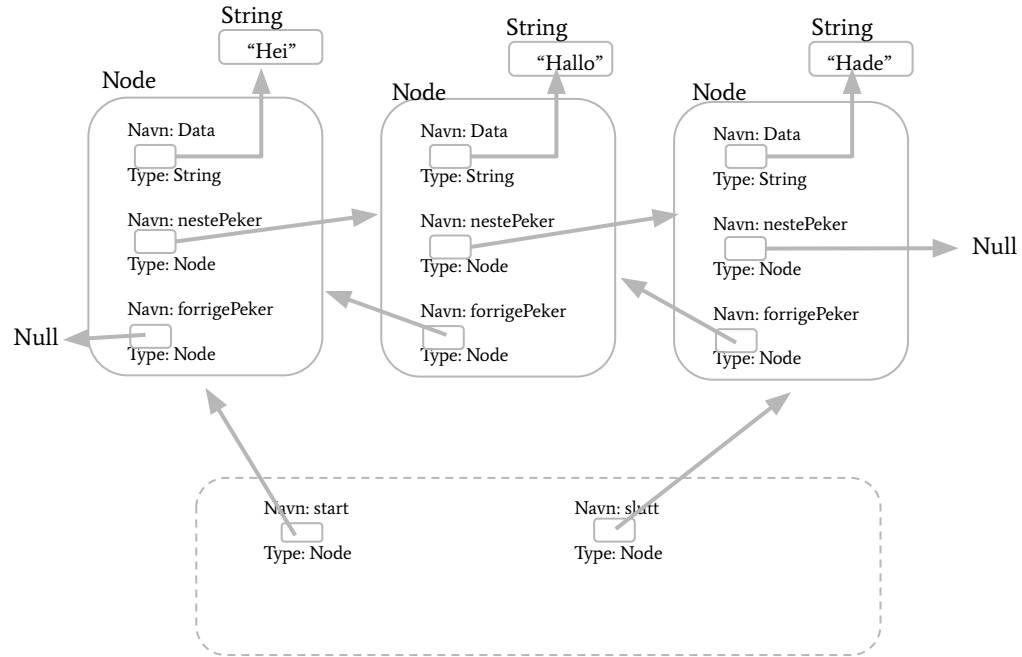


# Enkeltlenket vs dobbeltlenket lenkeliste



# Slette element fra dobbeltlenket liste

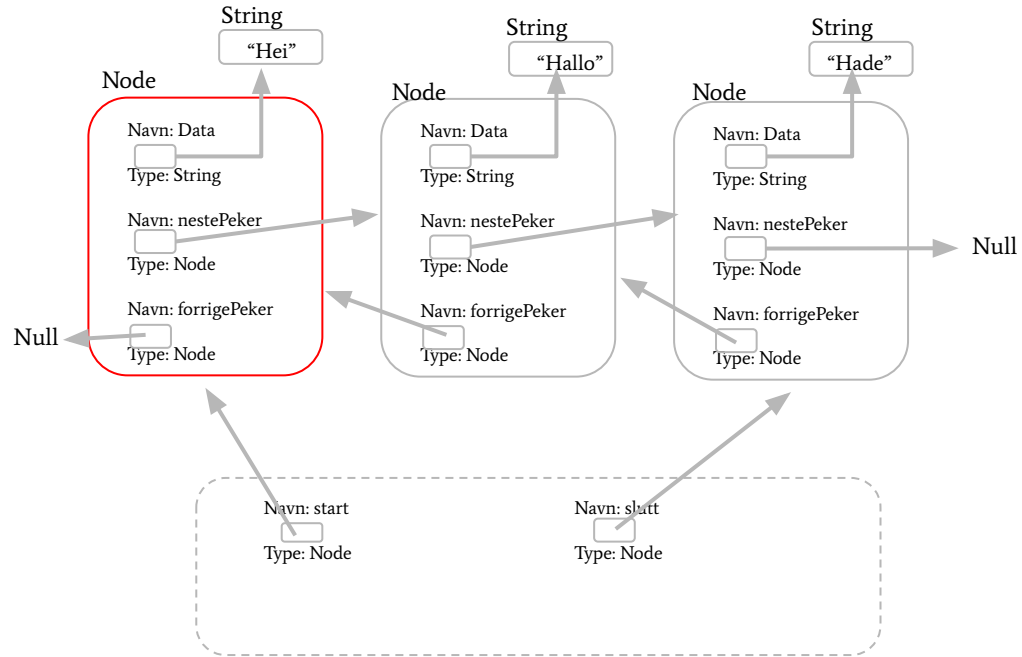
Se på alle ulike cases:





