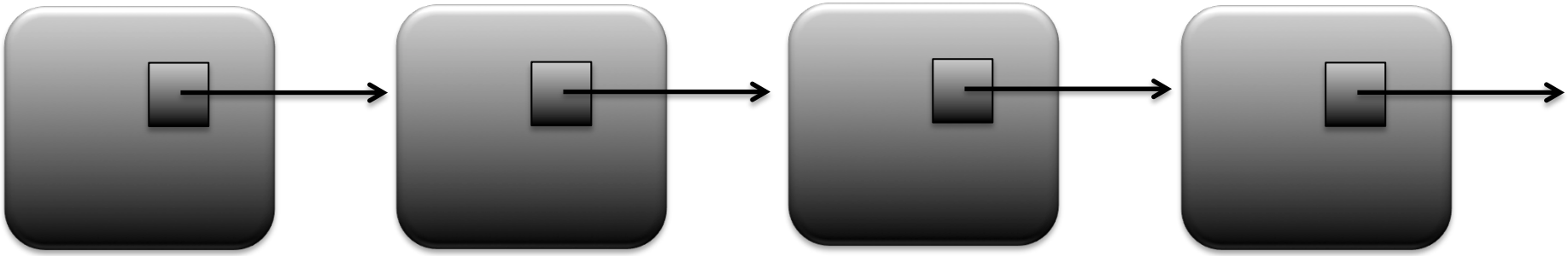
INF1010

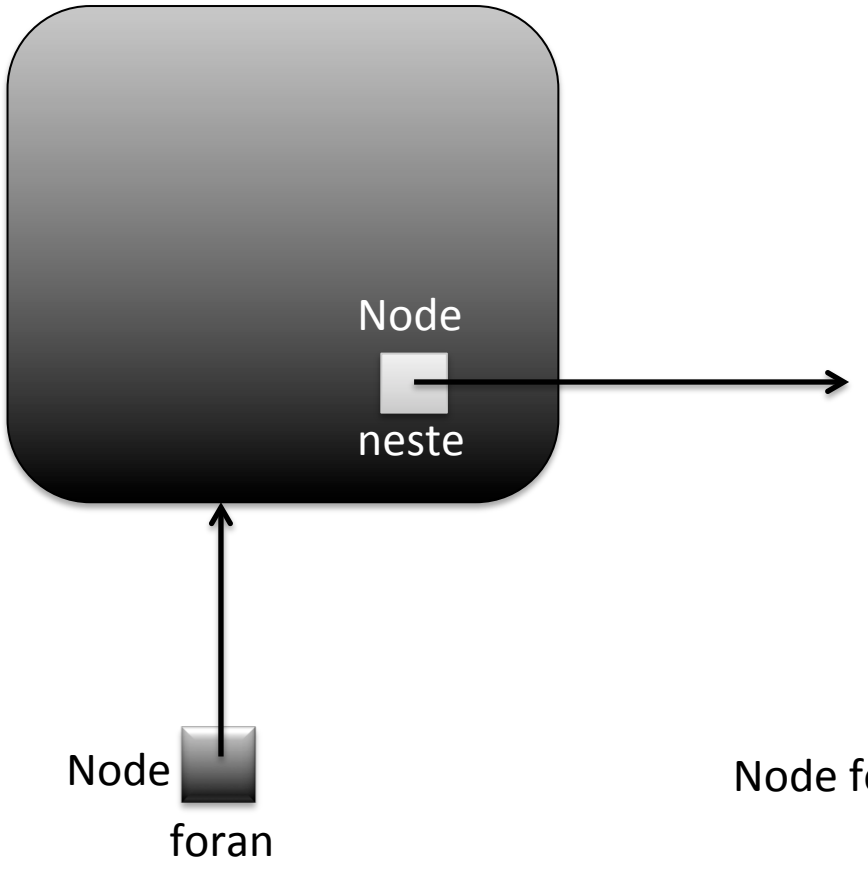
Stein Michael Storleer (michael)

Lenkelister



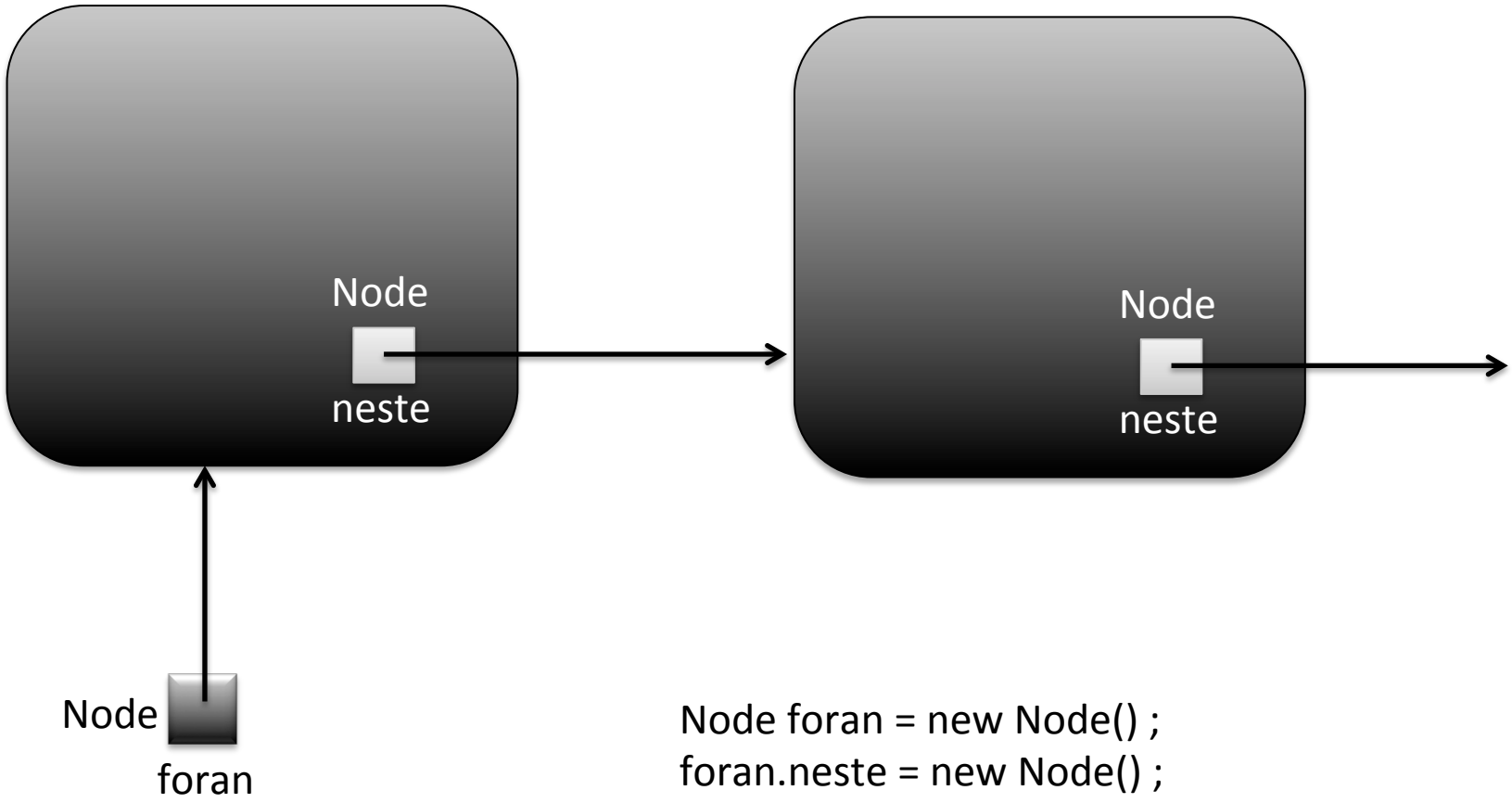
Datastrukturen lenkeliste

```
class Node {
    Node neste = null ;
}
```



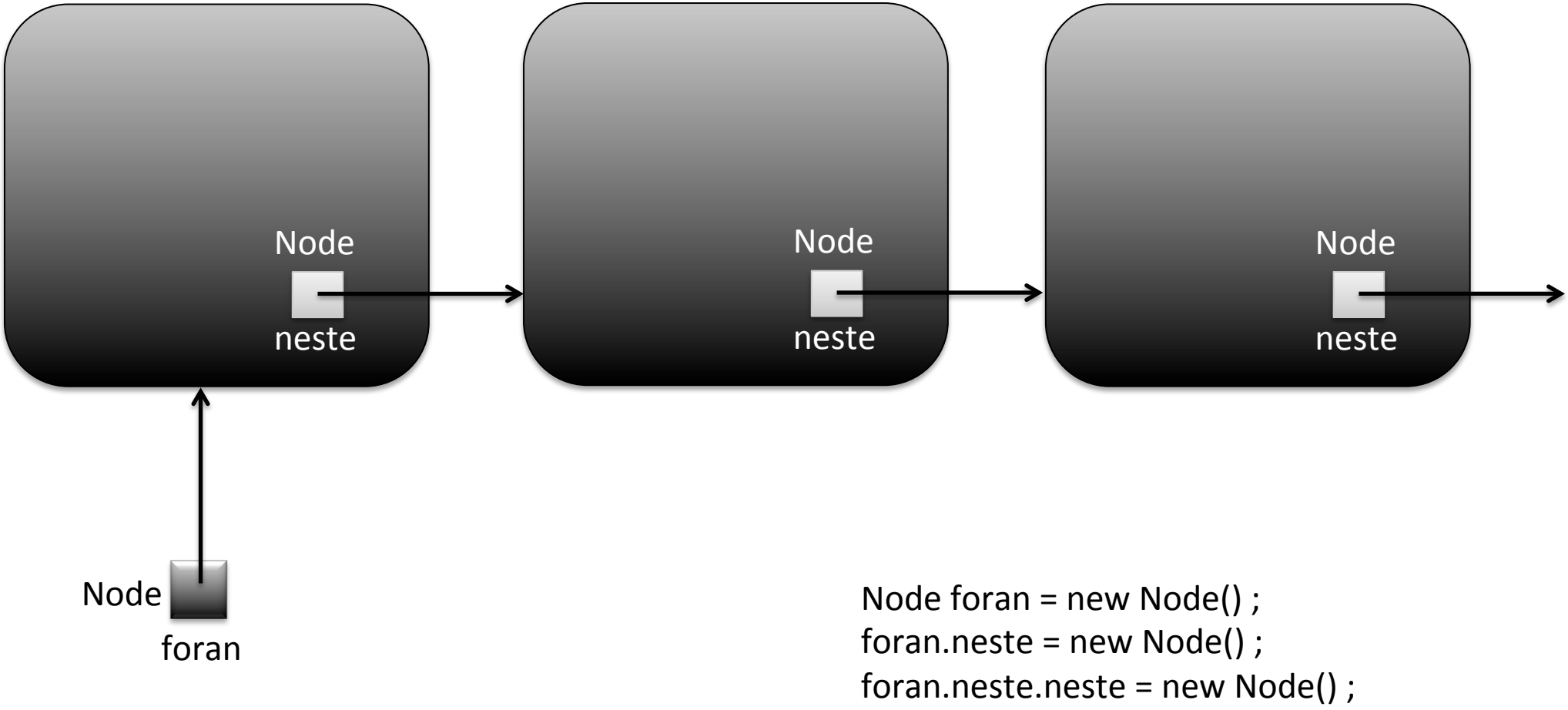

```
Node foran = new Node() ;
```

```
class Node {
    Node neste = null ;
}
```



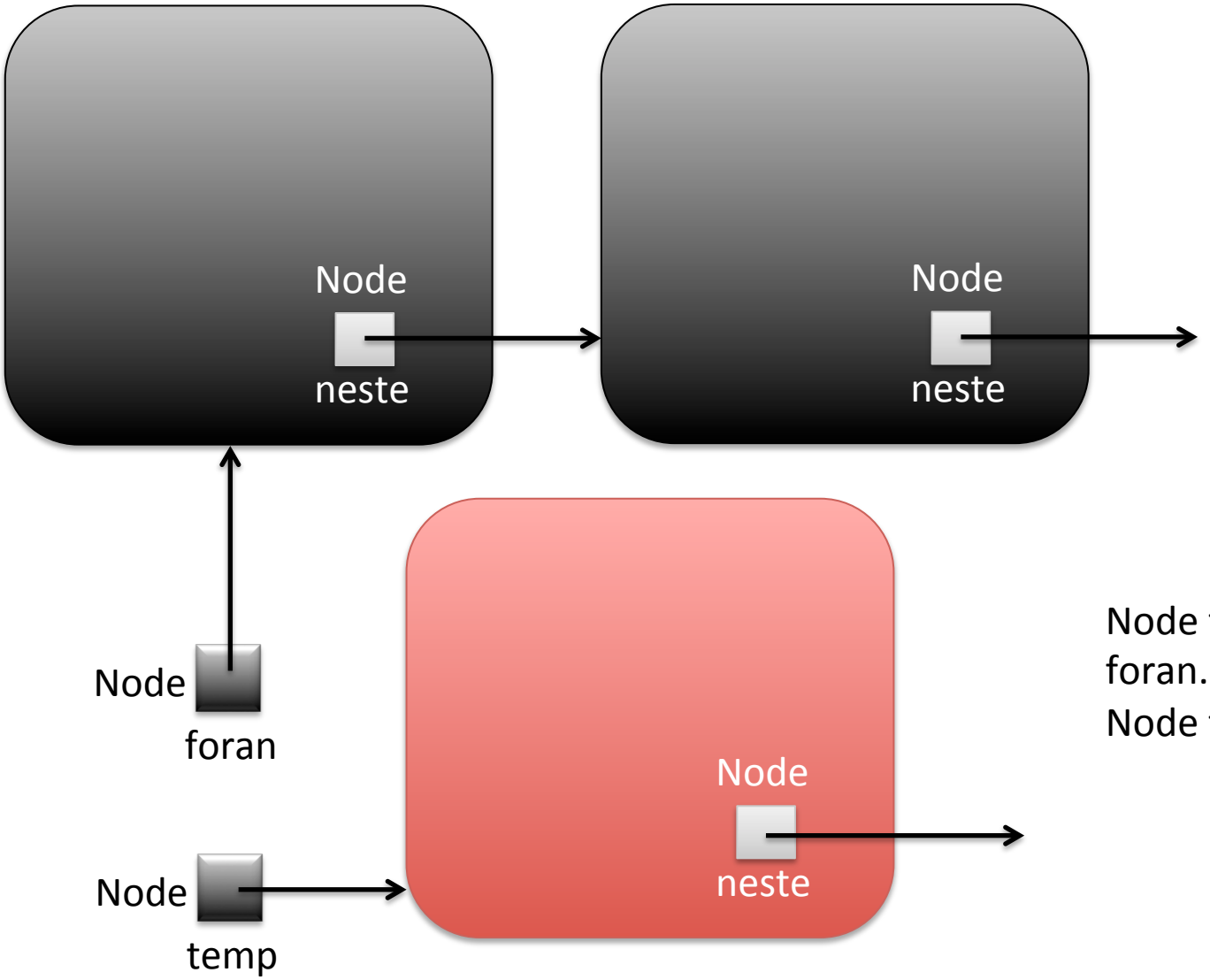

```
Node foran = new Node() ;
foran.neste = new Node() ;
```

```
class Node {
    Node neste = null ;
}
```



