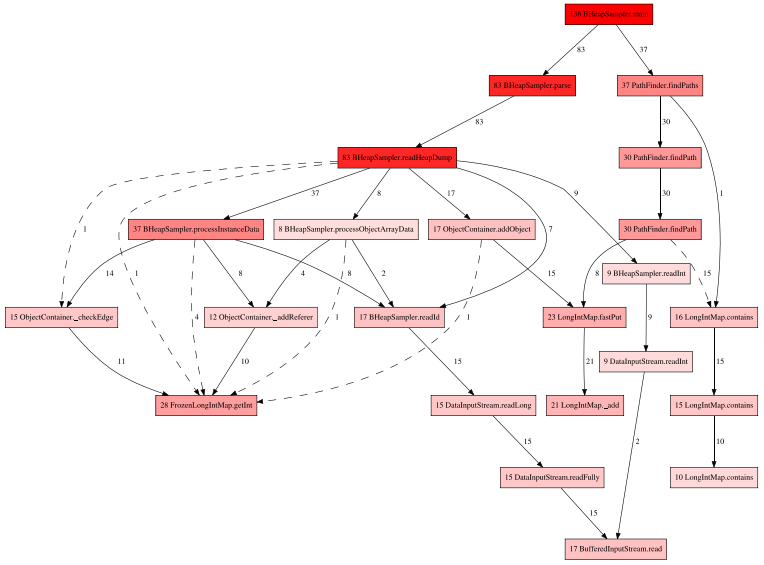
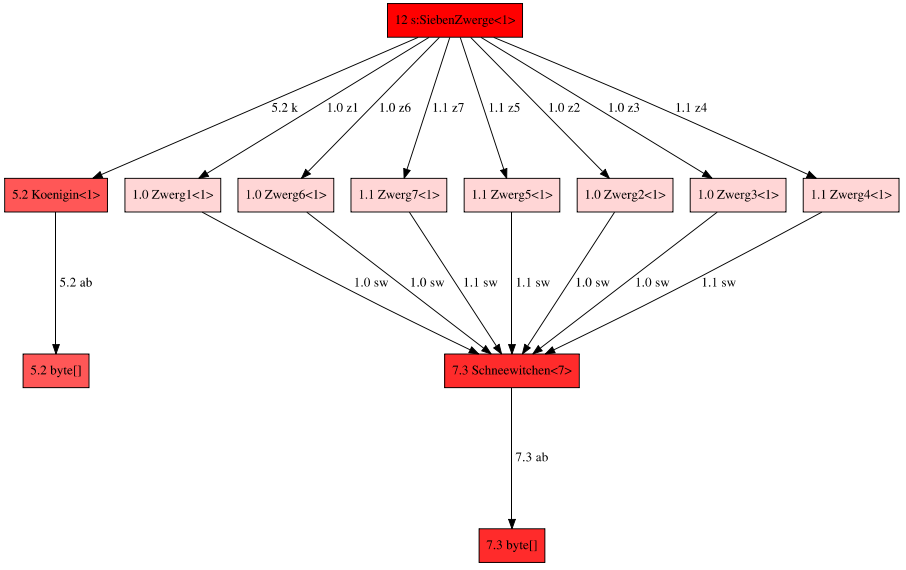
Graphenbasierte Performance- und Heap-Memory-Analyse

Dr. Arndt Brenschede / Java-Forum Stuttgart 2012





Teil 1: Performance Analyse

- Sampling <-> instrumentierende Profiler

- Beispiel: Performance-Analyse eines Heap-Analyse-Tools

- Darstellung der Ergebnisse: Listen, Bäume, Graphen

- “Parent-Identity” im Performance-Graphen: wo und warum

Sampling oder instrumentierender Profiler?

Sampling =

Abschätzung der Verweildauer (“elapsed time”) in Java-Methoden aus Momentaufnahmen der Callsstacks zu **zufällig** gewählten Zeitpunkten.

Instrumentierung =

die Verweildauer **und die Aufrufzahlen** werden direkt gemessen durch eine entsprechende Instrumentierung der VM oder des Bytecodes

-> Nachteil Sampling: nur Zeiten, keine Aufrufzahlen!

-> Nachteil Instrumentierung: verfälschte Zeiten durch Overhead!

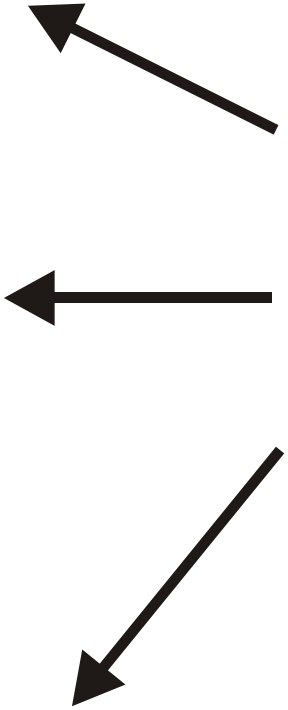
=> **Sampling liefert die genaueren Zeiten und ist produktiv möglich**

Beispiel: Profiling eines Heap-Analysetools durch Sampling

Stacktrace,threadid=1,1000ms
- LongIntMap._add(LongIntMap.java:219)
- LongIntMap.fastPut(LongIntMap.java:109)
- ObjectContainer.addObject(ObjectContainer.java:75)
- BHeapSampler.readHeapDump(BHeapSampler.java:463)
- BHeapSampler.parse(BHeapSampler.java:221)
- BHeapSampler.main(BHeapSampler.java:131)

Stacktrace,threadid=1,1000ms
- java.io.FileInputStream.available(Native Method)
- java.io.BufferedInputStream.read(Unknown Source)
- java.io.DataInputStream.readFully(Unknown Source)
- java.io.DataInputStream.readLong(Unknown Source)
- BHeapSampler.readId(BHeapSampler.java:610)
- BHeapSampler.readHeapDump(BHeapSampler.java:417)
- BHeapSampler.parse(BHeapSampler.java:221)
- BHeapSampler.main(BHeapSampler.java:131)

Stacktrace,threadid=1,1000ms
- FrozenLongIntMap.getInt(FrozenLongIntMap.java:110)
- ObjectContainer.getObjectNrForId(ObjectContainer.java:151)
- BHeapSampler.processObjectArrayData(BHeapSampler.java:517)
- BHeapSampler.readHeapDump(BHeapSampler.java:447)
- BHeapSampler.parse(BHeapSampler.java:221)
- BHeapSampler.main(BHeapSampler.java:131)



Momentaufnahmen des Callstacks zu Zufallszeitpunkten (3 von 136 dargestellt)

Einfache statistische Analyse der Samples: Method-und Total Time

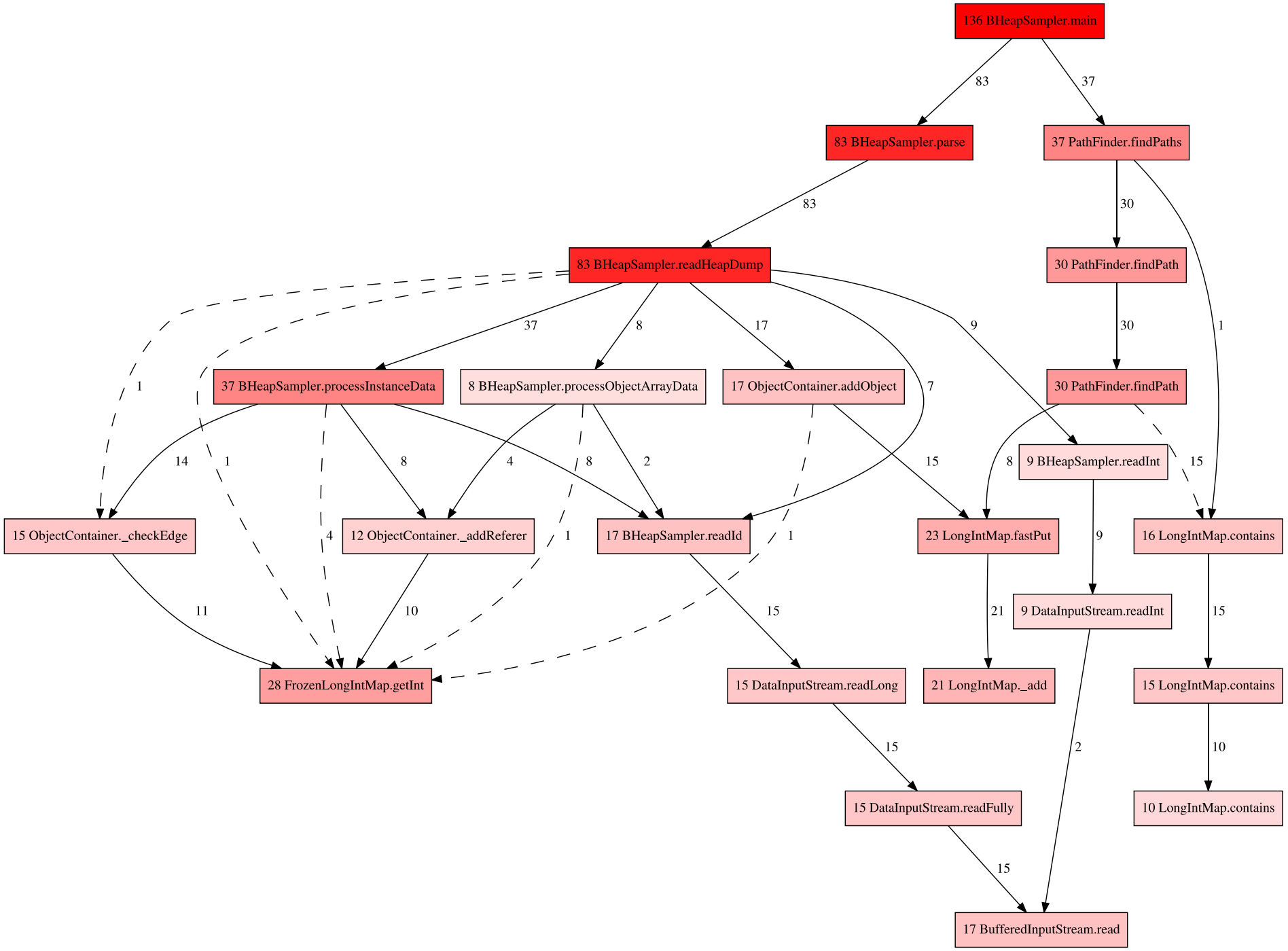
"Method-Time"