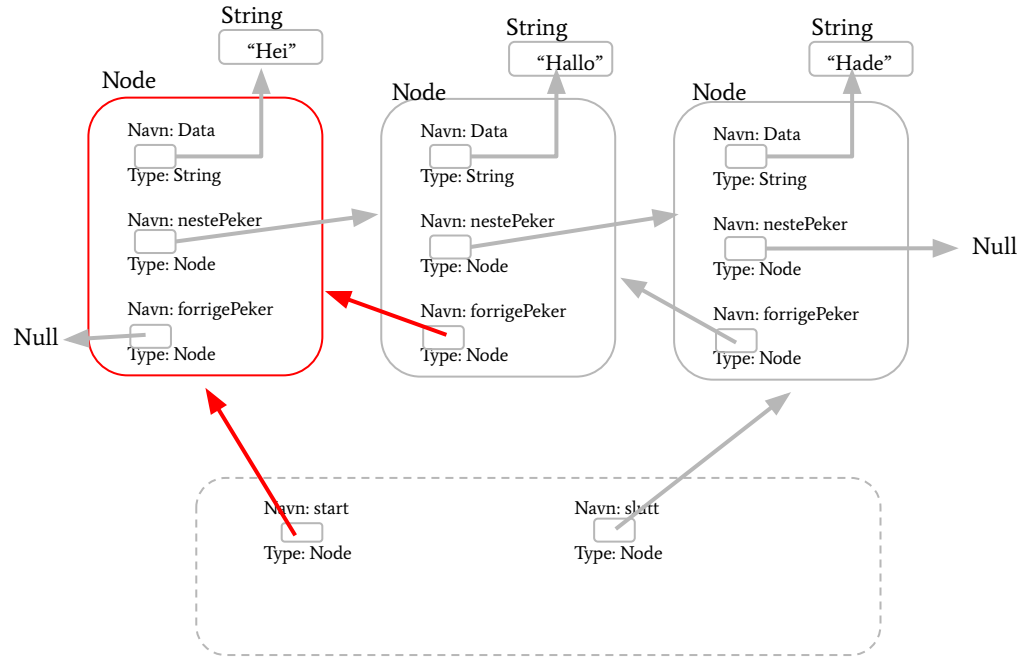
# Slette element fra dobbeltlenket liste

Slette første element



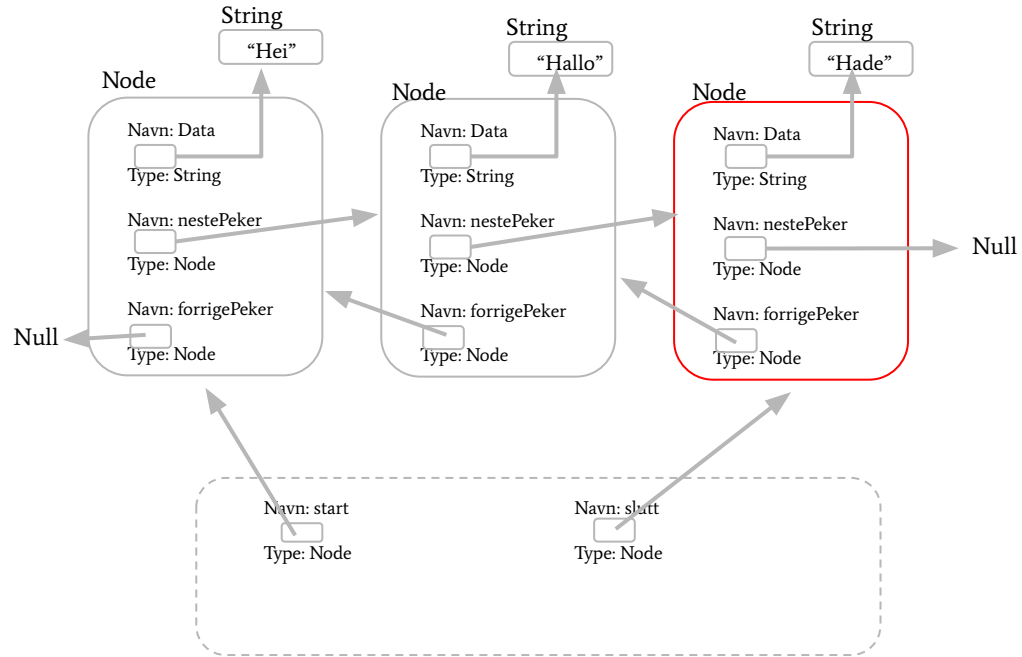
# Slette element fra dobbeltlenket liste