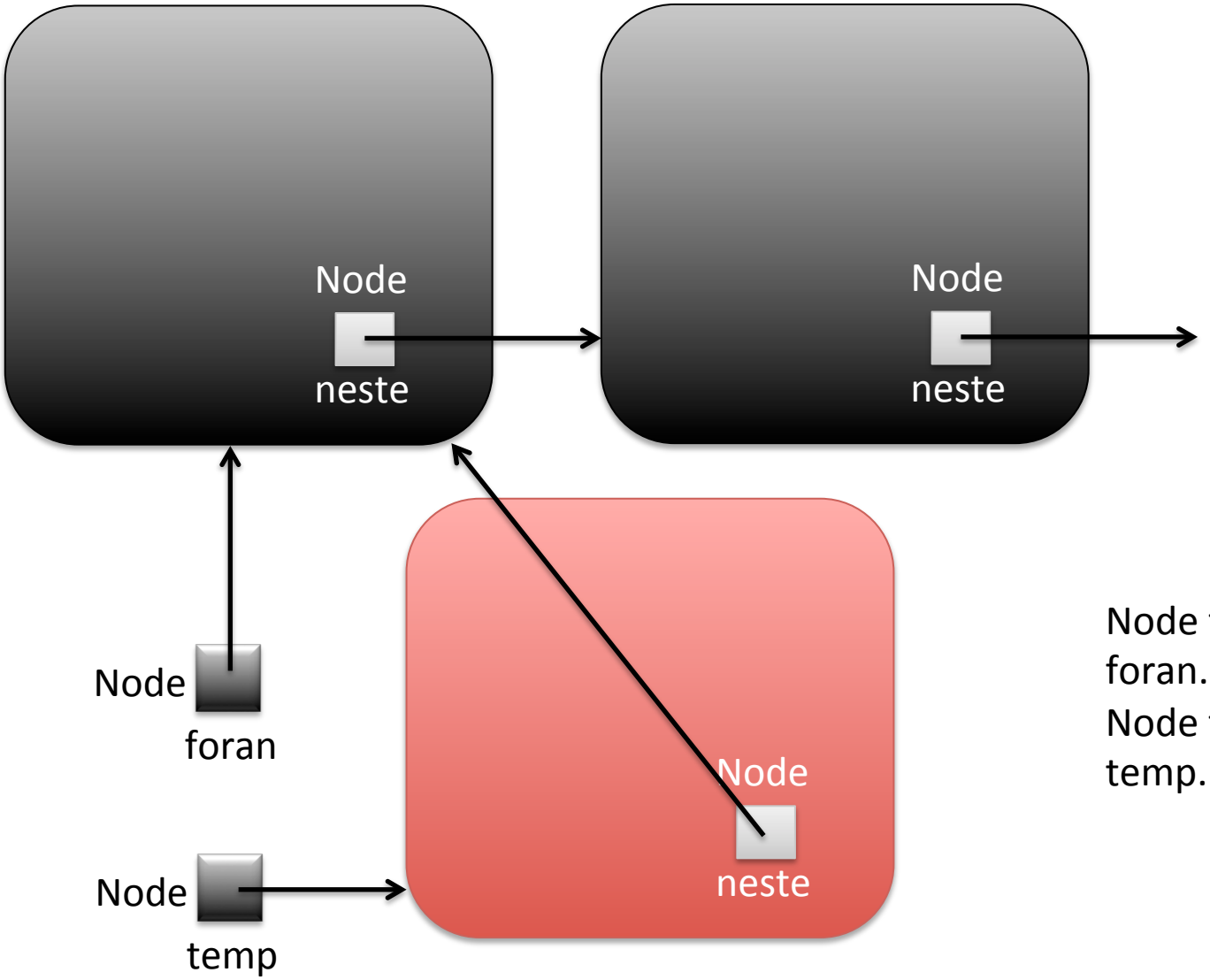

```
Node foran = new Node() ;
foran.neste = new Node() ;
foran.neste.neste = new Node() ;
```

```
class Node {
    Node neste = null ;
}
```



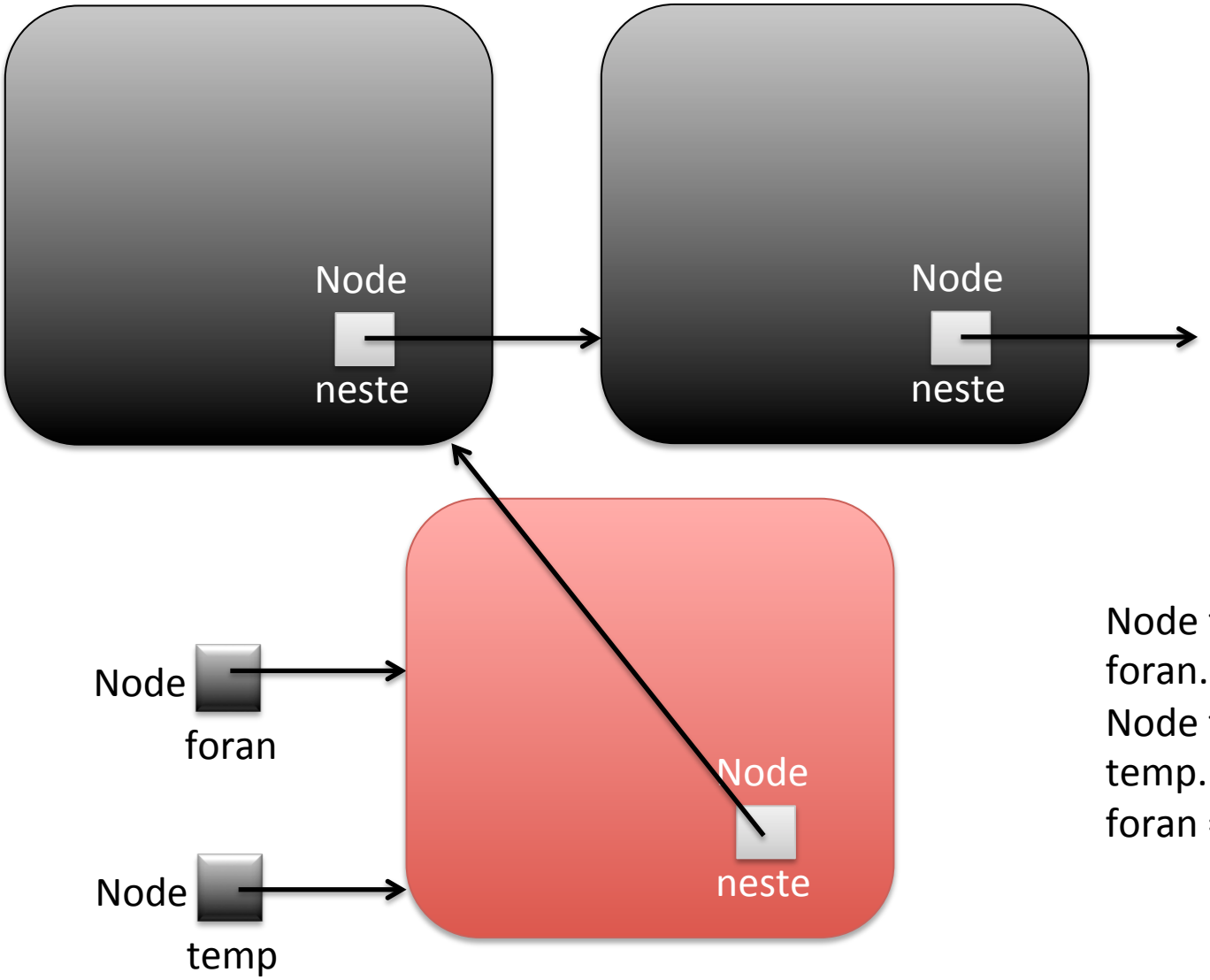

```
Node foran = new Node() ;
foran.neste = new Node() ;
Node temp = new Node() ;
```

```
class Node {
    Node neste = null ;
}
```



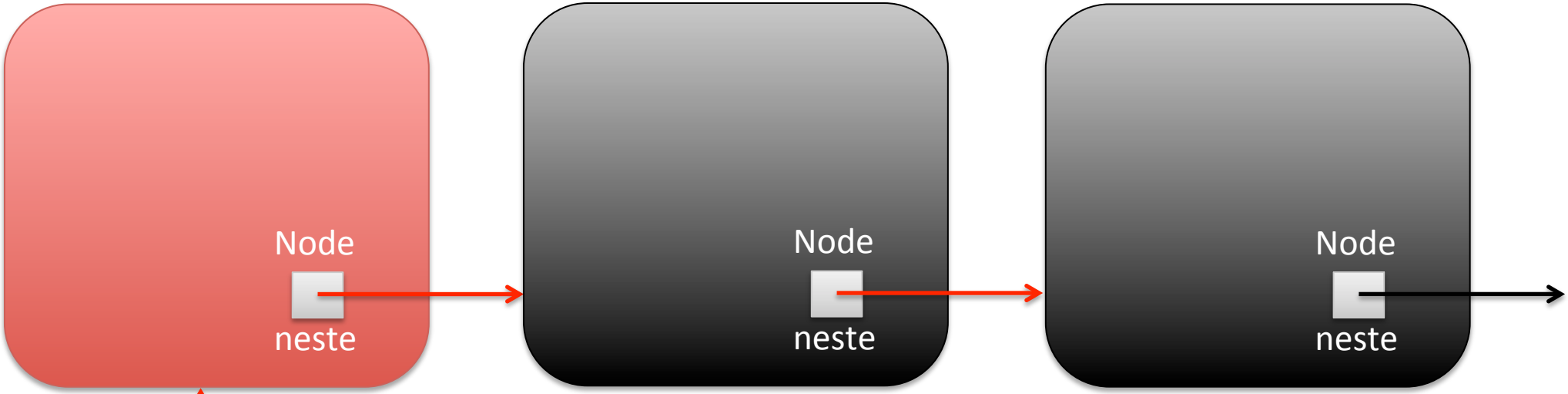

```
Node foran = new Node() ;
foran.neste = new Node() ;
Node temp = new Node() ;
temp.neste = foran ;
```

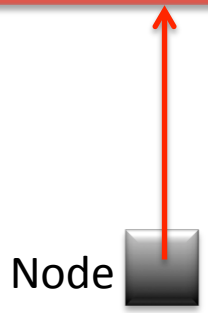
```
class Node {
    Node neste = null ;
}
```

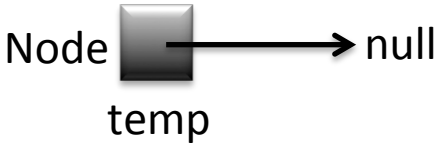



```
Node foran = new Node() ;
foran.neste = new Node() ;
Node temp = new Node() ;
temp.neste = foran ;
foran = temp ;
```

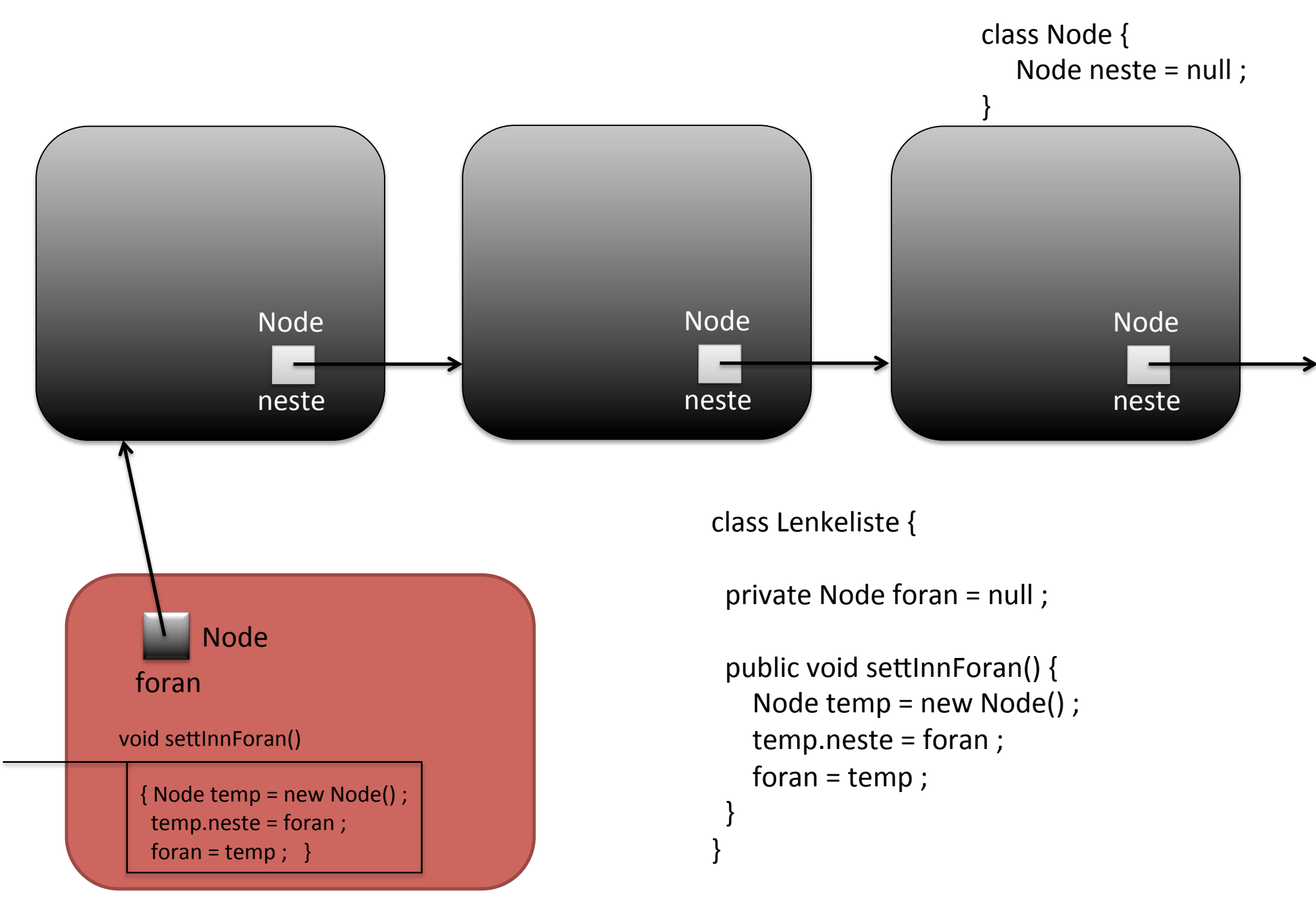
```
class Node {
    Node neste = null ;
}
```

