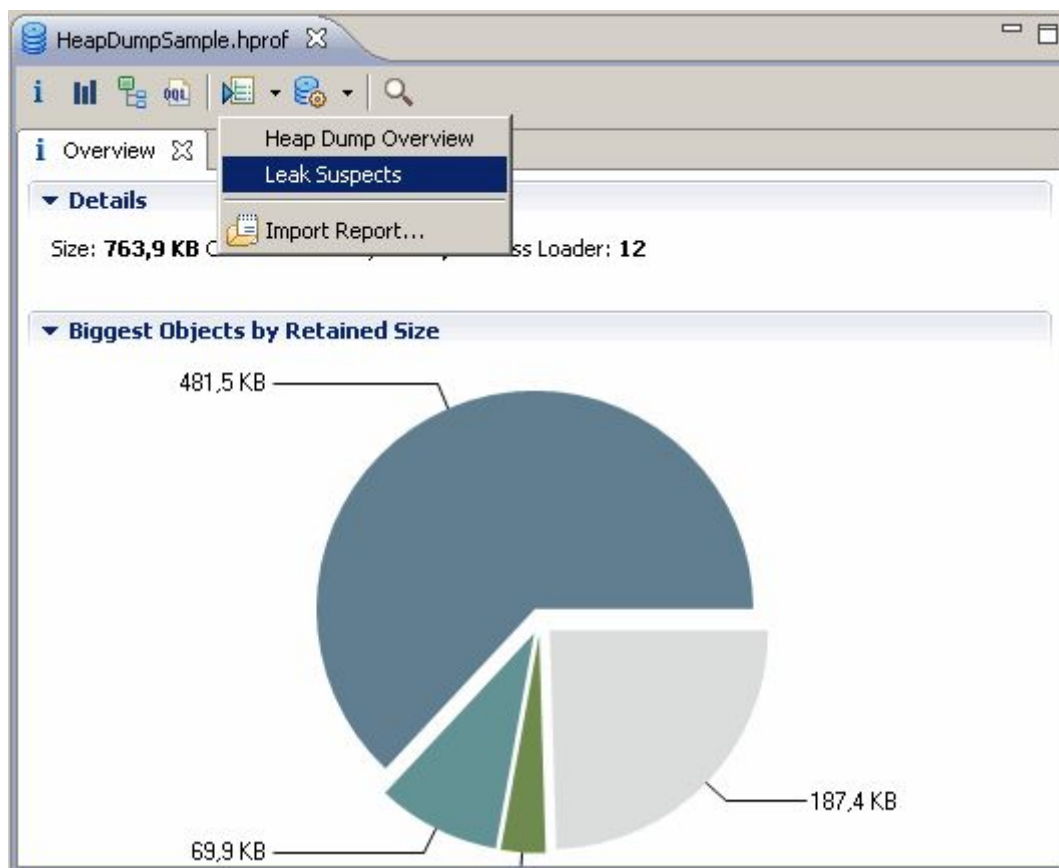


# Encontrar Memory Leaks con Eclipse Memory Analyzer

## Ejecución Leak Suspect Report

de la barra de herramientas seleccione el menú desplegable Run Expert System Test > Leak Suspects. Como resultado se abrirá un informe HTML. Contiene una visión general de la pila y fugas información sospechosa.



Este informe se almacena junto con el volcado del heap y se muestra cuando se abre el volcado de pila de nuevo.

Algunas de las secciones de los sospechosos de fugas reportar tener enlaces para volver a ejecutar las consultas individuales que componen el informe. Esto puede ser útil para el análisis adicional.

▼ Accumulated Objects by Class in Dominator Tree

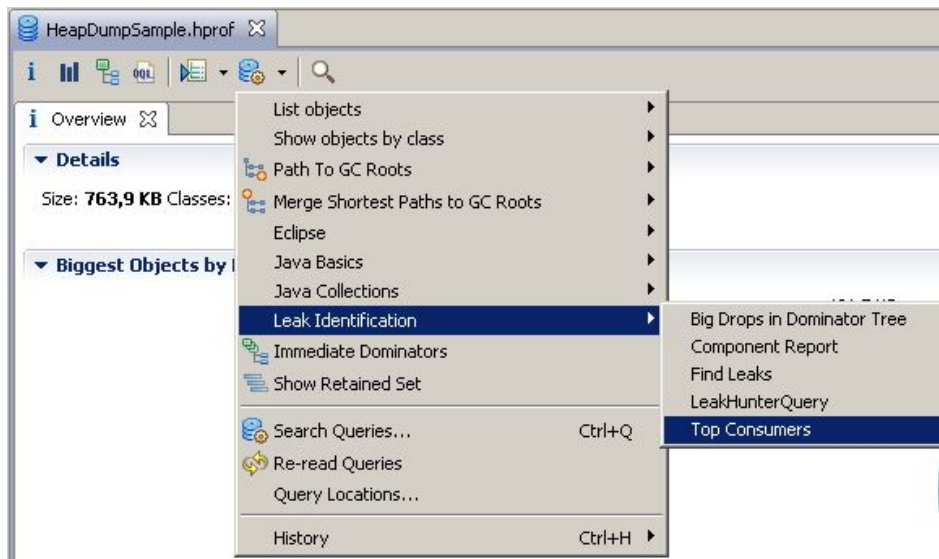
Label	Number of Objects	Used Heap Size	Retained Heap Size
<a href="#">java.util.LinkedHashMap\$Entry</a> First 10 of 564 objects	564	36,096	123,984
<a href="#">java.lang.String[]</a> First 10 of 67 objects	67	3,752	22,776
<a href="#">java.util.HashMap\$Entry[]</a> All 1 objects	1	8,216	8,216
<a href="#">java.lang.String</a> First 10 of 33 objects	33	1,320	2,496
Σ Total: 4 entries	665	49,384	157,472

▼ All Accumulated Objects by Class

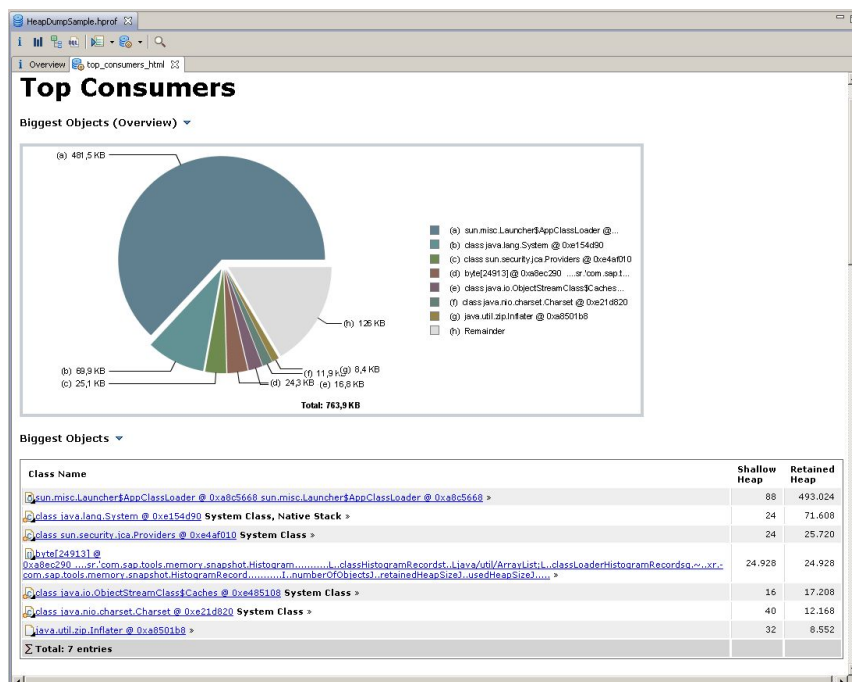
Class Name	Objects	Shallow Heap
<a href="#">char[]</a> First 10 of 1,116 objects	1,116	59,672
<a href="#">java.lang.String</a> First 10 of 1,116 objects	1,116	44,640
<a href="#">java.util.LinkedHashMap\$Entry</a> First 10 of 564 objects	564	36,096
<a href="#">java.lang.String[]</a> First 10 of 158 objects	158	8,848
<a href="#">java.util.HashMap\$Entry[]</a> All 1 objects	1	8,216
<a href="#">java.util.LinkedHashMap</a> All 1 objects	1	80
Σ Total: 6 entries	2,956	157,552

# Buscar los objetos más grandes

La consulta de *Top Consumers* muestra los objetos más grandes agrupados por clase, cargador de clases, y el paquete. Para ejecutar la consulta de selección principales consumidores en la categoría de identificación de fugas:



El resultado de la consulta se muestra como página HTML. Los objetos son representados por los hipervínculos, la activación de los hipervínculos se puede abrir un menú contextual y continuar con el análisis del objeto seleccionado.