Sekunden	Methodenname
28	FrozenLongIntMap.getInt
21	LongIntMap._add
16	LongIntMap.contains
10	BufferedInputStream.read
8	LongIntMap.moveToFrozenArrays
7	DataInputStream.readInt
5	FileOutputStream.writeBytes
5	FileInputStream.available
4	ArrayList.get
4	ArrayList.size
3	FileInputStream.skip
2	BHeapSampler.readId
2	ArrayList.add
2	LongIntMap.fastPut
2	BHeapSampler.processInstanceData
2	FrozenLongIntMap.contains
2	FileInputStream.readBytes
2	ObjectContainer._checkEdge
1	ObjectContainer.getReferers
1	AbstractStringBuilder.<init>
1	HashMap.get
1	String.compareTo
1	StackFilter.getGraphNode

"Total Time"

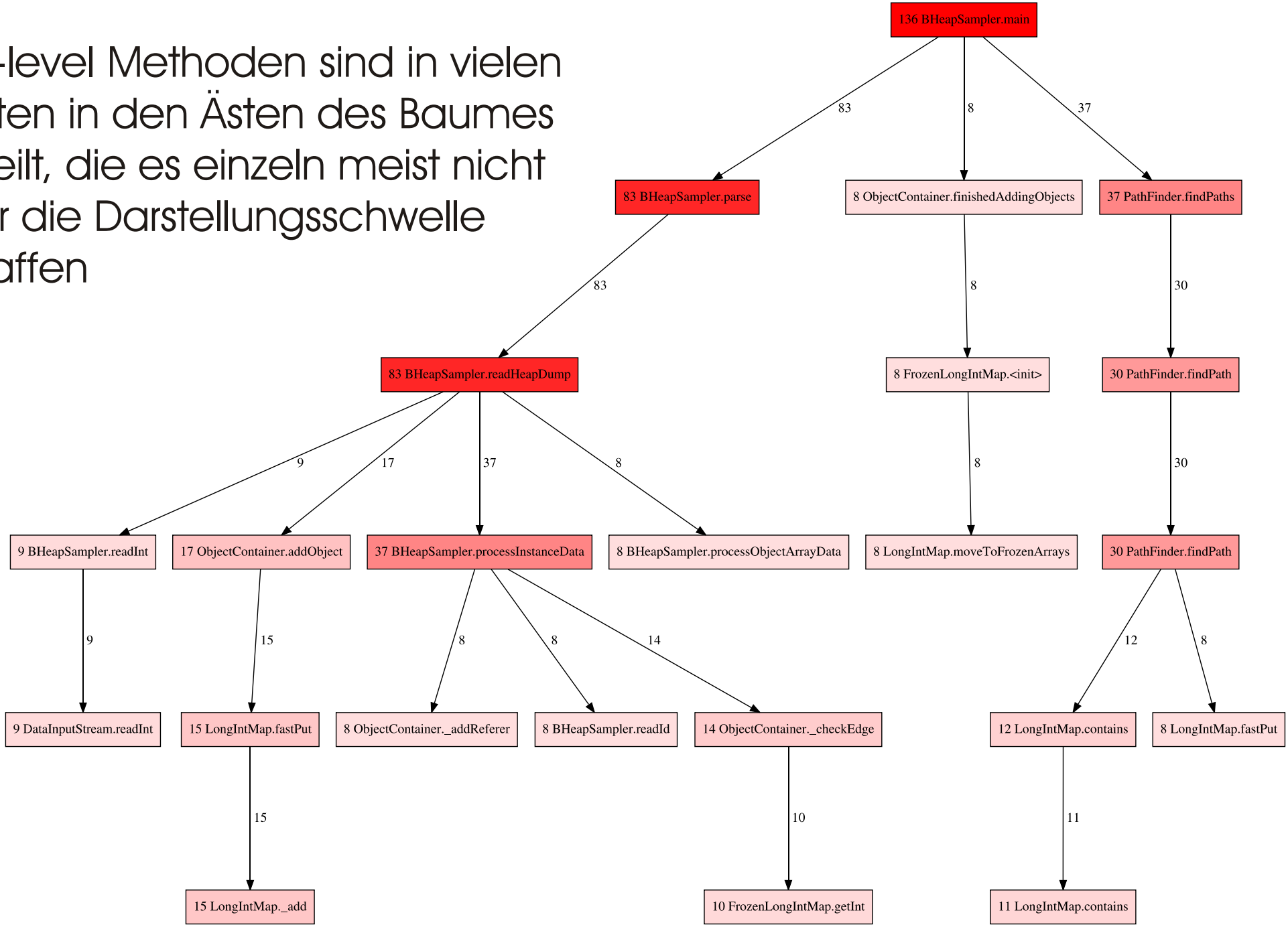
Sekunden	Methodenname
136	BHeapSampler.main
83	BHeapSampler.parse
83	BHeapSampler.readHeapDump
37	BHeapSampler.processInstanceData
37	PathFinder.findPaths
30	PathFinder.findPath
30	PathFinder.findPath
28	FrozenLongIntMap.getInt
23	LongIntMap.fastPut
21	LongIntMap._add
17	BHeapSampler.readId
17	BufferedInputStream.read
17	ObjectContainer.addObject
16	LongIntMap.contains
15	DataInputStream.readLong
15	DataInputStream.readFully
15	LongIntMap.contains
15	ObjectContainer._checkEdge
12	ObjectContainer._addReferer
10	LongIntMap.contains
9	BHeapSampler.readInt
9	DataInputStream.readInt
8	BHeapSampler.processObjectArrayData

Erweiterung der Total-Time um Aufrufbeziehungen: Laufzeit-Graph

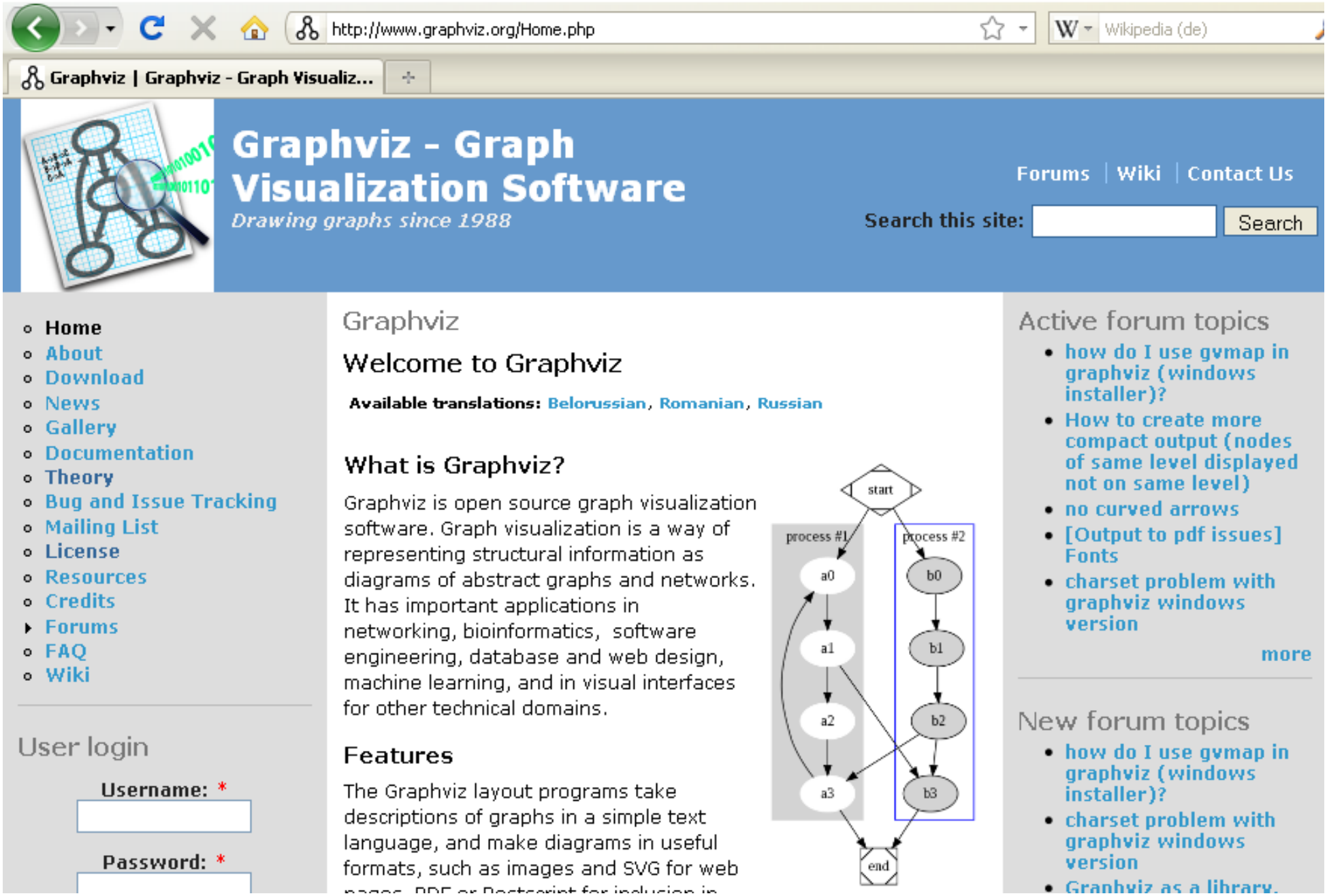


Baumdarstellung: jede Methode nur in ihrem Aufrufkontext dargestellt

-> Low-level Methoden sind in vielen Knoten in den Ästen des Baumes verteilt, die es einzeln meist nicht über die Darstellungsschwelle schaffen



Graphen-Visualisierung mit “GraphViz”