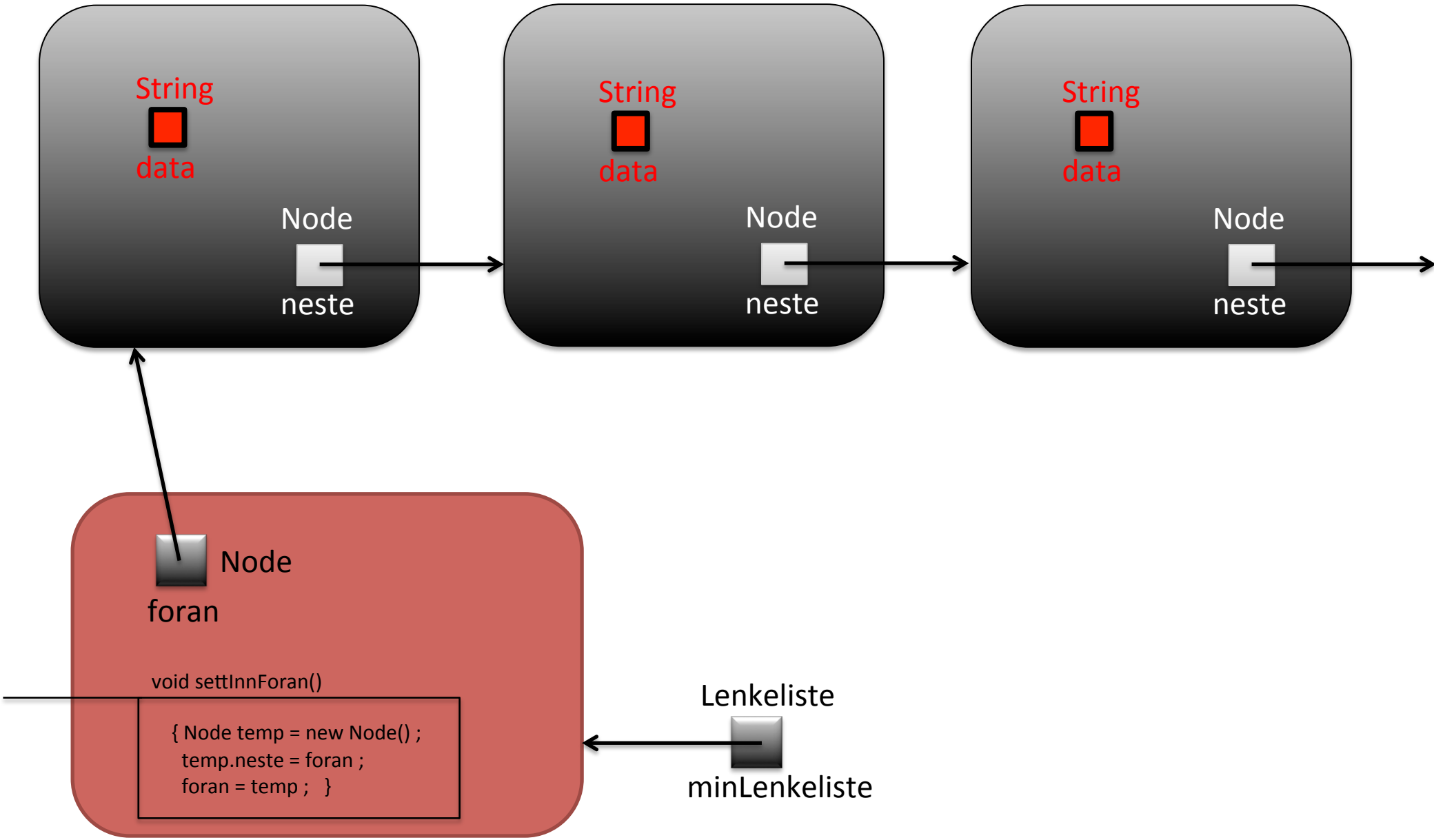


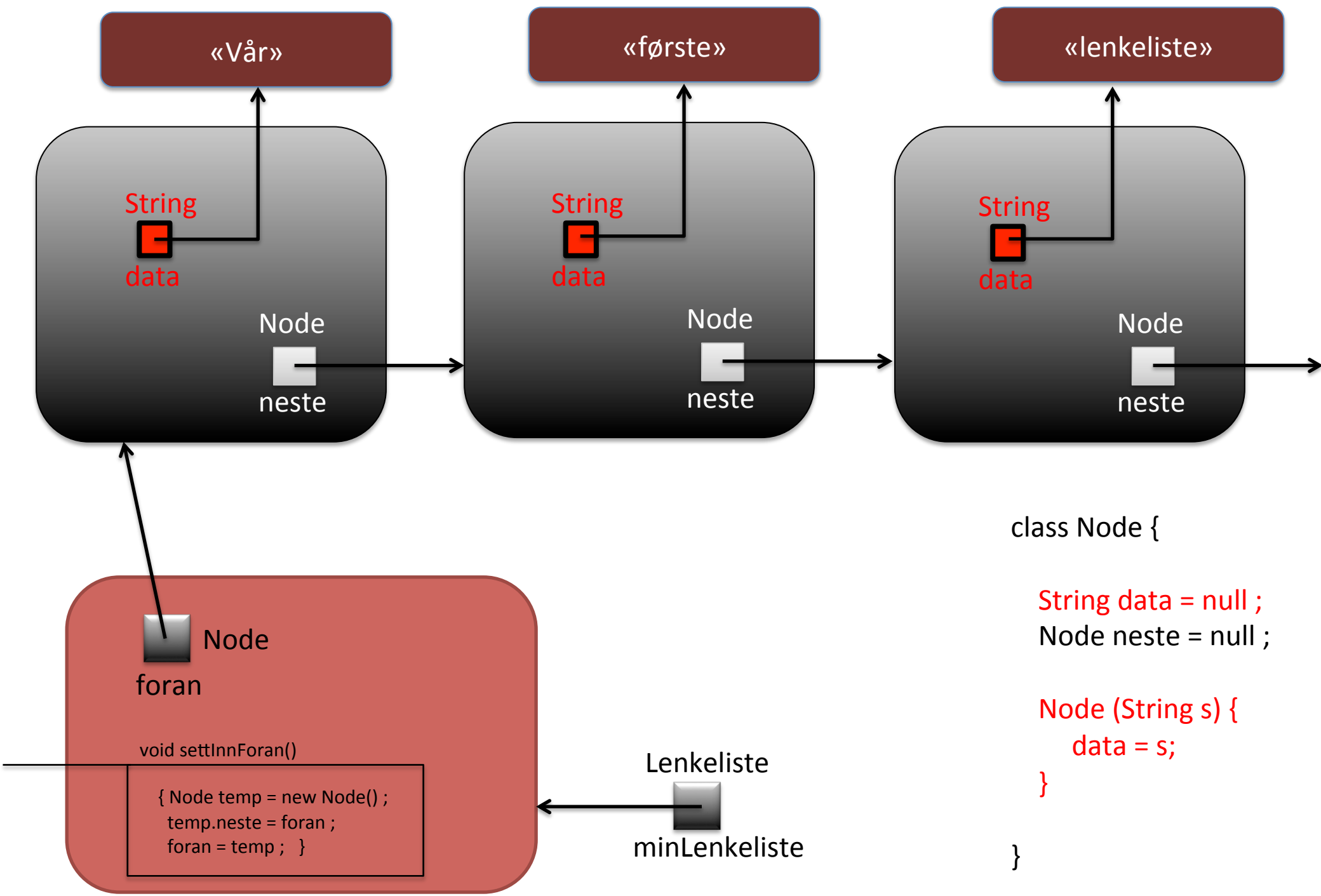




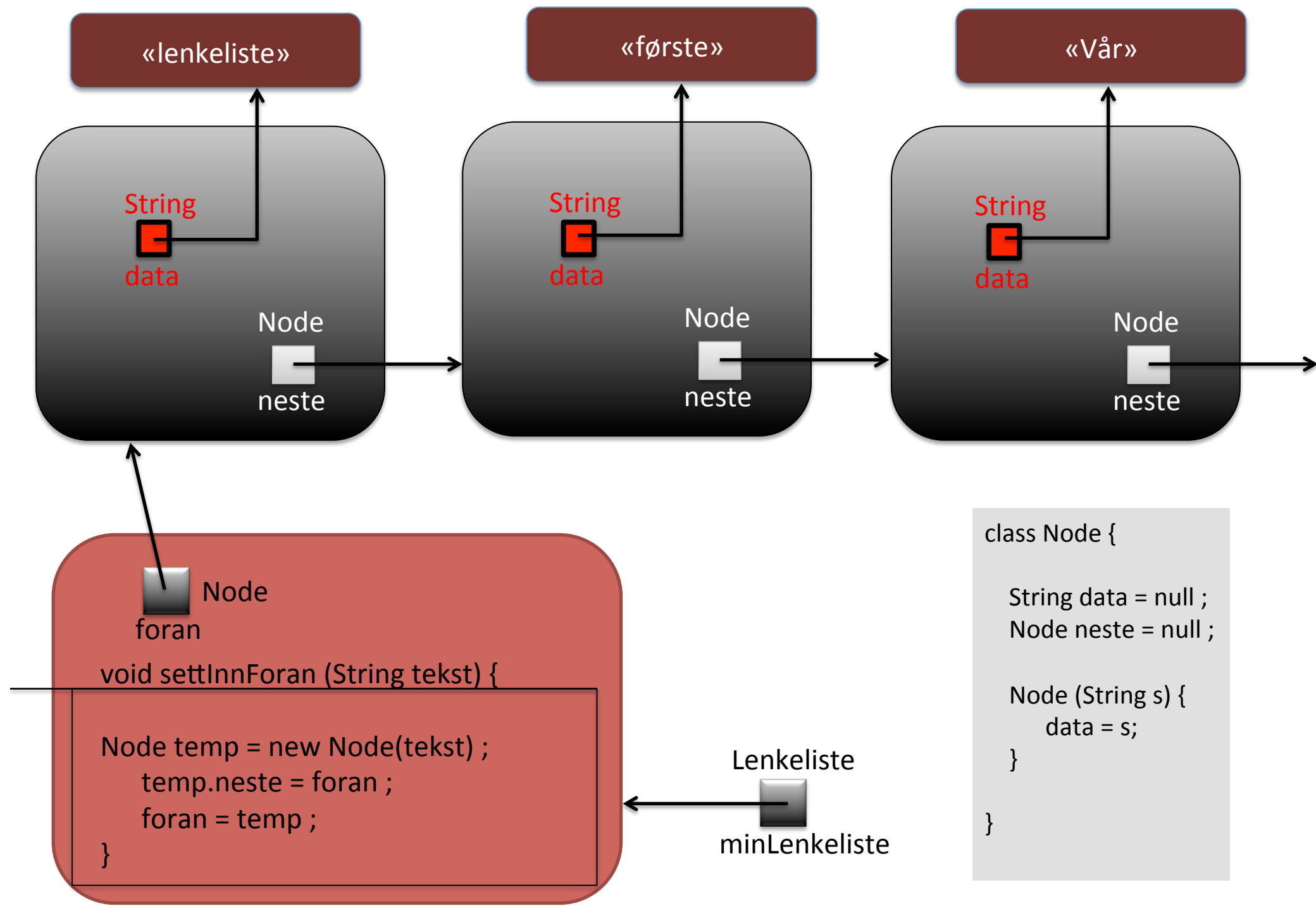

```
Node foran = new Node() ;
foran.neste = new Node() ;
Node temp = new Node() ;
temp.neste = foran ;
foran = temp ;
temp = null ;
```


class Node {
Node neste = null ;
}
Node
neste
Node
neste
Node
neste
Node
foran
void settInnForan()
{ Node temp = new Node() ;
temp.neste = foran ;
foran = temp ; }
class Lenkeliste {
private Node foran = null ;
public void settInnForan() {
Node temp = new Node() ;
temp.neste = foran ;
foran = temp ;
}
}


String
data
Node
neste
String
data
Node
neste
String
data
Node
neste
Node
foran
void settInnForan()
{ Node temp = new Node() ;
temp.neste = foran ;
foran = temp ; }
Lenkeliste
minLenkeliste



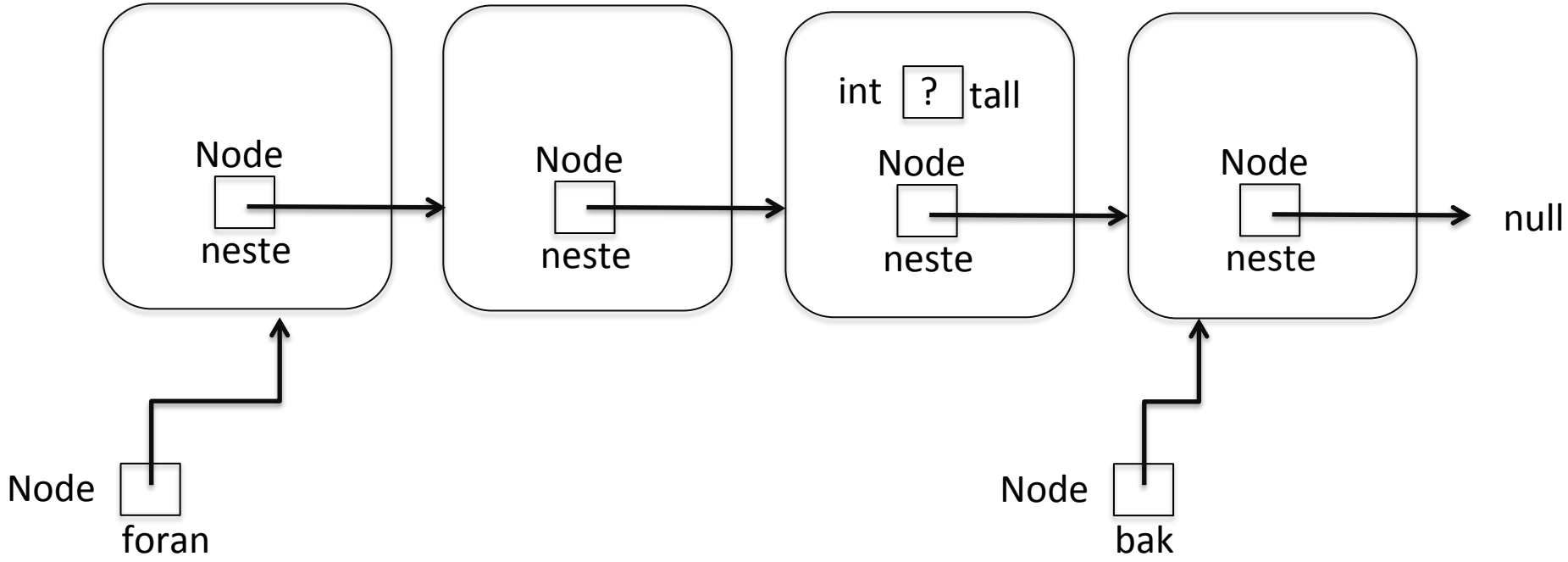

```
class Node {

   String data = null ;
   Node neste = null ;

   Node (String s) {
       data = s;
   }

}
```

