

Java Troubleshooting with ISA 5 using Health Center and Memory Analyzer

Java Troubleshooting with ISA 5 using Health Center and Memory Analyzer

What this lab is about.....	2
Lab requirements	3
What you should be able to do	3
Part 1: Lab Set Up.....	4
Part 2: IBM Support Assistant 5 Beta.....	5
Part 3: Setup Health Center to Monitor a running WebSphere JVM	15
Part 4: Use Health Center to Investigate Application Errors.....	21
Part 5: WebSphere Application Server Health Management.....	36
Part 6: Trigger a the Memory Leak to activate the Health Policy	46
Part 7: Using ISA and the Memory Analyzer to Analyze a Heapdump	60
Part 8: (Optional) Using the IBM Extensions to Memory Analyzer for Further Memory Analysis	85
Part 9: (Optional) Using the WebSphere Application Server Configuration Visualizer	100
Reference Links	105

What this lab is about

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

In this lab you will use IBM Support Assistant 5 Beta (ISA 5) to diagnose JVM issues experienced by a running WebSphere Application Server 8.5.

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.

The lab demonstrates how ISA 5 facilitates team team-based collaboration, and provides server-level diagnostic tools to carry out analysis. It also describes the data needed to debug the issues, and introduces the Java problem determination tools available as part of the IBM Support Assistant including Health Center, and Memory Analyzer.

The lab also uses the new health management self protecting and self healing features of WebSphere Application Server 8.5 to dynamically monitor and manage servers, helping to preserve service even if something is about to go wrong.

Lab requirements

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

- WebSphere Application Server V8.5.0.1
- IBM Support Assistant V5 (Beta 2) with the following tools installed:
 - ▶ WebSphere Application Server Configuration Visualizer
 - ▶ IBM Monitoring and Diagnostic Tools for Java™ - Health Center (ISA 5 desktop tool)
 - ▶ IBM Monitoring and Diagnostic Tools for Java™ - Memory Analyzer (ISA 5 desktop tool, web edition tool, and report tool)
 - ▶ IBM Extensions for Memory Analyzer (Packaged with the Memory Analyzer ISA 5 desktop tool)

What you should be able to do

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

- Launch 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
 - Define a WebSphere Application Server health policy to automatically restart a server if a memory leak is detected
 - Understand the basic techniques for debugging Java™ memory issues with Memory Analyzer, using desktop, web and report editions
 - Analyze a heap dump to determine those objects consuming the most heap space
 - Use IBM extensions for 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 : Impact2013

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 **55 minutes** and each part has an estimated duration. The final lab section is marked as **optional** so you can attempt this if there is sufficient time, or skip other sections according to your interests.

Part 2: IBM Support Assistant 5 Beta

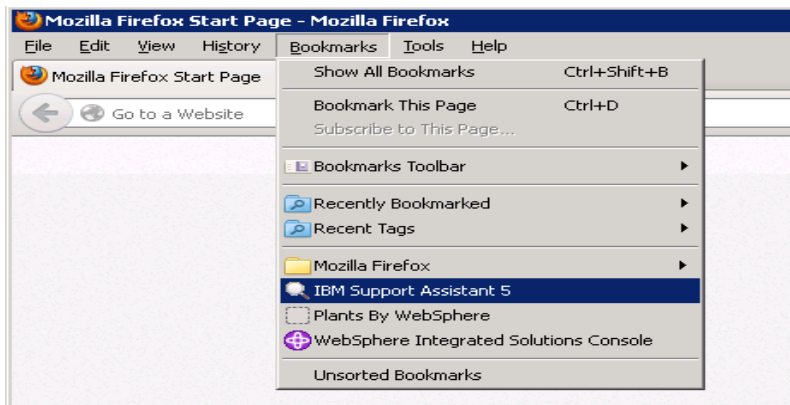
Note:

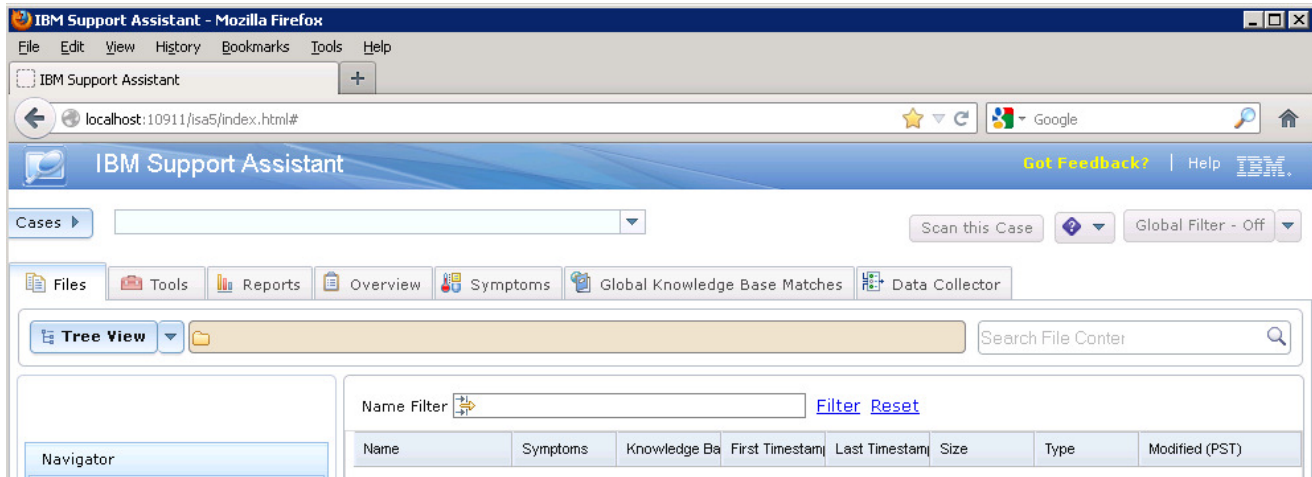
The IBM® Support Assistant (ISA) is a free application that provides features for problem determination, and a platform for obtaining diagnostic tools. The most recent release of ISA (version 5.0), which is currently in beta, brings these capabilities into a server environment. This enables an administrator to install a single instance of ISA that can be used by a group of users and accessed via a web browser. Therefore resources, files, information, and server-level tools can be shared. This facilitates team based collaboration and avoids the need for each team member to install diagnostic tools on their local workstation.

ISA v5.0 can be installed from an EAR file into an existing WebSphere Application Server, or using a simple “all-in-one” unzip install which contains everything required, including a lightweight application server and Java runtime. This lab uses the latter approach.

In this part of the lab, you will use ISA to understand its key concepts, and test some of the core functions.

_____ Launch the browser (Mozilla Firefox) and then use the bookmarks to load the ISA web interface as shown below.

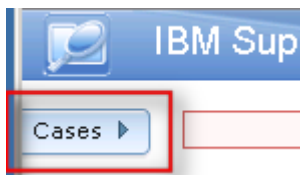


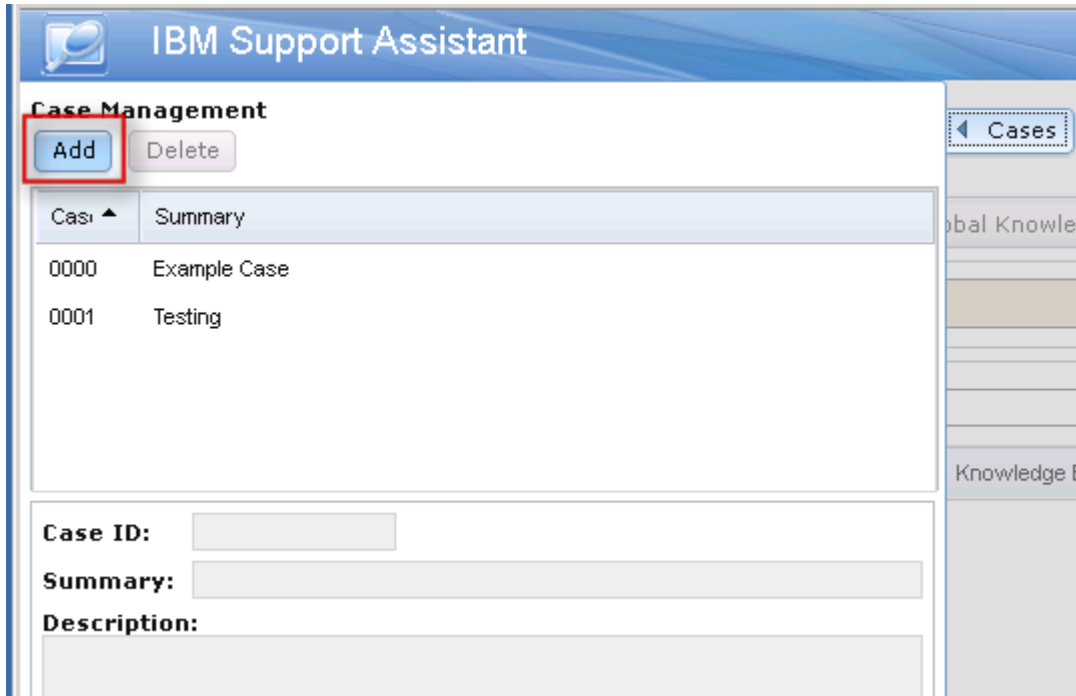


Note:

Like previous releases of ISA, the tool can be used to find problems, analyze data, and send information to IBM support. As it is a multi-user installation, ISA provides a case management component to help manage various problem determination activities that a team might require. A case is simply a container for a logical grouping of files and information. A typical practice would be to group artifacts pertaining to a single issue.

_____ Create a new case by clicking the **Cases** button, and then **Add**.

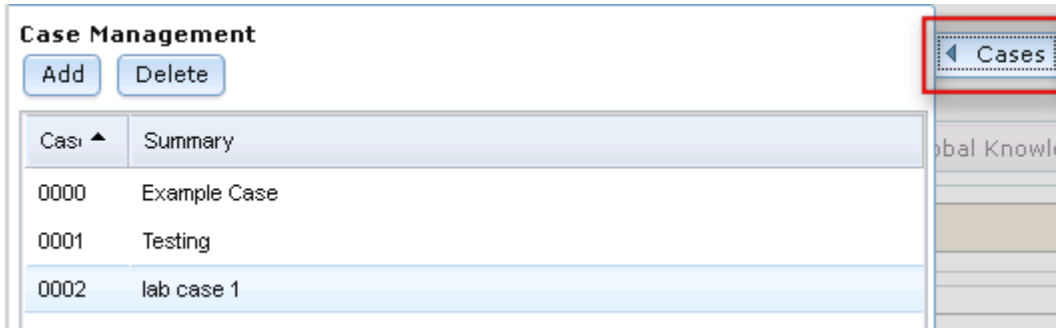




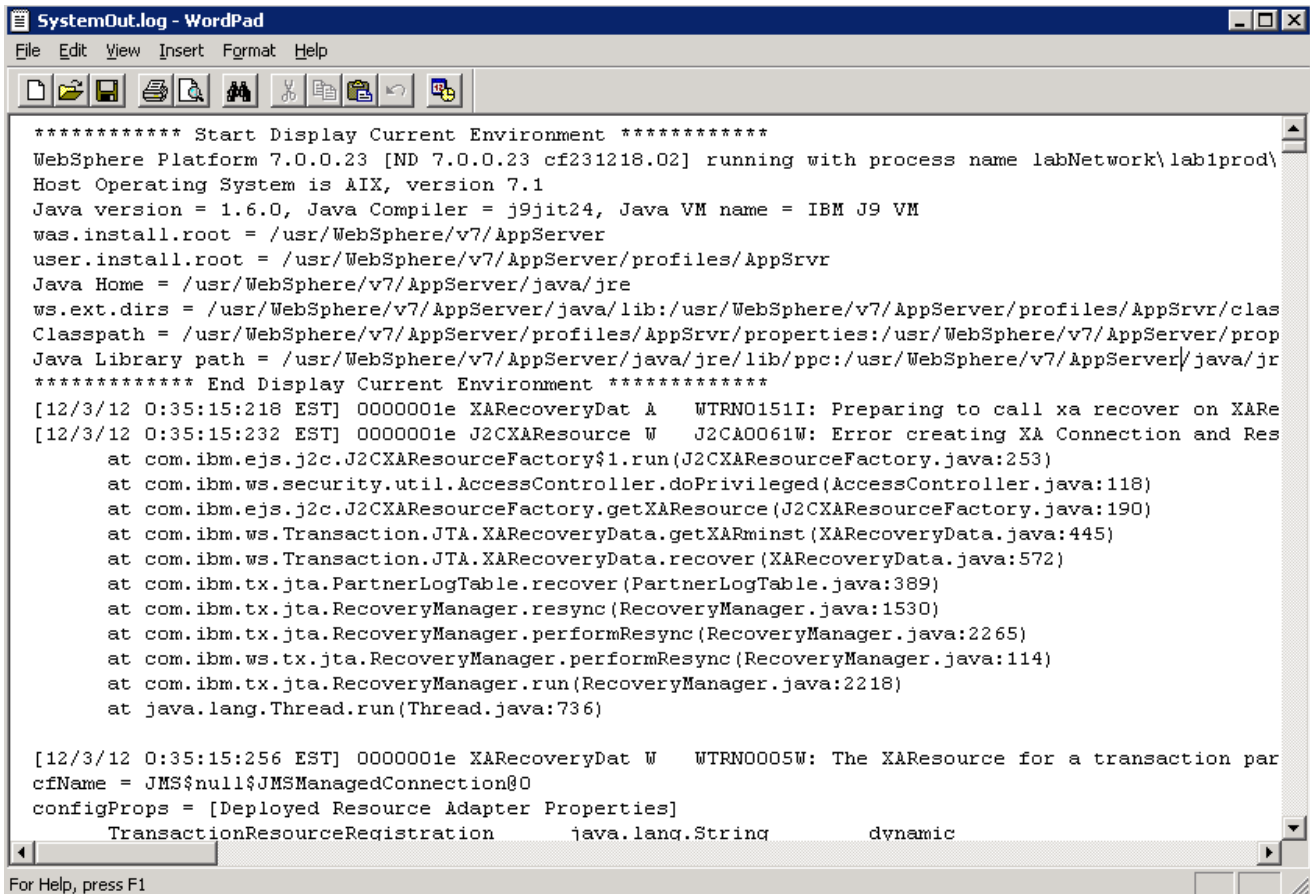
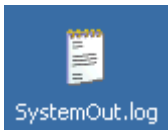
_____ Complete the summary and description, and click the **green tick as shown below**.



_____ Shrink the cases dialog by clicking **Cases**.

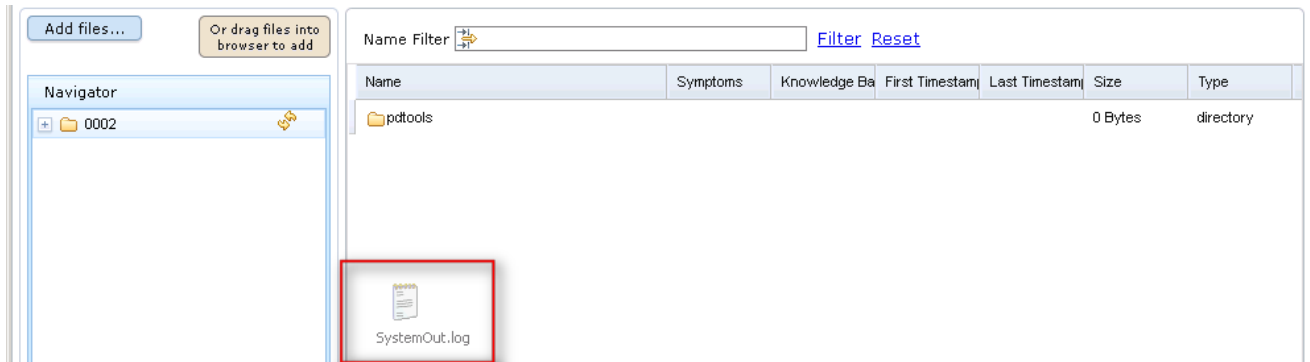


Having made a new case, diagnostic data can be added. This lab provides a sample WebSphere Application Server SystemOut log file as shown below on the Windows desktop. Open this file from the desktop, and note that it contains various WebSphere error messages and Java stack traces.

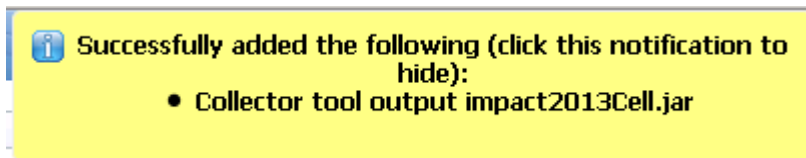


Close the text editor.

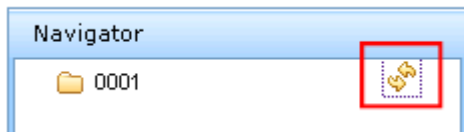
_____ Add this sample SystemOut log to the ISA case by dragging it from the desktop to the ISA file list in the browser.



_____ Click the **yellow box** to dismiss the notification.

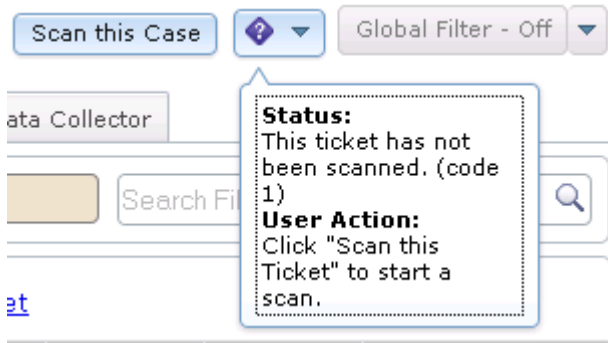


_____ Click the **refresh** icon in the Navigator to see the files in the case.

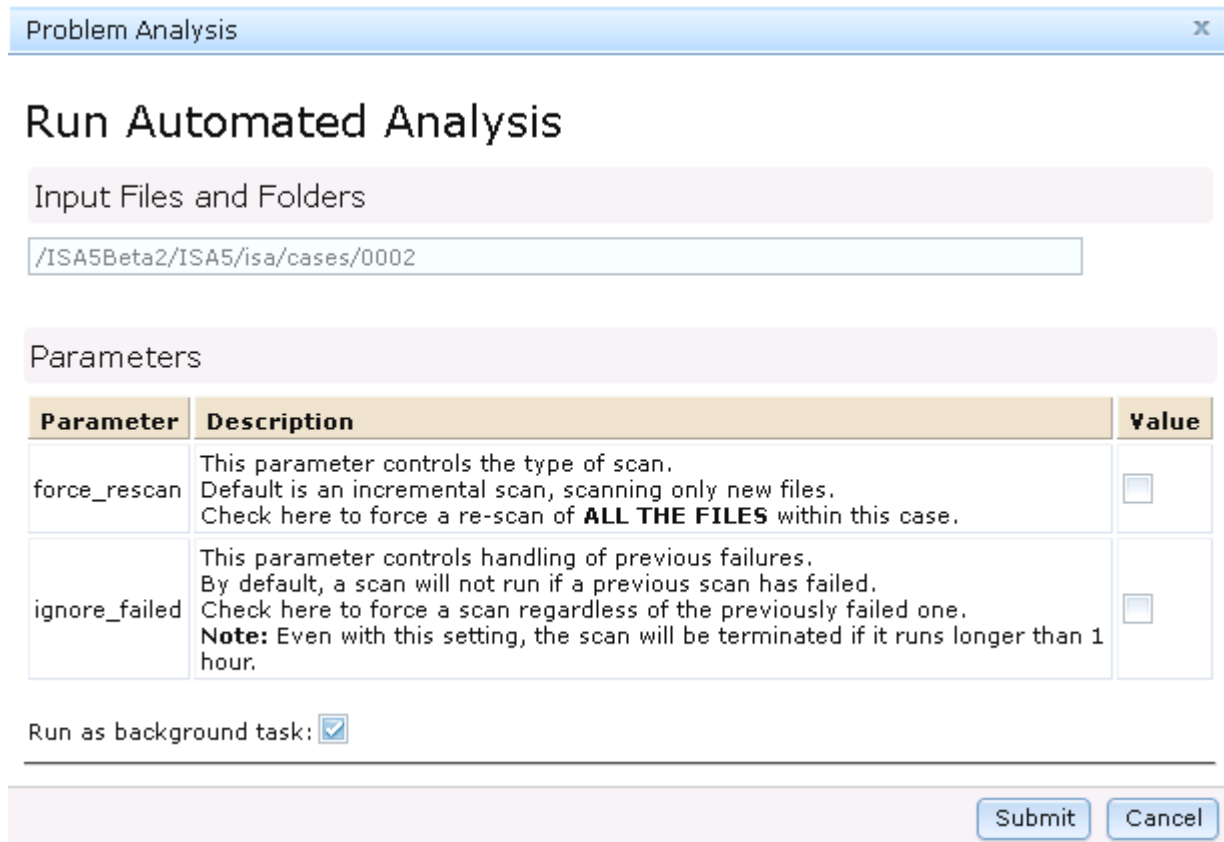


_____ Click the **Status** button. Notice the “Ticket” (case) has not been scanned.





_____ Click the **Scan this Case** button as shown above.



_____ Click **Submit** to start the Scan.

Note:

Scanning the case takes a few minutes. In the meantime, let's investigate the tools that are installed into ISA 5.

_____ Click the **Tools** tab. Note that only the tools that will be used in this lab have already installed, more tools are available.

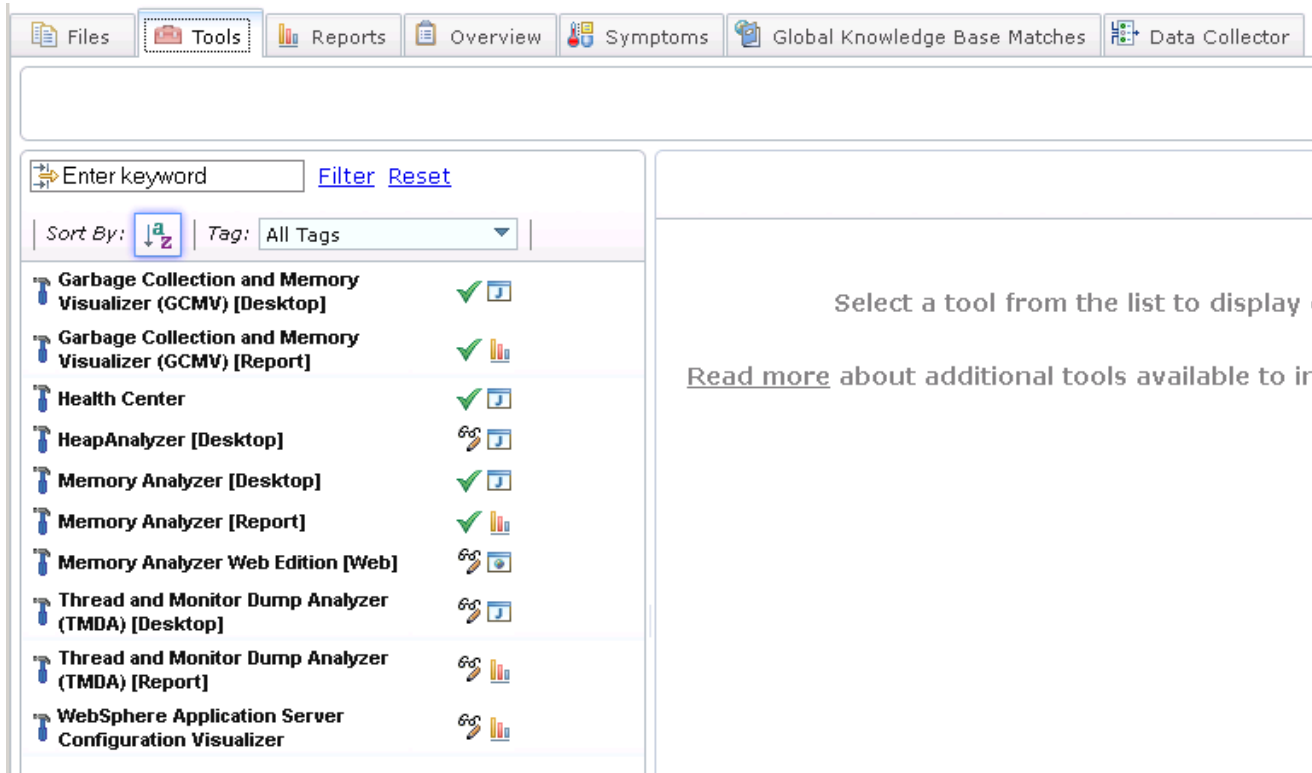
Note:

There are three key types of tools provided in ISA and each type has its benefits and compromises.

Report generator tools - process input data (e.g. log files or Java dumps) and generate a simple output file, usually in the form of an HTML or .txt report. These tools are not interactive but they are simple to run and have the benefit of consuming no local resources.

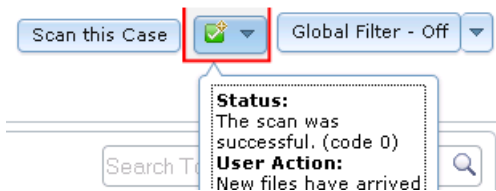
Web-based tools - run most of their analysis processing on the ISA server and provide a rich, interactive experience in the browser based user interface. These types of tools are ideal for activities where you want to off-load heavy processing of files to a more powerful server.

Desktop tools - typical desktop client-side applications that are launched via the ISA browser UI. By leveraging Java WebStart, the entire tool will be installed and run locally on your desktop. This type of tool has a few drawbacks - a Java plugin is required for your browser, local system resources are required to run the client tool, and you must have local access to the files you wish to analyze. However, some ISA tools are only available as desktop tools.



Note:

By now the scanning of the case should be finished, let's check the results. You can confirm this by clicking the **status** button as shown below.



Click the **Overview** tab. It shows useful system information determined from the log file(s) in the case.

Click the **Symptoms** tab. It shows a list of the errors encountered in the log file(s).

Click the **Global Knowledge Base** tab. This compares the symptoms to a local database (XML file) to suggest possible resolutions including APARs (IBM fix references) and Technotes. Click on a suggestion to see a detailed description below.

The screenshot displays the IBM Support Assistant 5.0 web interface in a Mozilla Firefox browser. The browser address bar shows the URL: localhost:10911/isa5/index.html#id=0002. The page title is "IBM Support Assistant".

The interface includes a navigation bar with tabs for "Files", "Tools", "Reports", "Overview", "Symptoms", "Global Knowledge Base Matches", and "Data Collector". The "Symptoms" tab is active, showing a search filter with the text "[0002] lab case 2".

The main content area displays a table of search results. The table has columns for "Global Score", "Type", "Knowledge Base Entry", "Symptom", "Tool", and "ID". There are 17 results shown out of 22 total.

Global Score	Type	Knowledge Base Entry	Symptom	Tool	ID
	APAR	PM15719: THE TRANSACTION MANAGER FAILS TO GET AN XARESOURCE TO ROLLBACK THE TRANSACTION.	Multiple symptoms (2) matched by this entry	LocalKBSe	9
	APAR	PK91826: CSCP0007E, WTRN0005W THE XARESOURCE FOR A TRANSACTION PARTICIPANT COULD NOT BE RECREATED.	WTRN0005W: The XAResource for a transaction participant could not be recreated and transaction recovery may not be able to complete properly. The resource was J2CXAResourceInfo :	LocalKBSe	16
	APAR	PK81814: MESSAGE "OPEN FOR E-BUSINESS" IS NOT BEING PRINTED IN SYSTEMOUT.LOG	CWZZZ0001W: Possible abnormal startup - did not find 'open for e-business'	LocalKBSe	1
	APAR	PK83560: WLM RETURNING TARGET WITH EMPTY ENDPOINTS CAUSES SIB EXCEPTIONS RESULTING IN CWSIT0019E OR CWSIA0241E ERRORS	CWZZZ0001W: Possible abnormal startup - did not find 'open for e-business'	LocalKBSe	3

Below the table, there are tabs for "Knowledge Base Matches", "Symptom Occurrences", "Symptom Details", and "Containing Files". The "Knowledge Base Matches" tab is selected, showing details for a specific match:

- Type:** APAR
- Found by Tool:** LocalKBSearch
- Global Score:** 1846
- Label:** PK83560: WLM RETURNING TARGET WITH EMPTY ENDPOINTS CAUSES SIB EXCEPTIONS RESULTING IN CWSIT0019E OR CWSIA0241E ERRORS
- Match ID:** 3
- Symptom IDs associated with this Match:** 85
- Description:**

A link is provided for more references: <http://www.ibm.com/Search?q=PK83560>

The **Abstract:** section contains the text: "WLM RETURNING TARGET WITH EMPTY ENDPOINTS CAUSES SIB EXCEPTIONS RESULTING IN CWSIT0019E OR CWSIA0241E ERRORS."

At the bottom of the page, the build ID is "5.0.0.0_Beta2_20121016-1409" and the copyright notice is "© Copyright IBM Corp. 2011, 2012. All rights reserved."

