# JPoint 2015

## *Conference overview*

Jpoint

# Agenda

- ✓ *Memory leaks profiling basics*
- ✓ *Some notes about java.lang.String class*
- ✓ *What technical debt is. How to measure it and how to sell.*
- ✓ *Regular expressions under hood*
- ✓ *Conclusion*

# Memory Leaks

SPERASOFT

# JVM and Memory

- Different regions of memory
- Default sizes
  - java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal –version
- Non-default sizes
  - -Xmx2g -Xms2g -XX:MaxPermSize=128m
  - -XX:MaxPermSize=128m
- Garbage Collector
  - JVM subsystem responsible for object allocation and removing
- OutOfMemoryError

# Memory Leaks: Unused Objects

- Unused object - object with no reference on it
- References:
  - From an object to the value of one of its instance fields
  - From an array to one of its elements
  - From an object to its class
  - From a class to its class loader
  - From a class to the value of one of its static fields
  - From a class to its superclass
- GC roots - special objects, always considered to be alive
- Reachability

# Memory Leaks: Examples

- Memory Leaks
  - Caches without look-ups and eviction
  - Immortal threads
  - Unclosed IO streams
  - String.substring()
- NOT Memory Leaks
  - Too high allocation rate
  - Cache with wrong size
  - Trying to load too much data at once
  - Fat data structures

# Memory Leaks: GC Logs

- JVM Options
  - -XX:+PrintGCDetails
  - -XX:+PrintGCTimeStamps
  - -Xloggc:file.log
  - -XX:+UseGCLogFileRotation
  - -XX:NumberOfGClogFiles=N
- GC Logs Analyzers
  - GCViewer
  - Fasterj

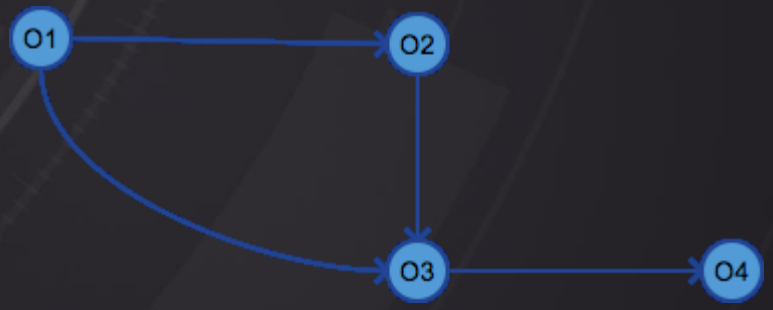
# Memory Leaks: Memory Dump

- Binary representation of objects graph written to a file
- One of the best ways to find out what consumes memory
- Getting memory dump:
  - jmap -dump:format=b,file=heap.hprof

