Practical Java Memory Management

Memory Management in real life

FrOSCon 2014

23.8.2014

Christian Esken

About me

Christian Esken

Doing OpenSource since 1996 (KDE project)

Java Architect at trivago® (Hotel Metasearch)

main()
Garbage collection in real life

Garbage Collection

Sometimes dirty and loud, but it has to be done.

Don't blame the garbage man!

main()
Whats the buzz?

1. #DEFINE

► What is a memory leak, anyhow?
► Types of memory / application taxonomy

2. ANALYZE()

► Calculating your memory requirements
► OpenSource Tools
► Finding Leaks

3. SOLVE{}

► Show techniques
► Dealing with the unavoidable … help the JVM
► Creative solutions, some outside the JVM

Extra

► No lengthy discussions about GC tuning (-XX...)

#DEFINE
What is a memory leak, anyhow?

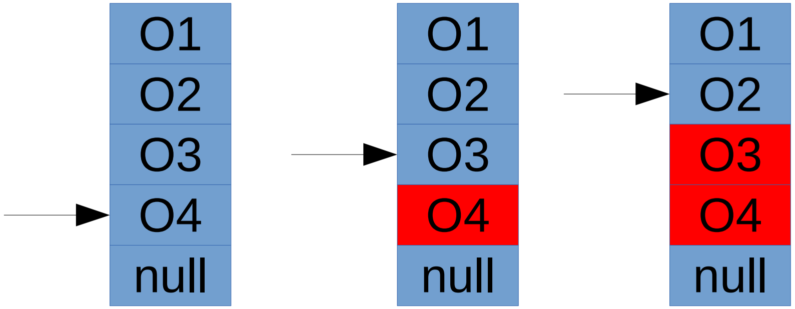
Object not reachable, but still allocated

- Classic leak in C, C++. Not possible in JVM

Object will never be referenced again by application code

- JEE: Classloader leak
- Stack implementation: pop() without null-ing reference

```
T pop()
{
  return stack[current--]; // leak
}
```



Adding to a Collection

- Add without removing
- Add rate exceeding remove rate

#DEFINE

How to create a memory leak: Part 1

```
static Map map = new HashMap();
public static void main(String[] args)
{
	changeRoom("Bath", "original");
	changeRoom("Bath", "renovated");
	System.out.println(set.size());    //  Output?
}

static void changeRoom(String key, String value)
{
	map.put(new Foo(key), value);
}
```

```
class Foo
{
	String key;
	Foo(String k) { key=k; }
 }
```

QUIZ:

Where is the leak?

Lets add an equals() to Foo:

```
public boolean equals(Object other) {
	return key.equals(((Foo)other).key);
}
```

