New Technologies for the Information Society
Research Center
University of West Bohemia in Pilsen

FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

RELISA
Reliable Software Architectures
research group

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 projetcs

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 → 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

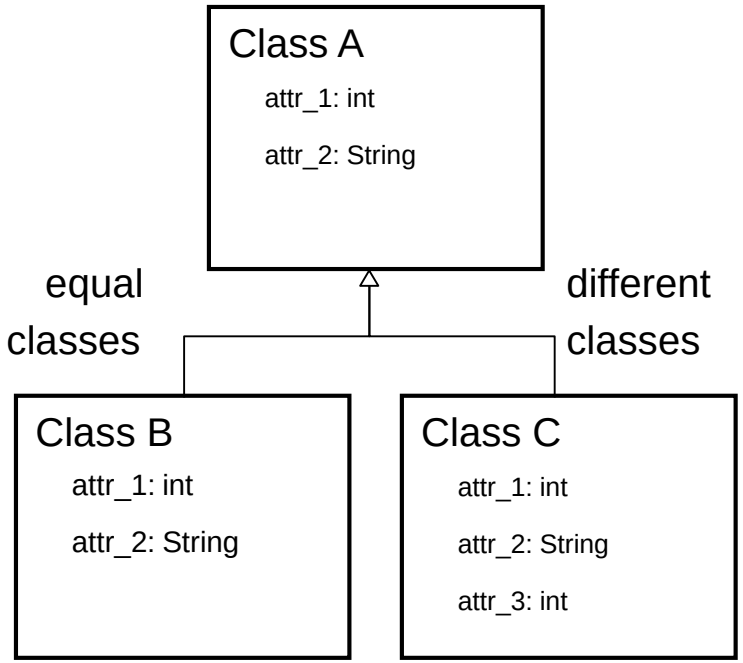
What makes instances identical?

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

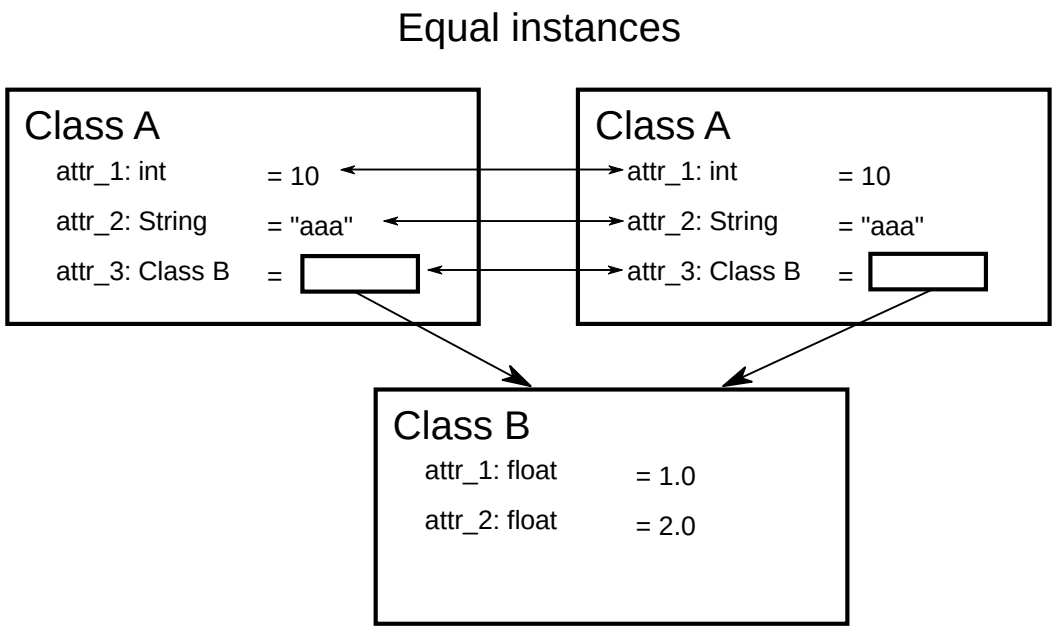
- Identical data in each attribute = identical instances
- Comparison only within one class?