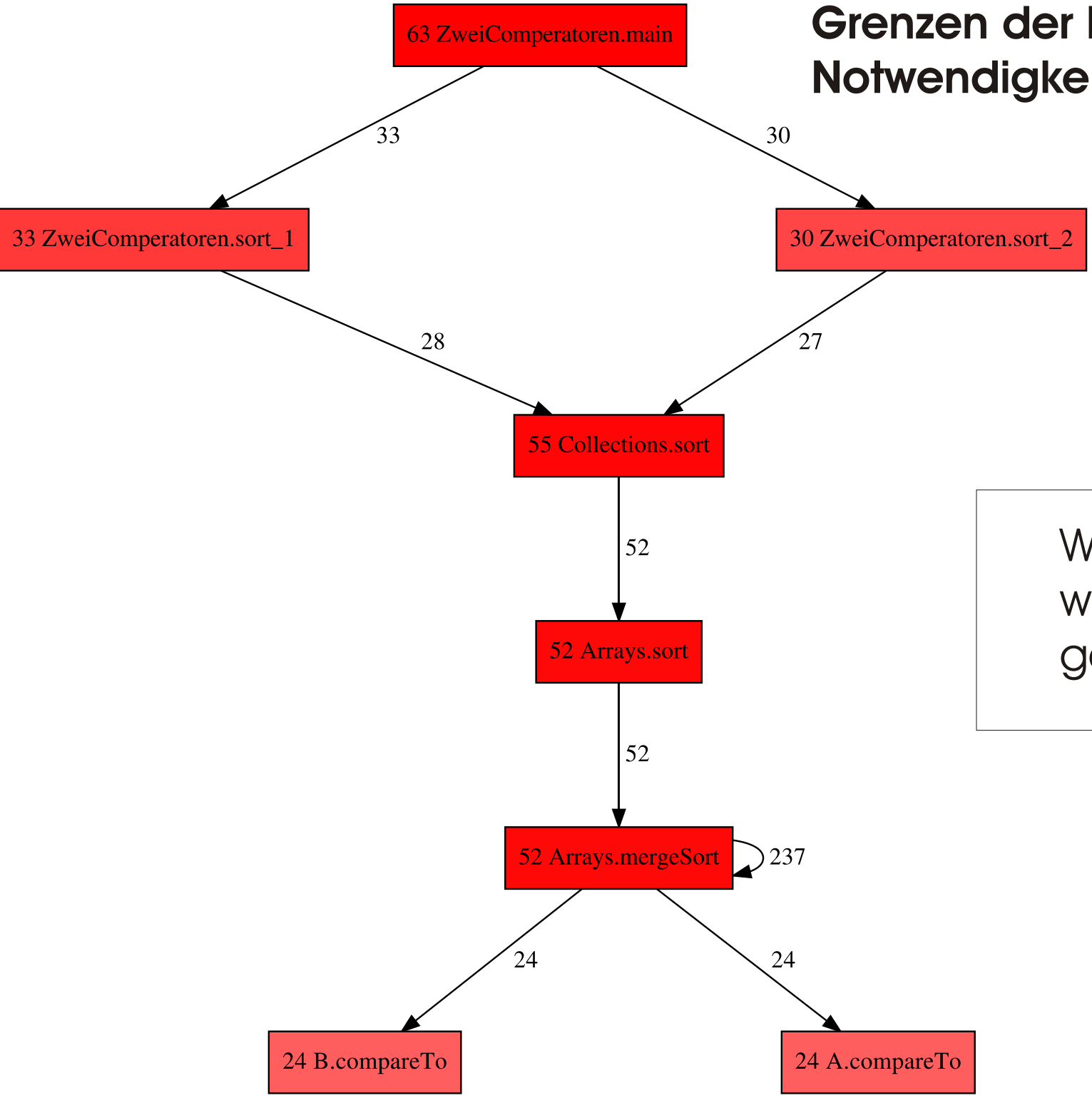
Beschreibung des Graphen in der “DOT”-Sprache:

```
digraph prof {
  size="12,9"; ratio = fill;
  node [style=filled];
  n0 -> n1 [label=" 83" style="solid"];
  n1 -> n2 [label=" 83" style="solid"];
  n2 -> n3 [label=" 7" style="solid"];
  n3 -> n4 [label=" 15" style="solid"];
  <... gekürzt ...>
  n22 -> n12 [label=" 11" style="solid"];
  n2 -> n12 [label=" 1" style="dashed"];
  n13 -> n22 [label=" 14" style="solid"];
  n8 -> n6 [label=" 2" style="solid"];

  n0 [label="136 BHeapSampler.main" shape="box" fillcolor="#ff0000" ]
  n1 [label="83 BHeapSampler.parse" shape="box" fillcolor="#ff2626" ]
  n2 [label="83 BHeapSampler.readHeapDump" shape="box" fillcolor="#ff2626" ]
  n3 [label="17 BHeapSampler.readId" shape="box" fillcolor="#ffc1c1" ]
  <... gekürzt ...>
  n19 [label="16 LongIntMap.contains" shape="box" fillcolor="#ffc4c4" ]
  n20 [label="15 LongIntMap.contains" shape="box" fillcolor="#ffc7c7" ]
  n21 [label="10 LongIntMap.contains" shape="box" fillcolor="#ffd8d8" ]
  n22 [label="15 ObjectContainer._checkEdge" shape="box" fillcolor="#ffc7c7" ]
}
```

Kanten

Knoten

Grenzen der Methoden-Identität, Notwendigkeit von “Parent-Identity”

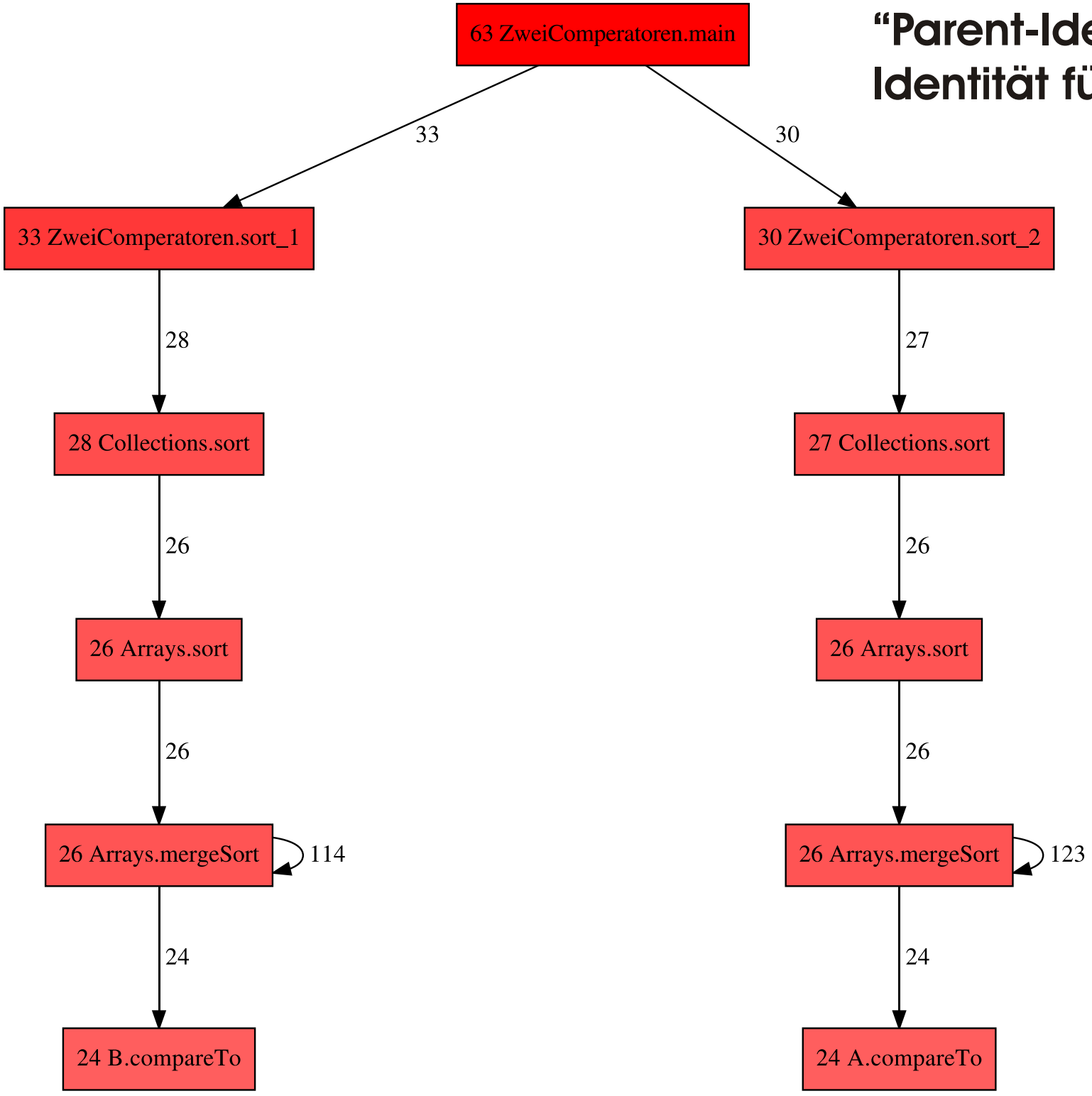


← 2 Sortier-Methoden

Welche Methode hat welchen Comperator gerufen???

← 2 Comperatoren

“Parent-Identity”: Aufhebung der Identität für spezielle Methoden



Teil 2: Heap-Memory Analyse

- Abgrenzung: was ist (statische) Heap-Memory Analyse

- Einführung “Memory-Graphen”

- Vergleich Memory-Graphen <-> Dominator-Tree

- Vorstellung Memory-Graph Generator Tool

Was ist Heap-Memory Analyse

- **Abgrenzung 1:** dynamische Aspekte (Allocation Tracking, ..)

- **Abgrenzung 2:** non-heap Memory Aspekte (perm-space, native, ..)

- Heapdump = Momentaufnahme des Java-Heap-Memories

- Statische Heap-Memory-Analyse sucht ein quantitatives Verständnis über Inhalt und Struktur eines Heapdumps

- durch Vergleich mehrerer Heapdumps auch im Zeitverlauf

- Memory-Analyse ist mehr als nur "Leak-Detection"

- Heap-Memory-Analyse ist Teil der QA-Strategie

Wie geht Heap-Memory-Analyse ?

