AVP-2928
WebSphere JVM Flight Simulator

WebSphere JVM Flight Simulator

What this lab is about

This lab is provided **AS-IS**, with no formal IBM support.

In this lab you will use IBM tools to monitor and diagnose JVM issues experienced by an 'in flight' running WebSphere Application Server. A badly implemented web application will be used to simulate common problems such as **memory leaks**, unexpected garbage collection cycles triggered by **System.gc()**, **large application objects** and **large HTTP session** sizes.

It will also describe the data needed to debug these issues, and introduce the Java problem determination tools available as part of the IBM Support Assistant (ISA) including Health Center, Garbage Collection Memory Visualizer and Memory Analyzer.

Lab requirements

List of system and software required for the attendee to complete the lab.

- WebSphere Application Server V7.0.0.13
- IBM Support Assistant V4.1 with the following tools installed:
  - IBM Monitoring and Diagnostic Tools for Java™ - Health Center
  - IBM Monitoring and Diagnostic Tools for Java™ - Memory Analyzer
  - IBM Monitoring and Diagnostic Tools for Java™ - Garbage Collection and Memory Visualizer
- IBM Extensions for Memory Analyzer

What you should be able to do

At the end of this lab you should be able to:-

- Install and configure Health Center to monitor a running WebSphere JVM
- Identify bugs in running code such as unnecessary calls to System.gc(), large object allocations and memory leaks
- Analyze a Javacore.txt file for evidence of a memory issue
- Analyze verbosegc logs to inspect memory usage and garbage collection performance
- Understand the basic techniques for debugging Java™ memory issues with Memory Analyzer
- Analyze a heap dump to determine those objects consuming the most heap space
- Use IBM extensions to Memory Analyzer to perform product specific memory analysis of a system dump

Part 1: Lab Set Up

_____ Login to the VMWare image with the username/password below:

Username :Administrator

Password : Happy2Be

NOTE:

Due to the physical memory on the VMware image being used for this lab please understand that certain operations may take time to perform – please be patient. The expected duration of this lab is **70 minutes** and each part has an estimated duration. Some lab sections and individual steps are marked **optional** so you can skip them to save time if necessary.

Part 2: Setup Health Center to Monitor a running WebSphere JVM (5 minutes)

NOTE:

Health Center is a free low-overhead diagnostic tool for monitoring applications running on an IBM Java Virtual Machine.

The Health Center tool is provided in two parts:

The **Health Center client** is a GUI-based diagnostics tool for monitoring the status of a running Java Virtual Machine (JVM). The Health Center client is installed into the IBM Support Assistant (ISA) workbench.

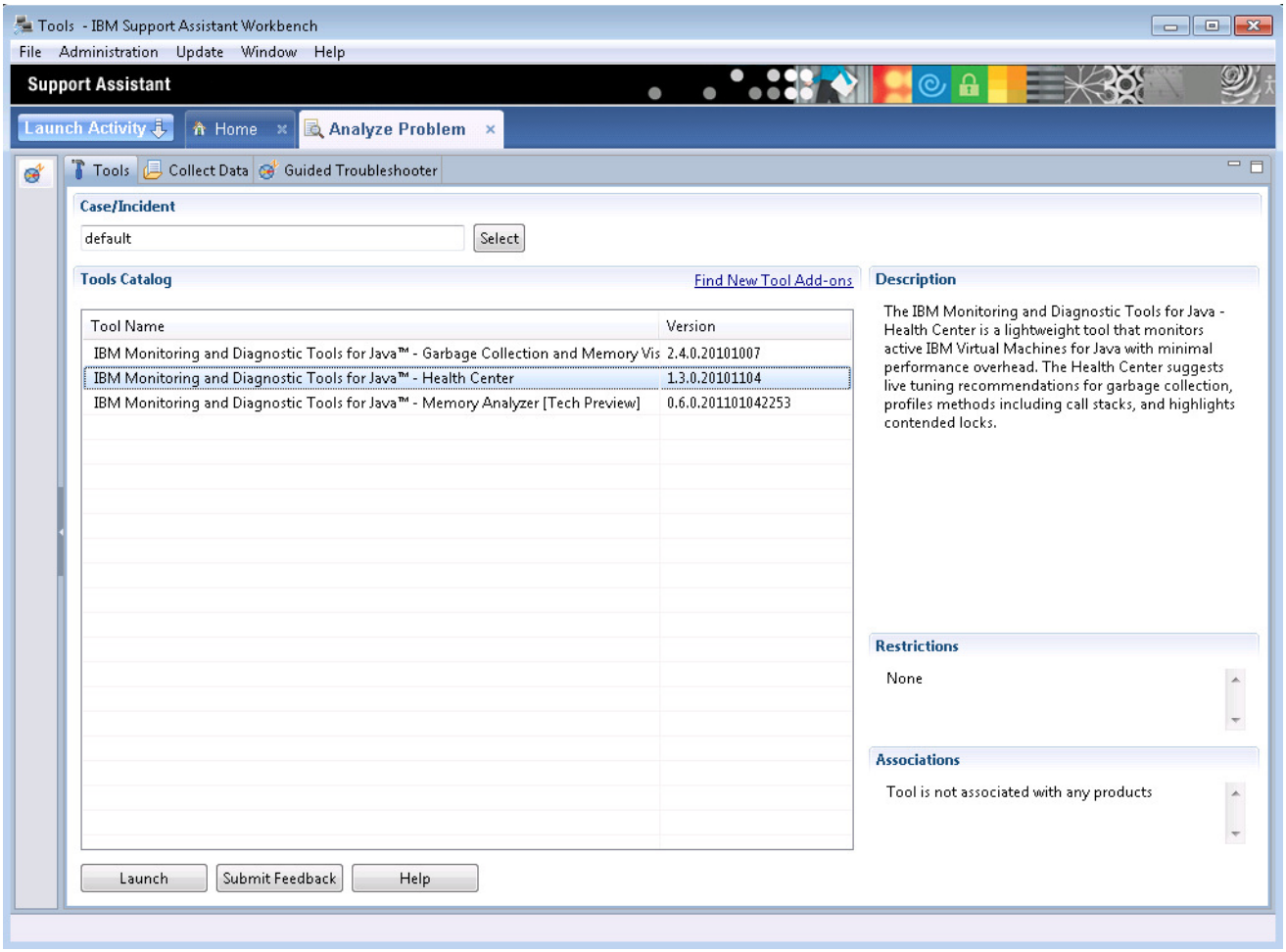
The **Health Center agent** provides the mechanism by which the Health Center client obtains information about your Java application.

The agent uses a small amount of processor time and memory on the server (less than 3%). It is installed by default in an IBM JVM at Java 5 SR8 or IBM Java 6 SR1 and above.

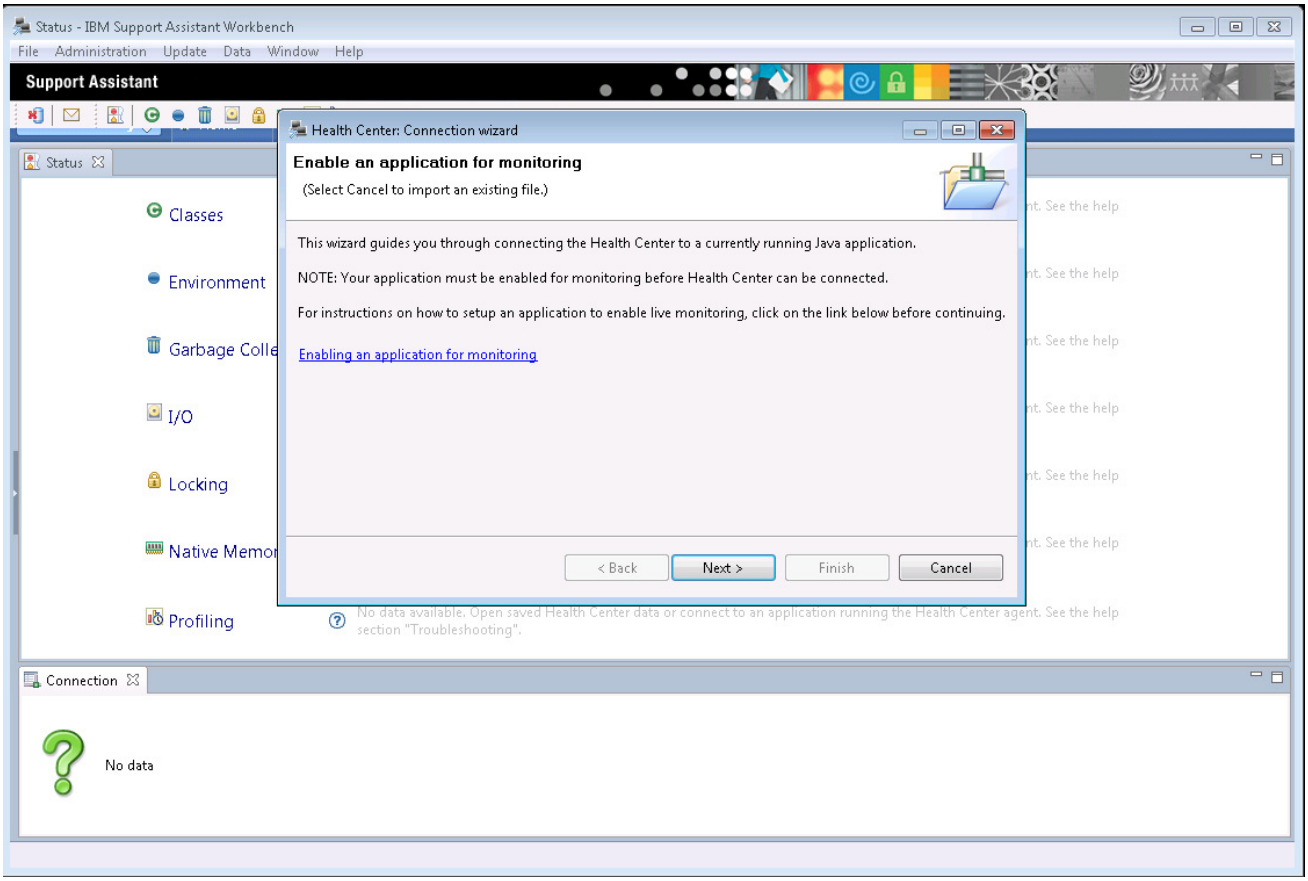
However, it is good practice to update the Health Center agent to benefit from the latest features in the client which is updated frequently. In addition, the agent must be manually enabled by setting a JVM property. Both these tasks are described later.

_____ ISA 4.1 and the Health Center client and agent are already installed. Double click the “IBM Support Assistant 4.1” shortcut on the desktop.

_____ Click “Launch Activity->Analyze Problem” and select the Health Center tool. Then click “Launch”.



_____ Click the “Enable an application for monitoring” link. This will display the help contents, including a link to the latest Health Center agent code.

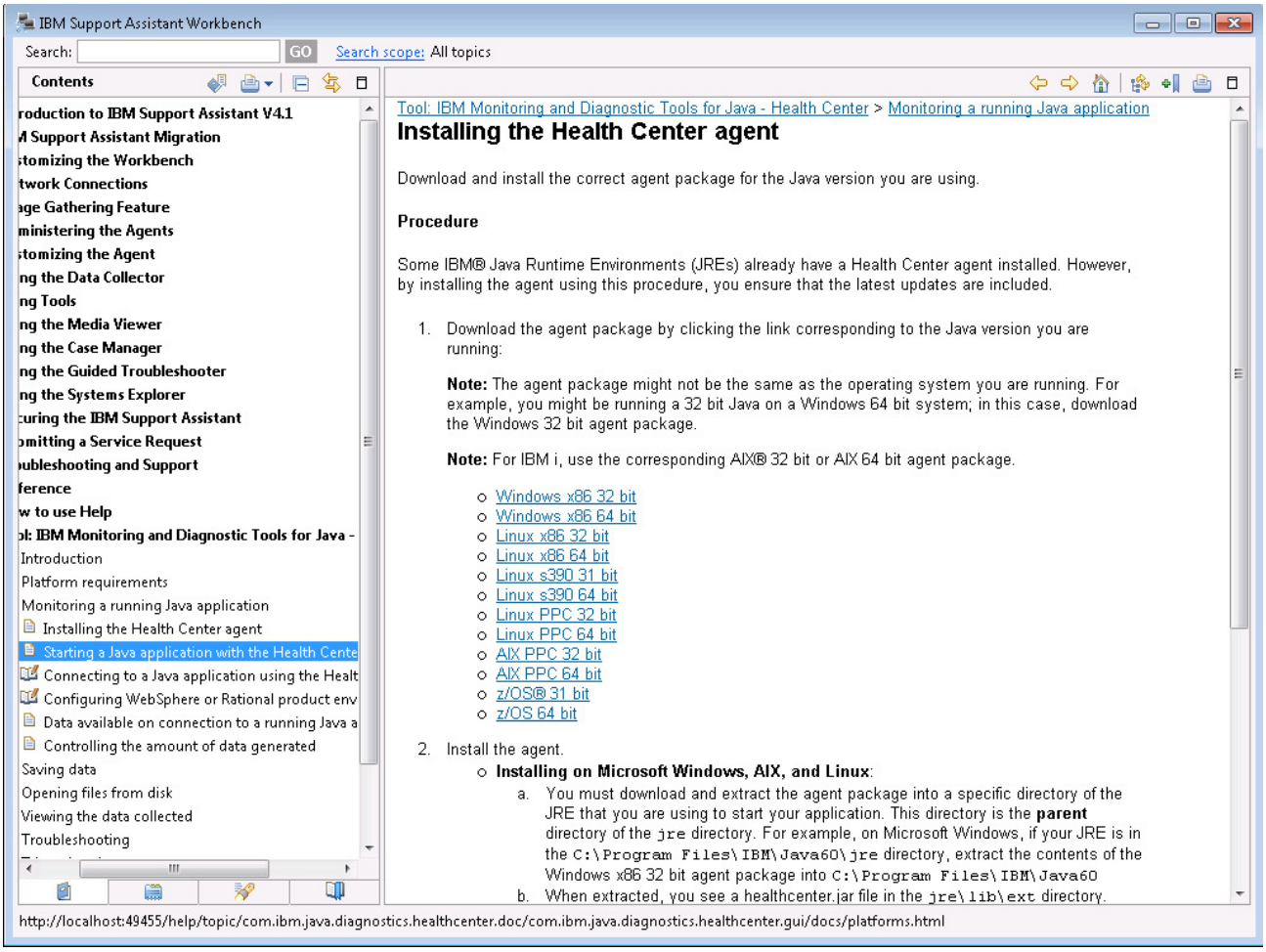


_____ Click the “Installing the Health Center agent” link.

_____ Observe that the Health Center client ships the latest agent code for various Java versions (do not click any links, see note below).

Note:

No action is required to update the agent code for this lab. To save time, the "Windows x86 32 bit" agent was already downloaded and installed. The installation process consists of extracting a small zip file over the existing WebSphere JRE. This updates the Health Center agent files originally shipped with the JVM to the latest version shipped with the Health Center client. **Only a few Health Center specific files are overwritten, not the entire JRE.**



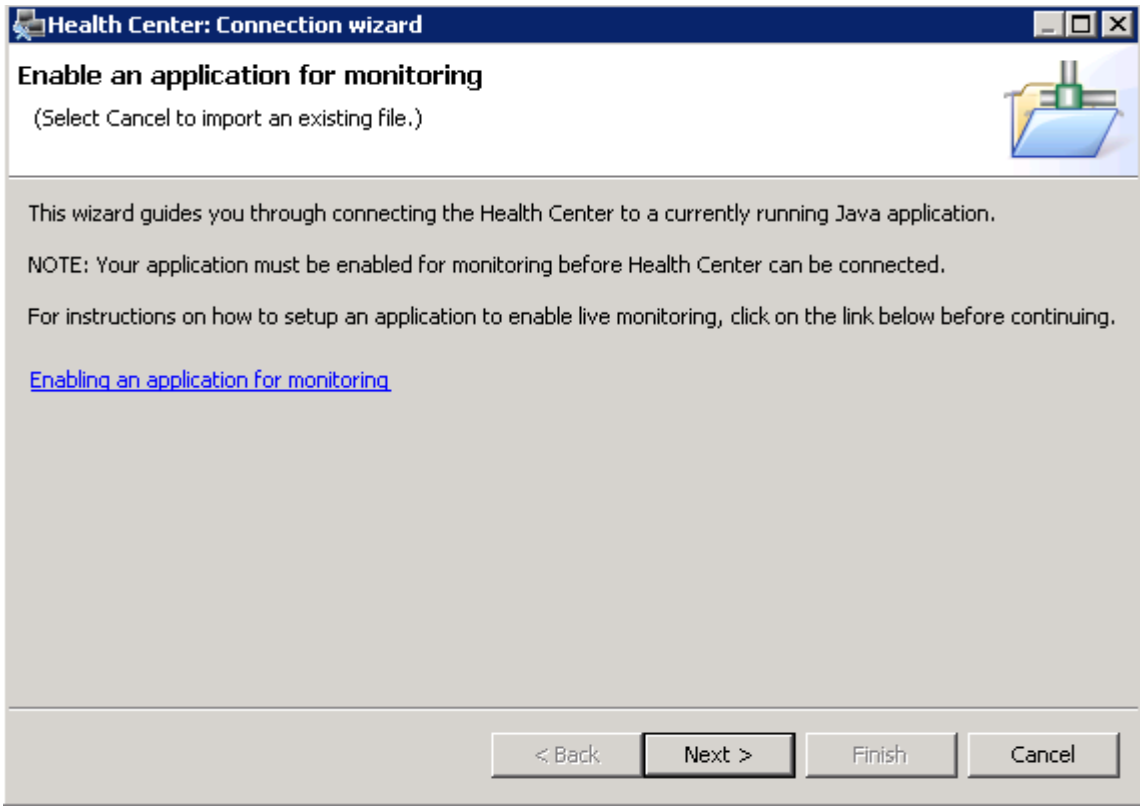
_____ Close the window showing help page "Installing the Health Center agent".

Note:

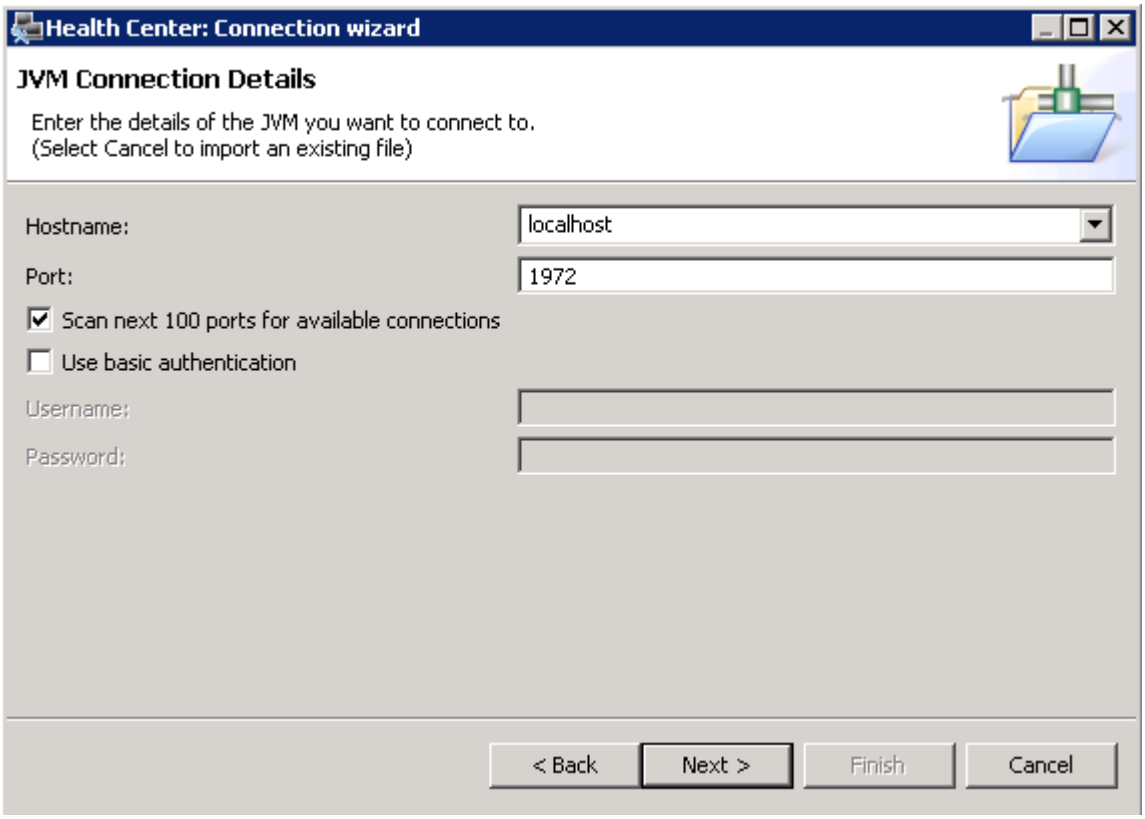
To save time, WebSphere Application Server has already been configured with "**-Xhealthcenter**" as a "Generic JVM Argument" via the WebSphere administration console. This enables the Health Center agent on the default port number (1972). If this port was in use, or if you needed to monitor multiple JVMs

on the same host, the port number can be customized with "**-Xhealthcenter:port=<port_number>**". No action is required for this lab.

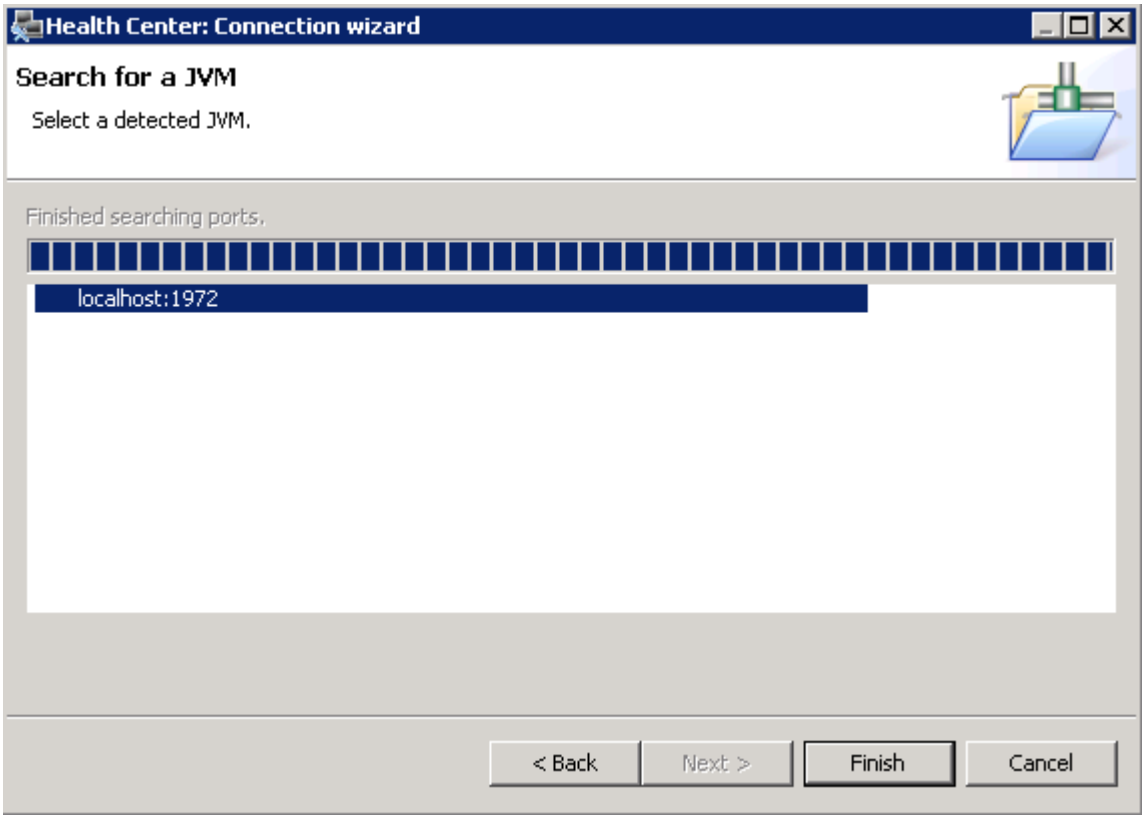
_____ Connect the Health Center client to the agent in the WebSphere JVM (which is already running in this lab). Return to the "Health Center Connection Wizard" window and click next to scan for available connections.



