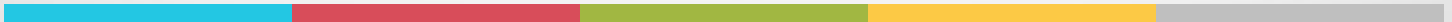
ADT: Verkettete Listen



Abstrakter Datentyp - Definition

```
public class Bruch{
 int zaehler, nenner;

 public Bruch(int zaehler, int nenner)
 {
    this.zaehler = zaehler;
    this.nenner  = nenner;
 }

 public void show() {
   System.out.println(zaehler + "/" + nenner);
 }
}
```

Konstruktor zum Initialisieren der Werte

Ausgabe des Bruches

Abstrakter Datentyp - Definition

- ✓ Zusammenfassung von Daten und Operationen über diesen Daten
- ✓ Implementierung der Datenstrukturen und Operationen ist außerhalb des Objektes nicht sichtbar
- ✓ Zugriff auf Daten nur über Operationen (get, set und andere)

Vorteile:

- ✓ Klar definierte Schnittstellen unterstützen arbeitsteilige Programmentwicklung
- ✓ Implementierungsdetails nur innerhalb des Objektes sichtbar
  -> Implementierung kann ohne Anpassung der übrigen Programmteile geändert werden (bei gleicher Signatur)
- ✓ Erleichtert Debugging, da Fehler besser lokalisierbar

Motivation I

```
public class Bruch{
 int zaehler, nenner;

 public Bruch(int zaehler, int nenner)
 {
    this.zaehler = zaehler;
    this.nenner  = nenner;
 }

 public void show() {
   System.out.println(zaehler + "/" + nenner);
 }
}
```

Konstruktor zum Initialisieren der Werte

Ausgabe des Bruches

Motivation II

```
public class List{
   private Bruch[] liste;
   private int count;

   public List()    {
      liste = new Bruch[10];
      count = 0;
   }
   ……..……..
}
```

Array von Bruchinstanzen (objekten)

Anzahl der Brüche

Initialisierung im Konstruktor:
- Max. 10 Brüche
- 0 Brüche aktuell

Motivation III

```
public void fill(){
    liste[0] = new Bruch(3,4);
    liste[1] = new Bruch(2,7);
    count = 2;
}
public void show(){
   for (int i=0; i < count; i++)
      liste[i].show();
}
public void showAdress(){
   for (int i=0; i < 10; i++)
      System.out.println(liste[i]);
}
}
```

Erstellen von zwei(willkürlich) neuen Brüchen

Ausgabe aller Brüche

Ausgabe aller Adressen der Instanzen der Bruchklasse:

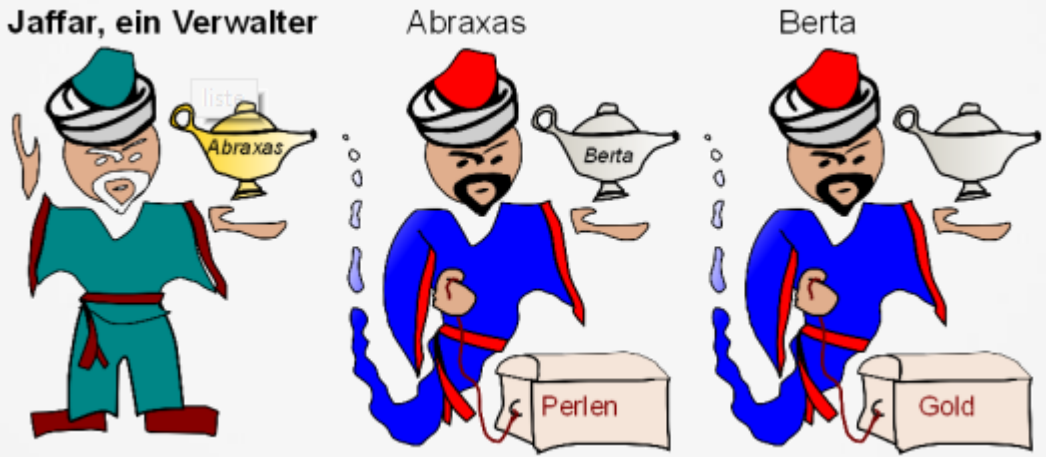
```
Bruch@55d29f62
Bruch@51ed1100
null
null
null
null
null
null
null
null
```

Objekte werden willkürlich im Speicher abgelegt!

