

```
public String getDomainIp() {
    try {
        // Get IP address
        InetAddress address = InetAddress.getByName(domain);
        byte[] ipAddress = address.getAddress();

        // Convert to dot
        String ipAddressString = "";
        for (int i = 0; i < ipAddress.length; i++) {
            if (i > 0) {
                ipAddressString += ".";
            }
            ipAddressString += ipAddress[i] & 0xFF;
        }
        return ipAddressString;
    } catch (UnknownHostException e) {
        return "UnknownHostException";
    }
}
```

java.lang.OutOfMemoryError

Kamil Porembiński

<http://thecamels.org/>

Wstęp

- ▶ Budzi Cię alarm w środku nocy
- ▶ Dowiadujesz się, że aplikacja przestała działać
 - ▶ Przecierasz oczy, logujesz się do systemu by sprawdzić logi
 - Na konsoli widzisz `java.lang.OutOfMemoryError`
 - Uruchamiasz aplikację ponownie i kładziesz się spać



```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
  at java.util.Arrays.copyOf(Arrays.java:2760)
  at java.util.Arrays.copyOf(Arrays.java:2734)
  at java.util.ArrayList.ensureCapacity(ArrayList.java:167)
  at java.util.ArrayList.add(ArrayList.java:351)
  at MemoryLeakDemo.main(MemoryLeakDemo.java:12)
```

Wyciek pamięci

- ▶ Niezamierzone użycie pamięci przez program komputerowy
- ▶ Program nie zwalnia pamięci gdy nie jest już potrzebna, a rezerwuje nową
- ▶ Wycieki pamięci są efektem bardzo niepożądanym
- ▶ Program bowiem zajmuje coraz więcej pamięci, ale nie jest w stanie jej wykorzystać ani zwolnić
- ▶ Efekt wycieku pamięci stopniowo narasta w długo działających aplikacjach
- ▶ Prowadzi do spadku wydajności aplikacji
 - ▶ Zawieszenie się programu
 - ▶ Zablokowanie całego systemu

Automatyczne zarządzanie pamięcią

- ▶ Zbieranie nieużytków to architektura zarządzania pamięcią, w której proces zwalniania nieużywanych jej obszarów odbywa się automatycznie.
- ▶ Mechanizm taki stosuje się na przykład w wysokopoziomowych językach programowania, przy czym za jego obsługę nie odpowiada sam język, lecz wirtualna maszyna.
- ▶ Jest to pewna funkcja systemu, zadaniem której jest uzyskanie informacji o nieużytkach w pamięci operacyjnej lub na dysku (obszarach, które nie są wykorzystywane przez aktualnie przechowywane dane) a następnie, przesunięcie nieużytków do puli wolnej pamięci, która używana jest przy alokacji nowych danych.

Garbage Collector

- ▶ Jeden z mechanizmów zarządzania zasobami (m.in. pamięcią)
- ▶ Odpowiedzialny za zwalnianie niepotrzebnych zasobów
- ▶ Często zmniejsza fragmentację (zewnątrzną i wewnętrzną) pamięci
- ▶ Nie zwalnia z racjonalnego korzystania z pamięci
- ▶ Nie służy do wykrywania wycieków
- ▶ Nie dotyka zewnętrznych zasobów
- ▶ Nie czyni cudów ;-)

Kiedy uruchamia się Garbage Collector?

// Przykładowa klasa

```
public class JakasKlasa {
```

```
    // Przykładowa metoda
```

```
    public Boolean robCos() {
```

```
        return true;
```

```
    }
```

```
}
```

```
[...]
```

```
{
```

```
    JakasKlasa klasa = new JakasKlasa();
```

```
    klasa.robCos();
```

```
    // Jesli do obiektu przypiszemy wartosc NULL, zostanie on usuniety
```