Part 3: Setup Health Center to Monitor a running WebSphere JVM

Note:

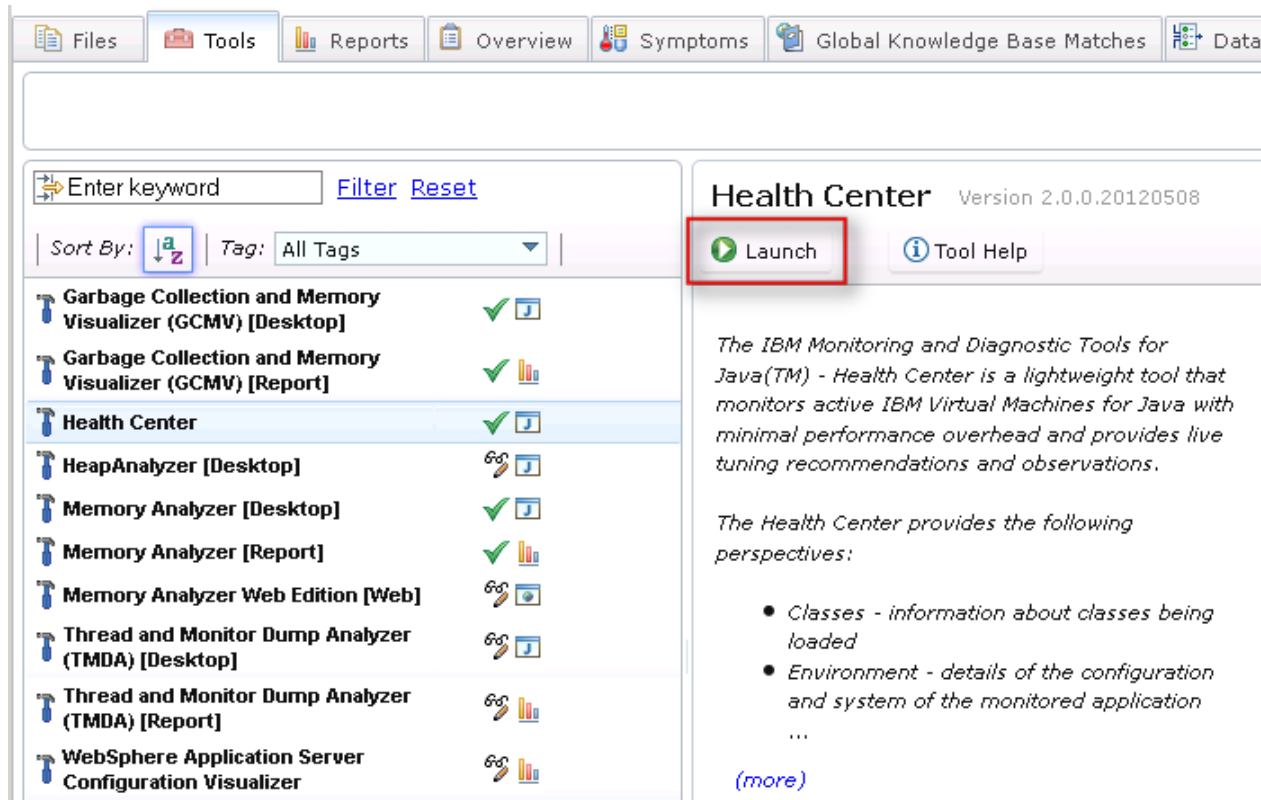
In this section the Health Center tool will be used to profile the code. Health Center is a free low-overhead diagnostic tool for monitoring applications running on an IBM Java Virtual Machines. It consists of a workstation client and a JVM agent. The agent uses a small amount of processor time (less than 3%) and memory on the server. It is installed by default in an IBM JVMs at Java 5 SR8 or IBM Java 6 SR1 and above.

The easiest way to attach the Health Center client to the agent is to configure the JVM to run the agent at start up with JVM “-Xhealthcenter” generic argument. In this lab, that argument has already been configured for both application server JVMs.

It is also possible to use a feature called “late attach” to connect the Health Center client to an already running JVM “on the fly”, without having pre-configured any command line properties. This lab does not demonstrate this technique, but more information can be found in the Health Center InfoCenter.

_____ If not launched already, launch the browser and use the bookmarks to load the ISA web interface.

_____ Switch to the Tools tab, select **Health Center** and click the **Launch** button.



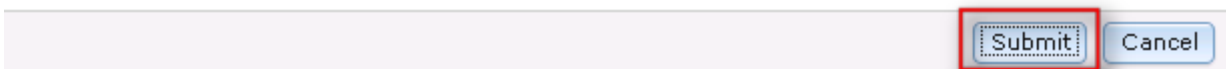
Problem Analysis x

Run Health Center (Version 2.0.0.20120508)

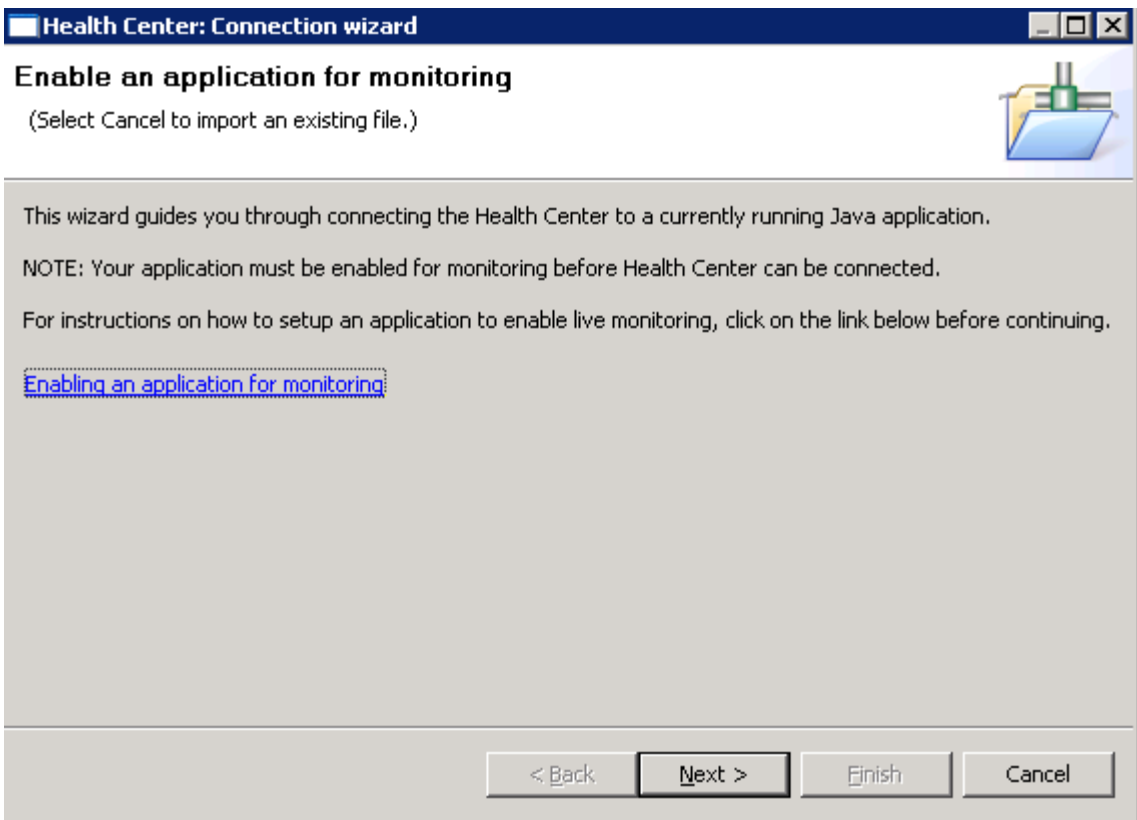
This tool is a desktop application. It will be launched using Java Web Start and will run on your workstation. Using the tool with files associated with ticket will require that you have access to the files from the workstation. If a file is located on a remote server, you can download the file to a local file system location or access the file through a shared storage area. Any existing local file may also be accessed by the tool.

In some cases, analysis of files on your workstation can noticeably degrade performance of other applications running on your workstation.

Click 'Submit' below to begin.



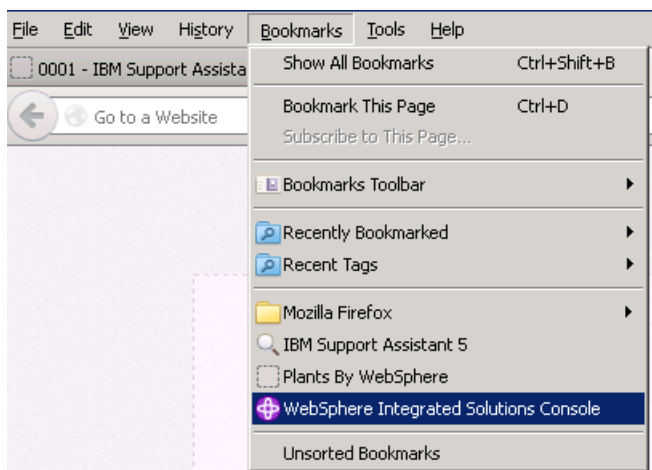
Click **Submit**. The Health Center client will be downloaded and launched using Java Web Start – this will take a few minutes, then a new connection dialogue will appear. In the meantime, continue with the next steps.



Note:

In this lab, the Plants by WebSphere application will be accessed via the IBM HTTP Server. It is configured to load balance requests between the application servers, and maintain session affinity once a HTTP session has been established. Multiple instances of Health Center can be started to monitor multiple JVMs if necessary. However, for now, to keep things simple only one JVM will be monitored, and the other JVM will be stopped.

_____ Launch the browser and use the bookmarks to load the WebSphere Integrated Solutions Console (admin console) in a new tab.



_____ Login using a blank user name.

_____ Navigate to **Servers->Server Types →WebSphere Application Servers**.



_____ Select **server1**.

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

New... Delete Templates... **Start** Stop Restart ImmediateStop Terminate

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input checked="" type="checkbox"/>	server1	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	✘
<input type="checkbox"/>	server2	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	✘

Total 2

_____ Press the Start button and wait for the admin console to report the server has started.

Application servers

Messages

Server impact2013Node/server1 started successfully. The collection may need to be refreshed to show the current server status. [View JVM logs](#) for further details.

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

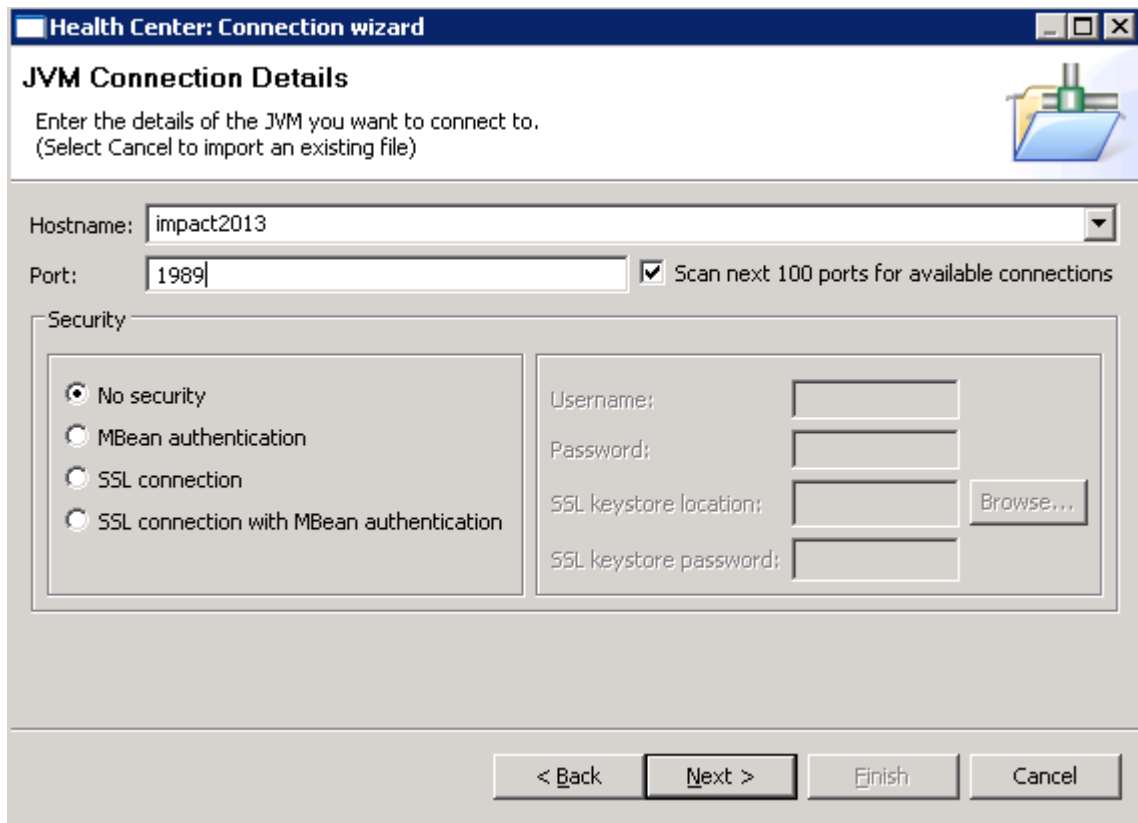
Preferences

New... Delete Templates... Start Stop Restart ImmediateStop Terminate

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input type="checkbox"/>	server1	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	✔
<input type="checkbox"/>	server2	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	✘

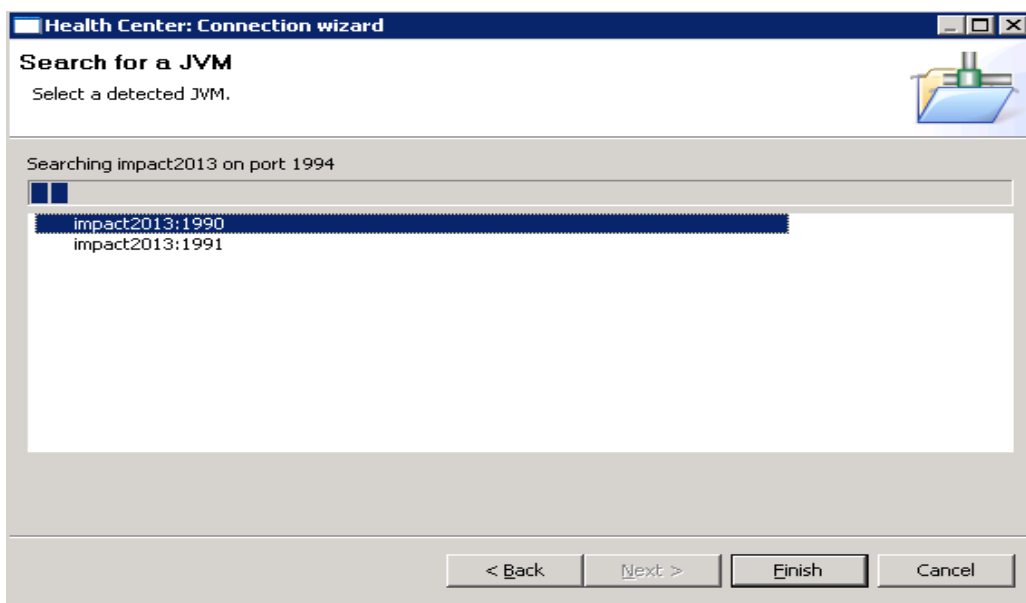
Total 2

_____ Return to the Health Center client window. Click **Next** on the connection wizard.

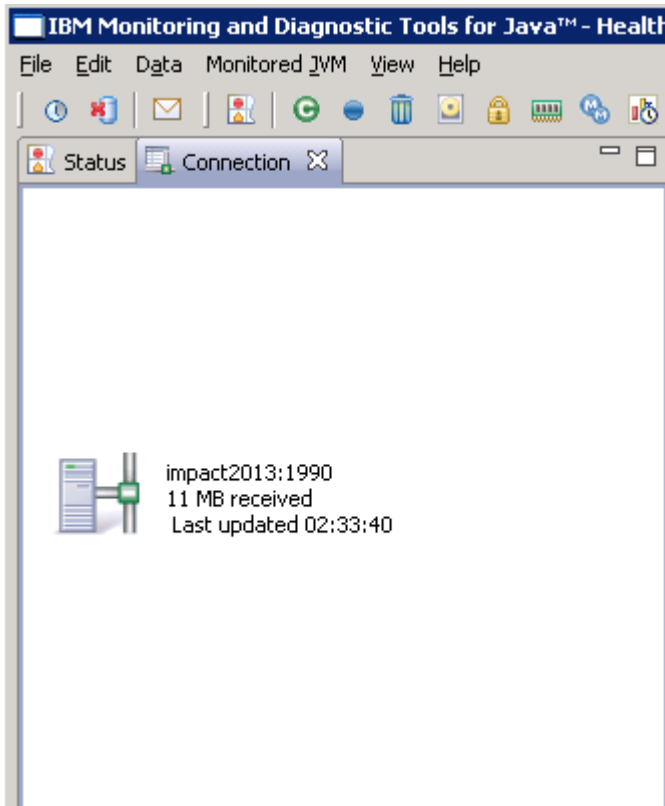


_____ The JVMs in this lab have been configured to run the Health Center agent on ports 1990 (server 1) and 1991 (server 2). Change the port to 1990 and Click **Next** to scan for these ports.

_____ Highlight the Health Center agent on port 1990 as soon as it appears and click **Finish** as shown below.



_____ Switch to the connection tab and verify the connection status shows an active connection as shown below.

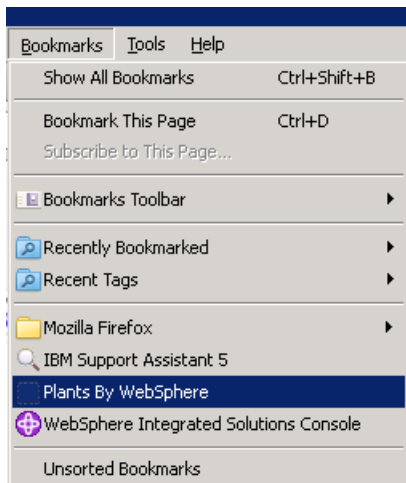


Part 4: Use Health Center to Investigate Application Errors

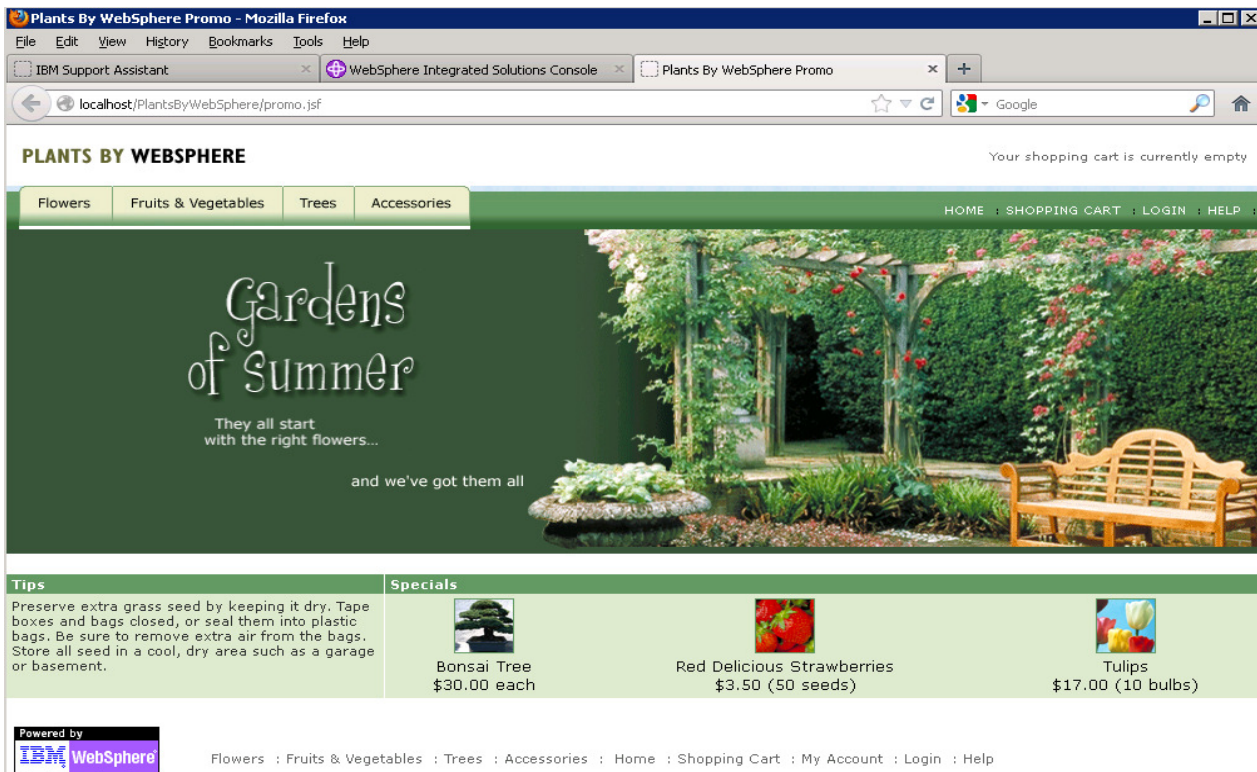
Note:

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. Launch the browser and use the bookmarks to visit the Plants by WebSphere site in a new tab.



_____ Feel free to have a look around, but to avoid some deliberate mistakes **do not** click any products on the **Accessories** tab.



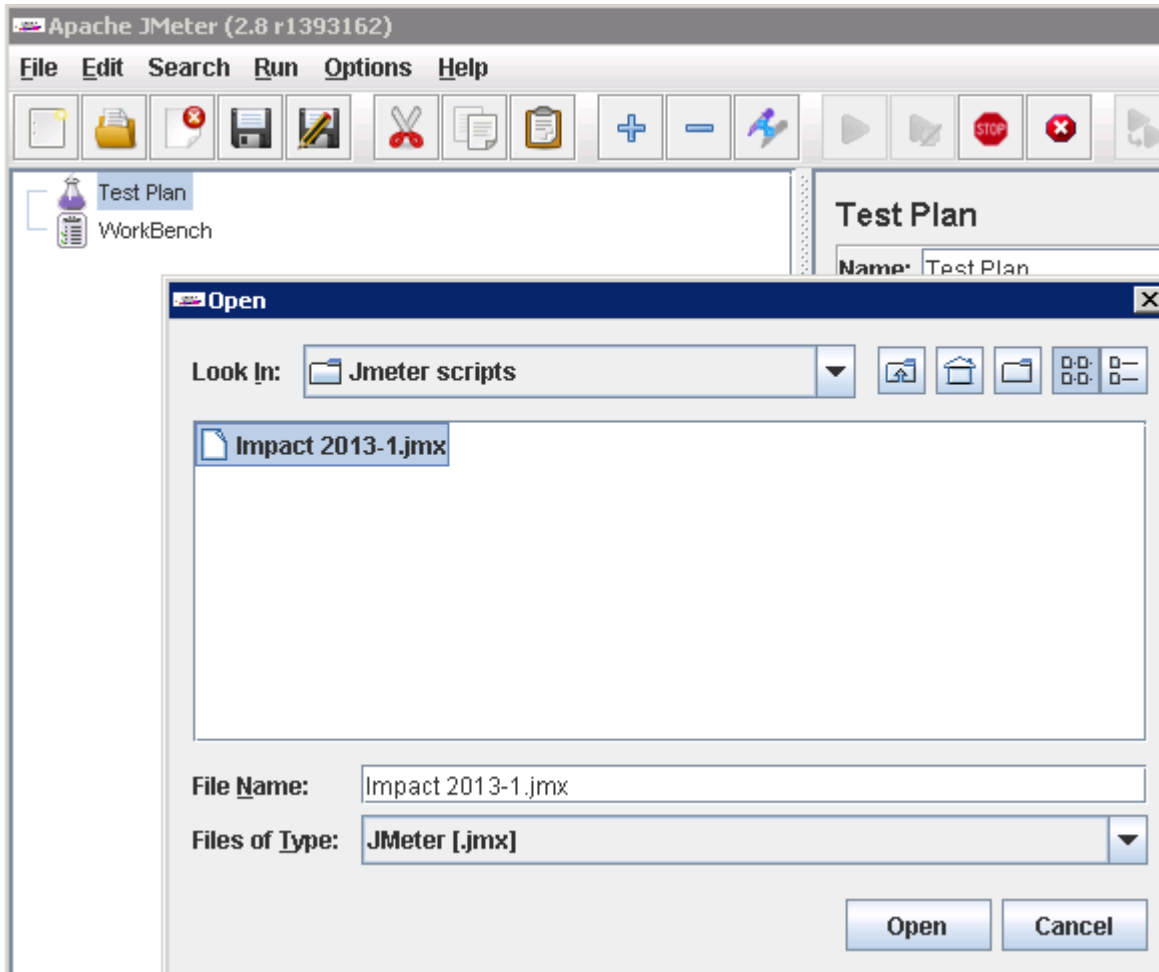
Note:

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.

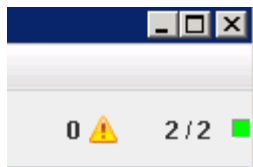
_____ Double click the **Jmeter** shortcut on the desktop.



_____ Click **File->Open** and navigate to **“E:\Impact lab files\Jmeter scripts\Impact 2013-1.jmx”**. Click **Open**.



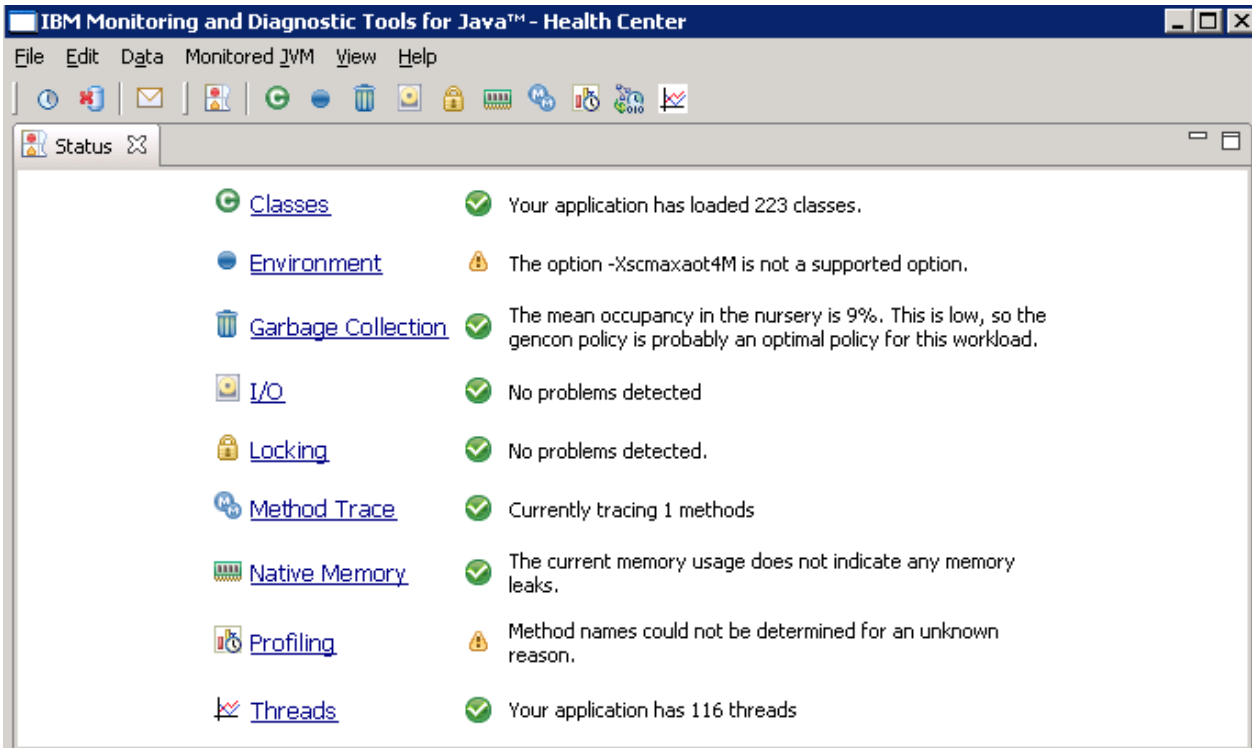
_____ Click **Run->Start**. Wait a few moments until the number of threads has reached 2, as indicated in the right hand corner of the Jmeter window.



_____ Switch to the Health Center window which should already be monitoring the WebSphere JVM (if not make a new connection with File->New Connection). Click the **Status** tab.

Note:

The Health Center status panel 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 also be customized from the Monitored JVM menu.



_____ Start by analyzing where the WebSphere JVM is spending most of its time to see if any optimizations can be made. Click the **Profiling** link.

Note:

Health Center uses a “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.

Note:

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.

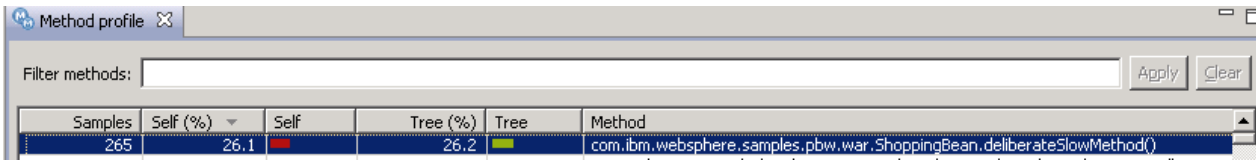
Samples	Self (%)	Self	Tree (%)	Tree	Method
2	0.59		66.7		com.ibm.ws.util.ThreadPool\$Worker.run()
0	0.0		56.9		com.ibm.ws.util.ThreadPool.getTask(boolean)
9	2.65		55.8		com.ibm.ws.util.BoundedBuffer.take()
10	2.95		53.4		com.ibm.ws.util.BoundedBuffer.waitGet_(long)
1	0.29		47.8		com.ibm.ws.util.BoundedBuffer\$GetQueueLock.await(long)
105	31.0	█	47.5		java.util.concurrent.locks.AbstractQueuedSynchronizer\$ConditionObject
0	0.0		18.9	█	java.lang.Thread.run()
23	6.78		15.9	█	com.ibm.ejs.util.am.AlarmManagerThreadCSLM.run()
22	6.49		6.49	█	java.util.concurrent.locks.LockSupport.parkNanos(java.lang.Object, long)
0	0.0		5.9	█	com.ibm.io.async.ResultHandler\$2.run()
6	1.77		5.9	█	com.ibm.io.async.ResultHandler.runEventProcessingLoop(boolean)
0	0.0		5.6	█	com.ibm.ws.dcs.vri.common.JobsProcessorThread.run()
0	0.0		5.6	█	com.ibm.ws.dcs.vri.common.JobsProcessorThread.executeJob(com.ibm.v
8	2.36		5.01	█	java.util.concurrent.locks.AbstractQueuedSynchronizer.acquireQueued(j
0	0.0		4.42	█	com.ibm.ws.util.ThreadPool.execute(java.lang.Runnable)

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

Note:

Now you can see the individual method “deliberateSlowMethod” in the **ShoppingBean** class is using a high percentage of the JVM’s time. Note, the “Self” and “Tree” columns (without the % symbol) are a graphical indication of the same data.