Slette første element



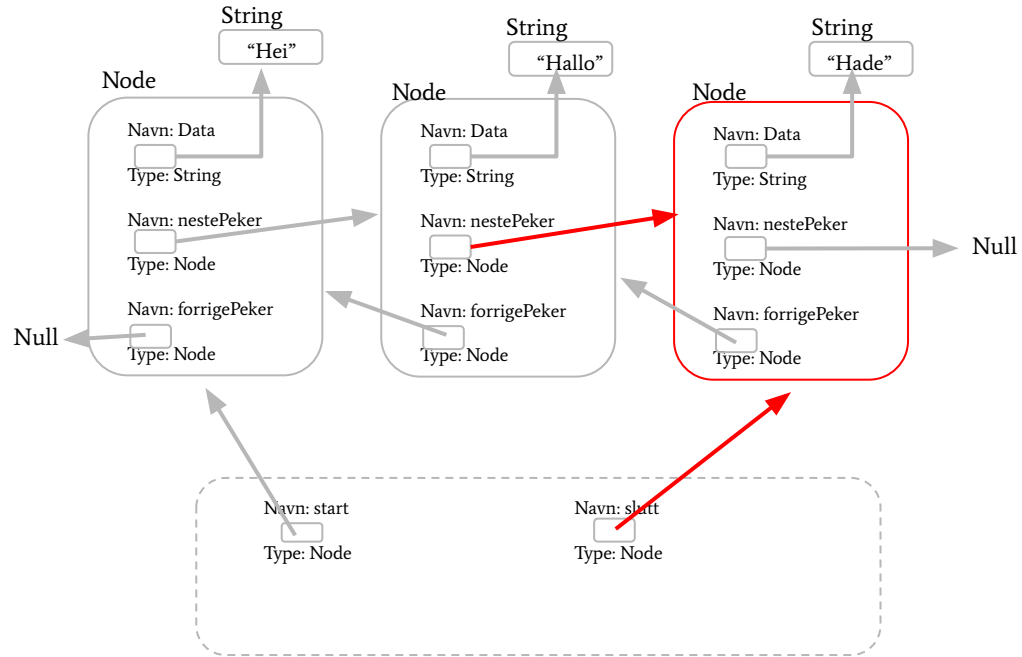
# Slette element fra dobbeltlenket liste