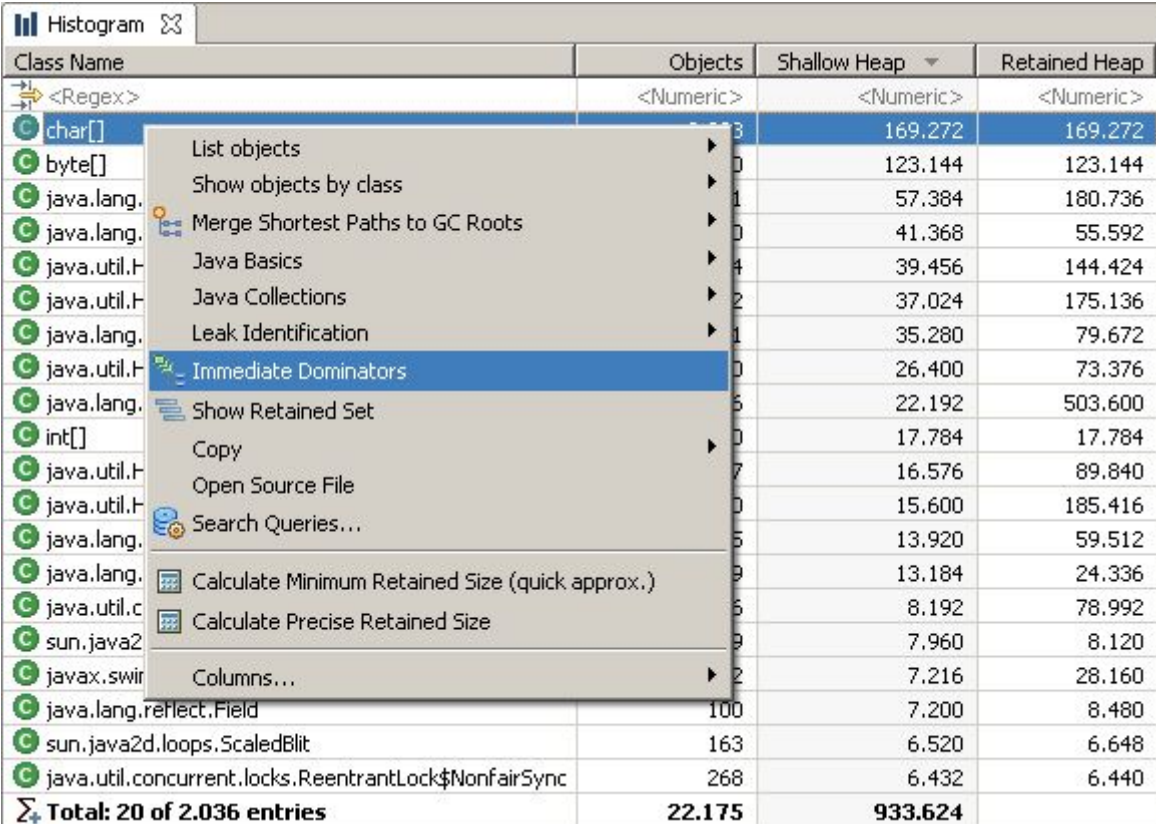
# Encontrar objetos responsables

## Dominators inmediatos

Esta consulta busca y agrega todos los objetos que dominan un conjunto dado de objetos en el nivel de clase. Es muy útil para encontrar rápidamente quién es el responsable de un conjunto de objetos, ya que responde directamente a la pregunta **"que mantiene estos objetos vivos"** en lugar de responder **"que tiene una referencia a estos objetos"**. Usando el hecho de que cada objeto tiene un solo dominador inmediato (a diferencia de múltiples referencias entrantes) de la herramienta ofrece la posibilidad de filtrar dominadores "sin interés" (por ejemplo java.\*) y ver directamente las clases de la aplicación responsables.

Elección de opciones:

- Selección de entradas y usando el menú contextual



Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
char[]	3	169,272	169,272
byte[]	0	123,144	123,144
java.lang.	1	57,384	180,736
java.lang.	0	41,368	55,592
java.util.H	4	39,456	144,424
java.util.H	2	37,024	175,136
java.lang.	1	35,280	79,672
java.util.H	0	26,400	73,376
java.lang.	6	22,192	503,600
int[]	0	17,784	17,784
java.util.H	7	16,576	89,840
java.util.H	0	15,600	185,416
java.lang.	5	13,920	59,512
java.lang.	9	13,184	24,336
java.util.c	5	8,192	78,992
sun.java2	9	7,960	8,120
javax.swir	2	7,216	28,160
java.lang.reflect.Field	100	7,200	8,480
sun.java2d.loops.ScaledBlit	163	6,520	6,648
java.util.concurrent.locks.ReentrantLock\$NonfairSync	268	6,432	6,440
<b>Total: 20 of 2,036 entries</b>	<b>22,175</b>	<b>933,624</b>	

# Consultando Heap Objets (OQL)

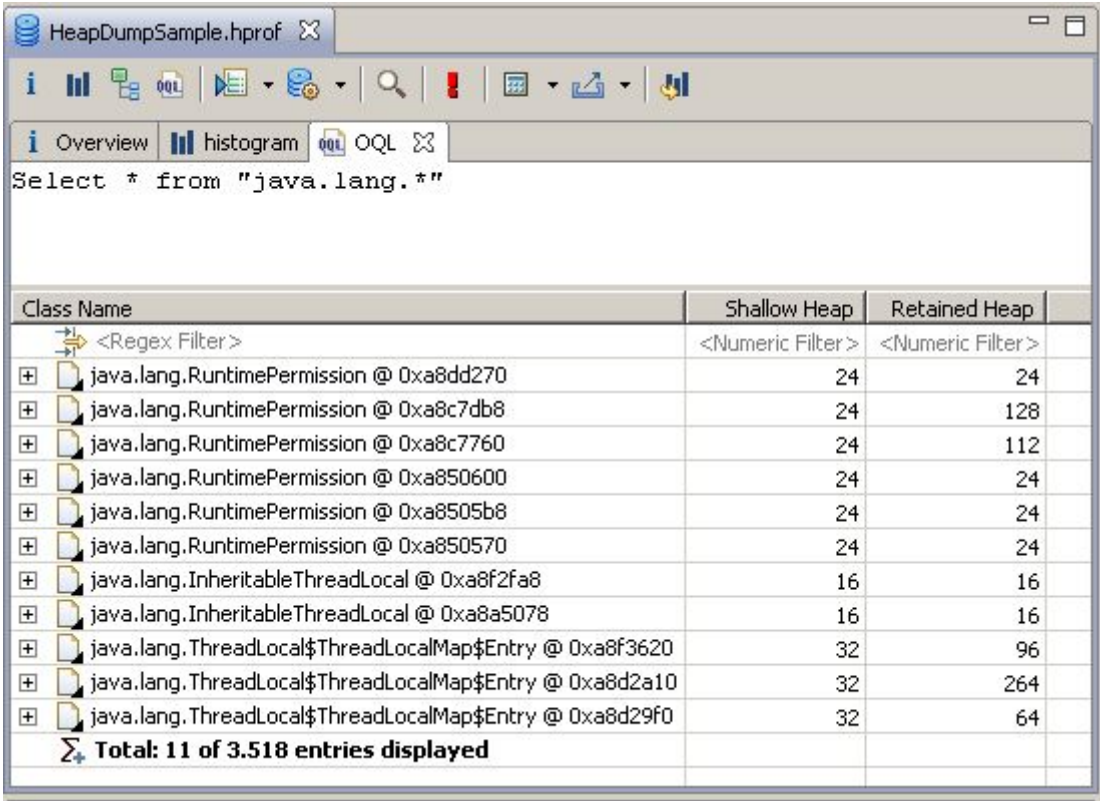
Memory Analyzer permite consultar el volcado del heap con las consultas SQL-como personalizados. OQL representa clases como tablas, objetos como filas, y los campos como columnas.

```
SELECT *
FROM [INSTANCIAD] <class name = "nombre">
[WHERE <filtro de expresión>]
</ filter-expresión> </ class>
```

Para abrir un editor OQL utilizar el botón de la barra de  herramientas:

El editor de OQL se divide en dos áreas:

- área de texto para escribir en la consulta (zona alta)
- esfera de resultados para mostrar el resultado de la ejecución de la consulta



The screenshot shows the Memory Analyzer interface with the OQL editor open. The query entered is "Select \* from 'java.lang.\*'". The results are displayed in a table with columns for Class Name, Shallow Heap, and Retained Heap. The table lists various Java classes and their instances, along with their memory addresses, shallow heap sizes, and retained heap sizes. A summary row at the bottom indicates that 11 of 3,518 entries are displayed.

Class Name	Shallow Heap	Retained Heap
<Regex Filter>	<Numeric Filter>	<Numeric Filter>
java.lang.RuntimePermission @ 0xa8dd270	24	24
java.lang.RuntimePermission @ 0xa8c7db8	24	128
java.lang.RuntimePermission @ 0xa8c7760	24	112
java.lang.RuntimePermission @ 0xa850600	24	24
java.lang.RuntimePermission @ 0xa8505b8	24	24
java.lang.RuntimePermission @ 0xa850570	24	24
java.lang.InheritableThreadLocal @ 0xa8f2fa8	16	16
java.lang.InheritableThreadLocal @ 0xa8a5078	16	16
java.lang.ThreadLocal\$ThreadLocalMap\$Entry @ 0xa8f3620	32	96
java.lang.ThreadLocal\$ThreadLocalMap\$Entry @ 0xa8d2a10	32	264
java.lang.ThreadLocal\$ThreadLocalMap\$Entry @ 0xa8d29f0	32	64
Total: 11 of 3.518 entries displayed		

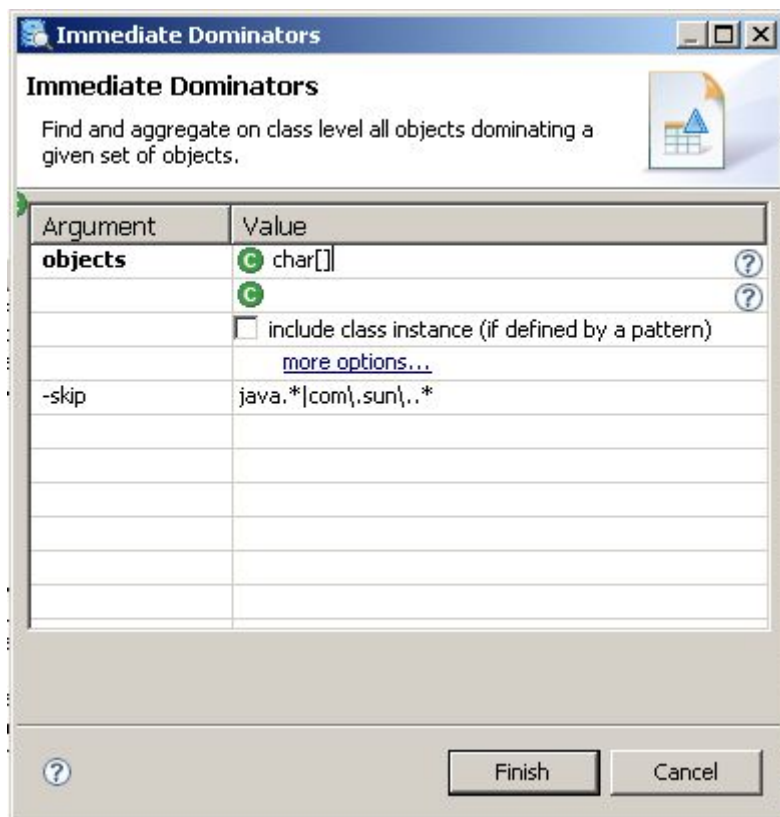
uso F5 o Ctrl-Enter o el botón de la barra de herramientas  para ejecutar la consulta.