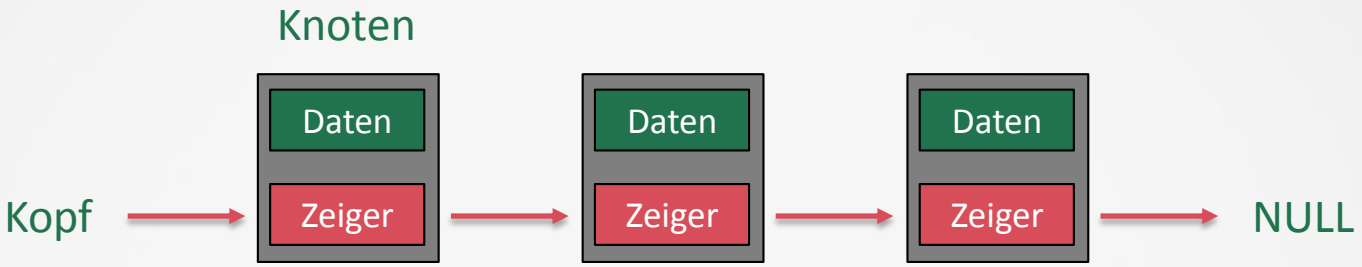
Einfache und doppelt verkettete Liste

- ✓ Folge von Elementen
- ✓ Spezialfall: **leere** Liste
- ✓ Länge der Liste: **Anzahl** der Elemente
- ✓ Elemente können **doppelt** vorkommen
- ✓ Zugriff über **Index**
- ✓ Jedes Element besitzt eine Referenz auf seinen Nachfolger:
  Einfach verkettete Liste
- ✓ Jedes Element besitzt eine Referenz auf seinen Nachfolger
  und seinen Vorgänger: **Doppelt** verkettete Liste

Motivation Es war einmal....

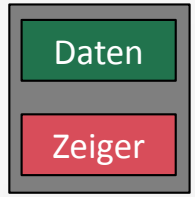


Darstellung: Einfach verkettete Liste



bel. Inhalt möglich!

```
public class Node {
   private Object content;
   private Node nextNode;
}
```



Gleicher Datentyp wie Klasse= Verweis auf nächstes Objekt der Klasse!

Klasse Node I

```
public Node(Object content, Node nextNode){
  this.content = content;
  this.nextNode = nextNode;
}
public Node getNextNode(){
  return nextNode;
}
```

Setzen des Inhaltes und dem Nachfolger

Liefert den Nachfolger zurück

Jede Klasse, auch die Klasse Knoten, ist Unterklasse von Object. Dies bedeutet, dass sich der Typ der Variablen content zur Laufzeit spezialisieren lässt (z.B. als int!)

Klasse Node II

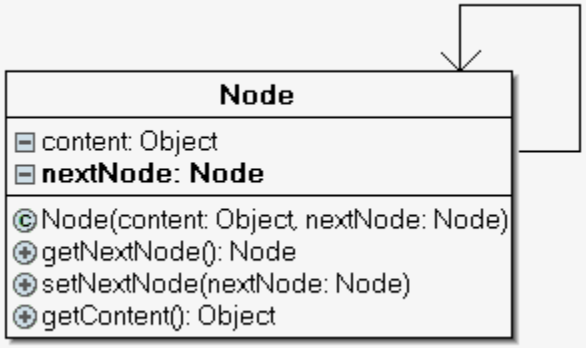
```
public void setNextNode (Node nextNode){
 this.nextNode = nextNode;
}
```

Setzt den Nachfolger

```
public Object getContent(){
 return content;
}
```

Liefert den Inhalt zurück

Klasse Node III



Klasse SinglyLinkedList I

```
public class SinglyLinkedList
{
 Node head;

 public SinglyLinkedList()
  {
   head = new Node(null,null);
  }
   ……..
}
```

Knotenobjekt für den Listenbeginn

Setzen der Werte für den Kopf=Listenbeginn
Inhalt: null, Verweis auf Nachfolger: null

