

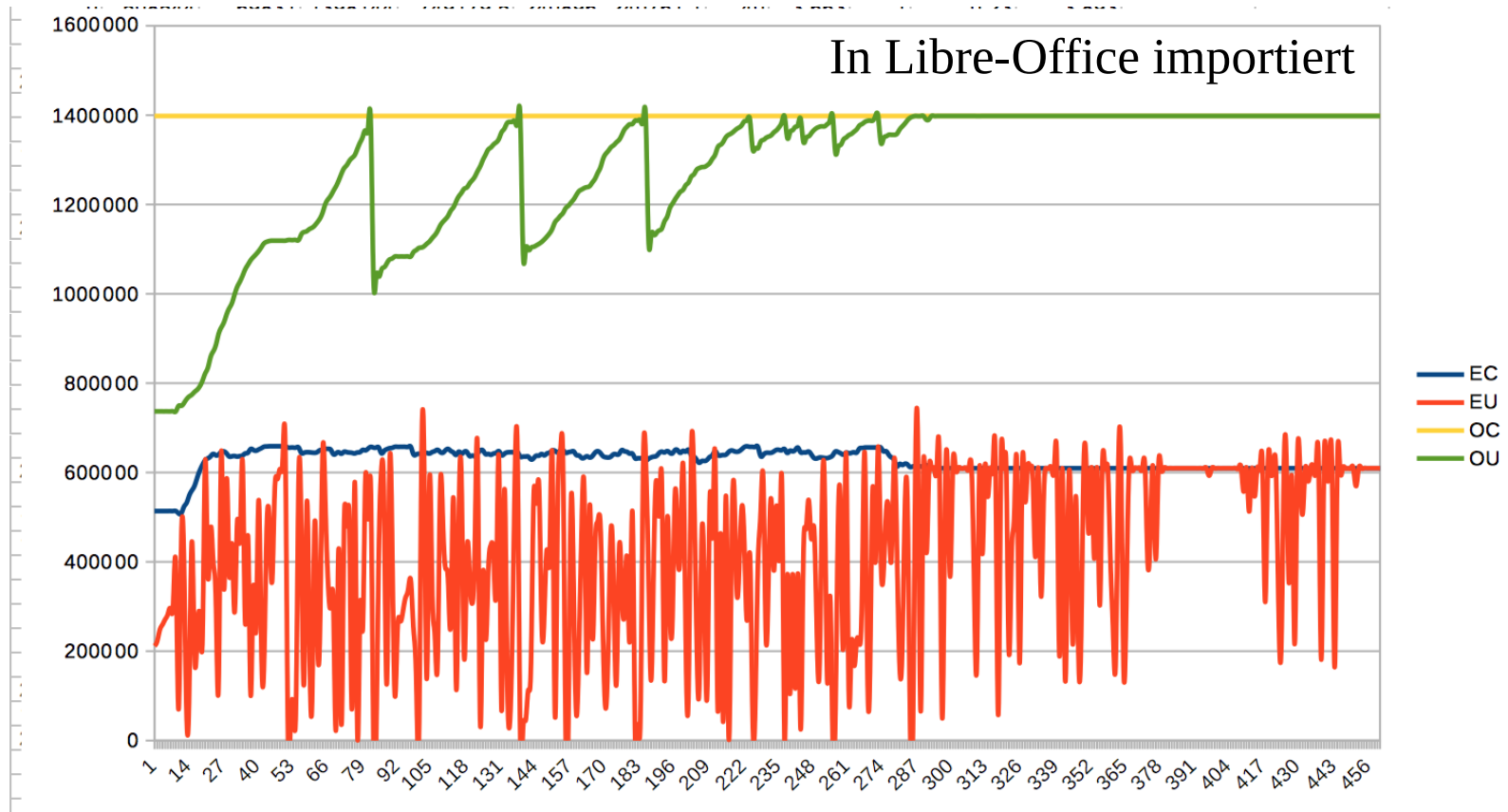
Java Memory Analyse



Java Memory

■ jstat -gc

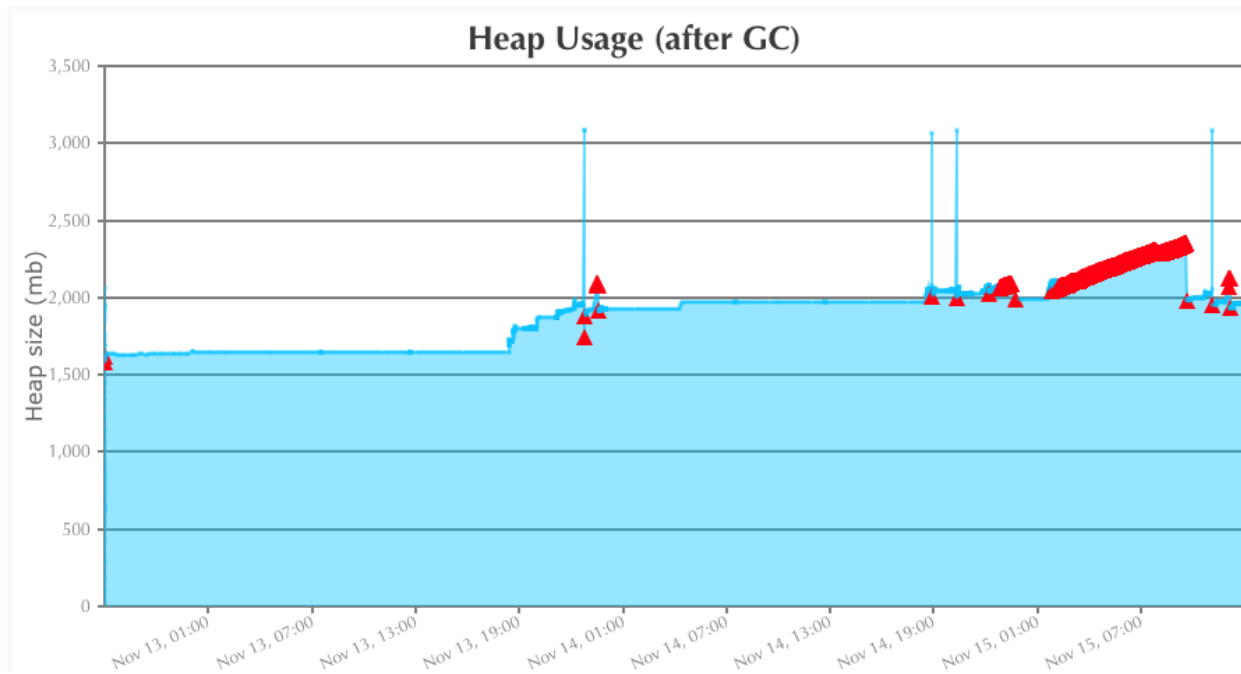
```
jstat -gc -t `eap7 pid` 10s | tee tmpmen.csv
```



Memory

