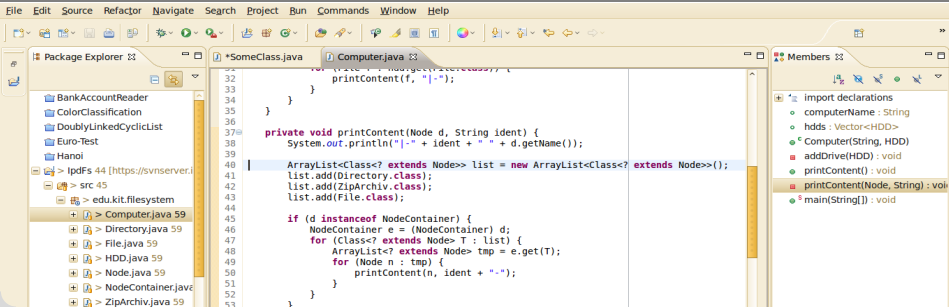


Programmieren-Tutorium Nr. 10 bei Martin Thoma

String interning, Assertions, Einfach verkettete Listen

Martin Thoma | 3. Dezember 2012

FAKULTÄT FÜR INFORMATIK



The screenshot shows an IDE with the following components:

- Package Explorer:** Lists project files including `BankAccountReader`, `ColorClassification`, `DoublyLinkedCyclicList`, `Euro-Test`, `Hanoi`, `src` (containing `edu.kit.filesystem`), and `Computer.java` (selected).
- Editor:** Displays the code for `Computer.java`. The code includes a `printContent` method and a `main` method. A line of code is highlighted: `ArrayList<Class? extends Node> list = new ArrayList<Class? extends Node>();`.
- Members:** Shows the class members for `Computer`, including `import declarations`, `computerName : String`, `hdds : Vector<HDD>`, `Computer(String, HDD)`, `addDrive(HDD) : void`, `printContent() : void`, `printContent(Node, String) : void`, and `main(String[]) : void`.

- 1 Einleitung
- 2 Assertions
- 3 Einfach verkettete Listen
- 4 Abspann

```
1 public class QuizString {
2     public static void main(String[] args) {
3         String string1 = new String("Hallo");
4         String string2 = new String("Hallo");
5         if (string1 == string2) {
6             System.out.println("string1 und string2 sind das selbe Objekt.");
7         } else {
8             System.out.println("string1 und string2 sind verschiedene Objekte.");
9         }
10
11         String string3 = "Hallo";
12         String string4 = "Hallo";
13         if (string3 == string4) {
14             System.out.println("string3 und string4 sind das selbe Objekt.");
15         } else {
16             System.out.println("string3 und string4 sind verschiedene Objekte.");
17         }
18     }
19 }
```

- Gibt es einen Compiler-Fehler? ✗
- Gibt es einen Laufzeit-Fehler? ✗
- Gibt es eine Ausgabe? ✓ Welche Ausgabe gibt es?

```
1 public class QuizString {
2     public static void main(String[] args) {
3         String string1 = new String("Hallo");
4         String string2 = new String("Hallo");
5         if (string1 == string2) {
6             System.out.println("string1 und string2 sind das selbe Objekt.");
7         } else {
8             System.out.println("string1 und string2 sind verschiedene Objekte.");
9         }
10
11         String string3 = "Hallo";
12         String string4 = "Hallo";
13         if (string3 == string4) {
14             System.out.println("string3 und string4 sind das selbe Objekt.");
15         } else {
16             System.out.println("string3 und string4 sind verschiedene Objekte.");
17         }
18     }
19 }
```

- string1 und string2 sind verschiedene Objekte.
- string3 und string4 sind das selbe Objekt.

- Erstellt man einen String mit `String abc = new String("Hallo");` wird ein neues Objekt angelegt
- Erstellt man einen String mit `String abc = "Hallo";` macht Java „String interning“

Achtung

Trotzdem mit `abc.equals(def);` vergleichen! Nur so ist garantiert, dass ihr auf Gleichheit (und nicht nur auf „Selbstheit“ vergleicht).