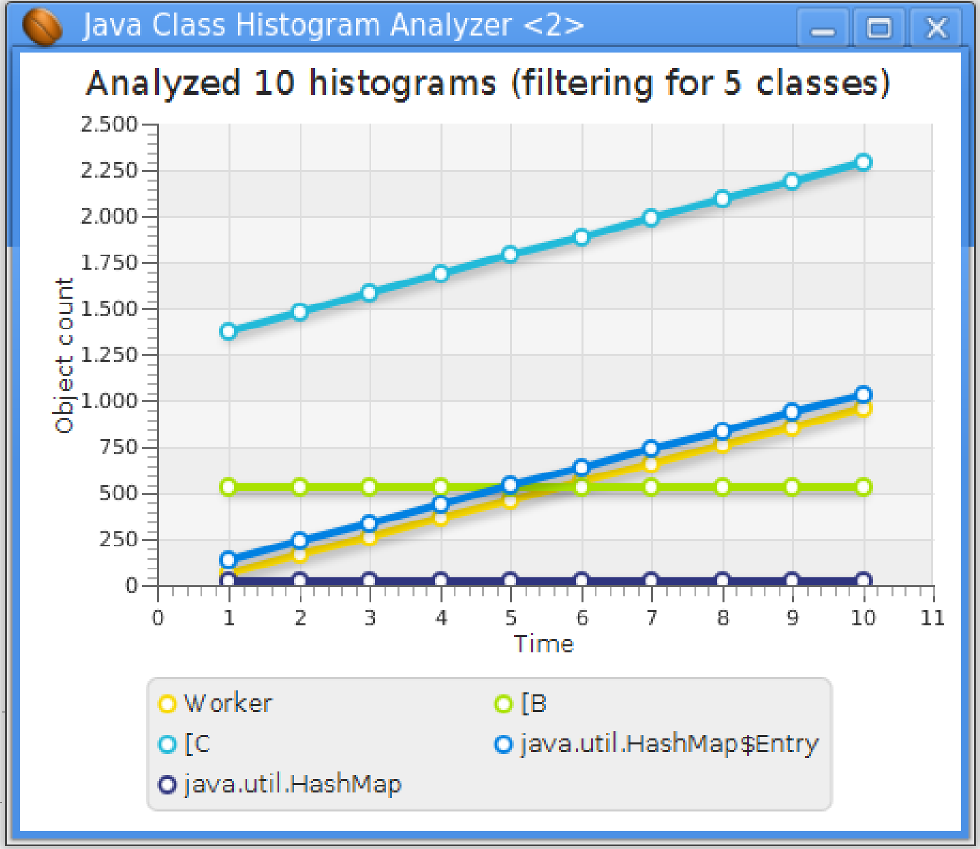
#DEFINE
How to create a memory leak: Part 2

```
enum HousePart
{ Door, Window, Roof, Floor, Plumbing;

 Set<Callable> workers = new HashSet<>();

 void register  (Callable w) { workers.add(w); }
 void unregister(Callable w) { workers.remove(w); }
}

abstract class Worker extends Callable
{
       String name;
       Worker(String n) { name = n; }

       void work(HousePart housePart)
       {
          housePart.register(worker);
          worker.call();  // do the work (abstract)
          housePart.unregister(worker);
       }
}
```

QUIZ:
Where is the leak?

#DEFINE
How to create a memory leak: Part 2

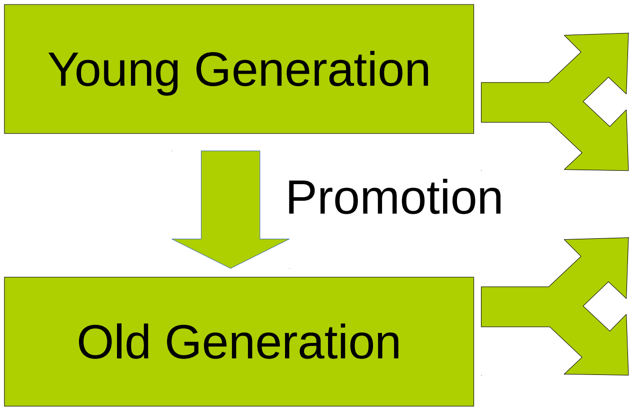
```
enum HousePart
{ Door, Window, Roof, Floor, Plumbing;

 Set<Callable> workers = new HashSet<>();

 void register  (Callable w) { workers.add(w); }
 void unregister(Callable w) { workers.remove(w); }
}

abstract class Worker extends Callable
{
    String name;
    Worker(String n) { name = n; }

    void work(HousePart housePart)
    {
      housePart.register(worker);
      worker.call();  // do the work (abstract)
      housePart.unregister(worker);
    }
}
```


Java Class Histogram Analyzer <2>

Analyzed 10 histograms (filtering for 5 classes)

#DEFINE
Memory types: Only The Good Die Young

Java Heap



Young Generation

Promotion

Old Generation

Example application: Webshop

Type	Lifetime	Tuning
Request based data (Stack)	short	Often unnecessary. Objects die young.
Cache	medium	Cache tuning Optimize data structures
Articles with description	long	Optimize data structures

analyze()
Do I have memory issues at all?