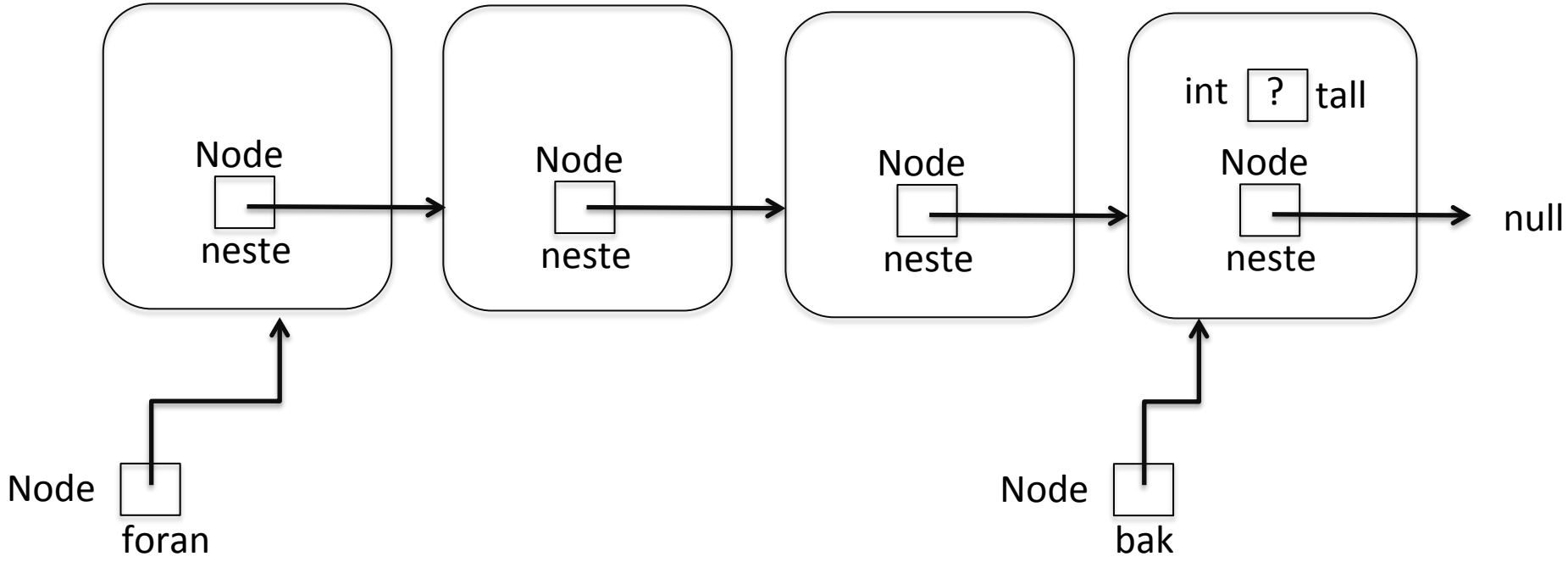

«lenkeliste»
«første»
«Vår»
String
data
Node
neste
String
data
Node
neste
String
data
Node
neste
Node
foran
void settInnForan (String tekst) {
Node temp = new Node(tekst) ;
temp.neste = foran ;
foran = temp ;
}
Lenkeliste
minLenkeliste
class Node {
String data = null ;
Node neste = null ;
Node (String s) {
data = s;
}
}

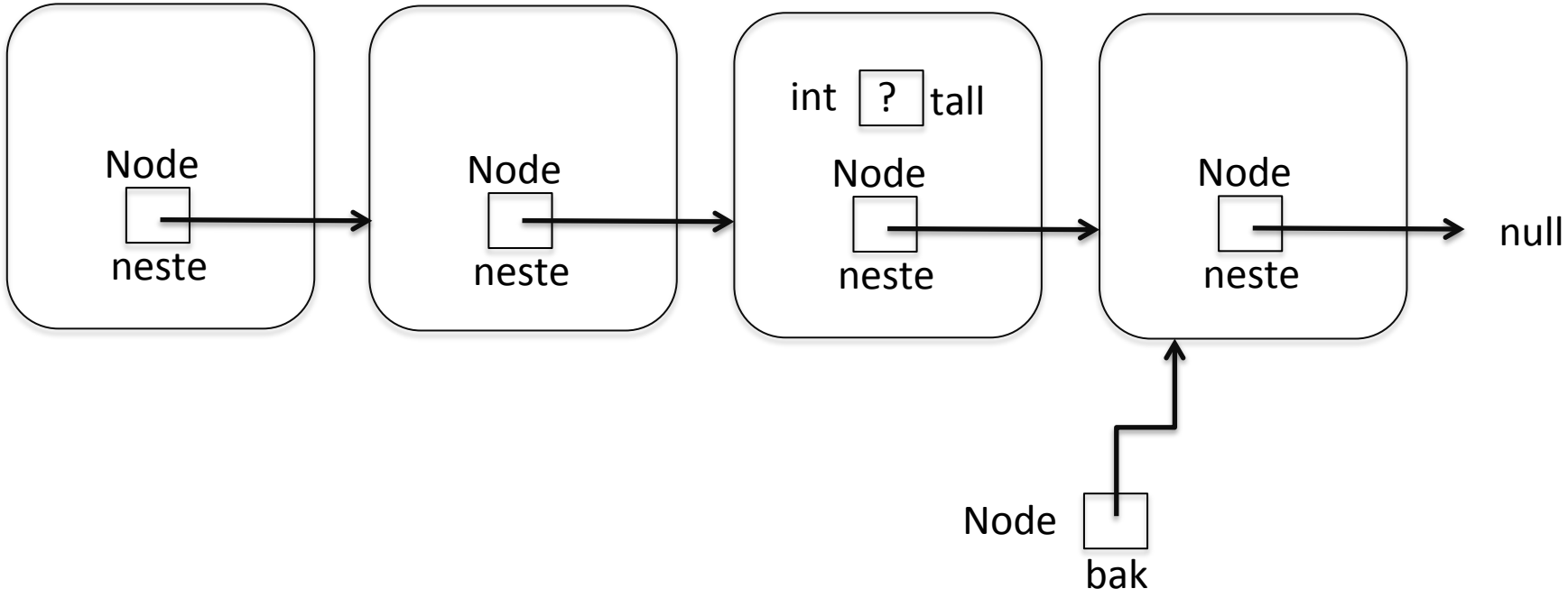
```
class Lenkeliste {

  private Node foran = null ;

  public void settInnForan(String tekst) {
     Node temp = new Node(tekst) ;
     temp.neste = foran ;
     foran = temp ;
     System.out.println("Satt inn: " + tekst);
  }

    public void skrivAlle() {
        Node iter = foran;
        while ( iter != null ) {
           System.out.println(iter.data);
           iter = iter.neste;
        }
    }

}
```

```
class Node {
   String data ;
   Node neste = null ;

   Node (String s) {
      data = s;
   }
}

class Eksempel02 {
   public static void main (String[ ] a) {
      Lenkeliste minLenkeliste = new Lenkeliste();
      minLenkeliste.settInnForan("Vår");
      minLenkeliste.settInnForan("første");
      minLenkeliste.settInnForan("lenkeliste");
      minLenkeliste.skrivAlle();
   }
}
```

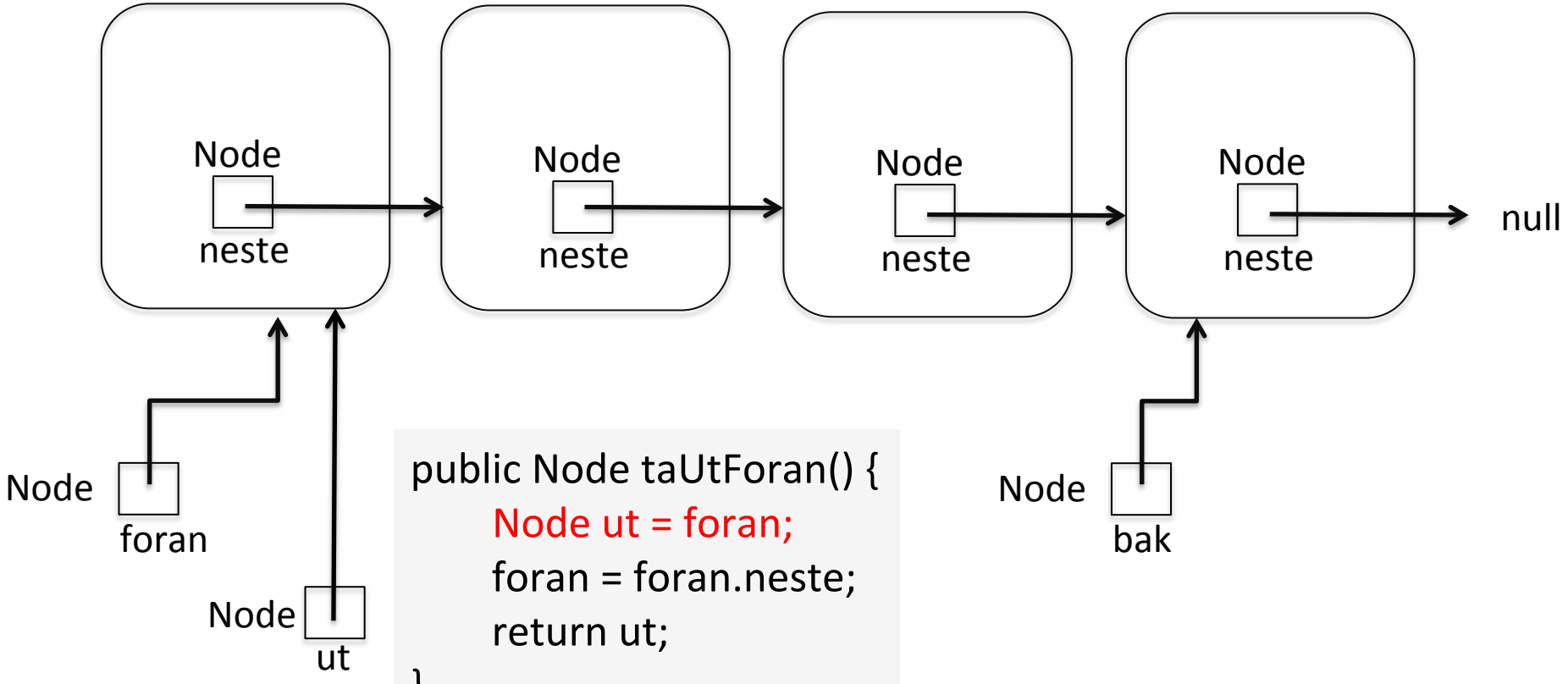
Hvordan får vi tak i en peker til objektet med variabelen tall?
Hvordan får vi tak i innholdet til variabelen tall?