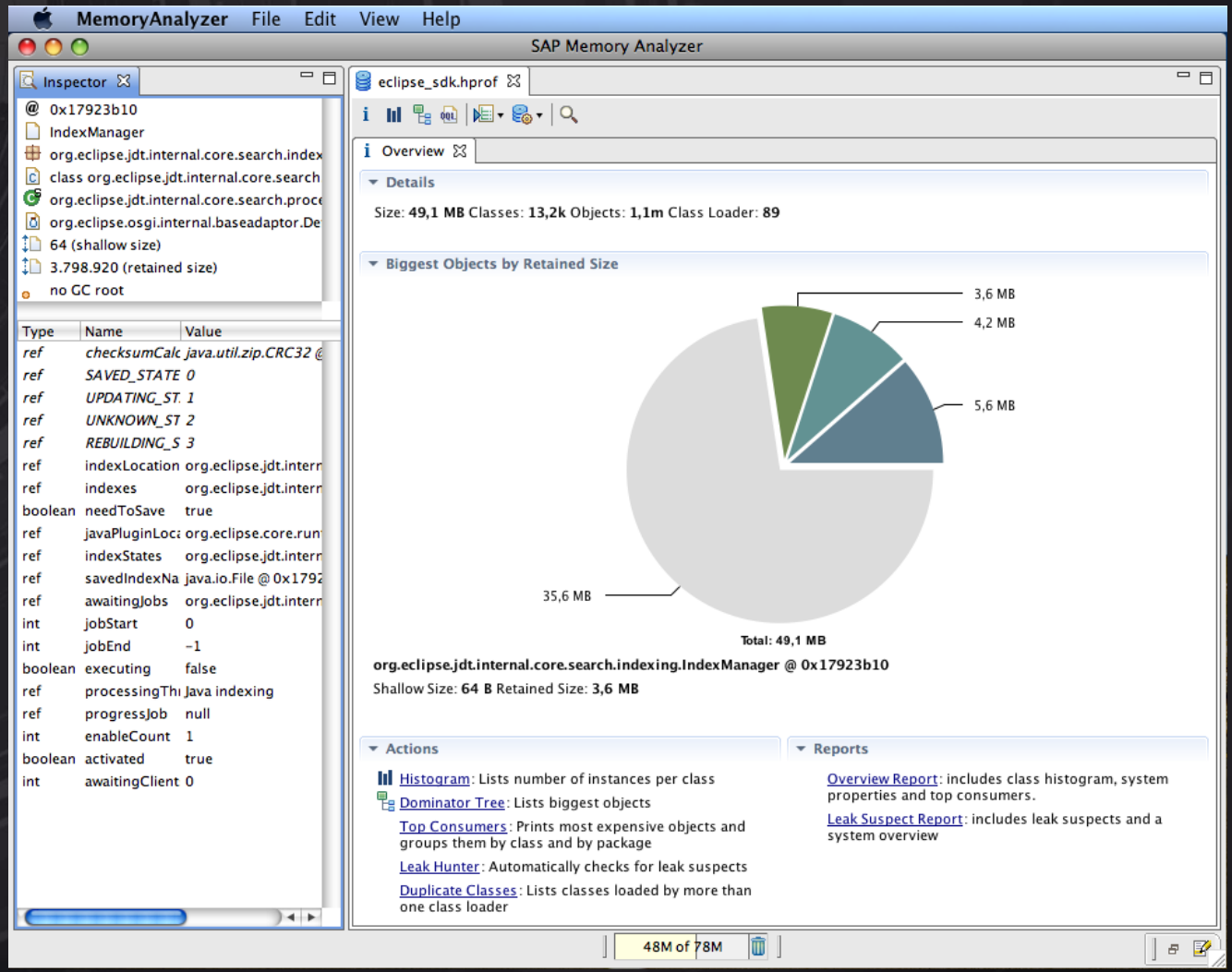
    (!) Stop The World, 1 Thread, 4Gb ~4 sec
  - -XX:+HeapDumpOnOutOfMemoryError
  - -XX:HeapDumpPath=./java_pid<pid>.hprof
- Get as late as possible
- Memory dump analyze tool:
  - MAT

# Memory Leaks: Object Size

- Shallow – size of the object itself with headers and fields but without fields values
- Deep – size of the object itself with headers, fields and fields values
- Retained – the sub-graph size that will be unreached if we remove the object



# Memory Leaks: MAT




SPERASOFT

# Some Notes about java.lang.String

SPERASOFT

# String

- Immutable
- 25% of objects are java.lang.String
- String concat vs StringBuilder (3250 op/sec vs 125000 op/sec)
- -XX:±OptimizeStringConcat

| OFFSET | SIZE | TYPE | DESCRIPTION |
|---|---|---|---|
| 0 | 12 | | object header |
| 12 | 4 | char[] | value |
| 16 | 4 | int | hash |
| 20 | 4 | | alligmeent loss |

# String: Notes

- (“” + x) is faster then Integer.toString(x) (!) -XX:+OptimizeStringConcat
- with side effects optimization is not working
- string in JDK8 have no seek and count properties
- do not use .intern() method as cache
  - StringTable is overflowed and can not be resized
  - -XX :+ PrintStringTableStatistics
- equals implementation is intrinsic function (chose by compiler)
  - -XX:+UnlockDiagnosticVMOptions -XX:DisableIntrinsic=::_equals

# String: HashCode Example

```
String str1 , str2 ;

public void setup () {
        str1 = “superinstrument␣neglected"; // same length
        str2 = " neglected␣ superinstrument "; // same length
}

int test1 () { return str1 . hashCode (); }

int test2 () { return str2 . hashCode (); }
```

# String: HashCode

- Uses polynomial  function
- Collisions (the same result for different values)

```
public int hashCode () {
   int h = hash ;
   if (h == 0) {
      for ( char v : value ) {
         h = 31 * h + v;
      }

      hash = h;
   }

   return h;
}
```

# String: Regexp

```
String text = “Funny␣girl␣danced␣fell␣hurt␣and␣cried␣aloud.";
String textDup = text.replaceAll ("␣", "␣␣");
Pattern pattern = Pattern.compile ("␣␣");

String [] charSplit () { return text.split ("␣"); } //191.6 ns/op

String [] strSplit () { return textDup.split ("␣␣"); } //527.9 ns/op

String [] strSplit_pattern () { return pattern.split(textDup); } //416.2 ns/op
```

- charSplit: fast path for 1 byte symbols
- strSplit: internally creates Pattern
- strSplit_pattern: reuse Pattern

# Technical debt

SPERASOFT

# Technical Debt

- Try to not create a technical debt
  - Code review
  - Precision estimates
- Bus factor (save information)
- Try to measure the date your application will start suffer from this debt
  - Collect metrics as much as possible
  - Try to build trend
- Fishy code

