Profilowanie

Czy w Javie są wycieki pamięci?

Wyciek pamięci

- Program alokuje pamięć, a jej nie zwalnia.
- Bardzo powszechne w C/ C++:

```
int fun() {
  int* wsk = new int; // zarezerwowane pamięci na liczbę typu int
  *wsk = 20; // wpisanie w zarezerwowane miejsce wartości
  return *wsk;
}
```

Wycieki pamięci w Javie

- Ale przecież mechanizm Garbage Collectora powinien usuwać wszystkie niepotrzebne elementy z pamięci.
- Racja, ale działa to tylko dla elementów, **które zostały oznaczone jako niepotrzebne.**
- Czasami to „oznaczenie” jest automatyczne, ale źle napisany program może powodować rozrost wykorzystanej pamięci

Najczęstsze wycieki pamięci w Javie

Static field holding object reference [esp final field]

```
class MemorableClass {
    static final ArrayList list = new ArrayList(100);
}
```

Calling `String.intern()` on lengthy String

```
String str=readString(); // read lengthy string any source db,textbox/jsp etc..
// This will place the string in memory pool from which you cant remove
str.intern();
```

(Unclosed) open streams (file , network etc...)

```
try {
    BufferedReader br = new BufferedReader(new FileReader(inputFile));
    ...
    ...
} catch (Exception e) {
    e.printStacktrace();
}
```

Unclosed connections

```
try {
    Connection conn = ConnectionFactory.getConnection();
    ...
    ...
} catch (Exception e) {
    e.printStacktrace();
}
```

Identyfikacja wycieków

- Zidentyfikuj symptomy
- Włącz tryb gadatliwy dla garbage collectora (verbose mode „*-verbosegc*”)
- Użyj profilowania
- Dokonaj analizy śladu (trace)

Log GC

```
<AF[72]: Allocation Failure. need 1040 bytes, 3120898 ms since last AF>
<AF[72]: managing allocation failure, action=1 (792/18748408) (968224/976896)>
<GC: Tue Aug 27 11:41:31 2002
<GC(72): freed 5923688 bytes in 93 ms, 34% free (6892704/19725304)>
  <GC(72): mark: 79 ms, sweep: 14 ms, compact: 0 ms>
  <GC(72): refs: soft 0 (age >= 32), weak 0, final 60, phantom 0>
<AF[72]: completed in 100 ms>
```

Memory leak jeżeli:

1. GC zwalnia coraz mniej pamięci
2. Liczba wolnej pamięci spada

Jak działa GC

http://www.toptal.com/java/hunting-memory-leaks-in-java

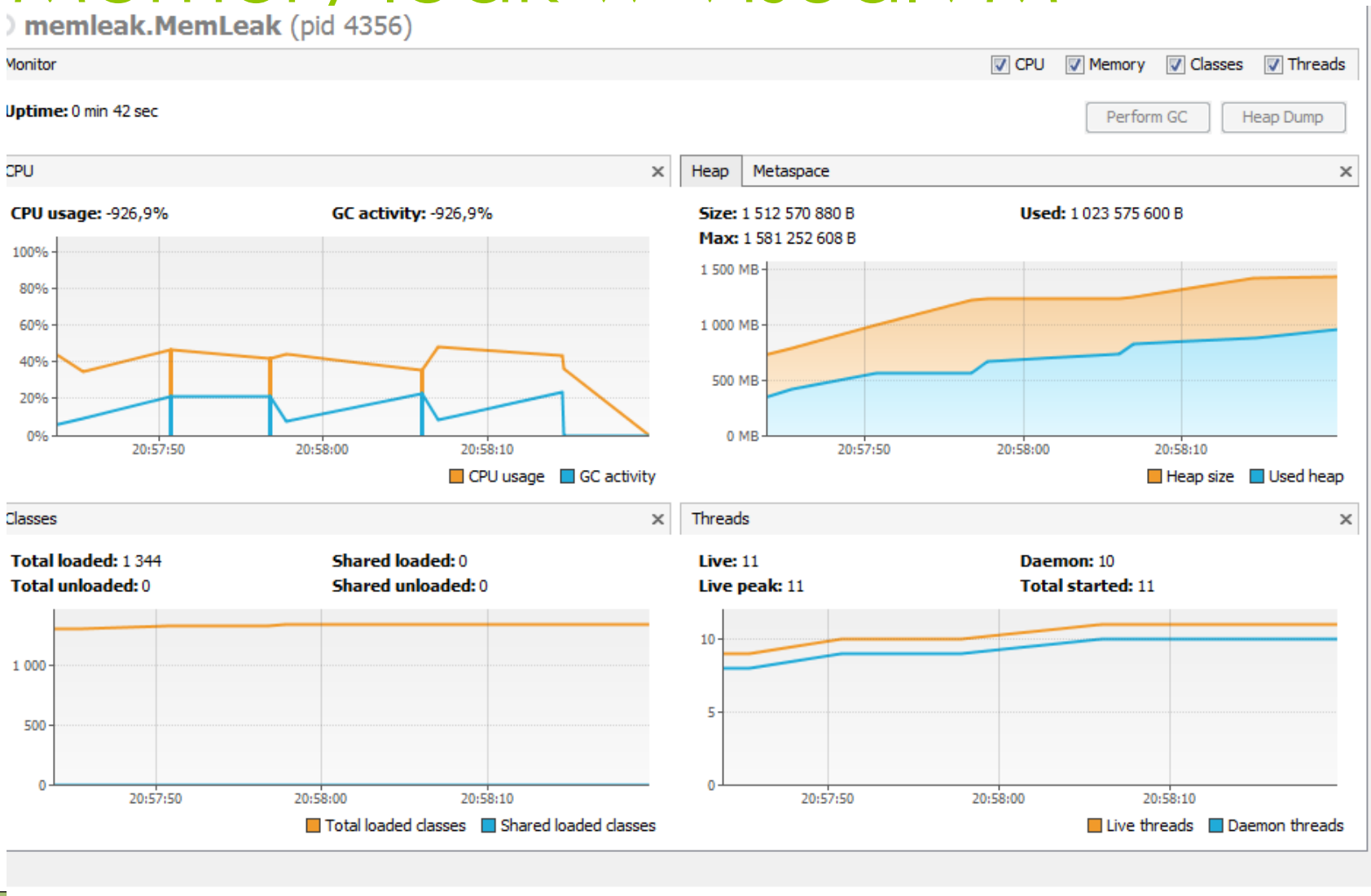
Przykładowy wyciek pamieci

```
package com.post.memory.leak;

import java.util.Map;

public class MemLeak {
    public final String key;

    public MemLeak(String key) {
        this.key =key;
    }

    public static void main(String args[]) {
        try {
            Map map = System.getProperties();

            for(;;) {
                map.put(new MemLeak("key"), "value");
            }
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

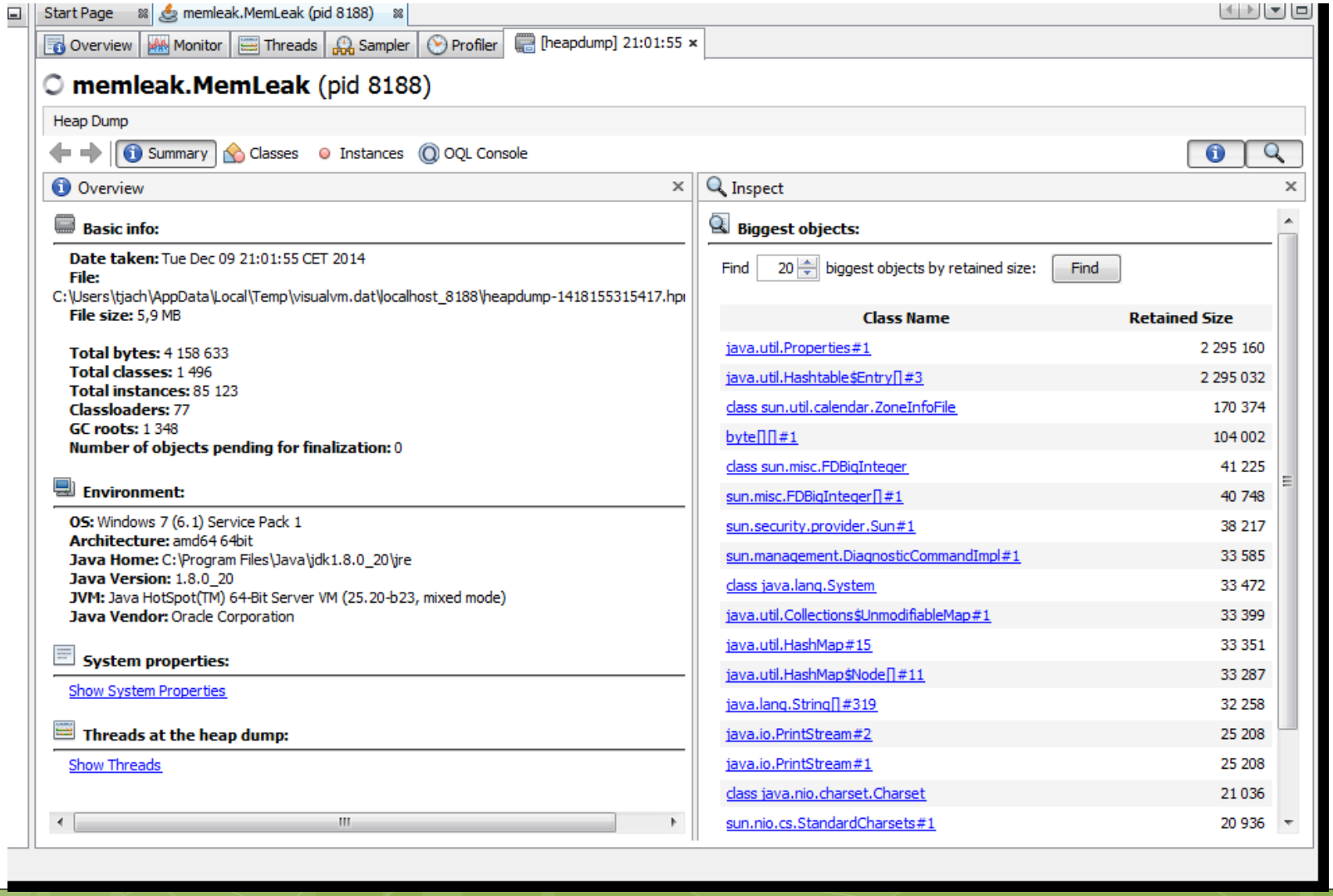
Visual VM

http://visualvm.java.net/

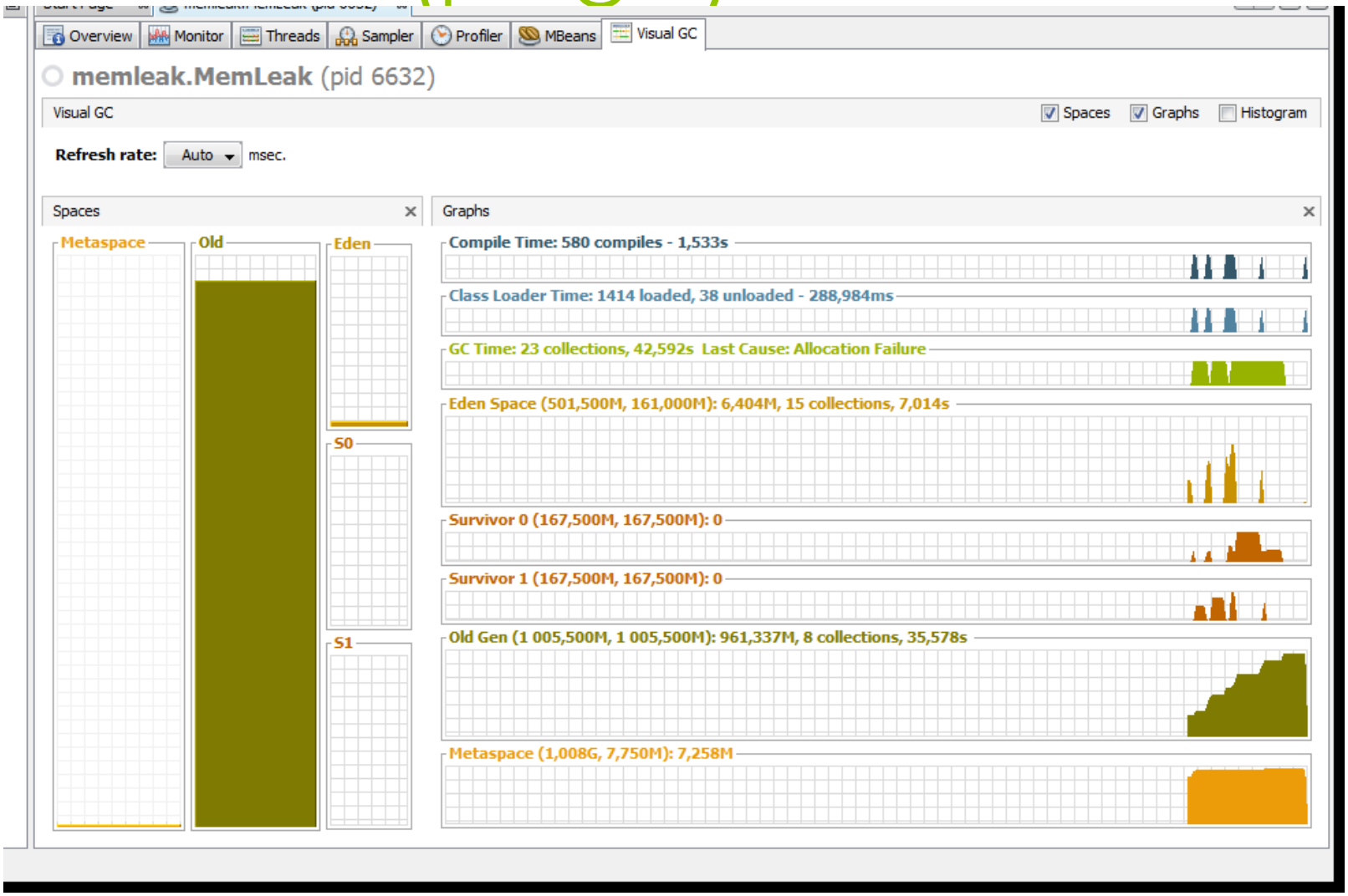
Memory leak w VisualVM



Heap Dump


Start Page
memleak.MemLeak (pid 8188)
Overview
Monitor
Threads
Sampler
Profiler