Klasse SinglyLinkedList II

```
public void add (Object object2Add)
 {
  Node actNode = head;

 while(actNode.getNextNode() != null)
   actNode = actNode.getNextNode();

  Node node2Add = new Node(object2Add,null);

  actNode.setNextNode(node2Add);
 }
```

Temporäre Variable wird auf den Listenkopf gesetzt

Solange ein Nachfolger existiert: Mache den aktuellen Knoten zum Nachfolgerknoten;
Wenn der Nachfolger nicht mehr exist., sind wir beim letzten Knoten angelangt

Erstelle ein neues Objekt mit dem Inhalt „object2Add“ und keinem Nachfolger

Der Zeiger des aktuellen Knotens=letzte Knoten wird auf den neuen Knoten gesetzt

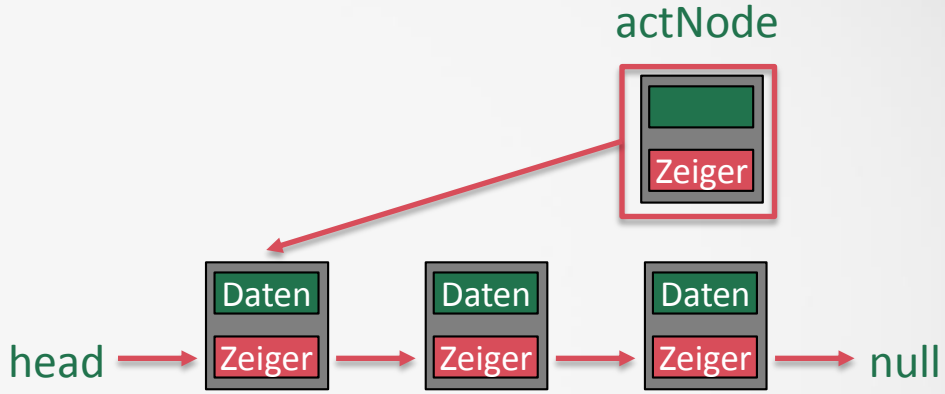
Klasse SinglyLinkedList III

```
public void add (Object object2Add)
 {
  Node actNode = head;

  while(actNode.getNextNode() != null)
    actNode = actNode.getNextNode();

  Node node2Add = new Node(object2Add,null);

  actNode.setNextNode(node2Add);
 }
```



Es wird eine Kopie von head
Variable erstellt, die auf den Listenanfang zeigt(Kein new!):
Man kann nun diese Kopie ändern(um die Liste durchlaufen)

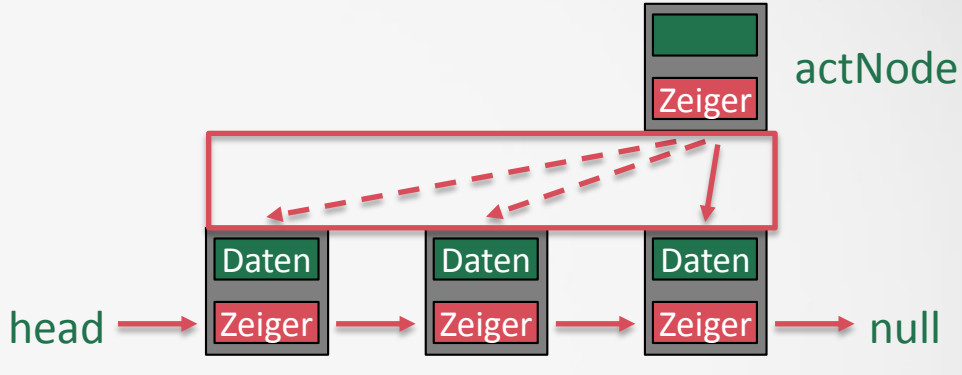
Klasse SinglyLinkedList IV

```
public void add (Object object2Add)
 {
  Node actNode = head;

  while(actNode.getNextNode() != null)
    actNode = actNode.getNextNode();

  Node node2Add = new Node(object2Add,null);

  actNode.setNextNode(node2Add);
 }
```



Alle Knoten durchlaufen, bis der Zeiger auf null zeigt:
Wir sind am Listenende!