- Erstellt man einen String mit `String abc = new String("Hallo");` wird ein neues Objekt angelegt
- Erstellt man einen String mit `String abc = "Hallo";` macht Java „String interning“

Achtung

Trotzdem mit `abc.equals(def);` vergleichen! Nur so ist garantiert, dass ihr auf Gleichheit (und nicht nur auf „Selbstheit“ vergleicht).

- Erstellt man einen String mit `String abc = new String("Hallo");` wird ein neues Objekt angelegt
- Erstellt man einen String mit `String abc = "Hallo";` macht Java „String interning“

Achtung

Trotzdem mit `abc.equals(def);` vergleichen! Nur so ist garantiert, dass ihr auf Gleichheit (und nicht nur auf „Selbstheit“ vergleicht).

- Problem: Es tritt ein falsches Ergebnis auf, es ist aber nicht klar warum.
- Lösung: Man macht Zusicherungen (engl. assertions)
- Man überlegt sich also, welche Variablen an kritischen Stellen welche Werte oder Beziehungen zueinander haben sollen

Wichtig: Assertions sind keine Exceptions!

Assertion	Exception
muss man aktivieren dient zum Entdecken von Fehlern z.B. $(a < b)$, $(a \neq 0)$, ...	wird immer ausgeführt dient zum behandeln von Fehlern z.B. <code>IOException</code>

- Problem: Es tritt ein falsches Ergebnis auf, es ist aber nicht klar warum.
- Lösung: Man macht Zusicherungen (engl. assertions)
- Man überlegt sich also, welche Variablen an kritischen Stellen welche Werte oder Beziehungen zueinander haben sollen

Wichtig: Assertions sind keine Exceptions!

Assertion	Exception
muss man aktivieren dient zum Entdecken von Fehlern z.B. $(a < b)$, $(a \neq 0)$, ...	wird immer ausgeführt dient zum behandeln von Fehlern z.B. <code>IOException</code>

- Problem: Es tritt ein falsches Ergebnis auf, es ist aber nicht klar warum.
- Lösung: Man macht Zusicherungen (engl. assertions)
- Man überlegt sich also, welche Variablen an kritischen Stellen welche Werte oder Beziehungen zueinander haben sollen

Wichtig: Assertions sind keine Exceptions!

Assertion

muss man aktivieren
dient zum Entdecken von Fehlern
z.B. $(a < b)$, $(a \neq 0)$, ...

Exception

wird immer ausgeführt
dient zum behandeln von Fehlern
z.B. `IOException`


- Problem: Es tritt ein falsches Ergebnis auf, es ist aber nicht klar warum.
- Lösung: Man macht Zusicherungen (engl. assertions)
- Man überlegt sich also, welche Variablen an kritischen Stellen welche Werte oder Beziehungen zueinander haben sollen

Wichtig: Assertions sind keine Exceptions!

Assertion	Exception
muss man aktivieren dient zum Entdecken von Fehlern z.B. $(a < b)$, $(a \neq 0)$, ...	wird immer ausgeführt dient zum behandeln von Fehlern z.B. <code>IOException</code>

```
for (int i = 0; i < image.length; i++) {  
    for (int j = 0; j < image[i].length; j++) {  
        assert(0 <= image[i][j] && image[i][j] <= 255);  
        histogram[image[i][j]]++;  
    }  
}
```

In Eclipse:

- 
- Default VM Arguments: „-enableassertions“ hinzufügen

- docs.oracle.com: [Programming With Assertions](#)
- galileo openbook: [Java ist auch eine Insel](#)
- Java Blog Buch: [06.09 Assertions](#)

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

Szenario

- Ihr wollt euch Druckaufträge speichern
- Funktioniert mit Array
- Problem:
 - Ihr belegt immer konstant viel Speicher
 - Eventuell braucht ihr mehr, eventuell weniger Speicher
- Lösung: Verkettete Listen