1. Einfache Summenstatistiken (z.B. “jmap -histo <pid>”)

2. Mit “klassischen” Tools

- einzelne Objektinstanzen herauspicken auf Basis von Summenstatistiken

- dafür Pfade zu GC-Wurzeln berechnen

} **Ansatz für automatisiertes “Heap-Sampling”**

- in diesen Pfaden nach einem gemeinsamen Muster suchen

3. Mit aggregierenden Tools (-> Dominator Tree)

- die Objektinstanzen werden nach ihren Dominanzbeziehungen in einen Baum sortiert

- zu jeder Instanz wird die “retained size” berechnet als Summe über den Unterbaum

4. Mischformen aus Dominator-Tree und klassischer Pfadsuche

-> z.B. Eclipse Memory Anaylzer Tool (“MAT”)


memory-path,10485
java/awt/geom/GeneralPath<7111>(43836:-)
java/util/HashMap$Entry<8305>(40797:value)
java/util/HashMap$Entry<8305>(36744:next)
[Ljava/util/HashMap$Entry;<122>(39019:)
java/util/HashMap<120>(3109:table)
bezier/BezierAnim<2>(182
:leakMap)
sun/awt/windows/WPanelPeer
ROOT_JNI_GLOBAL(76880:-)
memory-path,10485
byte[](29412:-)
java/awt/geom/GeneralPath<7111>(26749:pointTypes)
java/util/HashMap$Entry<8305>(24323:value)
[Ljava/util/HashMap$Entry;<122>(39019:)
java/util/HashMap<120>(3109:table)
bezier/BezierAnim<2>(182
:leakMap)
sun/awt/windows/WPanelPeer
ROOT_JNI_GLOBAL(76880:-)
memory-path,10485
float[](35732:-)
java/awt/geom/GeneralPath<7111>(35730:floatCoords)
java/util/HashMap$Entry<8305>(35734:value)
[Ljava/util/HashMap$Entry;<122>(39019:)
java/util/HashMap<120>(3109:table)
bezier/BezierAnim<2>(182
:leakMap)
sun/awt/windows/WPanelPeer
ROOT_JNI_GLOBAL(76880:-)
“Gemeinsames Muster”

Strukturelle Equivalenz von Memory-Pfad und Stack-Trace:

Memory-Pfad

Gewicht in bytes

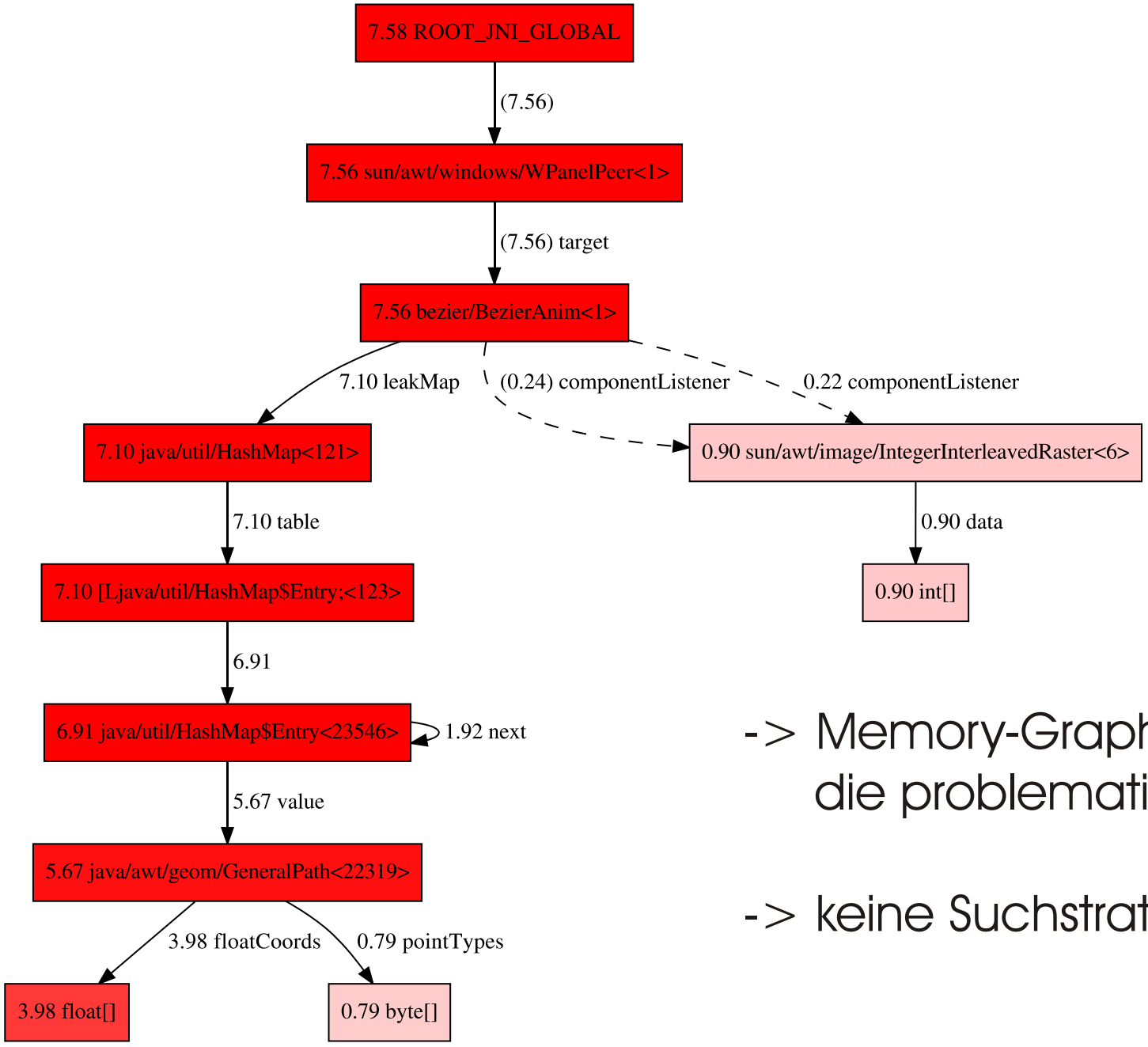
```
memory-path,10485
    java/awt/geom/GeneralPath<7111>(43836:-)
    java/util/HashMap$Entry<8305>(40797:value)
    java/util/HashMap$Entry<8305>(36744:next)
    [Ljava/util/HashMap$Entry;<122>(39019:)
    java/util/HashMap<120>(3109:table)
    bezier/BezierAnim<2>(1821:leakMap)
    sun/awt/windows/WPanelPeer<2>(1776:-target)
    ROOT_JNI_GLOBAL(76880:-)
```

Stack-Trace

Gewicht in ms

```
Stacktrace,threadid=1,1000ms
    java.io.FileInputStream.available(Native Method)
    java.io.BufferedInputStream.read(Unknown Source)
    java.io.DataInputStream.readFully(Unknown Source)
    java.io.DataInputStream.readLong(Unknown Source)
    BHeapSampler.readId(BHeapSampler.java:610)
    BHeapSampler.readHeapDump(BHeapSampler.java:417)
    BHeapSampler.parse(BHeapSampler.java:221)
    BHeapSampler.main(BHeapSampler.java:131)
```

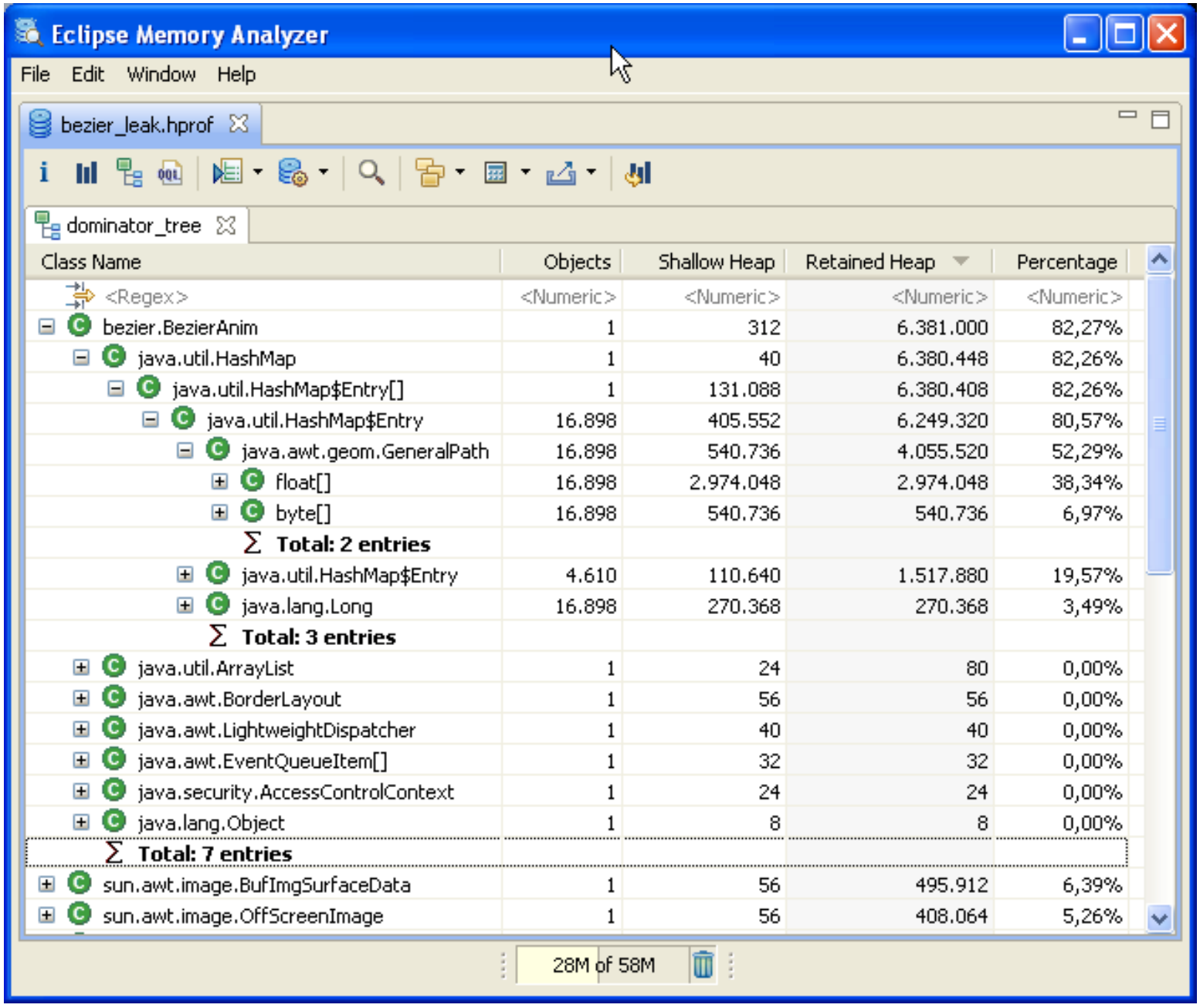
“Demo Memory Leak” im Class-Level Memory Graph:



-> Memory-Graph zeigt unmittelbar die problematische Struktur

-> keine Suchstrategie notwendig

Gleiches Beispiel im MAT Dominator Tree (Group by Classes):



Class Name	Objects	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
bezier.BezierAnim	1	312	6.381.000	82,27%
java.util.HashMap	1	40	6.380.448	82,26%
java.util.HashMap$Entry[]	1	131.088	6.380.408	82,26%
java.util.HashMap$Entry	16.898	405.552	6.249.320	80,57%
java.awt.geom.GeneralPath	16.898	540.736	4.055.520	52,29%
float[]	16.898	2.974.048	2.974.048	38,34%
byte[]	16.898	540.736	540.736	6,97%
Total: 2 entries				
java.util.HashMap$Entry	4.610	110.640	1.517.880	19,57%
java.lang.Long	16.898	270.368	270.368	3,49%
Total: 3 entries				
java.util.ArrayList	1	24	80	0,00%
java.awt.BorderLayout	1	56	56	0,00%
java.awt.LightweightDispatcher	1	40	40	0,00%
java.awt.EventQueueItem[]	1	32	32	0,00%
java.security.AccessControlContext	1	24	24	0,00%
java.lang.Object	1	8	8	0,00%
Total: 7 entries				
sun.awt.image.BufImgSurfaceData	1	56	495.912	6,39%
sun.awt.image.OffScreenImage	1	56	408.064	5,26%

-> brauchbar, aber Detailproblem bzgl. HashMapEntry-Linked-List

Eclipse Memory Analyzer Tool

Firefox

Eclipse Memory Analyzer Open Source Project

www.eclipse.org/mat/

Wikipedia (de)

eclipse

Visit other Eclipse Sites

Home Downloads Users Members Committers Resources Projects About Us

Google™ Custom Search

About This Project

Memory Analyzer »

- Screenshots
- Getting Started
- FAQ (Wiki)
- Forum
- Blog
- Unresolved bugs and enhancements
- Report Bug

Downloads »

- Releases
- Nightly Builds

Contributors »

Memory Analyzer (MAT)

The Eclipse Memory Analyzer is a fast and feature-rich **Java heap analyzer** that helps you find memory leaks and reduce memory consumption.

Use the Memory Analyzer to analyze productive heap dumps with hundreds of millions of objects, quickly calculate the retained sizes of objects, see who is preventing the Garbage Collector from collecting objects, run a report to automatically extract leak suspects.

Contributors

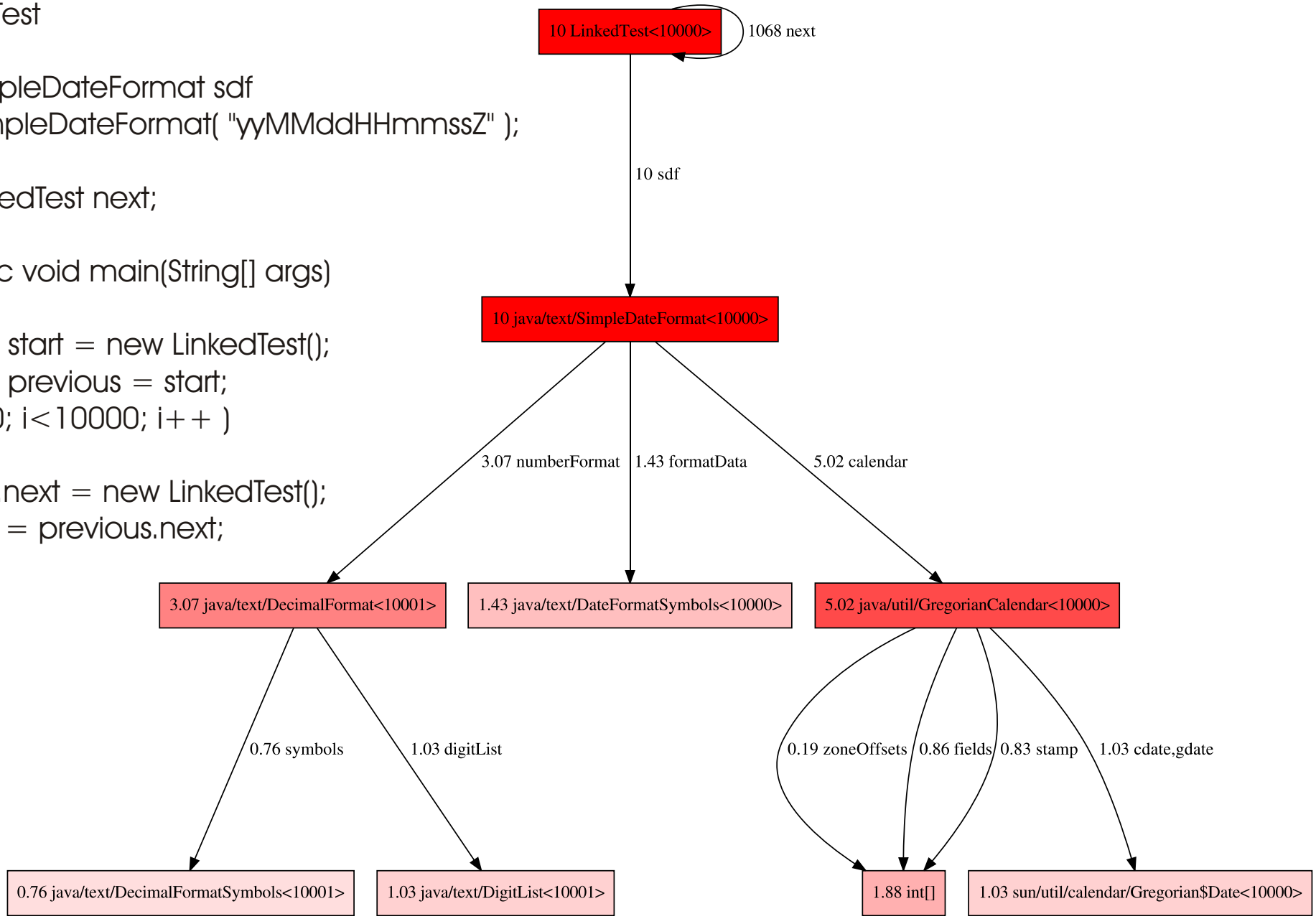
- SAP
- IBM

Previous Talks

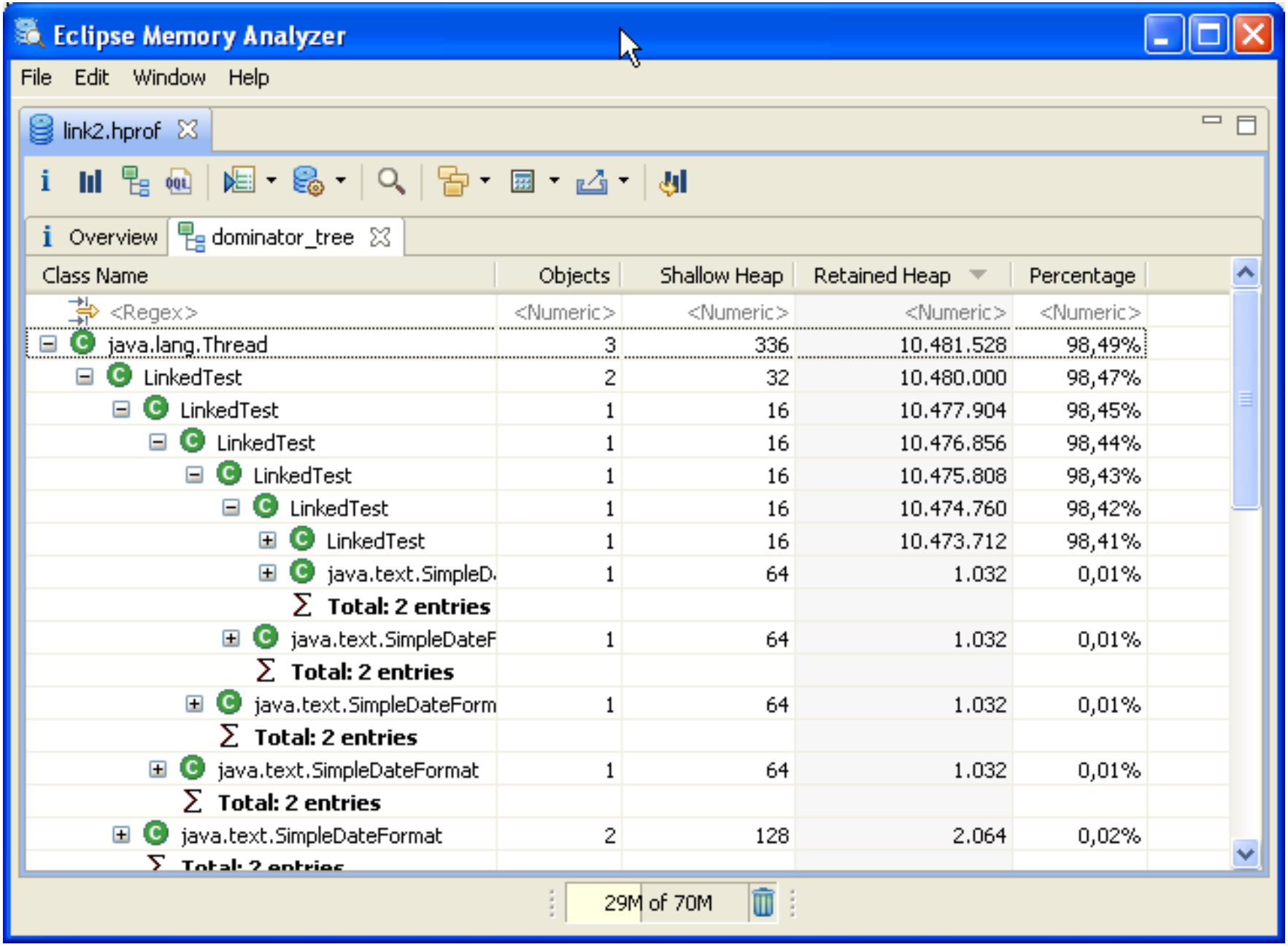
- Eclipse Sum… Ludwigsburg,
- TheServerSi… - Europe, Pra…
- JavaOne, Jun…
- Java Forum