OQL sintaxis básica es la siguiente:

```
SELECT *
FROM [INSTANCIAD] <nombre de clase>
```



- La barra de herramientas de comando *Query Browser > Immediate Dominators* inicia un asistente para seleccionar un conjunto de objetos.



Los dominadores inmediatos de todos los arrays de char son todos los objetos responsables de mantener vivo el char []. El resultado contendrá objetos java.lang.String más probables. Si se agrega el patrón de *skip* de java.\*, Y verá las clases no JDK responsables de los arrays de char.

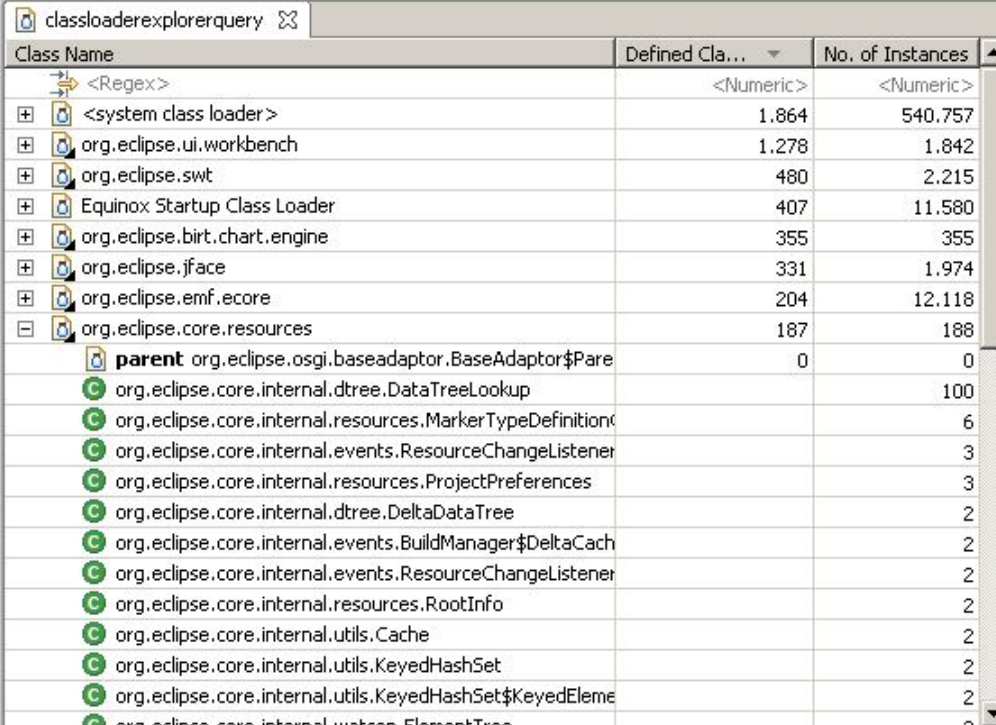
# Analizar el Class Loader

Cargadores de clases cargar las clases en la memoria de la JVM. Al analizar el heap, cargadores de clases son muy importante por dos razones: En primer lugar, las aplicaciones normalmente se cargan los componentes que utilizan cargadores de clases separadas. En segundo lugar, las clases cargadas se almacenan generalmente en un espacio separado (por ejemplo, el espacio de *perm-gen*), que también se pueden agotar.

## Explorador de classloaders

Para obtener una visión general, ejecutar el *Query Browser > Java Basics > Class Loader Explorer inspection* en el volcado del heap.

- El analizador de la memoria adjunta una etiqueta significativo para el cargador de clase - en el caso de OSGi paquetes es el ID de paquete. **Revisar las entradas duplicadas.**
- Junto al nombre del cargador de clases, la tabla contiene las clases definidas y el número de casos en vivo. Si uno y el mismo componente se carga varias veces, el número de casos en vivo puede indicar qué cargadores de clases está más vivo y cuál debe ser candidato del GC.



Class Name	Defined Cla...	No. of Instances
<Regex>	<Numeric>	<Numeric>
<system class loader>	1.864	540.757
org.eclipse.ui.workbench	1.278	1.842
org.eclipse.swt	480	2.215
Equinox Startup Class Loader	407	11.580
org.eclipse.birt.chart.engine	355	355
org.eclipse.jface	331	1.974
org.eclipse.emf.ecore	204	12.118
org.eclipse.core.resources	187	188
parent org.eclipse.osgi.baseadaptor.BaseAdaptor\$Pare	0	0
org.eclipse.core.internal.dtree.DataTreeLookup		100
org.eclipse.core.internal.resources.MarkerTypeDefinition		6
org.eclipse.core.internal.events.ResourceChangeListener		3
org.eclipse.core.internal.resources.ProjectPreferences		3
org.eclipse.core.internal.dtree.DeltaDataTree		2
org.eclipse.core.internal.events.BuildManager\$DeltaCach		2
org.eclipse.core.internal.events.ResourceChangeListener		2
org.eclipse.core.internal.resources.RootInfo		2
org.eclipse.core.internal.utils.Cache		2
org.eclipse.core.internal.utils.KeyedHashSet		2
org.eclipse.core.internal.utils.KeyedHashSet\$KeyedEleme		2
org.eclipse.core.internal.watson.ElementTree		2



# Análisis de Threads

Memory Analyzer proporciona varias consultas para inspeccionar los threads en el momento en que se tomó la instantánea.

## Threads Descripción general

Para obtener una visión general de todos los hilos en el volcado del heap debemos usar la "Thread Overview" botón en la barra, como se muestra en la siguiente imagen. Alternativamente se podría utilizar la consulta *Query Browser > Thread Overview and Stacks*:

Object / Stack Frame	Name	Shallow Heap	Retained Heap	Context Class Loader
<Regex>	<Regex>	<Numeric>	<Numeric>	<Regex>
java.lang.Thread @ 0xe02970b8	main	112	28.480	org.eclipse.core.runtime.inter
org.eclipse.jface.text.reconciler.AbstractReconcile	org.eclipse.jdt.internal.ui.text.J...	120	3.688	org.eclipse.core.runtime.inter
org.eclipse.osgi.framework.eventmgr.EventManag	Start Level Event Dispatcher	128	2.984	org.eclipse.core.runtime.inter
org.eclipse.osgi.framework.eventmgr.EventManag	Framework Event Dispatcher	128	2.192	org.eclipse.core.runtime.inter
org.eclipse.jface.text.reconciler.AbstractReconcile	org.eclipse.jdt.internal.ui.text.J...	120	1.856	org.eclipse.core.runtime.inter
org.eclipse.jface.text.reconciler.AbstractReconcile	org.eclipse.jdt.internal.ui.text.J...	120	1.848	org.eclipse.core.runtime.inter
org.eclipse.jface.text.reconciler.AbstractReconcile	org.eclipse.jdt.internal.ui.text.J...	120	1.456	org.eclipse.core.runtime.inter
java.lang.Thread @ 0xe1378d18	Support Information Request ...	112	1.128	org.eclipse.core.runtime.inter
java.lang.Thread @ 0xe13d6280	Java indexing	112	664	org.eclipse.jdt.core
at java.lang.Object.wait(J)V (Native Method)				
at java.lang.Object.wait(J)V (Object.java:485)				
<local> org.eclipse.jdt.internal.core.search		80	2.992.568	
at org.eclipse.jdt.internal.core.search.processing.J				
at java.lang.Thread.run(J)V (Unknown Source)				
Σ Total: 4 entries				
java.lang.Thread @ 0xe026b1c0	[Timer] - Main Queue Handler	112	632	org.eclipse.core.runtime.inter

La consulta proporciona algunas propiedades como nombre, objeto, classloader de contexto, etc.. por cada uno de los hilos.

Algunos formatos de volcado de pila (por ejemplo HPROF) contiene información sobre las pilas de llamadas de hilos, y los objetos Java locales por marco de pila.

La exploración de los call-stacks y los objetos locales de Java es una característica de gran alcance, dando un depurador como capacidades de más de un snapshot. Permite analizar en detalle el motivo de un uso intensivo de memoria operaciones. *También permite que los heap dumps y Memory Analyzer se utilizan no sólo para los problemas relacionados con la memoria, sino también para una amplia gama de otros problemas, por ejemplo, las aplicaciones que no responden.*

## Detalles de Threads

Puede proceder con el análisis de un solo hilo mediante el uso de *Java Basics > Thread Details* del menú contextual. Memory Analyzer proporciona un punto de extensión, de manera que las extensiones pueden proporcionar información semántica sobre la actividad hilos. El resultado de el *Thread Details* consulta dicha información (si está disponible), alguna información general, y posiblemente el StackTrace de el thread.