Node
neste
Node
neste
Node
neste
int
?
tall
Node
neste
null
Node
foran
Node
bak

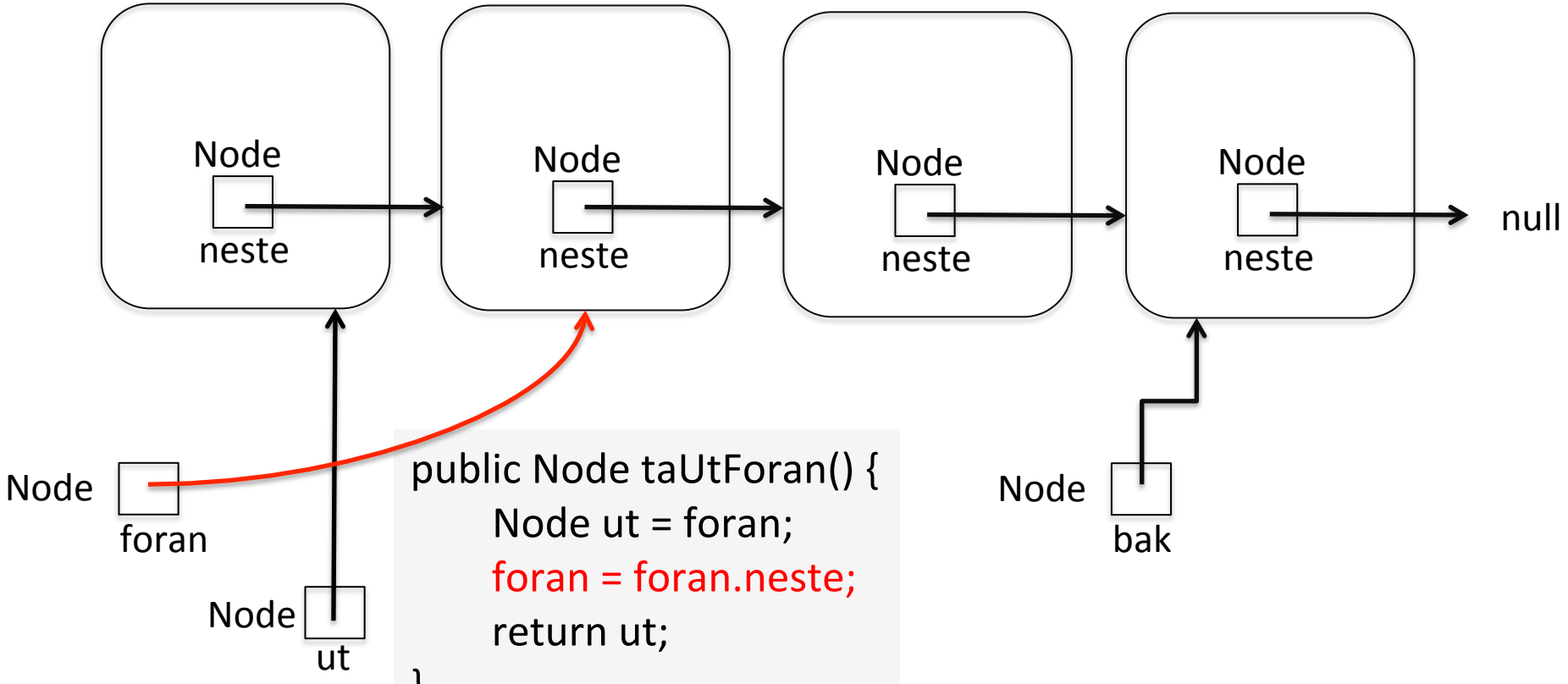

Node
neste
Node
neste
int
?
tall
Node
neste
Node
neste
null
Node
bak

Ta ut en node foran



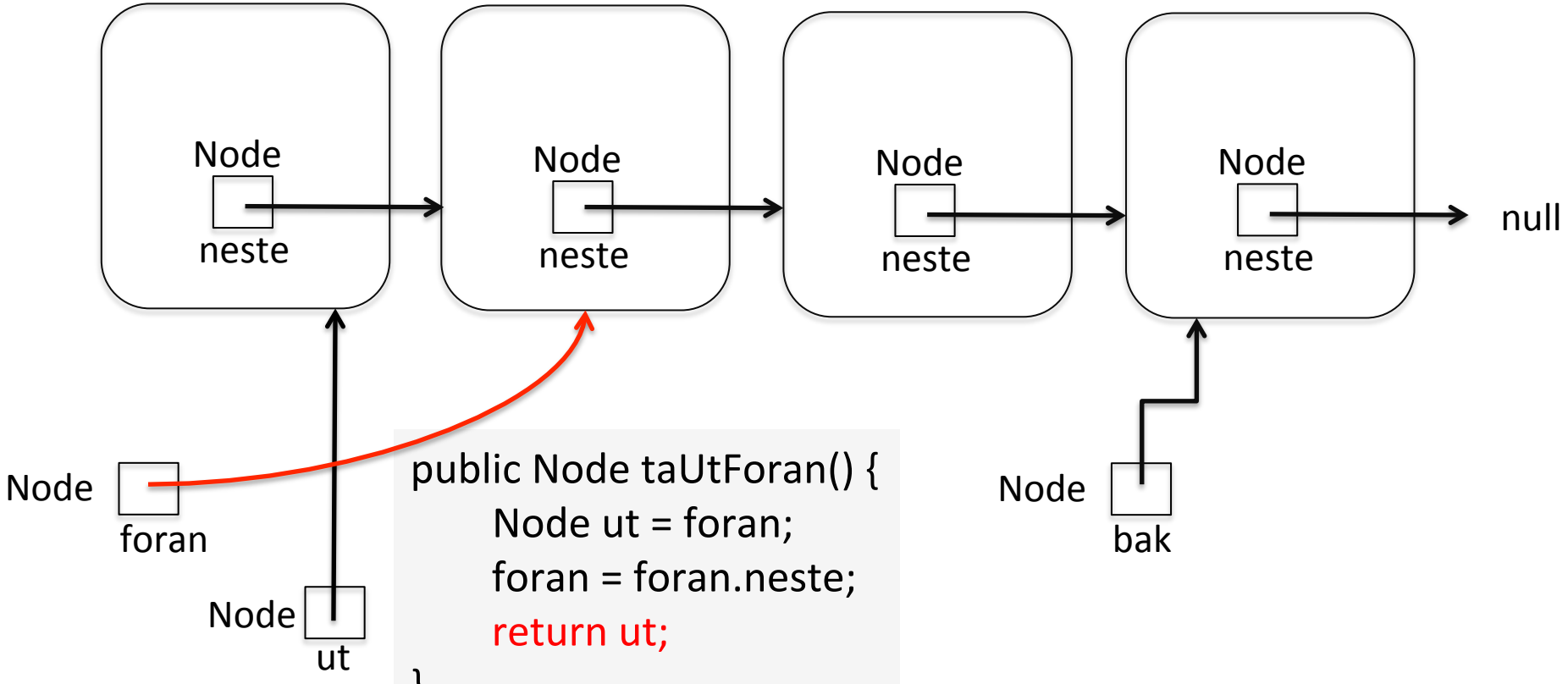

```
public Node taUtForan() {
        Node ut = foran;
        foran = foran.neste;
        return ut;
}
```

Ta ut en node foran



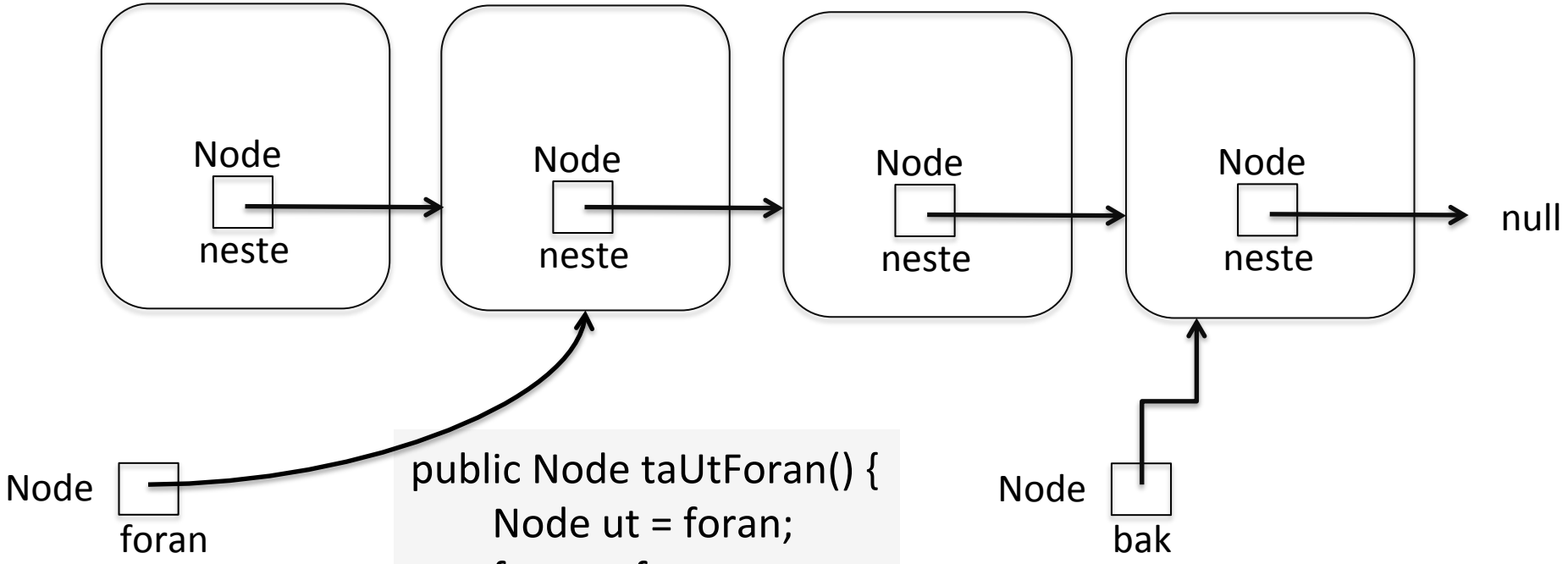

```
public Node taUtForan() {
        Node ut = foran;
        foran = foran.neste;
        return ut;
}
```

Ta ut en node foran



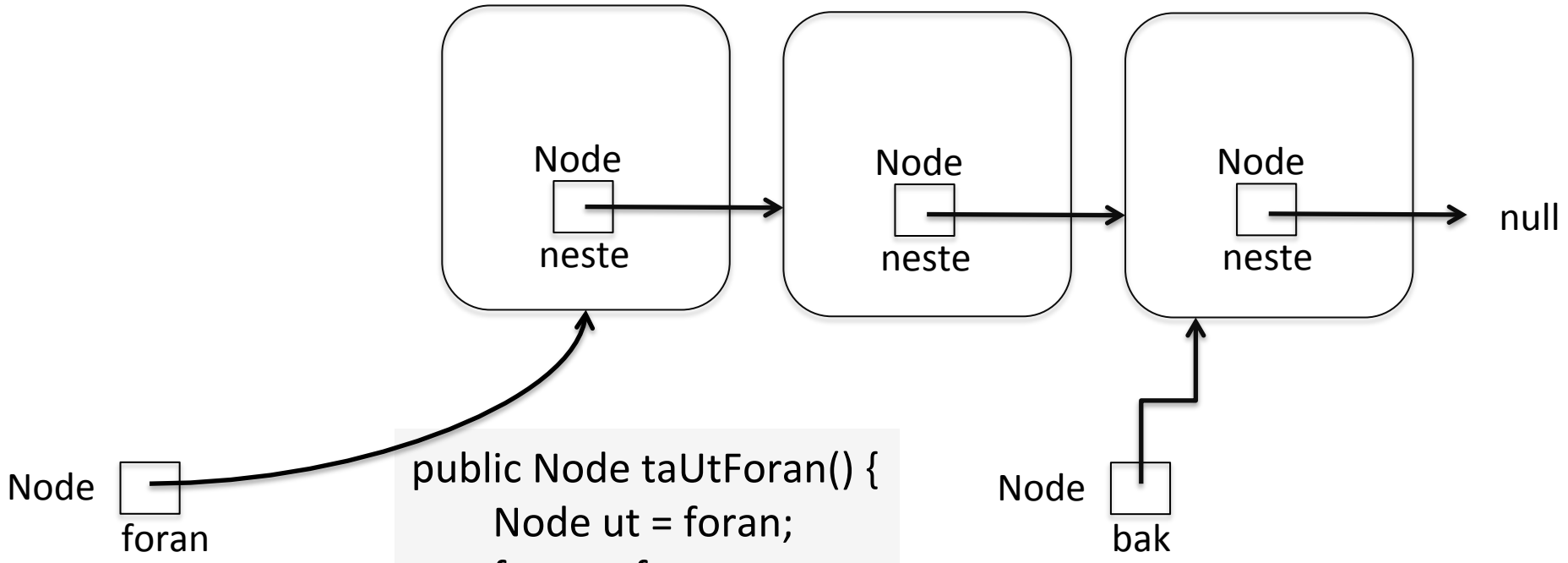

```
public Node taUtForan() {
        Node ut = foran;
        foran = foran.neste;
        return ut;
}
```

Ta ut en node foran




```
public Node taUtForan() {
        Node ut = foran;
        foran = foran.neste;
        return ut;
}
```

Ta ut en node foran




```
public Node taUtForan() {
        Node ut = foran;
        foran = foran.neste;
        return ut;
}
```

Hva må vi kreve av tilstanden i lista før vi kaller taUtForan *?*

Hvilken programstruktur har vi ?

- Et mønster for nodeobjekter med nestepeker og data
- Vi lager nye noder med new Node()
- Pekervariable for å peke ut første og/eller siste objekt i lenkelista
- Metoder for å sette inn og ta ut nodeobjekter
- Og mer trenger vi ikke….

Indre klasse

```
class Lenkeliste {

    private class Node  {
        Node neste ;
        String data ;
    }

    private Node foran = null ;

    public void settInnForan(String tekst) {… }

    public Node taUtForan() { … }

}
```

```
class Main {
    public static void main (String[ ] a) {
        Lenkeliste minLenkeliste = new Lenkeliste();
        minLenkeliste.settInnForan("Vår");
        minLenkeliste.settInnForan("første");
        minLenkeliste.settInnForan("lenkeliste");
    }

}
```

Kun synlig i blokka begrenset av klassens { }

```
class Main {
    public static void main (String[ ] a) {
        Lenkeliste minLenkeliste = new Lenkeliste();
        minLenkeliste.settInnForan("Vår");
        minLenkeliste.settInnForan("første");
        minLenkeliste.settInnForan("lenkeliste");
    }

}
```

```
class Lenkeliste {

    private class Node  {
        Node neste ;
        String data ;
    }

    private Node foran = null ;

    public void settInnForan(String tekst) {… }

    public Node taUtForan() { … }

}
```

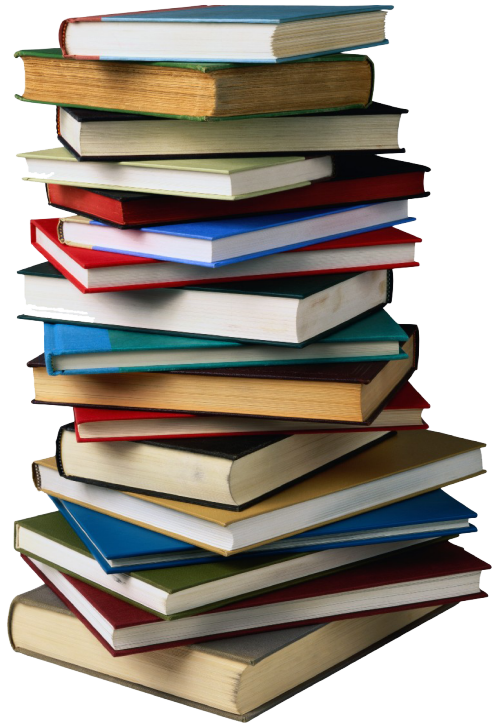
Hva er grense-snittet til Lenkeliste ?

```
class Main {
    public static void main (String[ ] a) {
        Lenkeliste minLenkeliste = new Lenkeliste();
        minLenkeliste.settInnForan("Vår");
        minLenkeliste.settInnForan("første");
        minLenkeliste.settInnForan("lenkeliste");
    }

}
```

```
class Lenkeliste {

    private class Node  {
        Node neste ;
        String data ;
    }

    private Node foran = null ;

    public void settInnForan(String tekst) {… }

    public Node taUtForan() { … }

}
```

Sett inn foran, ta ut foran

- Sist inn først ut
- Last In First Out
- LIFO
- stabel
- stack



Det er metodene som setter inn og tar ut som bestemmer om lenkelista fungerer som en stabel.

Sett inn bak, ta ut foran

- Først inn først ut
- First In First Out
- FIFO
- vanlig kø
- queue

Det er metodene som setter inn og tar ut som bestemmer om lenkelista fungerer som en FIFO-kø.

```
class Lenkeliste {

   private class Node {
      String data ;
      Node neste = null ;

      Node (String s) {
         data = s;
      }
   }

   private Node foran = null ;

   public void settInnForan(String tekst) {
     Node temp = new Node(tekst) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public void skrivAlle() { … }

}
```

```
class Eksempel03 {
   public static void main (String[] a) {
      Lenkeliste minLenkeliste = new Lenkeliste();
      minLenkeliste.settInnForan("lenkeliste");
      minLenkeliste.settInnForan("første");
      minLenkeliste.settInnForan("Vår");
      minLenkeliste.skrivAlle();
   }
}
```

Men her kan vi bare legge inn tekststrenger !?

```
class Lenkeliste {

   private class Node {
      Katt data ;
      Node neste = null ;

      Node (Katt s) {
         data = s;
      }
   }

   private Node foran = null ;

   public void settInnForan(Katt k) {
     Node temp = new Node(k) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public void skrivAlle() { … }

}
```

```
class Eksempel03 {
   public static void main (String[] a) {
      Lenkeliste minLenkeliste = new Lenkeliste();
      minLenkeliste.settInnForan(new Katt("Pus"));
      minLenkeliste.settInnForan(new Katt("Tom"));
      ….
   }
}
```

Ei lenkeliste med katter

```
class Lenkeliste {

   private class Node {
      Object data ;
      Node neste = null ;

      Node (Object o) {
         data = o;
      }
   }

   private Node foran = null ;

   public void settInnForan(Object obj) {
     Node temp = new Node(obj) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public void skrivAlle() { … }

}
```

```
class Eksempel04 {
   public static void main (String[] a) {
      Lenkeliste minLenkeliste = new Lenkeliste();
      minLenkeliste.settInnForan(new Katt("Pus"));
      minLenkeliste.settInnForan(new Hund("Tom"));
      minLenkeliste.settInnForan(new Student("Liv"));
      ….
   }
}
```

Ei lenkeliste med katter, hunder og hvasomhelst …

```
class Lenkeliste < T > {

   private class Node {
      T data ;
      Node neste = null ;

      Node (T ny) {
         data = ny;
      }
   }

   private Node foran = null ;

   public void settInnForan(T  t) {
     Node temp = new Node(t) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public T taUtForan () { … }

}
```

T er parameter til klassen
T står for en type
T kalles klasseparameter
T kalles generisk parameter
T kalles typeparameter (eng. type parameter)
Klassen kalles en generisk klasse

Når vi lager et objekt av en generisk klasse,
må vi bruke en aktuell typeparameter:

```
Lenkeliste<Katt> katter = new Lenkeliste<Katt>();

Lenkeliste<Bil> bilreg = new Lenkeliste<Bil>();
```