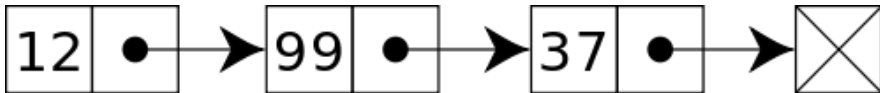
- Man speichert sich nur einen Zeiger
- Dieser Zeiger zeigt auf „Knoten“
- Jeder Knoten hat wieder einen Zeiger
- Jeder Knoten kann wieder auf einen Knoten zeigen

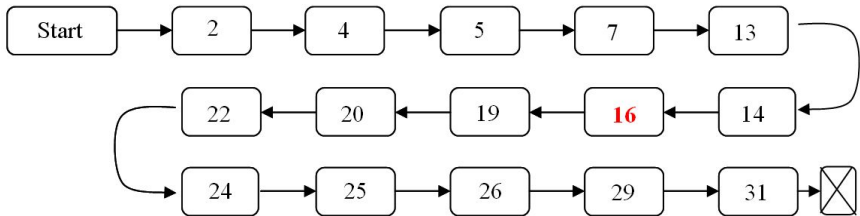
- Man speichert sich nur einen Zeiger
- Dieser Zeiger zeigt auf „Knoten“
- Jeder Knoten hat wieder einen Zeiger
- Jeder Knoten kann wieder auf einen Knoten zeigen

- Man speichert sich nur einen Zeiger
- Dieser Zeiger zeigt auf „Knoten“
- Jeder Knoten hat wieder einen Zeiger
- Jeder Knoten kann wieder auf einen Knoten zeigen

- Man speichert sich nur einen Zeiger
- Dieser Zeiger zeigt auf „Knoten“
- Jeder Knoten hat wieder einen Zeiger
- Jeder Knoten kann wieder auf einen Knoten zeigen

- Man speichert sich nur einen Zeiger
- Dieser Zeiger zeigt auf „Knoten“
- Jeder Knoten hat wieder einen Zeiger
- Jeder Knoten kann wieder auf einen Knoten zeigen





- Elemente hinzufügen
- Elemente löschen
- Elemente finden

Wichtig

Zwischenergebnisse ausgeben

Was wollen wir?

- Elemente hinzufügen
- Elemente löschen
- Elemente finden

Wichtig

Zwischenergebnisse ausgeben

- Elemente hinzufügen
- Elemente löschen
- Elemente finden

Wichtig

Zwischenergebnisse ausgeben

- Elemente hinzufügen
- Elemente löschen
- Elemente finden

Wichtig

Zwischenergebnisse ausgeben

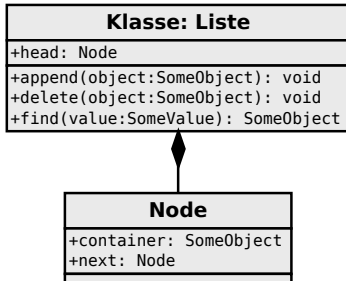
Wie sieht das aus?

Main.java

```
public class Main {  
    public static void main(String[] args) {  
        SinglyLinkedList list = new SinglyLinkedList();  
        list.printList();  
  
        // append new elements at the front of the list  
        list.append(12);  
        list.printList();  
        list.append(13);  
        list.printList();  
        list.append(14);  
        list.printList();  
        list.append(15);  
        list.printList();  
  
        // remove elements  
        list.remove(13);  
        list.printList();  
  
        // find elements  
        int numberA = list.find(14);  
        int numberB = list.find(13);  
        System.out.println(numberA + " " + numberB);  
    }  
}
```

Welche Klassen brauchen wir?

Welche Klassen brauchen wir?



Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

Hinweis 2

Oder - z.B. bei den Abschlusssaufgaben - man verwendet einfach Datentypen aus `java.util`:

- `LinkedList`
- `HashMap` / `TreeMap`
- `HashSet` / `TreeSet`

Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

Hinweis 2

Oder - z.B. bei den Abschlusssaufgaben - man verwendet einfach Datentypen aus `java.util`:

- `LinkedList`
- `HashMap` / `TreeMap`
- `HashSet` / `TreeSet`

Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

Hinweis 2

Oder - z.B. bei den Abschlusssaufgaben - man verwendet einfach Datentypen aus `java.util`:

- `LinkedList`
- `HashMap` / `TreeMap`
- `HashSet` / `TreeSet`

Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

Hinweis 2

Oder - z.B. bei den Abschlusssaufgaben - man verwendet einfach Datentypen aus [java.util](#):

- [LinkedList](#)
- [HashMap](#) / [TreeMap](#)
- [HashSet](#) / [TreeSet](#)

Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

Hinweis 2

Oder - z.B. bei den Abschlussaufgaben - man verwendet einfach Datentypen aus [java.util](#):

- [LinkedList](#)
- [HashMap](#) / [TreeMap](#)
- [HashSet](#) / [TreeSet](#)

Hinweis

- Noch erstellt ihr eine Liste für genau einen Datentyp
- Eigentlich macht der Code immer das gleiche, ist also vom Datentypen unabhängig
- Das löst man später mit „Generics“

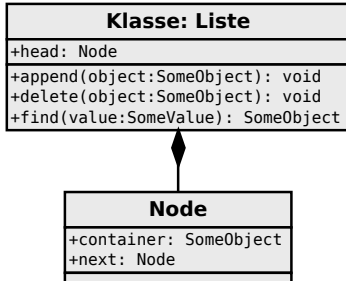
Hinweis 2

Oder - z.B. bei den Abschlusssaufgaben - man verwendet einfach Datentypen aus [java.util](#):

- [LinkedList](#)
- [HashMap](#) / [TreeMap](#)
- [HashSet](#) / [TreeSet](#)

Teil 1

Erstelle die Klasse Node

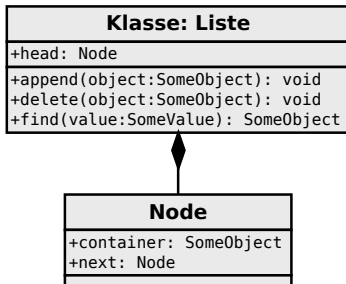


Node.java

```
1 public class Node {  
2     int container;  
3     Node next;  
4  
5     public Node(int container) {  
6         this.container = container;  
7     }  
8 }
```

Teil 2.1

Erstelle die Klasse SinglyLinkedList (noch ohne Funktionalität)



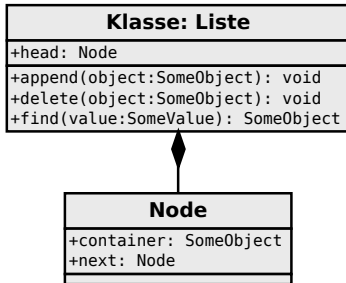
Teil 2.1: Die Struktur der Liste

```
1 public class SinglyLinkedList {
2
3     Node head;
4
5     public void append(int number) {
6
7     }
8
9     public void remove(int number) {
10
11    }
12
13    public int find(int number) {
14
15    }
16
17    public void printList() {
18
19    }
20 }
```

Teil 2.2: printList()

Teil 2.2

Erstelle die Methode „printList“

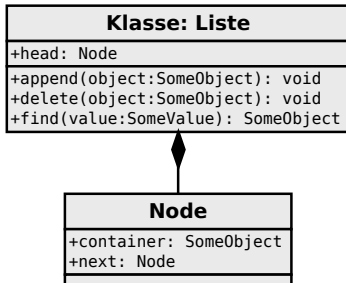


Teil 2.2: printList()

```
60     public void printList() {  
61         Node currentNode = head;  
62         System.out.print("head -> ");  
63         while (currentNode != null) {  
64             System.out.print(currentNode.container + " -> ");  
65             currentNode = currentNode.next;  
66         }  
67         System.out.println("null");  
68     }  
69 }
```