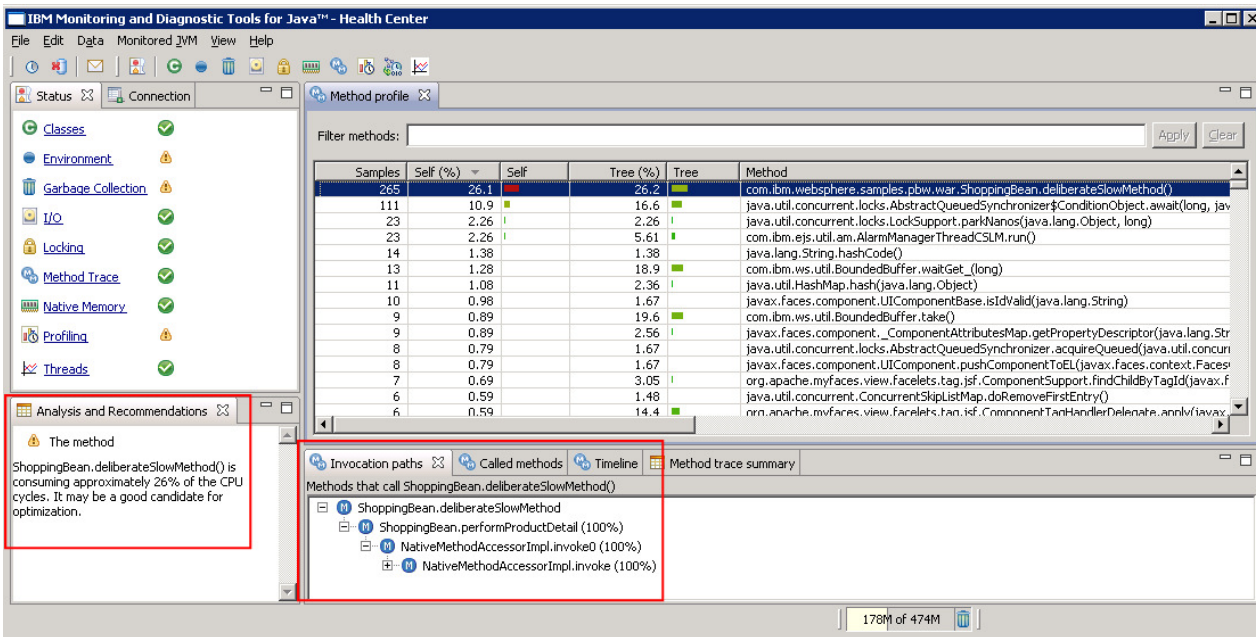
_____ Select the expensive method in the table by clicking it once.



Note:

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 time consuming 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**.

Note:

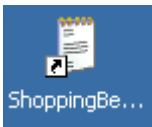
You can see only the “ShoppingBean.deliberateSlowMethod” in the Plants sample has a high value for the “Self (%)”.

Samples	Self (%)	Self	Tree (%)	Tree	Method
2623	50.9		50.9		com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateSlowMethod()
8	0.16		0.41		com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateLargeObjectAllocation()
0	0.0		0.56		com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$javassist_1.performShopping()
0	0.0		0.33		com.ibm.websphere.samples.pbw.ejb.CatalogMgr.getItemInventory(java.lang.String)
0	0.0		0.54		com.ibm.websphere.samples.pbw.ejb.EJSLocalNSFShoppingCartBean_f66be8da.getSize()
0	0.0		0.058		com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateSystemGCC()
0	0.0		0.019		com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateLargeSession(javax.face
0	0.0		51.8		com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$javassist_1.performProductDe
0	0.0		0.64		com.ibm.websphere.samples.pbw.ejb.ShoppingCartBean_\$\$javassist_2.getSize()
0	0.0		0.56		com.ibm.websphere.samples.pbw.war.ShoppingBean.performShopping()
0	0.0		0.33		com.ibm.websphere.samples.pbw.ejb.EJSLocalNSLCatalogMgr_c8d70688.getItemInvent
0	0.0		0.52		com.ibm.websphere.samples.pbw.ejb.CatalogMgr.getItemsByCategory(int)
0	0.0		51.8		com.ibm.websphere.samples.pbw.war.ShoppingBean.performProductDetail()
0	0.0		0.35		com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$javassist_1.getProducts()
0	0.0		0.078		com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$javassist_1.netCart()

_____ Press the Clear button.

Optional Steps:

_____ Double click the desktop shortcut as shown below to **ShoppingBean.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.

Note:

The “deliberateSlowMethod” is invoked every time the user clicks on the tulips. The “deliberateSlowMethod” executes a tight loop which does not end until a 10 second wait time has passed. You have found the first deliberate mistake.

```

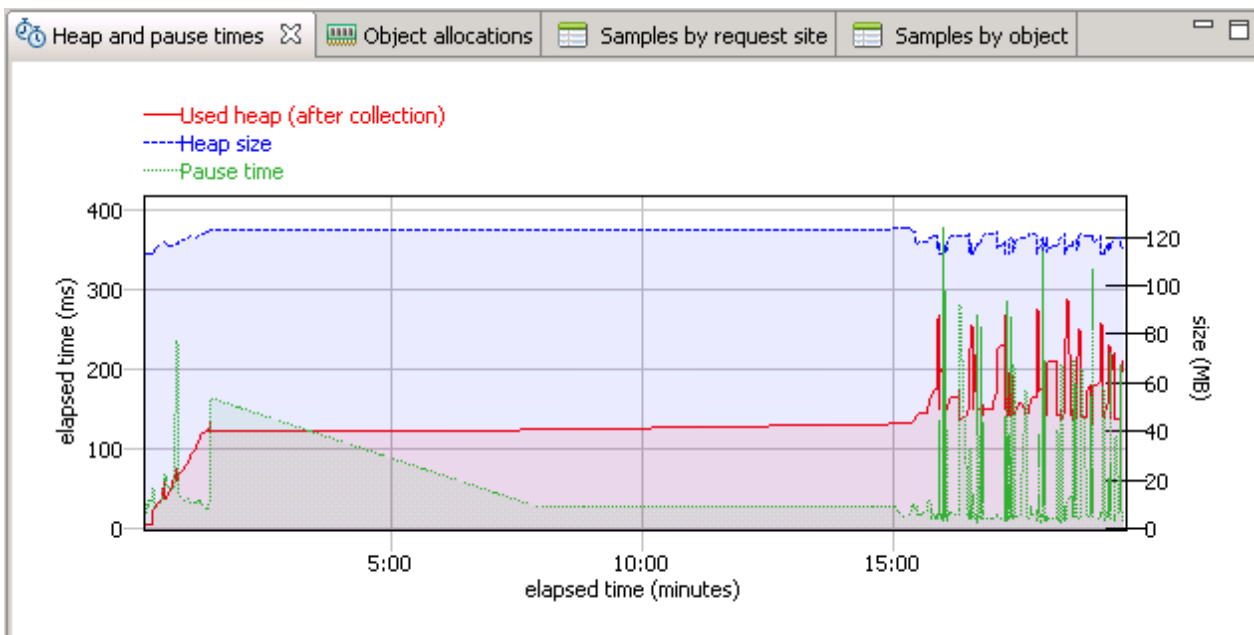
System.out.println("==> STARTING SLOW METHOD");

long timestamp = System.currentTimeMillis();
long target = timestamp + 10000;

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 select the **Garbage Collection** link to monitor the performance of garbage collection and memory usage

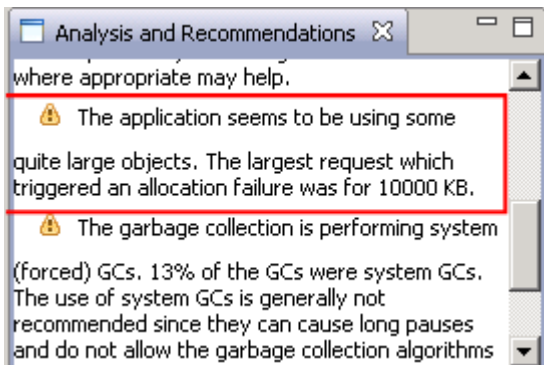


_____ Observe the current JVM heap size and used heap size after collection. After starting the load generator, you will notice the heap size and heap usage has increased but by now should have leveled out. There is currently no evidence of a memory leak.

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.

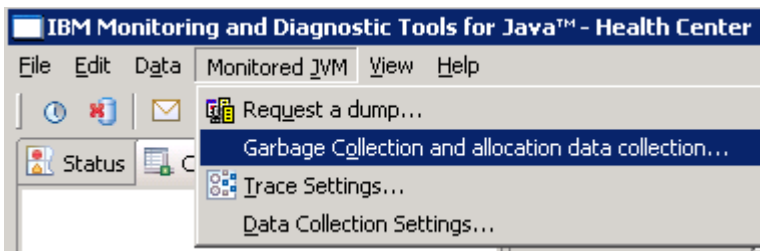
_____ Observe the Analysis and Recommendations window. It warns of large object allocations which of course are likely to trigger frequent garbage collections and may indicate the application code can be optimized.



Note:

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

_____ To investigate this further, select **Monitored JVM->Garbage Collection** and allocation data collection



GC Data Collection Settings

Configure Collection of GC and Allocation Data

Use these settings to control collection of verbose GC data and data for the analysis of object allocations requests

Write verbose GC data to file

Enable collection of call stacks for sampled object allocation events

Enable collection of object allocation events within thresholds:

Low threshold (bytes) High threshold (bytes)

Maximum number of stack entries to collect per event

_____ Select the box **Enable collection of object allocation events within thresholds**, and 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.

Enable collection of object allocation events within thresholds

Low threshold (bytes) High threshold (bytes)

_____ Click **Finish**.

_____ Select the **Object Allocations** tab. Wait a few moments until large object allocation data is parsed by the Health Center client – **this could take up to 1 minute**.

Count	%	Allocations	Average size (KB)	Request site
64	20.0		5000	java.lang.String.<init> (String.java:311)
64	20.0		5000	java.lang.StringCoding.decode (StringCoding.java:505)
64	20.0		2500	com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateLargeObjectAllocation (ShoppingBean.java:346)
127	39.7		7520	java.lang.StringBuilder.ensureCapacityImpl (StringBuilder.java:342)

Note:

Some large object allocations met the defined size threshold. They are associated with creating a very large String.

_____ Select the **Call Hierarchy** tab, then the **java.lang.StringBuilder** row in the table to see the stack trace leading to this large String allocation.

```

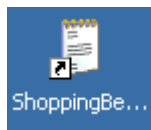
M java.lang.StringBuilder.ensureCapacityImpl (StringBuilder.java:342)
├── M java.lang.StringBuilder.append (StringBuilder.java:208) (100%)
│   ├── M java.lang.StringBuilder.append (StringBuilder.java:183) (50.0%)
│   │   ├── M com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateLargeObjectAllocation (ShoppingBean.java:346) (100%)
│   │   │   └── M com.ibm.websphere.samples.pbw.war.ShoppingBean.performProductDetail (ShoppingBean.java:161) (100%)
│   └── M com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateLargeObjectAllocation (ShoppingBean.java:346) (50.0%)
    
```

Note:

Once again you will see the ShoppingBean 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 **ShoppingBean.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.

Note:

The “deliberateLargeObjectAllocation” is invoked 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 **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.

Summary	
Minor collections - Mean garbage collection pause	35.6 ms
Minor collections - Mean interval between collections	2325 ms
Minor collections - Number of collections	457
Minor collections - Total amount flipped	1884759 KB
Number of collections triggered by allocation failure	463
Proportion of time spent in Garbage Collection pauses	3.93%
Proportion of time spent unpaused	96.1%
System (forced) garbage collection count	84

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.

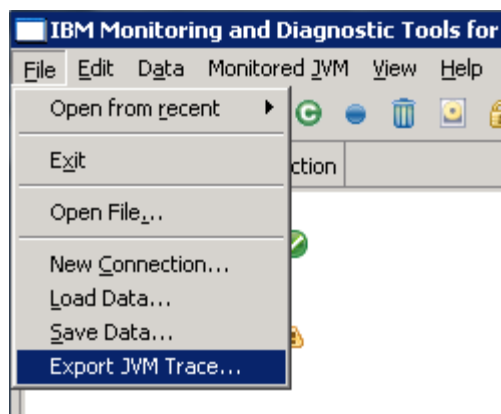
Health Center can be used to filter method innovations (for example, looking for System.gc). However, the method profiling works on a sampling basis, so infrequent method calls might not appear in Health Center's profiling results. Therefore, the best way to achieve this goal is to set a JVM trace, and use Health Center to extract this information from the JVM.

The JVM has been pre-configured with the following command line property:

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

This causes the JVM to print a stack trace each time the `java/lang/System.gc` is executed. Normally, this output goes to the `native_stderr.log`. However, if the Health Center agent is also configured on the JVM (with `-Xhealthcenter`, like in this lab), then the output is instead directed to the Health Center Client.

_____ In the Health Center window, click **File->Export JVM Trace**



_____ Export the trace to "E:\trace.trc"

_____ Open a command prompt, and issue command: `E: (` (to change to the E: drive).

_____ Then issue the following command:

```
java com.ibm.jvm.format.TraceFormat trace.trc
```

```

C:\> Command Prompt
Error: j9jit.4436 not in dat file: dat files old or from incorrect UM.
Error: j9jit.9715 not in dat file: dat files old or from incorrect UM.
Error: j9jit.14352 not in dat file: dat files old or from incorrect UM.
50% Error: j9jit.13584 not in dat file: dat files old or from incorrect UM.
Error: j9jit.5891 not in dat file: dat files old or from incorrect UM.
Error: j9jit.3344 not in dat file: dat files old or from incorrect UM.
Error: j9jit.11024 not in dat file: dat files old or from incorrect UM.
Error: j9jit.516 not in dat file: dat files old or from incorrect UM.
Error: IO.10593 not in dat file: dat files old or from incorrect UM.
Error: IO.13915 not in dat file: dat files old or from incorrect UM.
Error: IO.11115 not in dat file: dat files old or from incorrect UM.
Error: IO.7010 not in dat file: dat files old or from incorrect UM.
Error: j9jit.372 not in dat file: dat files old or from incorrect UM.
Error: IO.8306 not in dat file: dat files old or from incorrect UM.
Error: j9jit.12587 not in dat file: dat files old or from incorrect UM.
Error: j9jit.2052 not in dat file: dat files old or from incorrect UM.
60% Error: IO.-1 not in dat file: dat files old or from incorrect UM.
70% Error: IO.11108 not in dat file: dat files old or from incorrect UM.
80% 90% 100%
*** Number of formatted tracepoints = 505075
*** 1 buffers were discarded during trace data generation
*** Formatting complete
*** Formatted output written to file: trace.trc.fmt
C:\temp>
    
```

Note: This may take a few minutes.

_____ Open the resulting trace.trc.fmt in an editor such as Notepad++

_____ Search for java.lang.System.gc

```

E:\trace.trc.fmt - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugin
Find... Ctrl+F
Find in Files... Ctrl+Shift+F
System Find Next F3 pingBean.java

>java/lang/System.gc()V Bytecode static method
jstacktrace:
[1] java.lang.System.gc (System.java:312)
[2] com.ibm.websphere.samples.pbw.war.ShoppingBean.deliberateSystemGC (ShoppingBean.jav
[3] com.ibm.websphere.samples.pbw.war.ShoppingBean.performProductDetail (ShoppingBean.j
[4] sun.reflect.Na
[5] sun.reflect.Na
***** Context chan
SystemGC start: ne
Exclusive access:
GlobalGC start: gl
Mark start
Mark end
Class unloading st
Class unloading en
Sweep start
Sweep end
Compact start: rea
    
```

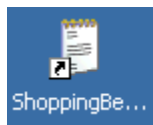
Note:

The formatted Java trace contains a stack trace for each time the `java.lang.System.gc` method was called.

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

Optional Steps:

_____ Double click the desktop shortcut for **ShoppingBean.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 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");
```

_____ Switch to the Jmeter load generator window. Stop the load test by clicking **File->Exit**.

Note:

In subsequent parts of this lab, you will manually trigger the final deliberate error in the plants sample, but first you will set up a WebSphere Application Server Health Policy to help minimize the business impact of the error.

Part 5: WebSphere Application Server Health Management

Note:

WebSphere Application Server 8.5 provides significant new features for administration and operations staff. This includes incorporating the Intelligent Management functions from IBM WebSphere Virtual Enterprise which can minimize end-user outages, and use operational monitoring to control the production environment. One such feature is health management which monitors the status of the application servers, as well as sense and respond to problem areas before an outage occurs.

The health of an application server can be managed with a policy-driven approach that enables specific actions to occur when monitored criteria are met. In this section of the lab, a **health policy** will be defined which is triggered when memory usage exceeds a percentage of the heap size for a specified time, and **health actions** will be defined to correct the situation.

_____ Launch the browser and use the bookmarks to load the WebSphere Integrated Solutions Console (admin console).

_____ Navigate to **Operational Policies->Autonomic Managers->Health Controller**

The screenshot shows the WebSphere Integrated Solutions Console interface. On the left is a navigation tree with 'Operational policies' expanded to 'Autonomic Managers' and 'Health Controller' selected. The main content area is titled 'Global Health Controller Parameters' and has two tabs: 'Configuration' and 'Runtime'. The 'Runtime' tab is active, showing the following settings:

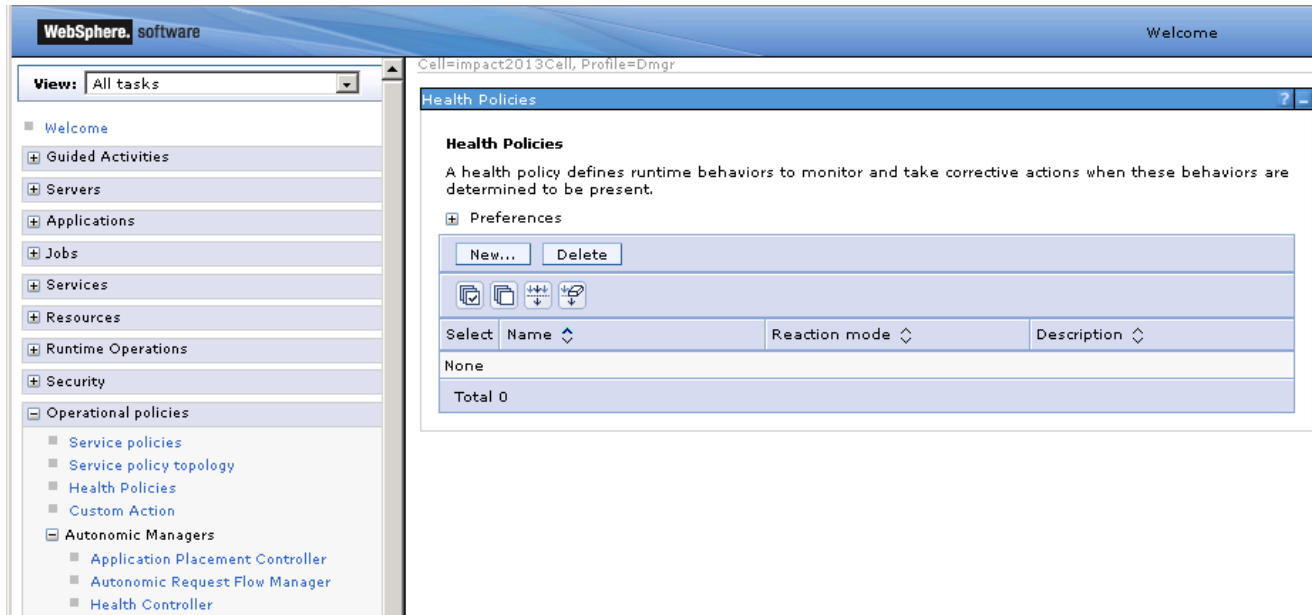
- Enable health monitoring
- Control cycle length: 1 Minutes
- Maximum consecutive restarts: 5
- Restart timeout: 1 Minutes
- Minimum restart interval: 0 Minutes
- Prohibited restart times: A table with columns for Start, End, and days of the week (Sun-Sat), all currently unchecked.

At the bottom of the configuration area are buttons for 'Apply', 'OK', 'Reset', and 'Cancel'.

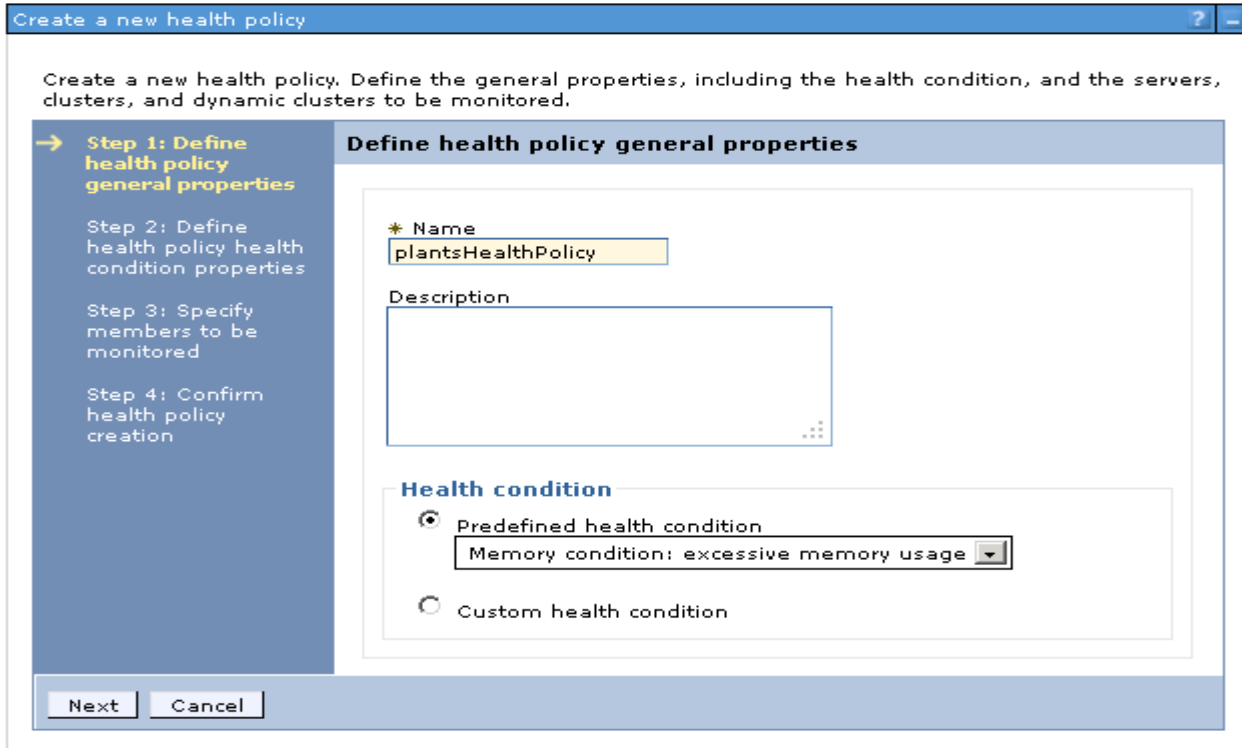
Note:

Note that health monitoring is already enabled. The Control cycle length specifies the time between consecutive health checks, it has been set to the lowest possible value for this lab.

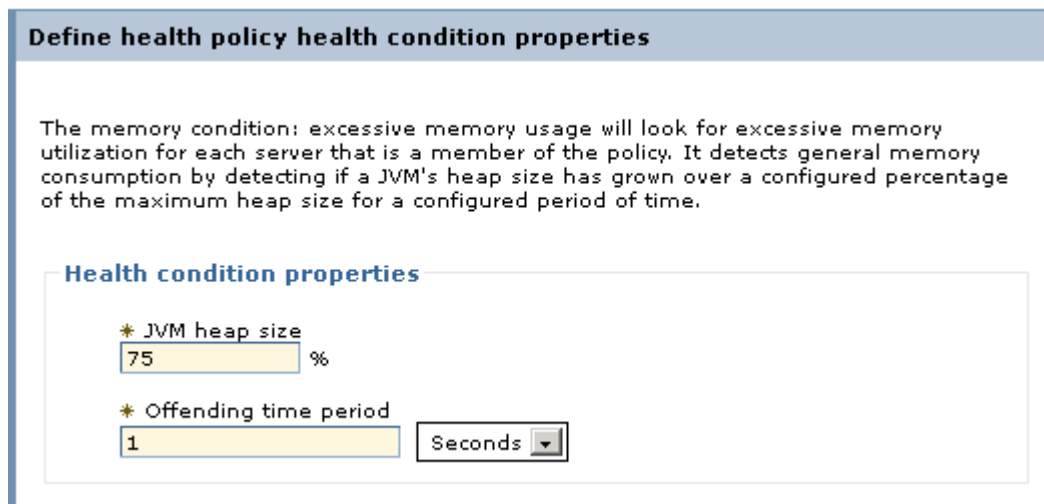
____ Navigate to **Operational Policies->Health Policies**.



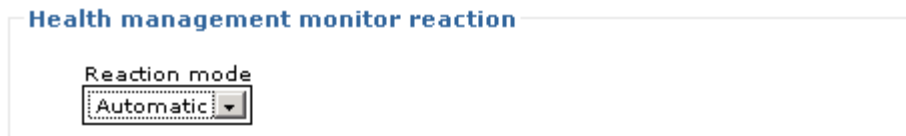
____ Click **New** and name the policy “plantsHealthPolicy”. Select the pre-defined health condition **Memory condition: excessive memory usage** as shown below and then click **Next**.



_____ First set the condition which should trigger the policy. Set the JVM heap size % to **75%**, and the offending time period to **1 second** (this is very aggressive, but we want the policy to be triggered quickly in this lab environment).

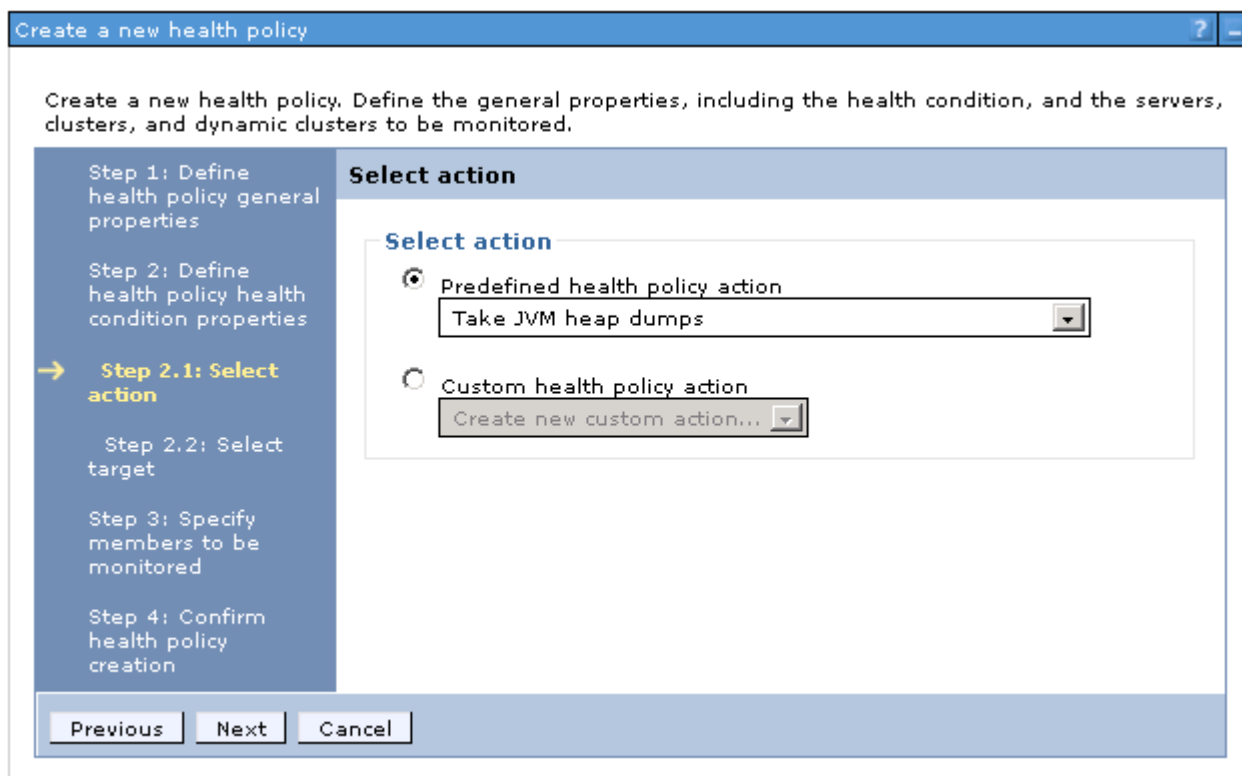


_____ Next specify the actions the Health Manager should take if the policy is triggered. Select a Reaction Mode of **Automatic**.



____ Press the Add Action button. Select the predefined action **Take JVM heap dumps**.

Take the following actions when the health condition breaches



____ Click **Next**

Create a new health policy
?

Create a new health policy. Define the general properties, including the health condition, and the servers, clusters, and dynamic clusters to be monitored.

Step 1: Define health policy general properties

→ **Step 2: Define health policy health condition properties**

Step 3: Specify members to be monitored

Step 4: Confirm health policy creation

Define health policy health condition properties

The memory condition: excessive memory usage will look for excessive memory utilization for each server that is a member of the policy. It detects general memory consumption by detecting if a JVM's heap size has grown over a configured percentage of the maximum heap size for a configured period of time.

Health condition properties

* JVM heap size
 %

* Offending time period
 Seconds

Health management monitor reaction

Reaction mode
Automatic

Take the following actions when the health condition breaches

Add Action...
Remove Action
Move Up
Move Down

Select	Step	Action	Target server	Target node
<input type="checkbox"/>	1	Restart server	Sick server	Node hosting sick server
<input type="checkbox"/>	2	Take JVM heap dumps	Sick server	Node hosting sick server

Previous
Next
Cancel

_____ To ensure heap dumps are triggered before the server is restarted, select **Take JVM heap dumps** and press the **Move Up** button.

Create a new health policy
?
-

Create a new health policy. Define the general properties, including the health condition, and the servers, clusters, and dynamic clusters to be monitored.

Step 1: Define health policy general properties

→ **Step 2: Define health policy health condition properties**

Step 3: Specify members to be monitored

Step 4: Confirm health policy creation

Define health policy health condition properties

The memory condition: excessive memory usage will look for excessive memory utilization for each server that is a member of the policy. It detects general memory consumption by detecting if a JVM's heap size has grown over a configured percentage of the maximum heap size for a configured period of time.