I objektet vi da *genererer*, blir alle forekomster av T
erstattet av den aktuelle typeparameteren.

```
class Lenkeliste < T > {

   private class Node {
      T data ;
      Node neste = null ;

      Node (T ny) {
         data = ny;
      }
   }

   private Node foran = null ;

   public void settInnForan(T  t) {
     Node temp = new Node(t) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public T taUtForan () { … }

}
```

```
class Eksempel05 {
   public static void main (String[] a) {
      Lenkeliste<Katt> minLenkeliste = new Lenkeliste<Katt>();
      minLenkeliste.settInnForan(new Katt("Pus"));
      minLenkeliste.settInnForan(new Katt("Tom"));
      minLenkeliste.settInnForan(new Katt("Jerry"));
      minLenkeliste.settInnForan(new Katt("Miss"));
      minLenkeliste.settInnForan(new Katt("Tigergutt"));
   }
}
```

Ei lenkeliste med katter

```
class Lenkeliste < T > {

   private class Node {
      T data ;
      Node neste = null ;

      Node (T ny) {
         data = ny;
      }
   }

   private Node foran = null ;

   public void settInnForan(T  t) {
     Node temp = new Node(t) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public T taUtForan () { … }

}
```

```
class Eksempel05 {
   public static void main (String[] a) {
      Lenkeliste<String> minLenkeliste = new Lenkeliste<String>();
      minLenkeliste.settInnForan("java-jive");
      minLenkeliste.settInnForan("the");
      minLenkeliste.settInnForan("love");
      minLenkeliste.settInnForan("We");
   }
}
```

Ei lenkeliste med tekststrenger

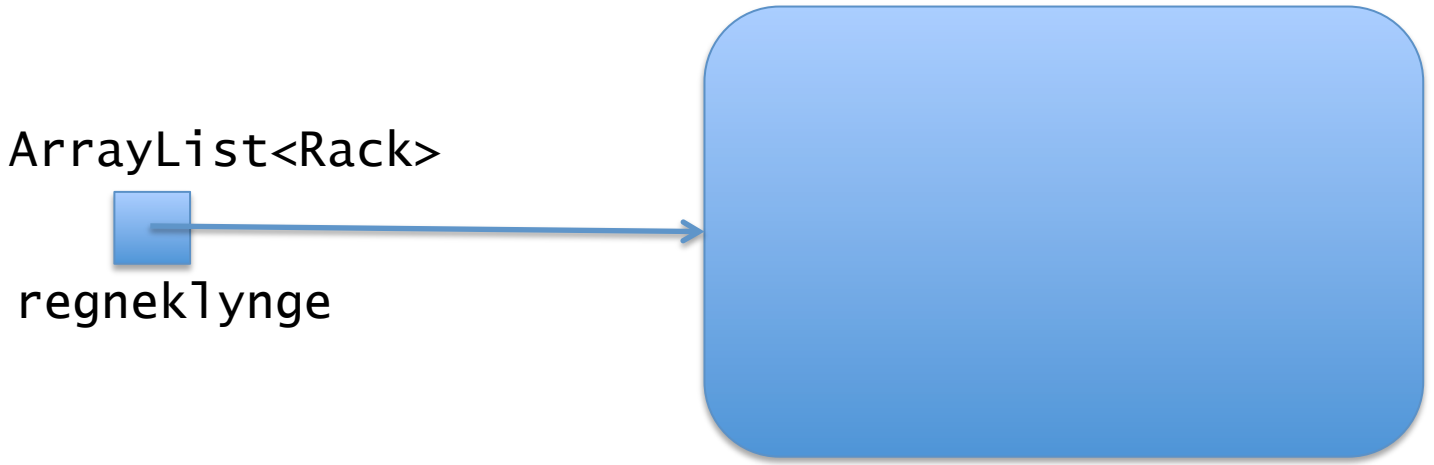
```
class Lenkeliste < T > {

   private class Node {
      T data ;
      Node neste = null ;

      Node (T ny) {
         data = ny;
      }
   }

   private Node foran = null ;

   public void settInnForan(T  t) {
     Node temp = new Node(t) ;
     temp.neste = foran ;
     foran = temp ;
   }

   public T taUtForan () { … }

   public void skrivAlle() { … }
}
```

```
class Eksempel06 {
   public static void main (String[] a) {
      Lenkeliste<Object> minLenkeliste = new Lenkeliste<Object>();
      minLenkeliste.settInnForan("lenkeliste");
      minLenkeliste.settInnForan(new Katt("Pus"));
      minLenkeliste.settInnForan(new Hund("Tom"));
      minLenkeliste.settInnForan(new Student("Liv"));
      minLenkeliste.settInnForan("Vår");
      minLenkeliste.skrivAlle();
   }
}
```

Ei lenkeliste med katter, hunder og hvasomhelst …

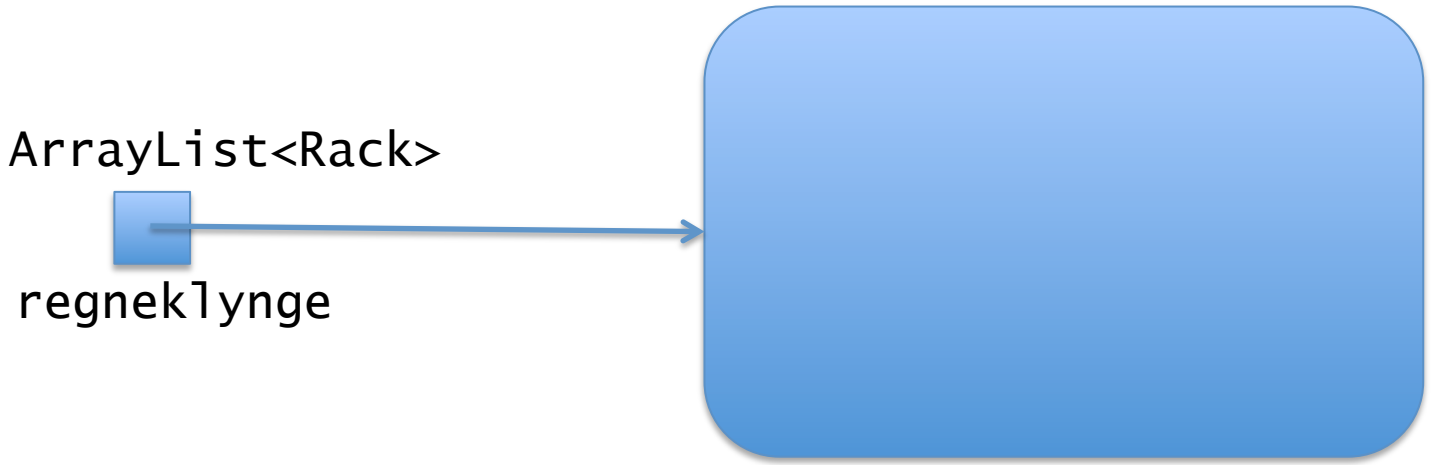
```
class ArrayList<E> {
    ...
    public boolean add(E e) { }
    public E get(int index) { }
    ...
}
```

```
ArrayList<Rack> regneklynge = new ArrayList<Rack>( ) ;
```



Hva er grensesnittet til objektet?

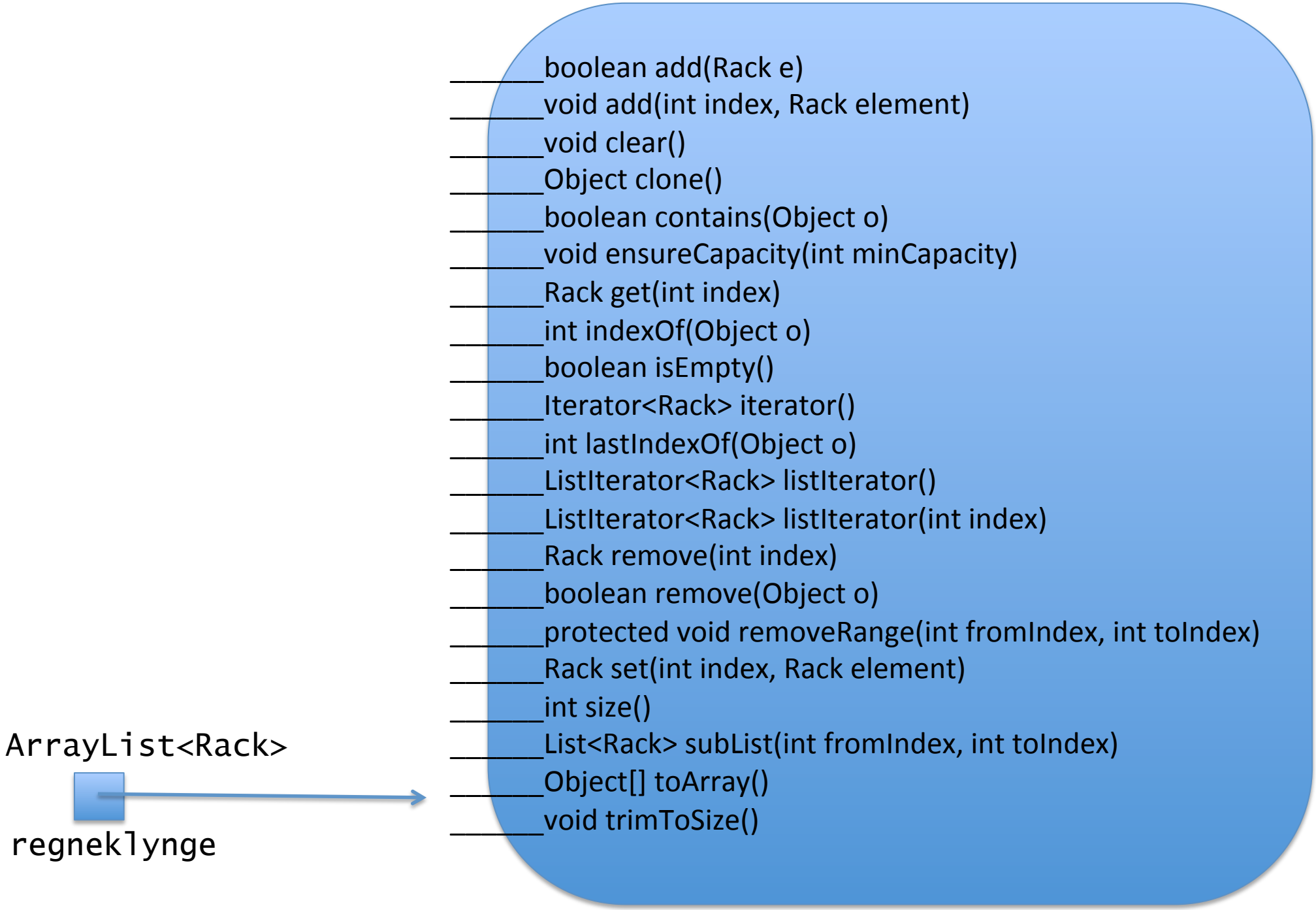
```
ArrayList<Rack> regneklynge = new ArrayList<Rack>( ) ;
```



De fleste av metodene i
grensesnittet til

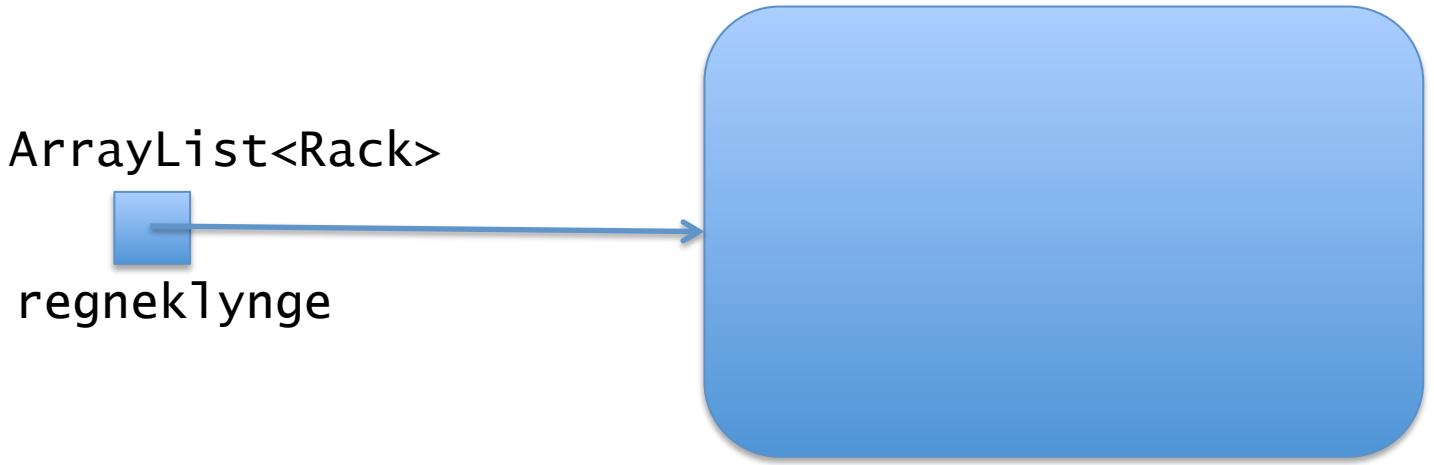
```
class ArrayList<E> {

}
```

```
boolean add(E e)
void add(int index, E element)
void clear()
Object clone()
boolean contains(Object o)
void ensureCapacity(int minCapacity)
E get(int index)
int indexOf(Object o)
boolean isEmpty()
Iterator<E> iterator()
int lastIndexOf(Object o)
ListIterator<E> listIterator()
ListIterator<E> listIterator(int index)
E remove(int index)
boolean remove(Object o)
protected void removeRange(int fromIndex, int toIndex)
E set(int index, E element)
int size()
List<E> subList(int fromIndex, int toIndex)
Object[] toArray()
void trimToSize()
```


_____boolean add(Rack e)
_____void add(int index, Rack element)
_____void clear()
_____Object clone()
_____boolean contains(Object o)
_____void ensureCapacity(int minCapacity)
_____Rack get(int index)
_____int indexOf(Object o)
_____boolean isEmpty()
_____Iterator<Rack> iterator()
_____int lastIndexOf(Object o)
_____ListIterator<Rack> listIterator()
_____ListIterator<Rack> listIterator(int index)
_____Rack remove(int index)
_____boolean remove(Object o)
_____protected void removeRange(int fromIndex, int toIndex)
_____Rack set(int index, Rack element)
_____int size()
_____List<Rack> subList(int fromIndex, int toIndex)
_____Object[] toArray()
_____void trimToSize()
ArrayList<Rack>
regneklynge