Part 2: analyze()

- ▶ Calculating memory requirements
- ▶ Understanding GC Logs
- ▶ Finding Leaks

analyze()
Calculating memory requirements

- ▶ Rough estimate:
  - *jconsole*
  - *jvisualvm*

- ▶ Where exactly?
  Measure Object graph
  - *twitter solution: ObjectSizeCalculator*
  - *Agent-based solutions*

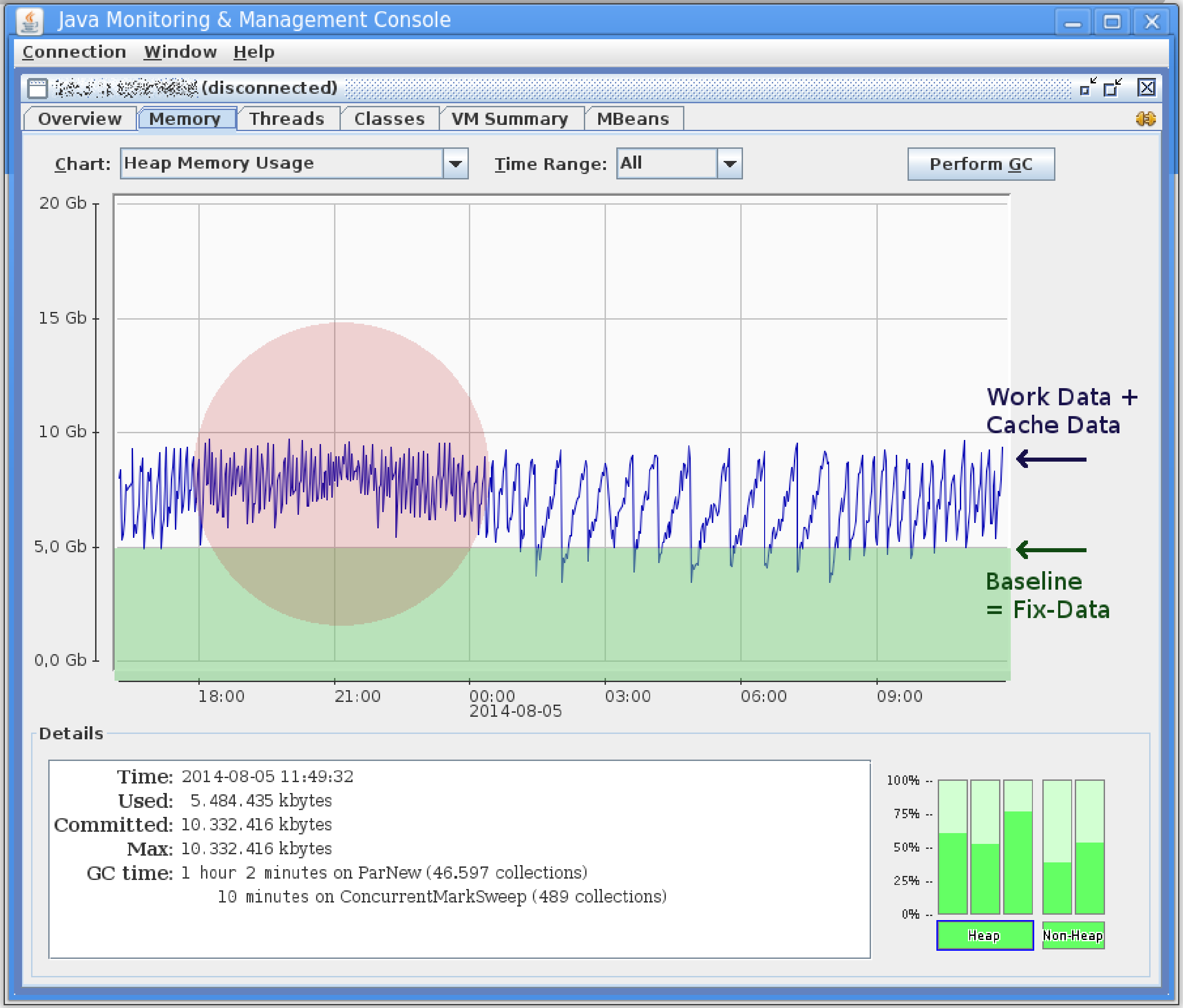


Chart: Heap Memory Usage

Time Range: All

Work Data + Cache Data

Baseline = Fix-Data

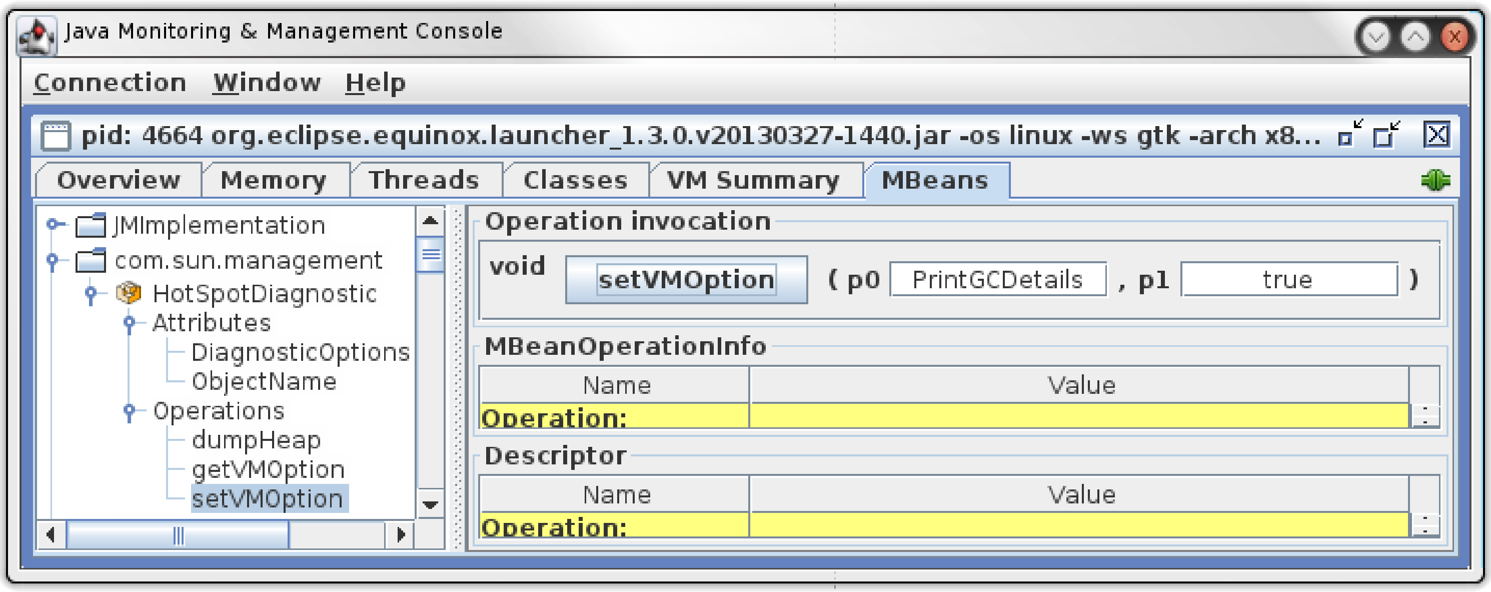
Details

```
      Time: 2014-08-05 11:49:32
      Used:  5.484.435 kbytes
 Committed: 10.332.416 kbytes
       Max: 10.332.416 kbytes
   GC time: 1 hour 2 minutes on ParNew (46.597 collections)
               10 minutes on ConcurrentMarkSweep (489 collections)
```

analyze()
Enabling GC Logs

- ▶ Static

-XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-XX:+PrintGCApplicationStoppedTime



pid: 4664 org.eclipse.equinox.launcher_1.3.0.v20130327-1440.jar -os linux -ws gtk -arch x8...

void setVMOption (p0 PrintGCDetails , p1 true)

- ▶ Dynamic:

  MBeans
  jinfo -flag +PrintGCDetails <pid>

analyze()
Understanding GC Logs

- Manual checks

  Look for long stopped threads

  Look for Full GC (stop the world!!!)