Health condition properties

* JVM heap size
 %

* Offending time period
 Seconds

Health management monitor reaction

Reaction mode
Supervise

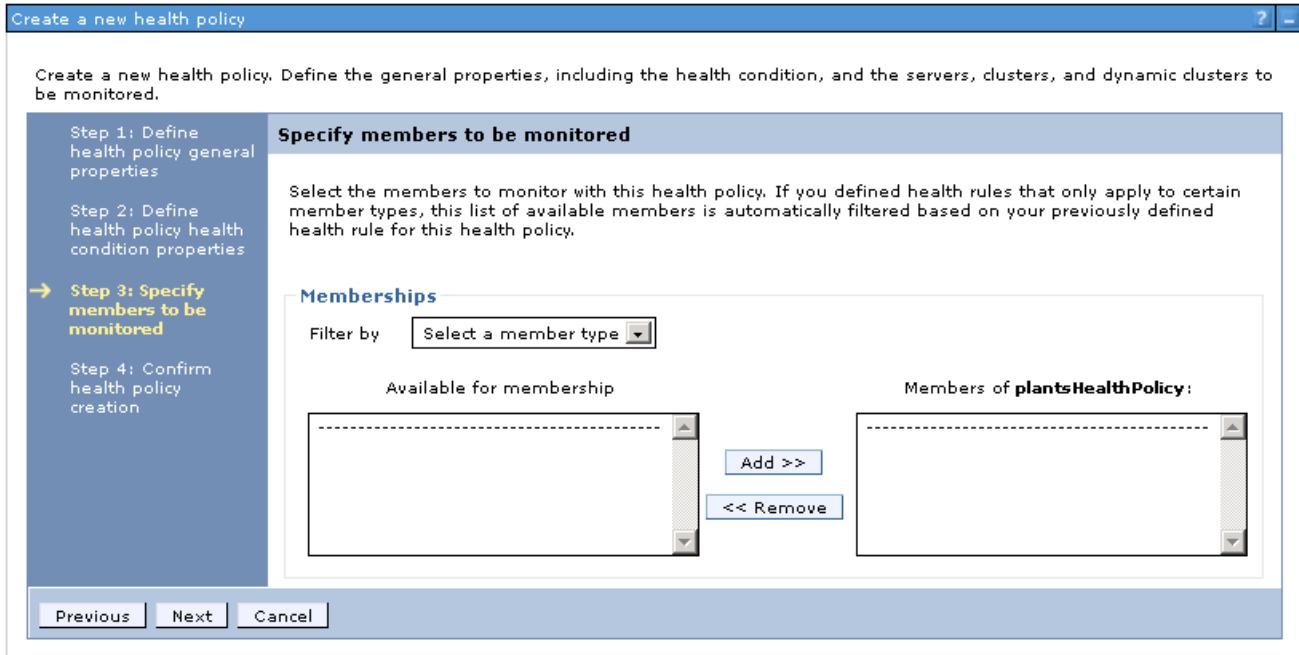
Take the following actions when the health condition breaches

Add Action...
Remove Action
Move Up
Move Down

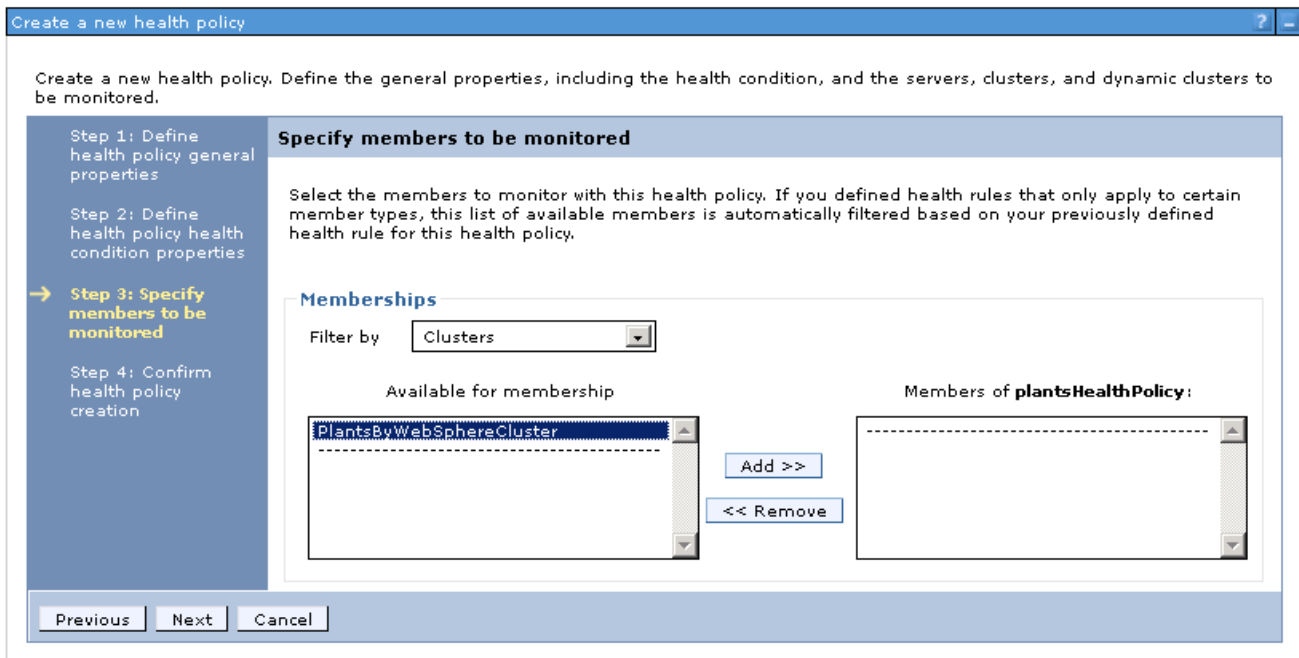
Select	Step	Action	Target server	Target node
<input type="checkbox"/>	1	Take JVM heap dumps	Sick server	Node hosting sick server
<input type="checkbox"/>	2	Restart server	Sick server	Node hosting sick server

Previous
Next
Cancel

_____ Click **Next**



_____ Select Filter by **Clusters**, and highlight **PlantsByWebSphereCluster**



_____ To add all members of the PlantsByWebSphereCluster to the health policy, click **Add**

Create a new health policy

Create a new health policy. Define the general properties, including the health condition, and the servers, clusters, and dynamic clusters to be monitored.

Step 1: Define health policy general properties

Step 2: Define health policy health condition properties

→ Step 3: Specify members to be monitored

Step 4: Confirm health policy creation

Specify members to be monitored

Select the members to monitor with this health policy. If you defined health rules that only apply to certain member types, this list of available members is automatically filtered based on your previously defined health rule for this health policy.

Memberships

Filter by: Clusters

Available for membership

Members of **plantsHealthPolicy**:

PlantsByWebSphereCluster (Clusters)

Add >>

<< Remove

Previous Next Cancel

_____ Click Next, then Finish

_____ Click the **Save** link to save the changes to the Deployment Manager master repository.

_____ Select **System Administration->Nodes**





Nodes




Nodes

Use this page to manage nodes in the application server environment. A node corresponds to a physical computer system with a distinct IP host address. The following table lists the managed and unmanaged nodes in this cell. The first node is the deployment manager. Add new nodes to the cell and to this list by clicking Add Node.

⊕ Preferences

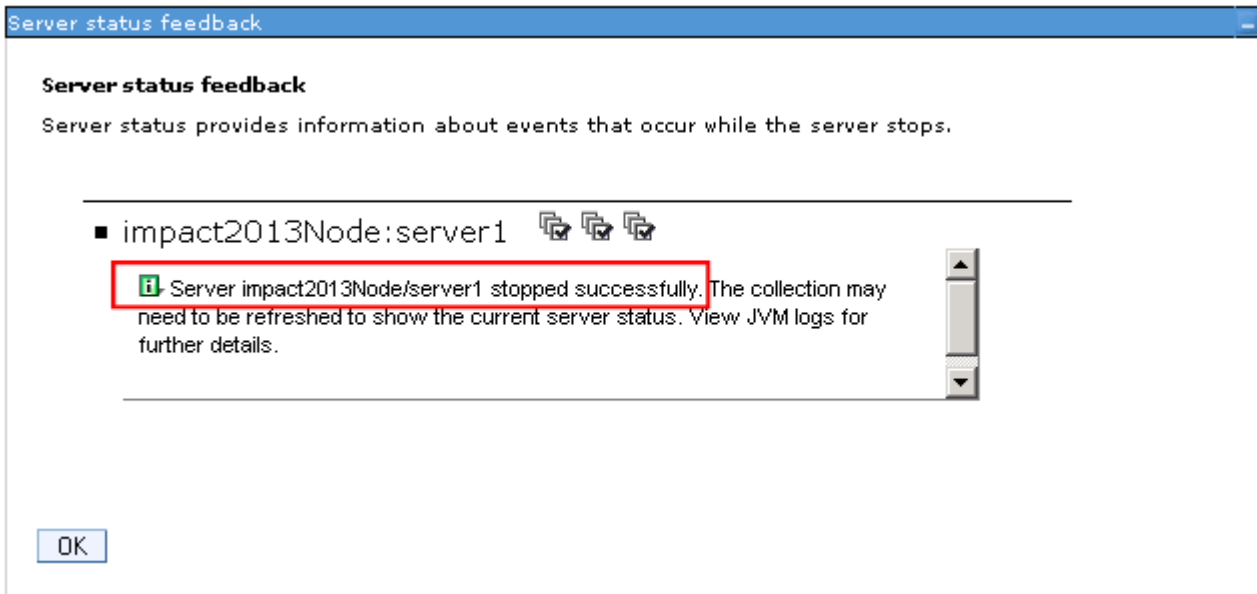
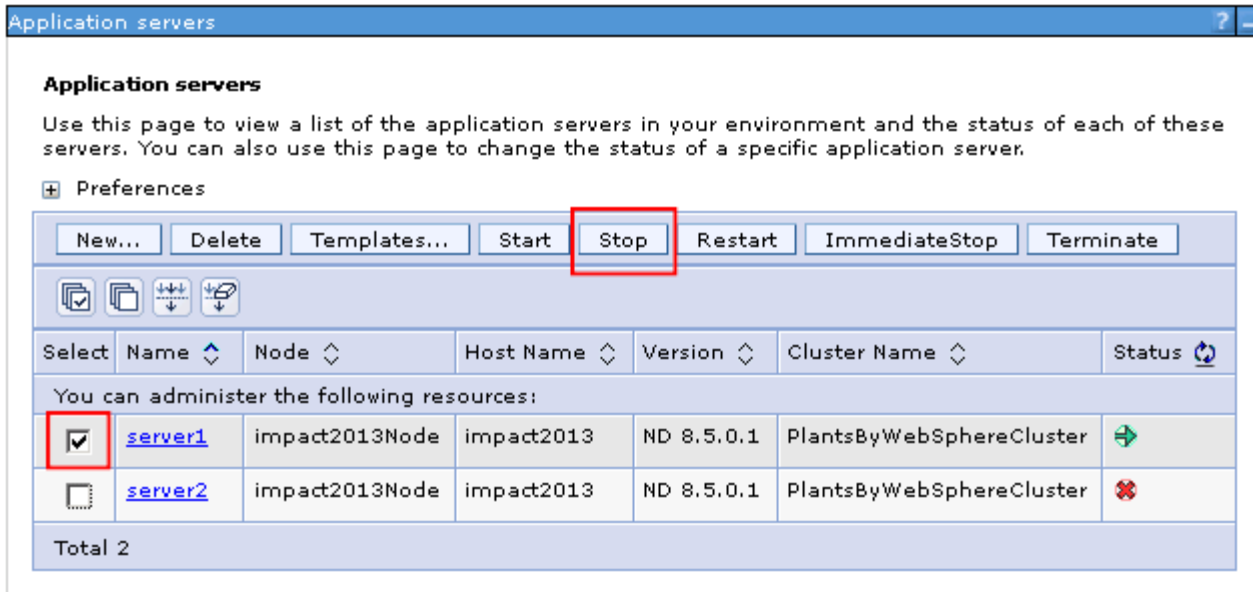
Add Node Remove Node Force Delete Synchronize Full Resynchronize Stop

Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	impact2013CellManager	impact2013	ND 8.5.0.1	TCP	
	impact2013Node	impact2013	ND 8.5.0.1	TCP	
Total 2					

_____ Select **impact2013Node**, and press **Synchronize**. (The Nodes may have already synchronized automatically).

_____ Navigate to **Servers->Server Types ->WebSphere application servers**. Select the running server and press the **Stop** button.



_____ Wait for the server to stop, then Click **OK**

Note:

In the next section of the lab, you will cause a memory leak which triggers the health policy and its associated actions. To demonstrate how diagnostic data (heap dumps) can be collected, without a significant impact to the application's users, all cluster members should be running.

_____ Select **Servers->Clusters->WebSphere Application Server Clusters**.

_____ Select **PlantsByWebSphereCluster**, and press Start. This starts both servers in parallel.

_____ Wait for the servers to start which will take a few minutes. You may read ahead, but you must wait for the servers to start before attempting any actions in the lab. You can check the status on the Servers->Server Types->Application Servers page, by pressing the refresh icon.

The screenshot shows the 'Application servers' management interface. At the top, there are buttons for 'New...', 'Delete', 'Templates...', 'Start', 'Stop', 'Restart', 'ImmediateStop', and 'Terminate'. Below these are icons for selection and refresh. A table lists the servers with columns for 'Select', 'Name', 'Node', 'Host Name', 'Version', 'Cluster Name', and 'Status'. Two servers, 'server1' and 'server2', are listed, both with a status of 'Running' (indicated by a green arrow). A 'Total 2' summary is shown at the bottom of the table.

Select	Name	Node	Host Name	Version	Cluster Name	Status
<input type="checkbox"/>	server1	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	Running
<input type="checkbox"/>	server2	impact2013Node	impact2013	ND 8.5.0.1	PlantsByWebSphereCluster	Running

Total 2

Part 6: Trigger a the Memory Leak to activate the Heath Policy

Note:

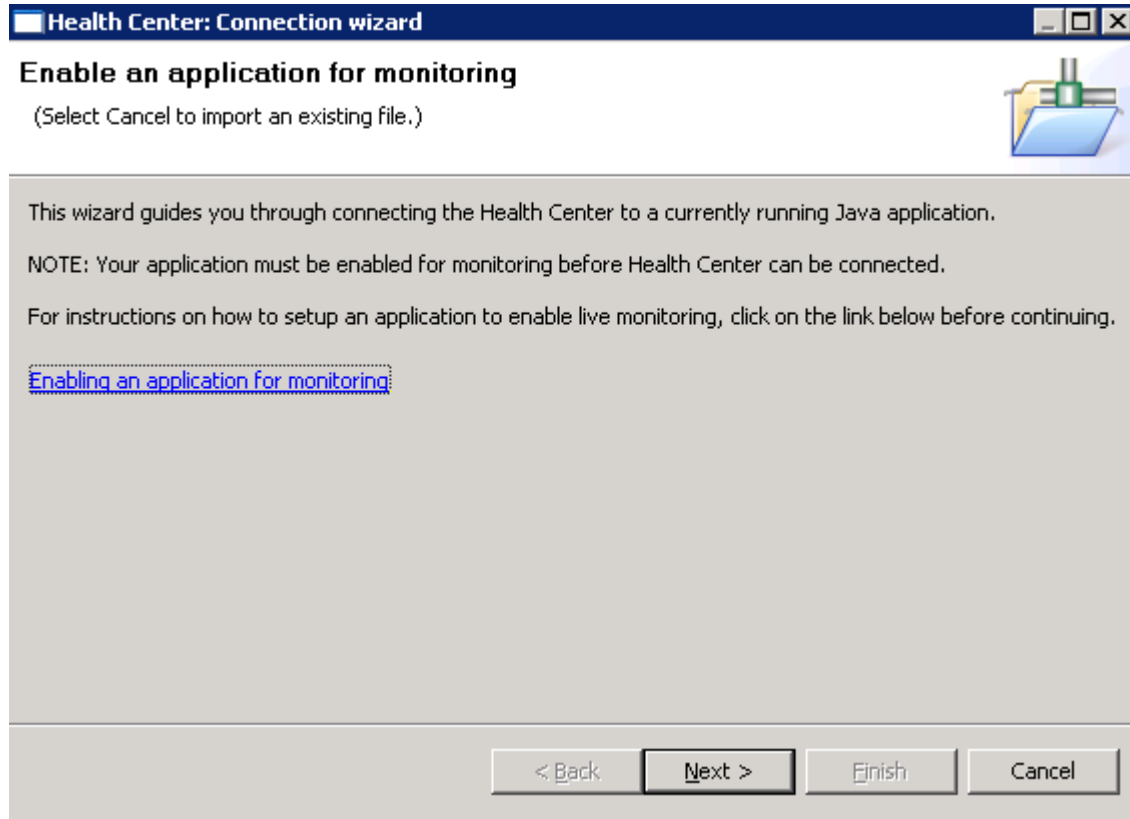
In this section you will trigger a deliberate error in the application which leaks memory. This will trigger the Health Policy defined in the previous section, causing heap dumps to be written, **before** the JVM exhausts all the available heap space. As the application runs in a cluster of two application servers, the corrective actions of the health policy (heap dumps followed by JVM restart) will have minimal impact to the user experience. Having the health policy action restart the application server is more desirable than letting the application server run completely out of memory, causing an out of memory exception. For example, when stopping the application server in a controlled way, it will be 'quiesced' meaning current requests will be completed before the server is stopped.

The heap dumps produced should be adequate to diagnose the memory leak in the next section.

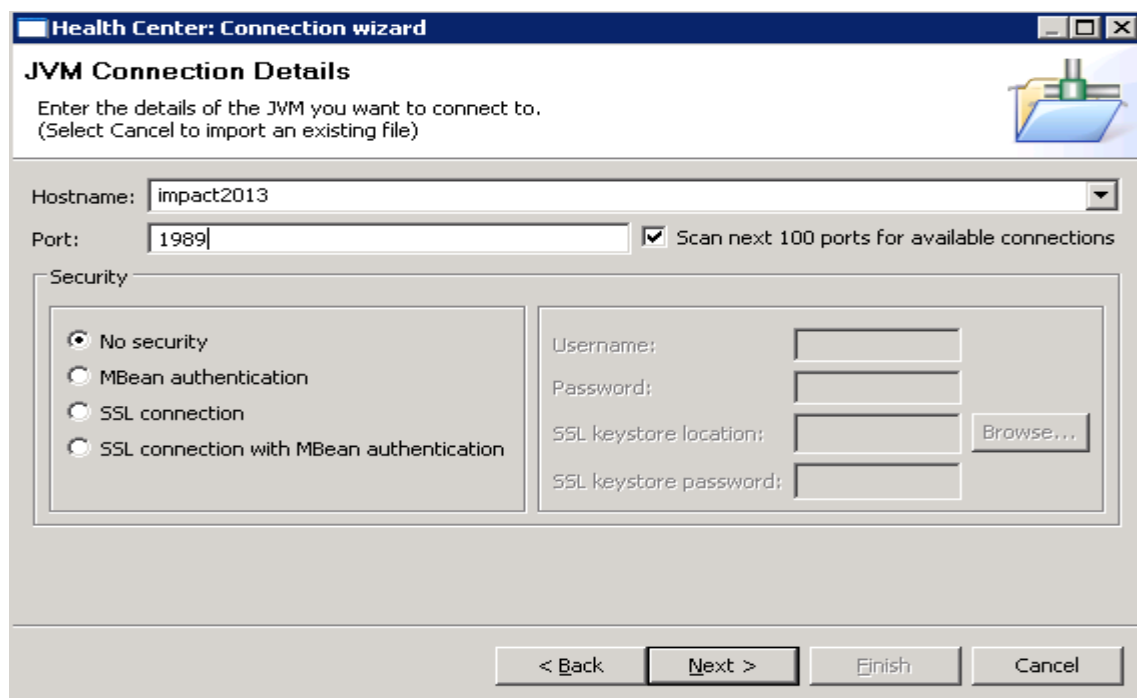
_____ Return to the existing Health Center window and click the **connection** tab - the connection will have been dropped when server1 was restarted in the previous section.



Click **File->New** Connection to re-establish a connection.

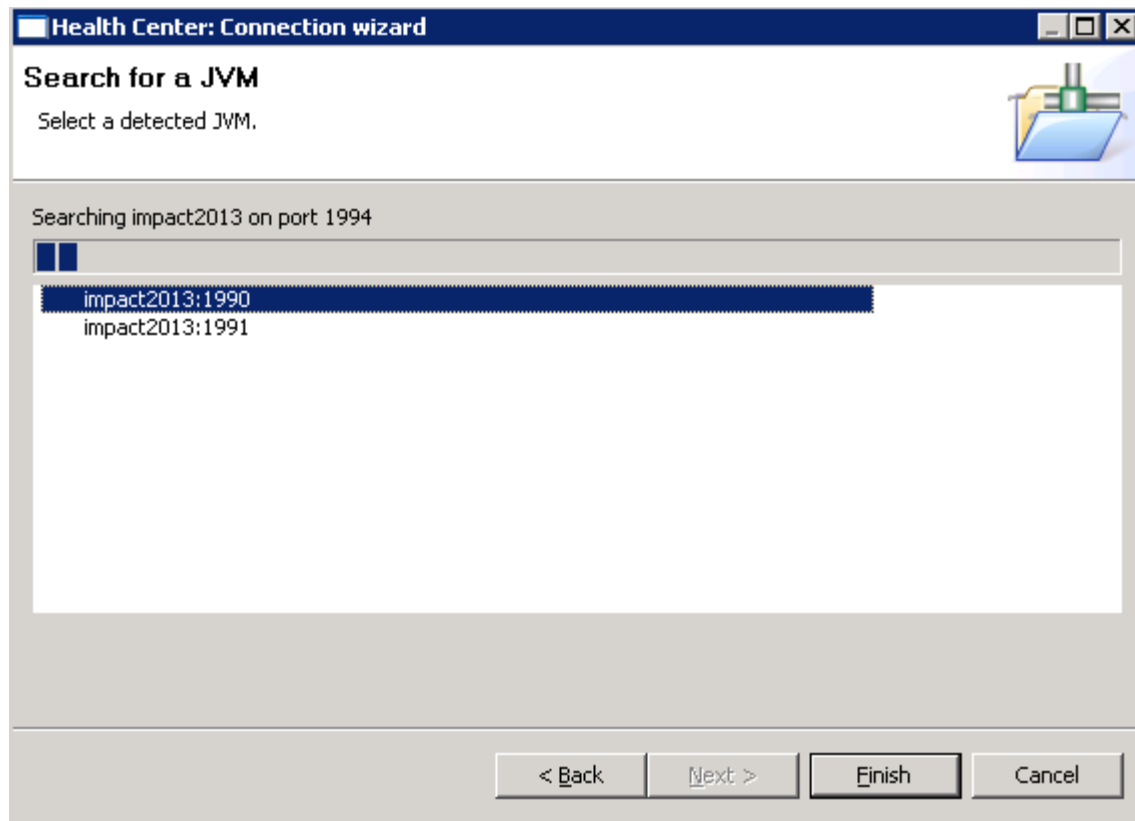


Click **Next**



_____ Click **Next**

_____ Highlight the Health Center agent on port 1990 (server1) and click **Finish**.

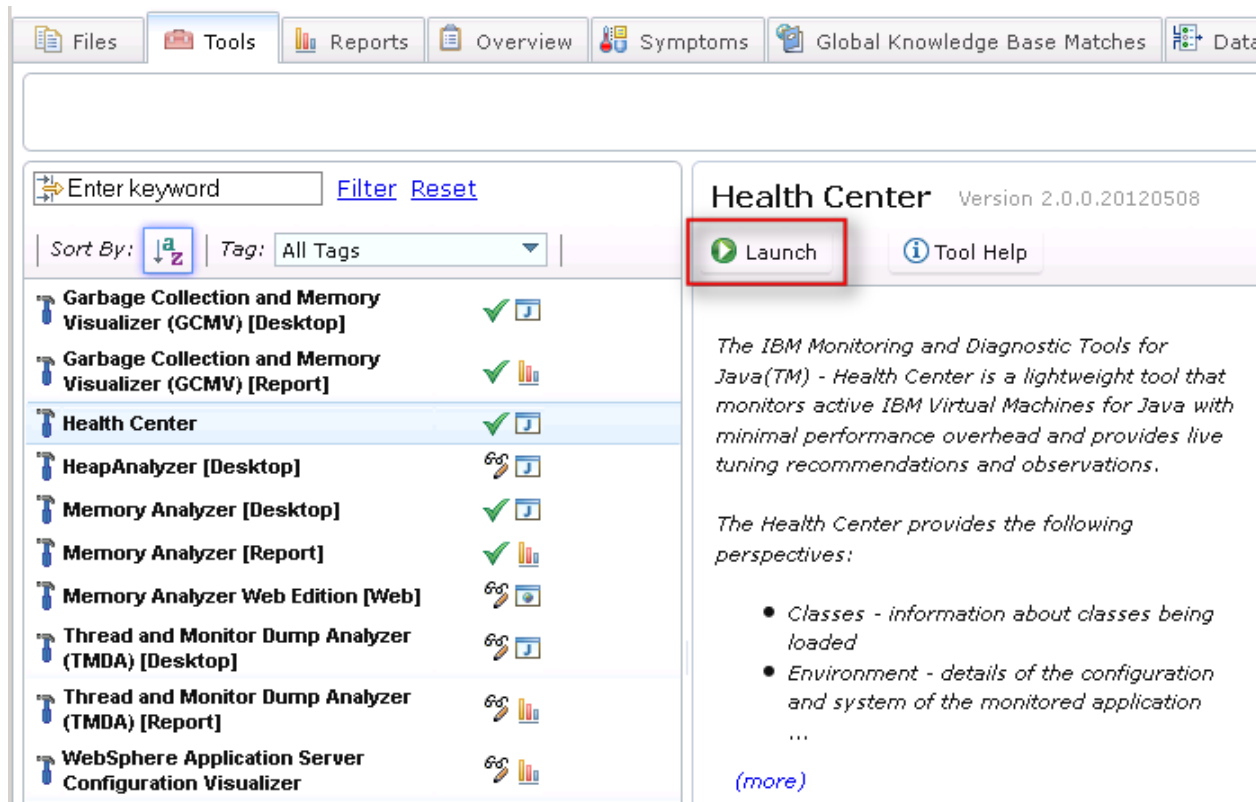


Note:

The goal of this section is to monitor the memory footprint, and see the health policy trigger its actions. However, with two JVMs it is more difficult to know which one serviced the client request, and therefore which JVM could be leaking memory.

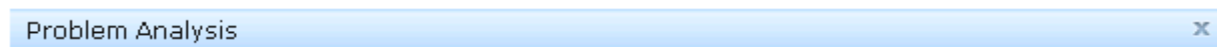
To determine this, you will use two instances of Health Center, and connect to each JVM on different port numbers. After further clicks on the Wheelbarrow, it should become obvious which JVM is experiencing the memory leak. Note the IBM HTTP Server is configured with session persistence, so all requests in the current session will be serviced by the same JVM.

Return to the ISA tab in the browser. Launch a second instance of Health Center by selecting the **Tools** tab, choosing **Health Center** and clicking the **launch** button.



Click **Submit**

A second Health Center client will be launched using Java Web Start. This will take a few moments, wait for the new connection dialogue to appear.

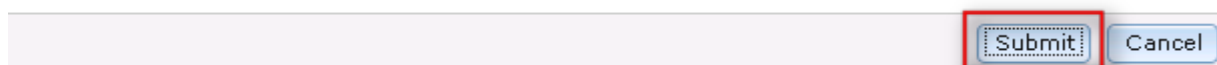


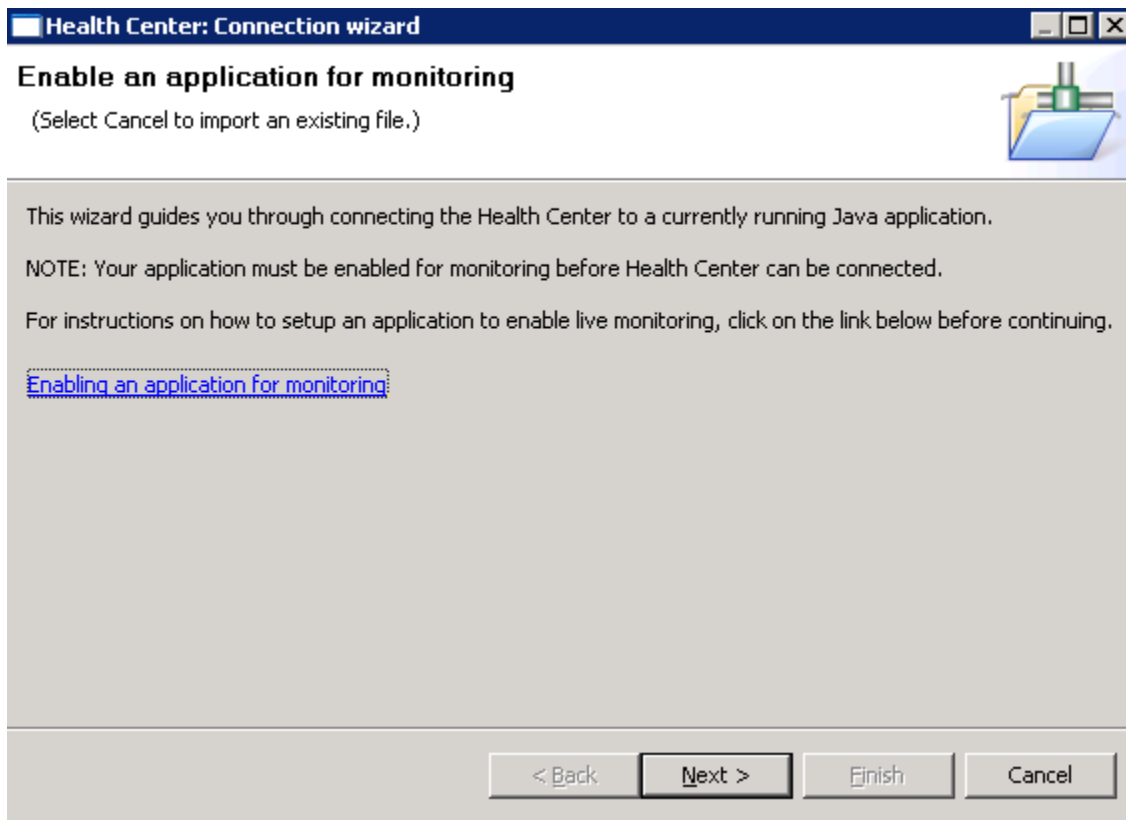
Run Health Center (Version 2.0.0.20120508)

This tool is a desktop application. It will be launched using Java Web Start and will run on your workstation. Using the tool with files associated with ticket will require that you have access to the files from the workstation. If a file is located on a remote server, you can download the file to a local file system location or access the file through a shared storage area. Any existing local file may also be accessed by the tool.