Slette siste element

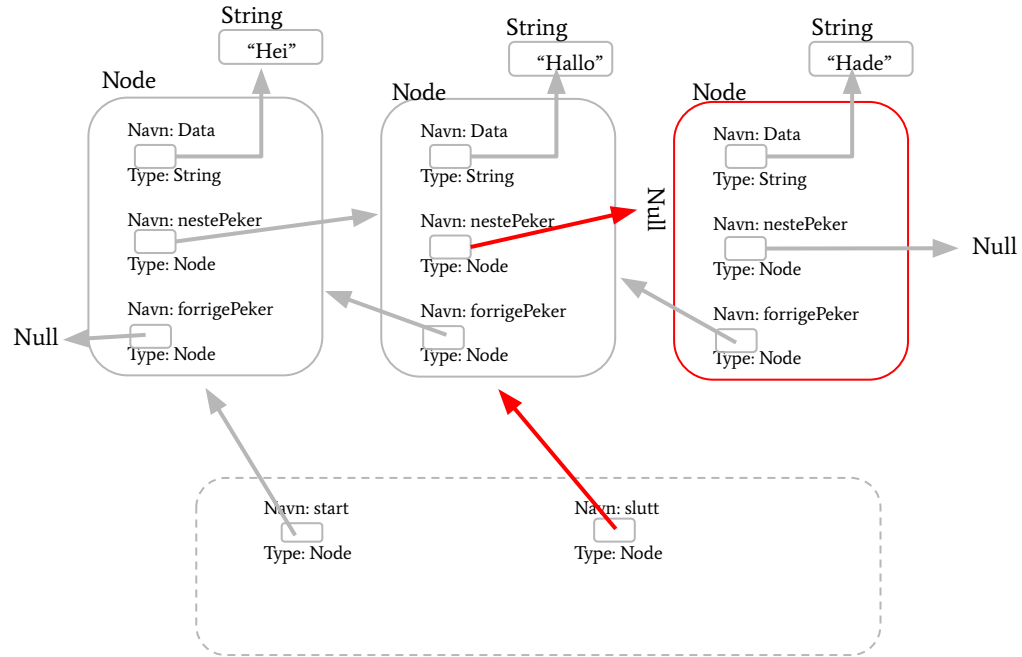


# Slette element fra dobbeltlenket liste

Slette siste element

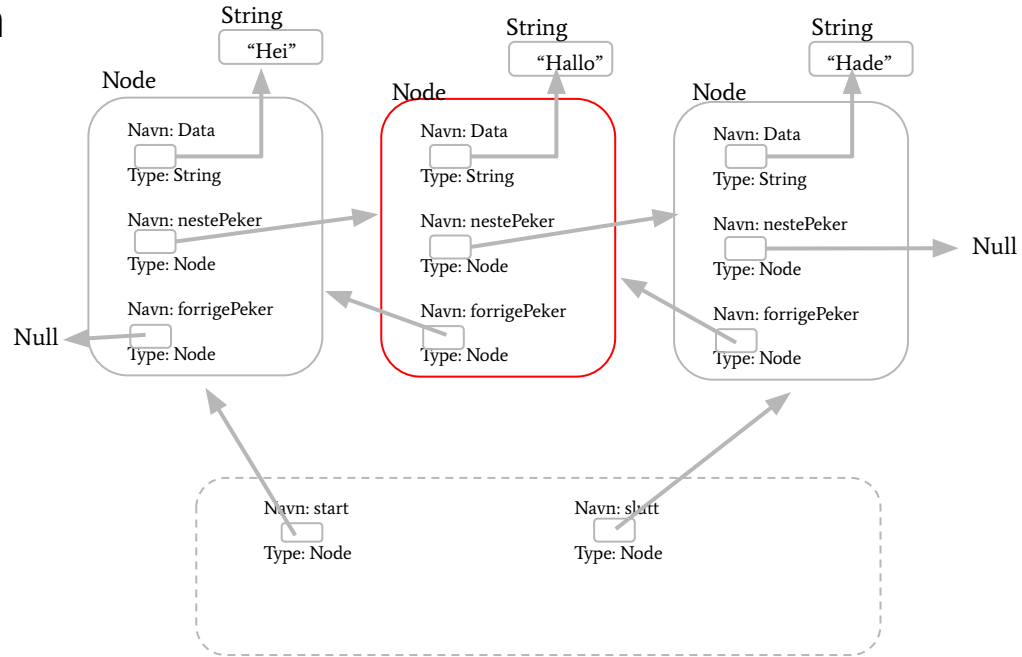


# Slette element fra dobbeltlenket