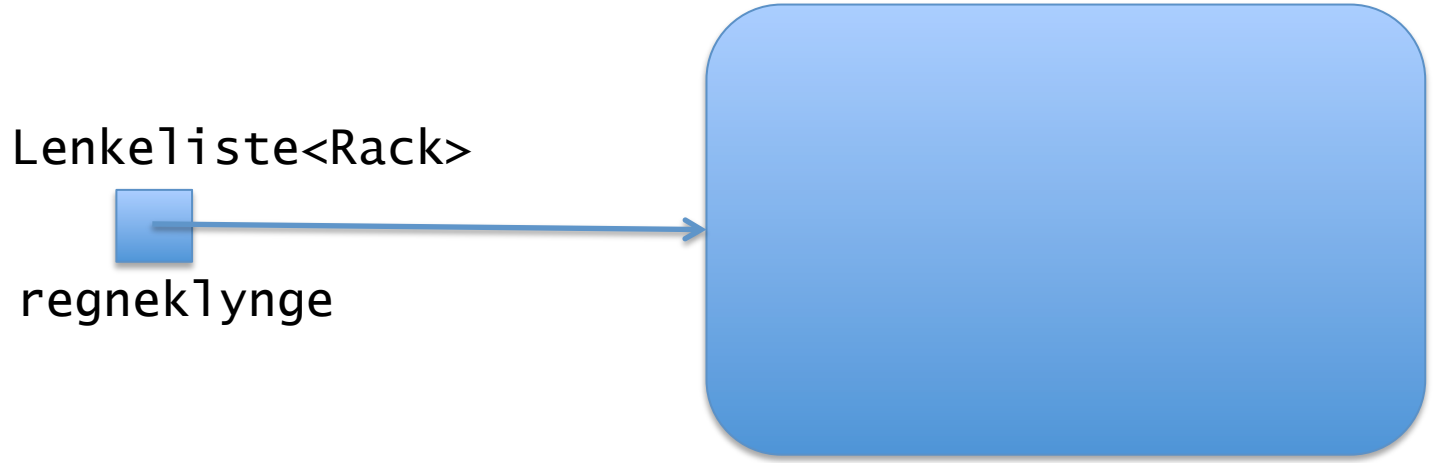
```
class ArrayList<E> {
    ...
    public boolean add(E e) { }
    public E get(int index) { }
    ...
}
```

```
ArrayList<Rack> regneklynge = new ArrayList<Rack>( ) ;
```



```
class Lenkeliste<E> {
    ...
    public void settInnForan(E e) { }
    public E taUtForan( ) { }
    public void skrivAlle( ) { }
    ...
}
```

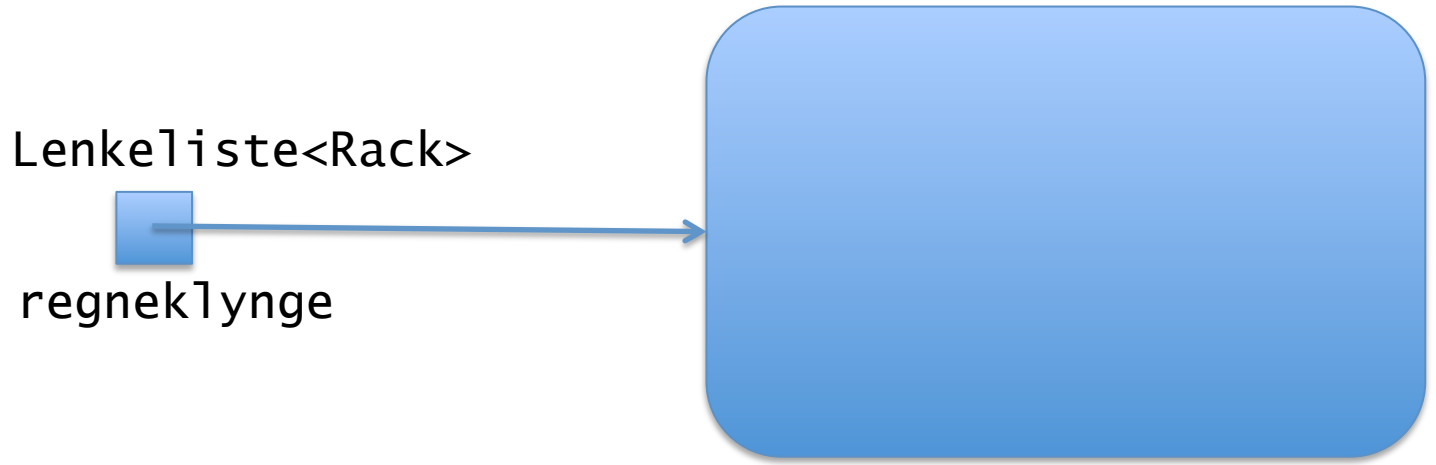
```
Lenkeliste<Rack> regneklynge = new Lenkeliste<Rack>( ) ;
```



```
class Lenkeliste<E> {
    ...
    public void settInnForan(E e) { }
    public E taUtForan( ) { }
    public void skrivAlle( ) { }
    ...
}
```

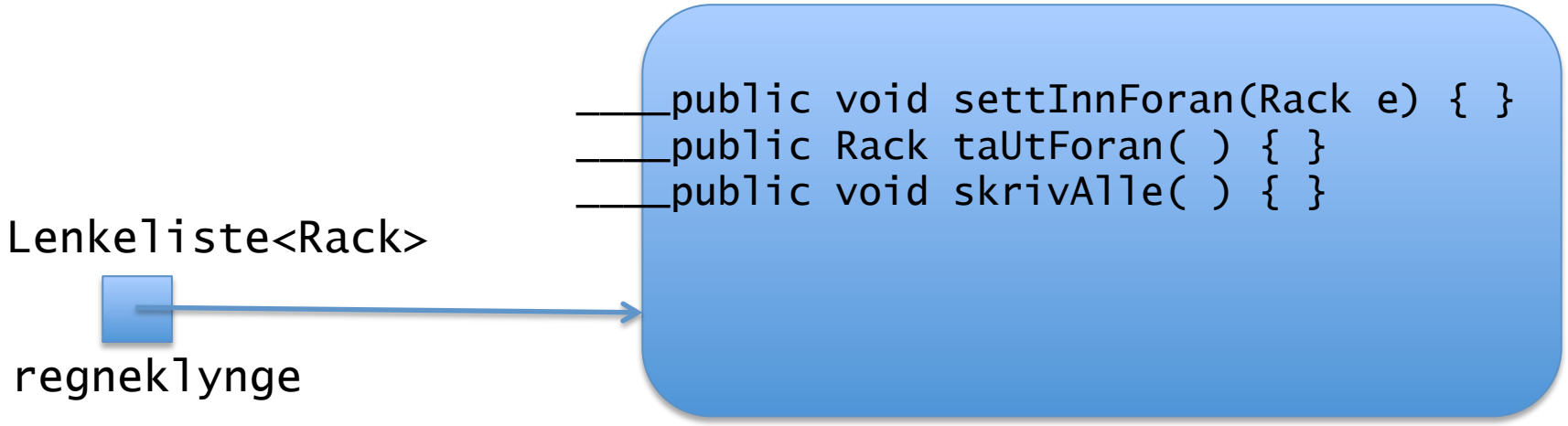
Hva er grensesnittet til objektet?

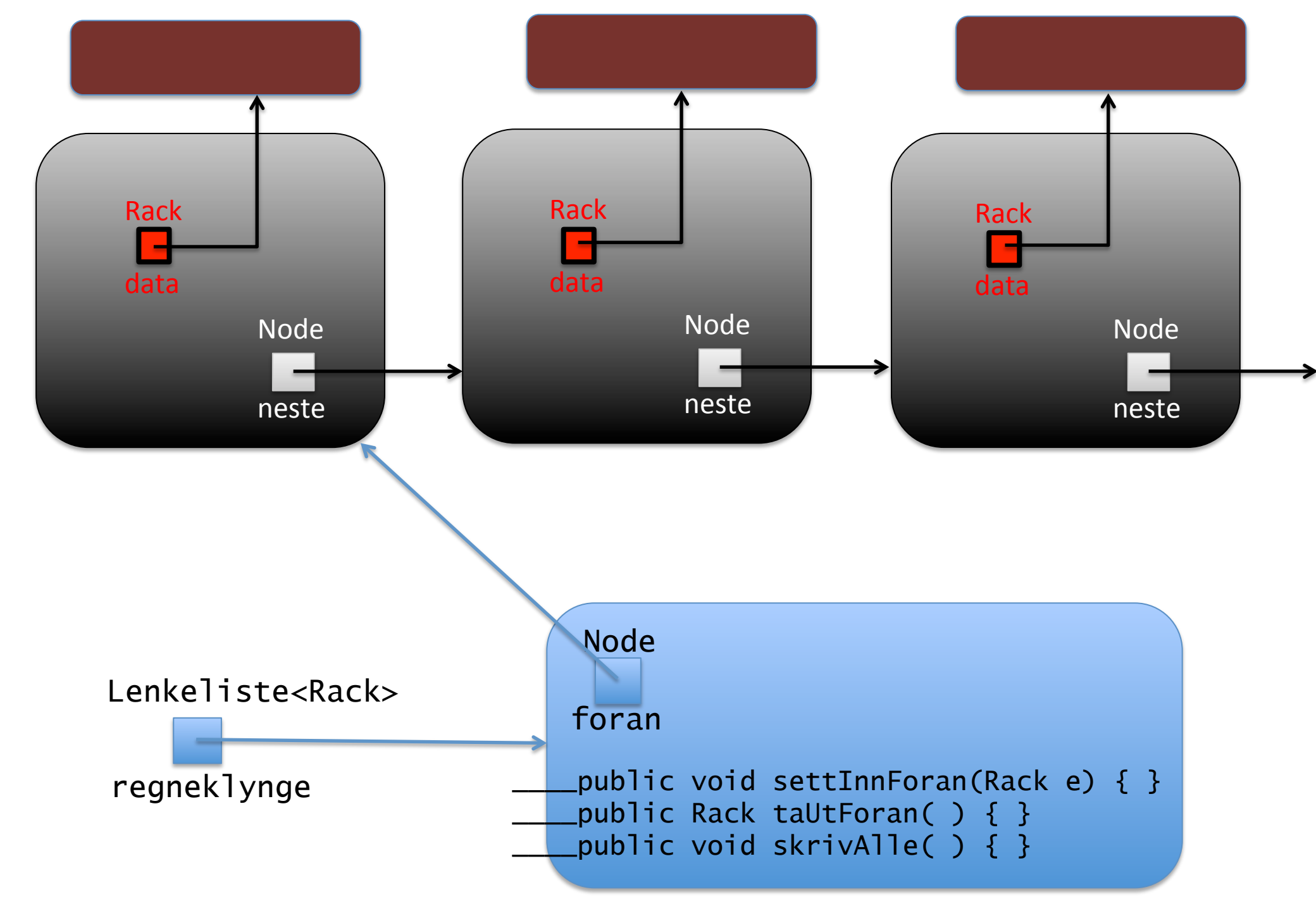
```
Lenkeliste<Rack> regneklynge = new Lenkeliste<Rack>( ) ;
```