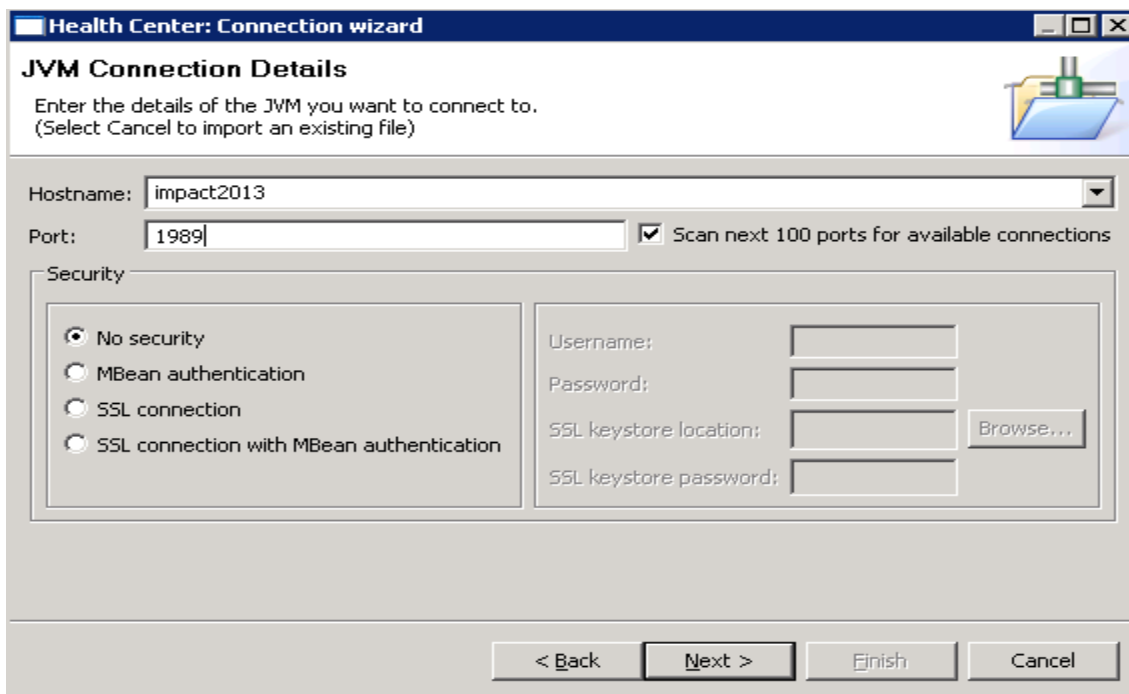
In some cases, analysis of files on your workstation can noticeably degrade performance of other applications running on your workstation.

Click 'Submit' below to begin.

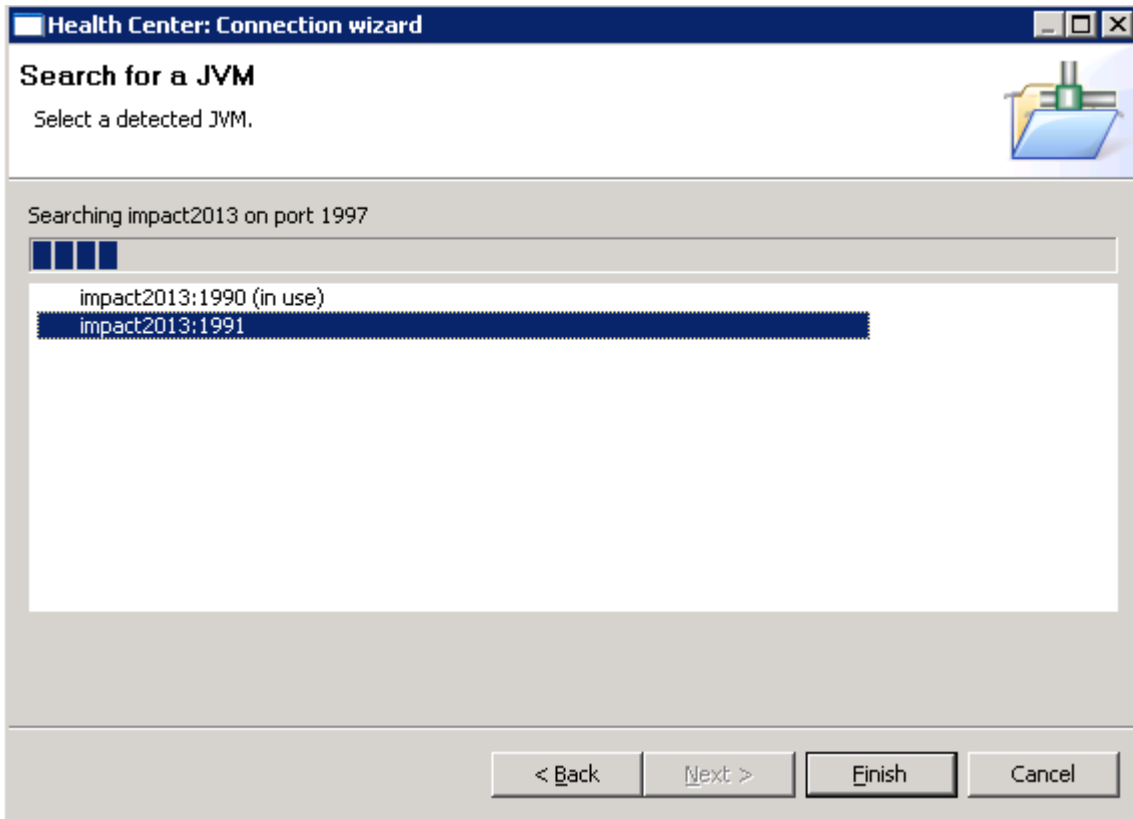




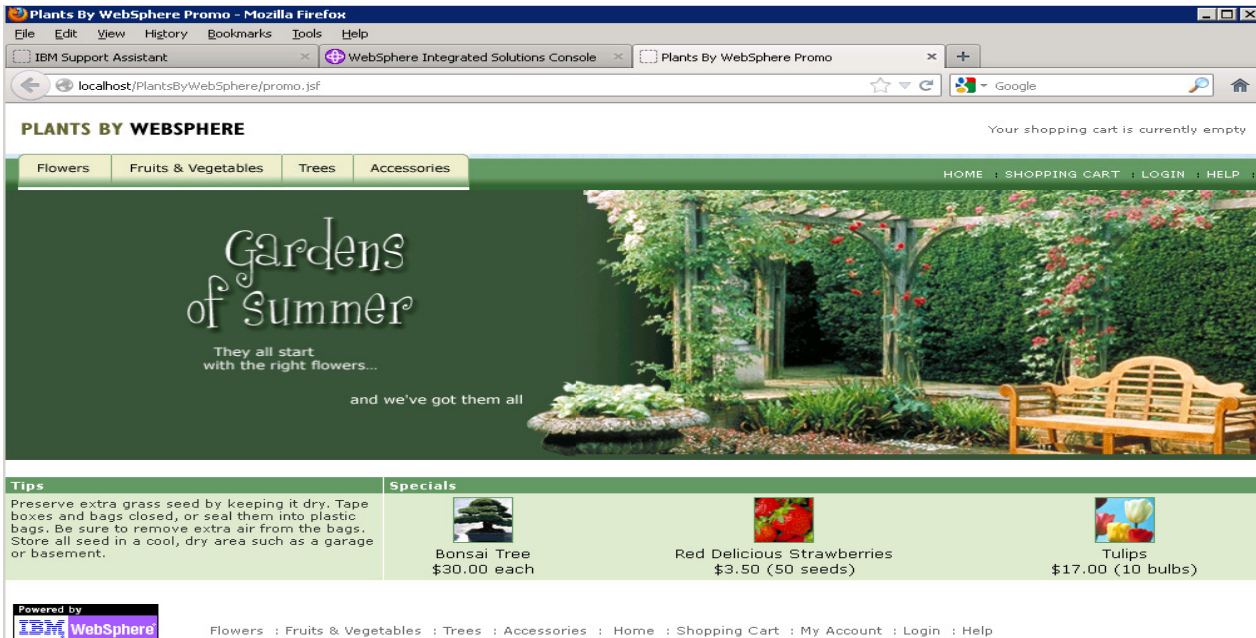
_____ Click **Next**.



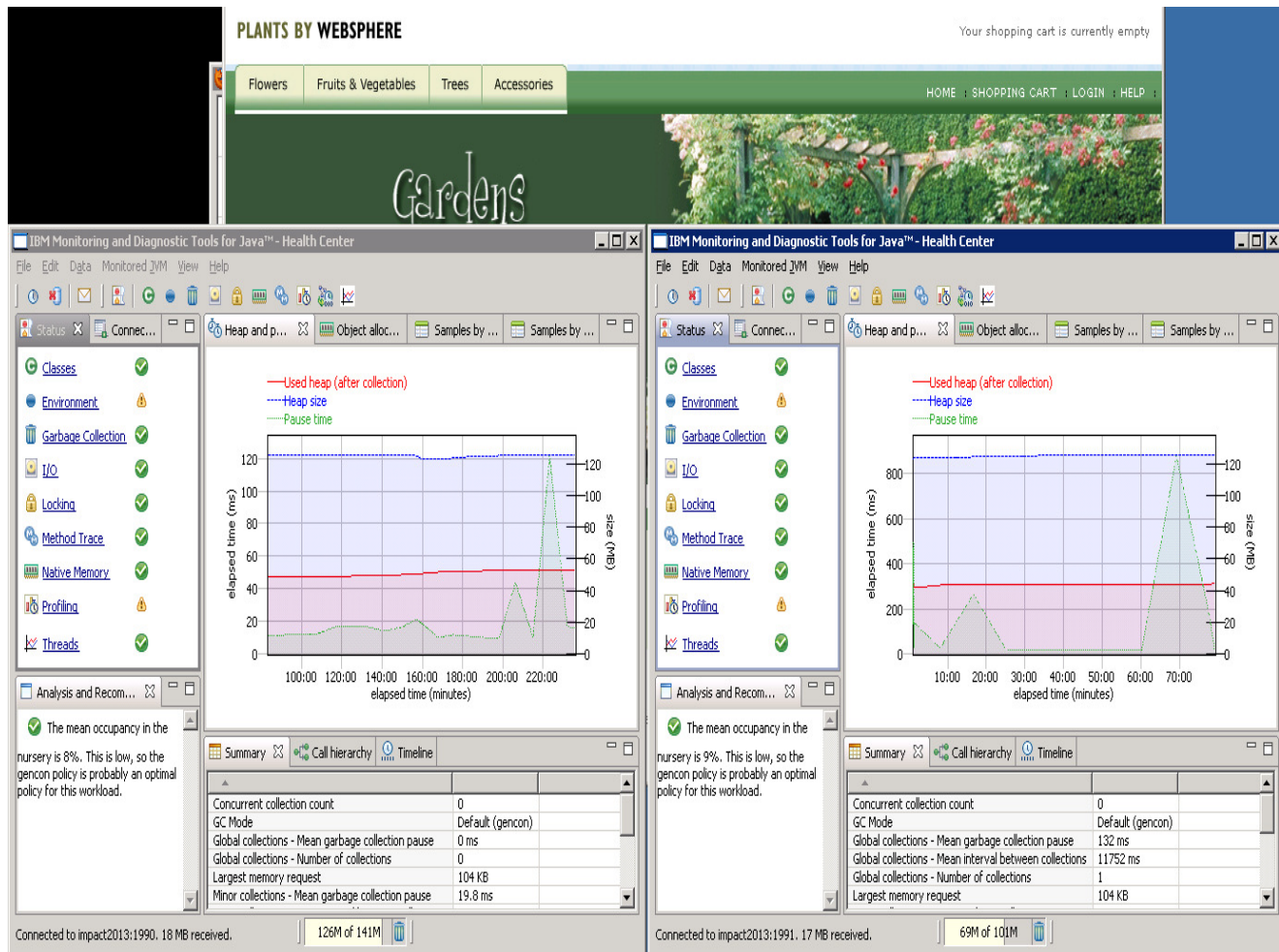
_____ Highlight the Heath Center agent on port 1991 (server2) and click **finish**.



_____ Ensure that the Plants by WebSphere application can be accessed through the browser as follows.(Use the bookmark to access it if needed)

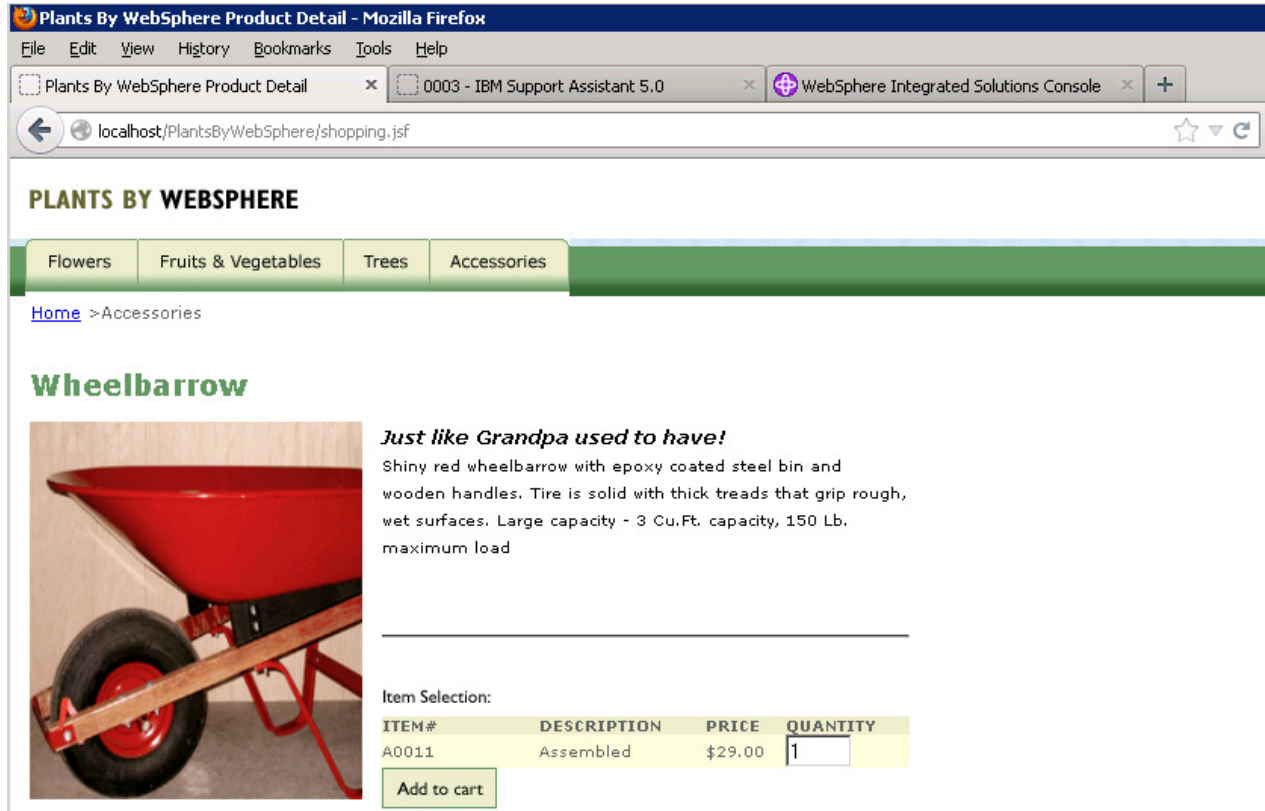


Align the windows so both Health Center clients, and the browser can be viewed simultaneously.

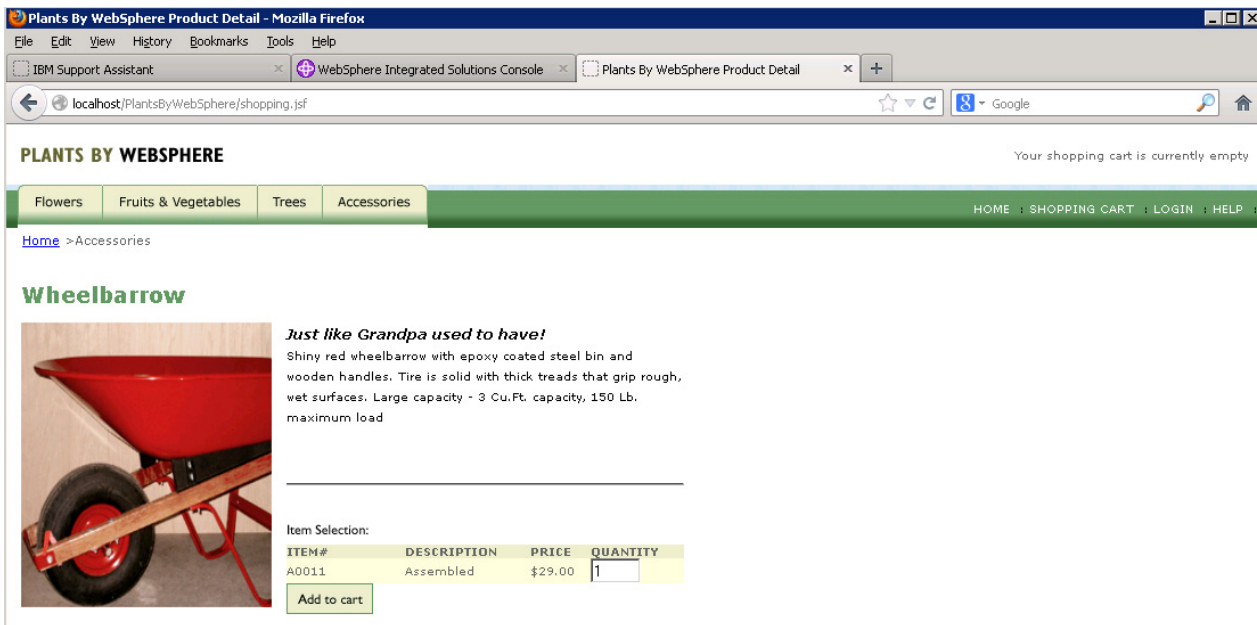


In both Health Center client windows, click the **status** tab, click the **Garbage Collector** link, and select the **Heap and Pause Time** tab on the right hand side (if necessary).

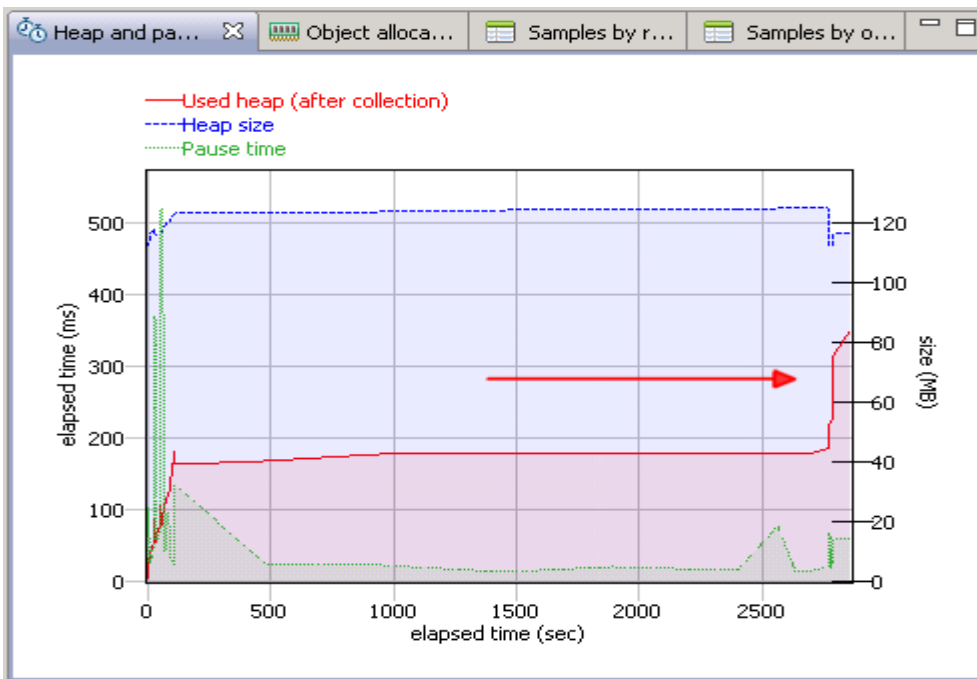
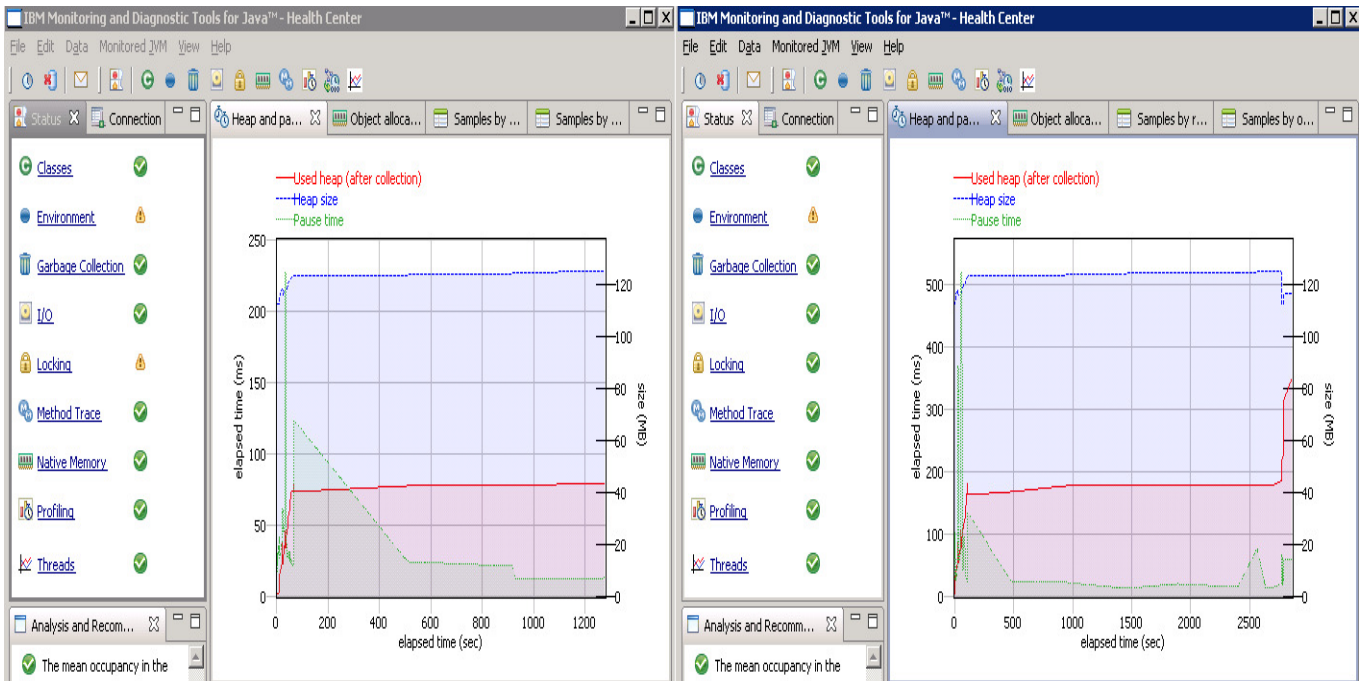
Use the browser to access the Accessories tab and then click the **Wheelbarrow** product on the Accessories page of the Plants by WebSphere application as shown below. This action triggers a deliberate error which causes the application's memory footprint to increase.



_____ In the browser window, click the **Accessories** tab. Click the **Wheelbarrow** to view the product, then return to the accessories page by clicking the **Accessories** tab. Repeat this step a further 3 times, giving a total of **4 wheelbarrow views** in this step. There is no need to add the item to the cart, just selecting it from the tools menu will trigger the memory leak.



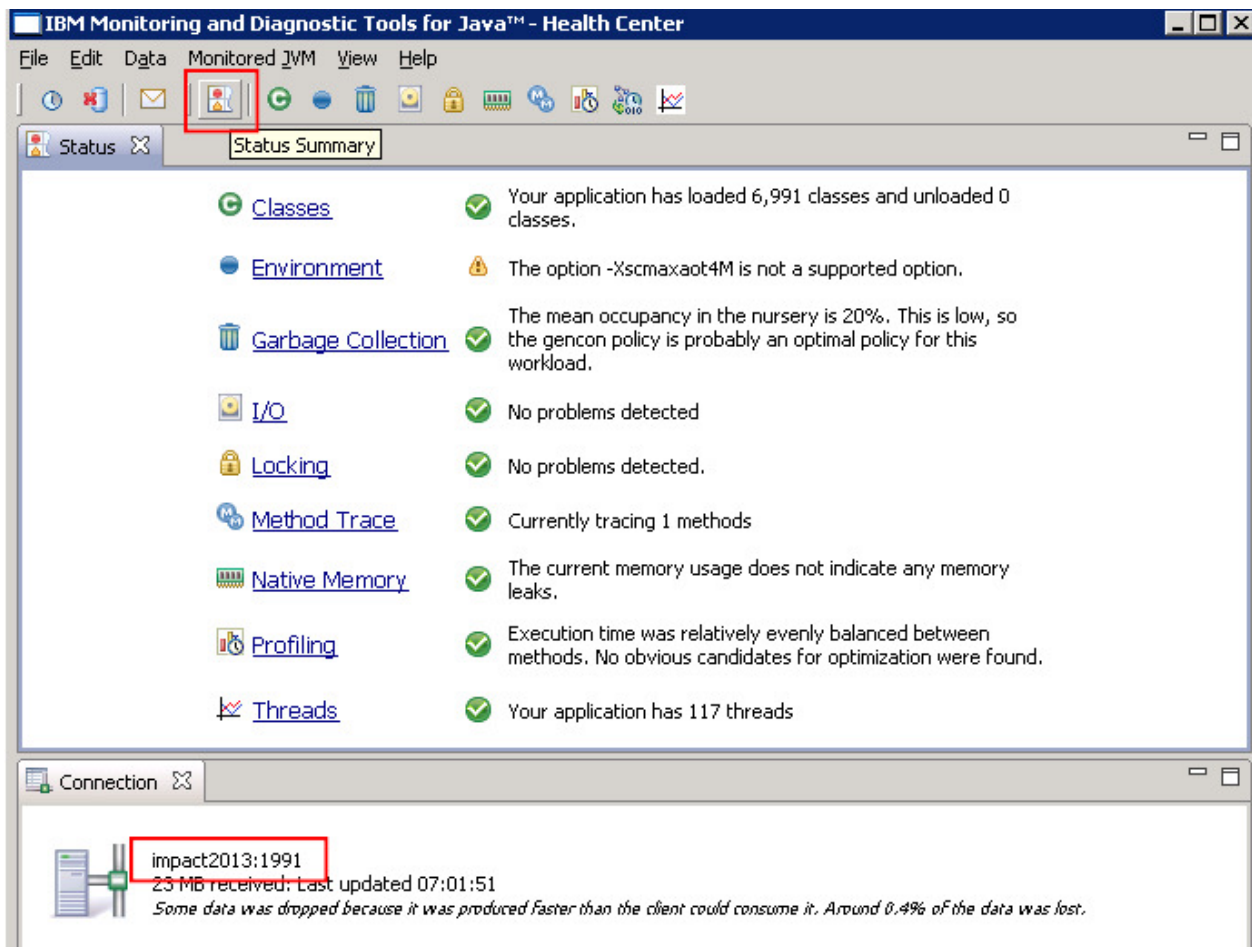
Return to the Health Center windows and wait for them to refresh their data a couple of times (there is a 10 second pause between each refresh). Take a look at the Garbage Collection statistics, you should see the used heap has increased significantly for one of the JVMs. It seems there is a large memory leak over a short period of time in that JVM.



Note:

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

_____ In the health center window where the memory increase was observed, click the **status summary** and make a note of the port number and therefore server. Recall port 1990 is server1 and 1991 is server2.



_____ Close both Health Center windows, disconnecting them from the JVMs.

_____ From the desktop, double click the **Tail** icon.



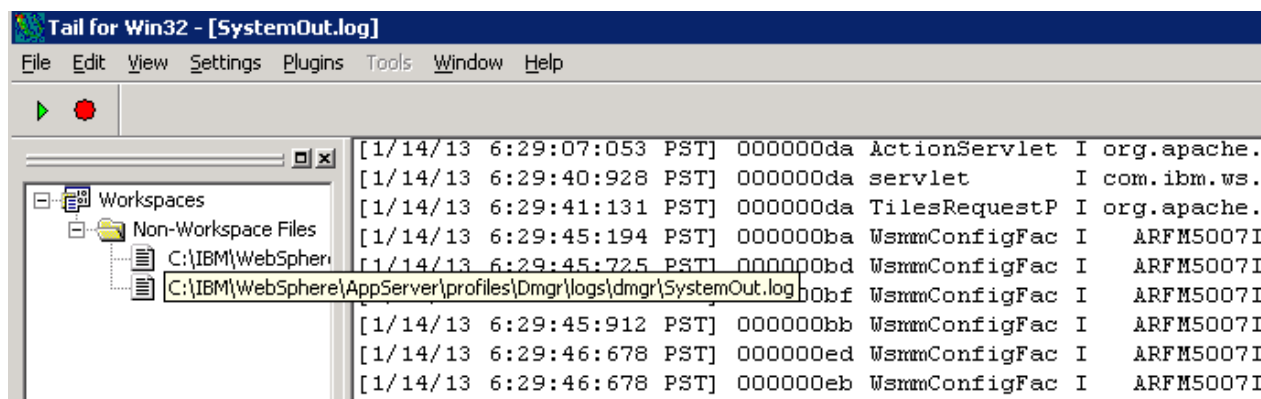
_____ Close any open files in Tail.

_____ Use Tail to monitor the SystemOut.log of the **deployment manager** and **either server1 or server2** (the one you have identified as having a memory leak). The files are located at:

C:\IBM\WebSphere\AppServer\profiles\Dmgr\logs\dmgr

C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server1

C:\IBM\WebSphere\AppServer\profiles\AppSrv01\logs\server2



_____ In the browser, return to the Accessories tab and click the **Wheelbarrow** product **one more time**. This should be sufficient to trigger the health policy actions as the JVM occupancy (after a garbage collection cycle) will have exceeded 85%.