_____ Connect on the default port number by clicking "Next", and "Next" again



_____ Click “Finish” to start the data collection, the “Connection” panel will confirm the connection status.


Health Center: Connection wizard
Search for a JVM
Select a detected JVM.
Finished searching ports.
localhost:1972
< Back
Next >
Finish
Cancel


Connection
localhost:1972
4.3 MB received
Last updated 06:21:02

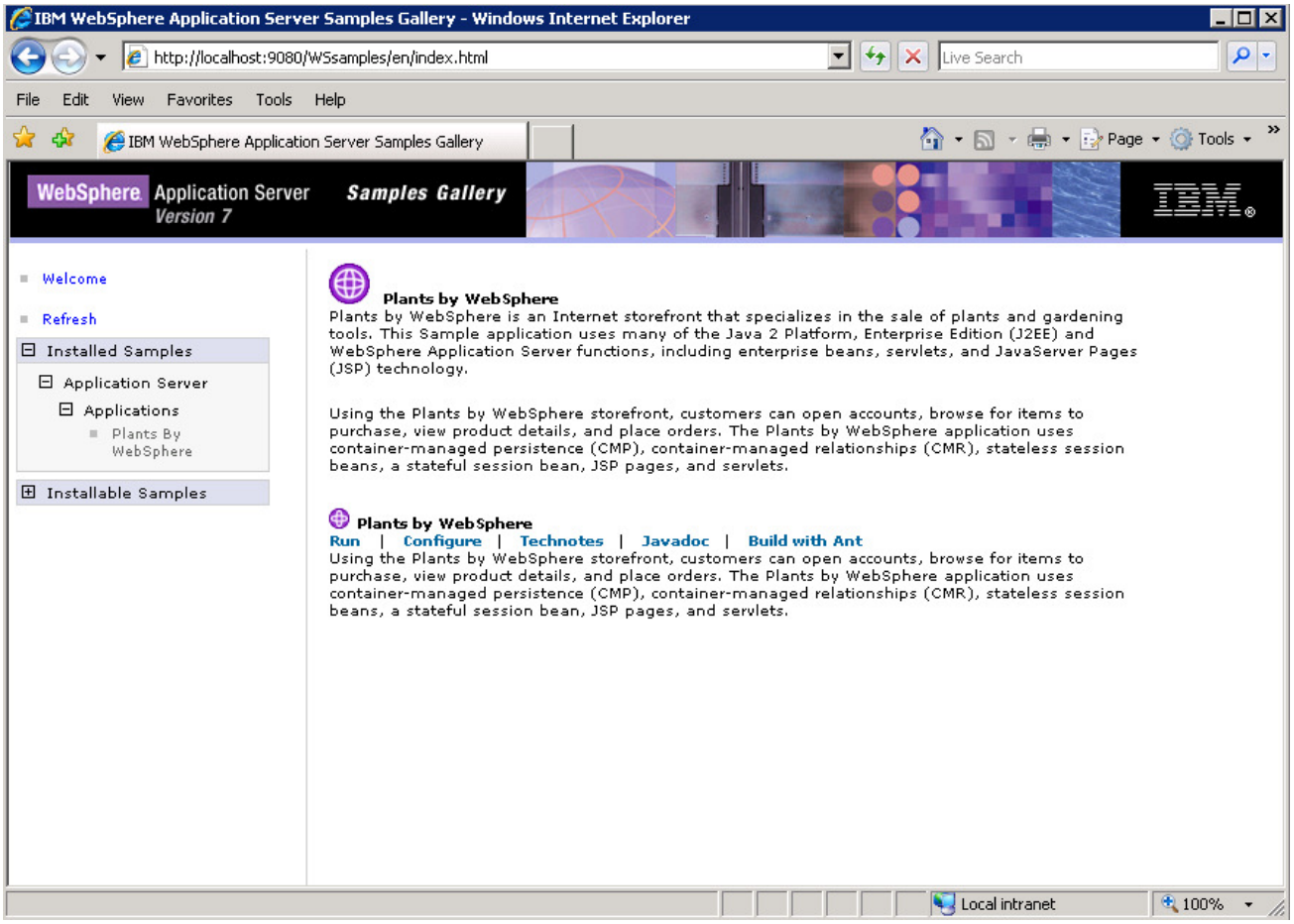
Part 3: Use Health Center to Investigate Application Errors (15 minutes)

Note:

WebSphere Application Server is running the Plants by WebSphere sample web application which has been modified with some deliberate programming errors.

_____ First verify the Plants by WebSphere sample is running. Double click the “First Steps” desktop short cut. Note this launches a DOS window and the GUI may take a few moments to appear.

_____ Click the “Samples Gallery” link in the “First Steps” window to launch a browser. Navigate to the “Plants by WebSphere” Sample and click the “Run” link.

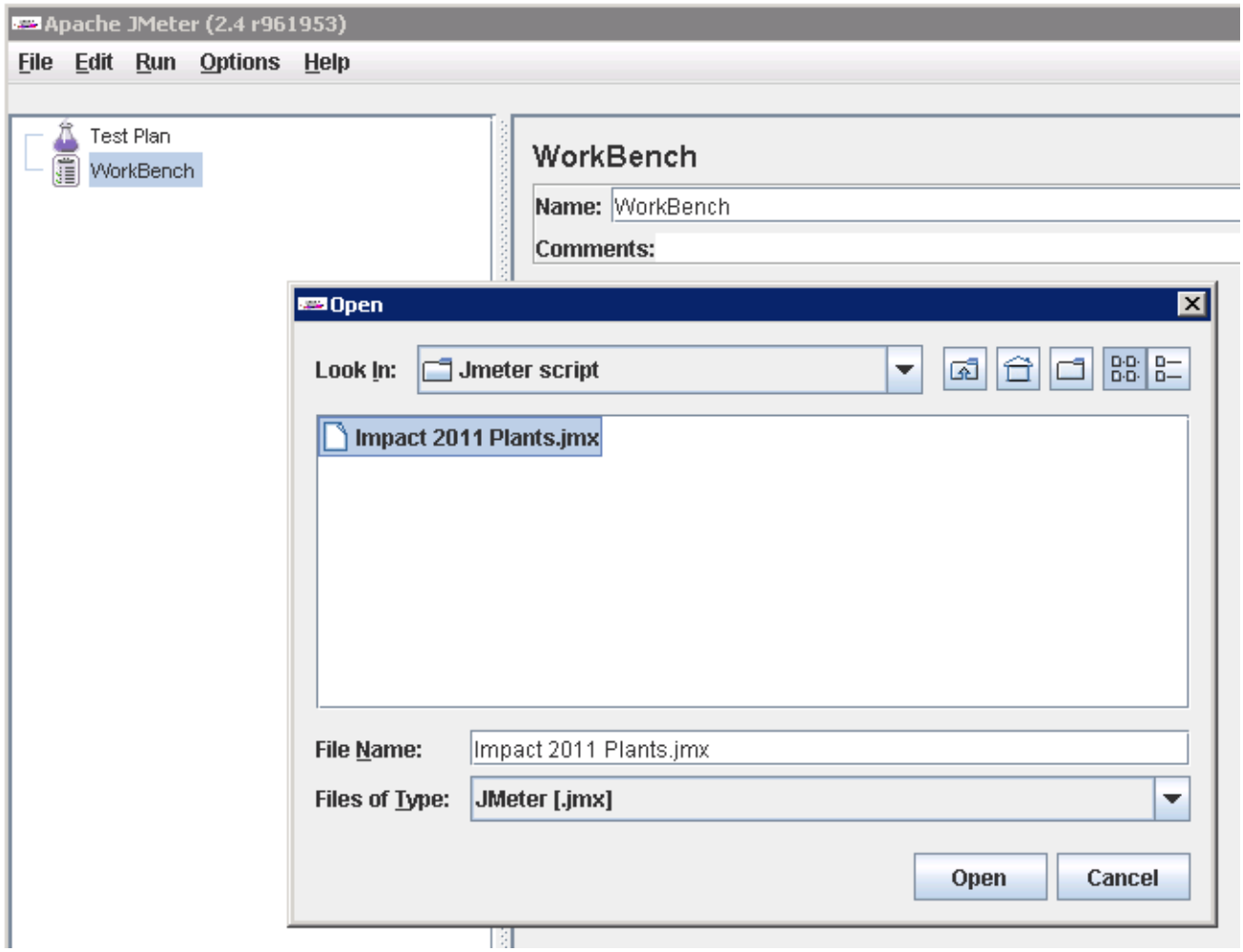


_____ The Plants sample will launch in a new browser window. Feel free to have a look around, but to avoid some deliberate mistakes **do not** click any products on the “Accessories” tab.

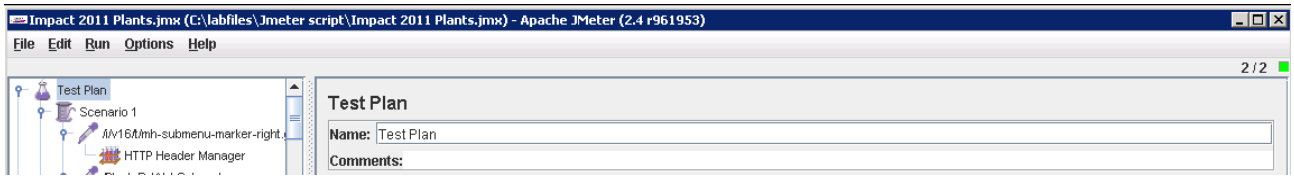
Next you will use the Jmeter load generating tool to simulate some user requests to the Plants web application. Some of these user requests will trigger deliberate errors which you will diagnose using Health Center.

_____ Double click the Jmeter shortcut on the desktop.

_____ Click “File->Open” and navigate to “**C:\labfiles\Jmeter script\Impact 2011 Plants.jmx**”. Click “Open”.

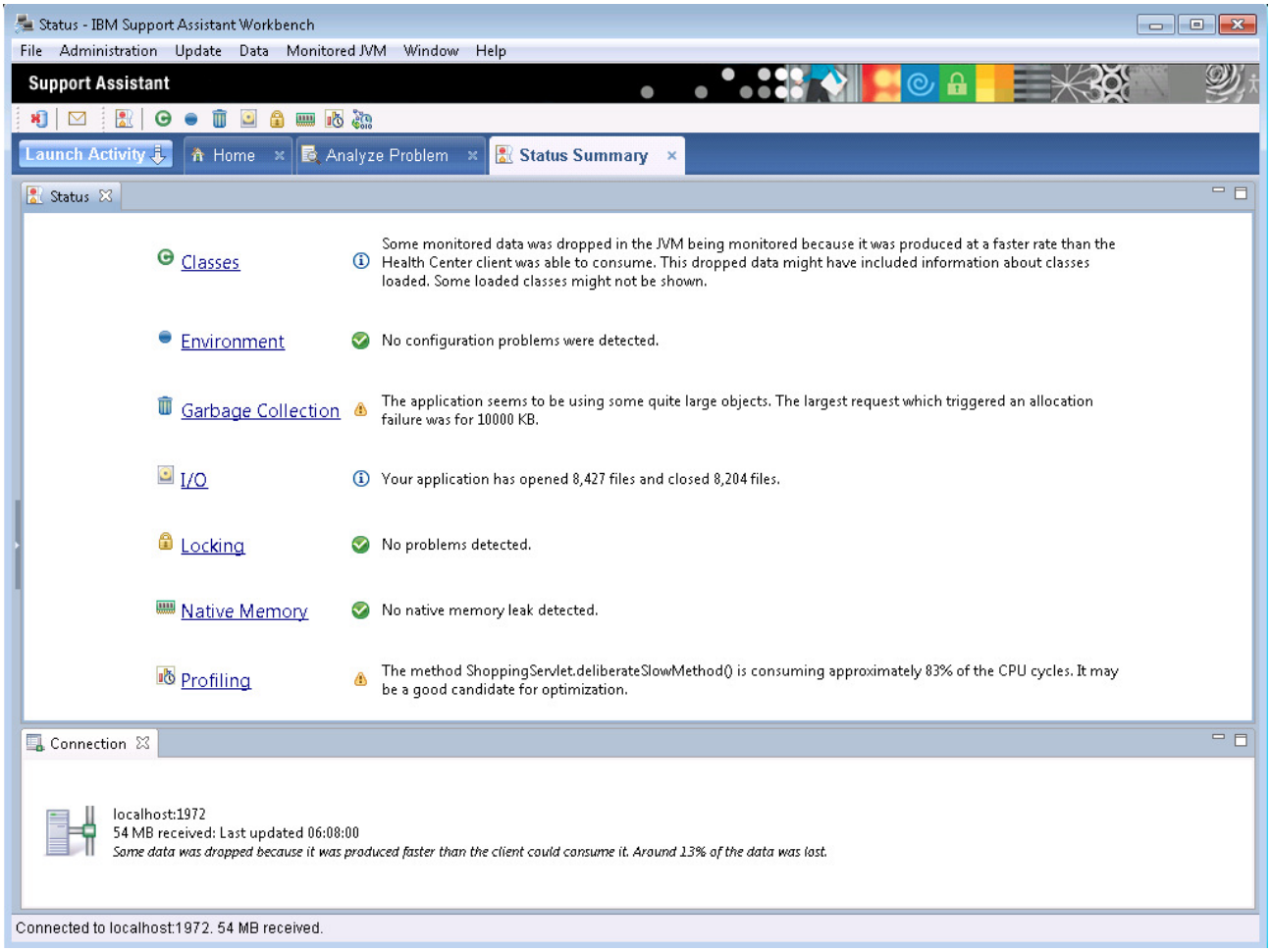


_____ Click "Run->Start". Wait a few moments until the number of threads has reached 2, as indicated in the right hand corner of the Jmeter window.



_____ Switch to the Health Center window which should already be monitoring the WebSphere JVM, if not make a new connection with File->New Connection.

Observe the Health Center status panel. This summarizes the main categories of data that Health Center is monitoring, and also summarizes current recommendations. Note that the data categories to be collected can be customized from the Monitored JVM menu, but for now leave this at the default setting.



Start by analyzing where the WebSphere JVM is spending most of its time and see if any optimizations can be made.

_____ Click the “Profiling” link.

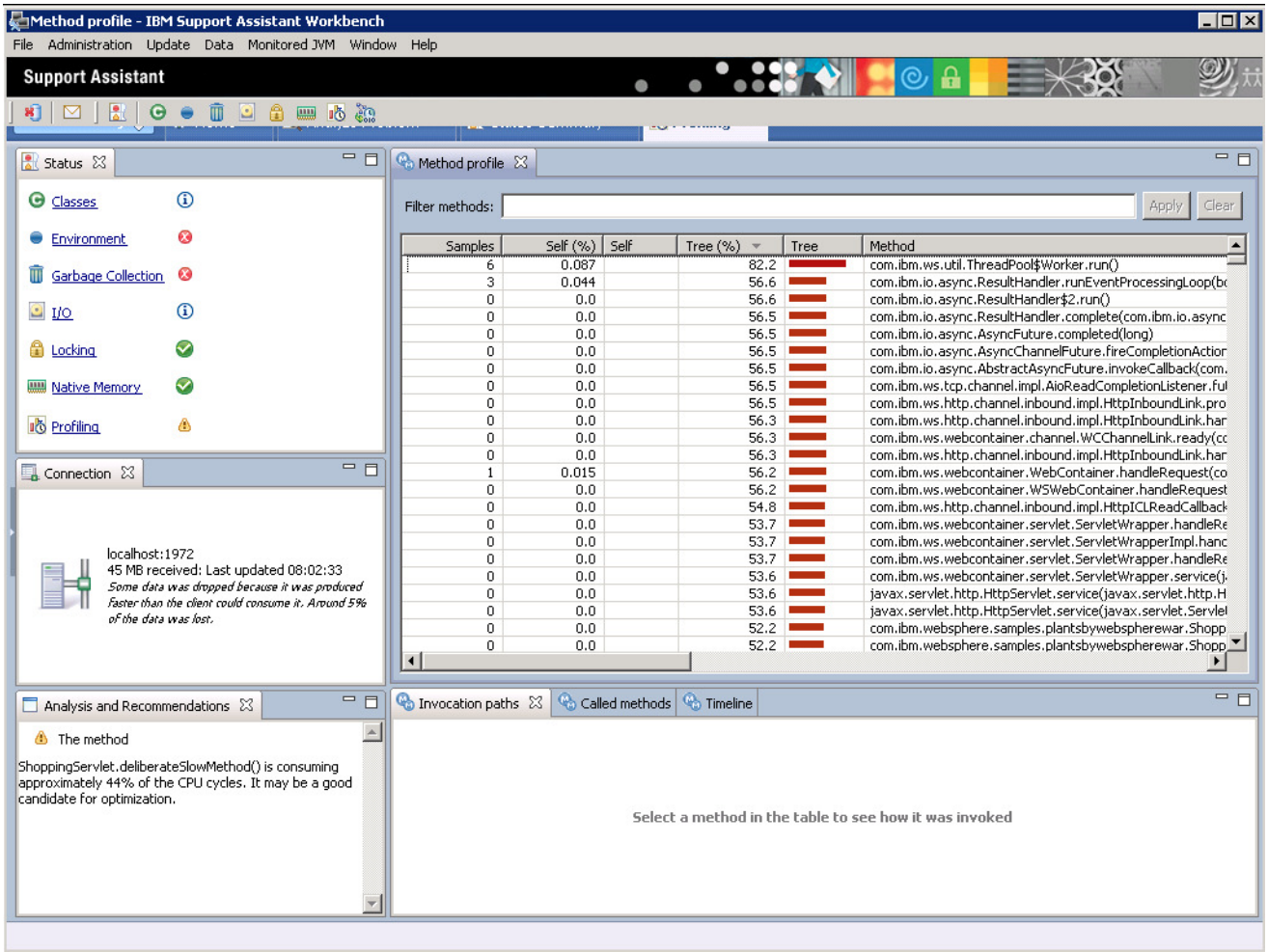
Health Center will show the results of its “sampling based” method profiler. This means it takes a periodic sample of the methods running and reports which are consuming the most time in the JVM.

_____ Sort the table of data by “Tree %” by clicking the “Tree %” column heading.

Within the Health Center, collections of methods are organized into structures called trees. You should see that in this case, a “**ThreadPool$Worker.run()**” method represents the **top of a tree** which is consuming a very high percentage of the JVM’s time.

However, also note the value in the “Self (%)” column, which indicates that the method “**ThreadPool$Worker.run()**” is actually using a **low percentage** of the JVM’s time. Therefore the problem must be in some code called by the “ThreadPool$Worker.run()” method, i.e. further down the tree / method call stack.

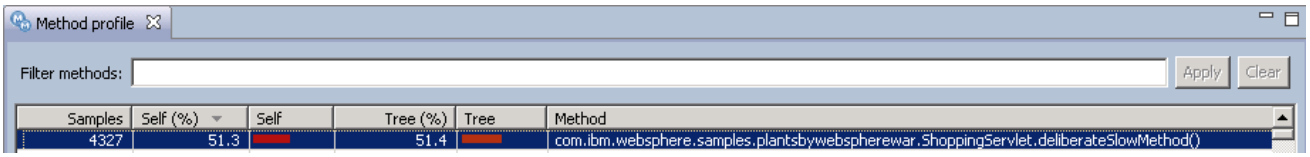
As incoming HTTP requests are handled by WebSphere using the “ThreadPool$Worker” class, this gives a clue that there could be something wrong in a running web application.



_____ Reorder the table to see results for individual methods by clicking “Self %”.

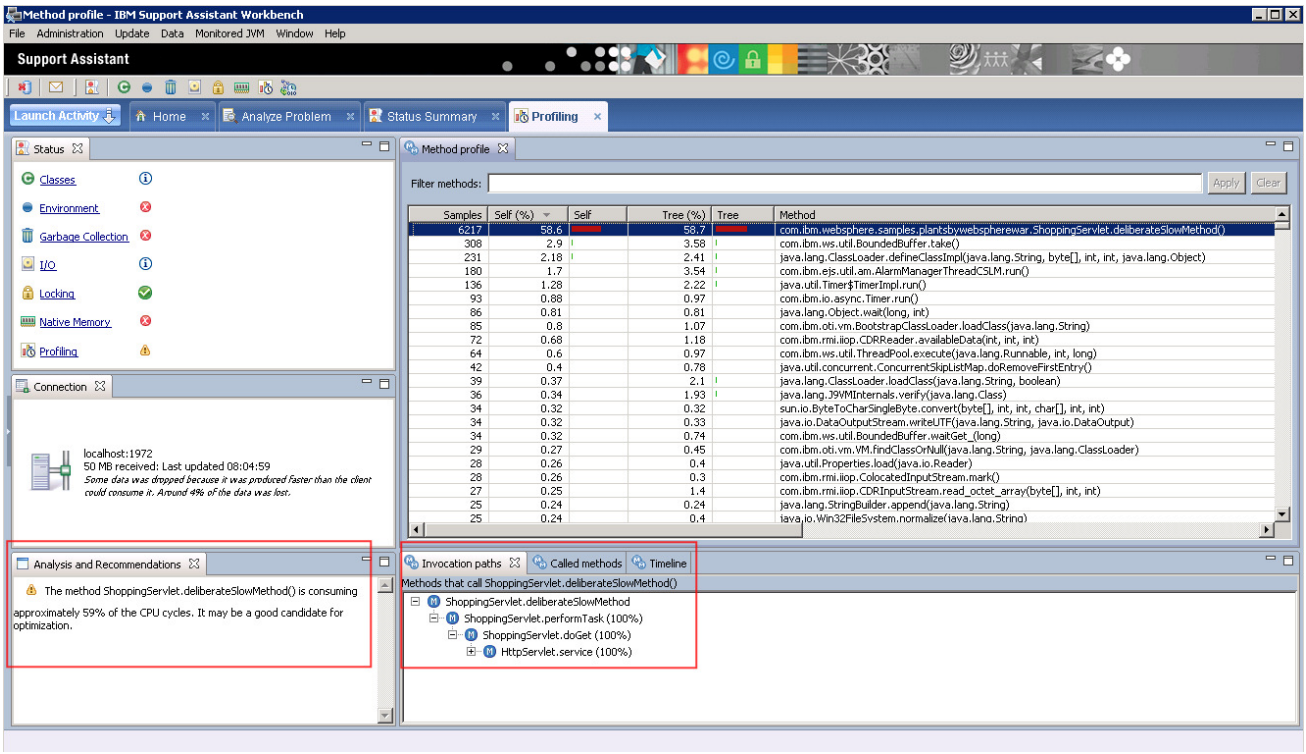
Now you can see the individual method “**deliberateSlowMethod**” in the **ShoppingServlet** class is using a high percentage of the JVM’s time. Note, the “Self” and “Tree” columns (without the % symbol) are a graphical indication that the method is very expensive, and is part of an expensive tree.

_____ Select the expensive method in the table by clicking it once.



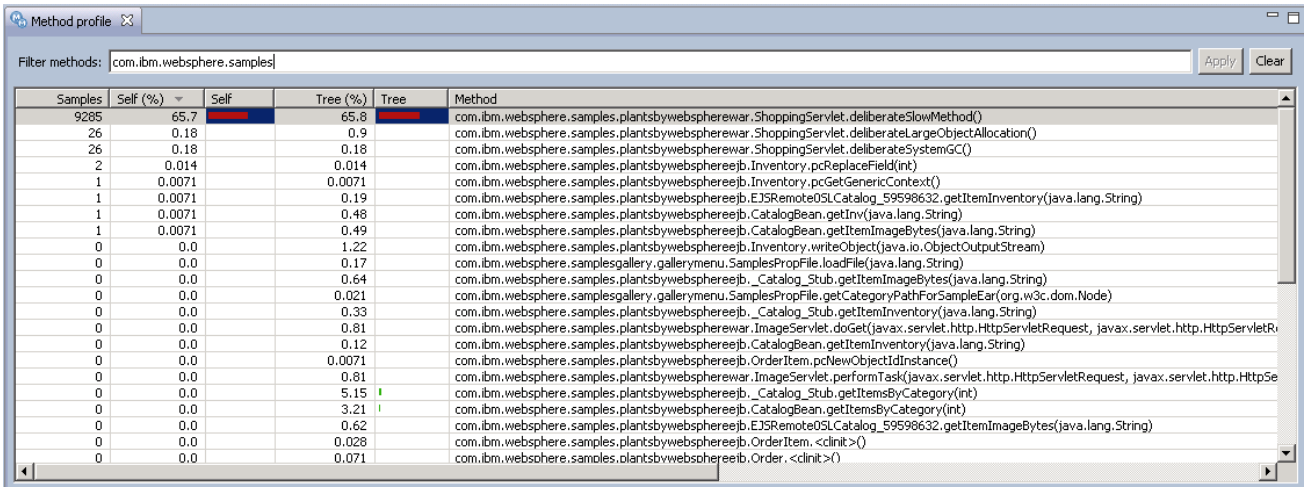
The "**Invocation paths**" tab shows what is calling the "deliberateSlowMethod". The "**timeline**" tab shows when the "deliberateSlowMethod" was invoked.

Also notice that Health Center has automatically identified the erroneous method and has highlighted this fact in the "Analysis and Recommendations" section.



_____ As the Plants sample is clearly suffering with at least one slow method, type "**com.ibm.websphere.samples**" in the "Filter Methods" box and click "Apply".

You can see only the "**ShoppingServlet.deliberateSlowMethod**" in the Plants sample has a high value for the "Self (%)".