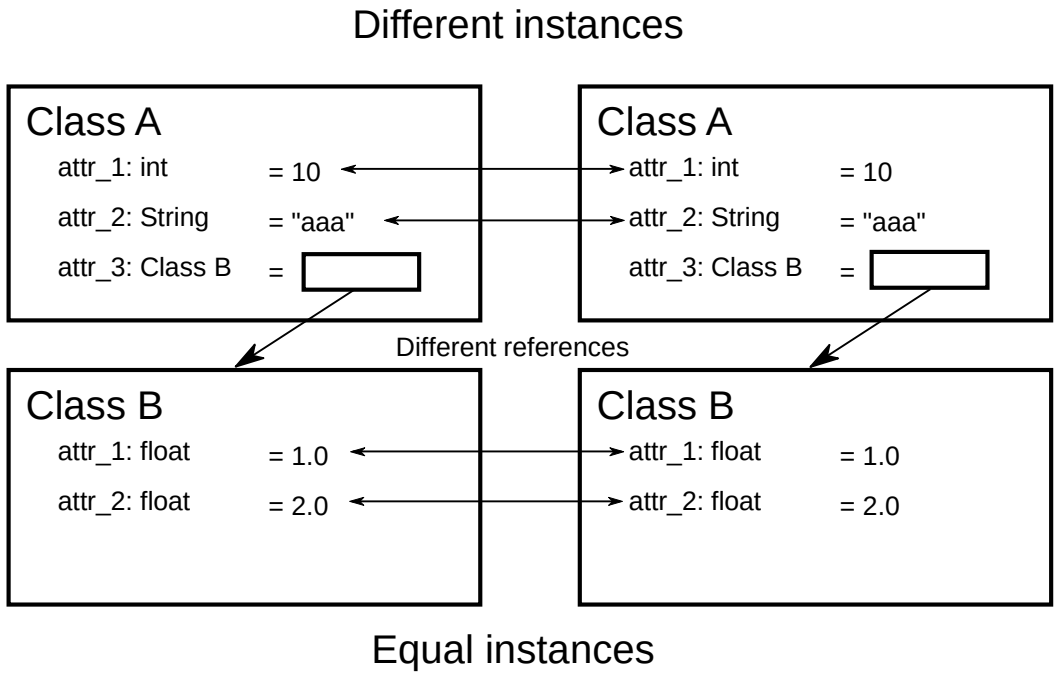


How to deal with references?