The screenshot shows the 'Thread Details' window in the Eclipse Memory Analyzer. The window has a title bar with tabs for 'Overview', 'thread\_overview', and 'Details: Thread Details'. The main content area is titled 'Thread Details' and contains a section for 'Thread main'. Under 'Thread Properties', there is a table with the following data:

Name	main
Instance	java.lang.Thread @ 0xcb16b8
Shallow Heap	96
Retained Heap	1,816
Context Class Loader	sun.misc.Launcher\$AppClassLoader @ 0xcfd6d0
DTFJ Name	main
JNIEnv	0xcbd00
Priority	5
State	[alive, runnable]
State value	0x5
Native id	8920
Σ Total: 11 entries	

Below the table is the 'Thread Stack' section, which shows the following stack trace:

```
main
  at java.io.FileInputStream.readBytes([BII)I (Native Method)
  at java.io.FileInputStream.read([BII)I (FileInputStream.java:213)
  at java.io.BufferedInputStream.fill()V (BufferedInputStream.java:229)
  at java.io.BufferedInputStream.read1([BII)I (BufferedInputStream.java:269)
  at java.io.BufferedInputStream.read([BII)I (BufferedInputStream.java:328)
  at com.ibm.jvm.io.ConsoleInputStream.read([BII)I (ConsoleInputStream.java:177)
  at com.ibm.jvm.io.ConsoleInputStream.read([B)I (ConsoleInputStream.java:167)
  at com.ibm.jvm.io.ConsoleInputStream.read()I (ConsoleInputStream.java:160)
  at org.eclipse.mat.tests.CreateSampleDump.main([Ljava/lang/String;)V (CreateSampleDump.java:31)
```

Below the stack trace is the 'Native stack' section, which shows the following native stack trace:

```
C:\WINDOWS\system32\ntdll.dll::KiFastSystemCallRet+0x0
C:\WINDOWS\system32\kernel32.dll::GetConsoleInputWaitHandle+0x318
C:\WINDOWS\system32\kernel32.dll::ReadConsoleA+0x3b
C:\WINDOWS\system32\kernel32.dll::ReadFile+0xa5
C:\program files\IBM\Java60\jre\bin\java.dll::JCL_ReadFile+0xe4
C:\program files\IBM\Java60\jre\bin\java.dll::handleRead+0x24
C:\program files\IBM\Java60\jre\bin\java.dll::_Java_sun_misc_NativeSignalHandler_handle0@20+0x13c
C:\program files\IBM\Java60\jre\bin\java.dll::_Java_java_io_FileInputStream_readBytes@20+0x1d
C:\program files\IBM\Java60\jre\bin\j9jit24.dll::_J9VMD11Main+0x137f15c
C:\program files\IBM\Java60\jre\bin\j9vm24.dll+0x171cc
C:\program files\IBM\Java60\jre\bin\j9vm24.dll+0x18498
C:\program files\IBM\Java60\jre\bin\java.exe+0x95c5
C:\WINDOWS\system32\kernel32.dll::GetModuleFileNameA+0x1b4
<unknown location>
```

## **Normales**

Stack de pila sólo se muestran en el thread pilas ver.

## **Sólo Stack-Frame de pila como pseudo-objects**

Stack de pila se muestran en todos los puntos de vista tales como caminos para GC raíces, referencias salientes del thread, como pseudo-objetos. Variables referencias locales en los marcos de pila se muestran como referencias salientes desde el stack. Esto hace que sea más fácil encontrar el que stack de marcos de mantener los objetos vivos. El tamaño del marco de pila es el tamaño en la pila Java, no el heap.

## **Stack de pila como pseudo-objects y métodos que se ejecutan como pseudo-classes**

Stack de pila se muestran en todas las vistas, tales como caminos para *GC Roots*, referencias salientes de hilos, como pseudo-objetos. Variables referencias locales en los marcos de pila se muestran como referencias salientes desde el marco. Esto hace que sea más fácil encontrar stack frame que mantener los objetos vivos. Los stack frames se les da un pseudo-type dependiendo el método que se está ejecutando en el frame. Al ver el número de instancias de ese tipo seudo es fácil ver que los métodos se están ejecutando a través de todas los threads y qué métodos utilizan una gran cantidad de la pila. Esto puede ayudar a resolver *StackOverflowErrors*.

## **Stack frames como pseudo-objects y todos los métodos como pseudo-classes**

Los Stack frames que se muestran en todas las vistas, tales como caminos para GC Root, referencias salientes de threads, como pseudo-objects. Variables referencias locales en los marcos de pila se muestran como referencias salientes desde el frame. Esto hace que sea más fácil encontrar el que pila de marcos de mantener los objetos vivos. Los stack frames se les da un pseudo-type dependiendo basado en el método que se está ejecutando en el marco. Al ver el número de instancias de ese tipo seudo es fácil ver que los métodos se están ejecutando a través de todas los threads y qué métodos utilizan una gran cantidad de la pila. Esto puede ayudar a resolver *StackOverflowErrors*. Todos los demás métodos también se crean como objetos de clase de pseudo. El tamaño del método objeto pseudo-clase es el tamaño del código de byte y el código compilado JIT, que en otros modos se acumula en el tamaño de la clase que define. Esto hace que sea más fácil encontrar el método que consumen una gran cantidad de memoria no heap de código de bytes y el código compilado JIT.

**El funcionamiento normal con Stack frames no se considera como objetos.**

core.20100112.141124.11580.0001.dmp.zip		
Overview thread_overview list_objects [selection of 'main']		
Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
java.lang.Thread @ 0xcb16b8 main Thread	96	1,344
<class> class java.lang.Thread @ 0xcb0ef8 System Class, JNI Global	1,985	2,065
lock java.lang.Thread\$ThreadLock @ 0xcb1718	16	16
threadGroup java.lang.ThreadGroup @ 0xcb1728 main	72	176
threadLocals java.lang.ThreadLocal\$ThreadLocalMap @ 0xcb1c10	24	648
name java.lang.String @ 0xcb1e38 main	32	56
<Java Local> java.io.FileDescriptor @ 0xcbd078	40	40
contextClassLoader sun.misc.Launcher\$AppClassLoader @ 0xcfd6d0 System Class	104	27,376
accessControlContext java.security.AccessControlContext @ 0xcfd6e8	24	24
<Java Local> java.io.BufferedInputStream @ 0xcfe030	40	40
<Java Local> java.io.FileInputStream @ 0xcfe058	24	24
<Java Local> byte[8192] @ 0xcfeba0 .....	8,208	8,208
<Java Local> org.eclipse.mat.tests.CreateSampleDump\$DominatorTestData @ 0xd15c80	16	272
<Java Local> org.eclipse.mat.tests.CreateSampleDump\$ReferenceTestData @ 0xd16da0	32	240
<Java Local> byte[1] @ 0xd165f8 .	24	24
<Java Local> char[1] @ 0xd16610 \u0000	24	24
Σ Total: 15 entries		

## Stack frames como pseudo-objects.

Tenga en cuenta que el tipo de stack frame es <stack frame>.

core.20100112.141124.11580.0001.dmp.zip		
Overview thread_overview list_objects [selection of 'main']		
Class Name	Shallow Heap	
<Regex>	<Numeric>	
java.lang.Thread @ 0xcb16b8 main Thread	96	
<class> class java.lang.Thread @ 0xcb0ef8 System Class, JNI Global	1,985	
<Java Stack Frame> <stack frame> @ 0x11b8d0 java.io.FileInputStream.readBytes([BII)I (FileInputStream.java)	0	
<class> class <stack frame> @ 0x1cb0018	0	
<Java Local> java.io.FileDescriptor @ 0xcbd078	40	
<Java Local> java.io.FileInputStream @ 0xcfe058	24	
<Java Local> byte[8192] @ 0xcfeba0 .....	8,208	
Σ Total: 4 entries		
<Java Stack Frame> <stack frame> @ 0x11b8ec java.io.BufferedInputStream.fill()V (BufferedInputStream.java:229)	0	
<Java Stack Frame> <stack frame> @ 0x11b910 java.io.BufferedInputStream.read1([BII)I (BufferedInputStream.java:269)	0	
<Java Stack Frame> <stack frame> @ 0x11b93c java.io.BufferedInputStream.read([BII)I (BufferedInputStream.java:328)	0	
<Java Stack Frame> <stack frame> @ 0x11b964 com.ibm.jvm.io.ConsoleInputStream.read([BII)I (ConsoleInputStream.java:177)	0	
<Java Stack Frame> <stack frame> @ 0x11b978 com.ibm.jvm.io.ConsoleInputStream.read([B)I (ConsoleInputStream.java:167)	0	
<Java Stack Frame> <stack frame> @ 0x11b98c com.ibm.jvm.io.ConsoleInputStream.read()I (ConsoleInputStream.java:160)	0	
<Java Stack Frame> <stack frame> @ 0x11b9a8 org.eclipse.mat.tests.CreateSampleDump.main([Ljava/lang/String;)V (CreateSampleDump.java:31)	0	
lock java.lang.Thread\$ThreadLock @ 0xcb1718	16	
threadGroup java.lang.ThreadGroup @ 0xcb1728 main	72	
threadLocals java.lang.ThreadLocal\$ThreadLocalMap @ 0xcb1c10	24	
name java.lang.String @ 0xcb1e38 main	32	
contextClassLoader sun.misc.Launcher\$AppClassLoader @ 0xcfd6d0 System Class	104	
accessControlContext java.security.AccessControlContext @ 0xcfd6e8	24	
Σ Total: 15 entries		

## Stack frames como pseudo-objects y métodos que se ejecutan como pseudo-classes



Nota los diferentes tipos para el marco de pila tal como java.io.InputStream.getBytes ([B]I) I ;.