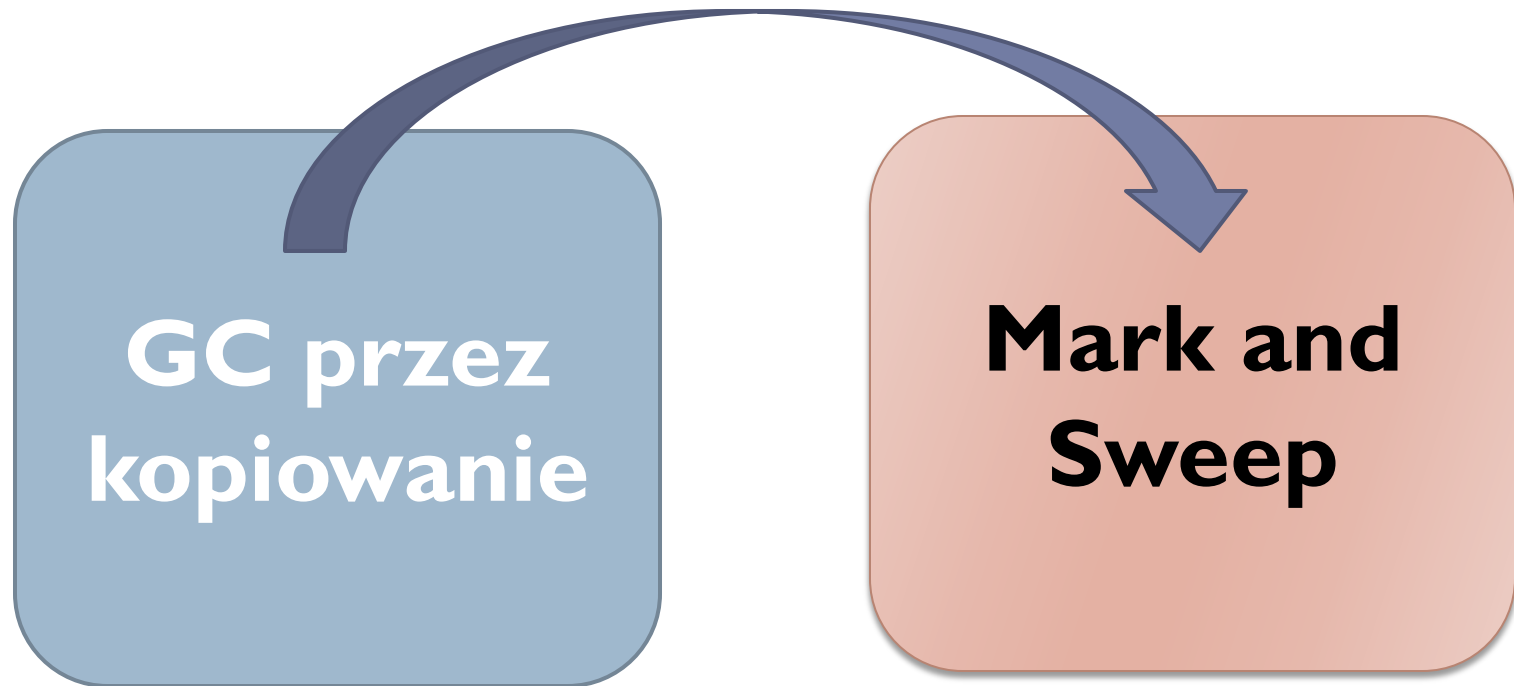
```
    // w cyku Garbage Collection
```

```
    // klasa = null;
```

```
} // W tym miejscu klasa bedzie nadawala sie do usuniecia przez GC
```