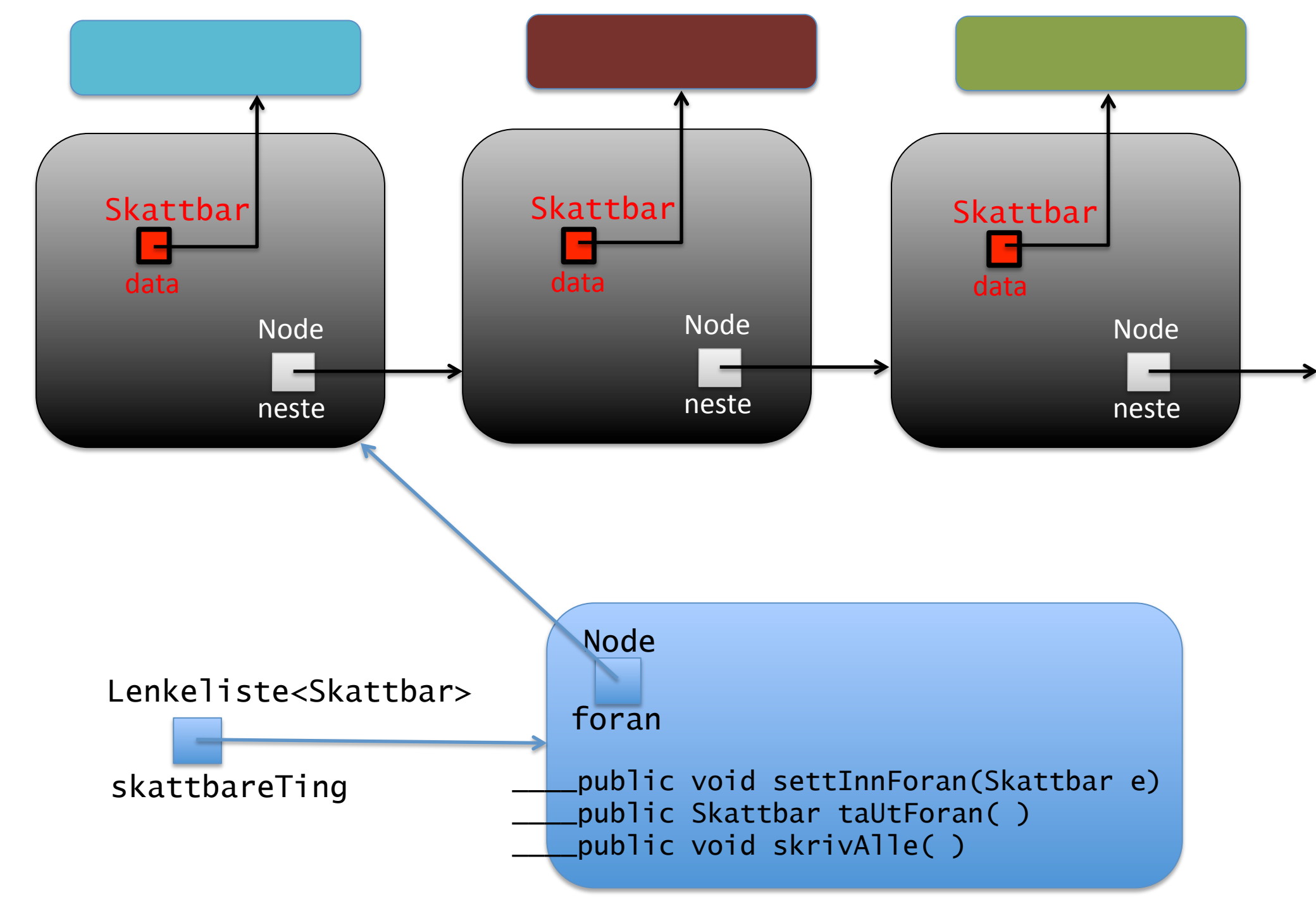


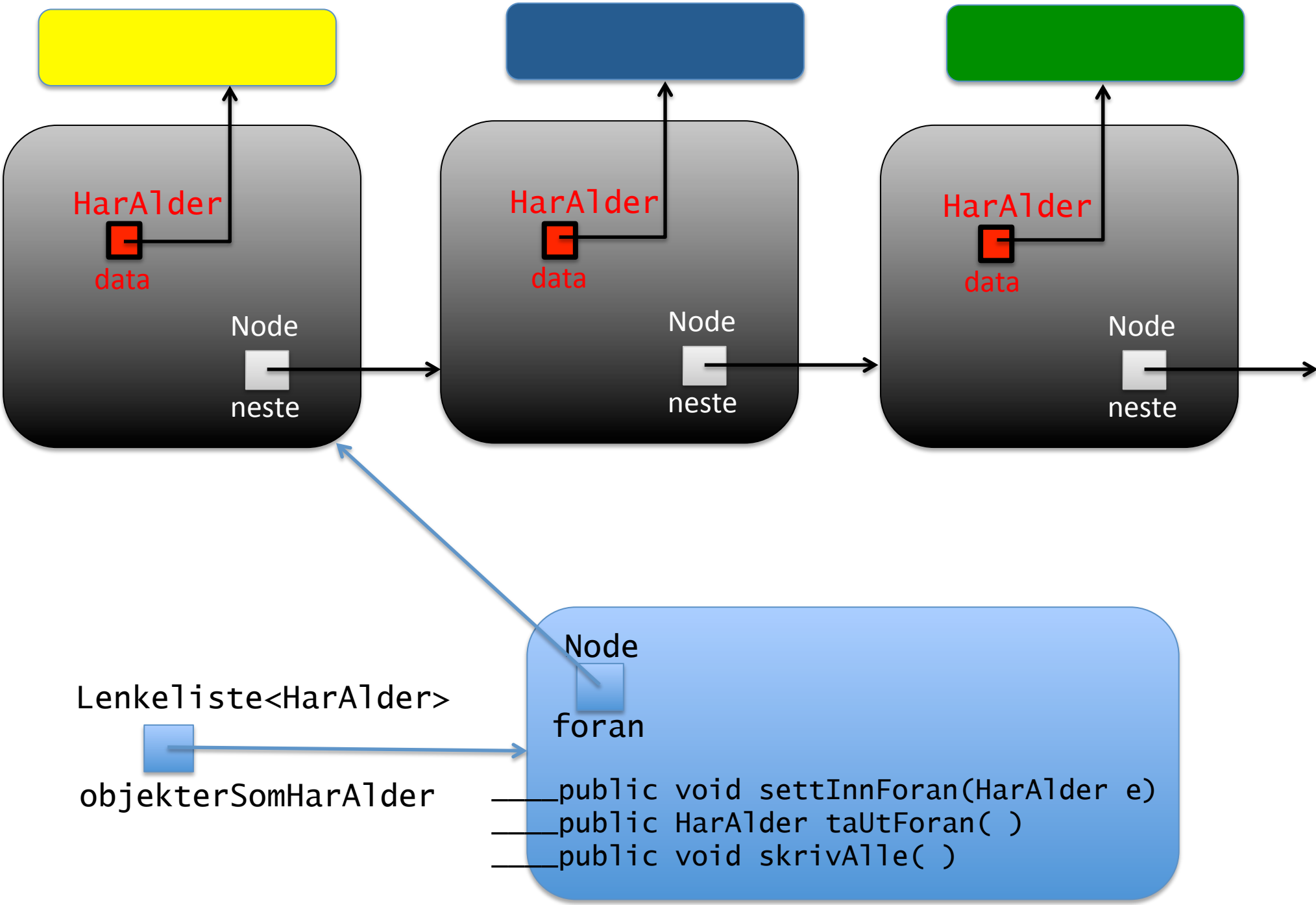
```
class Lenkeliste<E> {
    ...
    public void settInnForan(E e) { }
    public E taUtForan( ) { }
    public void skrivAlle( ) { }
    ...
}
```

```
Lenkeliste<Rack> regneklynge = new Lenkeliste<Rack>( ) ;
```



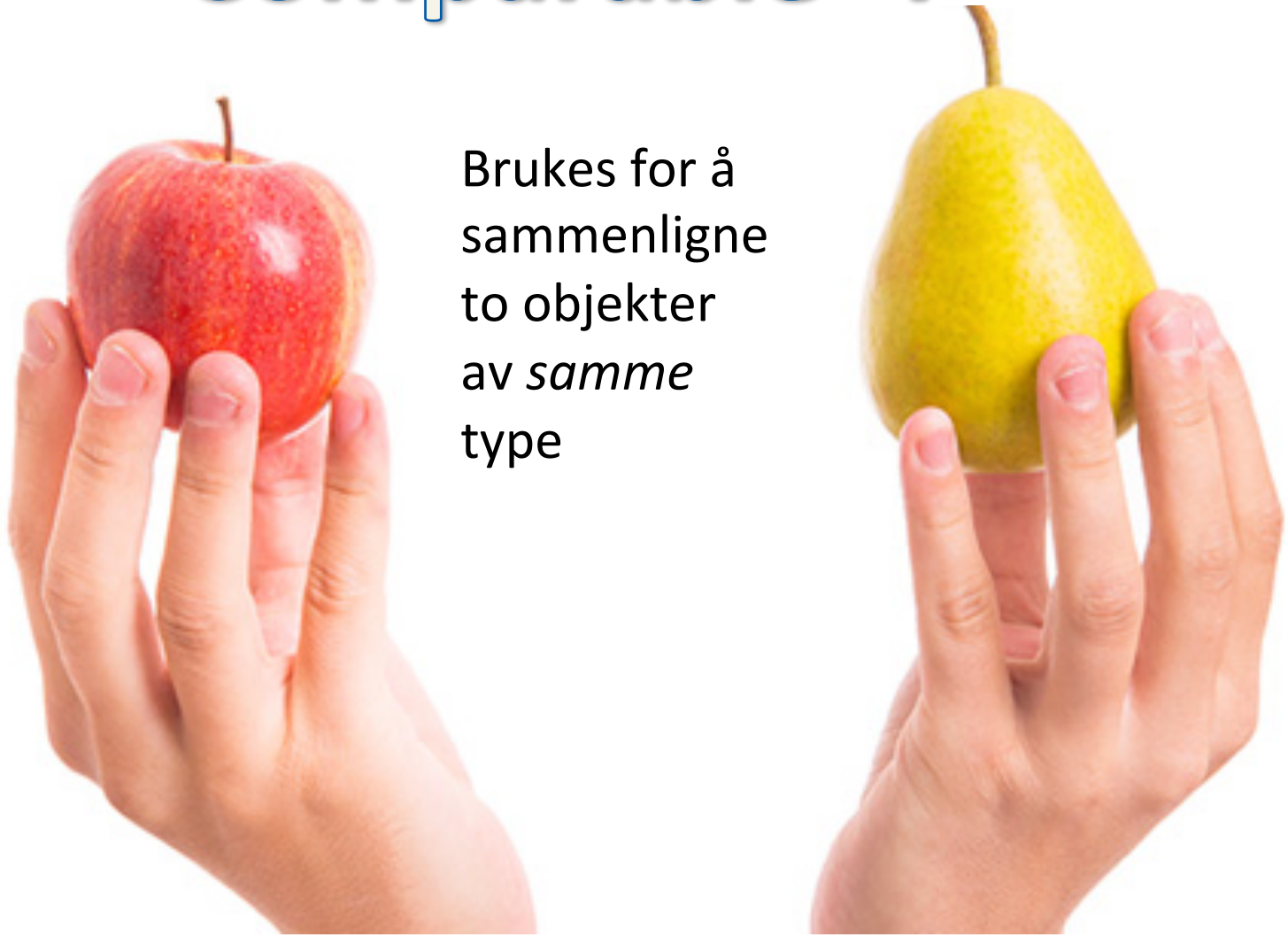

Rack
data
Node
neste
Rack
data
Node
neste
Rack
data
Node
neste
Node
foran
public void settInnForan(Rack e) { }
public Rack taUtForan( ) { }
public void skrivAlle( ) { }
Lenkeliste<Rack>
regneklynge


Skattbar
data
Node
neste
Skattbar
data
Node
neste
Skattbar
data
Node
neste
Lenkeliste<Skattbar>
skattbareTing
Node
foran
public void settInnForan(Skattbar e)
public Skattbar taUtForan( )
public void skrivAlle( )


HarAlder
data
Node
neste
HarAlder
data
Node
neste
HarAlder
data
Node
neste
Lenkeliste<HarAlder>
objekterSomHarAlder
Node
foran
public void settInnForan(HarAlder e)
public HarAlder taUtForan( )
public void skrivAlle( )

Grensesnittet Comparable<T>

Grensesnittet Comparable<T>



Brukes for å sammenligne to objekter av *samme* type

Grensesnittet Comparable<T>

java.lang

Interface Comparable<T>

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

Method Summary

Methods

Modifier and Type	Method and Description
`int`	**`compareTo(T o)`** Compares this object with the specified object for order.

```
class Person implements Comparable<Person> {

    private long fødselsnr;

    Person ( long f ) {
          fødselsnr = f;
    }

    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int compareTo ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen

       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen

       else        return  0;         // this == enAnnen
    }
}
```

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

Implemenatsjonen av `compareTo` *bestemmer hvordan vi sammenligner personobjekter. Denne implementasjonen gjør at den med det største (som tallverdi) fødselsnr. er 'større enn' de andre.*

```
class Person implements Comparable<Person> {
    private long fødselsnr;
    Person ( long f ) {
          fødselsnr = f;
    }

    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int compareTo ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}

class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);

     Integer i = new Integer(121278);
     System.out.println(i.toString() + "Govi");

     if (en.compareTo(to) > 0) System.out.println("en > to");
     else System.out.println("to <= en");

    }
}
```

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

Implemenatsjonen av `compareTo` *bestemmer hvordan vi sammenligner personobjekter. Denne implementasjonen gjør at den med det største (som tallverdi) fødselsnr. er 'større enn' de andre.*

```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) {
          fødselsnr = f;
    }

    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;          // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;         // this < enAnnen
       else        return  0;         // this == enAnnen
    }
}

class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);

     Integer i = new Integer(121278);
     System.out.println(i.toString() + "Govi");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("to <= en");

    }
}
```

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

```
interface Sammenlignbar<E> {
   int sammenlignMed(E e);
}
```

```
class Person implements Sammenlignbar<Person> {
    private long fødselsnr;
    Person ( long f ) { fødselsnr = f; }
    public long fnr () { return fødselsnr; }

    public boolean equals ( Person enAnnen ) {
         return fødselsnr == enAnnen.fnr();
    }

    public int sammenlignMed ( Person enAnnen ) {
       if      (fødselsnr - enAnnen.fnr() > 0 )
                   return 1;         // this > enAnnen
       else if (fødselsnr - enAnnen.fnr() < 0 )
                   return -1;        // this < enAnnen
       else        return  0;        // this == enAnnen
    }
}
```

```
interface Sammenlignbar<E> {
   int sammenlignMed(E e);
}
```

```
class Sammenlign {

    public static void main (String[] a) {
     Person en = new Person(12345678911L);
     Person to = new Person(12345678922L);
     Sammenlignbar<Person> sb1 = new Person(12345678913L);
     Sammenlignbar<Person> sb2 = new Person(12345678911L);

     Long lg = 123456789L;
     System.out.println(lg.toString() + "inf1010");

     if (en.sammenlignMed(to) > 0) System.out.println("en > to");
     else System.out.println("to <= en");

     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(sb2) );
     System.out.println( sb1.sammenlignMed(en) > sb2.sammenlignMed(to) );
     System.out.println( sb2.equals(sb1) );
    }
}
```

Oppgave til seminartimene

Programmer en klasse `MinMaksSil` med en begrensende typeparameter
`T extends Comparable<T>`.

Silen skal ta vare på to objekter av typen `T`, det største og det minste.

Klassen skal ha to offentlige metoder:

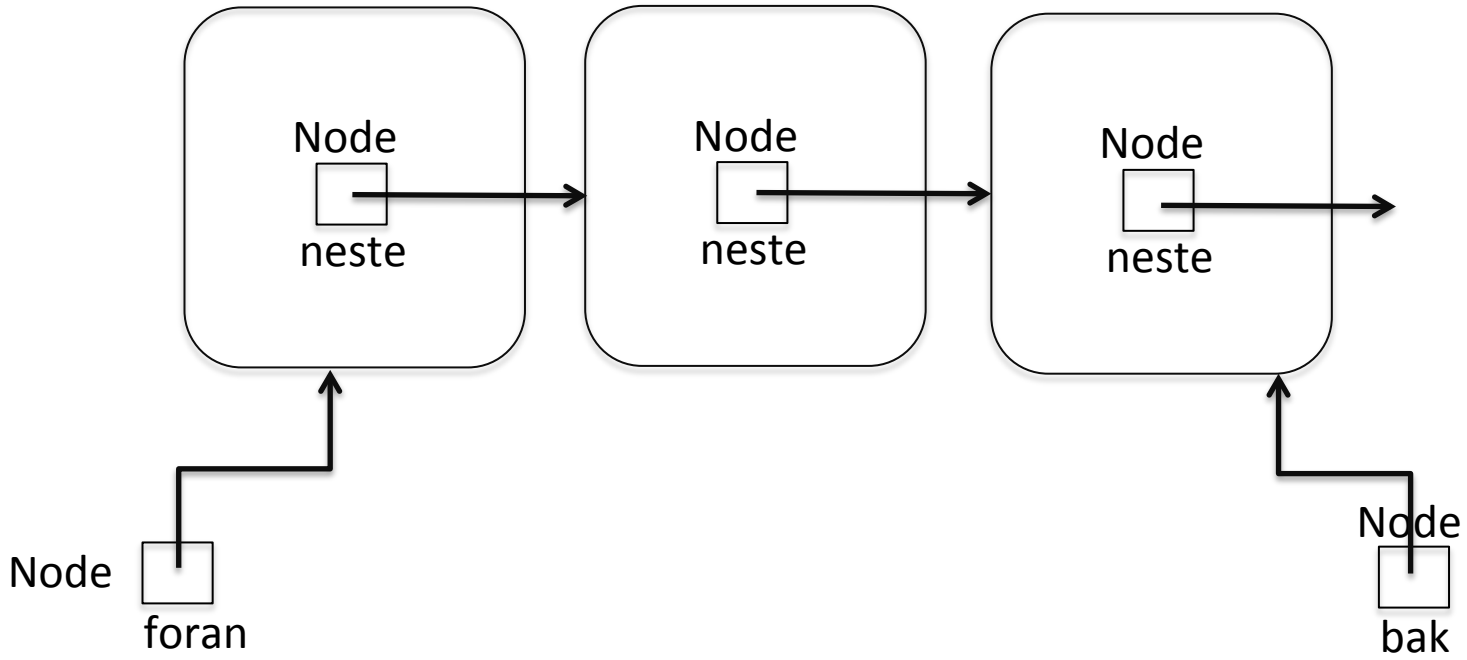
1. For å sile (legge inn objekter) bruker vi metoden `sil`.

2. For å skrive ut resultatet når alle objektene har vært gjennom silen, kan vi kalle på metoden `skriv`.

a) Skriv programmet og test det ved å bruke `String` som aktuell typeparameter.

b) Skriv en egen klasse som implementerer `Comparable<E>` (der `E` er navnet på den nye klassen), og test at silen også fungerer for objekter av denne hjemmesnekrede typen.

Ta ut en node bak



Datastruktur i lenkelisteobjektet

- enkelt- eller dobbeltlenket (to pekere)
- listehode- og hale
  - *slipper å særbehandle tilfellene:*
  - lista er tom
  - ett objekt, dvs. foran == bak
- intern ordning av objektene (sortering)
  - hvis mange: letter gjenfinning
  - vanskeligere å skrive metodene
  - lettere å gjøre feil

- tenk **tilstandspåstander** før og etter metodekall