Shallow comparison deals only with the attribute values

- But is much faster and performed only within one class

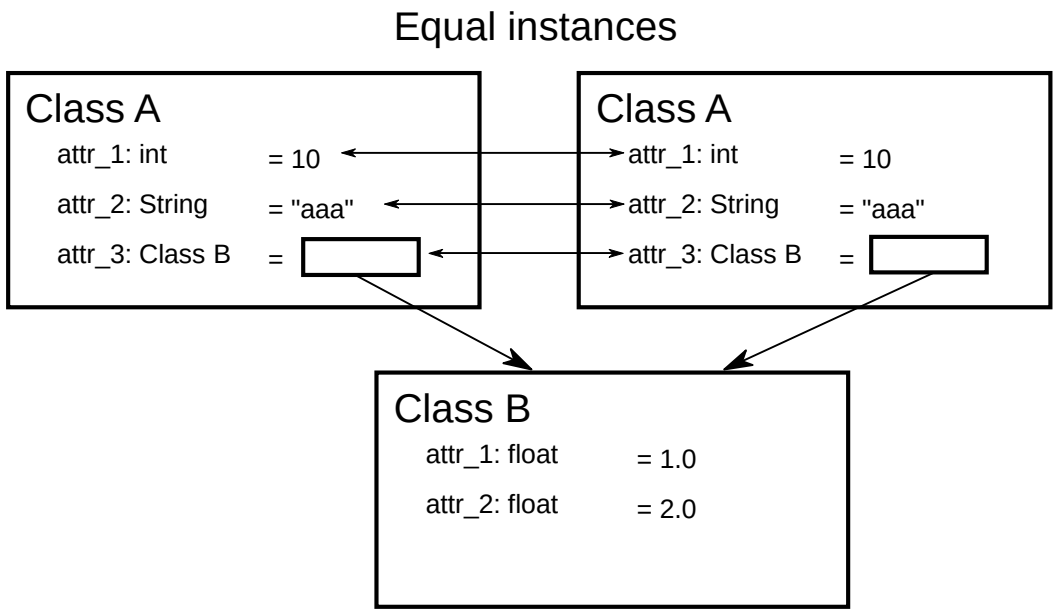


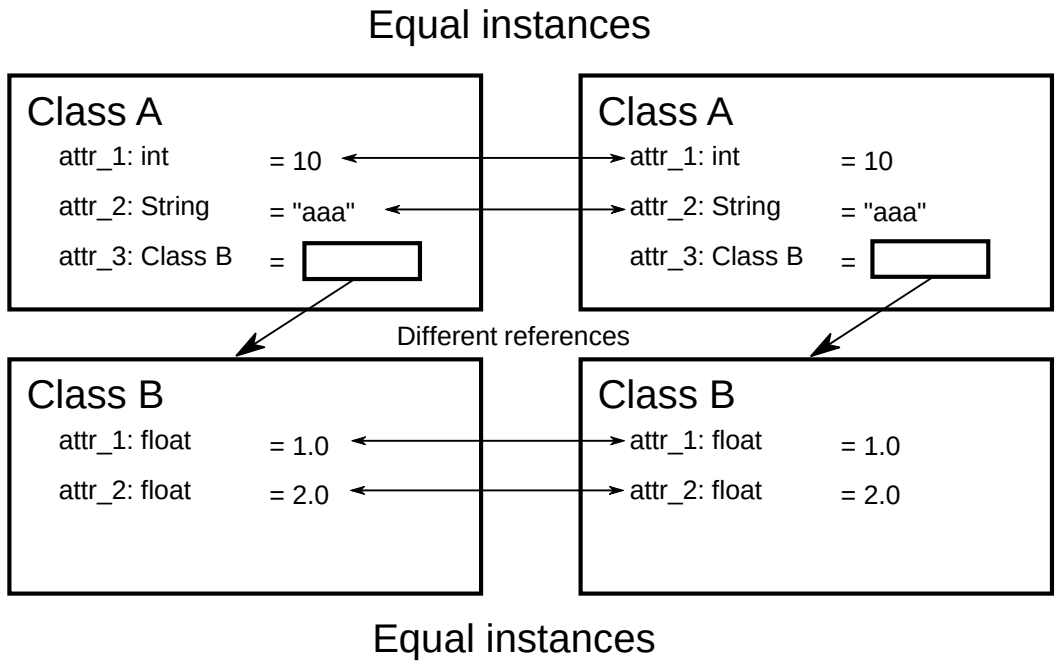


How to deal with references?