Algorytmy



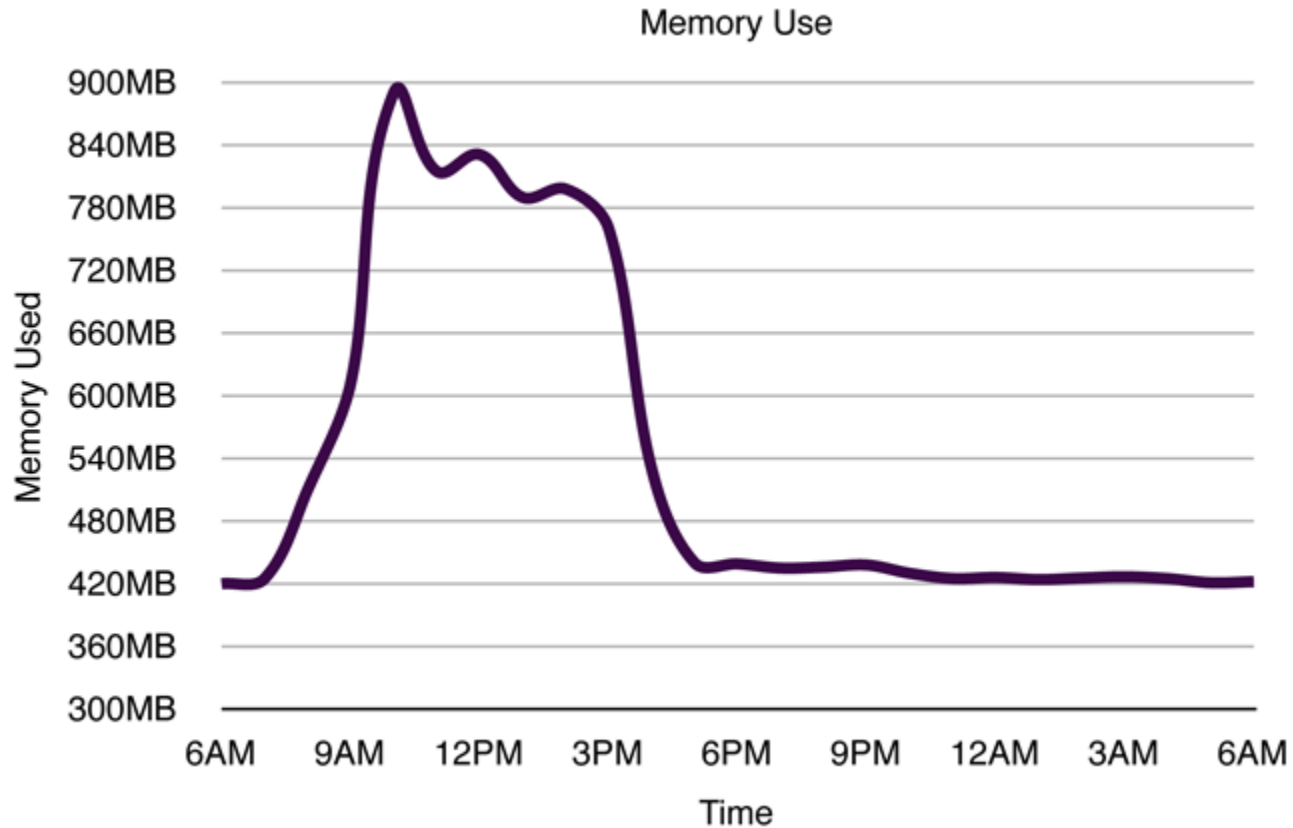
Wirtualna maszyna Javy stosuje rozwiązanie „**GC przez kopiowanie**”, lecz gdy uzna, że program generuje mało śmieci przechodzi do trybu **Mark and sweep**, przez co zyskuje na wydajności kosztem niewielkiej fragmentacji sterty pamięci.

Nie koniec problemu!