- GC-Log analysieren

- Garbagecat: `java -jar garbagecat-2.0.12-SNAPSHOT.jar gc.log.0.current`
- GCeasy: <http://www.gceasy.io>



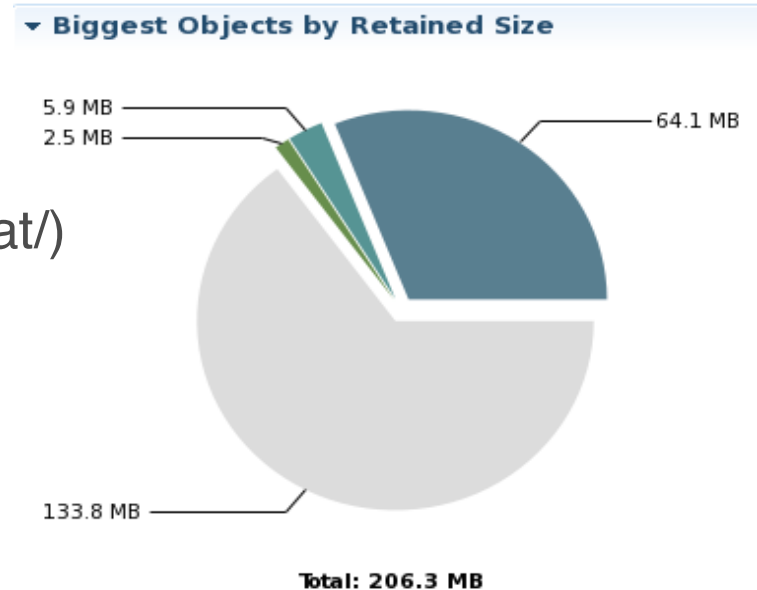
Memory Analyse

- Heap-Dumps:

- jvisualvm
- jmap -dump:live,file=dump.bin,format=b <pid>

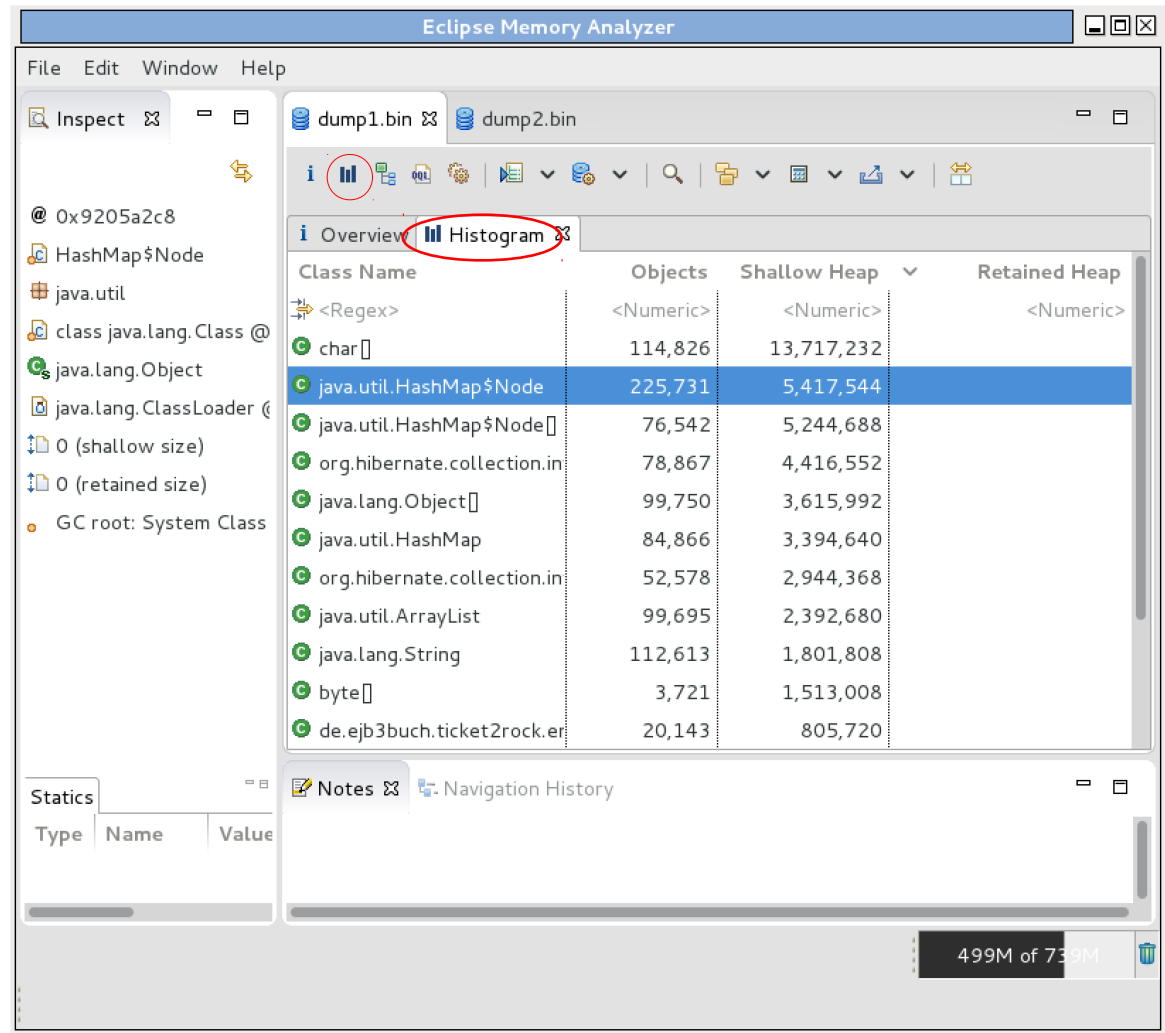
- Analyse

- jvisualvm
- jhat (goßer Speicherbedarf)
- MAT (<http://www.eclipse.org/mat/>)