Optional Steps:

_____ Double click the desktop shortcut to ShoppingServlet.java to inspect the programming error.

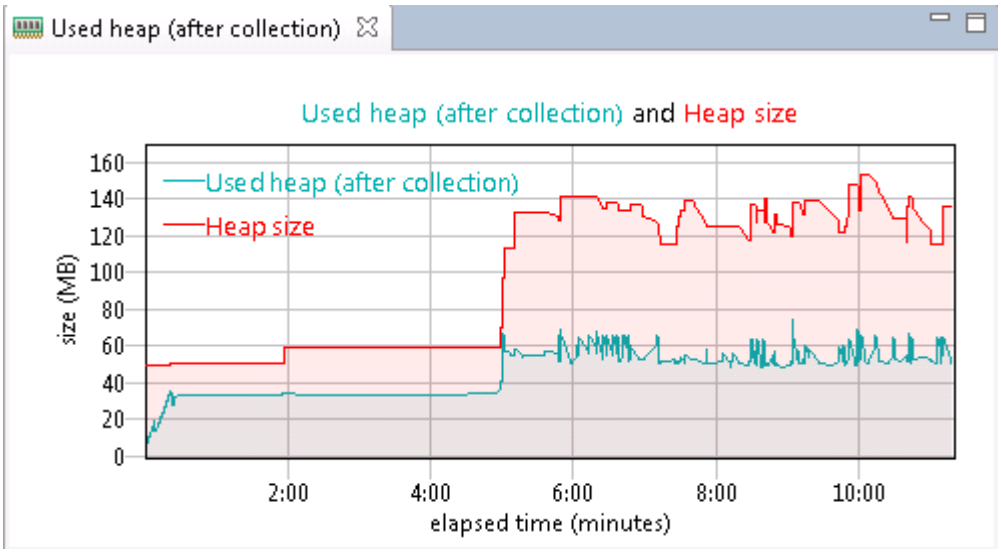
_____ Click “Edit->Find” and search for “deliberateSlowMethod”. Click “Find Next” to find the second occurrence of the search string.

The “deliberateSlowMethod” is invoked from the servlet’s “doGet” processing every time the user clicks on the tulips. The “deliberateSlowMethod” executes a tight loop which does not end until a 3.5 wait time has passed. You have found the first deliberate mistake.

```
System.out.println("==> STARTING SLOW METHOD");

long timestamp = System.currentTimeMillis();
long target = timestamp + 3500;

System.out.println("timestamp="+timestamp);
System.out.println("resume at="+target);
while(timestamp < target) {
        timestamp = System.currentTimeMillis();
}

System.out.println("==> ENDING SLOW METHOD");
```

_____ Return to the Health Center window and click the “Garbage Collection” link to monitor the performance of garbage collection and memory usage

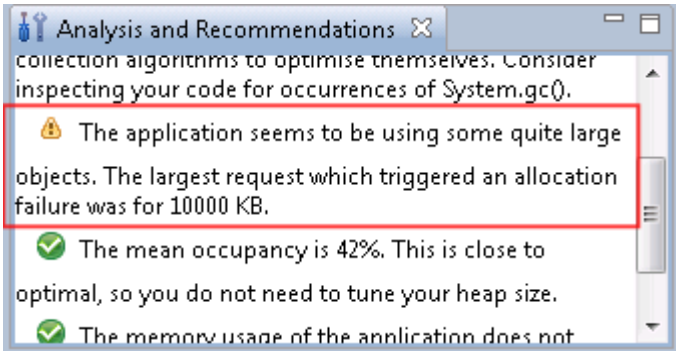
Observe the current JVM heap size and used heap size after collection. After starting the load generator, you will notice the heap size and heap usage has increased but by now should have leveled out. There is currently no evidence of a memory leak.



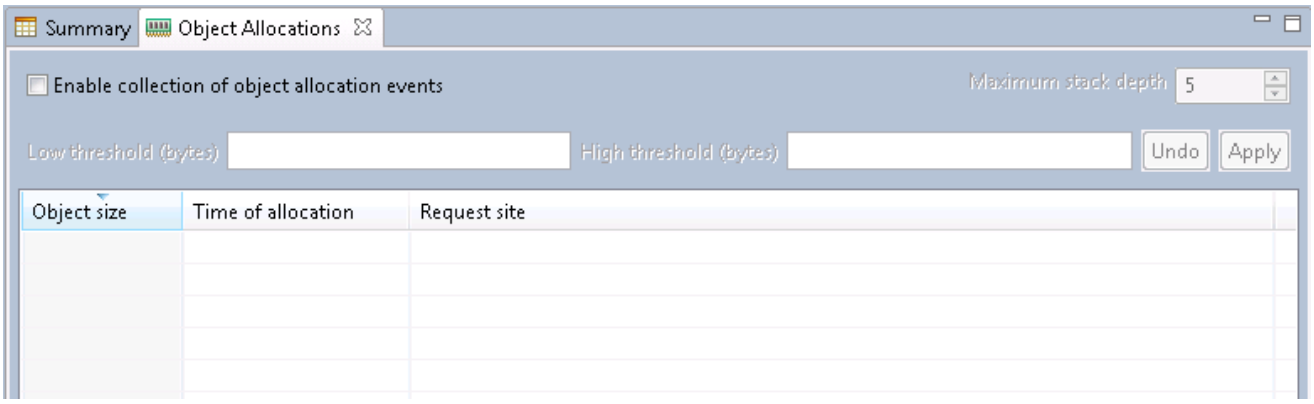
Note:

Garbage Collection (GC) affects the entire application and tuning GC correctly can potentially deliver significant performance gains. Health Center identifies where garbage collection is causing performance problems and suggests more appropriate command line options.

Observe the Analysis and Recommendations window. It warns of large object allocations which of course are likely to trigger frequent garbage collections and may indicate the application code can be optimized.



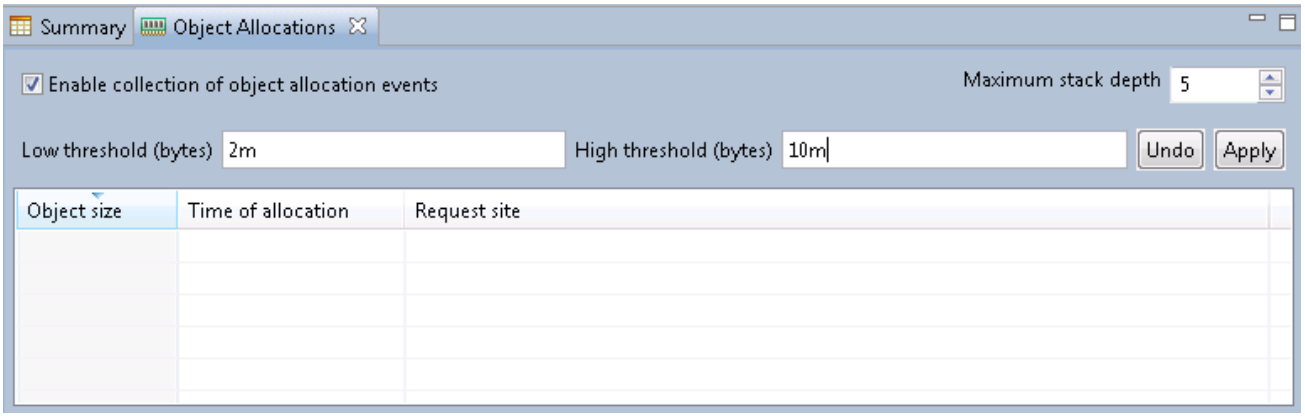
_____ Click the “Object Allocations” tab to investigate this further.



Health Center allows you to view the size, time and code location of an object allocation request that meets specific threshold criteria.

_____ Enable the “Enable collection of object allocation events” checkbox.

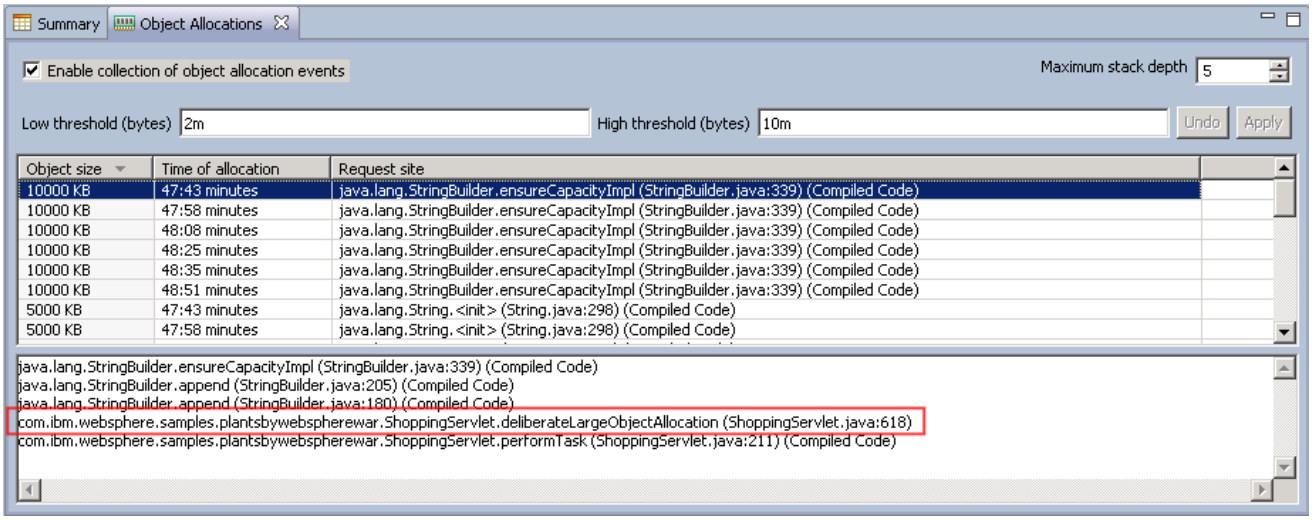
_____ Set a threshold to focus on the biggest objects. The threshold values can be entered in bytes, kilobytes or megabytes. Enter **2m** for the low threshold and **10m** for the high threshold and click “Apply”. Wait a few moments until large object allocation data is parsed by the Health Center client – **this could take up to 1 minute.**



Observe the large object allocations meeting the defined threshold. They are associated with creating a very large String.

_____ Click the rows in the table to see the stack trace leading to this large String allocation.

Once again you will see the ShoppingServlet class seems to be responsible, specifically a method named “deliberateLargeObjectAllocation”. You have identified another deliberate error in the plants sample.



Optional Steps:

_____ Double click the desktop shortcut for ShoppingServlet.java to inspect the programming error.

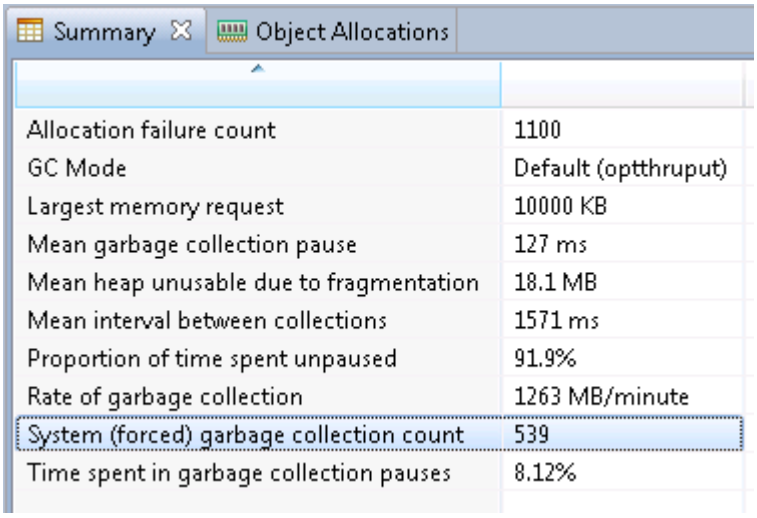
_____ Click “Edit->Find” and search for “deliberateLargeObjectAllocation”. Click “Find Next” to find the second occurrence of the search string.

The “deliberateLargeObjectAllocation” is invoked from the servlet’s “doGet” processing every time the user clicks on the grapes. The “deliberateLargeObjectAllocation” creates a large Array and fills it with a String of characters.

The variables used are local to the method so once the request has finished; the large objects are eligible for garbage collection. Therefore this is not a memory leak, but the creation of this large object makes unnecessary work for the JVM’s garbage collector.

```
System.out.println("==> STARTING LARGE OBJECT ALLOCATION");

// Handle to a large object.  Not a memory leak, just a LOA that will be GC'd
HashSet largeObject = null;

largeObject = new HashSet();
long timestamp = System.currentTimeMillis();
byte[] array = new byte[2560000];
Arrays.fill(array, (byte) 66);
largeObject.add(new String(array) + (timestamp));


System.out.println("==> ENDING LARGE OBJECT ALLOCATION");
```

_____ Return to the Heath Center window and click the "Garbage Collection" summary tab. This shows that System (forced) garbage collection is being called by some application code running in the JVM. There may also be a warning in the Analysis and Recommendations window, depending on how many times System (forced) garbage collection has been called.



Note:

The Java code "System.gc()" forces a full garbage collection cycle. This is generally not recommended as the garbage collector should manage its own schedule of garbage collection, and does not always need to execute the compaction phase of GC which is the most CPU intensive. An application calling System.gc() will always trigger the most expensive compaction phase. Health Center can be used to track down the source of the System.gc() events.

The easiest way to determine what is calling System.gc() is using **JVM trace** – a facility that is provided in all IBM supplied JVMs that has a minimal affect on performance. Some types of JVM trace can be conveniently configured via Health Center. However in this case we need to use "**Method Trace**" which must be configured as a JVM command line option. To save time, the following Java method trace has already been configured as a "Generic JVM Argument" via the WebSphere administration console.

-Xtrace:print=mt,methods={java/lang/System.gc},trigger=method{java/lang/System.gc,jstacktrace}

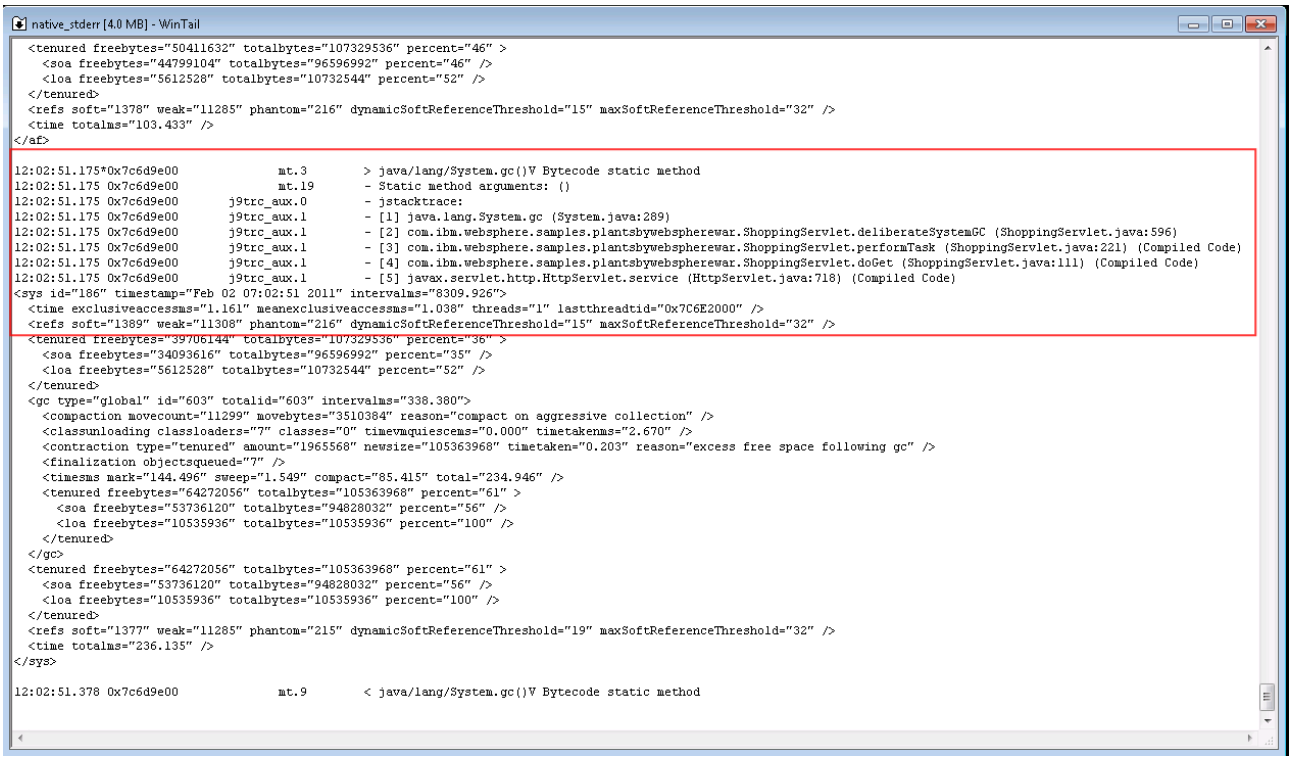
This prints a **stack trace** when the **System.gc()** method is executed. It would also be possible to trigger other diagnostic information such as a Java dump or Java core (this will be explained in more detail in part 4 of this lab).

_____ Double click the WinTail shortcut on the desktop.

_____ A “File Open” dialog box will appear. Navigate to:

“C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\native_stderr.log”

The file may be scrolling quite quickly, but every so often you will see a stack trace showing what is calling System.gc(). Alternatively use the scroll bar to find a previous stack trace.



Once again the source of the problem is ShoppingServlet which calls a method “**deliberateSystemGC**”. You have found another deliberate mistake in the plants sample.

Optional Steps:

_____ Double click the desktop shortcut for ShoppingServlet.java to inspect the programming error.

_____ Click “Edit->Find” and search for “deliberateSystemGC”. Click “Find Next” to find the second occurrence of the search string.

This “deliberateSystemGC” method is invoked from the servlet’s “doGet” processing every time the user clicks on the gloves. The “deliberateSystemGC” method calls System.gc().

```
System.out.println("==> STARTING SYSTEM.GC");

System.gc();

System.out.println("==> ENDING SYSTEM.GC");
```

You will manually trigger the final deliberate error in the plants sample as it will cause a memory leak that you will diagnose using heap dumps in the remaining parts of this lab.

Part 4: Trigger a Memory Leak and Generate Java Dumps (15 minutes)

Note:

To save time, WebSphere Application Server has already been configured to generate both an IBM heap dump and a system dump on an out of memory exception. The following dump option was configured as a "Generic JVM Argument" using the WebSphere administration console:

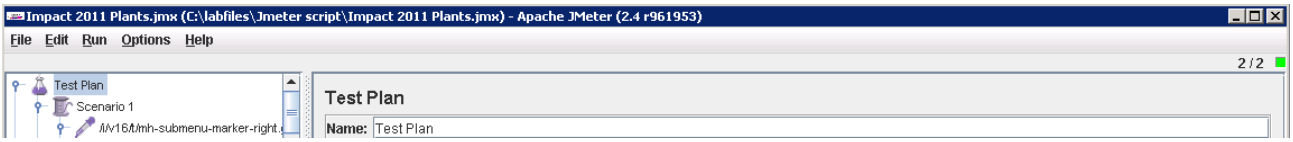
-Xdump:heap:none

-Xdump:java+heap+system:events=user+throw,filter=java/lang/OutOfMem*,range=1..1

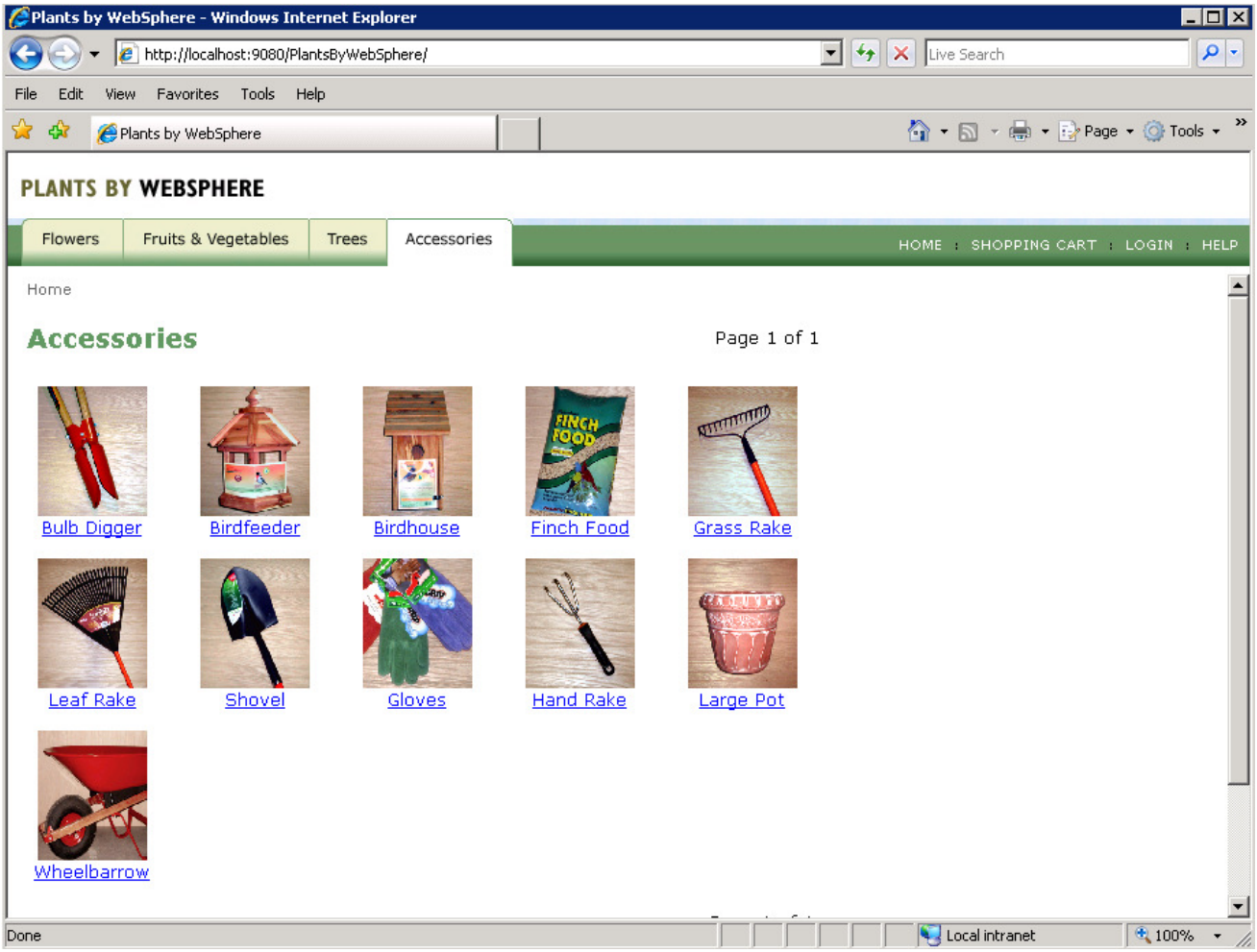
This overrides the default dump settings and specifies that exactly one heap dump (IBM PHD format) and one system core should be generated on an out of memory exception, or user signal to the process.

Java dumps (Javacore.txt files) are also configured. These are human readable text files containing summary information about the JVM, its memory and the running threads.

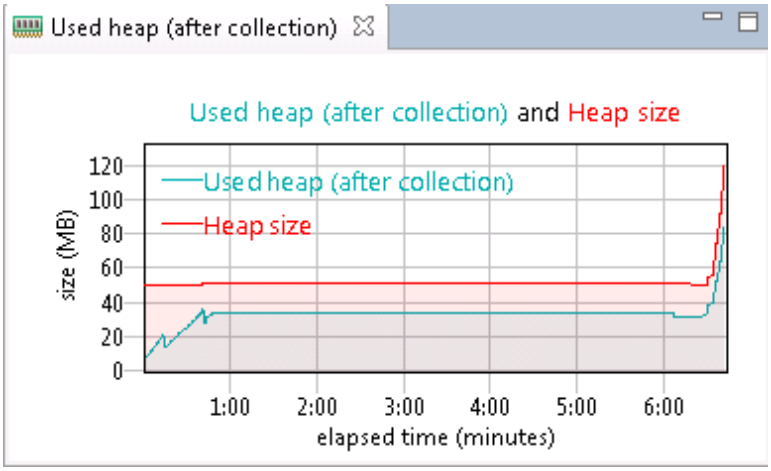
_____ Verify the Jmeter application is still running as some of the remaining parts of the lab require there to be active sessions when the heap dump is triggered (in the next few steps).



_____ Use the browser to click the Wheelbarrow product on the Accessories page of the Plants by WebSphere sample. Return to the Accessories page and click the Wheelbarrow again.



_____ Return to the Health Center window and ensure it has refreshed its data a couple of times (there is a 10 second pause between each refresh). Take a look at the Garbage Collection statistics, you should see notice the memory usage has increased. It seems there is a memory leak.



