Search for the Memory Duplicities in the Java Applications Using Shallow and Deep Object Comparison

RICHARD LIPKA, TOMÁŠ POTUŽÁK

FedCSIS 2019, 2. SEPTEMBER
Memory issues in Java

**Memory leaks**, real ones, are rare, as a garbage collection should prevent them completely.

**Memory bloat** (*Mitchell, 2010*) is common, as programmers often do not pay enough attention to the design of their programs:
- Collections are misused or left empty
- Null pointers can occupy significant amount of space
- Automated layers are creating instances without much control of the programmers
- Duplicitous instances occupy memory

Documented in real software, common in students projects.
Duplicities in memory

**Duplicities** (or clones) are often looked up in the source codes, as a well known source of problems

- But they can exist in the heap memory as well
- Causing similar issues – data consistency, security, performance

**Garbage collection** should be able to remove unnecessary instances

- But it is based only on the existence or non-existence of the reference → when programmer (or some automated layer) keeps references, GC cannot work properly
- Costs time, so the programs with large memory footprint tends to run slower

? How common is this problem?

? Can the identical instances be merged into one?
Causes

We do not really know, but there are some suspicions:

◦ ? Fast development using automation tools ?
◦ ? Lack of attention to the program design ?
◦ ? Lack of experience ?
◦ ? Relying on the magic of the garbage collection ?
Automated solution in virtual machine?

**Strings** are deduplicated automatically

- **They are final** – after creation cannot be changed $\Rightarrow$ no problems with copies intended for the change in the future
- **They are simple** – virtual machine can easily compare them

What about complex objects?

- There are proposal in the literature, but no implementation
- Runtime analysis of identical instances is time consuming, the time it takes is difficult to predict as the classes can be arbitrary complex
Analysis of the memory

Too expensive to perform on the runtime, but can be done on the stored heap dumps
  ◦ Java can safely store heap content on the disk in any time
  ◦ Search for duplicities is more troubleshooting, performed only when needed

**Managed memory** makes analysis of the heap much easier – memory contains not only data but also the description of the structures
  ◦ The same approach for C programs is a significant challenge, structure understandable only to the program itself
What makes instances identical?

Operator == compares only the references → useless for our purpose

equals() method can be implemented in any way

→ we need to compare instances attribute by attribute

◦ Identical data in each attribute = identical instances
◦ Comparison only within one class?

Class A
attr_1: int
attr_2: String

equal classes
different classes

Class B
attr_1: int
attr_2: String

Class C
attr_1: int
attr_2: String
attr_3: int
How to deal with references?

**Shallow comparison** deals only with the attribute values
- But is much faster and performed only within one class

Equal instances

Different instances
How to deal with references?

**Deep comparison** compares the whole structures
- The analysis has to be performed recursively
- Can be very time demanding – especially with arrays or collections
- Cycles have to be broken – graph transformed to spanning tree
Comparison within classes

Identical instances analysed within one class – shallow comparison
- Complexity $O(n^2)$, but reduced $n$
  (only within one class, comparison stops after first difference is found)

Deep comparison in two steps
- Shallow comparison to prepare information about identical attributes
- Then comparison of the graph structures
Experiments

**Simple application** for verification
- Known data structures and number of duplicities

*Spring Boot framework* (2.1.4) with Hello World application

*Eclipse* (4.10.0) with one project in workspace, just after starting

*IntelliJ Idea* (2018.3)

*TomEE* with complex graph analysing application

Memory dump obtained using

```
jmap -dump:live,
  file = <file-path>
  <pid>
```

Should provide memory content after GC
Results – complexity (simple application)
Results – Spring boot

<table>
<thead>
<tr>
<th>Package name</th>
<th>Classes</th>
<th>Instances</th>
<th>Found duplicates</th>
<th>Duration [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>2416</td>
<td>9093</td>
<td>347</td>
<td>14759</td>
</tr>
<tr>
<td>org.springframework</td>
<td>1555</td>
<td>6053</td>
<td>329</td>
<td>8214</td>
</tr>
<tr>
<td>org.springframework.boot</td>
<td>380</td>
<td>1506</td>
<td>27</td>
<td>4229</td>
</tr>
<tr>
<td>org.springframework.core</td>
<td>196</td>
<td>1585</td>
<td>5</td>
<td>4425</td>
</tr>
<tr>
<td>org.springframework.web</td>
<td>296</td>
<td>239</td>
<td>37</td>
<td>4108</td>
</tr>
<tr>
<td>org.springframework.boot.web</td>
<td>75</td>
<td>27</td>
<td>1</td>
<td>4002</td>
</tr>
</tbody>
</table>

27 MB of data, only org. package analysed

**Signature** class - 38 identical instances (duplicates in table – at least two clones)

**DefaultFlowMessageFactory** class - 34 identical instances.
Results – IntelliJ Idea

<table>
<thead>
<tr>
<th>Package name</th>
<th>Classes</th>
<th>Instances</th>
<th>Found duplicates</th>
<th>Duration [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>2016</td>
<td>157743</td>
<td>283</td>
<td>8425230</td>
</tr>
<tr>
<td>com</td>
<td>7687</td>
<td>77927</td>
<td>261</td>
<td>1290908</td>
</tr>
<tr>
<td>sun</td>
<td>1119</td>
<td>15620</td>
<td>31</td>
<td>26023</td>
</tr>
</tbody>
</table>

74 MB of data, packages listed in the table analysed

`org.jdom.Text` – several instances with many clones
(largest one – 11577 identical instances, several characters from DOM of the loaded project)
# Results - Eclipse

<table>
<thead>
<tr>
<th>Package name</th>
<th>Classes</th>
<th>Instances</th>
<th>Found duplicates</th>
<th>Duration [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>org</td>
<td>9647</td>
<td>141970</td>
<td>756</td>
<td>5007822</td>
</tr>
<tr>
<td>com</td>
<td>919</td>
<td>27906</td>
<td>865</td>
<td>90271</td>
</tr>
<tr>
<td>java</td>
<td>1155</td>
<td>313405</td>
<td>39</td>
<td>23596884</td>
</tr>
<tr>
<td>sun</td>
<td>929</td>
<td>28092</td>
<td>20</td>
<td>91228</td>
</tr>
<tr>
<td>ch</td>
<td>244</td>
<td>539</td>
<td>5</td>
<td>7335</td>
</tr>
</tbody>
</table>

92 MB of data, packages listed in the table analysed

- `org.eclipse.swt.widgets.TypedListener` - 444 identical instances
- `org.eclipse.sisu.plexus.ConfigurationImpl` - 16 identical instances, each 750 characters of XML fragment
Results – **TomEE** with visualisation server

Only domain objects of the application analysed

Largest heap dump (about 370 MB, only shallow comparison took about 3 hours)

3 identical graph structures hold in memory for each session + identical data in two sessions
Conclusion

Main contribution – prototype of the analysis tool
  ◦ Can work as additional support to the memory profilers

Confirmation of the existence of the clones in real programs

Future work
  ◦ Parallelisation of the comparison algorithm
    (current implementation is quite slow)

  ◦ Detection of the real causes of the duplicate existence – analysis on runtime?
  ◦ Advice if the instances can be merged – analysis on runtime?
Thank you for your attention

Questions?