- GCViewer

  A tool that reads GC logfiles

  Does all the statistics Voodo for you (stddev,…)

  Has a very colorful GUI

analyze()
Finding Leaks

- ►jhat / jmap :
  OpenSource, but limited: GUI, Drilldown, …
  Alternative : Commerical Profilers (YourKit, Jprobe u.ä.)

- ►jcmd – Swiss Army Knife
  Heap dump
  Thread Dump
  Class Histogram

- ►jcha – Java Class Histogram Analyzer
  New tool … public debut at FroSCon 2014
  Based on jcmd output
  Concentrates on leak analysis

analyze()
jcha - Java Class Histogram Analyzer - Live Demo

Run a cronjob to capture:

```
# jcmd <pid> GC.class_histogram
   > srv-<date-time>.classhist
```

Time-base capturing script:

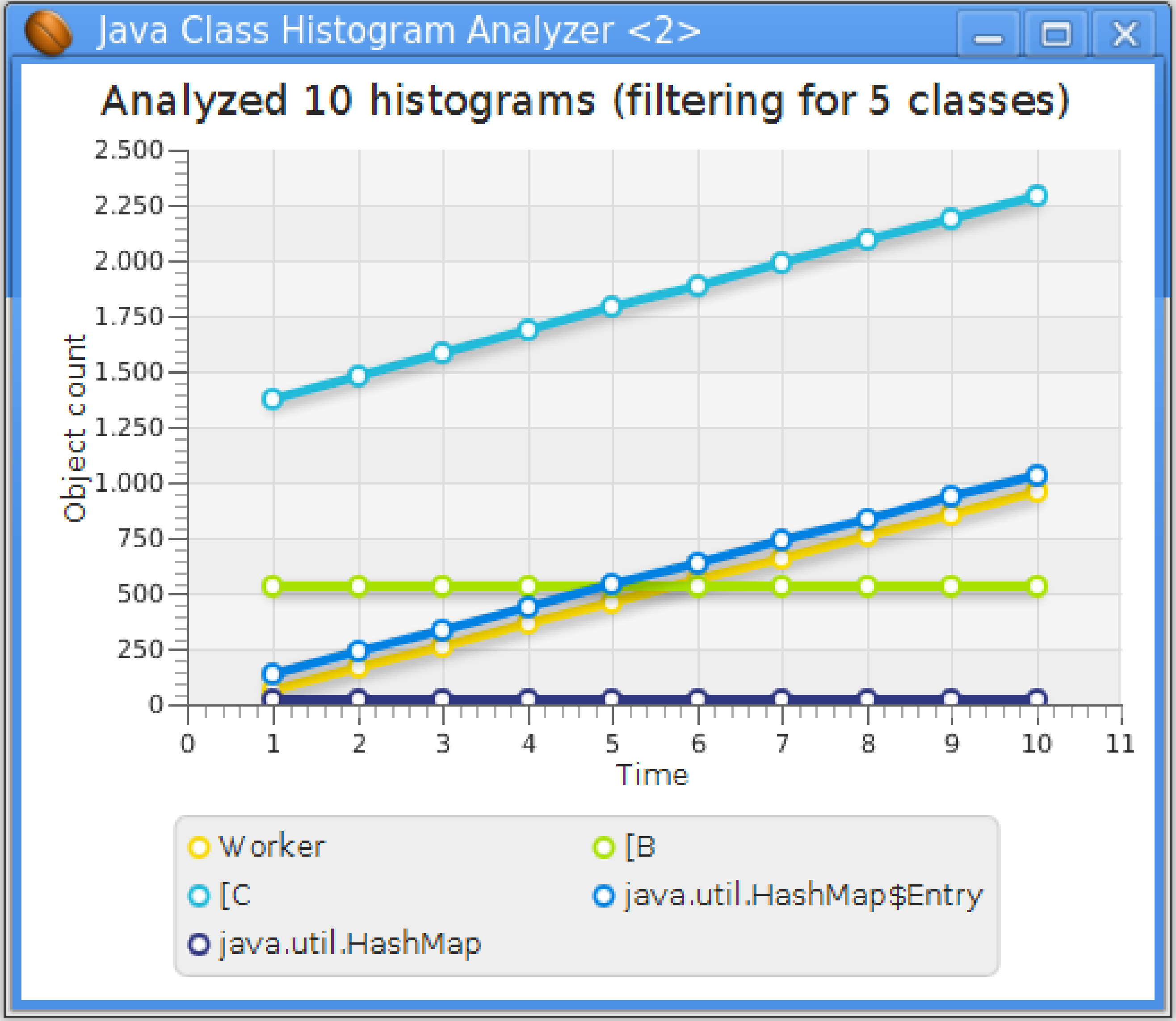
```
# jcha-capture <pid> fileprefix
```

Viusalize histograms:

```
# jcha-gui *.jch
```

Console version:

```
# jcha *.jch
```



Java Class Histogram Analyzer <2>

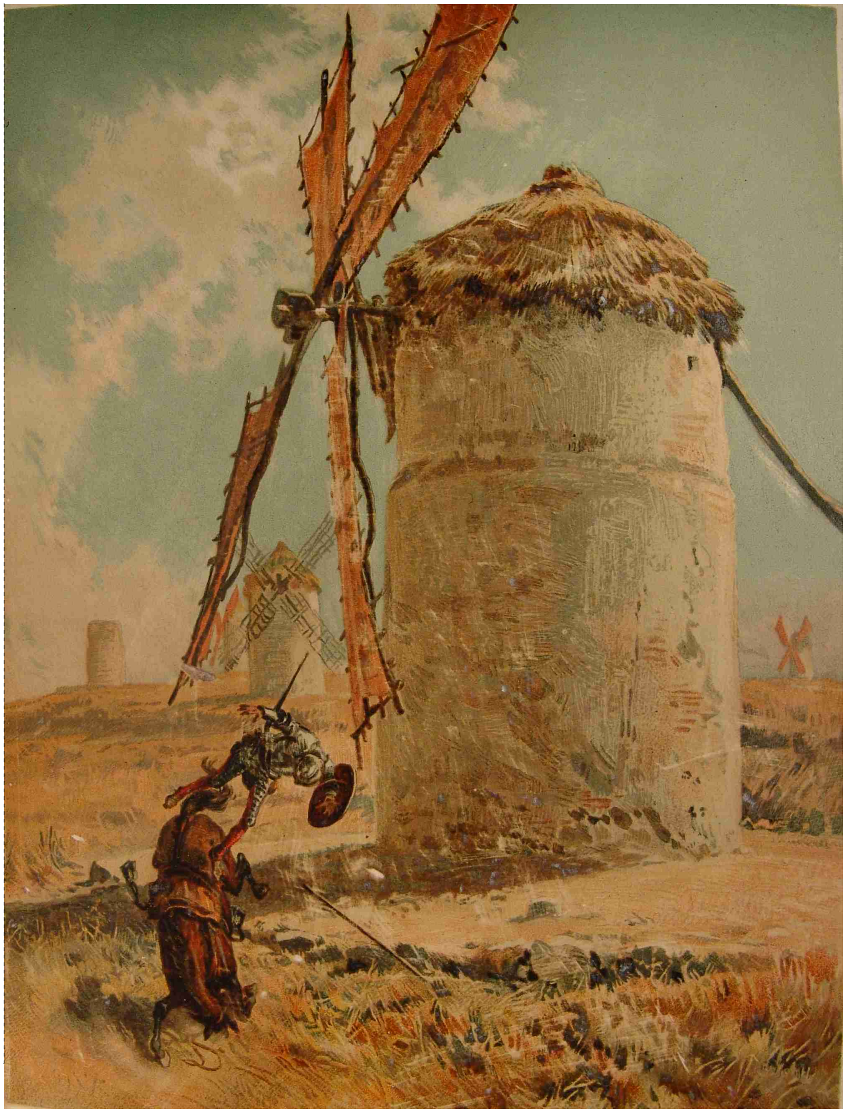
Analyzed 10 histograms (filtering for 5 classes)