liste



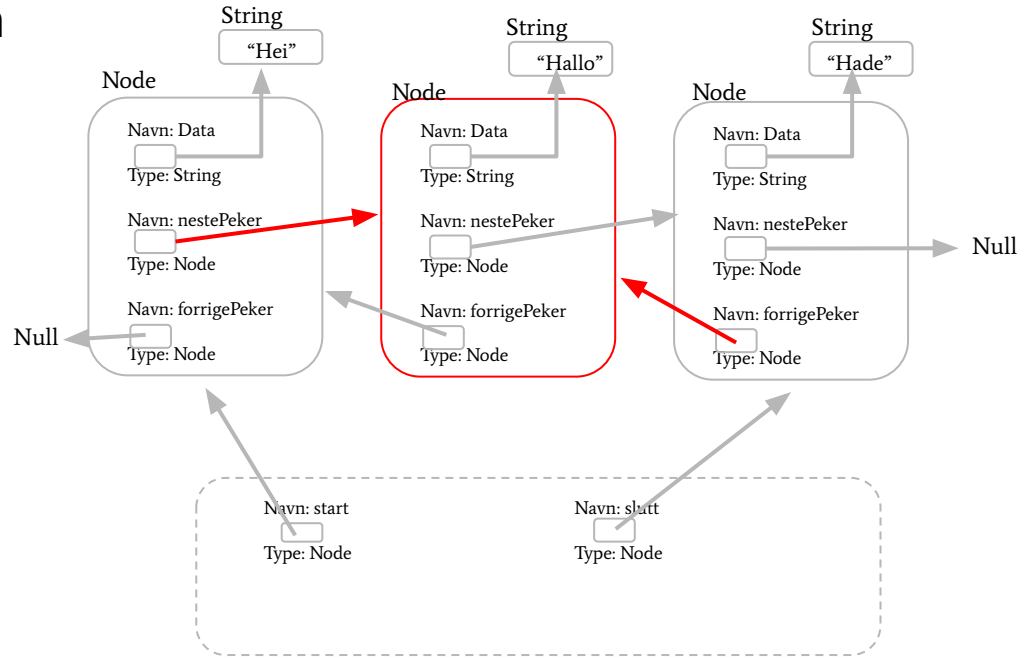
# Slette element fra dobbeltlenket liste

Slette element midt i listen



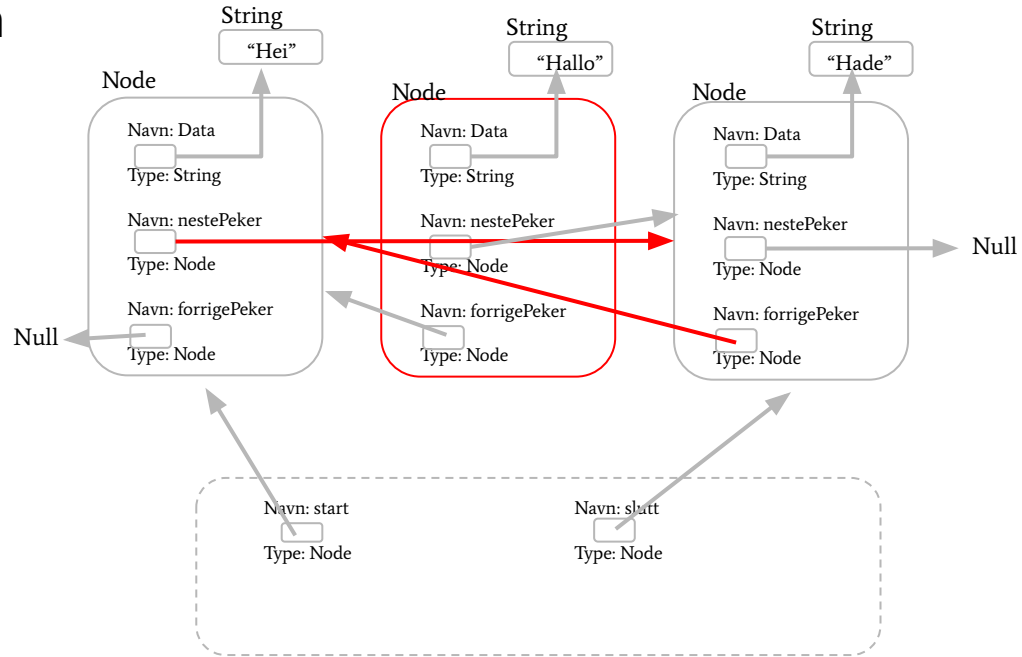
# Slette element fra dobbeltlenket liste