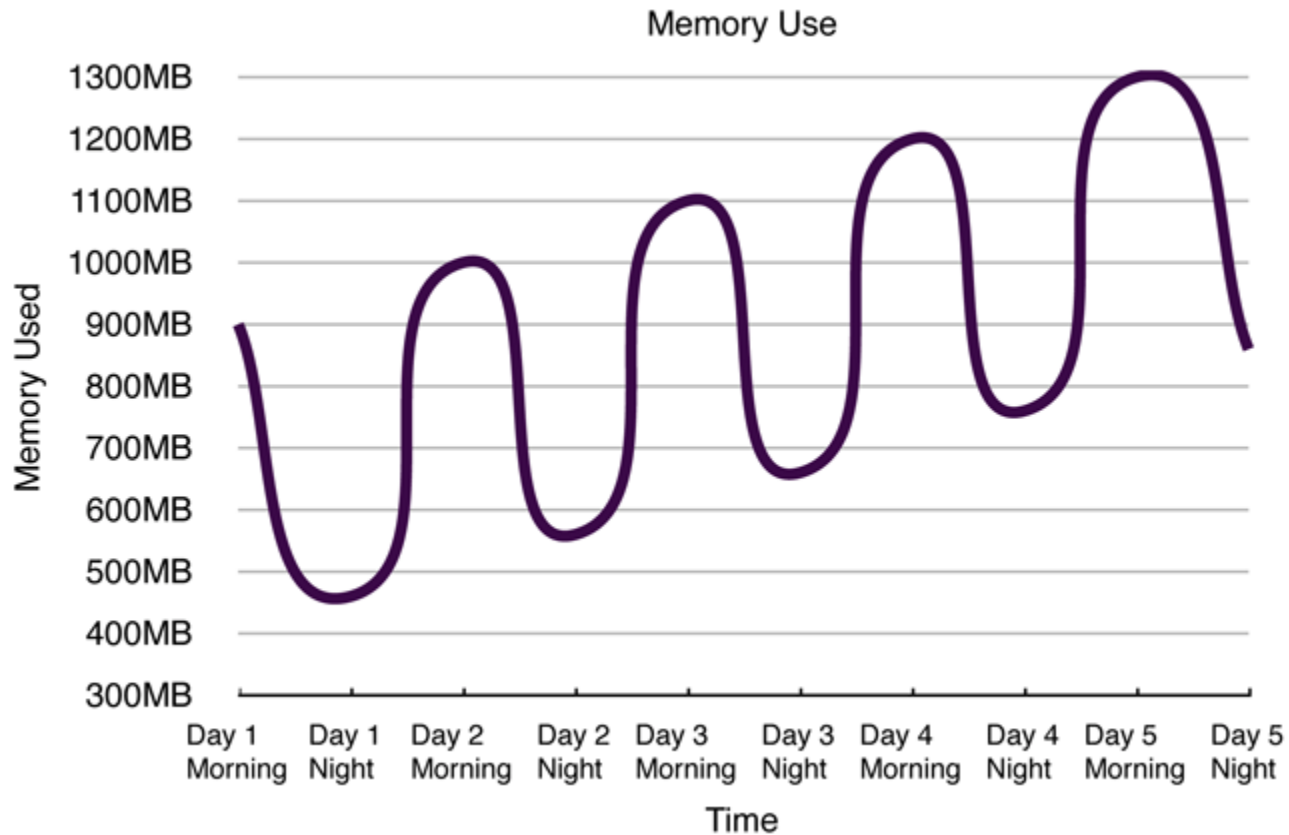
Garbage Collector nie rozwiązuje jednak całkowicie problemu, ponieważ zwalnia tylko pamięć, do której nie istnieją odwołania.

Niestarannie napisany kod powoduje, że śmieciarz nie zwolni pewnego obszaru pamięci, mimo że nie zamierzamy go używać.

Garbage Collection



Memory Leak



MemoryLeakDemo.java

```
import java.util.ArrayList;
import java.util.List;

public class MemoryLeakDemo {
    private static List<Integer> memoryLeakArea = new ArrayList<Integer>();

    public static void main(String[] args) {
        int i = 0;

        while (true) {
            Integer j = new Integer(i);
            memoryLeakArea.add(j);
            i++;
        }
    }
}
```