Note:

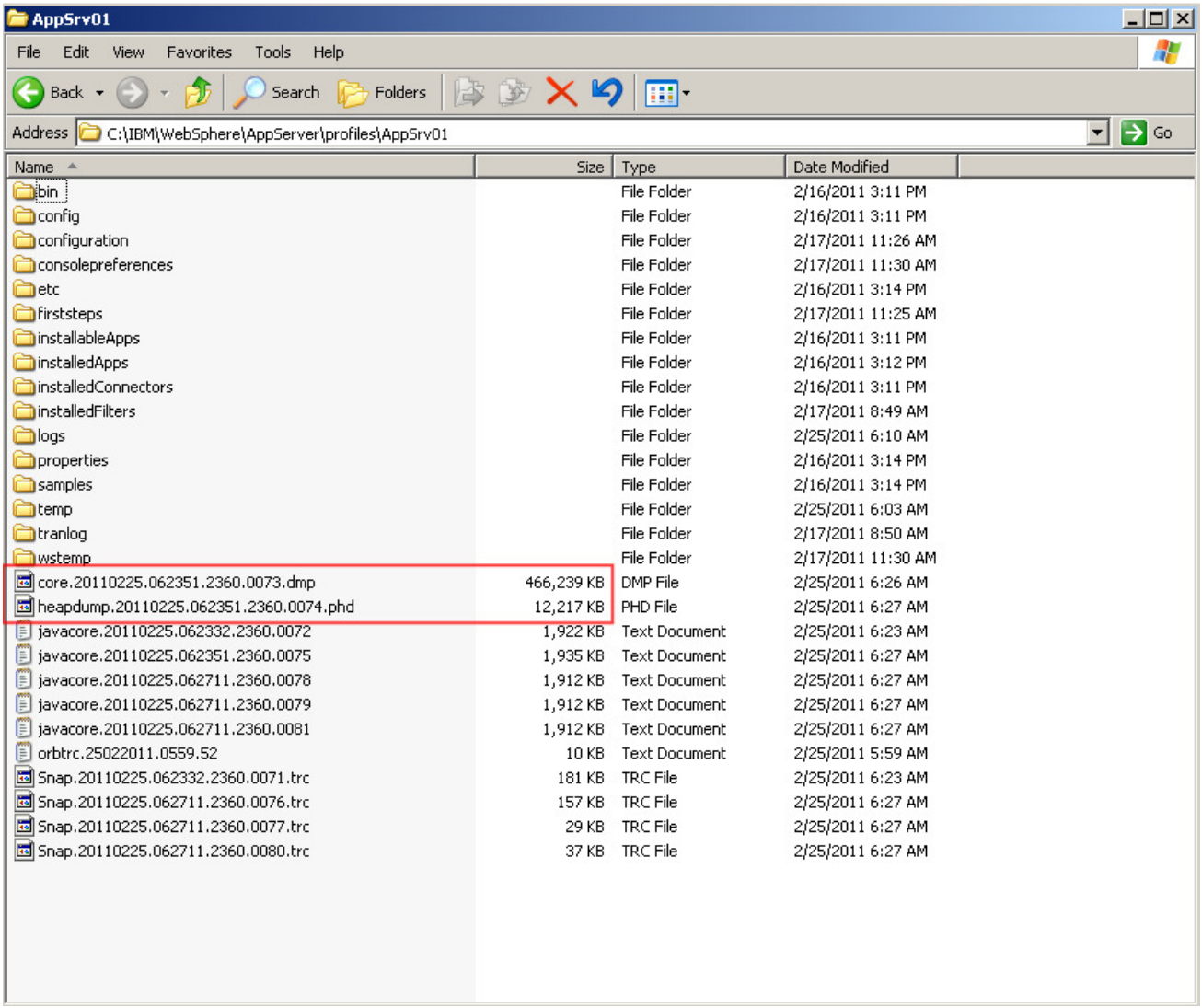
Health Center can be used to trigger a heap dump which is useful for analyzing memory leaks. However, to make the problem as easy to diagnose as possible, it is often beneficial to let the memory leak grow as large as possible.

_____ In the browser, continue clicking the Wheelbarrow until such time that the application no longer responds (**approximately 5-10 times**). This suggests the JVM has finally run out of memory and crashed. The JVM should begin to create some dumps.

_____ Verify that some heap dumps have been generated in the JVM's working directory, **"C:\IBM\WebSphere\AppServer\profiles\AppSrv01".**

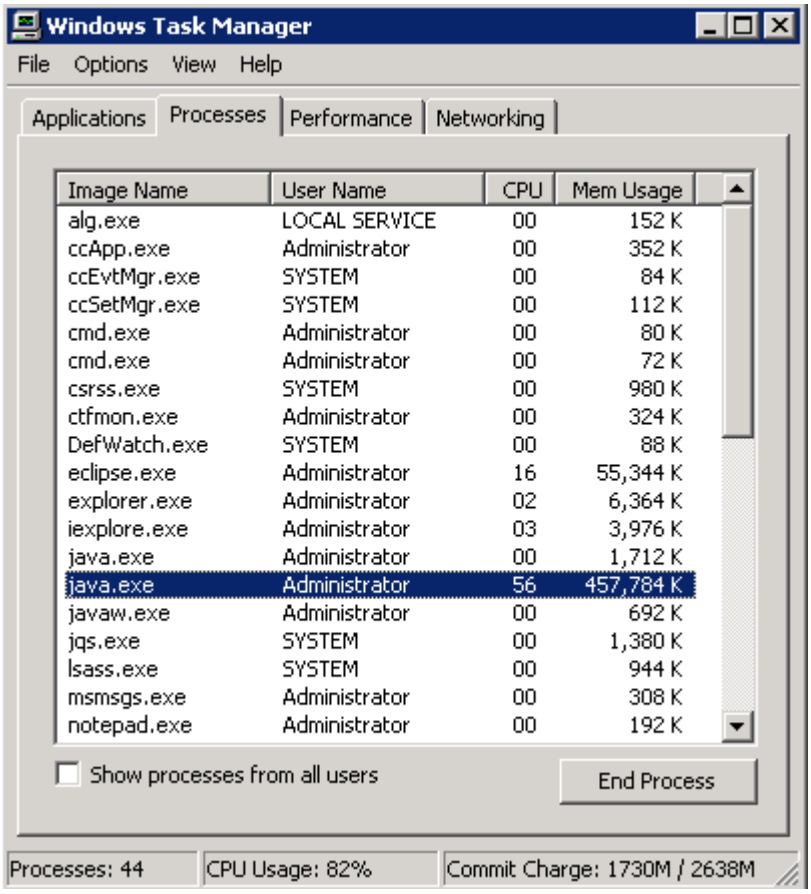
_____ Use F5 to refresh the directory until the size of the files has stopped growing. **This may take up to 5 minutes so please be patient.** The system core (.dmp extension) will grow to around **400Mb to 500Mb** in size. Do not proceed until both the system core (.dmp) and IBM heap dump (.phd) files have been completely written to disk.

You will notice the system core (.dmp extension) is much larger than the IBM heap dump (.phd extension).



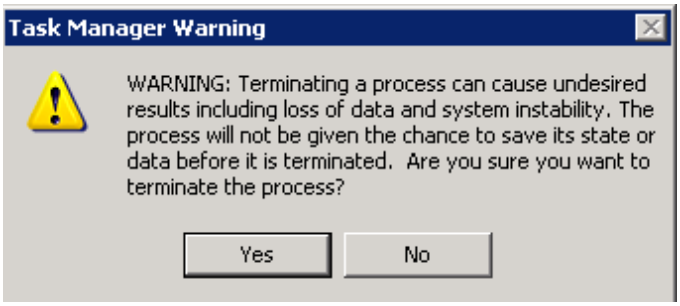
_____ Right click on the Windows taskbar and select “Task Manager”

_____ Select the “Process Tab”. Order the processes by name by clicking the “Image Name” column heading.



_____ Locate one or more Java processes using a large amount of memory (e.g. 200Mb or more). Click the "End Process" button for each one.

_____ Click the "Yes" button.



_____ Close the task manager window.

If for any reason the dump files are not present or complete, you can use some pre-prepared heap dumps in directory **"C:\Users\Administrator\Documents\Lab Files\Dumps".**

Note:

When a system dump (.dmp extension) is produced by the JVM, it is generated in a machine specific format and the internal structure of the information is specific to the VM that created it. To make the dump readable on other systems (e.g. heap dump analysis tools) it is necessary to run the **jextract** program on the dump.

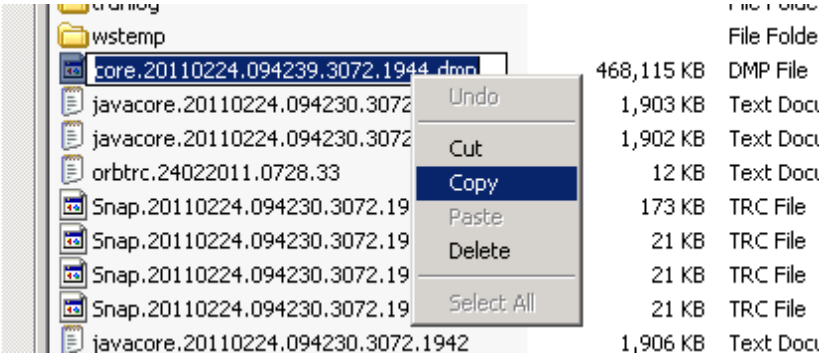
_____ Click "Start->Run" and type "cmd" in the search box, open a cmd window.

_____ In the cmd window, type **"cd C:\IBM\WebSphere\AppServer\java\jre\bin"**.

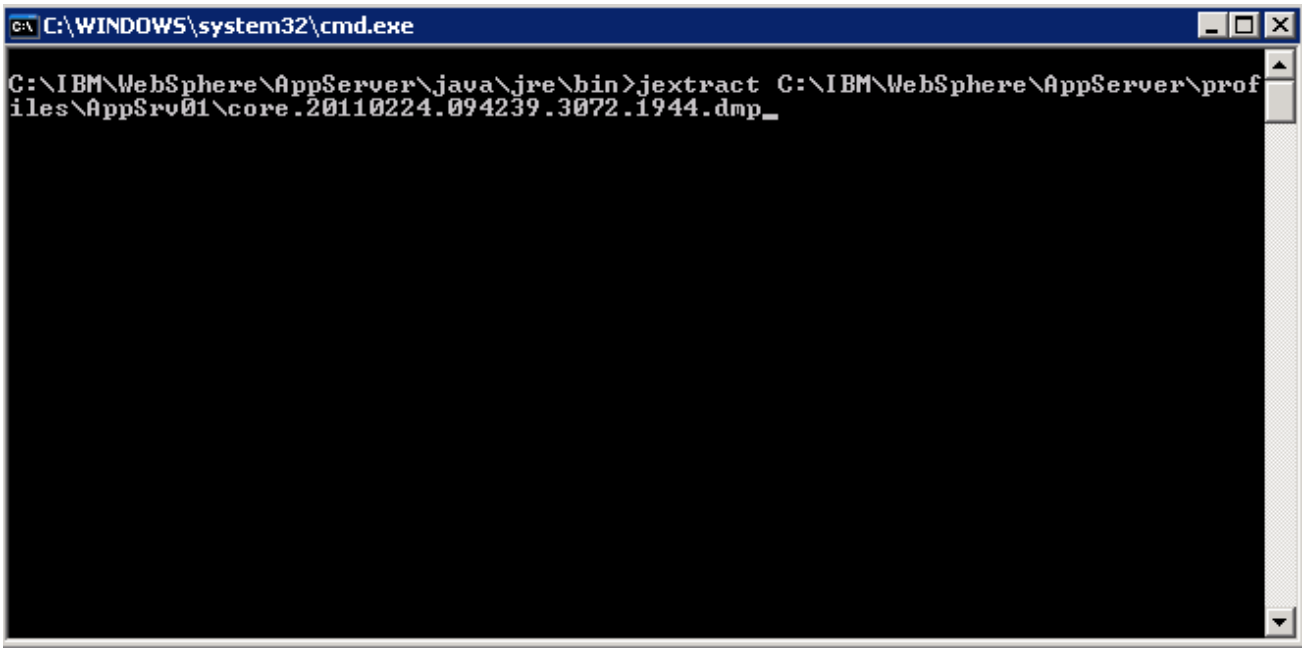
_____ Type command "**jextract C:\IBM\WebSphere\AppServer\profiles\AppSrv01**"

(do not press enter yet)

_____ Use windows explorer to copy the filename of the system dump (click "rename" to highlight the filename, then "copy").



_____ Right click and paste filename into the DOS command window. Press enter to execute the **jextract** command.



_____ The jextract process will take a few minutes to complete. However, you won't need this file for a while so continue with the next steps.

Note:

When the jextract process has completed, a new compressed version of the .dmp file will exist in the JVM working directory.

_____ Now that you have the heap dumps to investigate the memory leak, close the open windows (except the DOS window running extract).

Part 5: *Optional* - Analyzing a Java Core for Evidence of a Memory Issue (5 minutes)

Note:

As the JVM has created Java cores, a system dump and a heap dump file, the first assumption should be that an out of memory issue has occurred. Java cores can be produced by a number of triggers, but the creation of the combination of a Java core and heap dump is the default JVM action on an out of memory error. Therefore, reviewing the Java core for its trigger is the logical first step. **This part of the lab is optional.**

_____ Use a text editor to open one of the Java core files (named javacore*<datetimestamp>*.txt) from the JVM's working directory **"C:\IBM\WebSphere\AppServer\profiles\AppSrv01"**

_____ Determine the trigger that caused the Java core to be created. In the "**SIGINFO**" section you will see that a "**java/lang/OutOfMemoryError**" was indeed the signal that triggered the Java core. This confirms an out of memory condition and not a non-memory related crash.

javacore.20110224.094230.3072.1940 - Notepad

File Edit Format View Help

```
NULL            ------------------------------------------------------------------------
0SECTION        TITLE subcomponent dump routine
NULL            ================================
1TISIGINFO      Dump Event "systhrow" (00040000) Detail "java/lang/OutOfMemoryError" received
1TIDATETIME     Date:                2011/02/24 at 09:42:31
1TIFILENAME     Javacore filename:    C:\IBM\WebSphere\AppServer\profiles\AppSrv01\javacore.20110224.094230.3072.194
1TIREQFLAGS     Request Flags: 0x81 (exclusive+preempt)
1TIPREPSTATE    Prep State: 0x4 (exclusive_vm_access)
NULL            ------------------------------------------------------------------------
0SECTION        GPINFO subcomponent dump routine
NULL            ================================
2XHOSLEVEL      OS Level          : Windows XP 5.1 build 2600 Service Pack 3
2XHCPUS         Processors -
3XHCPUARCH        Architecture    : x86
3XHNUMCPUS        How Many        : 2
3XHNUMASUP        NUMA is either not supported or has been disabled by user
NULL
1XHERROR2       Register dump section only produced for SIGSEGV, SIGILL or SIGFPE.
NULL
NULL            ------------------------------------------------------------------------
0SECTION        ENVINFO subcomponent dump routine
NULL            ================================
1CIJAVAVERSION JRE 1.6.0 IBM J9 2.4 Windows XP x86-32 build jvmwi3260sr8ifx-20100923_65174
1CIVMVERSION    VM build 20100923_065174
1CIJITVERSION   JIT enabled, AOT enabled - r9_20100401_15339ifx6
1CIGCVERSION    GC - 20100308_AA
1CIRUNNINGAS    Running as a standalone JVM
1CICMDLINE      C:\IBM\WebSphere\AppServer/java/bin/java -Declipse.security -Dwas.status.socket=1087 -Dosgi.install.
lassAccessMode=allow -verbose:gc -Xms50m -Xmx256m -Dws.ext.dirs=C:\IBM\WebSphere\AppServer/java/lib;C:\IBM\WebSpher
ebug=off -Xtrace:print=mt,methods={java/lang/System.gc},trigger=method{java/lang/System.gc,jstacktrace} -Xdump:heap
1CIJAVAHOMEDIR Java Home Dir:    C:\IBM\WebSphere\AppServer\java\jre
1CIJAVADLLDIR  Java DLL Dir:     C:\IBM\WebSphere\AppServer\java\jre\bin
1CISYSCP        Sys Classpath:    C:\IBM\WebSphere\AppServer/java/jre/lib/ext/ibmorb.jar;C:\IBM\WebSphere\AppServer/
C:\IBM\WebSphere\AppServer\java\jre\lib\ibmjssefw.jar;C:\IBM\WebSphere\AppServer\java\jre\lib\ibmsaslfw.jar;C:\IBM
1CIUSERARGS     UserArgs:
2CIUSERARG                -Xjcl:jclscar_24
```

_____ The Java core also contains important Java heap configuration and status information. Observe the maximum heap size (-Xmx256m) in the "**CIUSERARG**" section.

```
javacore.20110224.094230.3072.1940 - Notepad
File  Edit  Format  View  Help
C:\IBM\WebSphere\AppServer\java\jre\lib\ibmjssefw.jar;C:\IBM\WebSphere\AppServer\java\jre\lib\ibmsaslfw.jar;C:\IBM\
1CIUSERARGS    UserArgs:
2CIUSERARG                -Xjcl:jclscar_24
2CIUSERARG                -Dcom.ibm.oti.vm.bootstrap.library.path=C:\IBM\WebSphere\AppServer\java\jre\bin
2CIUSERARG                -Dsun.boot.library.path=C:\IBM\WebSphere\AppServer\java\jre\bin
2CIUSERARG                -Djava.library.path=C:\IBM\WebSphere\AppServer\java\jre\bin;.;C:\IBM\WebSphere\AppServer\b
2CIUSERARG                -Djava.home=C:\IBM\WebSphere\AppServer\java\jre
2CIUSERARG                -Djava.ext.dirs=C:\IBM\WebSphere\AppServer\java\jre\lib\ext
2CIUSERARG                -Duser.dir=C:\IBM\WebSphere\AppServer\profiles\AppSrv01
2CIUSERARG                _j2se_j9=71168 0x7FBE7290
2CIUSERARG                -Xdump
2CIUSERARG                -Dconsole.encoding=Cp850
2CIUSERARG                -Djava.class.path=C:\IBM\WebSphere\AppServer\profiles\AppSrv01\properties;C:\IBM\WebSphere
2CIUSERARG                -Declipse.security
2CIUSERARG                -Dwas.status.socket=1087
2CIUSERARG                -Dosgi.install.area=C:\IBM\WebSphere\AppServer
2CIUSERARG                -Dosgi.configuration.area=C:\IBM\WebSphere\AppServer\profiles\AppSrv01/configuration
2CIUSERARG                -Dosgi.framework.extensions=com.ibm.cds,com.ibm.ws.eclipse.adaptors
2CIUSERARG                -Xshareclasses:name=webspherev70,nonFatal
2CIUSERARG                -Xscmx50M
2CIUSERARG                -Dsun.reflect.inflationThreshold=250
2CIUSERARG                -Xbootclasspath/p:C:\IBM\WebSphere\AppServer/java/jre/lib/ext/ibmorb.jar;C:\IBM\WebSphere\
2CIUSERARG                -Djava.class.path=C:\IBM\WebSphere\AppServer\profiles\AppSrv01/properties;C:\IBM\WebSphere
2CIUSERARG                -Dibm.websphere.internalClassAccessMode=allow
2CIUSERARG                -verbose:gc
2CIUSERARG                -Xms50m
2CIUSERARG                -Xmx256m
2CIUSERARG                -Dws.ext.dirs=C:\IBM\WebSphere\AppServer/java/lib;C:\IBM\WebSphere\AppServer\profiles\Apps
2CIUSERARG                -Dderby.system.home=C:\IBM\WebSphere\AppServer/derby
2CIUSERARG                -Dcom.ibm.itp.location=C:\IBM\WebSphere\AppServer/bin
2CIUSERARG                -Djava.util.logging.configureByServer=true
2CIUSERARG                -Duser.install.root=C:\IBM\WebSphere\AppServer\profiles\AppSrv01
2CIUSERARG                -Djavax.management.builder.initial=com.ibm.ws.management.PlatformMBeanServerBuilder
2CIUSERARG                -Dwas.install.root=C:\IBM\WebSphere\AppServer
2CIUSERARG                -Dpython.cachedir=C:\IBM\WebSphere\AppServer\profiles\AppSrv01/temp/cachedir
2CIUSERARG                -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
```

_____ Determine the free heap space and the total heap allocation from the "**MEMINFO**" section. In this case, there are 0 bytes free in the available heap space. Note that the values are in Hex, and 10000000 is 256 in decimal (the configured maximum heap size). If you wish to calculate the free heap for your Java core file, you can use the Windows calculator in scientific mode to convert from Hex to decimal.

```
javacore.20110224.094230.3072.1940 - Notepad
File  Edit  Format  View  Help
2CIENVVAR       USERPROFILE=C:\Documents and Settings\Administrator
2CIENVVAR       WAS_CELL=IMPACT2011Node01Cell
2CIENVVAR       WAS_CLASSPATH=C:\IBM\WebSphere\AppServer\profiles\AppSrv01\properties;C:\IBM\WebSphere\AppServer\pro
2CIENVVAR       WAS_EXT_DIRS=C:\IBM\WebSphere\AppServer\java\lib;C:\IBM\WebSphere\AppServer\classes;C:\IBM\WebSphere
2CIENVVAR       WAS_HOME=C:\IBM\WebSphere\AppServer
2CIENVVAR       WAS_LOGGING=-Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager -Djava.util.logging.config
2CIENVVAR       WAS_NODE=IMPACT2011Node01
2CIENVVAR       WAS_PATH=C:\IBM\WebSphere\AppServer\bin;C:\IBM\WebSphere\AppServer\java\bin;C:\IBM\WebSphere\AppServ
2CIENVVAR       WAS_USER_SCRIPT=C:\IBM\WebSphere\AppServer\profiles\AppSrv01\bin\setupCmdLine.bat
2CIENVVAR       WAS_USER_SCRIPT_FILE_NOT_EXISTS=false
2CIENVVAR       windir=C:\WINDOWS
NULL
1CIJVMMI        JVM Monitoring Interface (JVMMI)
NULL            ------------------------
2CIJVMMIOFF     [not available]
NULL
NULL            ------------------------------------------------------------------------
0SECTION        MEMINFO subcomponent dump routine
NULL            ================================
1STHEAPFREE     Bytes of Heap Space Free: 0
1STHEAPALLOC    Bytes of Heap Space Allocated: 10000000
NULL
1STSEGTYPE      Internal Memory
NULL            segment  start    alloc    end      type     bytes
1STSEGMENT      17BCD2D4 1B744FF8 1B744FF8 1B754FF8 01000040 10000
1STSEGMENT      185A939C 1B7B8068 1B7B8068 1B7C8068 01000040 10000
1STSEGMENT      17DCF2E4 1B7E8018 1B7E8018 1B7F8018 01000040 10000
1STSEGMENT      17DCF1C4 1B677018 1B677018 1B687018 01000040 10000
1STSEGMENT      1468DA64 1B7A8040 1B7A8040 1B7B8040 01000040 10000
1STSEGMENT      17DCF404 1B6BE018 1B6BE018 1B6CE018 01000040 10000
1STSEGMENT      1835A834 1B785028 1B785028 1B795028 01000040 10000
1STSEGMENT      18F0D274 1BAC93A0 1BAC93A0 1BAD93A0 01000040 10000
1STSEGMENT      1835A7D4 1B808038 1B808038 1B818038 01000040 10000
1STSEGMENT      1848C874 1B666FA8 1B666FA8 1B676FA8 01000040 10000
1STSEGMENT      1468DBE4 1B655728 1B655728 1B665728 01000040 10000
1STSEGMENT      181AD394 1B8EB7B0 1B8EB7B0 1B8FB7B0 01000040 10000
```

Note:

At this point, it appears that the JVM is experiencing a severe shortage of heap space. If you needed more detailed information you could study the verbose GC to help identify if the cause of the out of memory condition is a memory footprint problem, or a memory leak.

A footprint problem would manifest itself as an increase in used heap space associated with an increase in workload for the JVM. A memory leak would manifest itself as an increase in used heap space either gradually over time, or rapidly at different times when associated with a particular application event.

IBM provides to tool to assist in this analysis - the **IBM Monitoring and Diagnostic Tools for Java™ - Garbage Collection and Memory Visualizer (GCMV)** which is an add-on to the IBM Support Assistant (ISA). **Appendix A** contains an optional lab part where you can examine in more detail the garbage collection activity that led to the out of memory condition, using GCMV. It is suggested to complete Appendix A if you have time after the remaining parts of this lab.

In the next part of the lab, you will analyze the Java heap dump files and diagnose the code that caused the memory leak.