Grenzen des Dominator-Trees: Beispiel 1: verkettete Liste

```
class LinkedTest
{
  private SimpleDateFormat sdf
    = new SimpleDateFormat( "yyMMddHHmmssZ" );

  private LinkedTest next;

  public static void main(String[] args)
  {
    LinkedTest start = new LinkedTest();
    LinkedTest previous = start;
    for( int i=0; i<10000; i++ )
    {
      previous.next = new LinkedTest();
      previous = previous.next;
    }
  }
}
```



Ansicht im Dominator-Tree:



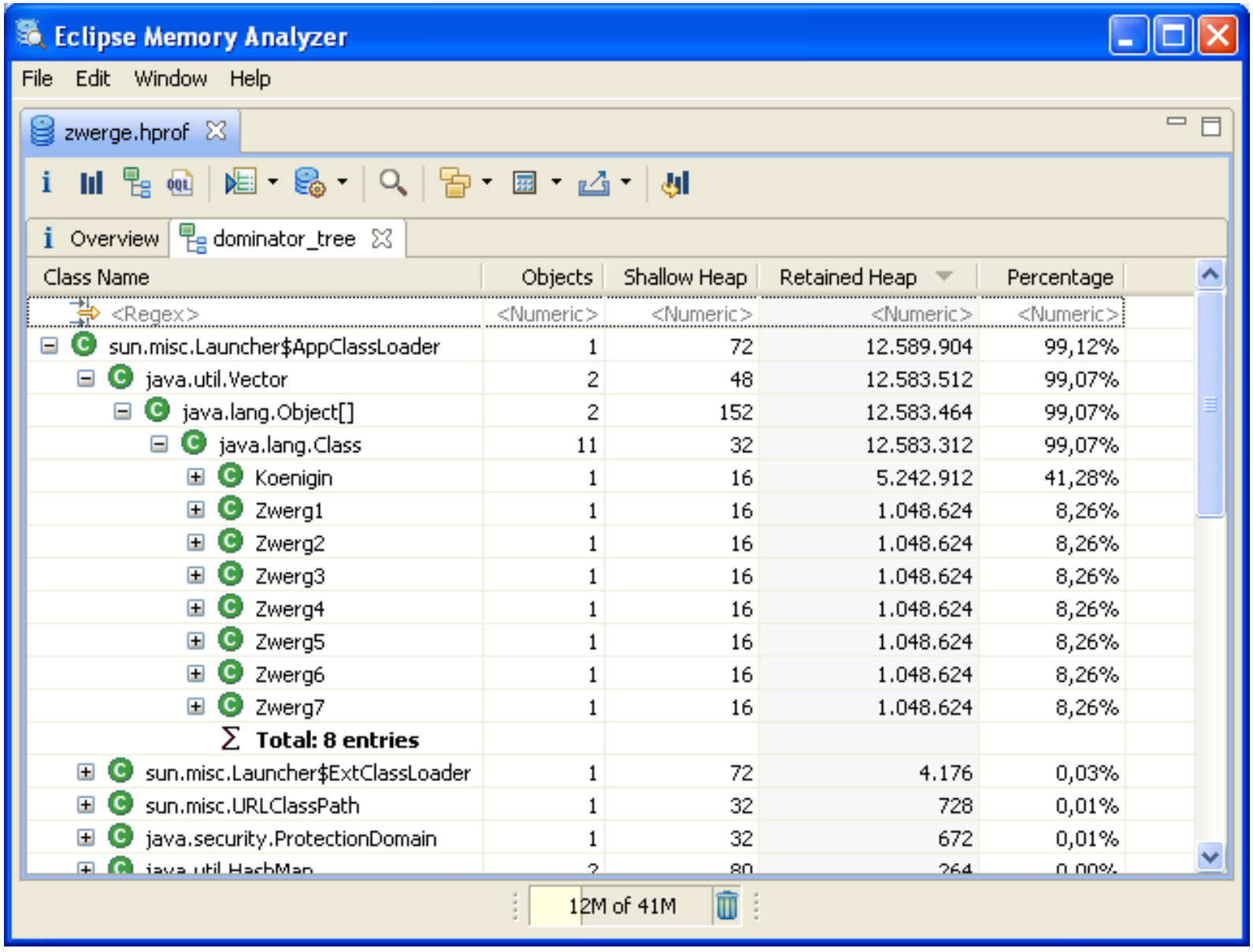
Class Name	Objects	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
java.lang.Thread	3	336	10.481.528	98,49%
LinkedTest	2	32	10.480.000	98,47%
LinkedTest	1	16	10.477.904	98,45%
LinkedTest	1	16	10.476.856	98,44%
LinkedTest	1	16	10.475.808	98,43%
LinkedTest	1	16	10.474.760	98,42%
LinkedTest	1	16	10.473.712	98,41%
java.text.SimpleD.	1	64	1.032	0,01%
Total: 2 entries				
java.text.SimpleDateF	1	64	1.032	0,01%
Total: 2 entries				
java.text.SimpleDateForm	1	64	1.032	0,01%
Total: 2 entries				
java.text.SimpleDateFormat	1	64	1.032	0,01%
Total: 2 entries				
java.text.SimpleDateFormat	2	128	2.064	0,02%
Total: 2 entries				

-> Der Dominator-Tree kann die Listenelemente nicht aggregieren

Grenzen des Dominator-Trees: Beispiel 2: die sieben Zwerge

```
class Koenigin          { byte[] ab = new byte[5*1024*1024]; }
class Schneewitchen { byte[] ab = new byte[1*1024*1024]; }

class BaseZwerg { Schneewitchen sw = new Schneewitchen(); }
class Zwerg1 extends BaseZwerg {}
class Zwerg2 extends BaseZwerg {}
class Zwerg3 extends BaseZwerg {}
class Zwerg4 extends BaseZwerg {}
class Zwerg5 extends BaseZwerg {}
class Zwerg6 extends BaseZwerg {}
class Zwerg7 extends BaseZwerg {}

class SiebenZwerge
{
  static Koenigin k = new Koenigin();
  static Zwerg1 z1 = new Zwerg1();
  static Zwerg2 z2 = new Zwerg2();
  static Zwerg3 z3 = new Zwerg3();
  static Zwerg4 z4 = new Zwerg4();
  static Zwerg5 z5 = new Zwerg5();
  static Zwerg6 z6 = new Zwerg6();
  static Zwerg7 z7 = new Zwerg7();
}
```

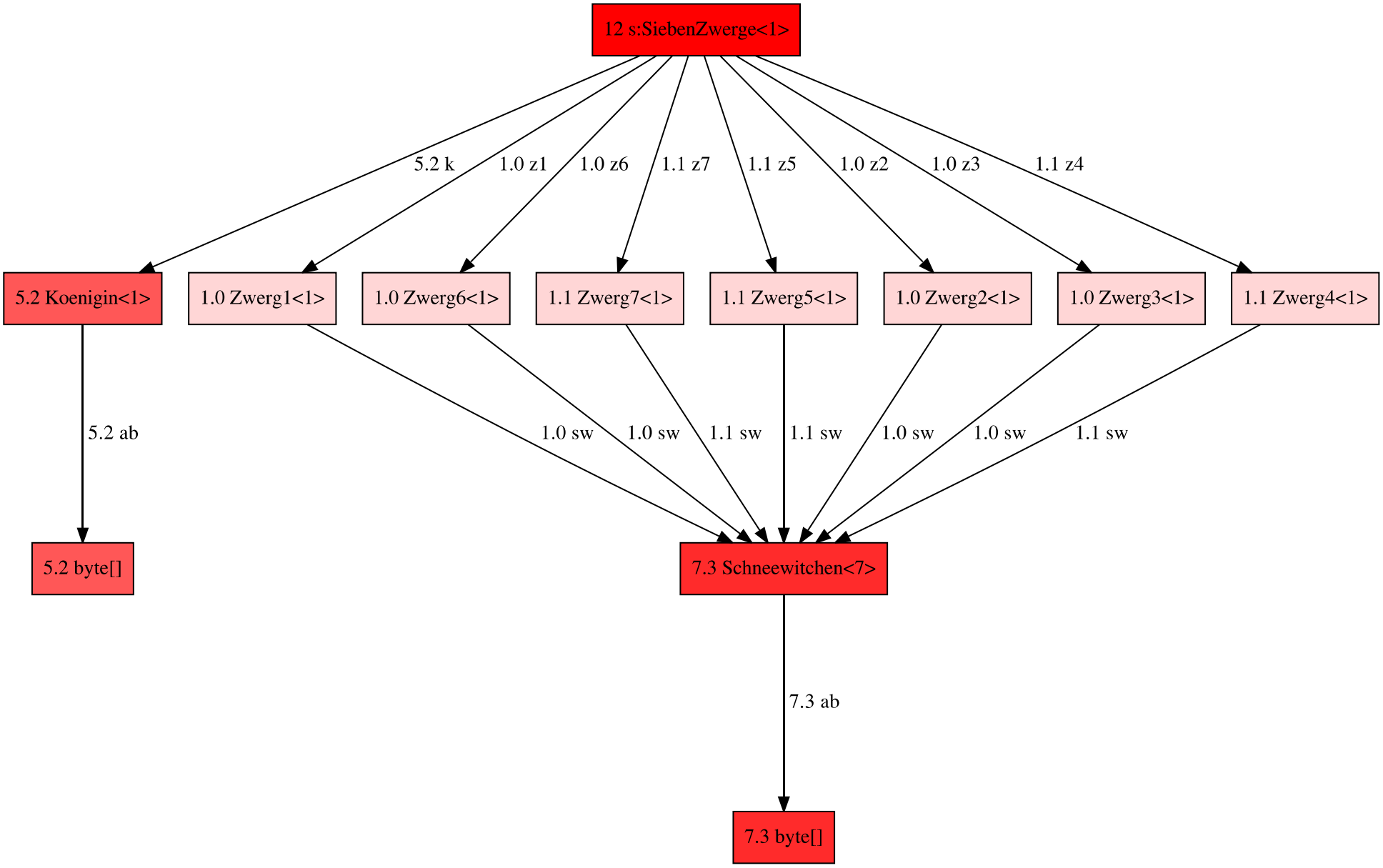
Spieglein, Spieglein an der Wand, wer ist die grösste im Land ?