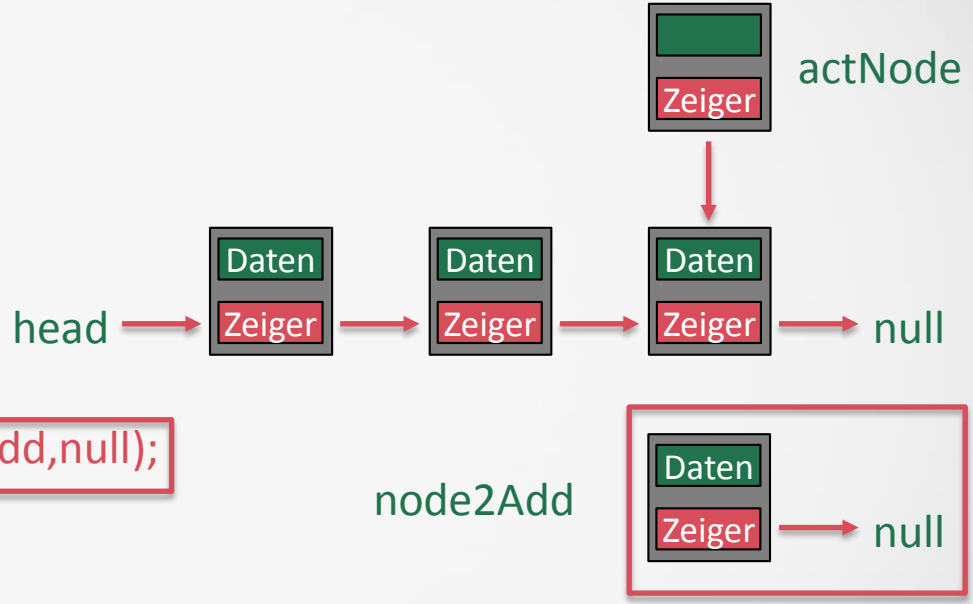
Klasse SinglyLinkedList V

```
public void add (Object object2Add)
 {
   Node actNode = head;

  while(actNode.getNextNode() != null)
    actNode = actNode.getNextNode();

  Node node2Add = new Node(object2Add,null);

   actNode.setNextNode(node2Add);
 }
```



Es wird ein neuer Knoten mit den gewünschten Daten erzeugt, der keinen Nachfolder hat

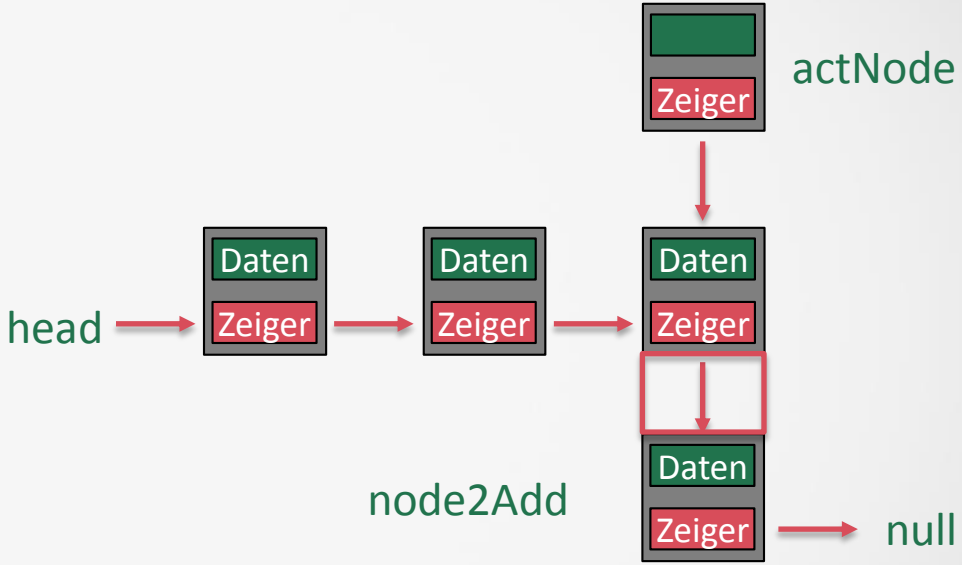
Klasse SinglyLinkedList VI

```
public void add (Object object2Add)
 {
   Node actNode = head;

  while(actNode.getNextNode() != null)
    actNode = actNode.getNextNode();

   Node node2Add = new
Node(object2Add,null);

  actNode.setNextNode(node2Add);
 }
```