MAT – Memory Analysing Tool

- **Objects**
 - Anzahl Objekte der Klasse
- **Shallow Heap**
 - Direkt von den Objekten verwendeter Speicher
- **Retained Heap**
 - Von den Objekten (indirekt) gehaltener Speicher



Vergleich von Heaps

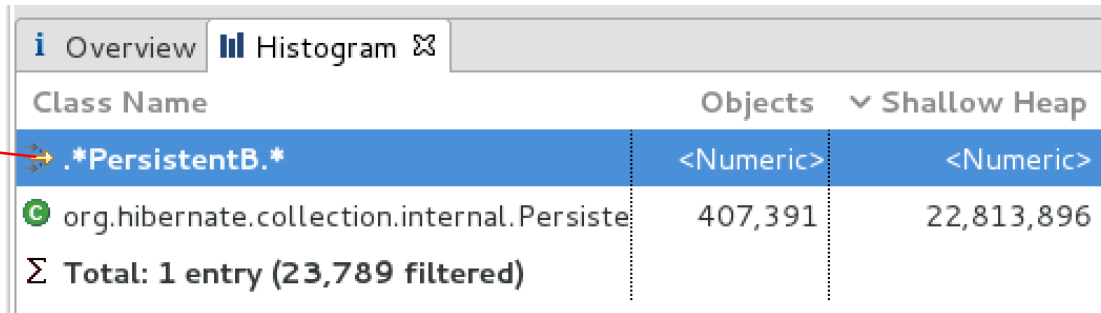
erster Heap zweiter Heap Vergleich

The screenshot shows the JVisualVM interface with two heap dumps loaded: 'dump1.bin' and 'dump2.bin'. The 'Overview' tab is selected, showing a table of memory usage. The table has three columns: 'Class Name', 'Objects', and 'Shallow Heap'. The class 'org.hibernate.collection.internal.PersistentBag' is highlighted, and its values are circled in red. The difference between the two heaps is also indicated by a red circle and the label 'Differenz'.

Class Name	Objects	Shallow Heap
<Regex>	<Numeric>	<Numeric>
java.util.HashMap\$Node	+396,857	+9,524,568
org.hibernate.collection.internal.PersistentBag	+328,524	+18,397,344
java.util.HashMap	+262,462	+10,498,480
java.util.HashMap\$Node[]	+244,302	+13,996,696
org.hibernate.collection.internal.PersistentSet	+219,016	+12,264,896
java.util.HashSet	+109,506	+1,752,096
java.util.HashMap\$KeySet	+109,502	+1,752,032
de.ejb3buch.ticket2rock.entity.Musiker	+83,958	+3,358,320
java.util.ArrayList	+76,561	+1,837,464
java.util.Collections\$SynchronizedRandomAccess	+65,718	+1,577,232
org.hibernate.context.CreationContextImpl	+51,114	+2,044,560