Class Name	Objects	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
sun.misc.Launcher$AppClassLoader	1	72	12.589.904	99,12%
java.util.Vector	2	48	12.583.512	99,07%
java.lang.Object[]	2	152	12.583.464	99,07%
java.lang.Class	11	32	12.583.312	99,07%
Koenigin	1	16	5.242.912	41,28%
Zwerg1	1	16	1.048.624	8,26%
Zwerg2	1	16	1.048.624	8,26%
Zwerg3	1	16	1.048.624	8,26%
Zwerg4	1	16	1.048.624	8,26%
Zwerg5	1	16	1.048.624	8,26%
Zwerg6	1	16	1.048.624	8,26%
Zwerg7	1	16	1.048.624	8,26%
Total: 8 entries				
sun.misc.Launcher$ExtClassLoader	1	72	4.176	0,03%
sun.misc.URLClassPath	1	32	728	0,01%
java.security.ProtectionDomain	1	32	672	0,01%
java.util.HashMap	2	80	264	0,00%

“Zwar seid Ihr, Königin, schon ganz schön gross ...”

“ ... aber hinter den sieben Zwergen ...”



-> ein (Memory-) Zauberspiegel, der die Wahrheit spricht, zeichnet einen Klassengraphen

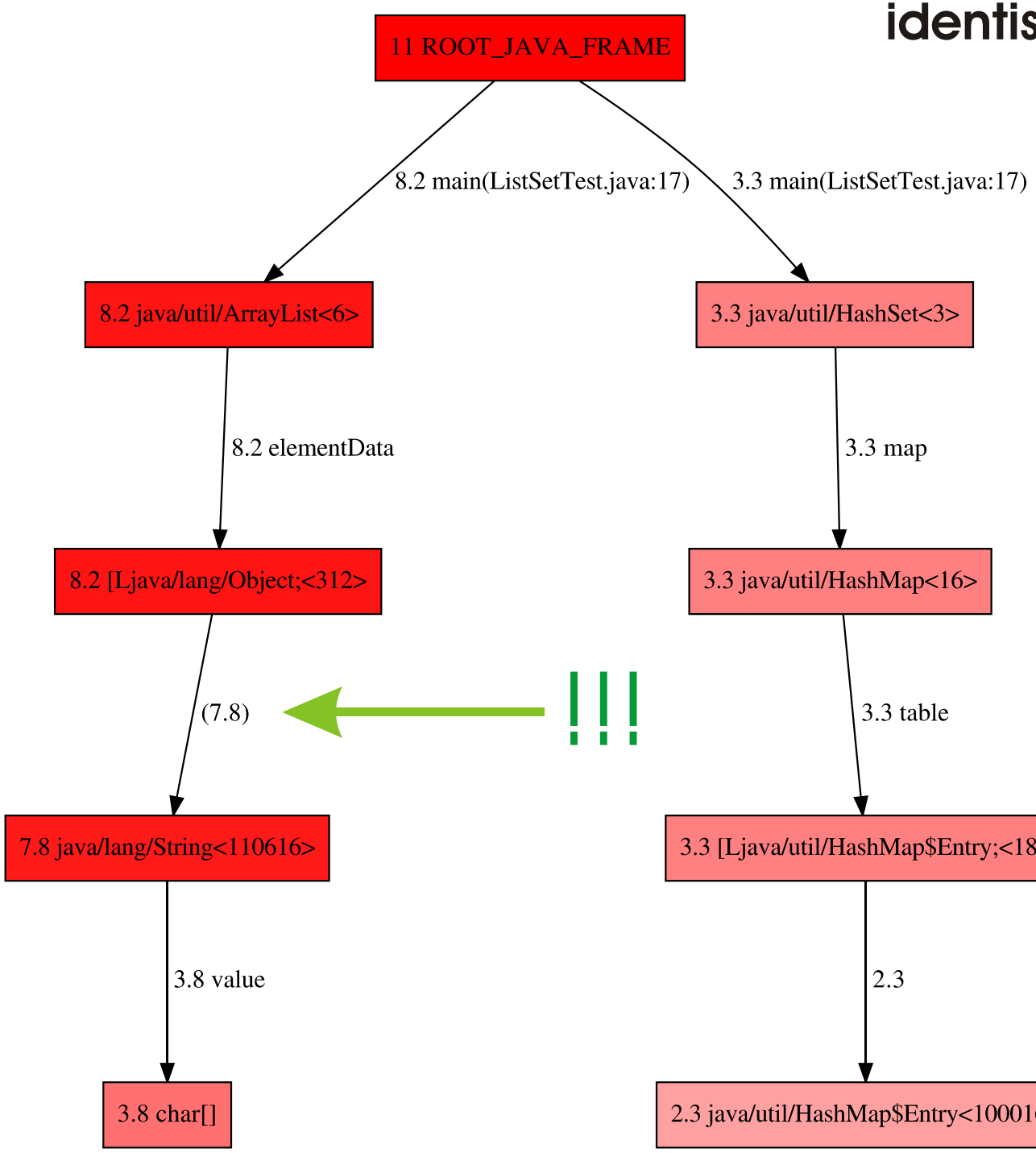
Wahre Zwerge und ihre Schneewitchens ...

```
public abstract class PanelBase extends JPanel
{
  private Font headerFonts
    = new Font("SansSerif", Font.BOLD, 20);

  ...
}
```

Klein aber häufig

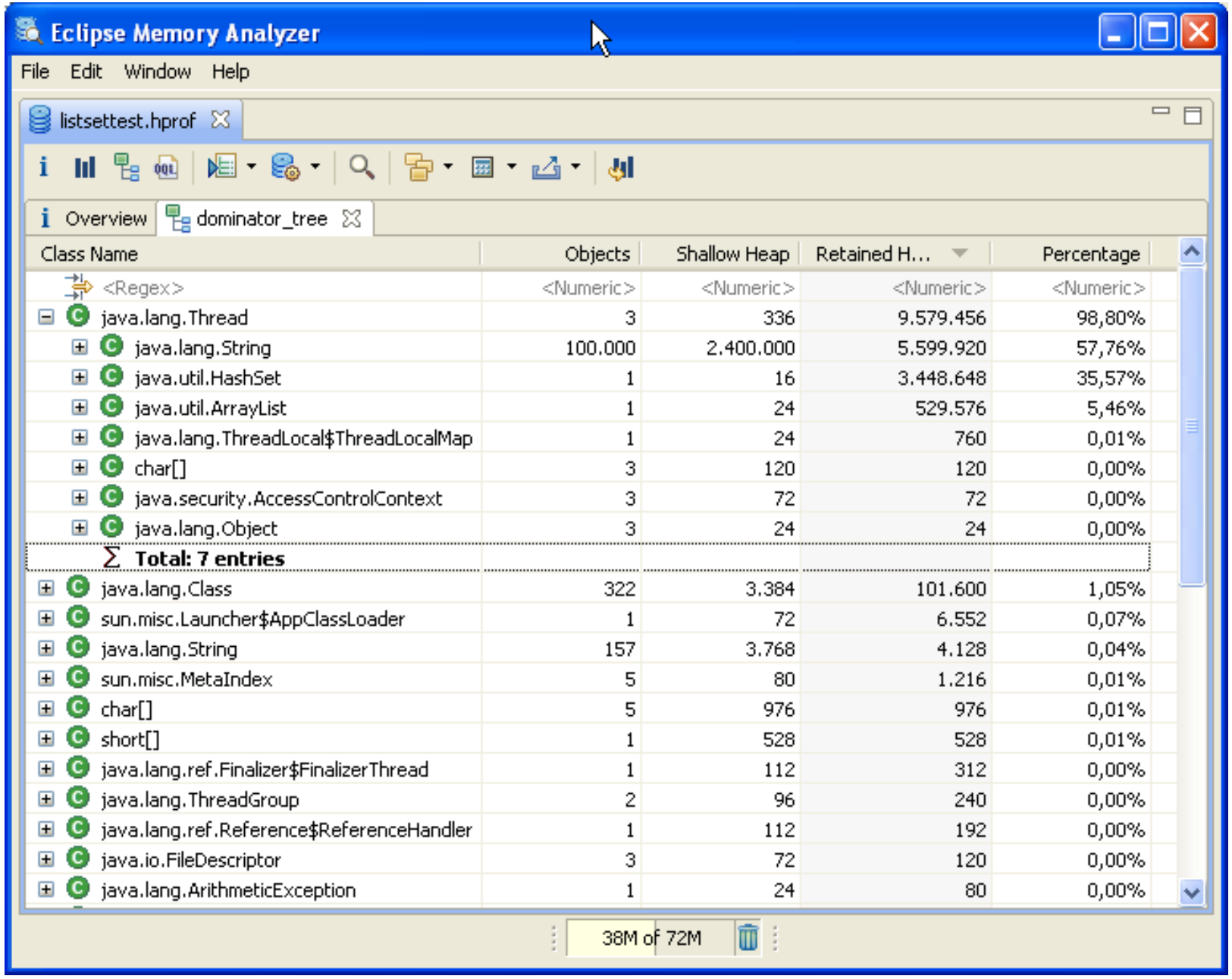
```
public abstract class ActionBase
{
  private SimpleDateFormat sdf
    = new SimpleDateFormat( "yyMMddHHmmssZ" );

  ...
}
```

Grenzen des Dominator-Trees: Beispiel 3: identische Objekte in 2 Containern:




```
List list = new ArrayList();
Set set = new HashSet();

for(int i=0; i<100000; i++ )
{
  String s = "Text_" + i;
  list.add( s );
  set.add( s );
}
```