Zeiger auf das neue Listenelement setzen

Klasse SinglyLinkedList VII

```
public class SinglyLinkedListTest
{
  public static void main (String args[])
  {
    SinglyLinkedList myList = new SinglyLinkedList();
     myList.add("Peter");
    myList.add("Paul");
     myList.add("Hava");
   }
}
```

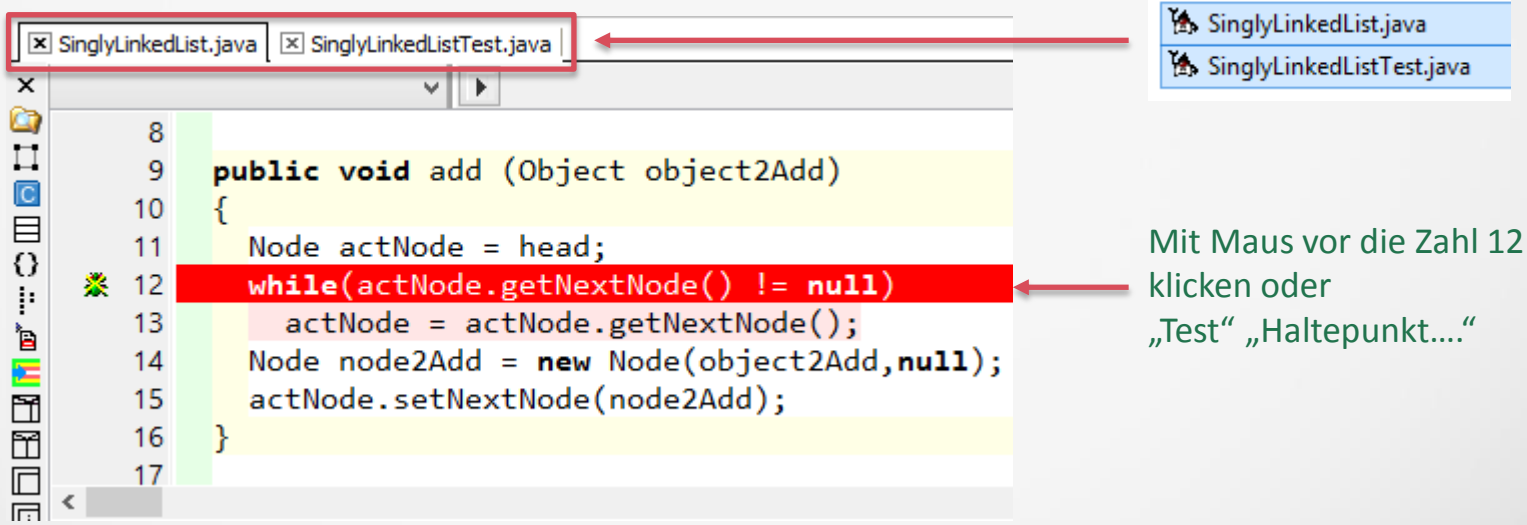
Neues Objekt erstellen

Element hinzufügen

Debugging von verketteten Strukturen I

1.