_____ In the tail window, monitor the **Dmgr log** and **wait for up to up to 60 seconds** until the logs similar to these are seen indicating that the Health Management controller has detected the condition as entered for the policy earlier and will be taking the appropriate actions:

JVMMemorySubS W WXDH3004W: The memory consumption limit specified by policy **plantsMemoryHealthPolicy** was exceeded by server server2 on node impact2013Node. The limit is 85 % and the current heap size is 90 % of the maximum of 131072 KB.

HeapDumpTask I WXDH1014I: Heap dumps for server impact2013Cell/impact2013Node/server2 are being issued automatically.

Note:

The health management controller is managed by the WebSphere High Availability Manager which means it could typically be running on any node agent, or deployment manager. For this lab, the health management controller has been configured to always start on the deployment manager, meaning it is easier to locate its logging output. For details of how to configure where such controllers start, see the references.

Note:

The logs indicate the JVM has occupied more than 85% of the available heap space and the health policy has been triggered. At this time, heap dumps are triggered and then the application server JVM is restarted.

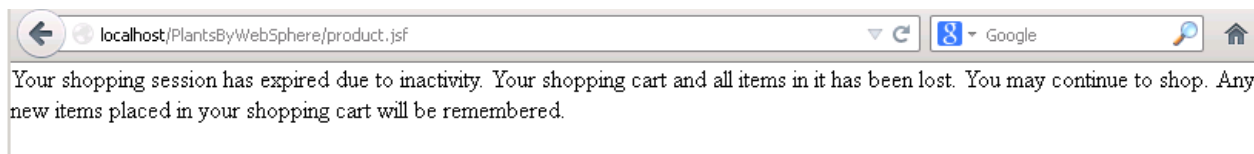
When the heapdumps are generated, you may notice that Health Center pops up three dialog boxes to alert you. You can ignore these.

_____ In the tail window, monitor the SystemOut log for the server identified previously (i.e. server1 or server2).

Note:

If you timed it correctly, you will see the logs show the application server is busy restarting – you may have to wait a few moments until the restart process begins. When it does, continue with the steps to see what happens to further application requests from the browser client.

_____ In the browser, click a product – preferably not the wheelbarrow! You can expect to see the warning below:



Note:

The IBM HTTP Server plugin has detected that one of the application servers is unavailable, and has routed the request to the other server in the cluster.

In this lab environment, HTTP session objects are not shared between the JVMs as session persistence has not been configured, although this is possible using either memory to memory replication or database persistence.

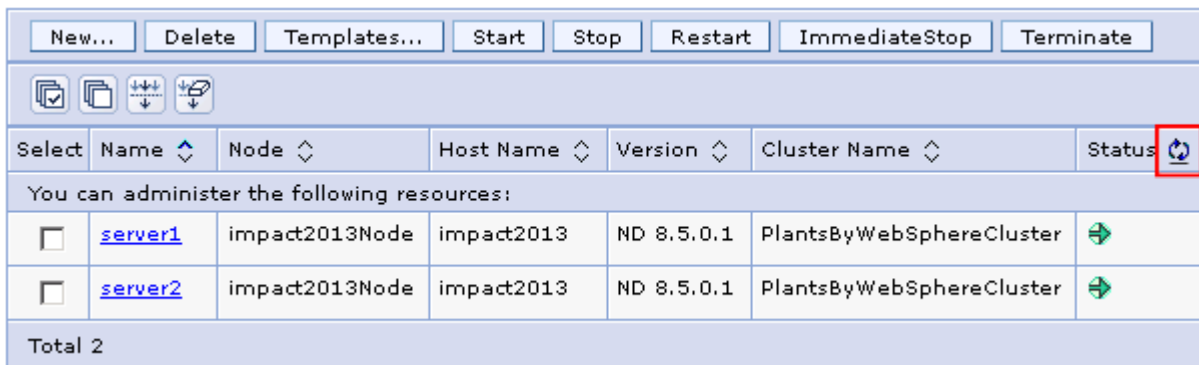
For this reason, the plants application displays a warning that the session data is unavailable, and to continue the user should start a new session.

Start a new session by returning to the plants application homepage using browser favorites or typing URL <http://localhost/PlantsByWebSphere>



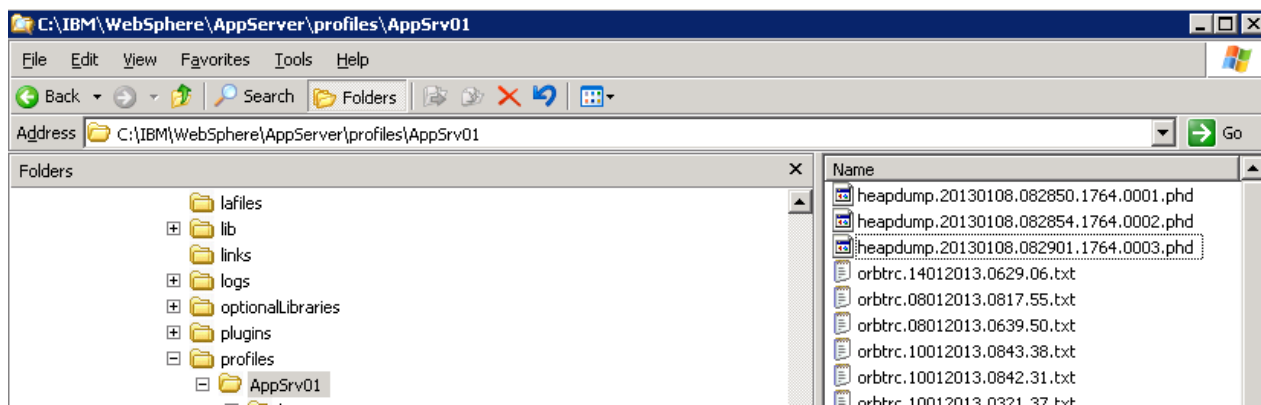
Click a few links to satisfy yourself that you can continue to use the application even though one of the application servers is being restarted.

At the admin console for the Application Server, refresh the status of the servers as shown below to ensure that the stopped server has restarted.



Use Windows explorer to verify that the health policy actions triggered some heap dumps. By default they are written to the JVM's working directory as shown below:

"C:\IBM\WebSphere\AppServer\profiles\AppSrv01".



Note:

In the next section, these heap dumps will be used to diagnose the memory leak. While they represent a Java heap that was only 85% occupied, this should be sufficient to identify the objects that have contributed to the memory leak.

Part 7: Using ISA and the Memory Analyzer to Analyze a Heapdump

Note:

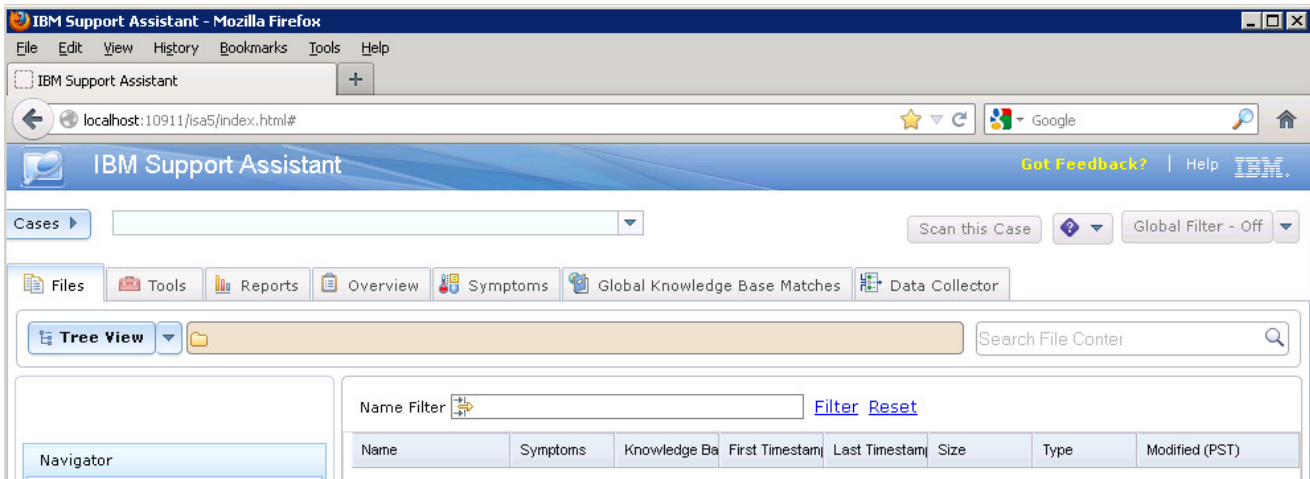
IBM Memory Analyzer is a powerful and flexible tool for analyzing Java heap memory, using heap dumps or system dumps.

ISA 5.0 provides three ways to use this tool – HTML report, web interface and desktop tool. This lab will demonstrate all three.

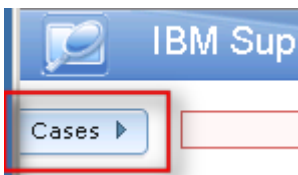
You will use the heap dumps generated in the previous section. However, as every heap dump is different, you may see some slight variation from the screenshots in this lab document, e.g. exact number of bytes for the object size or number of objects in a data structure etc.

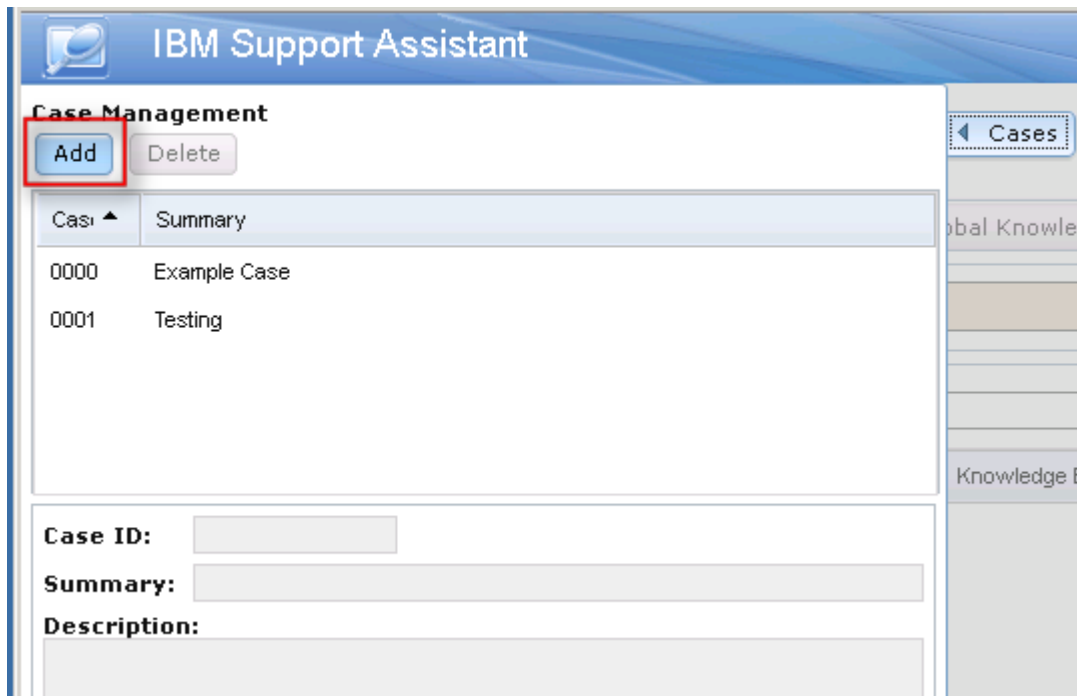
If you prefer, use the ready made heap dump in “E:\Impact lab files\heap dumps” – this files has previously been opened by Memory Analyzer which creates “index files”. Using these files will slightly reduce the amount of time required to complete the lab – it’s up to you.

_____ Launch the browser and use the bookmarks to load the ISA web interface.

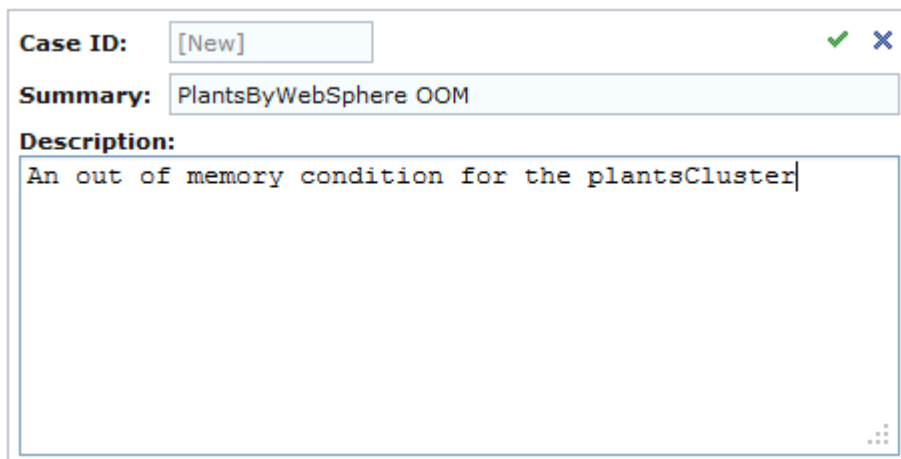


_____ Create a new case by clicking the **Cases** button, and then **Add**.

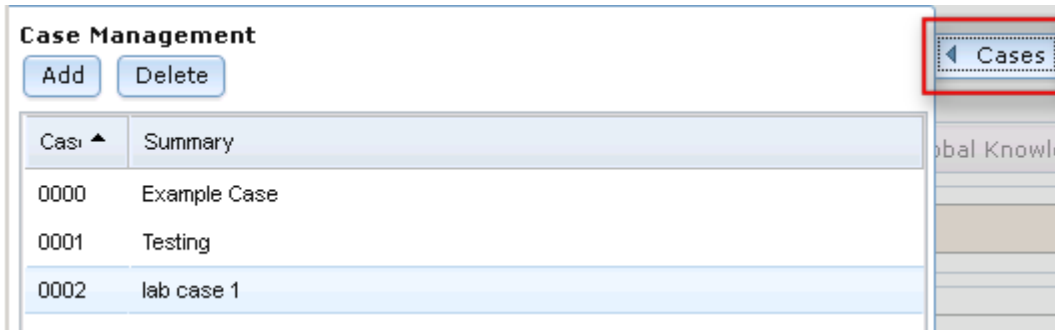




_____ Complete the summary and description, click the **green** tick.

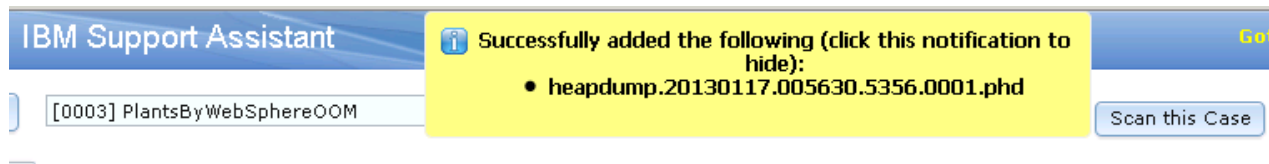
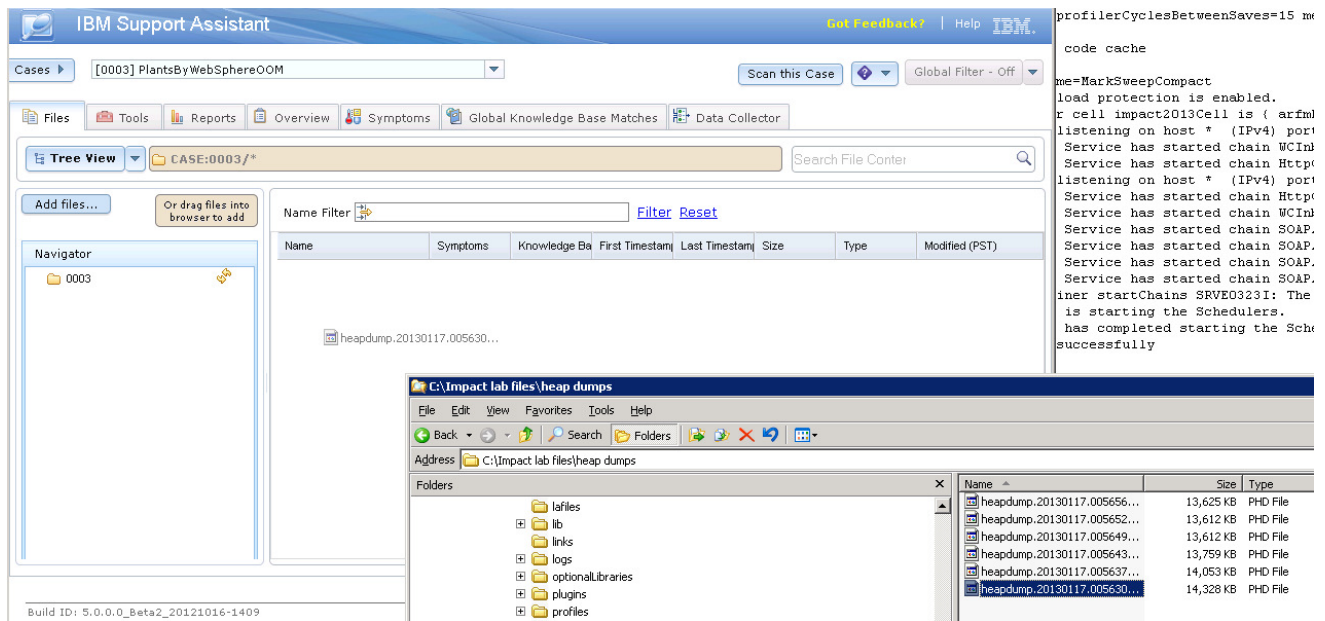


_____ Shrink the cases dialog by clicking **Cases**.



_____ Add one of the heap dumps to the case by dragging the file from Windows Explorer to the ISA case. The location of the heapdump file is:

C:\IBM\WebSphere\AppServer\profiles\AppSrv01



_____ Click the yellow box to dismiss the message

Note:

To make an initial assessment of a heap dump, it can be convenient to run a report on the ISA server. This avoids the need to download the heap dump or system dump to a local workstation. In the next steps you will generate a HTML leak suspects report.

Ensure the relevant case is selected, then choose **Memory Analyzer Report** from the tools tab. Click **Launch**.

The screenshot shows the IBM Software Accelerated Value Program interface. At the top, a 'Cases' dropdown menu is set to '[0003] PlantsByWebSphere OOM'. Below this is a navigation bar with tabs for 'Files', 'Tools', 'Reports', 'Overview', 'Symptoms', and 'Global Knowledge Base Matches'. The 'Tools' tab is active, displaying a list of tools. The 'Memory Analyzer [Report]' tool is highlighted with a red box. To the right of the tool list, a 'Memory Analyzer [Report]' panel is visible, featuring a 'Launch' button (also highlighted with a red box) and a 'Tool Help' link. The panel contains introductory text about the tool and a list of its versions.

Tool Name	Status	Icon
Garbage Collection and Memory Visualizer (GCMV) [Desktop]	✓	J
Garbage Collection and Memory Visualizer (GCMV) [Report]	✓	Bar Chart
Health Center	✓	J
HeapAnalyzer [Desktop]	✓	J
Memory Analyzer [Desktop]	✓	J
Memory Analyzer [Report]	✓	Bar Chart
Memory Analyzer Web Edition [Web]	✓	J

Memory Analyzer [Report]

Launch [Tool Help](#)

IBM Monitoring and Diagnostic Tools for Java

Memory Analyzer is a feature-rich Java heap

This tool is provided in three versions:

- as a report generating version that re
- as an interactive GUI version running
- an interac...

Problem Analysis x

Run Memory Analyzer [Report] (Version 1.2.0.201208221220)

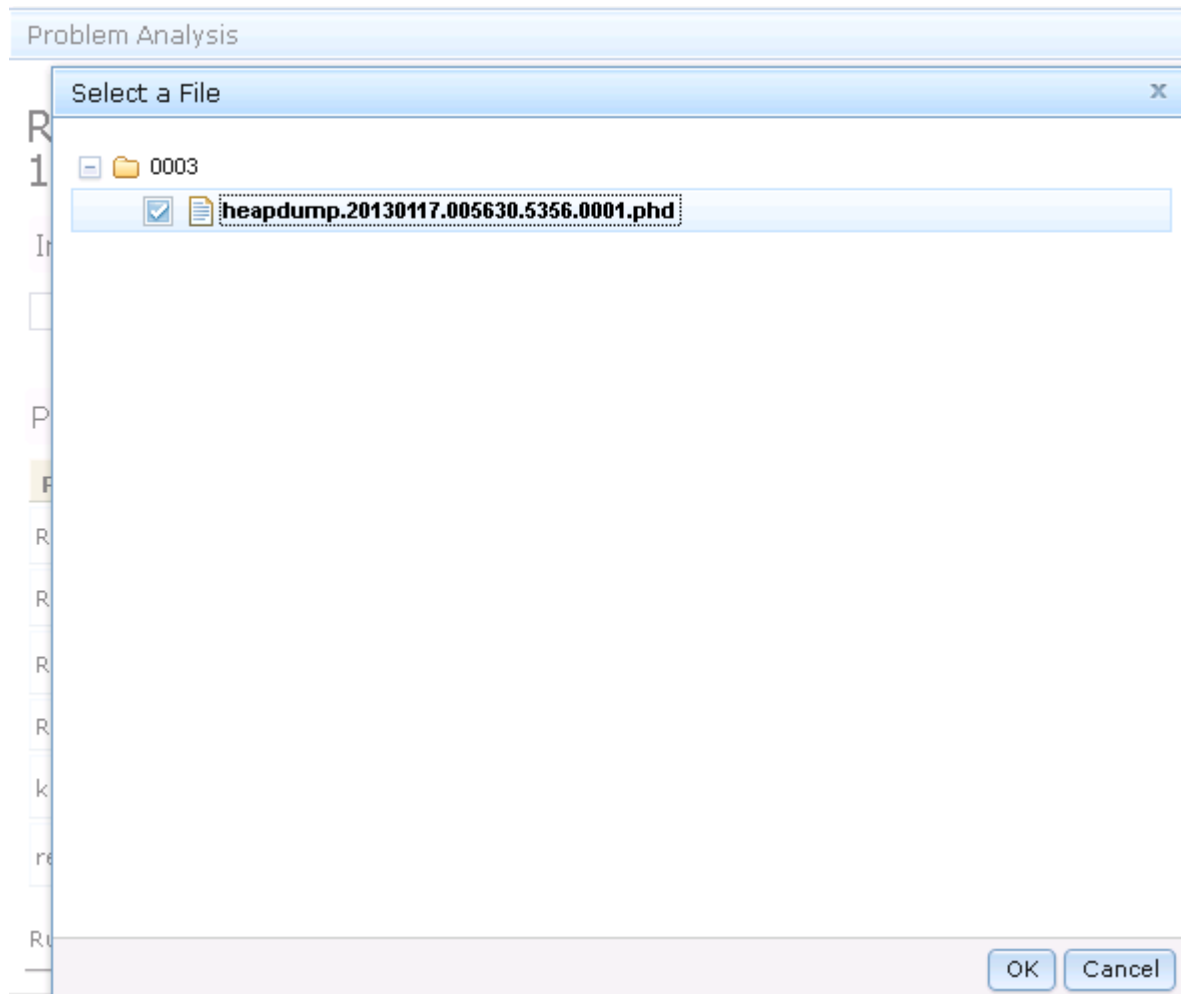
Input Files and Folders *

Parameters

Parameter	Description	Value
Report1	Leak Suspects	Generate Leak Suspects ▼
Report2	Top Components	Generate Top Components ▼
Report3	report 3 e.g. org.eclipse.mat.api:overview	<input style="width: 100%;" type="text"/>
Report4	Additional report/option	<input style="width: 100%;" type="text"/>
keep_unreachable_objects	Keep unreachable garbage objects in the heap	<input type="checkbox"/>
reparse	Do not use any existing indexes from a previous run	<input type="checkbox"/>

Run as background task:

_____ Click **Browse** and select the heap dump in this ISA case.



_____ Click OK (**but don't press Submit yet**)

Note:

The reports to run are already selected.

The leak suspects report is a key feature of Memory Analyzer. It has capabilities to look for probable memory leak suspects including large objects or collections of objects that contribute significantly to the Java heap usage, and displays this information in the form of a pie chart.

The top components report outlines the top memory consumers, i.e. the object at the top of the 'tree' which is responsible for keeping other objects alive in the heap. Top consumers are not necessarily the cause of memory leaks, but could provide information on potential memory inefficiencies in the components.

_____ The **Top Components Report** is not required for this lab, so de-select it.

Problem Analysis

Run Memory Analyzer [Report] (Version 1.2.0.201208221220)

Input Files and Folders *

/ISA5Beta2/ISA5/isa/cases/0003/heapdump.20130117.005630.5356.0001.phd Browse

Parameters

Parameter	Description	Value
Report1	Leak Suspects	Generate Leak Suspects ▼
Report2	Top Components	No Top Components report ▼
Report3	report 3 e.g. org.eclipse.mat.api:overview	
Report4	Additional report/option	
keep_unreachable_objects	Keep unreachable garbage objects in the heap	<input type="checkbox"/>
reparse	Do not use any existing indexes from a previous run	<input type="checkbox"/>

Run as background task:

Submit Cancel

_____ Click **Submit**.

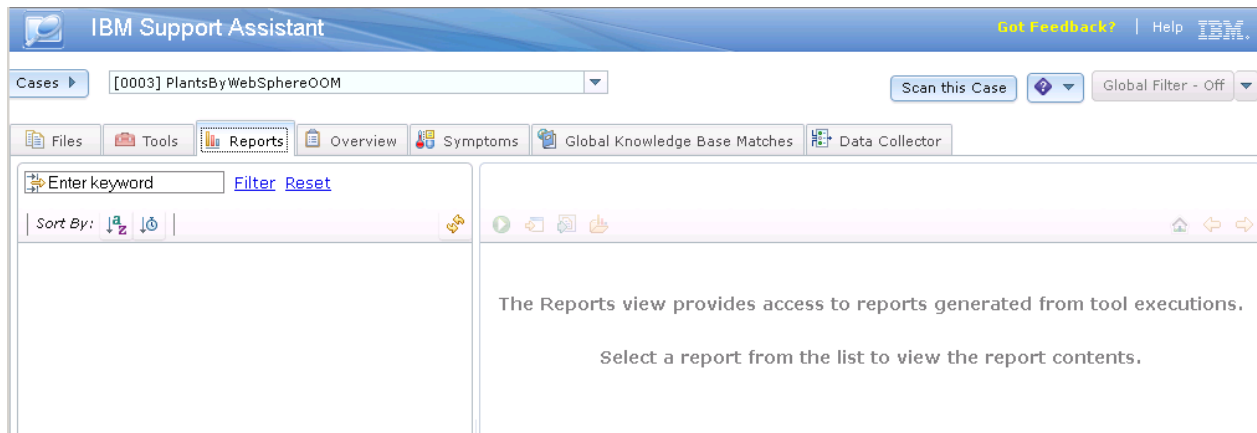
IBM Support Assistant

Cases ▾ [0001] PlantsByWebSphere

The **Memory Analyzer [Report]** tool request has been submitted.
[Go to output folder](#)

Files Tools Reports Overview Symptoms Global Knowledge Base Matches Data Collector

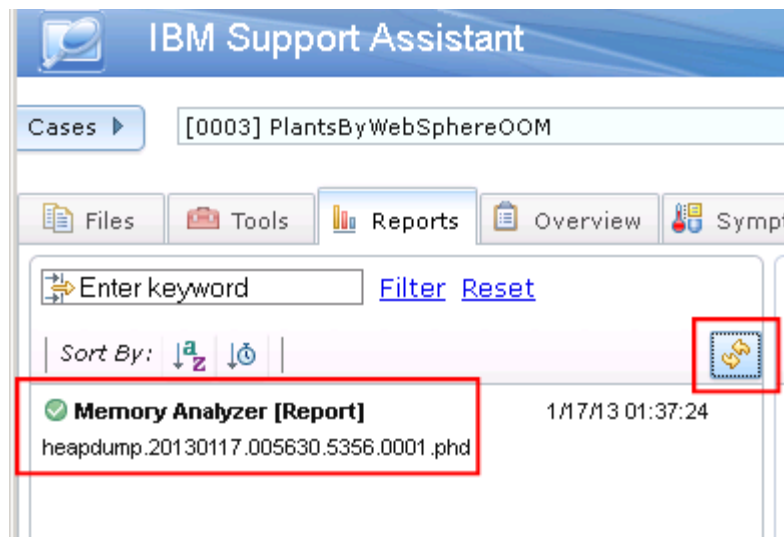
_____ Click the **yellow box** (not the link) to dismiss it, then click the **reports** tab.



Note:

The Memory Analyzer tool running on the ISA server will parse the heap dump, and generate the requested reports. This takes a few moments so please be patient.

Click **refresh** icon until the report is listed and the green tick appears. This can takes around three or four minutes so please be patient.



Select the **report**, then click the **Leak Suspects** Link as shown below.