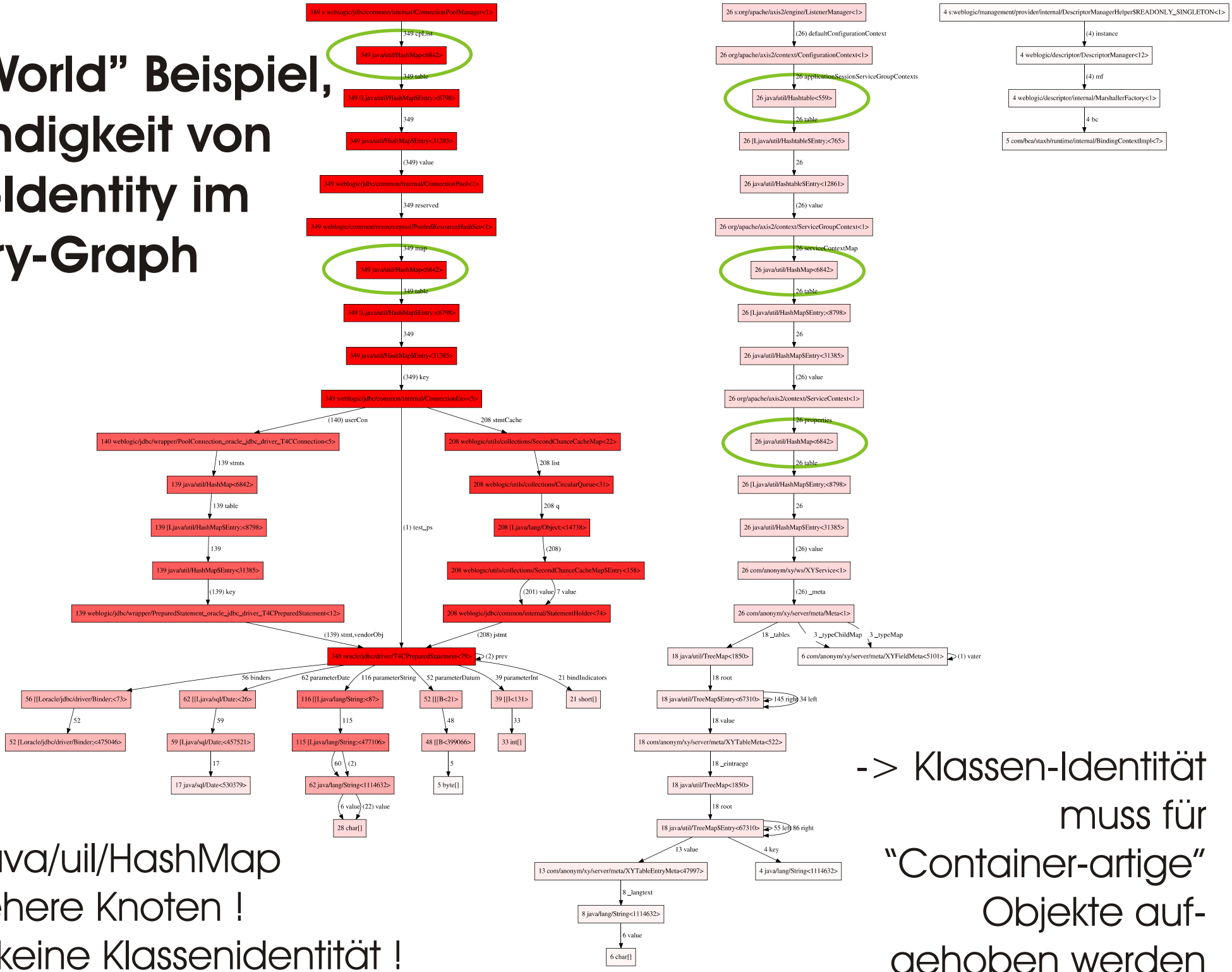
Ansicht im Dominator-Tree:



Class Name	Objects	Shallow Heap	Retained H...	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
java.lang.Thread	3	336	9.579.456	98,80%
java.lang.String	100.000	2.400.000	5.599.920	57,76%
java.util.HashSet	1	16	3.448.648	35,57%
java.util.ArrayList	1	24	529.576	5,46%
java.lang.ThreadLocal$ThreadLocalMap	1	24	760	0,01%
char[]	3	120	120	0,00%
java.security.AccessControlContext	3	72	72	0,00%
java.lang.Object	3	24	24	0,00%
Σ Total: 7 entries				
java.lang.Class	322	3.384	101.600	1,05%
sun.misc.Launcher$AppClassLoader	1	72	6.552	0,07%
java.lang.String	157	3.768	4.128	0,04%
sun.misc.MetaIndex	5	80	1.216	0,01%
char[]	5	976	976	0,01%
short[]	1	528	528	0,01%
java.lang.ref.Finalizer$FinalizerThread	1	112	312	0,00%
java.lang.ThreadGroup	2	96	240	0,00%
java.lang.ref.Reference$ReferenceHandler	1	112	192	0,00%
java.io.FileDescriptor	3	72	120	0,00%
java.lang.ArithmeticException	1	24	80	0,00%

-> keiner der beiden Container dominiert die Strings

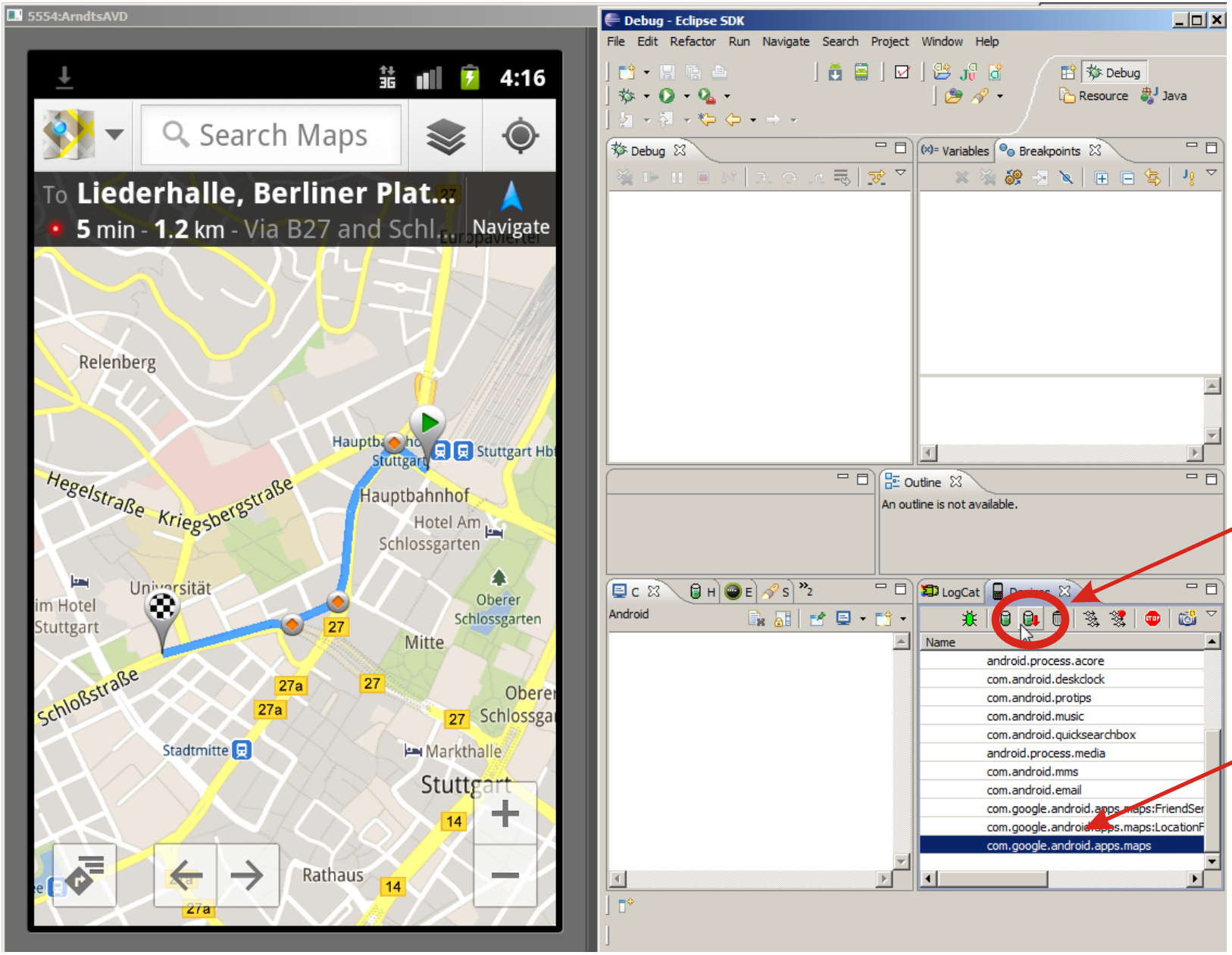
“Real-World” Beispiel, Notwendigkeit von Parent-Identity im Memory-Graph



= java/uil/HashMap
-->mehere Knoten !
-> keine Klassenidentität !

-> Klassen-Identität muss für “Container-artige” Objekte auf-gehoben werden

Gewinnung von Heap-Dumps im Android Development Kit



BHeapSampler: Memory-Graph Generator zum Download:

Http://www.dr-brenschede.de/bheapsampler/

- Binär-Version zum Download (kein Open Source)
- ähnlich schnell und memory-effizient wie MAT
  (getestet bis 4 GB, theoretisch unbegrenzt)

- Einschränkungen in der freien Version bzgl. der Konfigurierbarkeit:

  -> keine Filter
  -> keine Snapshot-Differenzen
  -> hardkodierte Parent-Indenity

Ziel: “run and see” Erfahrung, Strategie- und konfigurationsfrei

Eclipse MAT vs. Heap-Sampler: Konkurrenten oder Traumpaar?

Vorteil MAT:

- viel bessere Analysemöglichkeiten auf Detail-Ebene
- technisch ausgereift, gute Doku
- exakte Zahlen, kein statistischer Fehler
- Open Source

Vorteil Heap-Sampler:

- besser für Überblick
- "strategiefreie" Analyse -> ohne spezielle Erfahrung verwendbar
- gute, kompakte Darstellung des Ergebnisses (PDF, ...)
- Heapdump im Zielsystem verarbeitbar (Datenschutz!)

--> MAT + BHeapSampler ergänzen sich und sollten bei schwierigen Fragen beide benutzt werden

Vielen Dank für Ihre Aufmerksamkeit!

Fragen?

BHeapSampler download:
Http://www.dr-brenschede.de/bheapsampler/