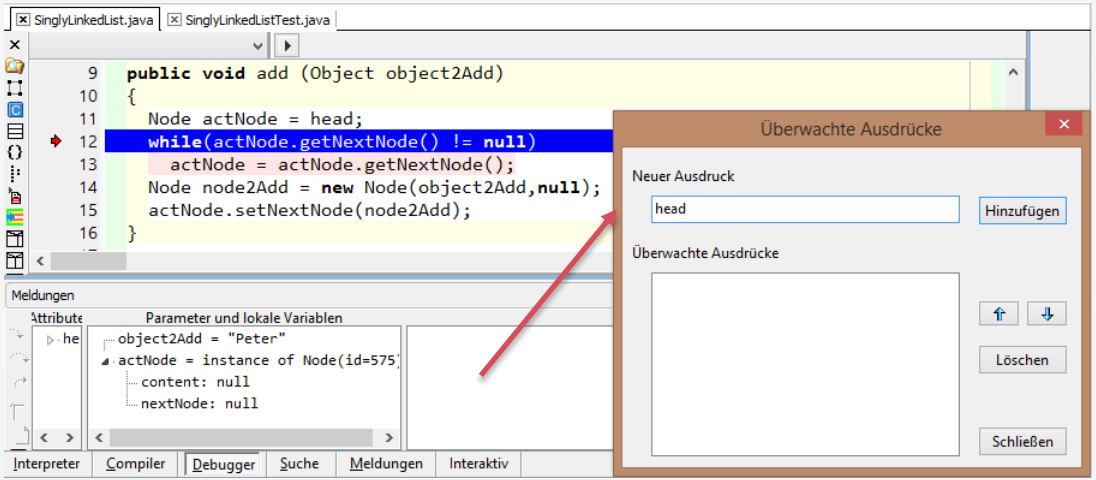
Laden zwei von den drei Beispieldateien(Download von Webseite!) und setzen eines Haltepunktes in der add-Methode der SinglyLinkedList-Klasse



Mit Maus vor die Zahl 12 klicken oder „Test“ „Haltepunkt….“

Debugging von verketteten Strukturen II

2. Starten Sie das Programm mit der Schaltfläche mit dem grünen Dreieck oder F9:
Es läuft bis zum Haltepunkt(Erkennbar am blauen Balken, der nun den roten „verdeckt“)!