Part 6: Using the ISA and the Memory Analyzer to Analyze a Heapdump (20 minutes)

Note:

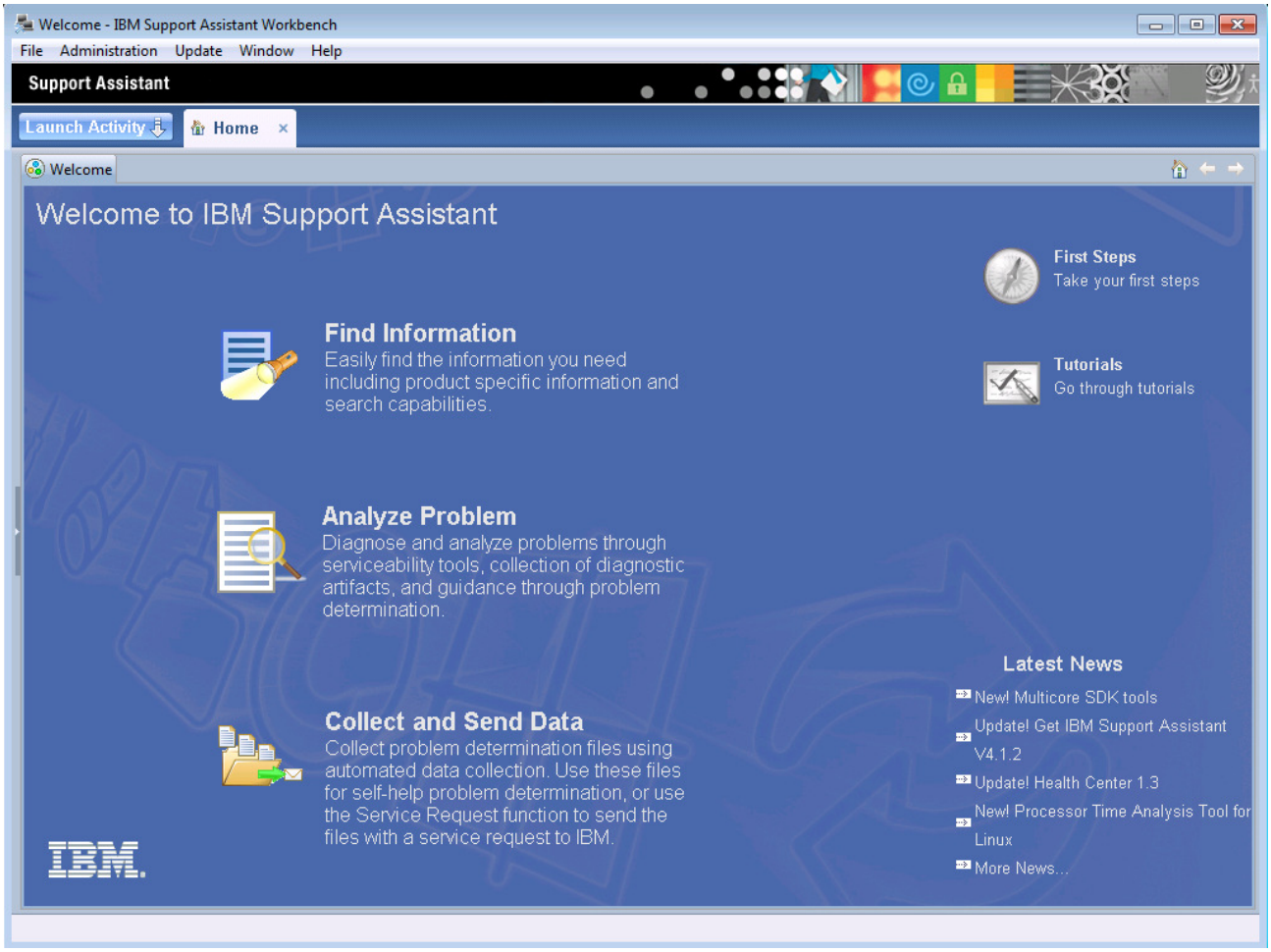
Memory Analyzer is a powerful and flexible tool for analyzing Java heap memory using system or heap dumps of the Java process. The maximum heap size for the tool has been increased in this lab to ensure the tool can handle larger heap dumps.

The next two parts of lab will direct you to open the IBM heap dump and system dump you generated previously. Opening these heap dumps for the first time can take up to **5 minutes**. If you prefer, use the ready made heap dump in **C:\labfiles\Dumps** – these files have previously been opened by Memory Analyzer which creates "index files". Using these files will slightly reduce the amount of time required to complete the lab – it's up to you.

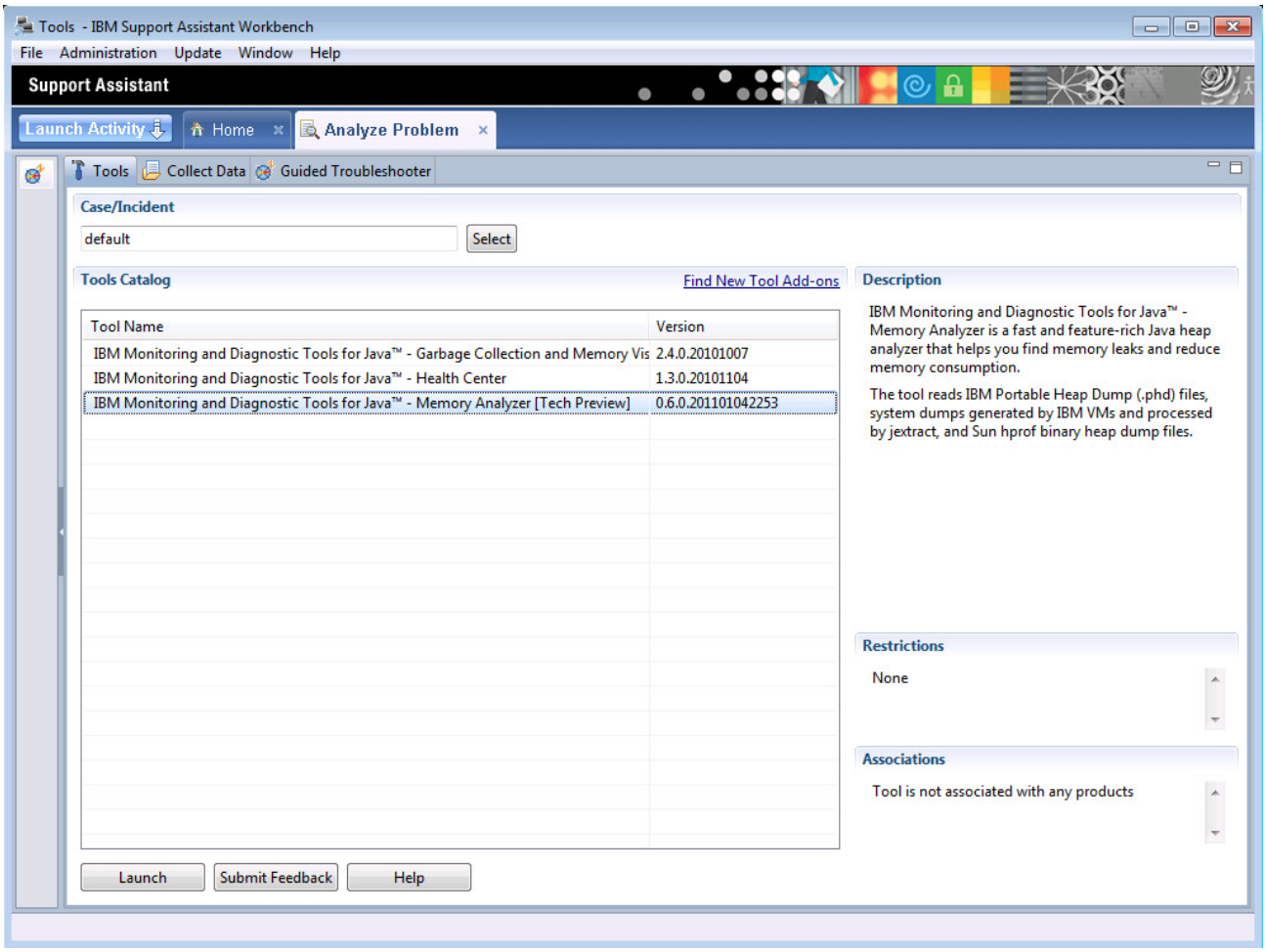
As every heap dump is different, you may see some slight variation from the screenshots in this lab document, e.g. exact number of bytes for the object size or number of objects in a data structure etc.

_____ Double click the "IBM Support Assistant 4.1" shortcut on the desktop.

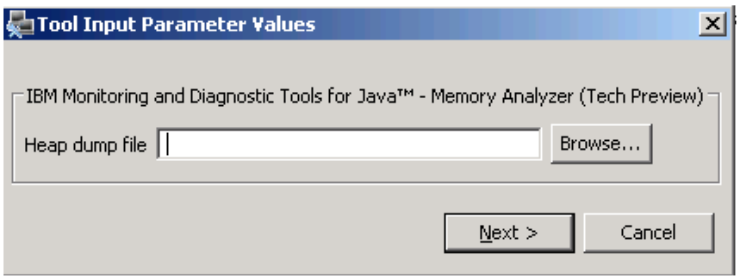
_____ Click the blue "Launch Activity" button and select "Analyze Problem".


Welcome - IBM Support Assistant Workbench
File Administration Update Window Help
Support Assistant
Launch Activity
Home
Welcome
Welcome to IBM Support Assistant
First Steps
Take your first steps
Find Information
Easily find the information you need including product specific information and search capabilities.
Tutorials
Go through tutorials
Analyze Problem
Diagnose and analyze problems through serviceability tools, collection of diagnostic artifacts, and guidance through problem determination.
Latest News
New! Multicore SDK tools
Update! Get IBM Support Assistant V4.1.2
Update! Health Center 1.3
New! Processor Time Analysis Tool for Linux
More News...
Collect and Send Data
Collect problem determination files using automated data collection. Use these files for self-help problem determination, or use the Service Request function to send the files with a service request to IBM.
IBM.

_____ Select “IBM Monitoring and Diagnostic Tools for Java™ - Memory Analyzer”, and then click the grey “Launch” button at the bottom of the screen.

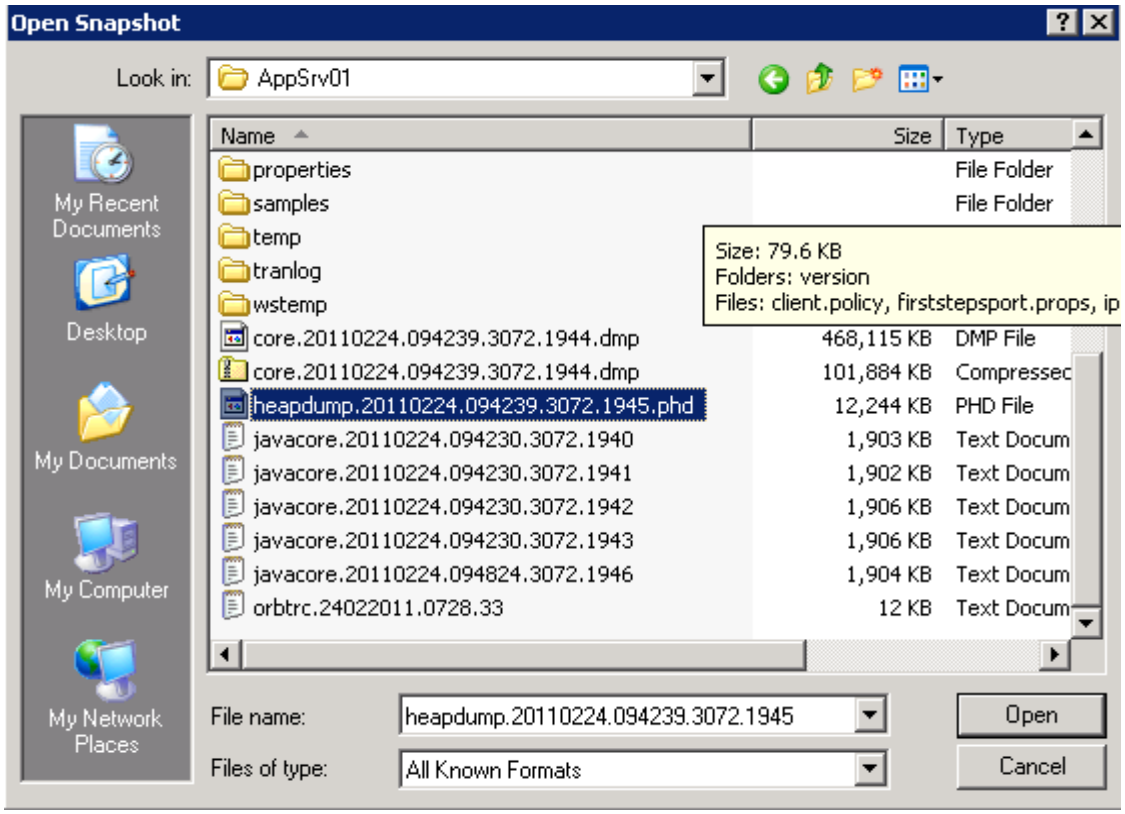


_____ On the “Tool Input Parameters Values” box, click “Next”. This will launch the tool immediately.

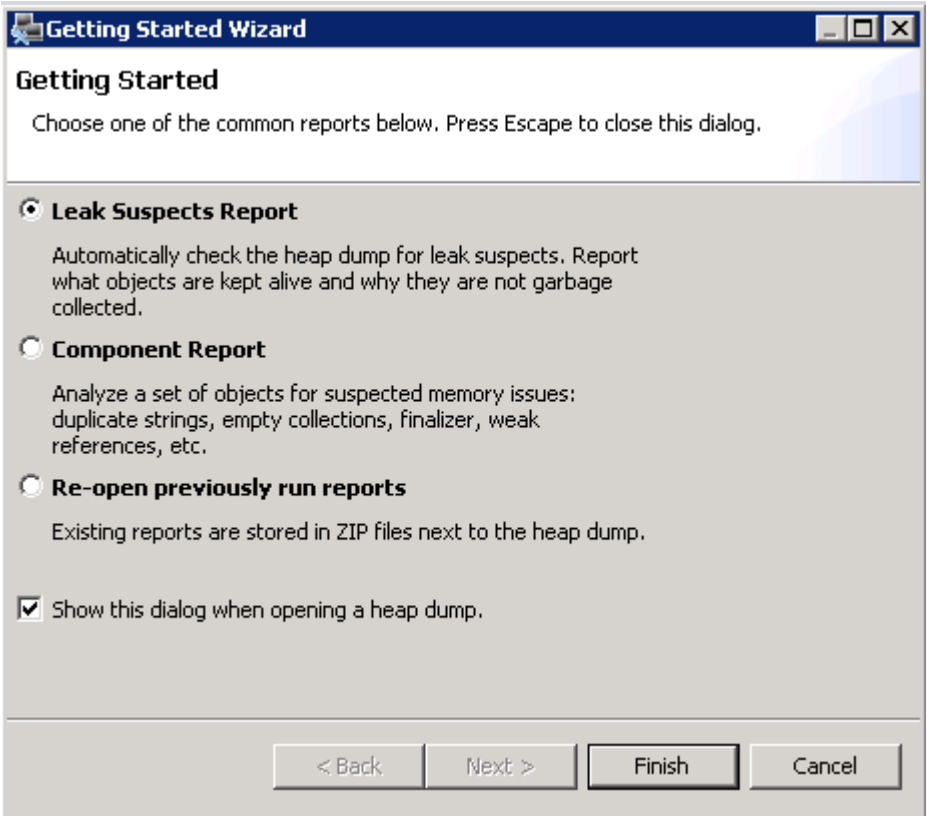


_____ Click "File->Open heap dump" and **navigate** to the JVM's working directory **"C:\IBM\WebSphere\AppServer\profiles\AppSrv01"** (note the default directory is for the 'ready made' heap dumps, not the JVM's working directory).

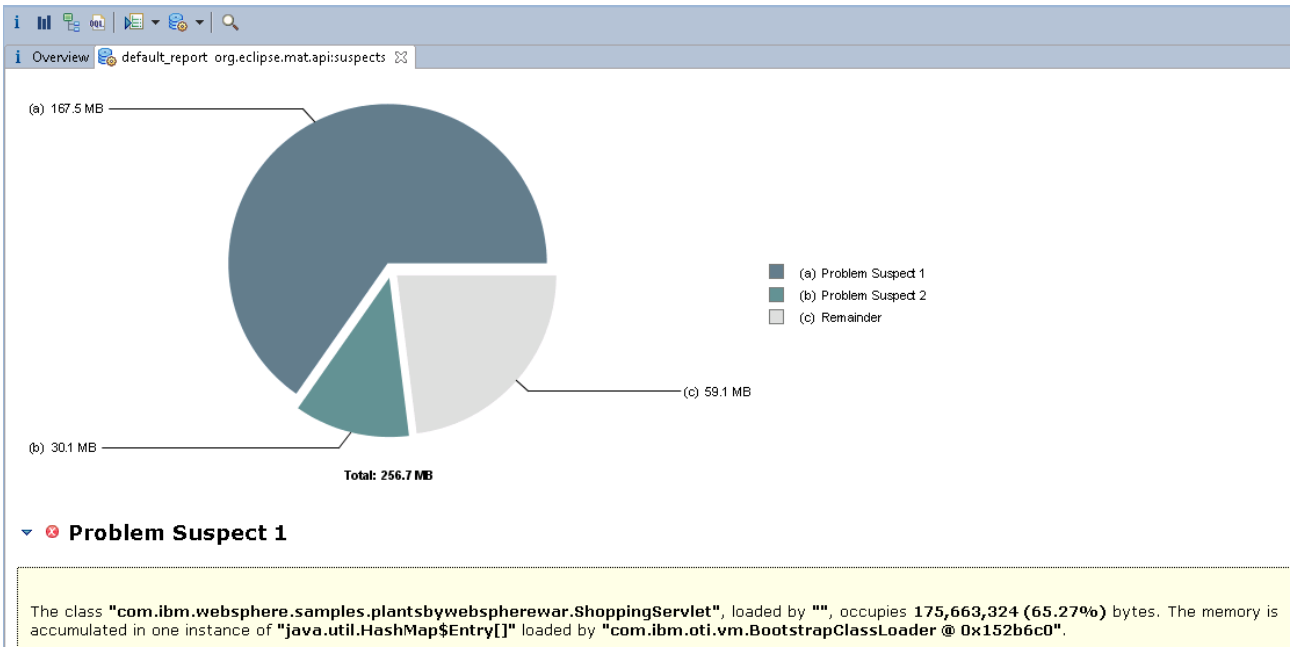
_____ Select the IBM heap dump (extension .phd) and click the "Open" button. **Please wait a few minutes for the heap dump to be processed.**



_____ In the "Getting Started Wizard" box, click the "Finish" button to open the "**Leak Suspects Report**" and wait a few moments more.



This report provides basic heap statistics as well as a list of possible leaking objects. Your heap dump will indicate that ShoppingServlet is responsible for a large percentage of memory, and that a single HashMap is involved.



_____ Click on the "Details" link to display the shortest paths to an **accumulation point**.

Note:

An accumulation point is simply a reference that is suddenly responsible for keeping lots of heap space alive. In this case, the single HashMap at the top of the table has been identified as the single object that is responsible for the large accumulation of further objects.

The shortest path to this accumulation point shows what is responsible for keeping that accumulation object alive. In this case, a class loader, Thread and ShoppingServlet Class are referring to the HashMap.

Shortest Paths To the Accumulation Point

Class Name	Shallow Heap	Retained Heap
java.util.HashMap$Entry[131072] @ 0xa3218f8	524,304	175,662,656
java.util.HashMap @ 0x72455c8	48	175,662,720
java.util.HashSet @ 0x72455b8	16	175,662,736
class com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet @ 0x7b79c28	76	175,663,324
<Java Local> com.ibm.ws.util.ThreadPool$Worker @ 0x2ca1318 Thread	**136**	**176,720**
<class> com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet @ 0x72455f8 »	24	24
com.ibm.ws.classloader.CompoundClassLoader @ 0x7138118 »	152	6,988
∑ Total: 3 entries		

_____ Scroll down to view the accumulated objects.

This view shows the objects referred to by the HashMap accumulation point. You will see the HashMap contains entries that total approximately 175Mb or 65% of the total heap space. Only the first 20 entries from the HashMap are shown in this report.

Accumulated Objects

Class Name	Shallow Heap	Retained Heap	Percentage
class com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet @ 0x7b79c28	76	175,663,324	65.27%
java.util.HashSet @ 0x72455b8	16	175,662,736	65.27%
java.util.HashMap @ 0x72455c8	48	175,662,720	65.27%
java.util.HashMap$Entry[131072] @ 0xa3218f8	524,304	175,662,656	65.27%
java.util.HashMap$Entry @ 0x6c90f78	32	211,888	0.08%
java.util.HashMap$Entry @ 0x8de8070	32	211,888	0.08%
java.util.HashMap$Entry @ 0x6f79bd0	32	209,304	0.08%
java.util.HashMap$Entry @ 0x718f4a8	32	209,304	0.08%
java.util.HashMap$Entry @ 0x75c46d0	32	209,304	0.08%
java.util.HashMap$Entry @ 0x75f3450	32	209,304	0.08%
java.util.HashMap$Entry @ 0x761e770	32	209,304	0.08%
java.util.HashMap$Entry @ 0x76f2870	32	209,304	0.08%
java.util.HashMap$Entry @ 0x786f5d8	32	209,304	0.08%
java.util.HashMap$Entry @ 0x93ad788	32	209,304	0.08%
java.util.HashMap$Entry @ 0x60afbd0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x68706e0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6b967e0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6f7d430	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6fba370	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6fe61a8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7068188	32	206,720	0.08%
java.util.HashMap$Entry @ 0x725b4c8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7478c30	32	206,720	0.08%
∑ Total: 20 entries	640	4,165,408	0.015

_____ Scroll down to view the "Accumulated Object by Class" table. This shows the total number of objects referred to by the HashMap accumulation point, in this case over 17,000 objects with a retained size of over 175Mb. Of course, the exact statistics in your heap dump will vary.

Accumulated Objects by Class

Label	Number Of Objects	Used Heap Size	Retained Heap Size
java.util.HashMap$Entry	17,928	573,696	175,138,352

Note:

Shallow heap refers to the size of an individual object in isolation, and **retained heap** includes all the objects that are referenced (and kept alive) by that object.

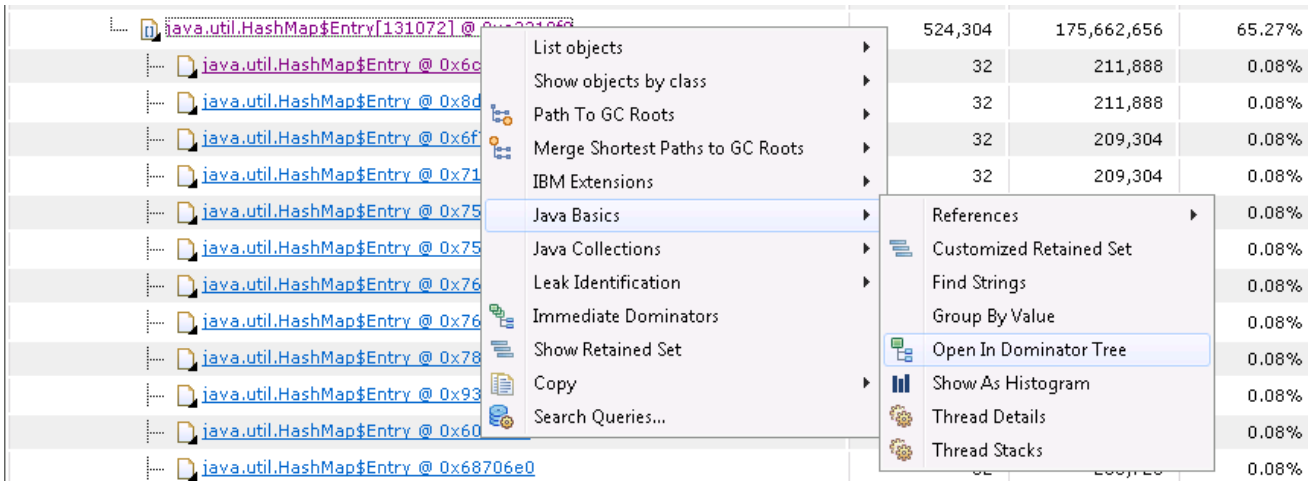
_____ Scroll to the "**Accumulated Objects**" table.

_____ To determine what is stored in each of these HashMap entries, **left click** the accumulation point, i.e. the HashMapEntry **with the array icon** .