Slette element midt i listen



# Slette element fra dobbeltlenket liste

Slette element midt i listen

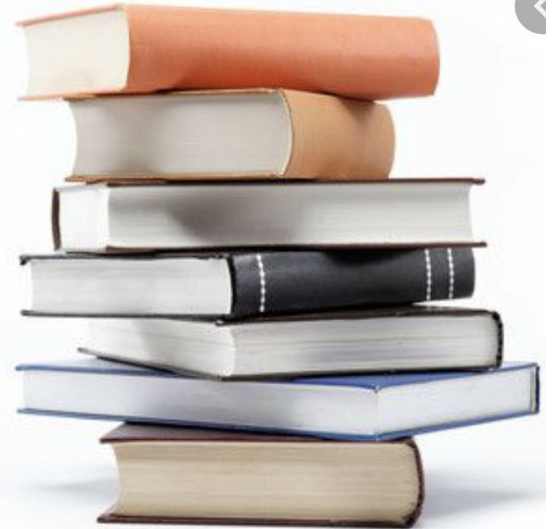
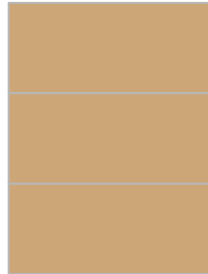




# Stabel/Stack: LIFO (Last In First Out)

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

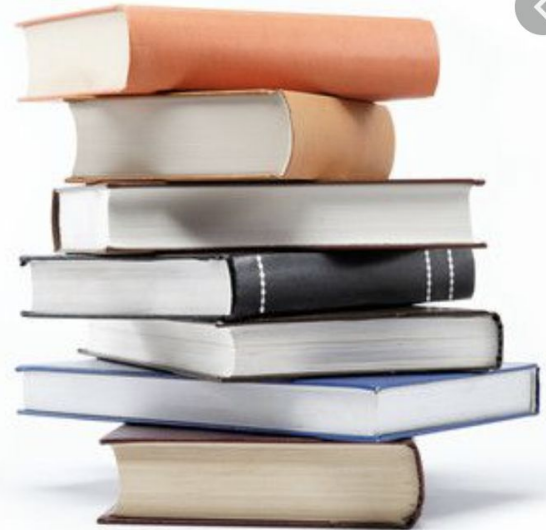
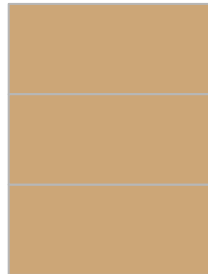
push: 



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

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

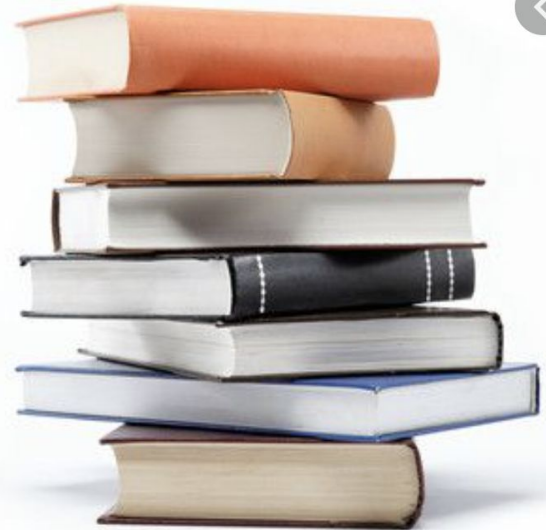
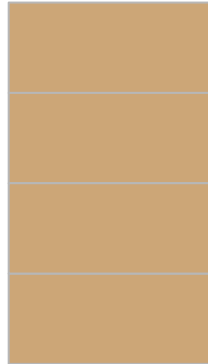
push:



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

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

push:

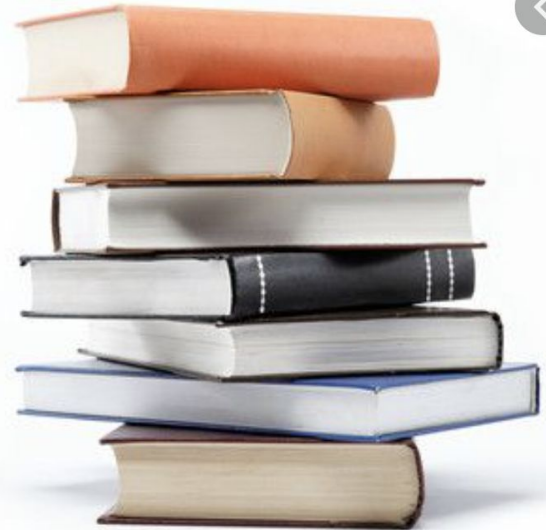
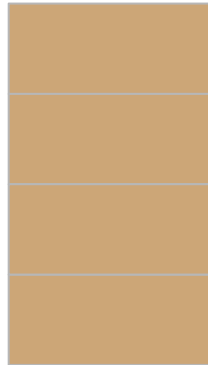