Class Name	Shallow Heap
<Regex>	<Numeric>
java.lang.Thread @ 0xcb16b8 main Thread	96
<class> class java.lang.Thread @ 0xcb0ef8 System Class, JNI Global	1,985
<Java Stack Frame> java.io.FileInputStream.readBytes([B]I) @ 0x11b8d0 (FileInputStream.java)	256
<class> class java.io.FileInputStream.readBytes([B]I) @ 0x1cb5654	0
<Java Local> java.io.FileDescriptor @ 0xcbd078	40
<Java Local> java.io.FileInputStream @ 0xcfe058	24
<Java Local> byte[8192] @ 0xcfeba0	8,208
<b>Total: 4 entries</b>	
<Java Stack Frame> java.io.BufferedInputStream.fill()V @ 0x11b8ec (BufferedInputStream.java:229)	28
<Java Stack Frame> java.io.BufferedInputStream.read1([B]I) @ 0x11b910 (BufferedInputStream.java:269)	36
<Java Stack Frame> java.io.BufferedInputStream.read([B]I) @ 0x11b93c (BufferedInputStream.java:328)	44
<Java Stack Frame> com.ibm.jvm.io.ConsoleInputStream.read([B]I) @ 0x11b964 (ConsoleInputStream.java:177)	40
<Java Stack Frame> com.ibm.jvm.io.ConsoleInputStream.read([B]I) @ 0x11b978 (ConsoleInputStream.java:167)	20
<Java Stack Frame> com.ibm.jvm.io.ConsoleInputStream.read()I @ 0x11b98c (ConsoleInputStream.java:160)	20
<Java Stack Frame> org.eclipse.mat.tests.CreateSampleDump.main([Ljava/lang/String;)V @ 0x11b9a8 (CreateSampleDump.java:31)	28
lock java.lang.Thread\$ThreadLock @ 0xcb1718	16
threadGroup java.lang.ThreadGroup @ 0xcb1728 main	72
threadLocals java.lang.ThreadLocal\$ThreadLocalMap @ 0xcb1c10	24
name java.lang.String @ 0xcb1e38 main	32
contextClassLoader sun.misc.Launcher\$AppClassLoader @ 0xcfd6d0 System Class	104
accessControlContext java.security.AccessControlContext @ 0xcfd6a8	24
<b>Total: 15 entries</b>	

El histograma muestra que clase métodos sólo se ejecutan son pseudo-clases, y el tamaño de la clase de objeto es 0.

Class Name	Objects	Shallow Heap	Retained Heap
<Numeric>	<Numeric>	<Numeric>	<Numeric>
com.ibm.misc.SignalDispatcher.waitForSignal()I	1	256	
java.io.FileInputStream.readBytes([B]I)	1	256	
java.io.BufferedInputStream.read([B]I)	1	44	
com.ibm.jvm.io.ConsoleInputStream.read([B]I)	1	40	
java.io.BufferedInputStream.read1([B]I)	1	36	
com.ibm.misc.SignalDispatcher.run()V	1	32	
java.io.BufferedInputStream.fill()V	1	28	
org.eclipse.mat.tests.CreateSampleDump.main([Ljava/lang/String;)V	1	28	
com.ibm.jvm.io.ConsoleInputStream.read([B]I)	1	20	
com.ibm.jvm.io.ConsoleInputStream.read()I	1	20	
<b>Total: 10 entries (415 filtered)</b>	<b>10</b>	<b>760</b>	

**Stack frames como pseudo-objects y todos los métodos como pseudo-classes.**

El árbol referencias saliente tiene el mismo aspecto, pero el histograma clase tiene muchas más pseudo-clases con 0 casos (es decir, sin métodos de funcionamiento), y de los objetos de la clase de pseudo tener un tamaño distinto de cero.



Inspector

@ 0x1cb5654  
readBytes([BII)I  
java.io.FileInputStream  
class <method type> @ 0x1cb0018  
<method>  
com.ibm.oti.vm.BootstrapClassLoader @ 0xcf...  
258 (shallow size)  
258 (retained size)  
no GC root

Statics	Attributes	Class Hierarchy	Value
Type	Name	Value	
ref	declaringClass	class java.io.FileInputStream	

core.20100112.141124.11580.0001.dmp.zip

Overview thread\_overview list\_objects [selection of 'main'] Histogram

Class Name	Objects	Shallow Heap	Retained Heap
<Numeric>	<Numeric>	<Numeric>	<Numeric>
com.ibm.misc.SignalDispatcher.waitForSignal()I	1	256	
java.io.FileInputStream.readBytes([BII)I	1	256	
java.io.BufferedInputStream.read([BII)I	1	44	
com.ibm.jvm.io.ConsoleInputStream.read([BII)I	1	40	
java.io.BufferedInputStream.read1([BII)I	1	36	
com.ibm.misc.SignalDispatcher.run()V	1	32	
java.io.BufferedInputStream.fill()V	1	28	
org.eclipse.mat.tests.CreateSampleDump.main([Ljava/lang/String;)V	1	28	
com.ibm.jvm.io.ConsoleInputStream.read()I	1	20	
com.ibm.jvm.io.ConsoleInputStream.read([B)I	1	20	
java.lang.Object.<init>()V	0	0	
java.lang.Object.equals(Ljava/lang/Object;)Z	0	0	
java.lang.Object.finalize()V	0	0	
<b>Total: 13 of 5,124 entries (415 filtered)</b>	<b>10</b>	<b>760</b>	

# El análisis de Java Collection Uso

Una colección es un objeto que se utiliza para almacenar, recuperar y manipular los datos. Memory Analyzer ofrece las siguientes consultas para analizar colecciones de Java:

<b>Array Fill Ratio Query</b>	Imprime una distribución de frecuencia de los coeficientes de llenado de matrices no primitivos. La relación de llenado es la proporción de elementos no nulos en la matriz. Las matrices se acumulan entonces en tantos segmentos como parametrizado. Matrices primitivas no pueden tener valores nulos por lo que esta consulta funciona sólo en tablas de objetos.
<b>Arrays Grouped by Size Query</b>	Histograma de distribución de las matrices dadas agrupados por el tamaño.

<b>Collection Fill</b> <b>Ratio Query</b>	<p>Imprime una distribución de frecuencia de los coeficientes de relleno de colecciones dadas. Los siguientes colecciones se pueden utilizar para la consulta:</p> <ul style="list-style-type: none"> <li>• java.util.ArrayList</li> <li>• java.util.HashMap</li> <li>• java.util.Hashtable</li> <li>• java.util.Properties</li> <li>• java.util.Vector</li> <li>• java.util.WeakHashMap</li> <li>• java.util.concurrent.ConcurrentHashMap \$ Segmento</li> <li>• java .beans.beancontext.BeanContextSupport</li> <li>• java.lang.ThreadLocal \$ ThreadLocalMap</li> <li>• java.util.ArrayDeque</li> <li>• java.util.HashSet</li> <li>• java.util.IdentityHashMap</li> <li>• java.util.PriorityQueue</li> <li>• java.util.Vector</li> <li>• java.util.WeakHashMap</li> <li>• java.util.concurrent.ConcurrentHashMap</li> <li>• java.util.concurrent.CopyOnWriteArrayList</li> <li>• java.util.concurrent.CopyOnWriteArraySet</li> <li>• java.util.concurrent.DelayQueue</li> <li>• java.util.jar.Attributes</li> <li>• javax.script.SimpleBindings</li> </ul> <p>Una colección personalizada adicional (por ejemplo no JDK) de recogida puede ser especificado por el ' colección', 'size_attribute' y 'argumento array_attribute'.</p>
--	---

<b>Collections Grouped By Size Query</b>	<p>Histograma de distribución de las colecciones dadas por su tamaño. Los siguientes colecciones se pueden utilizar para la consulta. Colecciones conocidas:</p> <ul style="list-style-type: none"> <li>• java.util.ArrayList</li> <li>• java.util.TreeMap</li> <li>• java.util.HashMap</li> <li>• java.util.Hashtable</li> <li>• java.util.Properties</li> <li>• java.util.Vector</li> <li>• java.util.WeakHashMap</li> <li>• java.util.concurrent.ConcurrentHashMap</li> <li>• java.util.concurrent.ConcurrentHashMap \$ Segmento</li> </ul> <ul style="list-style-type: none"> <li>• java.util.ArrayDeque</li> <li>• java.util.HashSet</li> <li>• java.util.IdentityHashMap</li> <li>• java.util.LinkedList</li> <li>• java.util.PriorityQueue</li> <li>• java.util.TreeSet</li> <li>• java.util.concurrent.ConcurrentSkipListMap</li> <li>• java.util.concurrent.ConcurrentSkipListSet</li> <li>• java.util.concurrent.CopyOnWriteArrayList</li> <li>• java.util.concurrent.CopyOnWriteArraySet</li> <li>• Java.util.concurrent.DelayQueue</li> <li>• java.util.concurrent.LinkedBlockingDeque</li> <li>• java.util.concurrent.LinkedBlockingQueue</li> <li>• java.util.concurrent.SynchronousQueue</li> <li>• java.util.jar.Attributes</li> <li>• java.beans.beancontext.BeanContextSupport</li> <li>• java.lang.ThreadLocal \$ ThreadLocalMap</li> <li>• javax.script.SimpleBindings</li> <li>• javax.swing.UIDefaults</li> </ul> <p>Una colección personalizada adicional colección(por ejemplo, no JDK) puede ser especificado por el 'recogida', 'size_attribute' y el argumento 'array_attribute'.</p>
<b>Extract List Values Query</b>	<p>Listas elementos de un solo LinkedList, ArrayList, Vector, CopyOnWriteArrayList, PriorityQueue,ArrayDeque. objeto</p>