Files Tools Reports Overview Symptoms Global Knowledge Base Matches Data Collector

Enter keyword Filter Reset

Sort By: [Icons]

Memory Analyzer [Report]	1/17/13 01:37:24
heapdump.20130117.005630.5356.0001.phd	

Memory Analyzer [Report]

Dump file
/ISA5Beta2/ISA5/isa/cases/0003/heapdump.20130117.005630.5356.0001.phd

Metadata file

Options
-output_folder="/DOCUME~1/ADMINI~1/LOCALS~1/Temp/1/com.ibm.java.diagno

Memory Analyzer version: 1.2.0.201208221220

Analysis completed: January 17 2013 01:37

- Report: [heapdump.20130117.005630.5356.0001 Leak Suspects](#)
- Log file(standard out stream): [t0117.013534.421583.log](#)
- Log file(standard err stream): [t0117.013534.421583-err.log](#)

Files Tools Reports Overview Symptoms Global Knowledge Base Matches Data Collector

Enter keyword Filter Reset

Sort By: [Icons]

Memory Analyzer [Report]	24/13 07:14:14
heapdump.20130201.085755.9312.0001.phd	

Memory Analyzer [Report]

Leak Suspects

System Overview

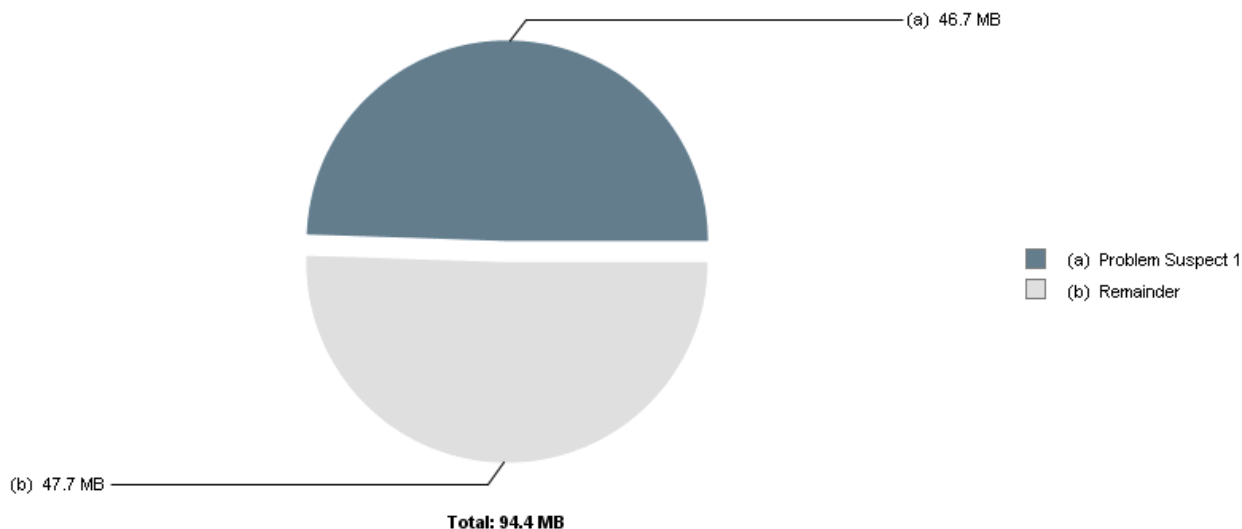
- Leaks
- Overview

(a) 46.7 MB

Legend:
■ (a) Problem Suspect 1
□ (b) Remainder

_____ Examine the overview pie chart. It indicates that a problem suspect occupies around 40Mb of the total heap occupancy. **The exact statistics in your heap dump may vary.**

▼ Overview



_____ Scroll down to examine the detailed description. It identifies an object called **ShoppingBean** which responsible for a large percentage of memory, and that a single **HashMap\$Entry[]** is involved.

▼ ❌ Problem Suspect 1

The class "**com.ibm.websphere.samples.pbw.war.ShoppingBean**", loaded by "**<system class loader>**", occupies **49,004,464 (49.48%)** bytes. The memory is accumulated in one instance of "**java.util.HashMap\$Entry[]**" loaded by "**<system class loader>**".

Keywords
 com.ibm.websphere.samples.pbw.war.ShoppingBean
 java.util.HashMap\$Entry[]

[Details »](#)

Note:

The object syntax “HashMap\$Entry[]” refers to a HashMap object that contains an array of type Entry. The dollar symbol indicates that Entry is an inner class of HashMap. This appears to be the JVM's internal representation of a data structure used in the Plants application code.

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

Note:

An accumulation point is simply a reference in the chain 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 an instance of the ShoppingBean class references the HashMap\$Entry[].

▼ **Shortest Paths To the Accumulation Point**

Class Name	Shallow Heap	Retained Heap
 java.util.HashMap\$Entry[131072] @ 0x67220b0	524,304	40,924,304
 java.util.HashMap @ 0x36d3fc0	48	40,924,368
 java.util.HashSet @ 0x36bcba8	16	40,924,384
 class com.ibm.websphere.samples.pbw.war.ShoppingBean @ 0x29484e8 System Class	80	40,924,464

_____ Scroll down to view the accumulated objects.

▼ Accumulated Objects

Class Name	Shallow Heap	Retained Heap	Percentage
 class com.ibm.websphere.samples.pbw.war.ShoppingBean @ 0x29484e8	80	40,924,464	44.57%
└─  java.util.HashSet @ 0x36bcb8	16	40,924,384	44.57%
└─  java.util.HashMap @ 0x36d3fc0	48	40,924,368	44.57%
└─  java.util.HashMap\$Entry[131072] @ 0x67220b0	524,304	40,924,304	44.57%
└─  java.util.HashMap\$Entry @ 0x322c1b0	24	5,656	0.01%
└─  java.util.HashMap\$Entry @ 0x3abbbf8	24	5,656	0.01%
└─  java.util.HashMap\$Entry @ 0x2e2af18	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3158810	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x32f4410	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x34040f0	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3932fb0	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3d46a28	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3e91c08	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3f6c288	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x3ff6d60	24	4,848	0.01%
└─  java.util.HashMap\$Entry @ 0x420c588	24	4,848	0.01%


Note:

This view shows the objects referred to by the HashMap\$Entry[] accumulation point. You will see the HashMap\$Entry[] contains entries that total approximately 40Mb or 44% of the occupied heap space. Each object is around 4Mb or more in size. Only the first 20 entries from the HashMap\$Entry[] are shown in this report.

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 down to view the “Accumulated Object by Class” table.

▼ Accumulated Objects by Class

Label	Number of Objects	Used Heap Size	Retained Heap Size
 java.util.HashMap\$Entry First 10 of 37,170 objects	37,170	892,080	40,400,000

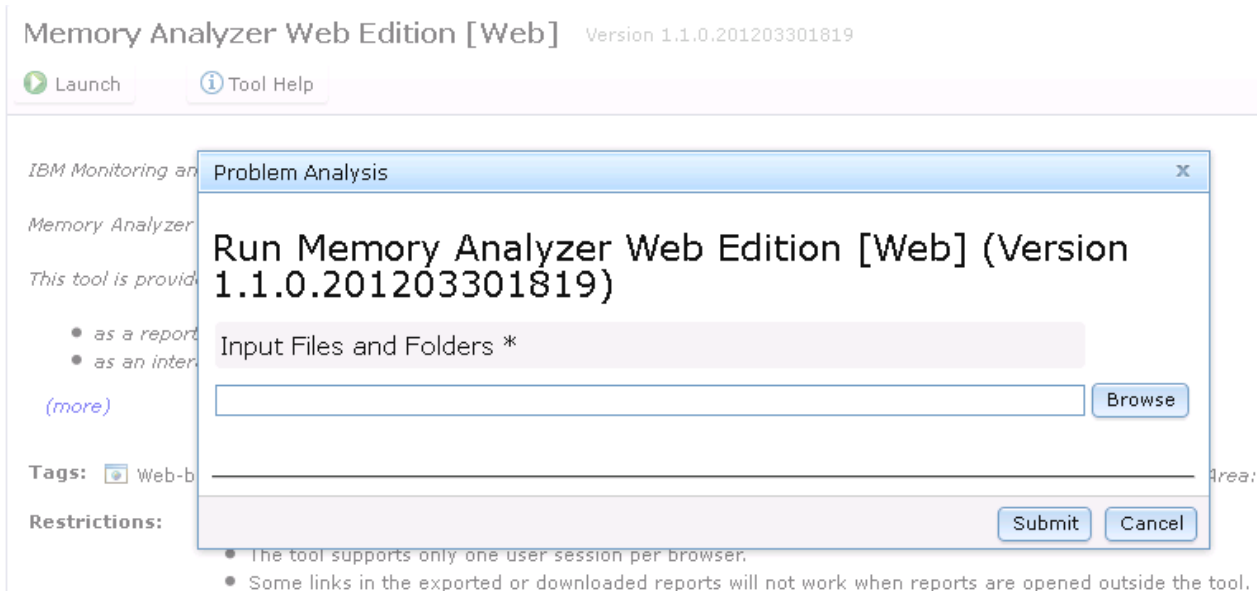
Note:

This shows the total number of objects referred to by the HashMap\$Entry[] accumulation point, in this case over 37,000 objects with a retained size of over 40Mb. The exact statistics in your heap dump may vary.

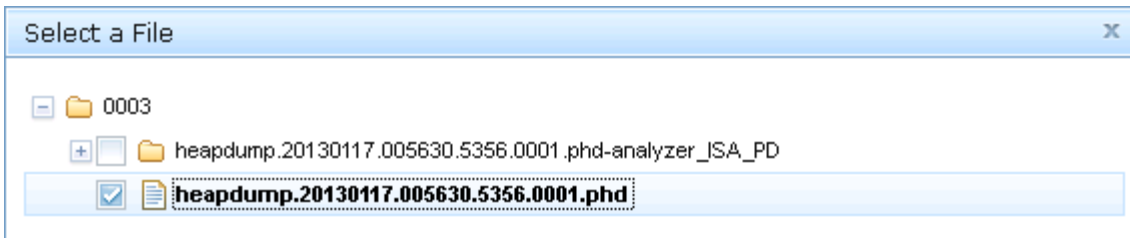
Diagnosing the memory leak would require further analysis of the HashMap\$Entry[], to determine exactly what is stored in that data structure. However, the Memory Analyzer Report is static and it is not possible to follow the links to examine the chain of references between the objects. To facilitate this, either the desktop or web version of Memory Analyzer could be used. In the remainder of this lab section the web version will be used. In the final section of the lab, the desktop edition will be used.

Click the **ISA Tools** tab. Select the **Memory Analyzer Web** Edition. Click **Launch**.

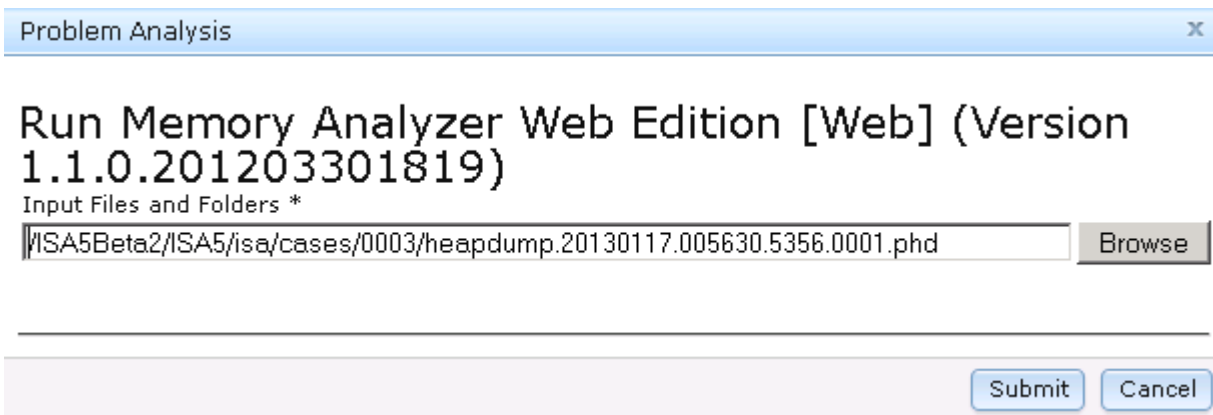
The screenshot shows the IBM Software Accelerated Value Program interface. At the top, there is a 'Cases' dropdown menu with the value '[0003] PlantsByWebSphereOOM'. Below this is a navigation bar with tabs for 'Files', 'Tools', 'Reports', 'Overview', 'Symptoms', 'Global Knowledge Base Matches', and 'Data Collector'. The 'Tools' tab is active, displaying a list of tools on the left and a detailed view of the 'Memory Analyzer Web Edition [Web]' tool on the right. The tool list includes 'Garbage Collection and Memory Visualizer (GCMV) [Desktop]', 'Garbage Collection and Memory Visualizer (GCMV) [Report]', 'Health Center', 'HeapAnalyzer [Desktop]', 'Memory Analyzer [Desktop]', 'Memory Analyzer [Report]', and 'Memory Analyzer Web Edition [Web]'. The 'Memory Analyzer Web Edition [Web]' tool is highlighted with a red box. To the right of this tool, there is a 'Launch' button, also highlighted with a red box, and a 'Tool Help' button. The detailed view of the 'Memory Analyzer Web Edition [Web]' tool includes the text: 'IBM Monitoring and Diagnostic Tools for Java(TM) - Memory Anal. Memory Analyzer is a feature-rich Java heap analyzer that helps This tool is provided in three versions: as a report generating version that reads the dump, build: as an interactive GUI version running on the desktop (more)'. There is also a 'Filter Reset' link and a 'Sort By: java' dropdown menu in the tool list area.

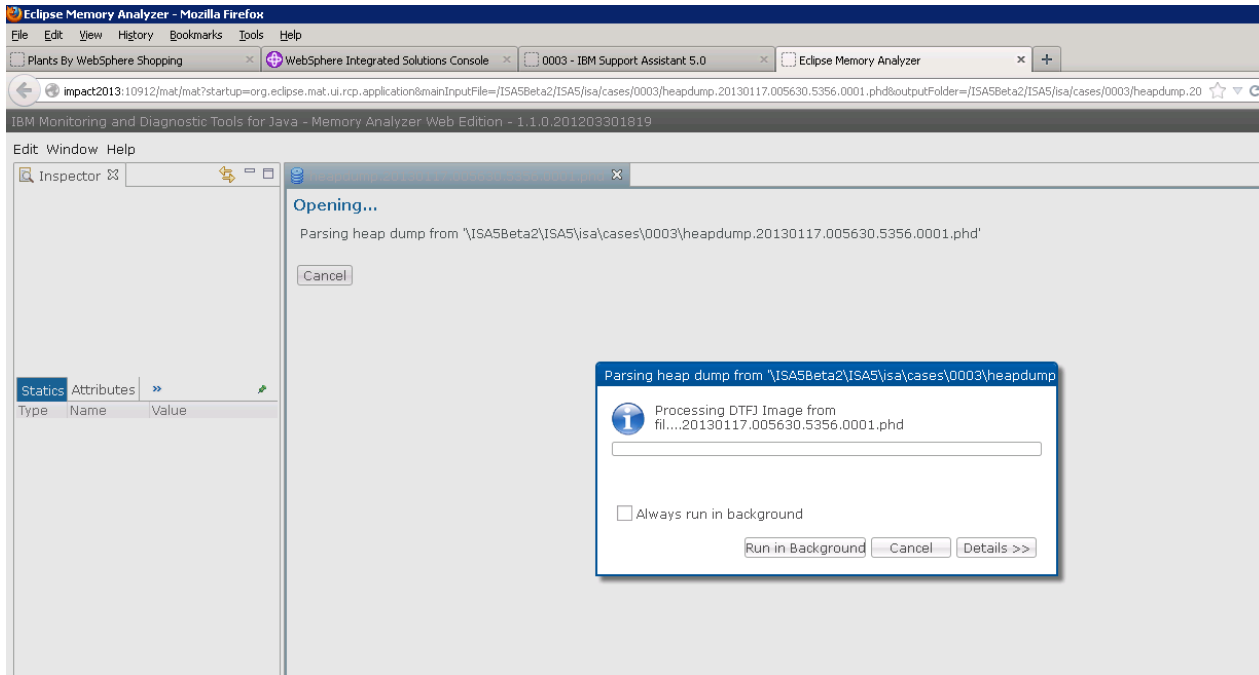


_____ On the “Problem Analysis” window that pops up as shown above, Click **Browse** and select the same heap dump in the ISA case as shown below. Then, Click **OK**.

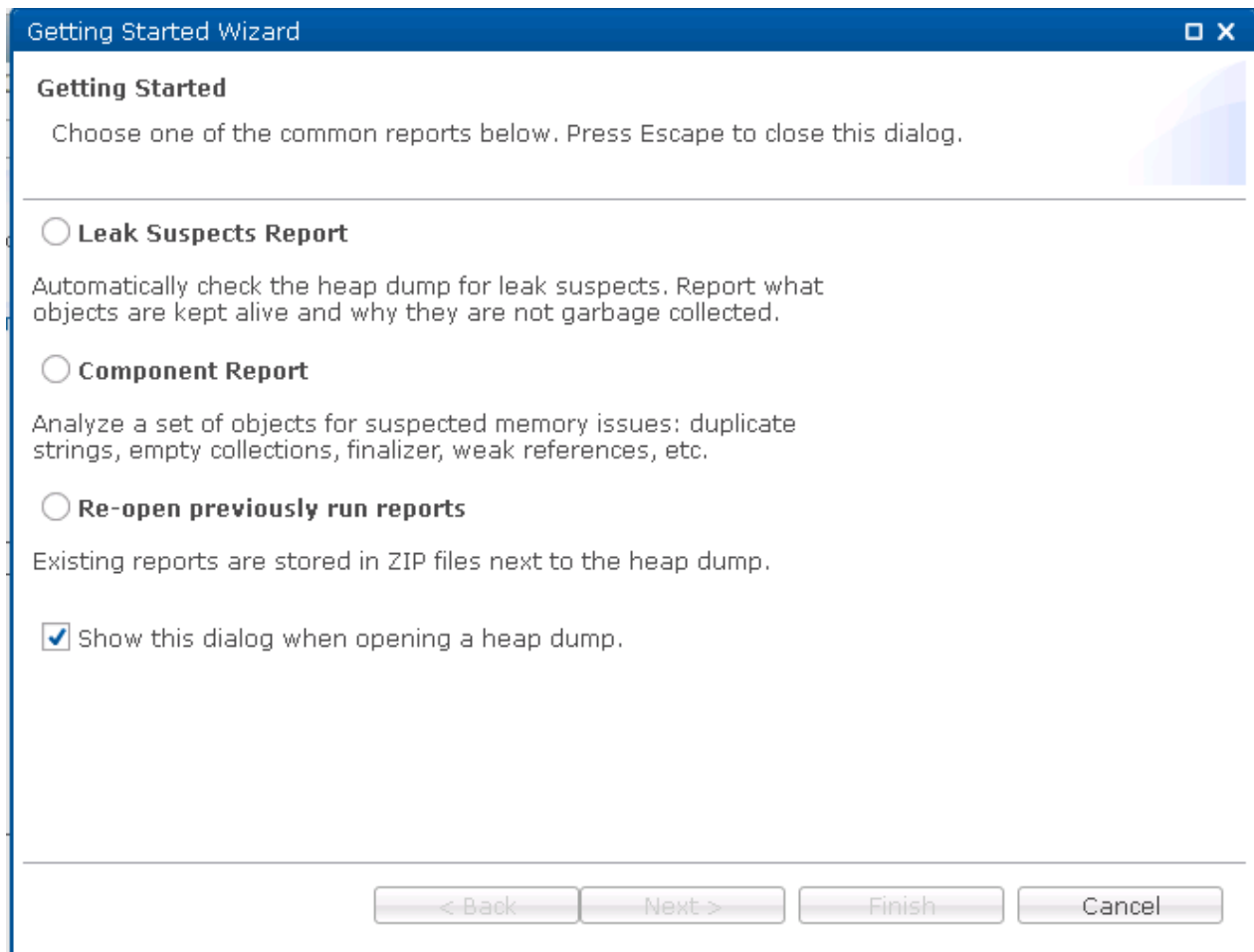


_____ Click **Submit**. The browser window may remain blank for a few moments.

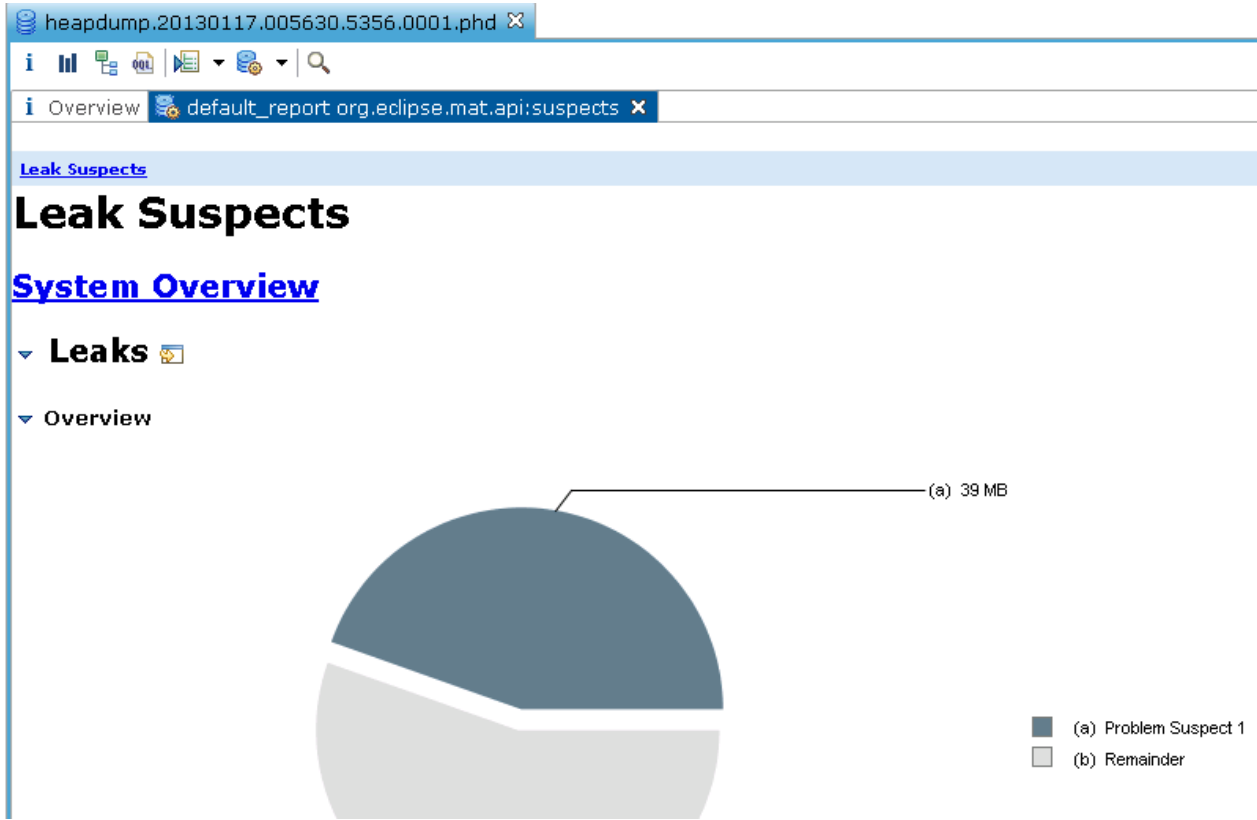





_____ Wait for the heap dump to be parsed, this time the operation will be quicker as Memory Analyzer 'index files' already exist on the filesystem from the previous analysis.









_____ Select the **Leak Suspects** Report to generate the same report you generated previously with the Memory Analyzer Report tool. Click **Finish**.



_____ Click the details link and scroll down to the “**Accumulated Objects**” table.

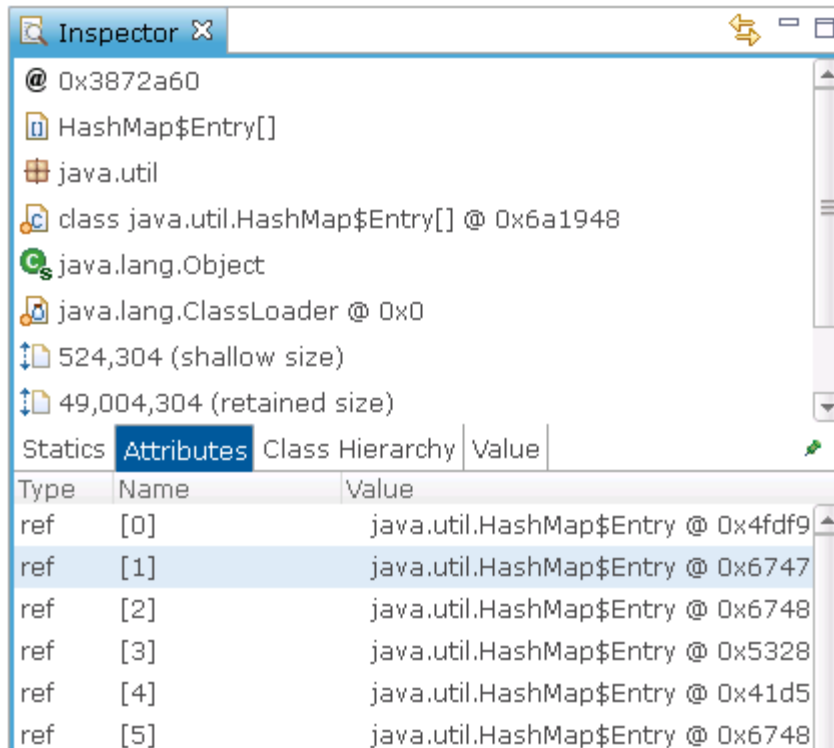
_____ To determine what is stored in each of these HashMap entries, click the **accumulation point link**, i.e. the HashMapEntry **with the array icon** 

▼ Accumulated Objects

Class Name	Shallow Heap	Retained Heap	Percentage
 class com.ibm.websphere.samples.pbw.war.ShoppingBean @ 0x2987be0	80	49,004,464	49.48%
└─  java.util.HashSet @ 0x3968288	16	49,004,384	49.48%
└─  java.util.HashMap @ 0x3af9e20	48	49,004,368	49.48%
└─  java.util.HashMap\$Entry[131072] @ 0x3872a60	524,304	49,004,304	49.48%
└─  java.util.HashMap\$Entry @ 0x3f19510	24	6,464	0.01%
└─  java.util.HashMap\$Entry @ 0x317d538	24	5,656	0.01%
└─  java.util.HashMap\$Entry @ 0x3a8f400	24	5,656	0.01%

Note:

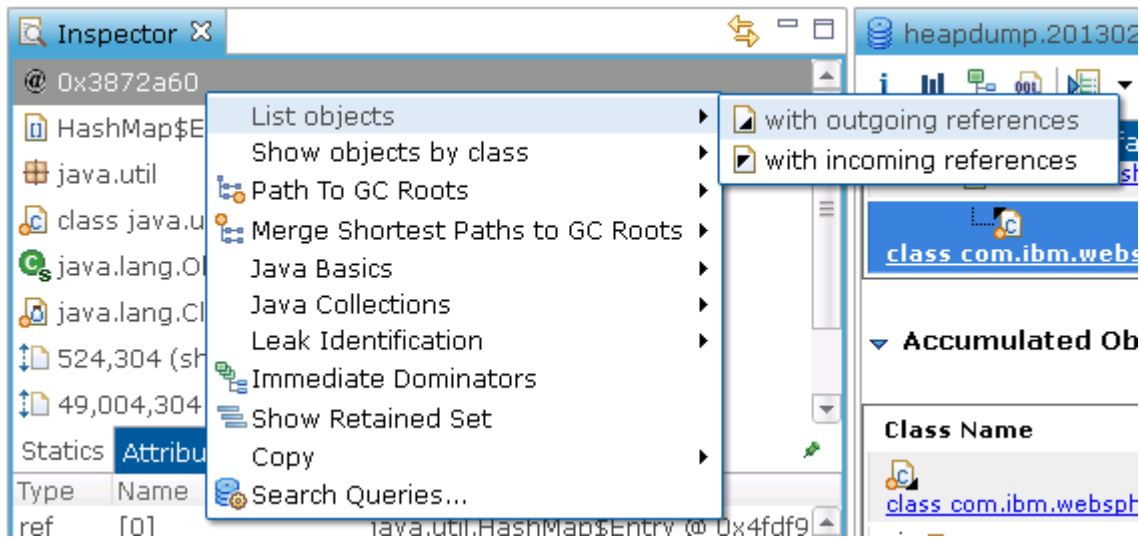
The left Inspector panel show details for the HashMap, including its attributes as shown below.



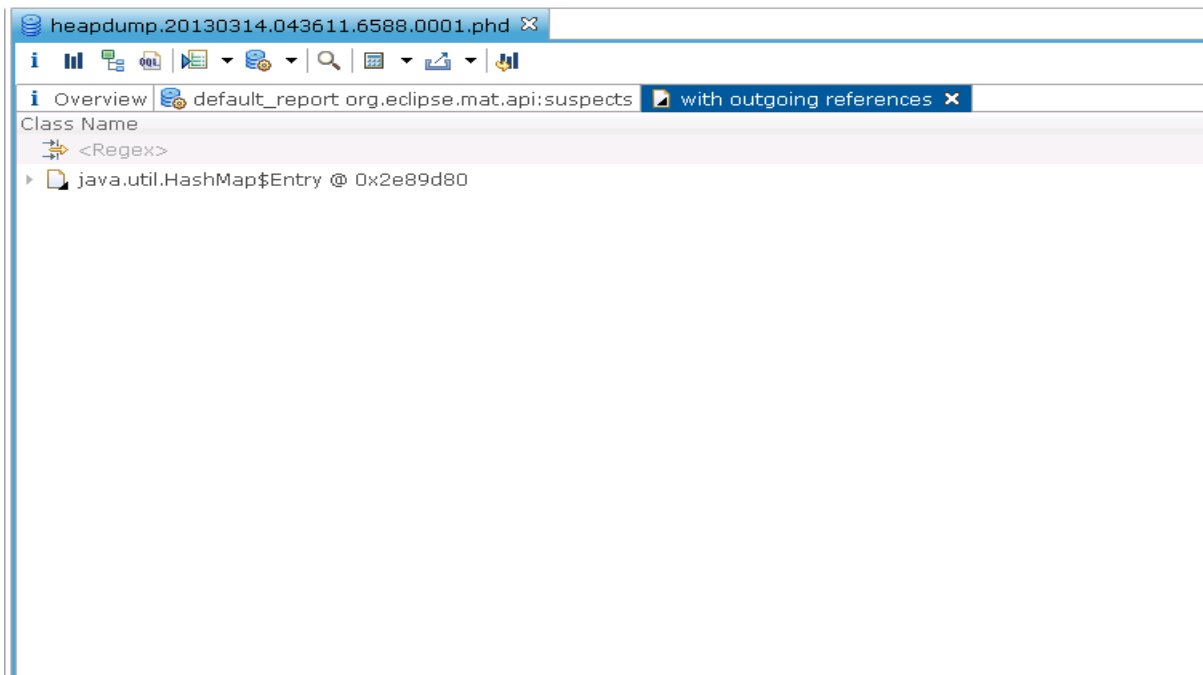
Right click the object reference in the Inspector tab, and select **List Objects->with outgoing references** to display all references from the selected HashMap object.

Note:

There are other more advanced ways to view outgoing references such as "Immediate Dominators" which can help simplify the chain of references by showing only the most significant references, i.e. only those that keep an object alive in the heap.