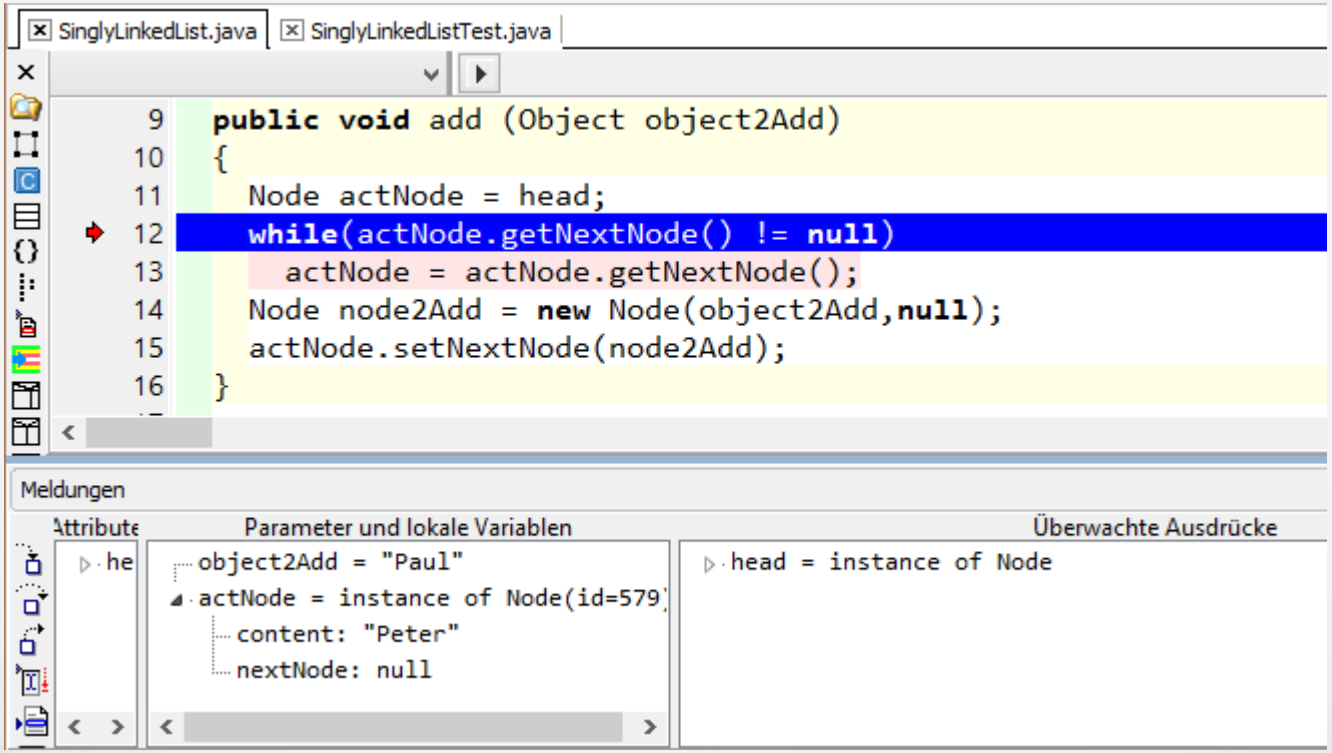


3. Fügen Sie die Variable head(die Ihnen die Wurzel zur Liste wiedergibt) mit Doppelklick auf das dritte Fenster von links zu den überwachten Ausdrücken hinzu! Da „head“ in der Klasse und nicht in der Methode deklariert ist, ist diese kein Parameter oder lokale Variable und wird daher nicht automatisch angezeigt!

Debugging von verketteten Strukturen III

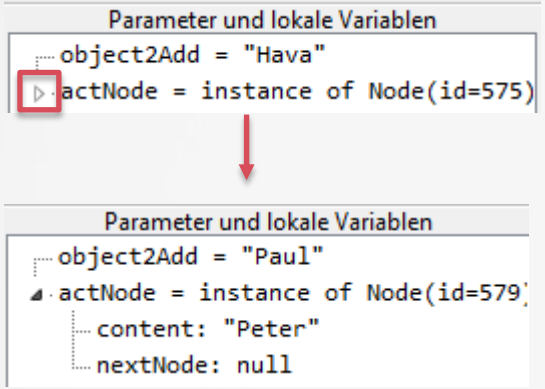
4.

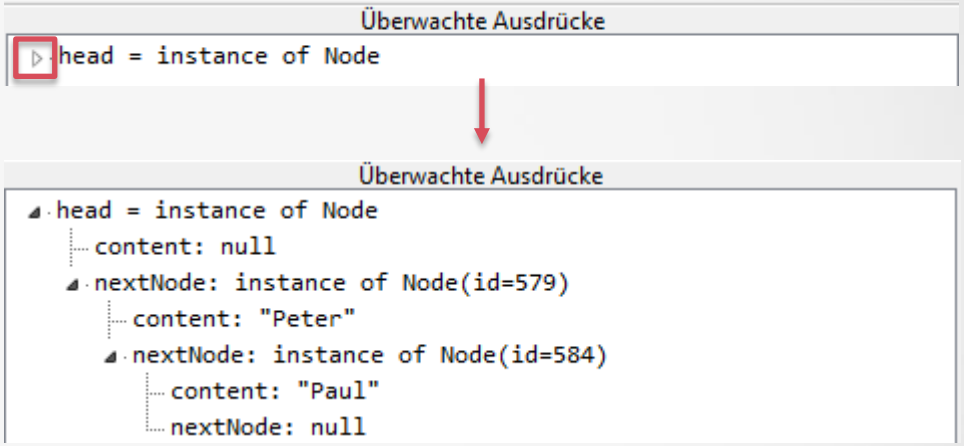
Bestätigen Sie nun zweimal die F9-Taste: Das Programm läuft weiter, aber jeweils erneut nur bis zum Haltepunkt(da im Hauptprogramm 3-mal die Methode add aufgerufen wird!)



Debugging von verketteten Strukturen IV

5. Erweitern Sie die Ansicht der Variablen actNode und head, die Referenzen auf weitere Objekte enthalten so lange, bis alle Elemente sichtbar sind

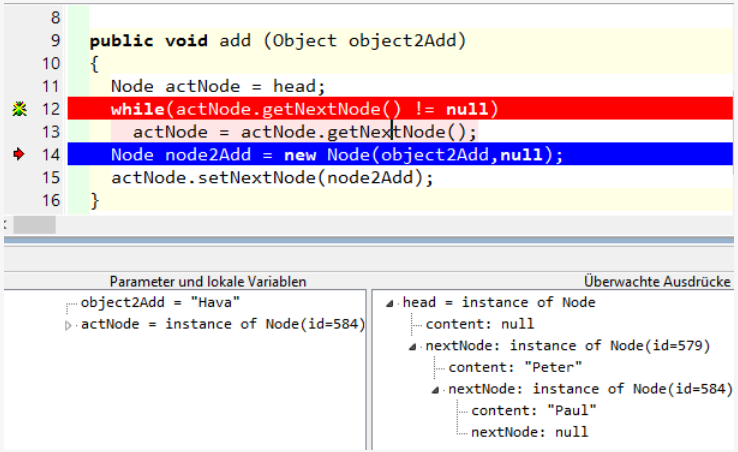




Resüme: Damit haben Sie die Möglichkeit, beim Kodieren jeder bel. Liste Haltepunkte zu setzen und sofort die Wirkung ihres Programmkodes auf den Zustand der gesamten Liste zu beobachten!