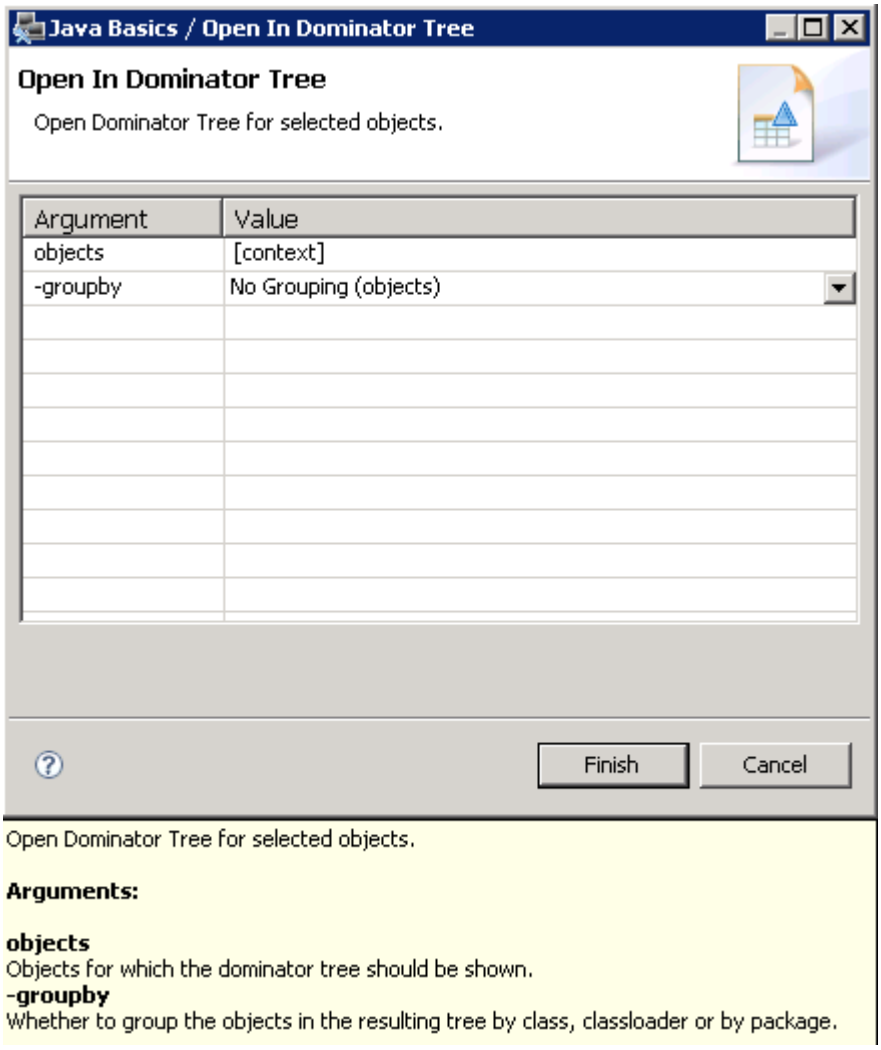
_____ Select "Java Basics->Open in Dominator Tree" to display the most significant references from the selected HashMap object.

Note:

As an alternative to the Dominator Tree, you could show **all** references with List objects->with outgoing references.



_____ Click "Finish" on the "Open In Dominator" dialog.

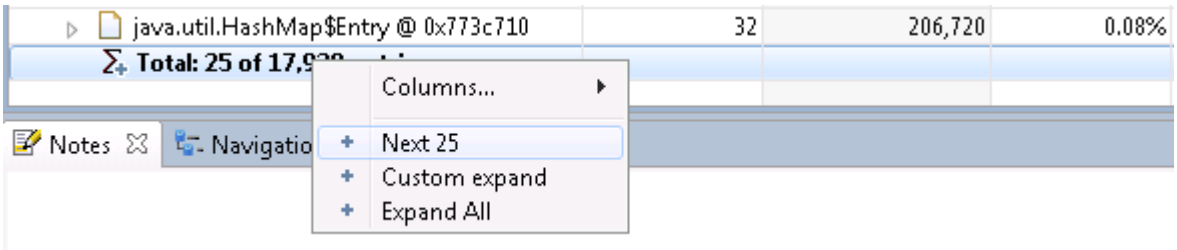


_____ Expand the top level HashMap object to show the thousands of entries that compose this HashMap.

Overview | Histogram | default_report org.eclipse.mat.api:suspects | show_dominator_tree [context]

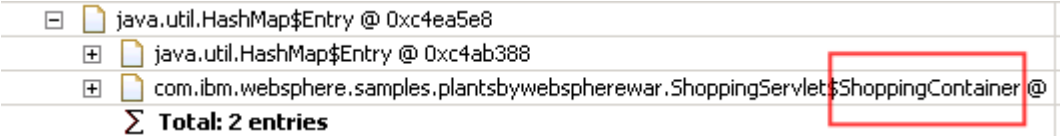
Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.HashMap$Entry[131072] @ 0xa3218f8	524,304	175,662,656	65.27%
java.util.HashMap$Entry @ 0x6c90f78	32	211,888	0.08%
java.util.HashMap$Entry @ 0x8de8070	32	211,888	0.08%
java.util.HashMap$Entry @ 0x6f79bd0	32	209,304	0.08%
java.util.HashMap$Entry @ 0x718f4a8	32	209,304	0.08%
java.util.HashMap$Entry @ 0x75c46d0	32	209,304	0.08%
java.util.HashMap$Entry @ 0x75f3450	32	209,304	0.08%
java.util.HashMap$Entry @ 0x761e770	32	209,304	0.08%
java.util.HashMap$Entry @ 0x76f2870	32	209,304	0.08%
java.util.HashMap$Entry @ 0x786f5d8	32	209,304	0.08%
java.util.HashMap$Entry @ 0x93ad788	32	209,304	0.08%
java.util.HashMap$Entry @ 0x60afbd0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x68706e0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6b967e0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6f307a0	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6f7d430	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6fba370	32	206,720	0.08%
java.util.HashMap$Entry @ 0x6fe61a8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7068188	32	206,720	0.08%
java.util.HashMap$Entry @ 0x725b4c8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7478c30	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7565828	32	206,720	0.08%
java.util.HashMap$Entry @ 0x7739498	32	206,720	0.08%
java.util.HashMap$Entry @ 0x773a8c8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x773bcf8	32	206,720	0.08%
java.util.HashMap$Entry @ 0x773c710	32	206,720	0.08%
Total: 25 of 17,928 entries			

_____ Display some further entries by **right clicking** the “Total” and selecting “Next 25”. The “Expand All” option is likely to take some time so **avoid** clicking that.



_____ Expand one of the HashMap entries.

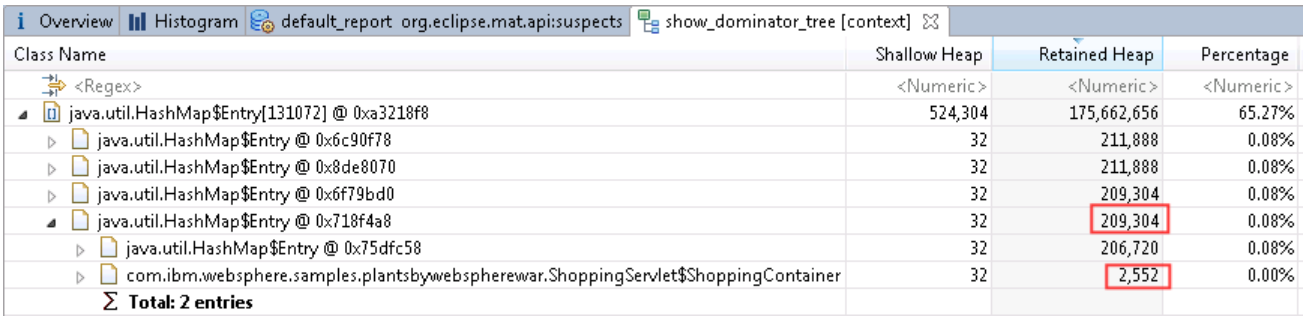
You will notice that the HashMap entry refers to a ShoppingContainer class in the plants sample.



Note:

The class name "ShoppingServlet$ShoppingContainer" means the "ShoppingContainer" class is an inner class of "ShoppingServlet".

This ShoppingContainer object is responsible for keeping approximately 2,552 bytes alive in the heap. The HashMapEntry actually keeps 209,304 bytes alive in total, including the ShoppingContainer. All the other HashMapEntry objects in the list follow an identical pattern. Their retained heap varies, but they always contain a ShoppingContainer.



Class Name	Shallow Heap	Retained Heap	Percentage
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.util.HashMap$Entry[131072] @ 0xa3218f8	524,304	175,662,656	65.27%
java.util.HashMap$Entry @ 0x6c90f78	32	211,888	0.08%
java.util.HashMap$Entry @ 0x8de8070	32	211,888	0.08%
java.util.HashMap$Entry @ 0x6f79bd0	32	209,304	0.08%
java.util.HashMap$Entry @ 0x718f4a8	32	209,304	0.08%
java.util.HashMap$Entry @ 0x75dfc58	32	206,720	0.08%
com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet$ShoppingContainer	32	2,552	0.00%
Total: 2 entries			

You have established that objects from the plants sample seem to be involved with the memory leak - a **ShoppingServlet** is referring to a large **HashMap** that contains many instances of **ShoppingContainer**. Let's look at the overall footprint of the entire plants sample.

_____ Click on the  icon to open a histogram.

_____ Type "**com.ibm.websphere.samples.plantsbywebspherewar.***" in the "Regex" filter box and press the Enter key. The Regex filter box is the first line of the table, starting with the icon <Regex>.

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

Class Name	Objects	Shallow Heap	Retained Heap
com.ibm.websphere.samples.plantsbywebspherewar.*	<Numeric>	<Numeric>	<Numeric>
com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet$ShoppingContainer	67,779	2,168,928	>= 172,969,564
com.ibm.websphere.samples.plantsbywebspherewar.ImageServlet	1	24	>= 100
com.ibm.websphere.samples.plantsbywebspherewar.ShoppingServlet	1	24	>= 175,663,348
com.ibm.websphere.samples.plantsbywebspherewar.AdminServlet	0	0	>= 76
com.ibm.websphere.samples.plantsbywebspherewar.AccountServlet	0	0	>= 76
Σ Total: 5 entries (19,926 filtered)	**67,781**	**2,168,976**	

In this heap dump there are 67,779 instances of ShoppingContainer with a shallow heap of 2Mb and a retained heap of 172Mb. The cause of this memory leak is becoming increasingly clear, and it seems only the ShoppingContainer and ShoppingServlet classes are involved.

Optional Steps:

_____ Double click the desktop shortcut for ShoppingServlet.java to inspect the programming error.

_____ Click “Edit->Find” and search for “ShoppingContainer” to locate the definition of the inner class.

_____ Click “Edit->Find” and search for method “deliberateMemoryLeak”, click “Find Again”.

You will find method “deliberateMemoryLeak” adds 10000 instances of the ShoppingContainer inner class to a **static** HashMap every time the wheelbarrow image is clicked. The constructor of the ShoppingContainer class fills a pointless array of size 2500 bytes. This is the source of the memory leak.

```
private void deliberateMemoryLeak() {


    // ---------------------------------------------------------------------------
    // User clicked on the wheelbarrow, let's fill it with a
    // memory leak.
    // ---------------------------------------------------------------------------
    System.out.println("==> STARTING MEMORY LEAK");


    int LEAK_SIZE = 10000;


    for (int i = 0; i < LEAK_SIZE; i++) {
        leakObject.add(new ShoppingContainer());
    }

    System.out
    .println("==> Added "
            + LEAK_SIZE
            + " objects to memory leak.  The leak now contains "
            + leakObject.size() + " objects.");


    System.out.println("==> ENDING MEMORY LEAK");
}



class ShoppingContainer {

    private byte[] array;
    private long timestamp;

    ShoppingContainer() {
        array = new byte[2500];
        Arrays.fill(array, (byte) 65);
        timestamp = System.currentTimeMillis();
    }
}
```

Part 7: Using the IBM Extensions to Memory Analyzer for Further Memory Analysis (10 minutes)

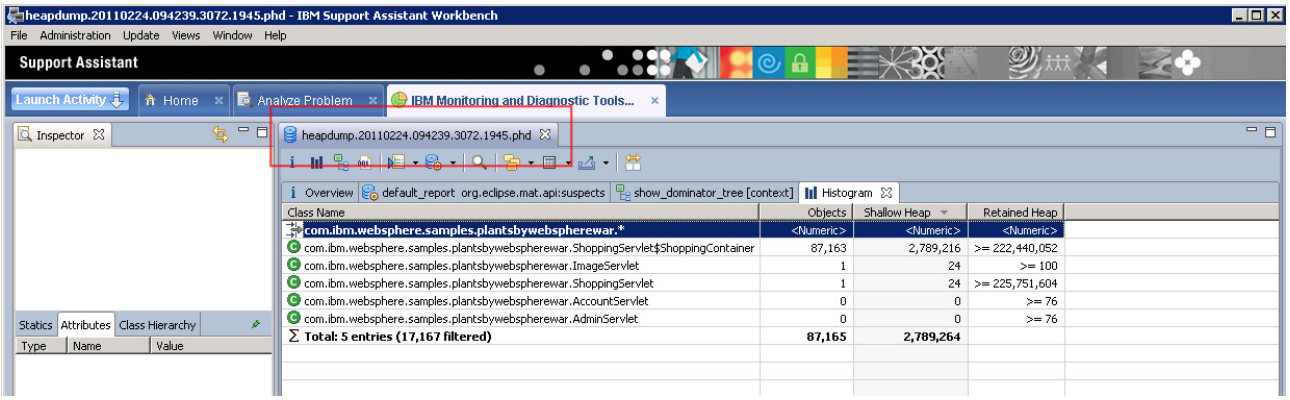
Note:

"**IBM Extensions for Memory Analyzer**" add-ons have been installed into ISA. These have recently been released by IBM via the **IBM Alphaworks** site. They are unsupported, but very useful extensions to the Memory Analyzer tool.

They provide IBM product specific analysis of heap dumps by applying knowledge of the internal data structures of the IBM products into useful reports. For example, there is a WebSphere Application Server report to view the size and other details of the WebSphere objects in the heap dump that hold the HTTP sessions. This enables the Memory Analyzer tool to be used for **more than just diagnosing memory leaks**.

Most of these extensions require the additional data only available in a full system dump which you will analyze in this final part of the lab.

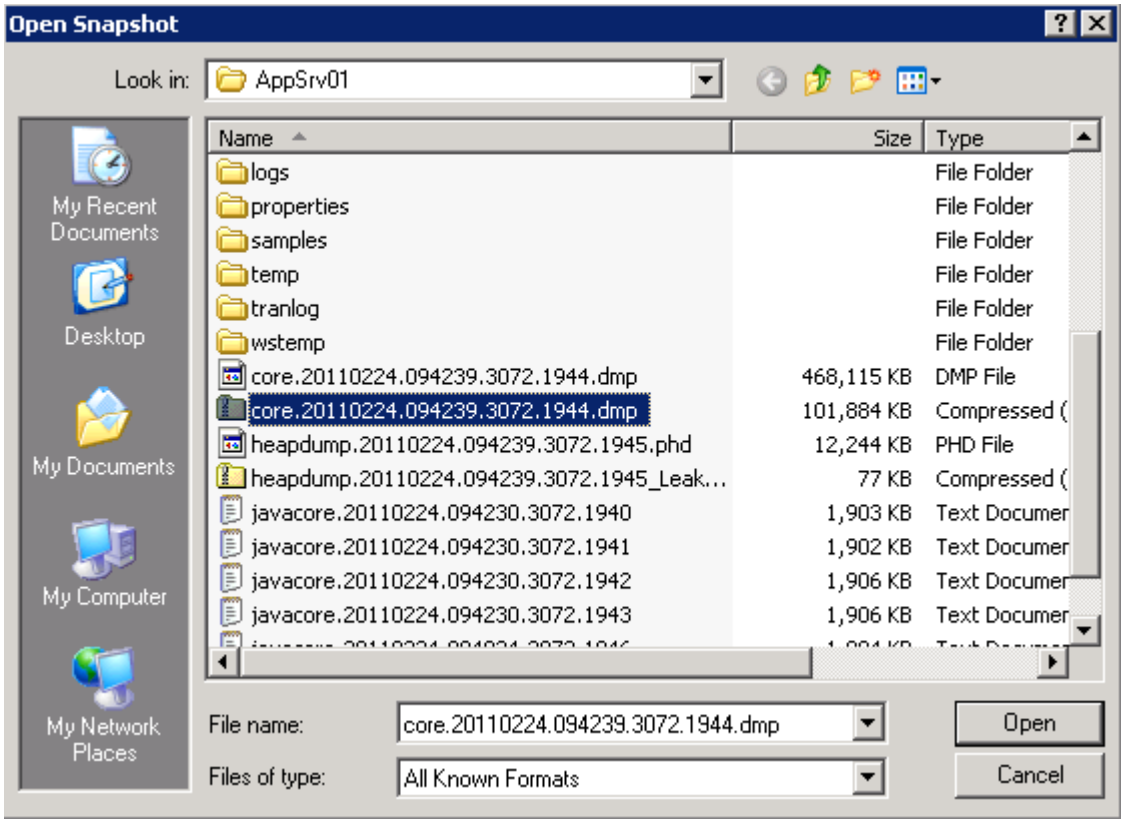
_____ Return to the ISA window and close the Memory Analyzer tab which contains the analysis of the IBM heap dump.



_____ Click "File->Open heap dump".

_____ Navigate to the system dump you previously processed with jextract.

_____ Select the compressed .dmp file and click "Open". **Please be patient, parsing a system dump can take longer that a compressed IBM heap dump. Approximately 3-5 minutes may pass with little nothing reported by the progress bar.**



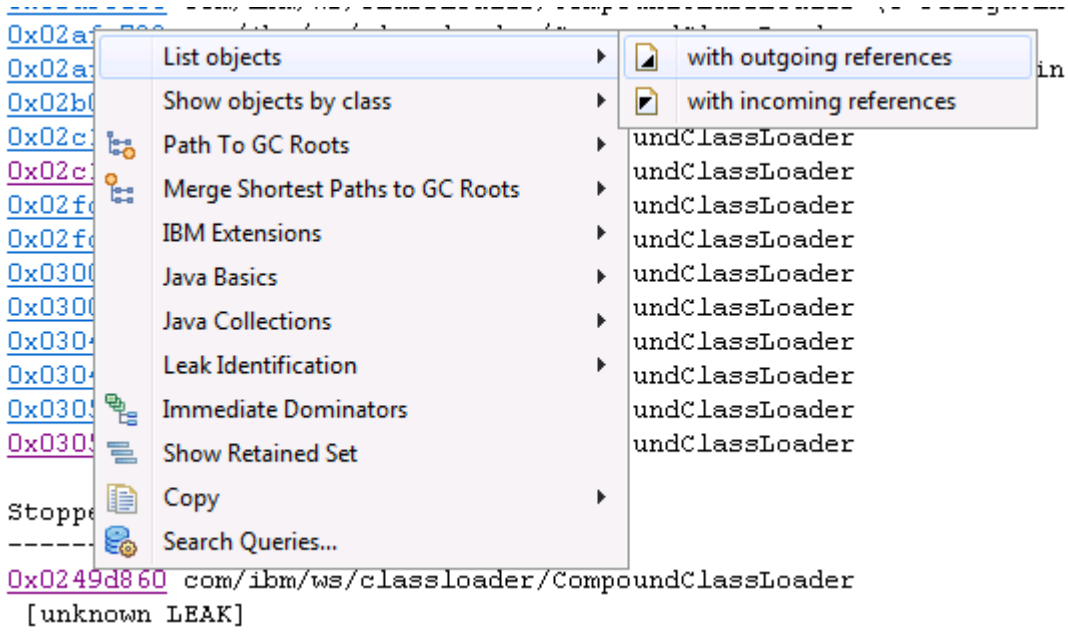
_____ On the “Getting Started Wizard” box, click “**Cancel**” to avoid generating the standard reports.

_____ Click the reports icon  and select “IBM Extensions->WebSphere Application Server->Application Classloader Leaks”.

This is a useful way of visualizing which class loaders, and therefore which WebSphere applications are exhibiting memory leaks. In this lab, you will find one has been identified as exhibiting signs of a memory leak.

```
Overview   application_classloader_leaks
0x0294f450 com/ibm/ws/classloader/CompoundClassLoader
0x02988a48 com/ibm/ws/classloader/CompoundClassLoader
0x02991978 com/ibm/ws/classloader/CompoundClassLoader
0x02992380 com/ibm/ws/classloader/CompoundClassLoader
0x029933b0 com/ibm/ws/classloader/CompoundClassLoader
0x02995018 com/ibm/ws/classloader/CompoundClassLoader
0x029a9d18 com/ibm/ws/classloader/CompoundClassLoader
0x02a9bd80 com/ibm/ws/classloader/CompoundClassLoader
0x02a9c778 com/ibm/ws/classloader/CompoundClassLoader
0x02a9f150 com/ibm/ws/classloader/CompoundClassLoader (0 DelegatingClassLoader, 1 JSPExtensionClassLoader)
0x02afe788 com/ibm/ws/classloader/CompoundClassLoader
0x02aff090 com/ibm/ws/classloader/CompoundClassLoader (0 DelegatingClassLoader, 3 JSPExtensionClassLoader)
0x02b00598 com/ibm/ws/classloader/CompoundClassLoader
0x02c193d8 com/ibm/ws/classloader/CompoundClassLoader
0x02c19da8 com/ibm/ws/classloader/CompoundClassLoader
0x02fc0248 com/ibm/ws/classloader/CompoundClassLoader
0x02fc0d20 com/ibm/ws/classloader/CompoundClassLoader
0x03002fe8 com/ibm/ws/classloader/CompoundClassLoader
0x03003d48 com/ibm/ws/classloader/CompoundClassLoader
0x03049710 com/ibm/ws/classloader/CompoundClassLoader
0x0304f128 com/ibm/ws/classloader/CompoundClassLoader
0x030567d0 com/ibm/ws/classloader/CompoundClassLoader
0x03058aa8 com/ibm/ws/classloader/CompoundClassLoader

Stopped application class loaders
---------------------------------
0x0249d860 com/ibm/ws/classloader/CompoundClassLoader
 [unknown LEAK]
```

_____ Left click the class loader identified as leaking and select List Objects->with outgoing references



_____ Expand the class loader references.

The resulting objects loaded by this class loader relate to the Plants sample which gives a clear indication of the application which is leaking memory.

Overview | application_classloader_leaks | list_objects [context]

Class Name	Shallow Heap	Retained Heap
com.ibm.ws.classloader.CompoundClassLoader @ 0x249d860	152	39,088
<class> class com.ibm.ws.classloader.CompoundClassLoader @ 0x1b6bca0	9,496	10,680
parent, parent com.ibm.ws.classloader.ProtectionClassLoader @ 0x2170cd0	96	648
assertionLock java.lang.ClassLoader$AssertionLock @ 0x249d8f8	16	16
packages java.util.Hashtable @ 0x249daa8	40	136
lazyInitLock java.lang.ClassLoader$LazyInitLock @ 0x249db10	16	16
packageSigners java.util.Hashtable @ 0x249db20	40	168
methodCache java.util.Hashtable @ 0x249db88	40	104
fieldCache java.util.Hashtable @ 0x249dbf0	40	18,616
constructorCache java.util.Hashtable @ 0x249dc58	40	104
pds java.util.HashMap @ 0x249dcc0	48	160
nativelibpaths java.lang.String[0] @ 0x249dd40	16	16
libraryClassLoaders com.ibm.ws.classloader.CompoundClassLoader[0] @ 0x249	16	16
reloadableParents java.util.Vector @ 0x249dd60	32	88
resourceRequestCache java.util.Collections$SynchronizedMap @ 0x249ddf0	24	9,136
providers com.ibm.ws.classloader.SinglePathClassProvider[1] @ 0x24a6d88	24	2,544
localClassPath java.lang.String @ 0x24a6da0 C:\IBM\WebSphere\AppServer\pro	32	296
preDefinePlugins java.util.ArrayList @ 0x24a6ec8	32	88
class com.ibm.websphere.samples.plantsbywebsphereejb.Supplier @ 0x3193f60	229	229
class com.ibm.websphere.samples.plantsbywebsphereejb.OrderItem @ 0x31ca1	391	391
class com.ibm.websphere.samples.plantsbywebsphereejb.OrderItem$PK @ 0x31	265	265
class com.ibm.websphere.samples.plantsbywebsphereejb.IdGenerator @ 0x3279	154	154
class com.ibm.websphere.samples.plantsbywebsphereejb.Inventory @ 0x345122	644	644
annotationCache java.util.Hashtable @ 0x34918e8	40	7,432
class com.ibm.websphere.samples.plantsbywebsphereejb.BackOrder @ 0x34aa4	303	303
class com.ibm.websphere.samples.plantsbywebsphereejb.Order @ 0x37c9f28	666	666

_____ Click the reports icon  and select “IBM Extensions->WebSphere Application Server->WAS Cache Analysis”.

This illustrates the contents of the WebSphere caches such as dynacache. These types of caches reside in memory. If the WAS cache report shows a high memory footprint, the size of the caches can be limited with WebSphere administration, or the cache contents can be automatically offloaded to disk. For this lab, there is no action required.

Overview | was_cache_analysis

Cache Analysis

Cache Analysis

WAS Caches Found

Address	Cache Name	Cache Size	Cache Size Limit	Disk Offload	Shallow Heap	Retained Heap
0x2a9dde0	com.ibm.workplace/ExtensionRegistryCache	14	5000	disabled	320	169024
0x2a2b610	com.ibm.ws.wssecurity.sctServiceCacheMap	0	2000	disabled	320	64184
0x2a061e8	com.ibm.ws.wssecurity.sctClientCacheMap	0	2000	disabled	320	64184
∑ Total: 3 entries						

EJS Caches Found

Address	Cache Name	Entries	Soft Limit	Hard Limit	Eviction Strategy	Shallow Heap	Retained Heap
0x2b64d70	Wrapper Cache	4	4106	4516	SweepLruEvictionStrategy	72	147984
0x2b2ce08	EJB Cache	0	2053	2258	BackgroundLruEvictionStrategy	72	205416
∑ Total: 2 entries							

_____ Click the reports icon  and select “IBM Extensions->WebSphere Application Server->Web Container Analysis”.

This shows details of all the configured web applications. Observe that in this case only the Plants by WebSphere application has any active sessions. In the next step, we will check the memory size of these sessions.