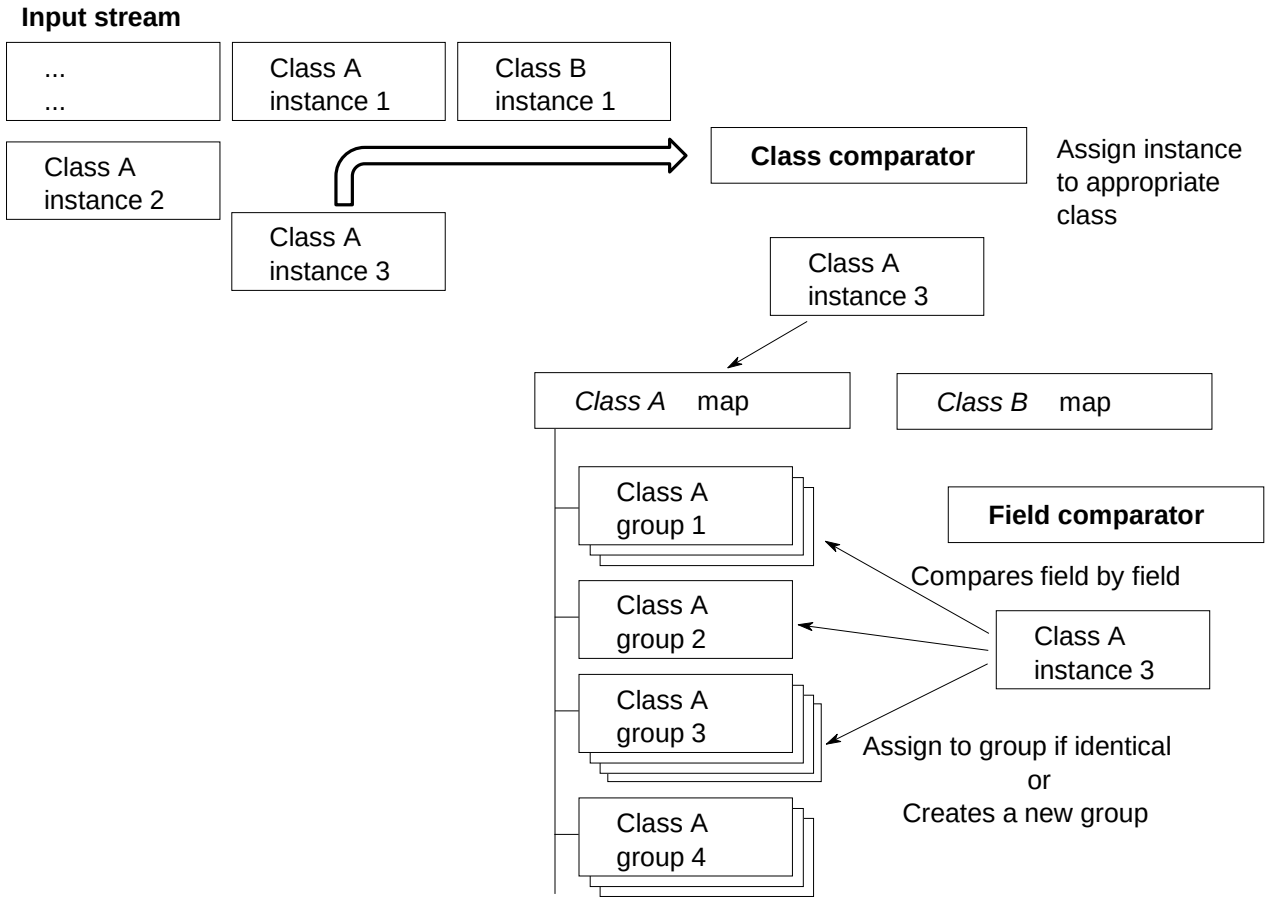
Deep comparison compares the whole structures

- 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)