MemoryLeakDemo.sh

```
#!/bin/bash

# Enable the jconsole agent remotely on port 6485
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.port=6485"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.jmxremote.ssl=false"

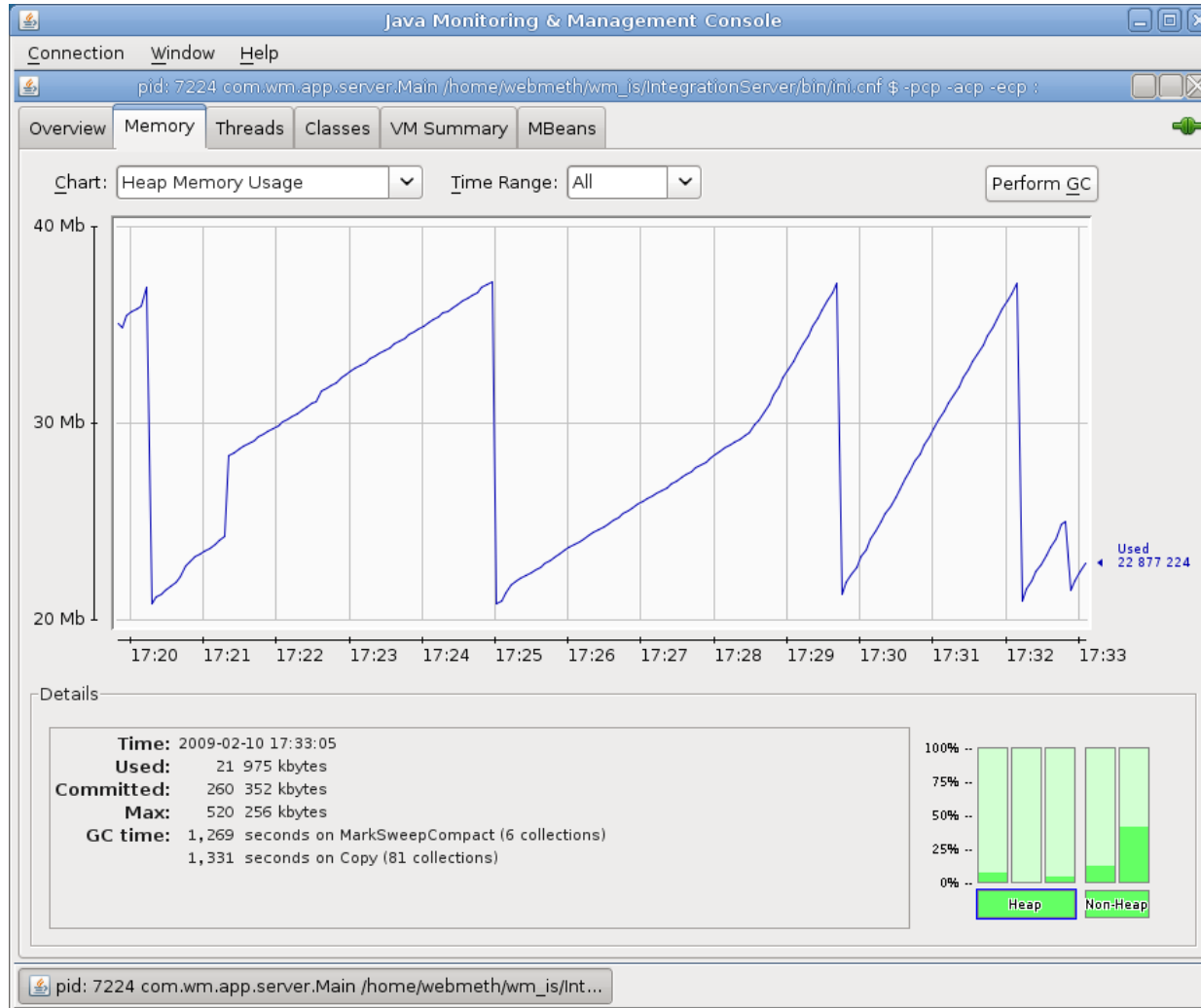
# Memory settings
JAVA_MIN_MEM=1024M
JAVA_MAX_MEM=1024M