# Stabel/Stack: LIFO (Last In First Out)

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

push:

pop:

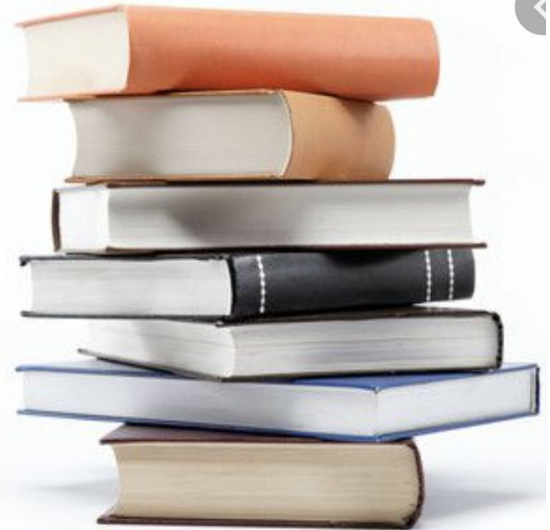
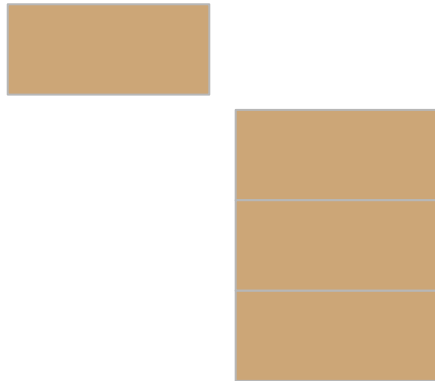


# Stabel/Stack: LIFO (Last In First Out)

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

push:

pop:

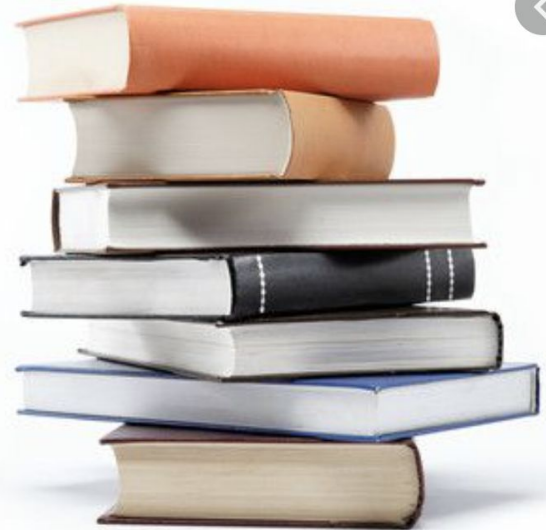
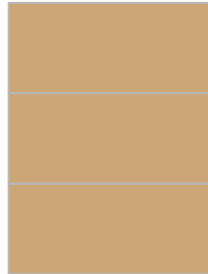


# Stabel/Stack: LIFO (Last In First Out)

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:

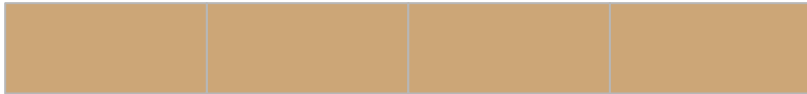




# Kø/queue: FIFO (First In First Out)

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

add:

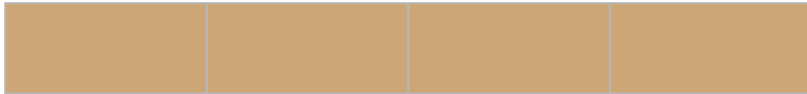


# Kø/queue: FIFO (First In First Out)

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 compareTo() 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!



# Oppmøteregistrering:

<https://nettskjema.no/a/188717>



# Send meg en direkte melding 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

Jobbe med oppgaver