[heapdump] 21:01:55
memleak.MemLeak (pid 8188)
Heap Dump
Summary
Classes
Instances
OQL Console
Overview
Basic info:
Date taken: Tue Dec 09 21:01:55 CET 2014
File:
C:\Users\tjach\AppData\Local\Temp\visualvm.dat\localhost_8188\heapdump-1418155315417.hpr
File size: 5,9 MB
Total bytes: 4 158 633
Total classes: 1 496
Total instances: 85 123
Classloaders: 77
GC roots: 1 348
Number of objects pending for finalization: 0
Environment:
OS: Windows 7 (6.1) Service Pack 1
Architecture: amd64 64bit
Java Home: C:\Program Files\Java\jdk1.8.0_20\jre
Java Version: 1.8.0_20
JVM: Java HotSpot(TM) 64-Bit Server VM (25.20-b23, mixed mode)
Java Vendor: Oracle Corporation
System properties:
Show System Properties
Threads at the heap dump:
Show Threads
Inspect
Biggest objects:
Find 20 biggest objects by retained size:
Find
Class Name
Retained Size
java.util.Properties#1 2 295 160
java.util.Hashtable$Entry[]#3 2 295 032
class sun.util.calendar.ZoneInfoFile 170 374
byte[][]#1 104 002
class sun.misc.FDBigInteger 41 225
sun.misc.FDBigInteger[]#1 40 748
sun.security.provider.Sun#1 38 217
sun.management.DiagnosticCommandImpl#1 33 585
class java.lang.System 33 472
java.util.Collections$UnmodifiableMap#1 33 399
java.util.HashMap#15 33 351
java.util.HashMap$Node[]#11 33 287
java.lang.String[]#319 32 258
java.io.PrintStream#2 25 208
java.io.PrintStream#1 25 208
class java.nio.charset.Charset 21 036
sun.nio.cs.StandardCharsets#1 20 936

VisualGC (plugin)



Ćwiczenie 1

- Przeprowadź sam analizę przedstawionego kodu. Odkryj jakie elementy powodują wyciek pamięci
- Popraw kod, aby nie generował wycieków

Ćwiczenie 2

- Dokonaj profilowania aplikacji z poprzednich zajęć (bazodanowej)
- Dokonaj profilowania „zip-bomby” z poprzednich zajęć