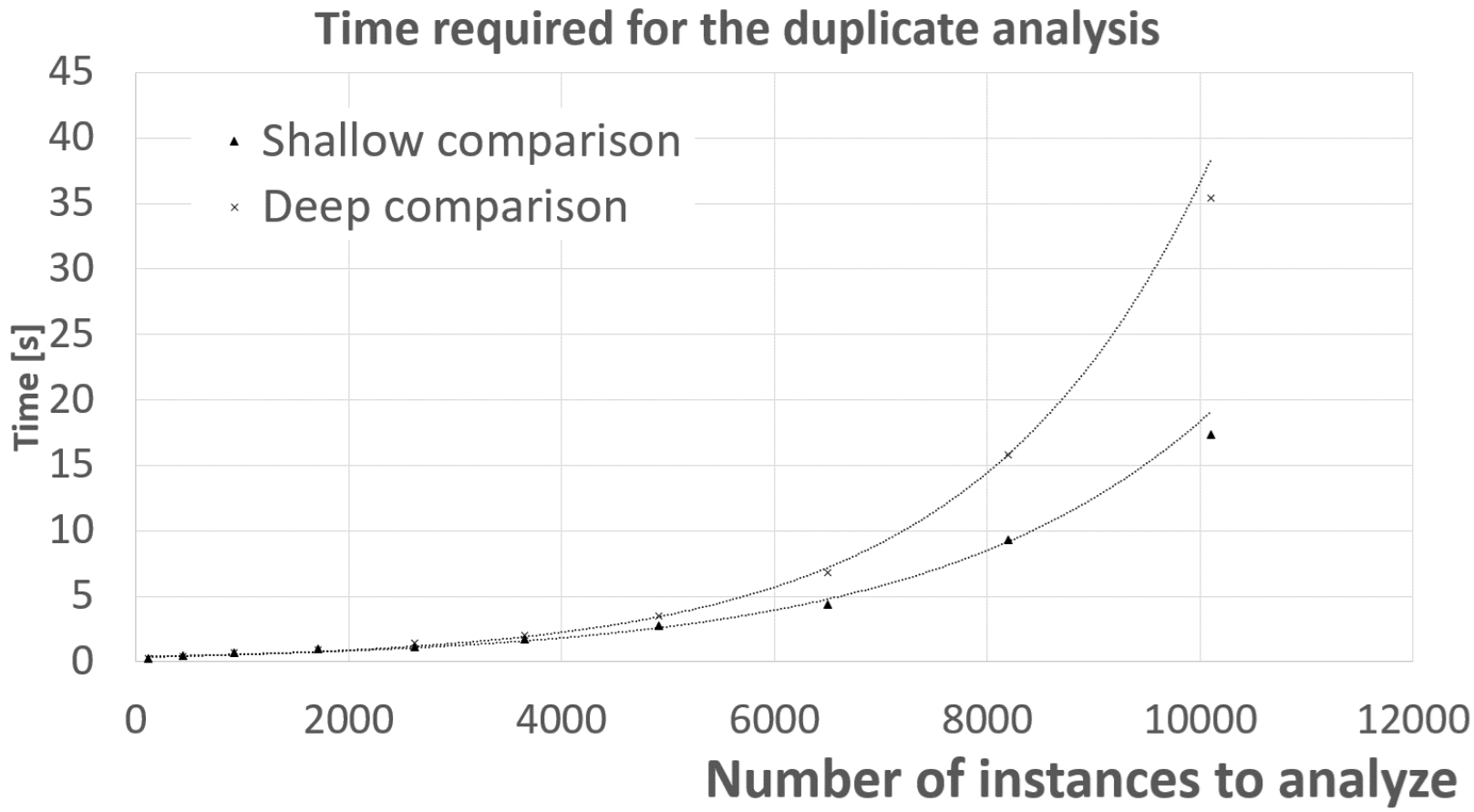
TomEE with complex graph analysing application

Memory dump obtained using

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

Should provide memory content after GC

Results – complexity (**simple application**)



Results – Spring boot

Package name	Classes	Instances	Found duplicates	Duration [ms]
org	2416	9093	**347**	14759
org.springframework	1555	6053	**329**	8214
org.springframework.boot	380	1506	**27**	4229
org.springframework.core	196	1585	**5**	4425
org.springframework.web	296	239	**37**	4108
org.springframework.boot.web	75	27	**1**	4002

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**

Package name	Classes	Instances	Found duplicates	Duration [ms]
org	2016	157743	283	8425230
com	7687	77927	261	1290908
sun	1119	15620	31	26023

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**

Package name	Classes	Instances	Found duplicates	Duration [ms]
org	9647	141970	**756**	5007822
com	919	27906	**865**	90271
java	1155	313405	**39**	23596884
sun	929	28092	**20**	91228
ch	244	539	**5**	7335

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 o runtime?
- Advice if the instances can be merged – analysis on runtime?

Thank you for your attention

Questions?