<b>Hash Entries Query</b>	Extrae los pares de valores clave de mapas de patata y tablas hash.
<b>Extract Hash Set Value Query</b>	muestra elementos de un mismo HashSet.
<b>Map Collision Ratio Query</b>	<p>Imprime una distribución de frecuencias de los coeficientes de colisión de las colecciones de mapas similares. Las siguientes colecciones de mapas similares se pueden utilizar para la consulta.</p> <ul style="list-style-type: none"> <li>• java.util.HashMap</li> <li>• java.util.Properties</li> <li>• java.util.Hashtable</li> <li>• java.util.WeakHashMap</li> <li>• java.util.concurrent.ConcurrentHashMap \$ Segmento</li> <li>• java.util.HashMap</li> <li>• java.util.HashSet</li> <li>• java.util.Hashtable</li> <li>• java.util.IdentityHashMap</li> <li>• java .util.concurrent.ConcurrentHashMap</li> <li>• java.util.concurrent.ConcurrentSkipListMap</li> <li>• java.util.concurrent.ConcurrentSkipListSet</li> <li>• java.util.jar.Attributes</li> <li>• java.beans.beancontext.BeanContextSupport</li> <li>• java.lang.ThreadLocal \$ ThreadLocalMap</li> <li>• javax.script.SimpleBindings</li> <li>• javax.swing.UIDefaults</li> </ul> <p>Un customizado adicional en forma de mapa (por ejemplo, no JDK) de recogida puede ser especificado por el 'recogida', 'size_attribute' y el argumento 'array_attribute'</p>
<b>Primitive Arrays with a Constant Value</b>	Listas todas las matrices primitivas (a partir de una selección) que están llenos de uno y el mismo valor

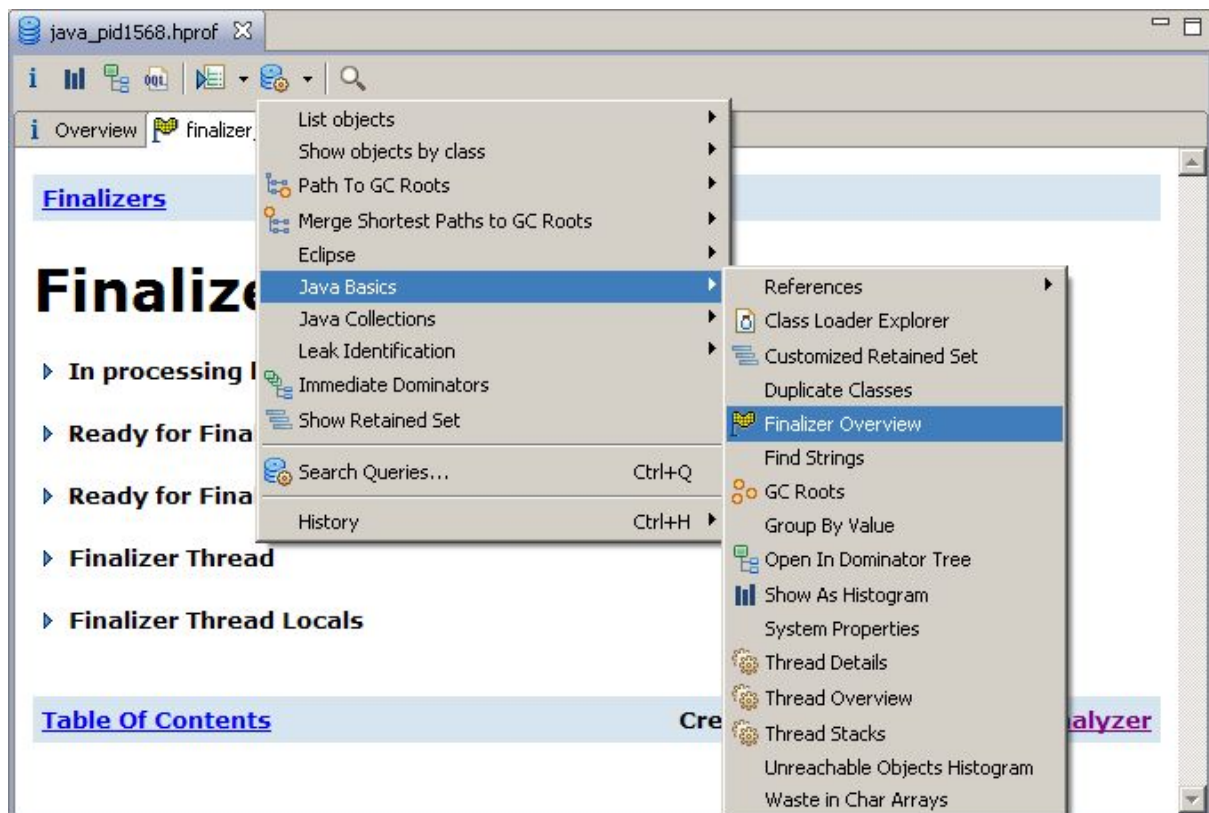
Todas estas consultas son accesibles desde el menú desplegable de la barra de herramientas: **Open Query Browser > Java Collections**





# Analizando el Finalizer

Los *Finalizers* se ejecutan cuando el GC limpia los objetos. Puesto que usted no tiene control sobre la ejecución *finalize*, se recomienda no utilizarlos. Debido a que la memoria sólo puede ser liberada cuando el método *finalize* terminado, puede bloquear la recolección del GC. Para obtener una visión de estos utilice *Finalizer Overview Query*:



Esta consulta incluye las siguientes sub consultas:

## Finalizers en el ejecución

Objeto actualmente procesada por Finalizer Thread. Esta consulta devuelve el objeto actualmente procesado por el subproceso finalizador si los hay. El objeto devuelto puede ser procesado por una de las siguientes

- razones:es el bloqueo
- que está ejecutando larga
- que la cola era finalizador o todavía está lleno.

Utilice la consulta cola finalizador para comprobar la cola.

### **Ready for Finalizer Thread**

Esta consulta muestra los objetos listos para su finalización en su orden de procesamiento. Siguiendo razones pueden causar una cola finalizador completo:

- El objeto que se está procesando está bloqueando o larga duración (por favor utilice nuestro finalizador en la consulta de procesamiento para comprobar).
- El uso de aplicaciones hecha de demasiados objetos con finalize (), que están en la cola en la memoria.

Además se proporciona un resumen de nivel de clase de los objetos

### **Finalizer Thread**

Esta consulta muestra el hilo daemon que realiza las finalizaciones de objeto.

### **Finalizer Thread Locals**

Esta consulta muestra los thread locales del daemon thread que están ejecutando las finalizaciones de objeto. Si los hay, esto indica mal uso en al menos uno de los finalizadores procesados y podría causar problemas graves de memoria que mantenga permanentemente por el subproceso finalizador o finalizador procesado bajo los locales de rosca inútiles dañar lógica de la aplicación)

# Comparación de objetos

## Introducción

Antes de investigar las posibilidades que ofrece la memoria del analizador en el área de comparación hagamos alguna explicación. ID de objeto que se proporcionan en los formatos soportados por volcado de pila MAT son sólo las direcciones en las que se ubican los objetos. A medida que los objetos se suelen mover y reordenados por la JVM durante un GC éstos abordan el cambio. Por lo tanto no pueden ser usados para comparar los objetos. Esto significa básicamente que si se comparan dos vuelcos de almacenamiento dinámico diferentes (aunque desde el mismo proceso), no es posible señalar el hormigón objetos diferentes entre los dos vuelcos de almacenamiento dinámico. Sin embargo, todavía se puede llevar a cabo la comparación de los resultados agregados (por ejemplo, el histograma de clase) y analizar cómo ha cambiado la cantidad de objetos y la memoria que toman.

Memory Analyzer ofrece la posibilidad de comparar no sólo los histogramas de clase mundial de dos vuelcos de almacenamiento dinámico diferentes, pero un número arbitrario de resultados de la tabla con formato - por ejemplo, los conjuntos retenidas de tres objetos diferentes. No importa si las tablas que se comparan provienen de uno y el mismo o diferentes vuelcos de almacenamiento dinámico.

Esto significa que uno tiene la posibilidad de hacer cosas como:

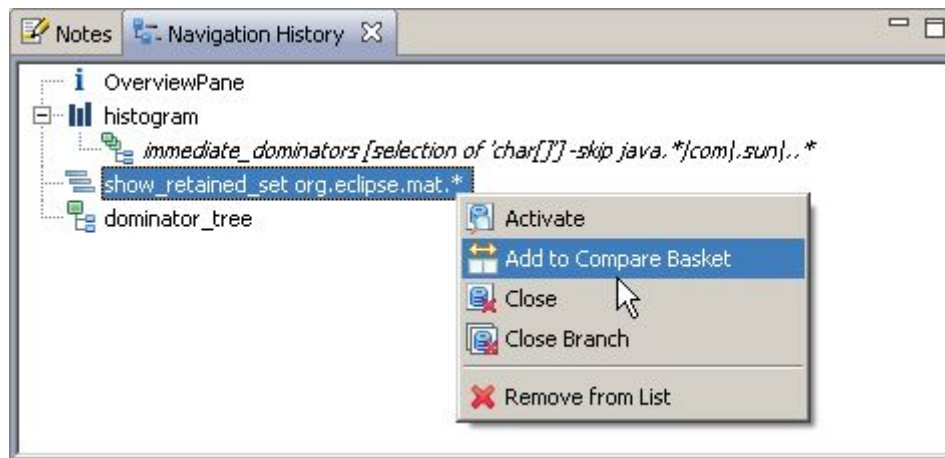
- Comparar el conjunto de retenida de un paquete específico a través de varios vuelcos de almacenamiento dinámico
- comparar cómo los conjuntos retenidos para los objetos de aplicación A1, A2 y A3 (todos en la misma descarga del heap) difieren entre sí

Aquí es una descripción rápida cómo comparar varios cuadros de conjunto retenidas.