Overview | was_overview | web_container_analysis

Web Application Analysis

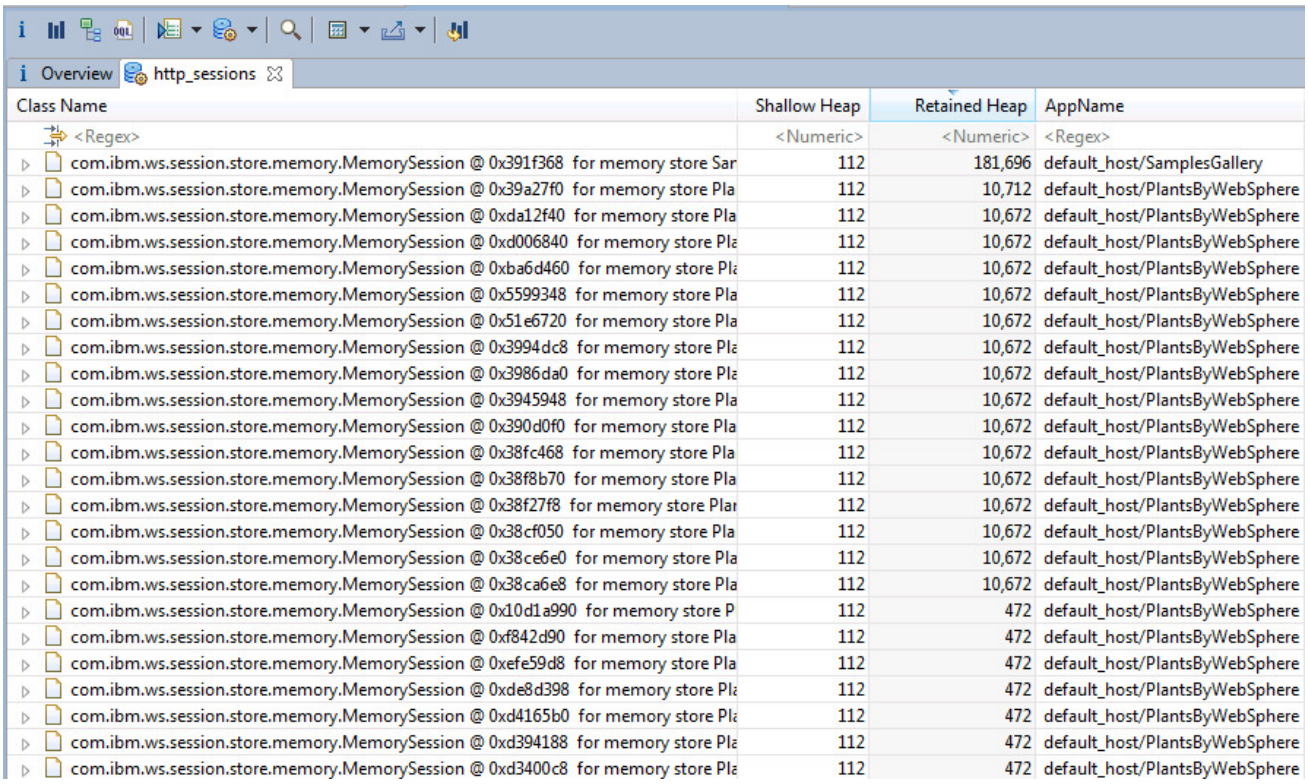
Web Application Analysis

Web Application Details

Address	Virtual Host Name	Web Group Name	Web App Name	Loader	Destroyed	Current Sessions
0x3717a60	default_host	/PlantsByWebSphere/docs/*	PlantsByWebSphere	war:PlantsByWebSphere/PlantsGallery.war	false	0
0x27f9430	default_host	/SamplesGallery/*	SamplesGallery	war:SamplesGallery/GalleryMenu.war	false	1
0x2a79258	default_host	/IBM_WS_SYS_RESPONSESERVLET/*	ibmasyncrsp	war:ibmasyncrsp/ibmasyncrsp.war	false	0
0x30200e8	default_host	/WSsamples/*	SamplesGallery	war:SamplesGallery/Gallery.war	false	1
0x1f48198	default_host	/*	DefaultApplication	war:DefaultApplication/DefaultWebApplication.war	false	0
0x2d815c8	default_host	/PlantsByWebSphere/*	PlantsByWebSphere	war:PlantsByWebSphere/PlantsByWebSphere.war	false	391
0x32321c8	admin_host	/ISCAdminPortlet/*	isclite	war:isclite/ISCAdminPortlet.war	false	0
0x262c598	admin_host	/FileTransfer/*	filetransferSecured	war:filetransferSecured.ear/filetransfer.war	false	0
0x312e568	admin_host	/ibm/help/*	isclite	war:isclite/iehs.war	false	0
0x36e1de8	admin_host	/wim/*	isclite	war:isclite/WIMPortlet.war	false	0
0x2661028	admin_host	/ibm/console/*	isclite	war:isclite/isclite.war	false	0
0x37fb5c8	admin_host	/wasportlet/*	isclite	war:isclite/wasportlet.war	false	0
∑ Total : 12 entri						

_____ Click the reports icon  , select "IBM Extensions->WebSphere Application Server->HTTP Sessions". This shows details of all HTTP sessions including the size, session attributes, timeout, user ID and session ID.

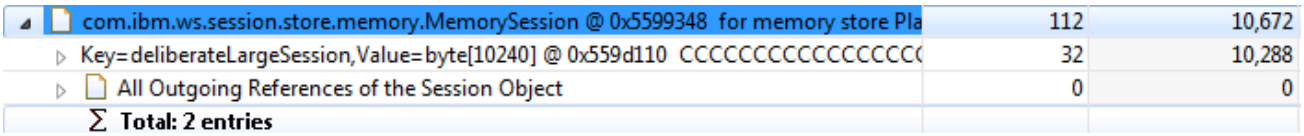
_____ Ensure the table of sessions is ordered by "Retained Heap". Notice how some sessions for PlantsByWebSphere are around 10k while others are just 0.5k.



Overview | http_sessions

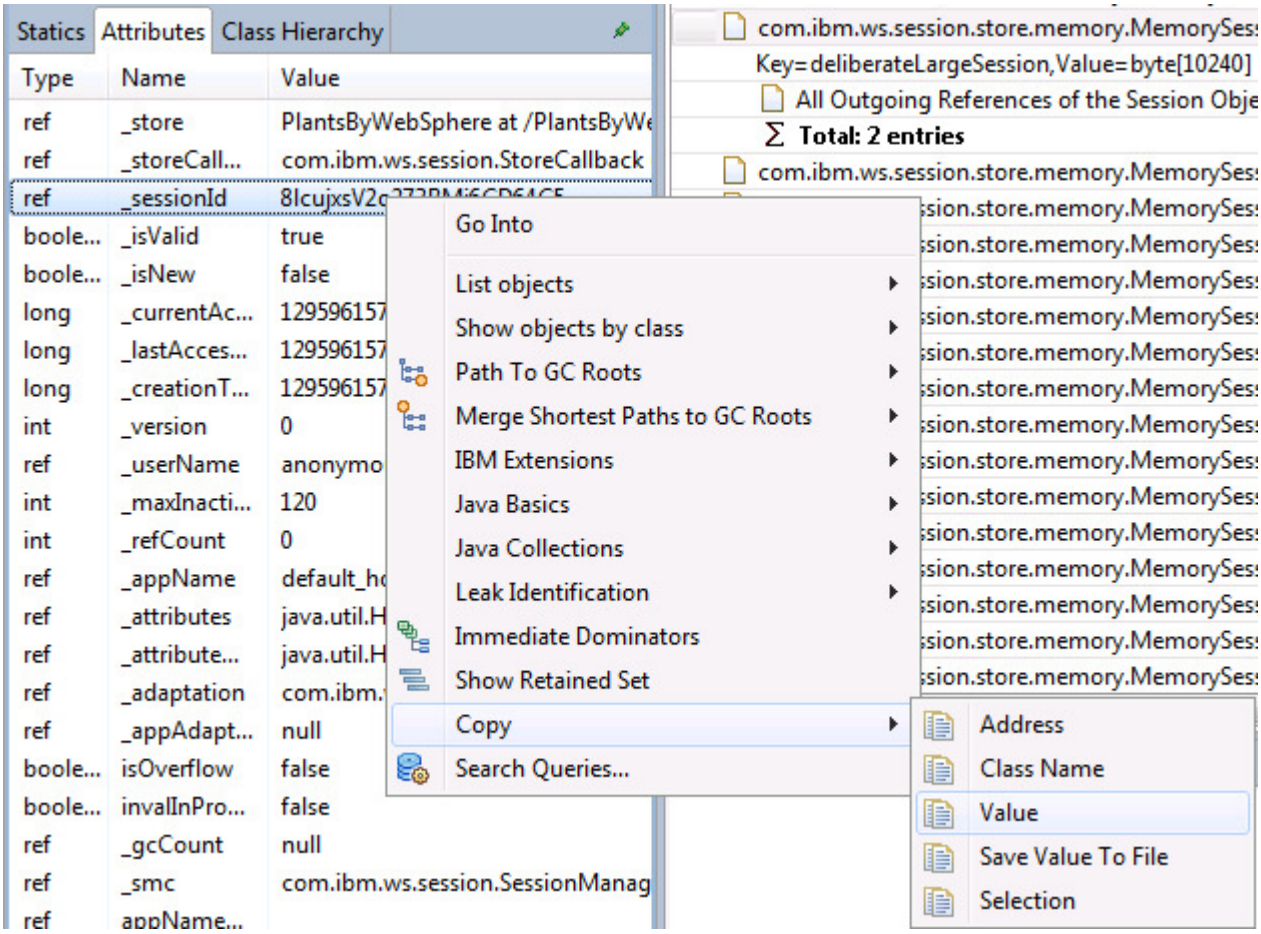
Class Name	Shallow Heap	Retained Heap	AppName
<Regex>	<Numeric>	<Numeric>	<Regex>
com.ibm.ws.session.store.memory.MemorySession @ 0x391f368 for memory store Sar	112	181,696	default_host/SamplesGallery
com.ibm.ws.session.store.memory.MemorySession @ 0x39a27f0 for memory store Pla	112	10,712	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xda12f40 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xd006840 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xba6d460 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x5599348 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x51e6720 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x3994dc8 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x3986da0 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x3945948 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x390d0f0 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38fc468 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38f8b70 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38f27f8 for memory store Plar	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38cf050 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38ce6e0 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x38ca6e8 for memory store Pla	112	10,672	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0x10d1a990 for memory store P	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xf842d90 for memory store Pla	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xefe59d8 for memory store Pla	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xde8d398 for memory store Pla	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xd4165b0 for memory store Pla	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xd394188 for memory store Pla	112	472	default_host/PlantsByWebSphere
com.ibm.ws.session.store.memory.MemorySession @ 0xd3400c8 for memory store Pla	112	472	default_host/PlantsByWebSphere

_____ **Highlight** and then expand one of the **10k** session objects. The session attributes have been automatically extracted from the WebSphere objects and presented by the IBM Memory Analyzer extensions. Notice there is a key called "deliberateLargeSession" which contains a very long String of letter "C"'s. This looks like another deliberate mistake in the Plants application.



com.ibm.ws.session.store.memory.MemorySession @ 0x5599348 for memory store Pla	112	10,672
Key=deliberateLargeSession,Value=byte[10240] @ 0x559d110 CCCCCCCCCCCCCCCC(	32	10,288
All Outgoing References of the Session Object	0	0
Σ Total: 2 entries		

_____ Make a note of the "_sessionId" for the object. This can be seen on the "Attributes" tab, right click it and choose Copy->Value.



The best way to relate this unusually large session to the application code is to search the log files for the session ID. If the application uses this in its logging, you may be able to determine what the user did to cause the large session.

Note:

If you are using the "pre-prepared" system core instead of an "in-flight" system core generated during this lab, you will not be able to relate the session ID to the log files. In which case, simply read the remaining steps for this part of the lab.

_____ Launch Windows Explorer. Navigate to and open file:
"C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\SystemOut.log"

_____ Click "Edit->Find" and search the file for the Session ID you identified in the heap dump. It should reveal a log statement that gives a clue about the application's actions.

```
SystemOut     0 ==> STARTING DELIBERATE LARGE SESSION for ID=Qxo_5N8FZub0W38CoGqrYno
SystemOut     0 ==> ENDING DELIBERATE LARGE SESSION
```

Optional Steps:

_____ Double click the desktop shortcut for ShoppingServlet.java

_____ Click "Edit->Find" and and search for **"STARTING DELIBERATE LARGE SESSION"**.

The deliberate mistake is clear – for any user that clicks on the white poinsettia image, their session is loaded with an attribute containing a 10k string of "C"'s (ASCII code 67). Congratulations, you have successfully located the final deliberate mistake in the plants sample.

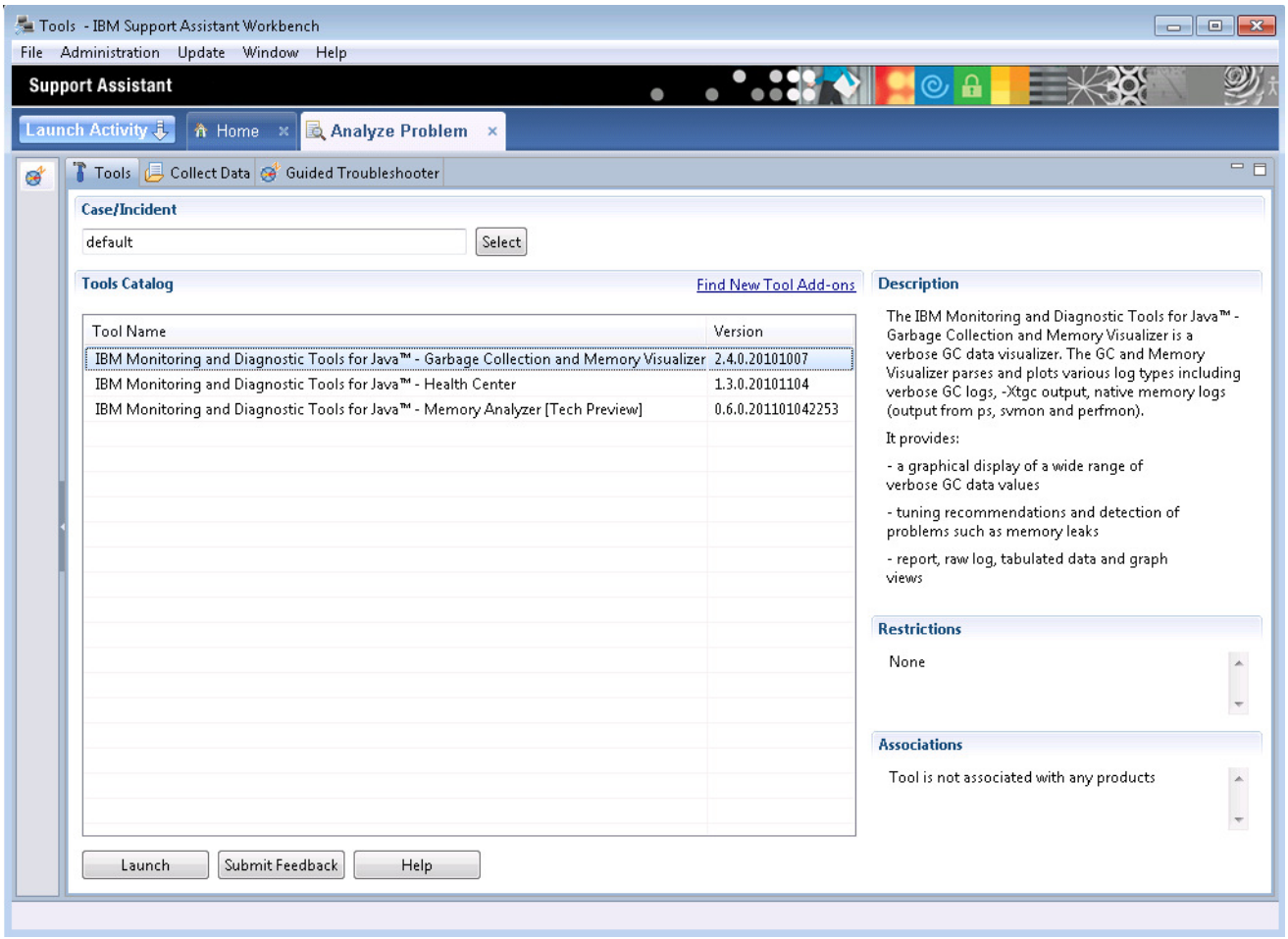
```
System.out.println("==> STARTING DELIBERATE LARGE SESSION for ID="+req.getSession().getId());

byte[] sessionAttr = new byte[10240];
Arrays.fill(sessionAttr, (byte) 67);
req.getSession().setAttribute("deliberateLargeSession", sessionAttr);

System.out.println("==> ENDING DELIBERATE LARGE SESSION");
```

Appendix A: *Optional* - Using the ISA and GCMV to Analyze GC Data (10 minutes)

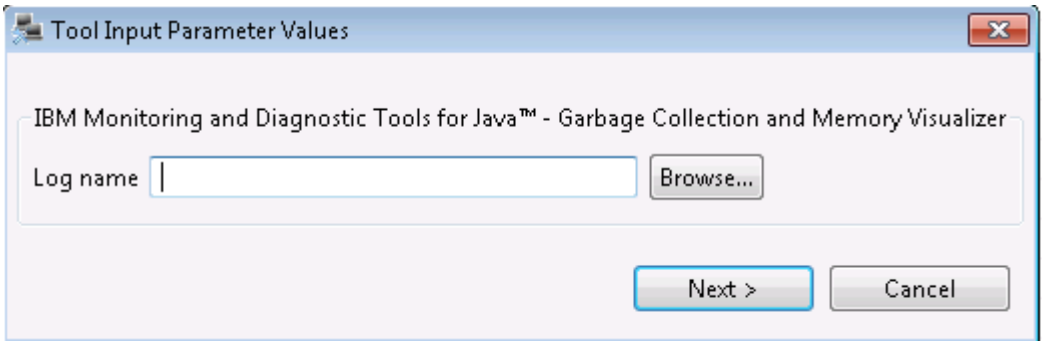
_____ Double click the IBM Support Assistant 4.1 shortcut on the desktop.

_____ Click the blue “Launch Activity” button and choose “Analyze Problem”.

_____ Select “Garbage Collection and Memory Visualizer” and click the grey “Launch” button at the bottom of the screen.



_____ Click “Next” to go directly into the GCMV tool.



_____ Click “File->Open File” and navigate to
“C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1\navtive_stderr.log”

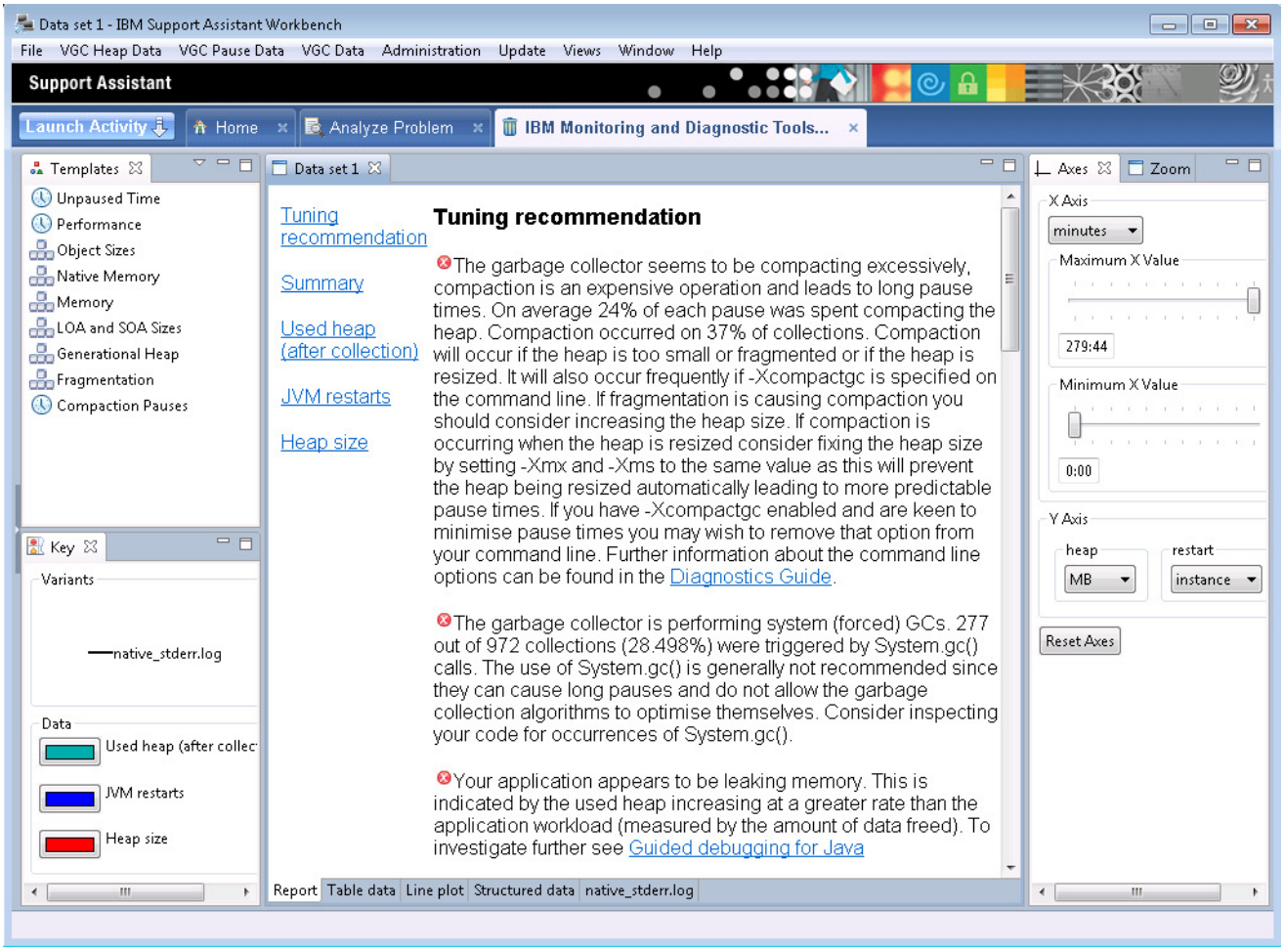
_____ Click “OK” and wait a few moments for the log file to be parsed.

_____ Click the “Report” tab at the bottom of the screen.

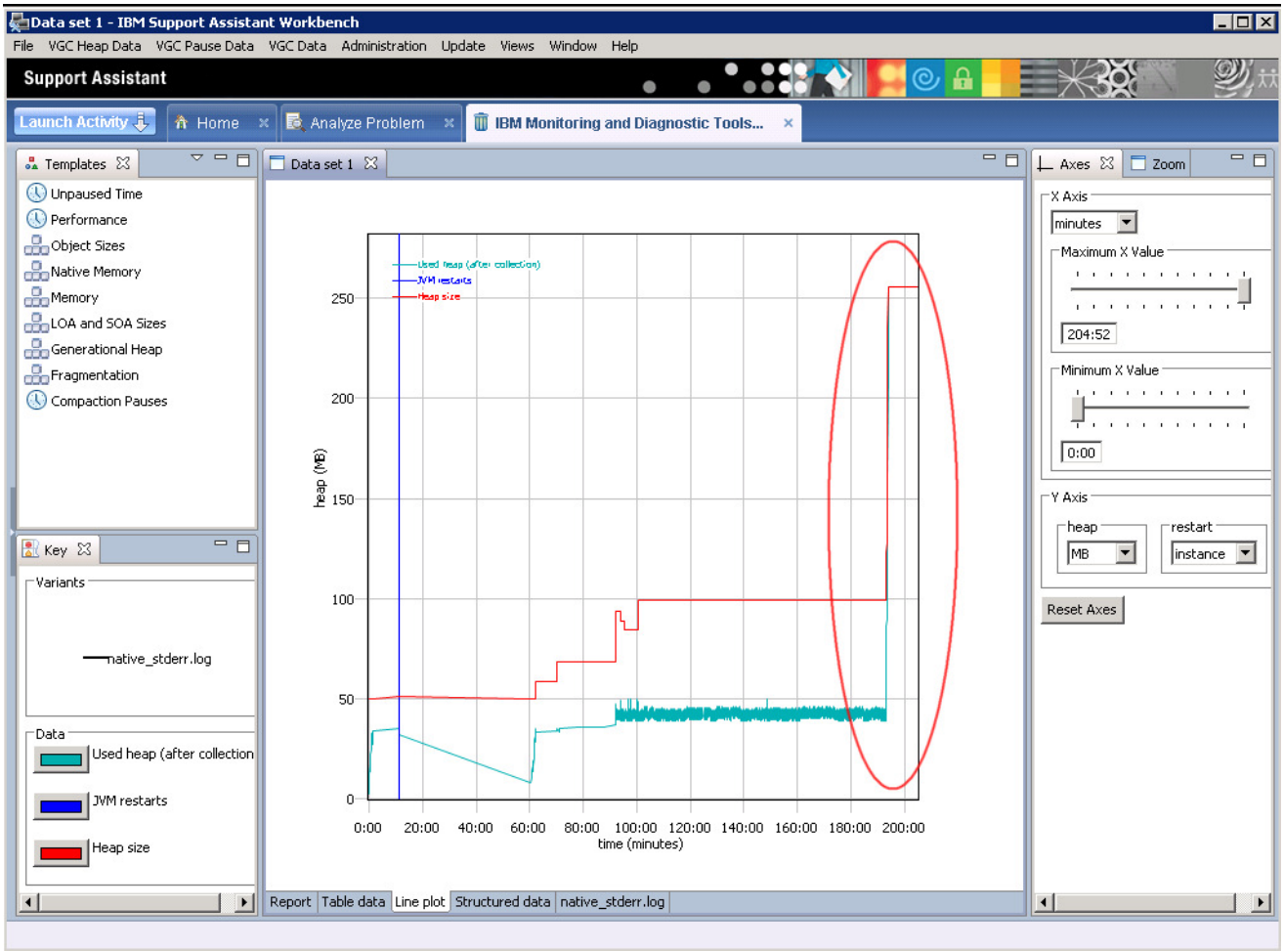
Review the recommendations - they relate to the same garbage collection issues you identified with live monitoring using Health Center, i.e.:

- **Excessive compaction** caused by excessive calls to **System.gc()**
- **Memory leakage** leading to an out of memory condition
- **Large object allocations**

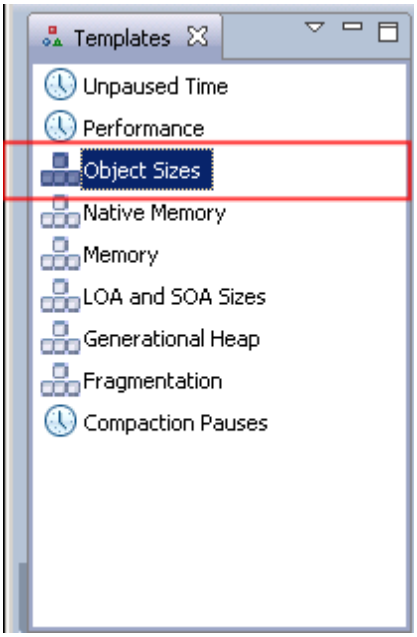
The report page also shows a summary table of garbage collection statistics.