JAVA_MEMSET="-Dsun.lang.ClassLoader.allowArraySyntax=true -Xms${JAVA_MIN_MEM} -Xmx${JAVA_MAX_MEM}"

JAVA_DIR="/home/poremkam/java/64/jdk1.6.0_16/jre"

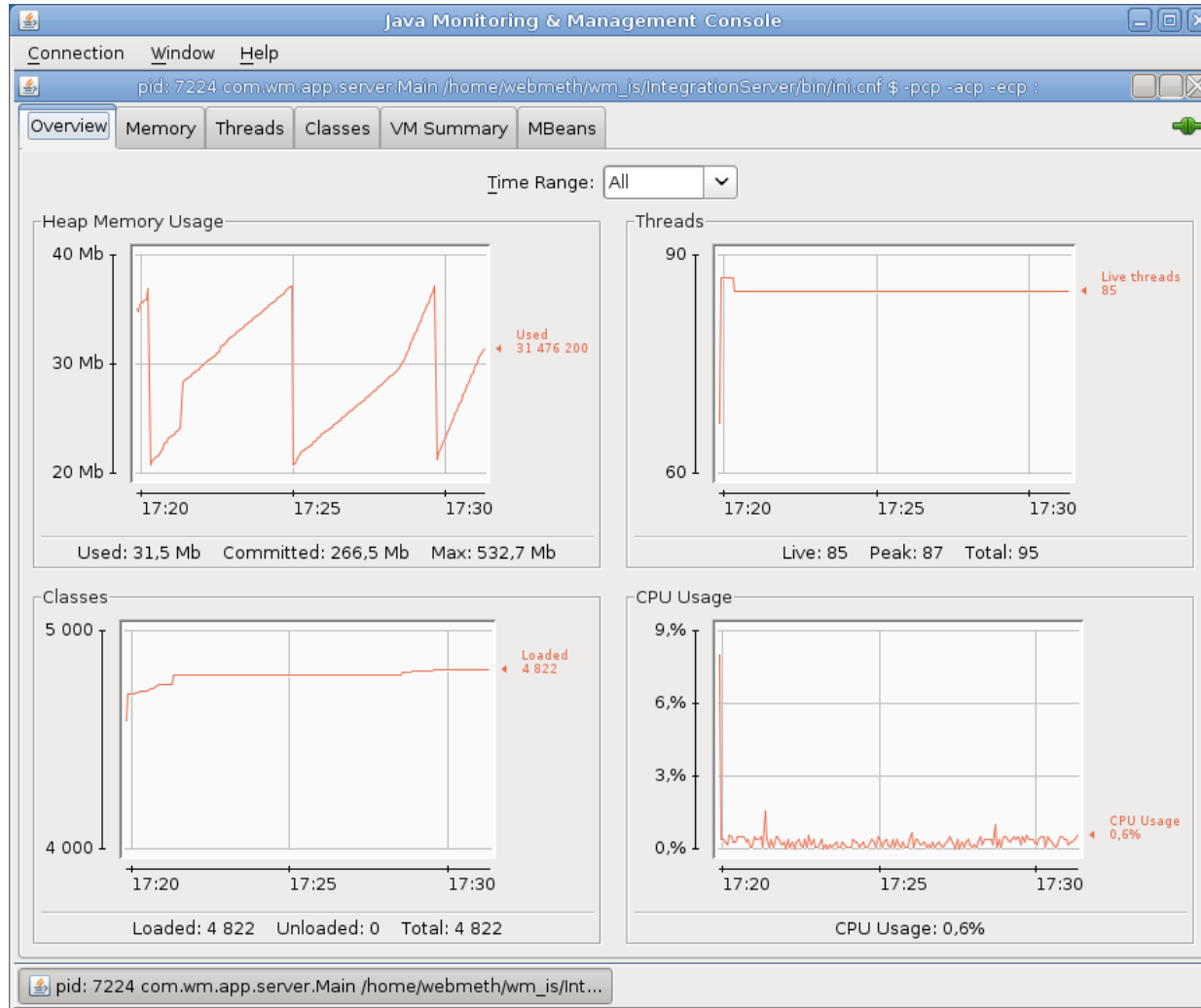
JAVA_EXE="bin/java"
JAVA_RUN="${JAVA_DIR}/${JAVA_EXE} ${JAVA_MEMSET}"

# Run MemoryLeakDemo
${JAVA_RUN} ${JAVA_OPTS} MemoryLeakDemo
```