SOLVE{}

Help the Collector – but do not fight Windmills

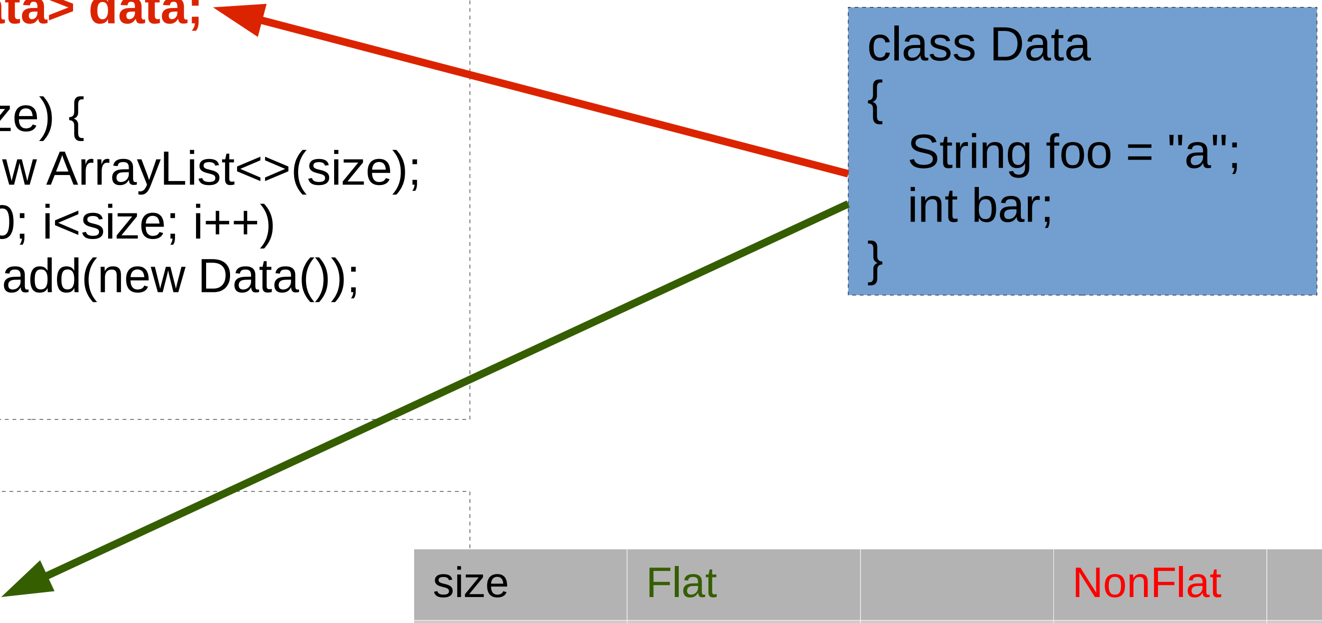
Part 3: SOLVE{}

- Class design has impact
- Technology has impact
- Caching = Fighting Windmills



SOLVE{}
Class design has impact: Example „Object flattening“

```
class NonFlat {
        ArrayList<Data> data;

        NonFlat(int size) {
                data = new ArrayList<>(size);
                for (int i=0; i<size; i++)
                        data.add(new Data());
        }
}
```



```
class Data
{
   String foo = "a";
   int bar;
}
```

```
class Flat {
        String foo[] ;
        int bar[];

        Flat(int size)  {
                foo = new String[size];
                Arrays.fill(foo, "a");
                bar = new int[size];
        }
}
```

size	Flat		NonFlat	
	Bytes	Objects	Bytes	Objects
1000	8104	5	28104	1.005
100.000	800104 (0,8MB)	5	2800104 (2,8MB)	100.005
10 Mio	80000104 (80 MB)	5	280000104 (280MB)	10.000.005 (10Mio)