Teil 2.3

Erstelle die Methoden `boolean isEqual(Node node, int content)` und `Node findNode(int number)`



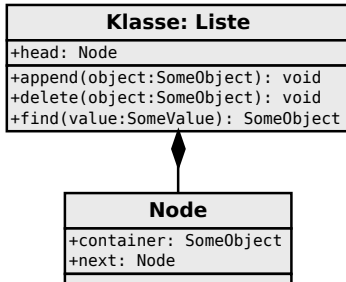
Teil 2.3: Hilfsmethoden

```
5     private boolean isEqual(Node node, int content) {  
6         return node.container == content;  
7     }  
8  
9     private Node findNode(int number) {  
10        Node currentNode = head;  
11  
12        while (!isEqual(currentNode, number) && currentNode.next != null) {  
13            currentNode = currentNode.next;  
14        }  
15  
16        if (isEqual(currentNode, number)) {  
17            return currentNode;  
18        } else {  
19            return null;  
20        }  
21    }
```

Teil 2.4: append

Teil 2.4

Erstelle die Methode `append`



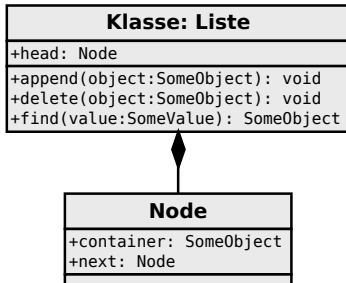
Teil 2.4: append

```
23 public void append(int number) {
24     Node toInsert = new Node(number);
25     toInsert.next = head;
26     head = toInsert;
27
28     /*
29      * schlecht: Node currentNode = head;
30      *
31      * while (currentNode.next != null) { currentNode = currentNode.next; }
32      * currentNode.next = n;
33      */
34 }
```

Teil 2.5: remove

Teil 2.5

Erstelle die Methode `remove`

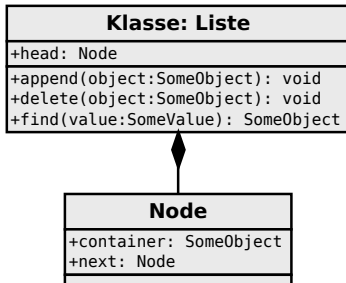


Teil 2.5: remove

```
36 public void remove(int number) {
37     Node previous = head;
38     Node currentNode = head;
39
40     while (!isEqual(currentNode, number) && currentNode.next != null) {
41         previous = currentNode;
42         currentNode = currentNode.next;
43     }
44
45     if (currentNode.next != null) {
46         previous.next = currentNode.next;
47     }
48 }
```

Teil 2.6

Erstelle die Methode `find`



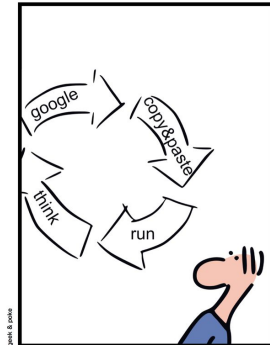
Teil 2.6: find

```
50     public int find(int number) {  
51         Node node = findNode(number);  
52  
53         if (node == null) {  
54             return 0;  
55         } else {  
56             return node.container;  
57         }  
58     }
```

7. 03.12.2012: JUnit-Tests, [toString?](#), Vererbung?
6. 10.12.2012: Generics?
5. 17.12.2012: Video „Library“ zeigen
 - 24.12.2012: Heiligabend - [Kein Tutorium](#)
 - 31.12.2012: Silvester - Kein Tutorium
4. 07.01.2013
3. 14.01.2013
2. 21.01.2013
1. 28.01.2013: Abschlussprüfunsvorbereitung
0. 04.02.2013: Abschlussprüfunsvorbereitung
 - 10.02.2013: Ende der Vorlesungszeit des WS 2012/2013 ([Quelle](#))

Vielen Dank für eure Aufmerksamkeit!

SIMPLY EXPLAINED



gert & pake

DEVELOPMENT CYCLE