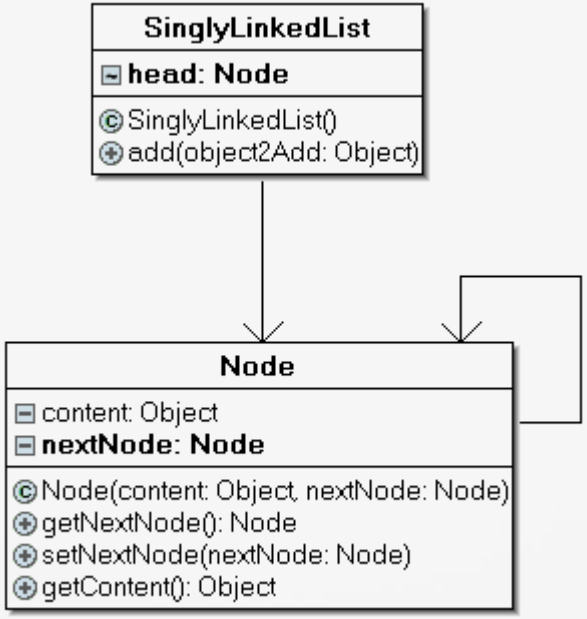
Debugging von verketteten Strukturen V

6. Betätigen Sie die F8-Taste bis Sie in Zeile 14 sind(Fatal wäre F7, dann würden Sie die Klasse CNode beim Debuggen aufrufen(passiert hier allerdings im Java-Editor nicht(obwohl in andern Editoren!))):

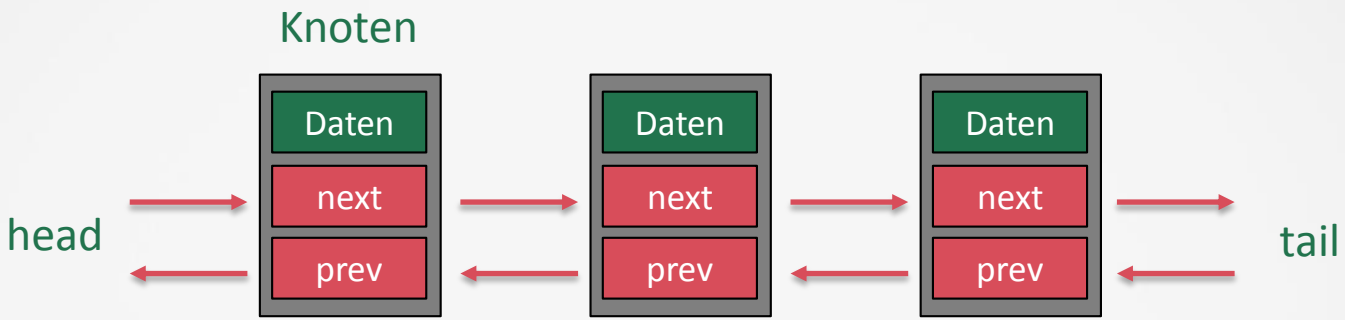


Resüme: Damit haben Sie die Möglichkeit, beim Kodieren jeder bel. Liste Haltepunkte zu setzen und sofort die Wirkung ihres Programmkodes auf den Zustand der gesamten Liste zu beobachten!
Die Qualität ihrer Fehlersuche ist für ihren „Gesamterfolg“!

Klasse SinglyLinkedList VIII



Darstellung: Doppelt verkettete Liste



bel. Inhalt möglich!

Verweis auf nächstes und letztes Objekt der Klasse!

```
public class CdNode {
   private Object content;
   private CdNode next;
   private CdNode prev;
}
```