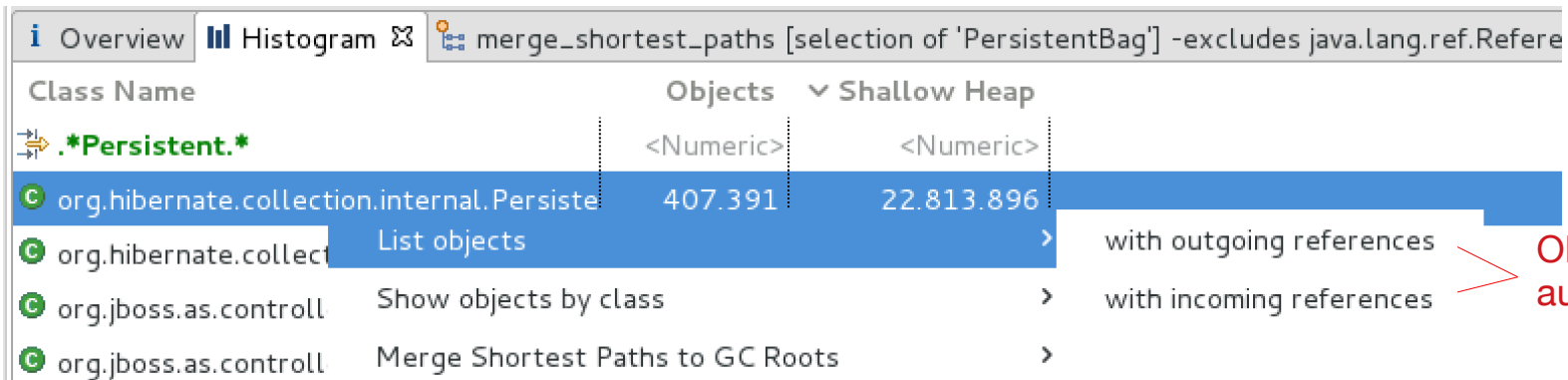
Analysen mit MAT

Filter



The screenshot shows the MAT Overview tab with a filter applied to the Class Name column. The filter is `.*PersistentB.*`. The table displays one entry: `org.hibernate.collection.internal.PersistentBag` with 407,391 objects and 22,813,896 bytes in the shallow heap. The total count is 1 entry (23,789 filtered).

Class Name	Objects	Shallow Heap
<code>.*PersistentB.*</code>	<Numeric>	<Numeric>
<code>org.hibernate.collection.internal.PersistentBag</code>	407,391	22,813,896
Σ Total: 1 entry (23,789 filtered)		



The screenshot shows the MAT Overview tab with the context menu open for the `org.hibernate.collection.internal.PersistentBag` class. The menu options are: List objects, Show objects by class, and Merge Shortest Paths to GC Roots. The 'Merge Shortest Paths to GC Roots' option is highlighted, and a red arrow points to it with the text 'Pfad zu Garbage Collection Root'. The 'List objects' option is also highlighted, and a red arrow points to it with the text 'Objekte auflisten'.

Class Name	Objects	Shallow Heap
<code>.*Persistent.*</code>	<Numeric>	<Numeric>
<code>org.hibernate.collection.internal.PersistentBag</code>	407.391	22.813.896
<code>org.hibernate.collection.internal.PersistentBag</code>	List objects	> with outgoing references
<code>org.jboss.as.controller</code>	Show objects by class	> with incoming references
<code>org.jboss.as.controller</code>	Merge Shortest Paths to GC Roots	>

Pfad zu Garbage
Collection Root

Objekte
auflisten

Garbage Collection Roots

A garbage collection root is an object that is accessible from outside the heap.

- **Java Local**

- Local variable. For example, input parameters or locally created objects of methods that are still in the stack of a thread.

- **System Class**

- Class loaded by bootstrap/system class loader. For example, everything from the rt.jar like `java.util.*`.

- **Thread**

- A started, but not stopped, thread.

Garbage Collection Roots

■ Busy Monitor

- Everything that has called `wait()` or `notify()` or that is synchronized. For example, by calling `synchronized(Object)` or by entering a synchronized method. Static method means class, non-static method means object.

■ JNI

- Local variable in native code, such as user defined JNI code or JVM internal code.
- Global variable in native code, such as user defined JNI code or JVM internal code.
- In or out parameters in native code, such as user defined JNI code or JVM internal code. This is often the case as many methods have native parts and the objects handled as method parameters become GC roots. For example, parameters used for file/network I/O methods or reflection.

Garbage Collection Roots

- **Thread Block**
 - Object referred to from a currently active thread block.
- **Finalizable / Unfinalized**
 - An object which is in a queue awaiting its finalizer to be run.
 - An object which has a finalize method, but has not been finalized and is not yet on the finalizer queue.

Java Memory: Referenz-Typen

■ Strong Reference

- Normale Referenz in Java
- Bsp: `Person p = new Person();`

■ Weak Reference

- Verhindert GC nicht. Wenn nur noch schwache Referenzen, wird trotzdem collected
- Bsp: `WeakReference<Person> wp = new WeakReference<Person>(p);`

■ Soft Reference

- Wie Weak Reference, nur wird erst dann collected, wenn Speicher knapp wird.

■ Phantom Reference

- Keine eigentliche Referenz auf ein Objekt. Kann verwendet werden um festzustellen, wann ein Object aus Memory entfernt wurde.

MAT Object Query

- Object Query Language

```
select * FROM de.ejb3buch.ticket2rock.entity.Band b  
where toString(b.name) = "Audioslave"
```