# Technical Debt: Selling

- Include small refactoring in estimates
- Formal code quality metrics (clear for you, not so clear for customer)
- Developers group motivation (to not lose)

# Regular expressions under hood

SPERASOFT

# Regular Expressions: Backtrack

- Backtrack
  - /^.*ab$/  matches “ab”
  - /^.*?b$/  matches “aab”
  - can took 30 sec in 100Kb text
- Pattern.compile(**"(Joker|JPoint)+"**)
  - String "JokerJPoint…700 times…Joker“ will cause StackOverflowError
  - Solutions:
    - Pattern.compile(**"(Joker|JPoint)++"**) – possessive
    - Pattern.compile(**"(?>Joker|JPoint)+"**) – forget back-reference

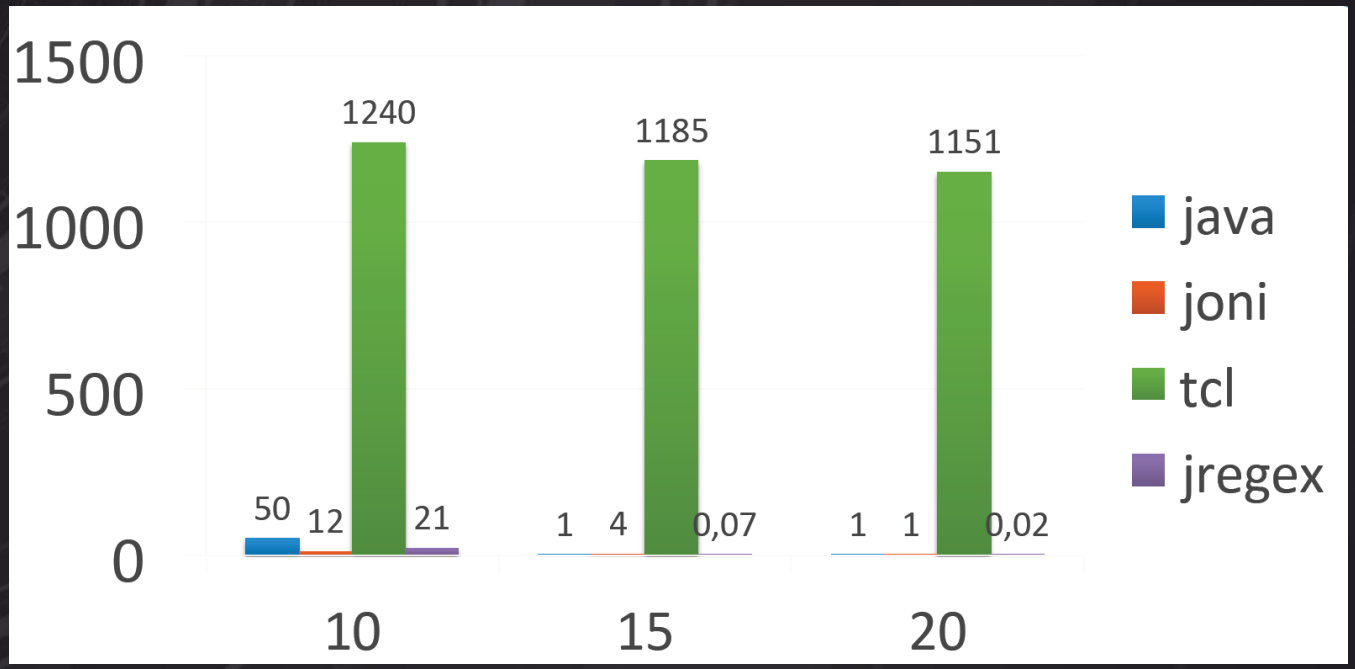
# Regular Expressions: Impl

- Deterministic automa
- Non-deterministic automa

| TITLE | RELEASED | ALGORITHM |
|---|---|---|
| java.util.regex.Pattern | 2015 | Backtrack |
| com.basistech.tclre | 2015 | DFA |
| org.jruby.joni (Nashorn) | 2014 | Backtrack |
| Apache Oro | 2000 | Backtrack |
| JRegex | 2013 | backtrack |

# Regular Expressions: Test

- /(x+x+)+y/.test("xxxxx……xxx")



# Regular Expressions: Tradeoff

- Java pattern
  - Non-deterministic response time
  - Managed backtrack
  - Low memory usage
- TCL
  - DFA algorithm with deterministic response time
  - High memory usage

# Thanks!

Visit Sperasoft online:

www/sperasoft.com
www.facebook.com/Sperasoft
www.twitter.com/sperasoft

SPERASOFT