### 1. **Move all tables to be compared to the Compare Basket**

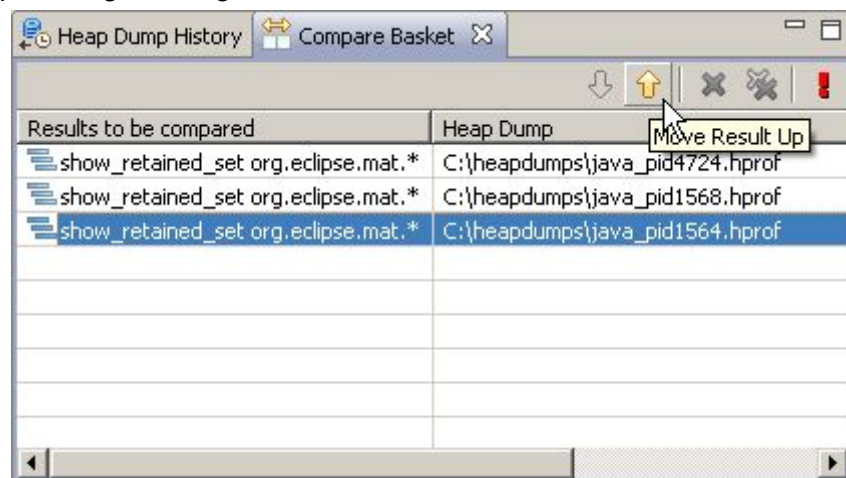
Todas las consultas que se ejecutan en la memoria del analizador se pueden ver en el historial de navegación Vista. Desde este punto de vista se puede añadir que los resultados se compararon con la cesta de comparación. El historial de navegación está todavía por volcado del heap, por lo tanto, si se quiere comparar las tablas de diferentes vuelcos de almacenamiento dinámico, entonces tienen que añadir uno a uno. Varias tablas de un volcado de pila pueden ser añadidos a la vez.

Los árboles pueden ser comparados, así, a pesar de que se convierten en tablas con el objetivo de la comparación. Los resultados Object Query Language (OQL) también se puede comparar, aunque sólo el último resultado de cada editor de OQL puede ser añadido a la cesta de comparación. Si otra consulta se emite a continuación el resultado anterior se eliminará de la canasta. Si dos resultados OQL necesitan ser comparados a continuación, se deben abrir dos editores OQL.



## 1. Modificar el orden de las tablas

Uso de la barra de herramientas de la cesta de comparación se puede modificar el orden en que se deben comparar las tablas, es decir, seleccionar qué resultado debe ser la línea de base, que ocupa el segundo lugar, etc ...

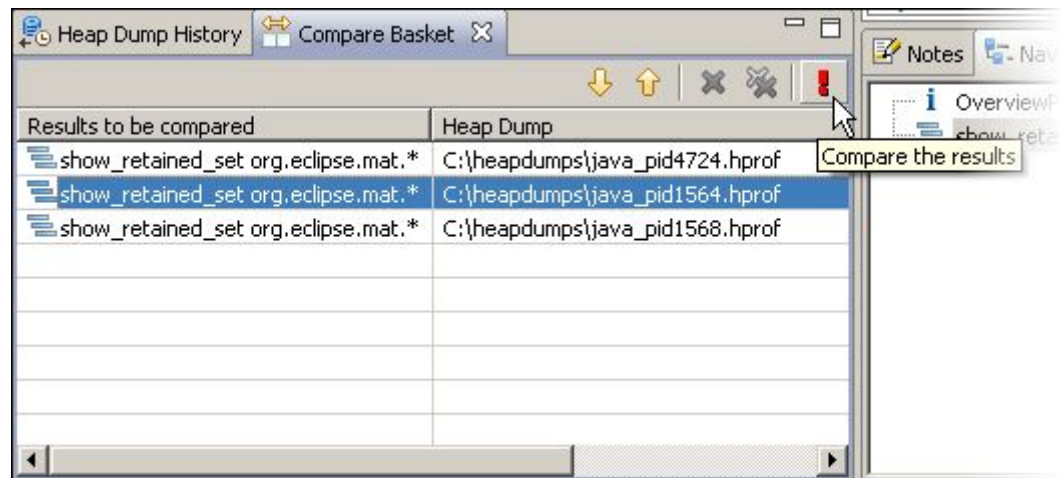


## 2. Ejecutar la comparación

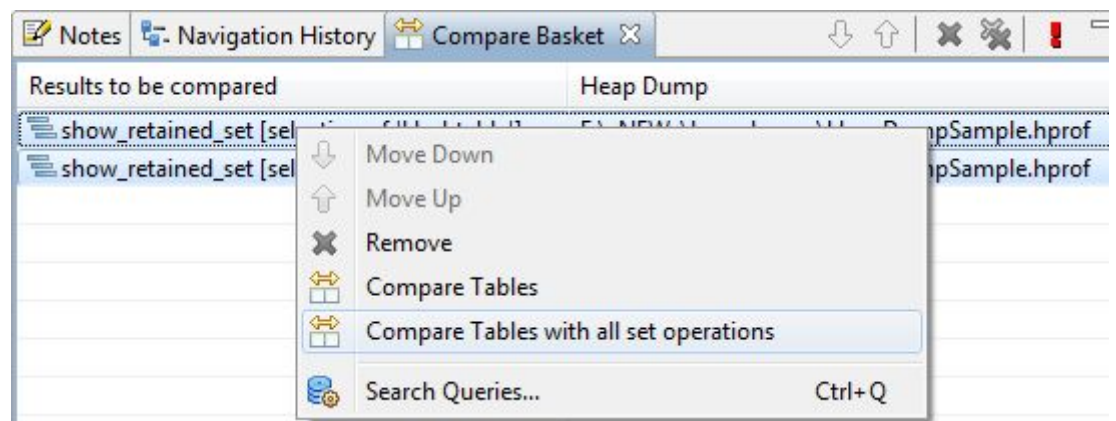
- Una vez que se logra el orden preferido simplemente haga clic en el botón de ejecución ...



○



- o para comparar un subconjunto de las tablas, que aparezca el menú contextual de las entradas de la tabla seleccionada. Al comparar las tablas de una y la misma volcado del heap, ahora es posible realizar diferentes operaciones de conjunto sobre el resultado de la comparación.

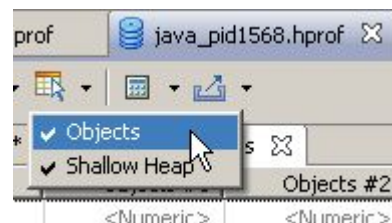
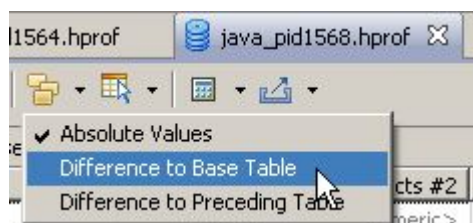


... y ver el resultado.

Class Name	Objects #1	Objects #2	Objects #3	Shallow Heap #1	Shallow Heap #2	Shallow Heap #3
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
char[]	150	136	135	43.888	43.104	42.976
byte[]	4	4	4	16.480	16.480	16.480
java.util.HashMap\$Entry[]	43	38	39	3.568	3.168	3.248
java.lang.String	145	126	125	3.480	3.024	3.000
java.util.HashMap	31	26	27	1.240	1.040	1.080
java.util.LinkedHashMap\$Entry	36	36	36	1.152	1.152	1.152
org.eclipse.mat.snapshot.SnapshotInfo	18	13	14	1.008	728	784
java.util.HashMap\$Entry	38	34	34	912	816	816
java.lang.Object[]	23	24	23	904	960	904
java.util.LinkedHashMap	12	12	12	576	576	576
org.eclipse.swt.widgets.Shell	2	2	2	528	560	528
java.util.LinkedList\$Entry	20	15	16	480	360	384
org.eclipse.mat.ui.SnapshotHistoryService\$E...	19	14	15	456	336	360
java.lang.reflect.Field	6	6	6	432	432	432
java.util.Date	18	13	14	432	312	336
org.eclipse.swt.custom.StyledText	1	1	1	352	360	352
org.eclipse.mat.ui.rcp.actions.AddHistoryTo...	10	10	10	320	320	320
java.util.ArrayList	13	14	13	312	336	312
java.lang.Long	19	14	15	304	224	240
java.net.URL	5	6	5	280	336	280
org.eclipse.mat.ui.internal.browser.QueryCo...	2	2	2	240	240	240
org.eclipse.mat.internal.acquire.HeapDumpP...	5	5	5	240	240	240
org.eclipse.swt.widgets.Composite	2	2	2	224	240	224
org.eclipse.mat.hprof.acquire.LocalJavaProc...	2	2	2	224	224	224
sun.reflect.annotation.AnnotationInvocation...	8	8	8	192	192	192
org.eclipse.core.runtime.ListenerList	11	11	11	176	176	176
<b>Σ Total: 26 of 161 entries</b>	<b>865</b>	<b>809</b>	<b>798</b>	<b>84.408</b>	<b>82.552</b>	<b>81.864</b>

### 3. Personalizar el resultado mostrado

Por defecto se muestran los valores absolutos de todas las tablas para cada propiedad en comparación, por ejemplo, número de objetos, el tamaño superficial, etc ... ahora se puede cambiar entre los deltas y los valores absolutos, así como seleccionar las columnas que se deben comparar:



<div> <div>java_pid4724.hprof</div> <div>java_pid1564.hprof</div> <div>java_pid1568.hprof</div> </div>				
<div> <div>Overview</div> <div>Retained by 'org.eclipse.mat.*'</div> <div>Compared Tables</div> </div>				
Class Name	Shallow Heap #1	Shallow Heap #2	Shallow Heap #3	
<Regex>	<Numeric>	<Numeric>	<Numeric>	
char[]	43.888	-784	-912	
byte[]	16.480	+0	+0	
java.util.HashMap\$Entry[]	3.568	-400	-320	
java.lang.String	3.480	-456	-480	
java.util.HashMap	1.240	-200	-160	
java.util.LinkedHashMap\$Entry	1.152	+0	+0	
org.eclipse.mat.snapshot.SnapshotInfo	1.008	-280	-224	
java.util.HashMap\$Entry	912	-96	-96	
java.lang.Object[]	904	+56	+0	
java.util.LinkedHashMap	576	+0	+0	
org.eclipse.swt.widgets.Shell	528	+32	+0	
java.util.LinkedList\$Entry	480	-120	-96	
org.eclipse.mat.ui.SnapshotHistoryService\$Entry	456	-120	-96	
java.lang.reflect.Field	432	+0	+0	
java.util.Date	432	-120	-96	
org.eclipse.swt.custom.StyledText	352	+8	+0	
org.eclipse.mat.ui.rcp.actions.AddHistoryToMenuAction\$2	320	+0	+0	
java.util.ArrayList	312	+24	+0	
java.lang.Long	304	-80	-64	
java.net.URL	280	+56	+0	
org.eclipse.mat.ui.internal.browser.QueryContextHelp	240	+0	+0	
org.eclipse.mat.internal.acquire.HeapDumpProviderDescriptor	240	+0	+0	
org.eclipse.swt.widgets.Composite	224	+16	+0	
org.eclipse.mat.hprof.acquire.LocalJavaProcessesUtils\$StreamC...	224	+0	+0	
sun.reflect.annotation.AnnotationInvocationHandler	192	+0	+0	
org.eclipse.core.runtime.ListenerList	176	+0	+0	
org.eclipse.mat.ui.rcp.actions.AddHistoryToMenuAction\$2\$1	160	+0	+0	
<b>Σ Total: 27 of 161 entries</b>	<b>84.408</b>	<b>-1.808</b>	<b>-2.496</b>	

#### 4. Menú contextual con operaciones de conjuntos

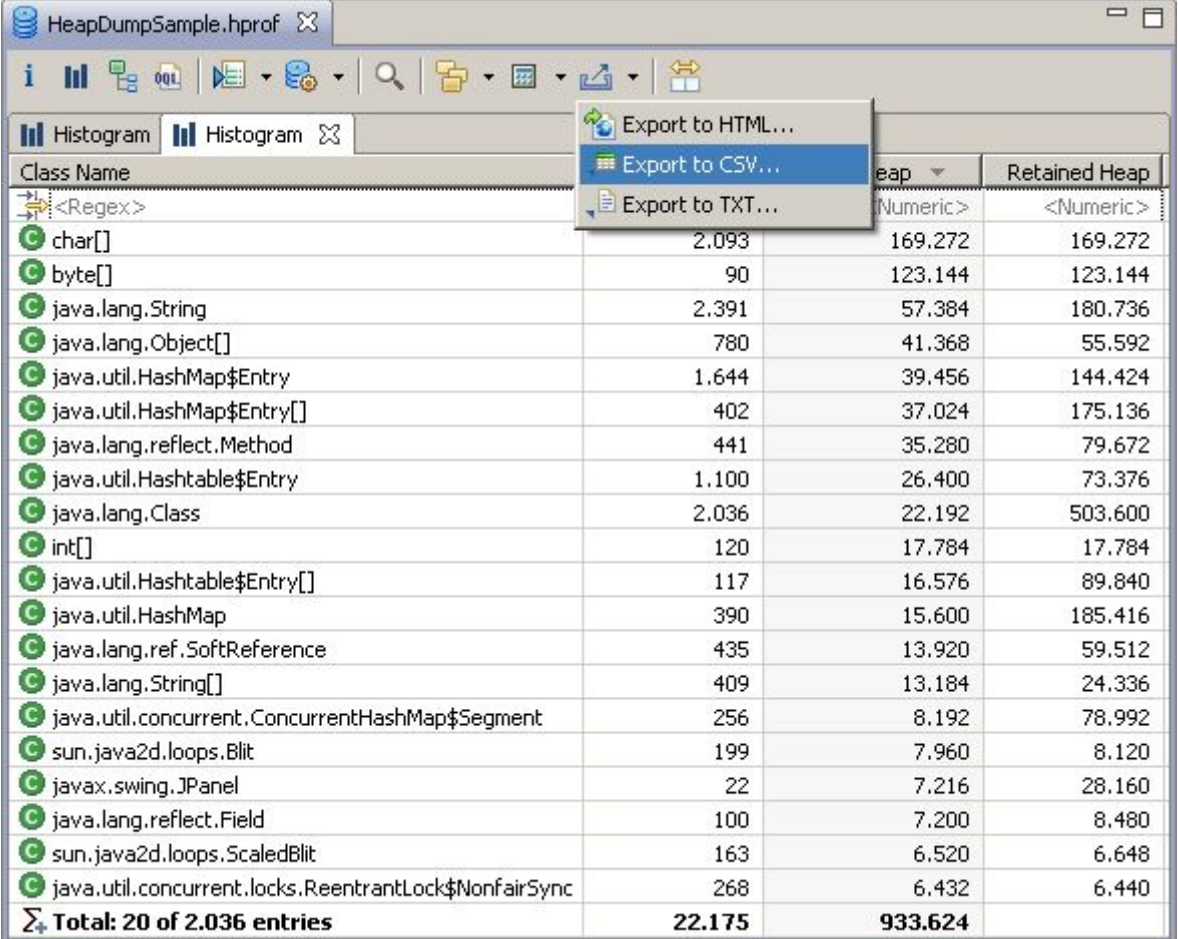
Si las tablas se han comparado mediante comparación tablas con todas las operaciones de ajuste, entonces es posible realizar diferentes operaciones de conjunto sobre el resultado de la comparación.

Retained by 'selection of 'Hashtable''		Retained by 'selection of 'HashMap''		comparetablesquery -setop ALL	
Class Name	Objects #0	Objects #1	Shallow Hea...	Shallow Hea...	
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>	
char[]	170	561	12,680	28,704	
java.util.Hash	Table 1		168	288	
java.util.Hash	Intersection of Table 1 and Table 2				List objects
java.lang.Strir	Union of Table 1 and Table 2				Show objects by class
java.util.Hash	Symmetric difference of Table 1 and Table 2				Merge Shortest Paths to GC Roots
sun.net.www.	Table 1 without Table 2				Java Basics
java.lang.Strir	Table 2 without Table 1				Java Collections
sun.font.Font	Table 2				Leak Identification
java.lang.ref.V	Columns...				Immediate Dominators
sun.java2d.De					Show Retained Set
java.text.Deci					Copy
java.lang.ref.PhantomRef...	4				Search Queries...
sun.font.FileFont\$FileFon...	3				Calculate Minimum Retained Size (q
java.text.DecimalFormatS...	1				Calculate Precise Retained Size
java.text.DigitList	1				
javax.swing.plaf.metal.M...	1				
sun.misc.Signal	1				

# Los datos de exportación

Una vez analizados los datos se pueden exportar desde el editor del heap, ya sea por:

- Usando el menú de exportación barra de herramientas (se puede elegir entre exportar a HTML, CSV y TXT)



The screenshot shows the 'HeapDumpSample.hprof' window with the 'Class Name' list. A context menu is open over the list, showing options: 'Export to HTML...', 'Export to CSV...', and 'Export to TXT...'. The list contains various Java classes and their memory statistics.

Class Name	Count	Heap	Retained Heap
<Regex>			
char[]	2,093	169,272	169,272
byte[]	90	123,144	123,144
java.lang.String	2,391	57,384	180,736
java.lang.Object[]	780	41,368	55,592
java.util.HashMap\$Entry	1,644	39,456	144,424
java.util.HashMap\$Entry[]	402	37,024	175,136
java.lang.reflect.Method	441	35,280	79,672
java.util.Hashtable\$Entry	1,100	26,400	73,376
java.lang.Class	2,036	22,192	503,600
int[]	120	17,784	17,784
java.util.Hashtable\$Entry[]	117	16,576	89,840
java.util.HashMap	390	15,600	185,416
java.lang.ref.SoftReference	435	13,920	59,512
java.lang.String[]	409	13,184	24,336
java.util.concurrent.ConcurrentHashMap\$Segment	256	8,192	78,992
sun.java2d.loops.Blit	199	7,960	8,120
javax.swing.JPanel	22	7,216	28,160
java.lang.reflect.Field	100	7,200	8,480
sun.java2d.loops.ScaledBlit	163	6,520	6,648
java.util.concurrent.locks.ReentrantLock\$NonfairSync	268	6,432	6,440
<b>Total: 20 of 2,036 entries</b>	<b>22,175</b>	<b>933,624</b>	

- Copiar y pegar para ver notas, correo electrónico o cualquier archivo basado en texto.
- Usando el menú contextual Copiar para copiar y pegar la dirección, nombre de la clase o el valor de un objeto. El contenido de grandes arreglos se pueden guardar en un archivo utilizando la opción Guardar Valor de opción de archivo.



HeapDumpSample.hprof

list\_objects [selection of 'String']

Class Name	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>
java.lang.String @ 0x26ba8a30 returnOpenType	24	64
java.lang.String @ 0x26ba8a30 returnOpenType	24	152
java.lang.String @ 0x26ba8a30 returnOpenType	24	152
java.lang.String @ 0x26ba8a30 returnOpenType	24	72
java.lang.String @ 0x26ba8a30 returnOpenType	24	56
java.lang.String @ 0x26ba8a30 returnOpenType	24	64
java.lang.String @ 0x26ba8a30 returnOpenType	24	48
java.lang.String @ 0x26ba8a30 returnOpenType	24	80
java.lang.String @ 0x26ba8a30 returnOpenType	24	64
java.lang.String @ 0x26ba8a30 returnOpenType	24	56
java.lang.String @ 0x26ba8a30 returnOpenType	24	136
java.lang.String @ 0x26ba8a30 returnOpenType	24	120
java.lang.String @ 0x26ba8a30 returnOpenType	24	88
java.lang.String @ 0x26ba8a30 returnOpenType	24	80
java.lang.String @ 0x26ba8a30 returnOpenType	24	80
java.lang.String @ 0x26ba8a30 returnOpenType	24	72
java.lang.String @ 0x26ba8a30 returnOpenType	24	152
java.lang.String @ 0x26ba8a30 returnOpenType	24	72
java.lang.String @ 0x26ba8a30 returnOpenType	24	88
java.lang.String @ 0x26ba8a30 returnOpenType	24	80
Total: 20 of 2.391 entries		