jConsole



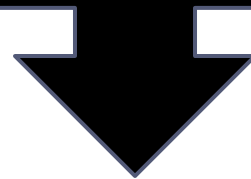
jConsole



Verbose GC Log

- ▶ `-verbose:gc` — drukuje podstawowe informacje na temat GC na STDOUT
- ▶ `-XX:+PrintGCTimeStamps` — drukuje czasy uruchomienia GC
- ▶ `-XX:+PrintGCDetails` — pokazuje szczegółowe statystyki na temat GC
- ▶ `-Xloggc:<plik>` — wynik logowany jest do pliku

```
1.854: [GC 1.854: [DefNew: 570K->62K(576K), 0.0012355 secs] 2623K->2175K(3980K), 0.0012922 secs]
1.871: [GC 1.871: [DefNew: 574K->55K(576K), 0.0009810 secs] 2687K->2229K(3980K), 0.0010752 secs]
1.881: [GC 1.881: [DefNew: 567K->30K(576K), 0.0007417 secs] 2741K->2257K(3980K), 0.0007947 secs]
1.890: [GC 1.890: [DefNew: 542K->64K(576K), 0.0012155 secs] 2769K->2295K(3980K), 0.0012808 secs]
```



Zebrane dane można potem analizować
w *trybie offline* w innych programach

Eclipse Memory Analyzer (MAT)

The screenshot displays the Eclipse Memory Analyzer (MAT) interface. The main window shows a report for a memory leak analysis. The report includes the following details:

- Size: 96,7 MB
- Classes: 334
- Objects: 5,1m
- Class Loader: 3
- Unreachable Objects Histogram

The 'Biggest Objects by Retained Size' section shows a pie chart with three segments:

- 96,6 MB (the largest segment)
- 45,5 KB
- 65,5 KB

The console window at the bottom shows the following output:

```
<terminated> MemoryLeakDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2010-05-25 11:33:50)
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid5428.hprof ...
Heap dump file created [129808903 bytes in 7.624 secs]
```

Eclipse Memory Analyzer (MAT)

Memory Analysis - E:\subversion\MemoryLeakDemo\java_pid5428.hprof - Eclipse

File Edit Refactor Navigate Search Project Scripts Run Window Help

Inspector

MemoryLeakDemo.java My Studio java_pid5428.hprof

Overview default_report org.eclipse.mat.api:suspects Histogram

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.lang.Integer	5 057 404	80 918 464	>= 80 919 088
java.lang.Object[]	304	20 369 576	>= 101 290 632
char[]	183	49 128	>= 49 128
byte[]	7	24 944	>= 24 944
java.lang.String	542	13 008	>= 26 744
java.lang.Class	335	3 448	>= 101 380 800
java.lang.String[]	49	1 824	>= 4 928
java.util.Hashtable\$Entry	58	1 392	>= 8 440
java.util.HashMap\$Entry[]	17	1 360	>= 3 752
int[]	3	1 152	>= 1 152
java.util.Hashtable\$Entry[]	7	680	>= 9 120
java.util.Locale	20	640	>= 2 768
java.net.URL	11	616	>= 2 368
java.util.HashMap	15	600	>= 4 176
short[]	1	528	>= 528
java.util.concurrent.ConcurrentHashMap\$Segment	16	512	>= 1 848
java.util.concurrent.ConcurrentHashMap\$HashEntry	20	480	>= 528
java.util.concurrent.ConcurrentHashMap\$HashEntry[]	16	416	>= 944
java.util.LinkedHashMap\$Entry	13	416	>= 2 184
Total 19 of 334 entries	5 059 286	101 396 320	

Notes Navigation History Console

```
<terminated> MemoryLeakDemo [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (2010-05-25 11:33:50)
java.lang.OutOfMemoryError: Java heap space
Dumping heap to java_pid5428.hprof ...
Heap dump file created [129808903 bytes in 7.624 secs]
```

Inne narzędzia

- ▶ **jstat - Java Virtual Machine Statistics Monitoring Tool**
- ▶ **jvmstat**
- ▶ **IBM Monitoring and Diagnostic Tools for Java jako plugin dla IBM Support Assistant**
- ▶ **SUN GC Portal**
- ▶ **VisualVM**

Bibliografia

- ▶ <http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html>
- ▶ <http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html>
- ▶ http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf
- ▶ <http://java.sun.com/developer/technicalArticles/Programming/GCPortal/>
- ▶ http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html
- ▶ <http://www.eclipse.org/mat/>
- ▶ <https://visualvm.dev.java.net/>
- ▶ <http://www.ej-technologies.com/products/jprofiler/overview.html>
- ▶ <http://www.yourkit.com/>
- ▶ <http://www.quest.com/jprobe/>
- ▶ http://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf
- ▶ <http://java.sun.com/javase/6/docs/technotes/tools/share/jhat.html>
- ▶ <http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jmap.html>
- ▶ <http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html>

Koniec