_____ Click the “Line Plot” tab at the bottom of the screen. The graph shows the heap size and used heap after garbage collection. You will see these were stable for a while until a sudden increase when the JVM experienced a rapid memory leak (highlighted in this document by the red oval)

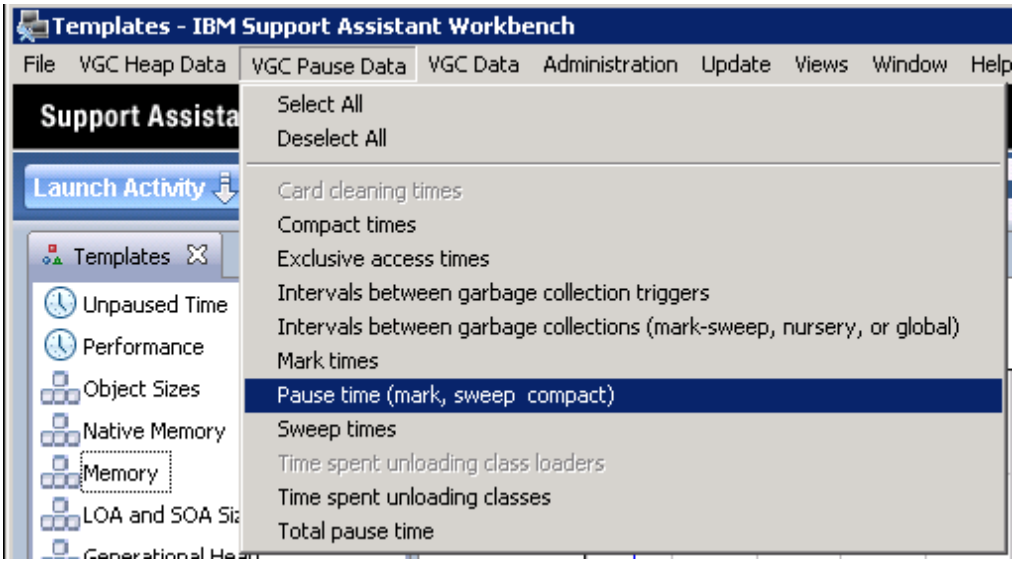


_____ The templates on the left hand side can be double clicked to show different garbage collection statistics. For example, double click the "**Object Sizes**" template graph which shows the size of objects that triggered an allocation failure. You will see very frequent requests for objects ranging from 5Mb to 10Mb (recall a large object allocation was one of the deliberate errors in the plants sample code)

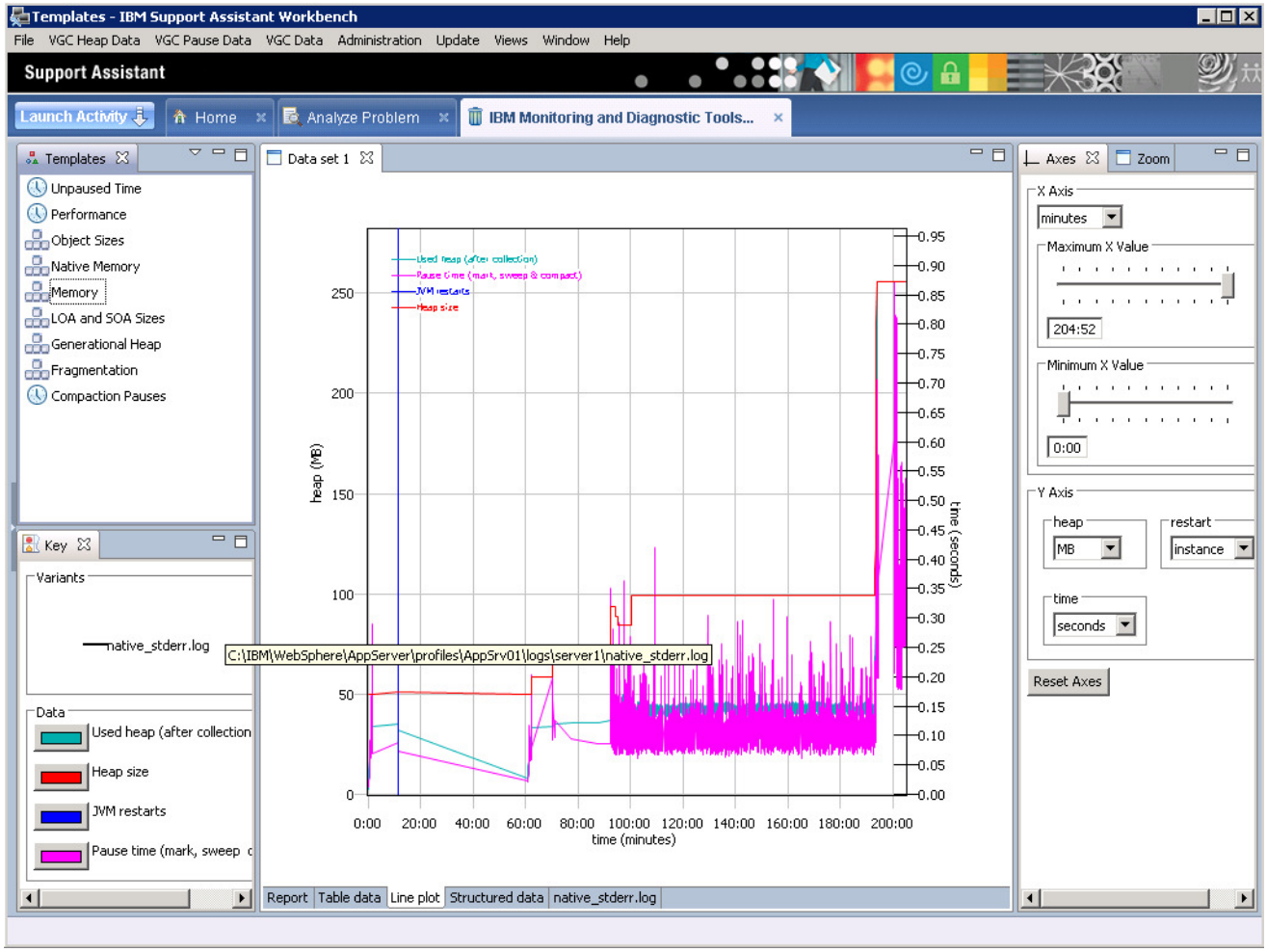


_____ Double click "**Memory**" in the templates window.

_____ You can customize any graph by choosing the data to plot. Click the menu "**VGC Pause data**" and enable (with a tick) menu item "**Pause Time (mark sweep compact)**".

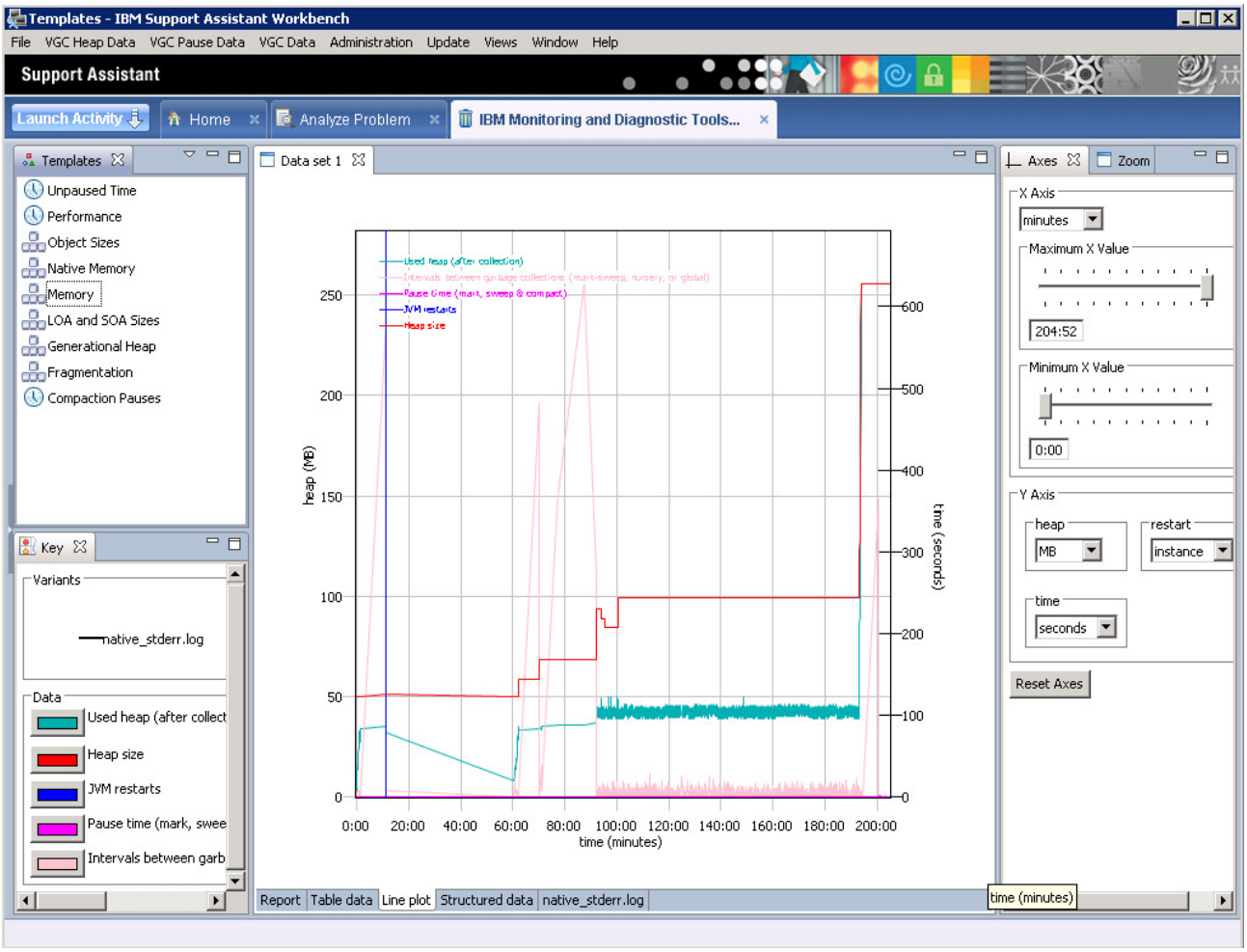


This will plot the total GC time which you see increasing dramatically as the JVM struggles to cope with the rapid demands of the **memory leak**.

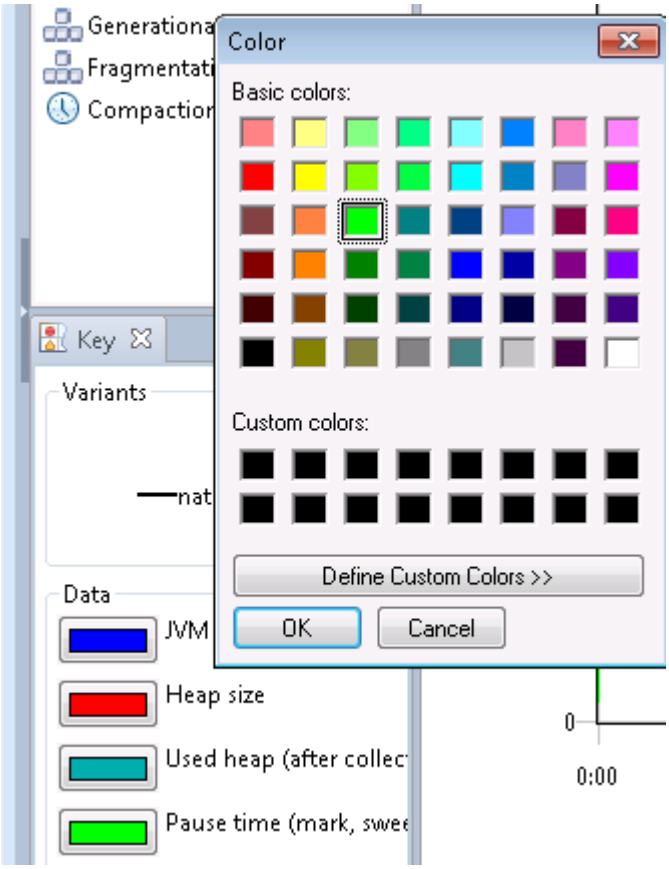


_____ Another interesting statistic is "**Intervals Between Garbage Collections**". Click the menu "**VGC Pause Data**" and enable (with a tick) menu item "**Intervals Between Garbage Collections**".

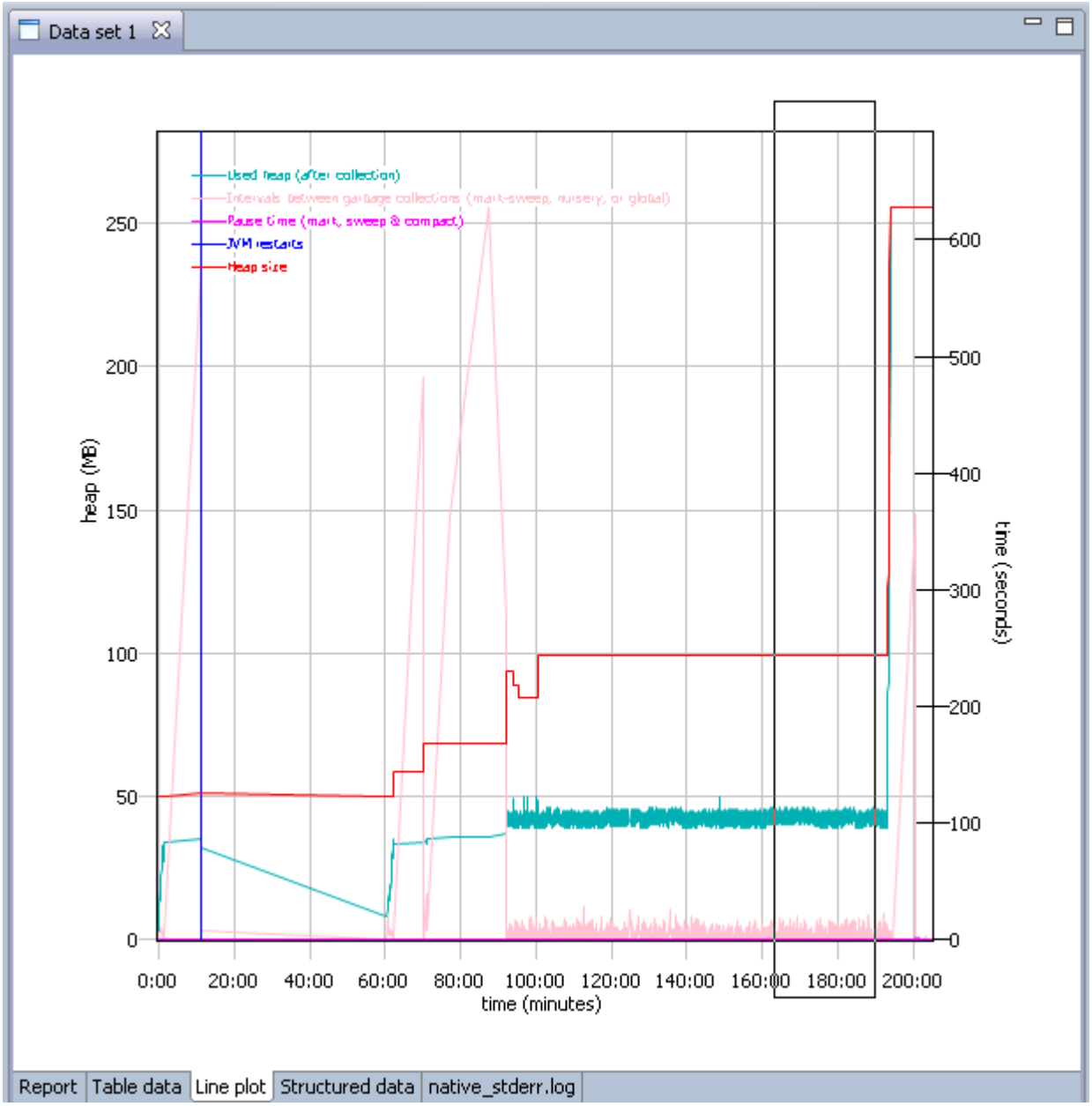
In this case, while the jmeter load generator was running, there is a often very short interval because garbage collections as the plants sample application was constantly calling **System.gc()**.



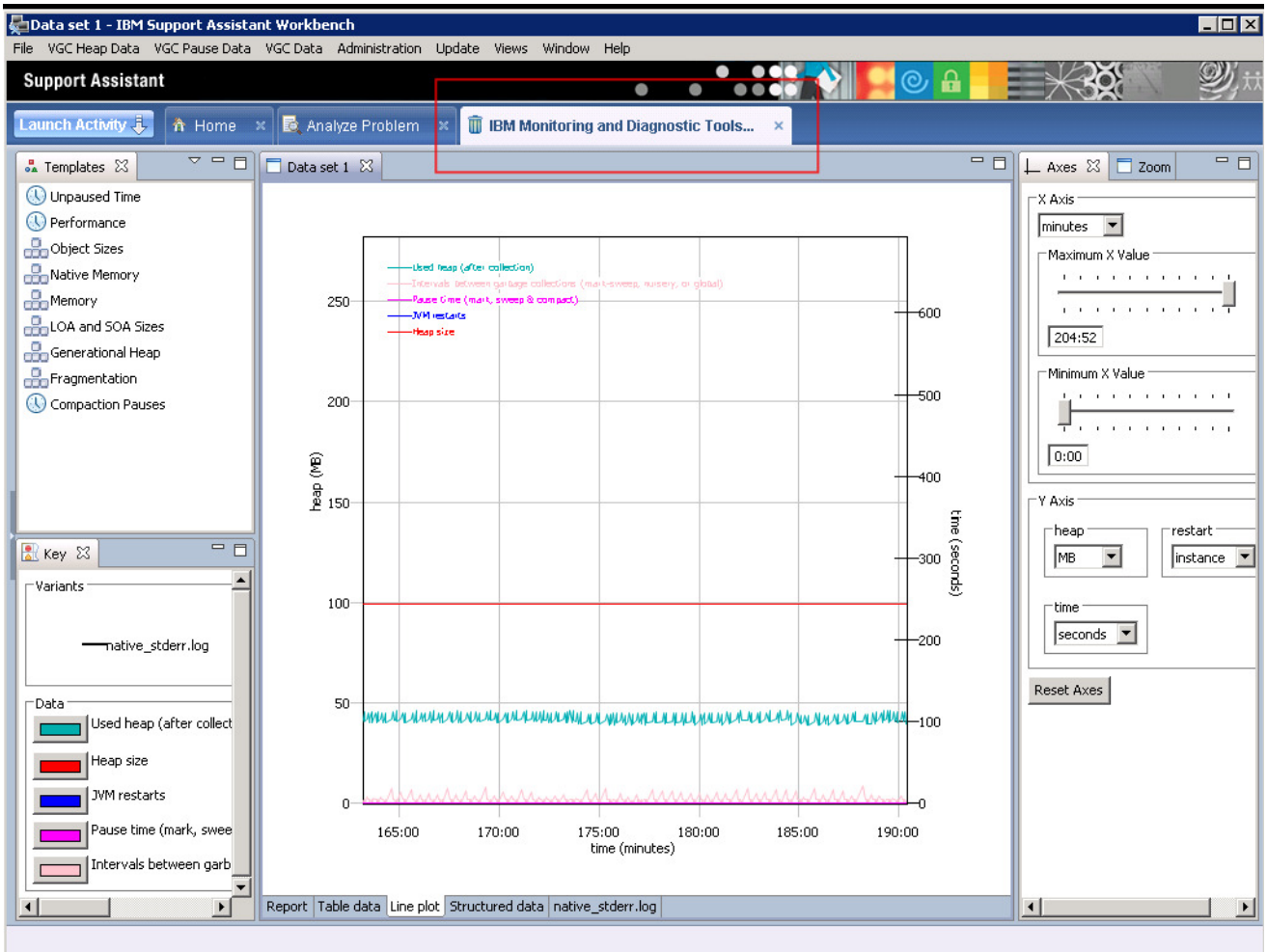
_____ Try changing the color of the lines by clicking the colored line icon.



_____ Try dragging a box in the graph area to zoom on a particular time period. Hint – zooming works best if you include the axes in your zoom box. You can right click and select “Reset Zoom” at any time.



_____ Close the GCMV tab but leave IBM Support Assistant Workbench open.


Data set 1 - IBM Support Assistant Workbench
File VGC Heap Data VGC Pause Data VGC Data Administration Update Views Window Help
Support Assistant
Launch Activity
Home
Analyze Problem
IBM Monitoring and Diagnostic Tools...
Templates
Unpaused Time
Performance
Object Sizes
Native Memory
Memory
LOA and SOA Sizes
Generational Heap
Fragmentation
Compaction Pauses
Key
Variants
native_stderr.log
Data
Used heap (after collect
Heap size
JVM restarts
Pause time (mark, swee
Intervals between garb
Data set 1
Used heap (after collection)
Intervals between garbage collections (mark-sweep, nursery, or global)
Pause time (mark, sweep & compact)
JVM restarts
Heap size
heap (MB)
time (seconds)
time (minutes)
165:00
170:00
175:00
180:00
185:00
190:00
Report
Table data
Line plot
Structured data
native_stderr.log
Axes
Zoom
X Axis
minutes
Maximum X Value
204:52
Minimum X Value
0:00
Y Axis
heap
MB
restart
instance
time
seconds
Reset Axes

Reference Links

- IBM Support Assistant Information and Downloads:

http://www-01.ibm.com/software/support/isa/

- How to Install JVM Tools into ISA:

http://www-01.ibm.com/support/docview.wss?uid=swg27013279

- Memory Analyzer Tool for ISA:

http://www.ibm.com/developerworks/java/jdk/tools/memoryanalyzer/

- Alphaworks IBM Memory Analyzer Extensions:

http://www.alphaworks.ibm.com/tech/iema

- Eclipse Memory Analyzer:

http://www.eclipse.org/mat/

- Using the IBM DTFJ with the Eclipse Memory Analyzer Tool (i.e. 64-bit):

http://www.ibm.com/developerworks/java/jdk/tools/mat.html

- MustGather: Using the -Xdump Option:

http://www-01.ibm.com/support/docview.wss?uid=swg21242497#Limiting_Dumps_Using_Filters

- Java Diagnostics Guide:

http://www.ibm.com/developerworks/java/jdk/diagnosis/

- Guided debugging for Java:

http://publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp?topic=/com.ibm.java.doc.igaa/_1vg00011e17d8ea-1163a087e6c-7ffe_1001.html

- IBM Java Troubleshooting Blog:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/troubleshootingjava/?lang=en