SOLVE{}
Technology has impact: Stream your data …

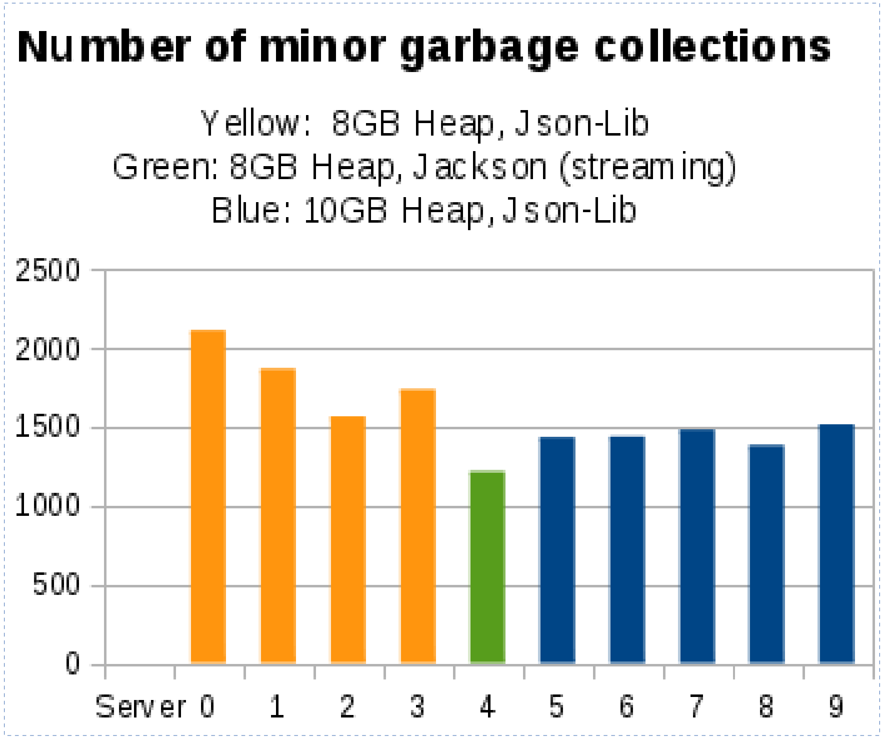
```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
{
    Object result = getResult(req);                              // 1 - Object
    String json = jsonMapper.writeValueAsString(result);         // 2 - String
    byte[] resultAsByte = json.getBytes("UTF-8");                 // 3- as byte[]
    response.setContentLength(resultAsByte.length);
    response.setContentType("application/json; charset=UTF-8");
    response.getOutputStream().write(resultAsByte);
}
```

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
{
    Object result = getResult(req);   // 1 - Object
    response.setContentType("application/json; charset=UTF-8");
    jsonMapper.writeValue(response.getOutputStream(), result);
}
```

SOLVE{}
Stream your data … results

Real-world application, processing lots of JSON:

- Streaming saved
  -33% in Minor Garbage Collections
- Increasing Heap by 2GB saved
  -20% in Minor Garbage Collections
- Combining both:
  -50% in Minor Garbage Collections



SOLVE{}
Creative or evil solutions and workarounds

If you cannot avoid GC stop-the-world:

► Not evil: Be stateless and run a cluster!

► Not evil: Fail-Fast. e.g. let the client switch quickly to another server
- Async design, and low timeouts (answer quick, possibly with „ask later“)
- No TCP backlog
- Low number of connections

Example for Tomcat: `<Connector port="8080" backlog="0" maxThreads="40"/>`

► Evil: Run GC at „harmless“ times, or even restart complete JVM.

► Less evil: Monitor your servers, and restart if service quality goes down.

finally{}

What you should take out of this talk:

► Don't blame the Garbage Collector (at least not initially)

► Reduce memory baseline and object count

► Accept Major Collections if you cache data

► Do only minimal GC tuning (-Xmx, -Xms, Old:New-Ratio, Survivor space)

System.exit()

Christian Esken

Java Class Histogram Analzer ▶ https://github.com/trivago/jcha

Slides and material ▶ https://github.com/cesken

trivago

Booth at FrOSCon 2014 ▶ Come, take a look, have a (tech) talk

trivago hires ▶ http://www.trivago.com/jobs