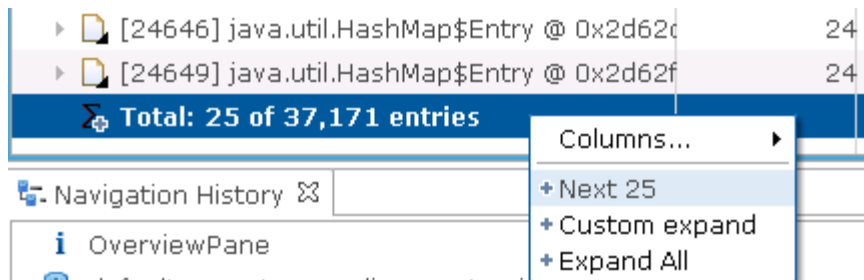


_____ Expand the top level HashMap object to show the thousands of entries that compose this HashMap.



Class Name	Shallow Heap	Retained Heap
[6138] java.util.HashMap\$Entry @ 0x2d5f06	24	808
[34127] java.util.HashMap\$Entry @ 0x2d5f3	24	1,616
[1488] java.util.HashMap\$Entry @ 0x2d5f6b	24	808
[20006] java.util.HashMap\$Entry @ 0x2d5f6	24	808
[10747] java.util.HashMap\$Entry @ 0x2d5fd	24	808
[1489] java.util.HashMap\$Entry @ 0x2d6002	24	1,616
[6140] java.util.HashMap\$Entry @ 0x2d6035	24	808
[20009] java.util.HashMap\$Entry @ 0x2d606	24	808
[29462] java.util.HashMap\$Entry @ 0x2d609	24	808
[20010] java.util.HashMap\$Entry @ 0x2d60c	24	1,616
[6142] java.util.HashMap\$Entry @ 0x2d613	24	808
[20011] java.util.HashMap\$Entry @ 0x2d616	24	1,616
[29464] java.util.HashMap\$Entry @ 0x2d619	24	808
[10750] java.util.HashMap\$Entry @ 0x2d61f	24	808
[20014] java.util.HashMap\$Entry @ 0x2d622	24	808
[6143] java.util.HashMap\$Entry @ 0x2d626	24	808
[15340] java.util.HashMap\$Entry @ 0x2d629	24	808
[24646] java.util.HashMap\$Entry @ 0x2d62c	24	808
[24649] java.util.HashMap\$Entry @ 0x2d62f	24	1,616
Σ Total: 25 of 37,171 entries		

Scroll down and 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. Notice that the HashMap entry refers to a ShoppingContainer class in the plants sample (you will need to adjust the column widths).

▾ [20011] java.util.HashMap\$Entry @ 0x2d61640	24	1,616
▸ <class> class java.util.HashMap\$Entry @ 0x6a1920 System Class	80	80
▸ com.ibm.websphere.samples.pbw.war.ShoppingBean\$ShoppingContainer @ 0x2d61658	16	784
▸ java.util.HashSet @ 0x36bcba8	16	40,924,384
▸ java.util.HashMap\$Entry @ 0x6720dd8	24	808
Σ Total: 4 entries		

Note:

The class name “ShoppingBean\$ShoppingContainer” means the “ShoppingContainer” class is an inner class of “ShoppingBean”.

This ShoppingContainer object is responsible for keeping 784 bytes alive in the heap. The HashMapEntry actually keeps 1616 bytes alive in total, including the ShoppingContainer. All the other HashMapEntry objects in the list follow an identical pattern. **The exact statistics in your heap dump may vary.**

_____ Try expanding another HashMap entry. Their retained heap may vary, but they always contain a ShoppingContainer of the same size.

▾ [1488] java.util.HashMap\$Entry @ 0x2d5f6b0	24	808
▸ <class> class java.util.HashMap\$Entry @ 0x6a1920 System Class	80	80
▸ com.ibm.websphere.samples.pbw.war.ShoppingBean\$ShoppingContainer @ 0x2d5f6c8	16	784
▸ java.util.HashSet @ 0x36bcba8	16	40,924,384
Σ Total: 3 entries		

Note:

You have established that objects from the plants sample seem to be involved with the memory leak - a ShoppingBean 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 on the Memory Analyzer toolbar to open a histogram.

Class Name	Objects	Shallow Heap	Retained Heap
<Regex>	<Numeric>	<Numeric>	<Numeric>
java.lang.String	188,812	4,531,488	>= 19,011,6E
char[]	166,359	17,639,112	>= 17,639,2E
java.util.HashMap\$Entry	117,958	2,830,992	>= 46,496,44
byte[]	54,042	42,619,192	>= 42,619,2E
com.ibm.websphere.samples.pbw	50,000	800,000	>= 39,200,0E

_____ Type **“com.ibm.websphere.samples.pbw.*”** 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> .

Class Name	Objects	Shallow Heap	Retained Heap
com.ibm.websphere.samples.pbw.*	<Numeric>	<Numeric>	<Numeric>
byte[]	54,042	42,619,192	>= 42,619,2E

heapdump.20130117.005630.5356.0001.phd

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

Class Name	Objects	Shallow Heap	Retained Heap
com.ibm.websphere.samples.pbw.*	<Numeric>	<Numeric>	<Numeric>
com.ibm.websphere.samples.pbw.war.ShoppingBean\$Sh	50,000	800,000	>= 39,200,080
com.ibm.websphere.samples.pbw.jpa.Inventory	13	1,040	>= 862,936
com.ibm.websphere.samples.pbw.war.ProductBean	12	192	>= 863,016
com.ibm.websphere.samples.pbw.ejb.CatalogMgr	5	80	>= 160
com.ibm.websphere.samples.pbw.war.ShoppingBean	1	32	>= 41,787,872
com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$	1	40	>= 848
com.ibm.websphere.samples.pbw.ejb.EJSLocalNSLCatalo	1	24	>= 328
com.ibm.websphere.samples.pbw.ejb.ShoppingCartBean	1	24	>= 504
com.ibm.websphere.samples.pbw.jpa.OrderItem	1	64	>= 176
com.ibm.websphere.samples.pbw.war.ImageServlet	1	16	>= 96
com.ibm.websphere.samples.pbw.jpa.Order	1	120	>= 480
com.ibm.websphere.samples.pbw.jpa.Supplier	1	48	>= 152
com.ibm.websphere.samples.pbw.ejb.ShoppingCartBean	1	16	>= 216
com.ibm.websphere.samples.pbw.ejb.EJSLocalNSFShoppi	1	24	>= 424
com.ibm.websphere.samples.pbw.jpa.OrderItem\$PK	1	24	>= 128

Click the Retained Heap column heading to order the table by the largest retained heap.

heapdump.20130201.085755.9312.0001.phd

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

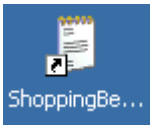
Class Name	Objects	Shallow Heap	Retained Heap
com.ibm.websphere.samples.pbw.	<Numeric>	<Numeric>	<Numeric>
com.ibm.websphere.samples.pbw.war.ShoppingBean	1	32	>= 49,867,872
com.ibm.websphere.samples.pbw.war.ShoppingBean\$Shoppin	60,000	960,000	>= 47,040,000
com.ibm.websphere.samples.pbw.war.ProductBean	12	192	>= 863,016
com.ibm.websphere.samples.pbw.jpa.Inventory	13	1,040	>= 862,936
com.ibm.websphere.samples.pbw.utils.Util	0	0	>= 4,016
com.ibm.websphere.samples.pbw.war.ShoppingBean_\$\$_java:	1	40	>= 848

Note:

In this heap dump there are 60,000 instances of ShoppingContainer with a shallow heap of 9Mb and a retained heap of 47Mb. The cause of this memory leak is becoming increasingly clear, and it seems only the ShoppingContainer and ShoppingBean classes are involved. **The exact statistics in your heap dump may vary.**

Optional Steps:

_____ Double click the desktop shortcut for ShoppingBean.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".

Note:

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 750 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;  
    ShoppingContainer() {  
        array = new byte[750];  
        Arrays.fill(array, (byte) 65);  
    }  
}
```

Part 8: (Optional) Using the IBM Extensions to Memory Analyzer for Further Memory Analysis

This part of the lab as the title indicates is optional. However, this is available for you to do as time permits.

Note:

“**IBM Extensions for Memory Analyzer**” have been installed into ISA. These were initially released by IBM via the **IBM Alphaworks** site. They are now integrated with ISA 5 and have become a very useful extension 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 the IBM extensions require the additional data only available in a full system dump, as opposed to a Java heap dump.

A **heap dump** is a dump of the state of the Java heap memory. This is very useful for analyzing the use of memory by a Java application, and therefore very useful for diagnosing out of memory issues. A **system dump** consists of all the memory that is being used by the JVM process; this includes the Java heap, along with all JVM and user libraries (native memory). System dumps also contain **field values** of Java objects. System dumps are often used in the diagnosis of other Java problem scenarios such as crashes, as they show detailed information about the state of the JVM. Because a system dump contains all of the memory allocated by the JVM process, system dump files can be very large.

In previous versions of the IBM SDK shipped with WebSphere Application Server, it was necessary to post-process a system core file using a command called jextract before the system core could be used by tools such as Memory Analyzer. However, newer SDKs do not require this jextract step. For reference, the IBM SDKs **1.6.0 SR9** or later, or any version of **SDK 1.7**, no longer require jextract to be run. A technote in the references sections correlates the WebSphere Application Server versions to the SDK shipped. This shows the SDKs which do not require jextract were shipped with WebSphere Application Server 7.0.0.15 and above.

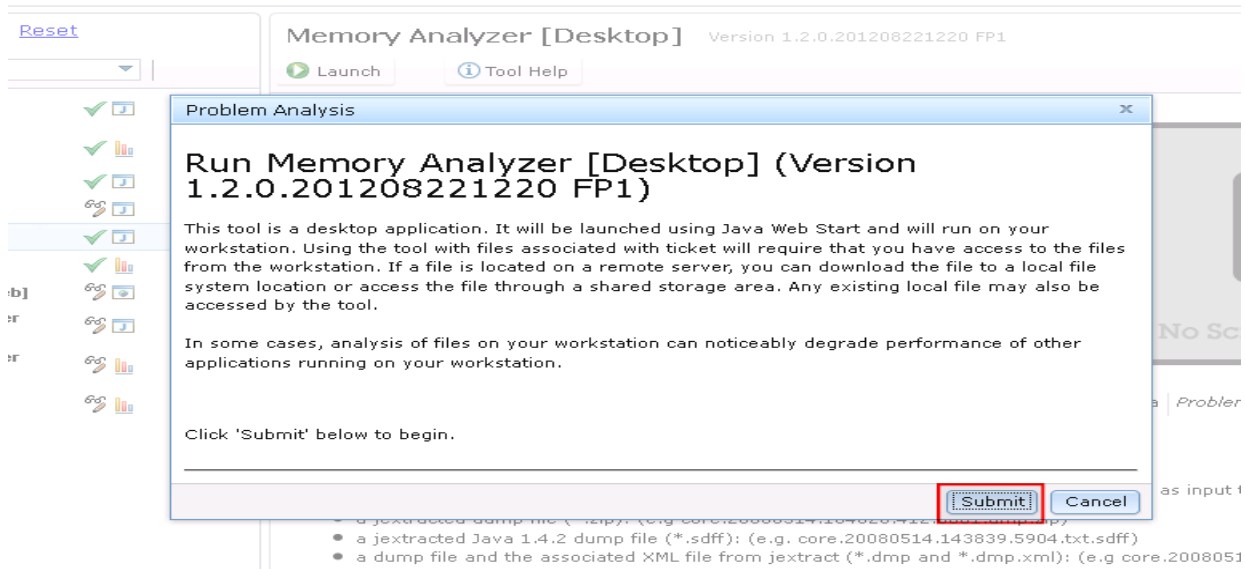
To save time, for this section of the lab, a system dump is provided in “**E:\Impact Lab Files\IBM Extensions**”. The dump was generated as a result of an **out of memory** condition.

_____ Close the browser tab showing the Memory Analyzer web edition, where you analyzed the IBM heap dump.

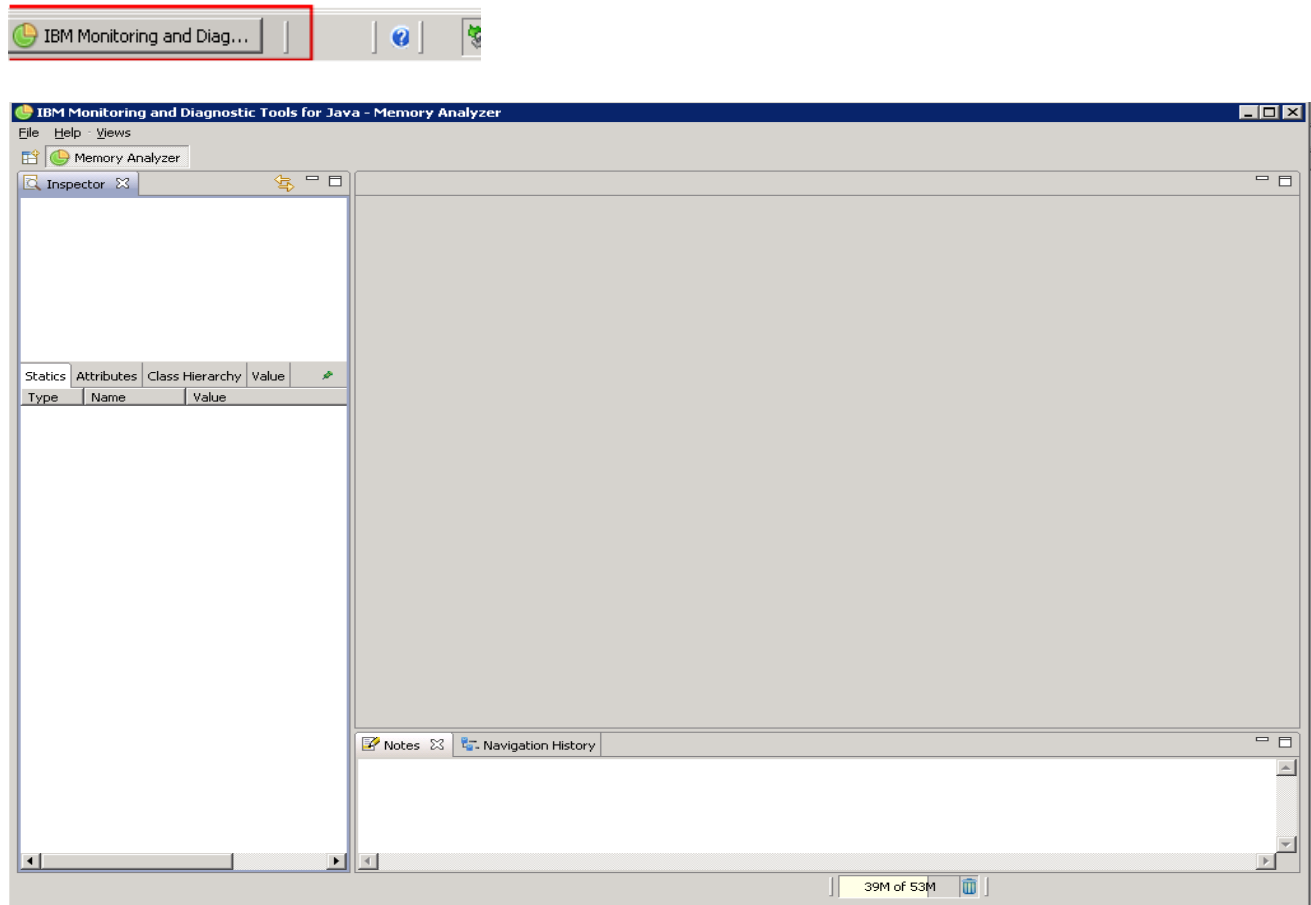
_____ In the ISA tab, click **Tools**, then select **Memory Analyzer (Desktop)**

The screenshot shows the IBM Support Assistant web interface. At the top, there are browser tabs for 'WebSphere Integrated Solutions Console', '0003 - IBM Support Assistant 5.0', and 'Plants By WebSphere Shopping'. The address bar shows 'localhost:10911/isa5/index.html#view=tools&id=0003&toolId=com.ibm.java.diagnostics.memory.analyzer.jws'. The main header is 'IBM Support Assistant'. Below it, a search bar contains '[0003] PlantsByWebSphere OOM'. A navigation bar includes 'Files', 'Tools', 'Reports', 'Overview', 'Symptoms', 'Global Knowledge Base Matches', and 'Data Collector'. The main content area is divided into two columns. The left column is a search results list with a search bar 'Enter keyword' and filters 'Filter Reset', 'Sort By: a-z', and 'Tag: All Tags'. The list includes items like 'Garbage Collection and Memory Visualizer (GCMV) [Desktop]', 'Health Center', 'HeapAnalyzer [Desktop]', 'Memory Analyzer [Desktop]', 'Memory Analyzer [Report]', 'Memory Analyzer Web Edition [Web]', 'Thread and Monitor Dump Analyzer (TMDA) [Desktop]', 'Thread and Monitor Dump Analyzer (TMDA) [Report]', and 'WebSphere Application Server Configuration Visualizer'. The 'Memory Analyzer [Desktop]' item is highlighted. The right column shows the details for 'Memory Analyzer [Desktop] Version 1.2.0.2012', with 'Launch' and 'Tool Help' buttons. The description reads: 'IBM Monitoring and Diagnostic Tools for Java(TM) - Memory Analyzer. Memory Analyzer is a feature-rich Java heap analyzer that helps you find memory leaks and reduce memory consumption. This tool is provided in three versions: as a report generating version that reads the dump, builds some index files and generates some zipped HTML reports. as an interactive GUI version running on the desktop. an interac... (more)'

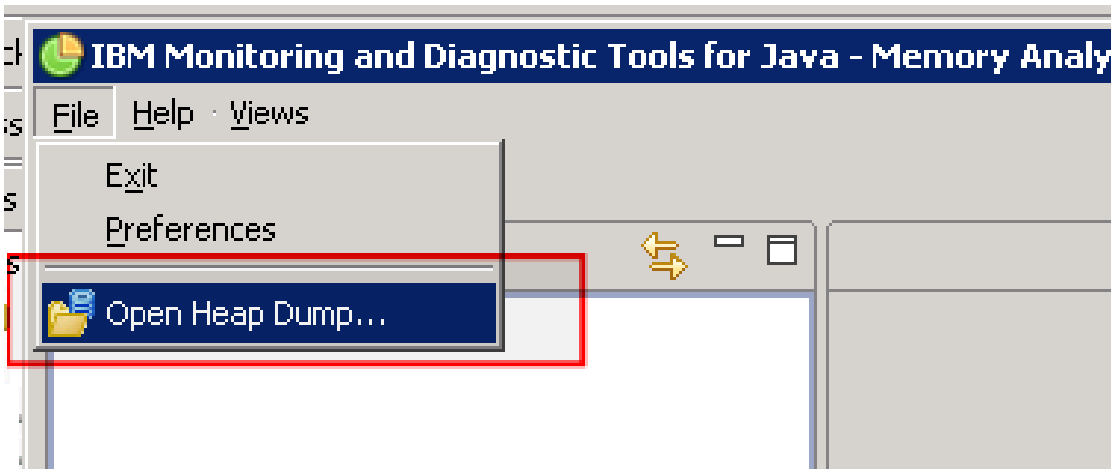
Click **Launch**, and then at the **Run Memory Analyzer[Desktop]** version pop window as shown below Click **Submit**.



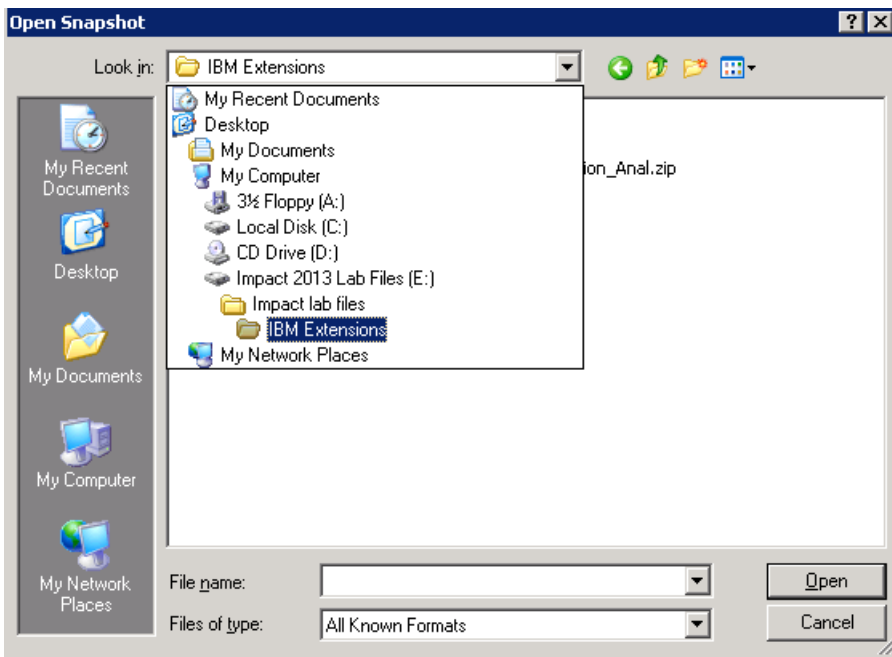
Wait a few minutes for the Memory Analyzer Desktop tool to launch. Once launched you will see the following window. (If the window is not visible, select the **IBM Monitoring and Diagnostic Tool** on the taskbar at the bottom of the windows desktop to bring up the tool.)



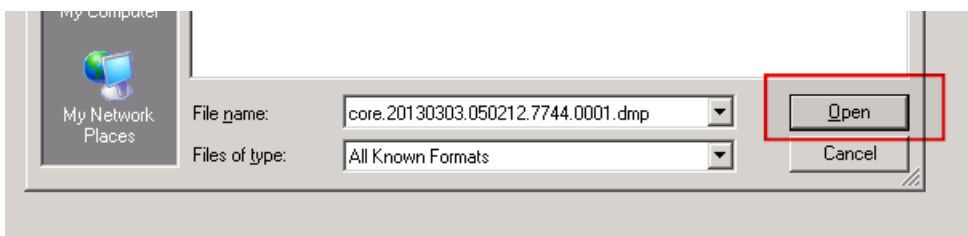
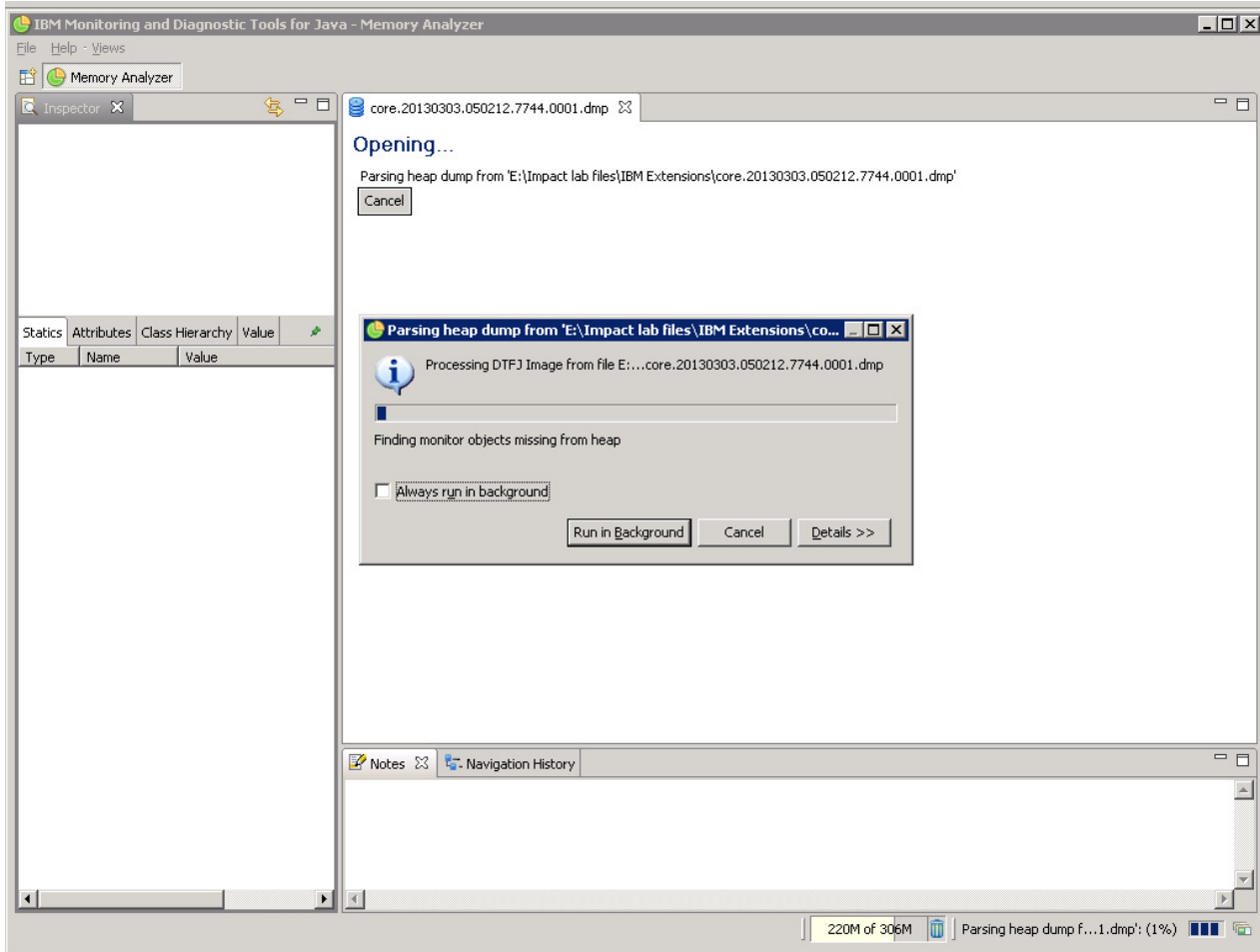
Click **File->Open heap dump**.



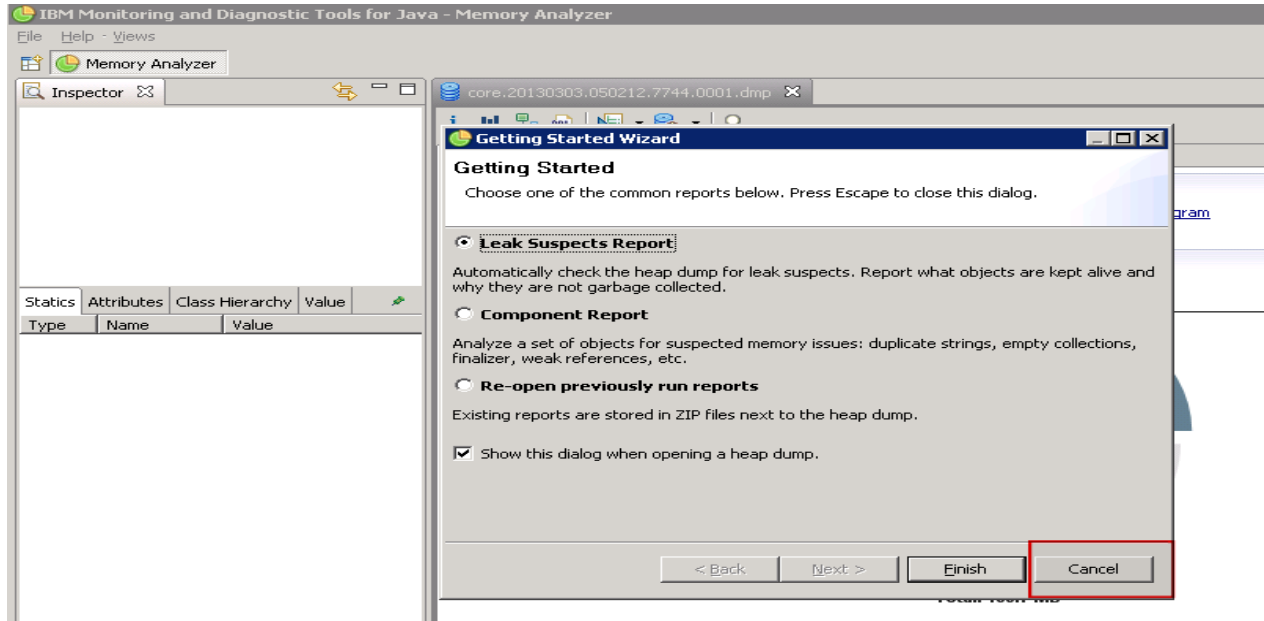
Navigate to the system dump in **E:\Impact Lab Files\IBM Extensions**.



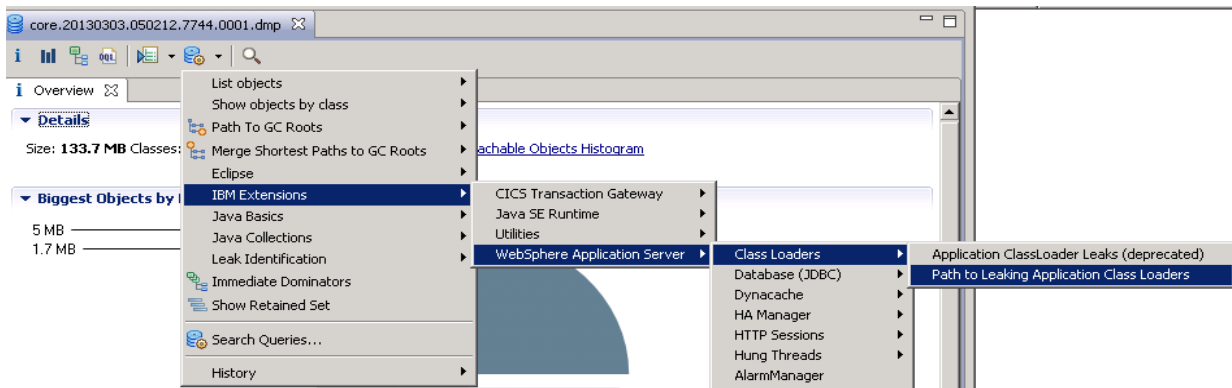
Select the dump file labeled **core.20130303.050212.7744.0001.dmp** and click **Open**. Please be patient, parsing a system dump can take some time. **Approximately 3-5 minutes may pass with little nothing reported by the progress bar.**



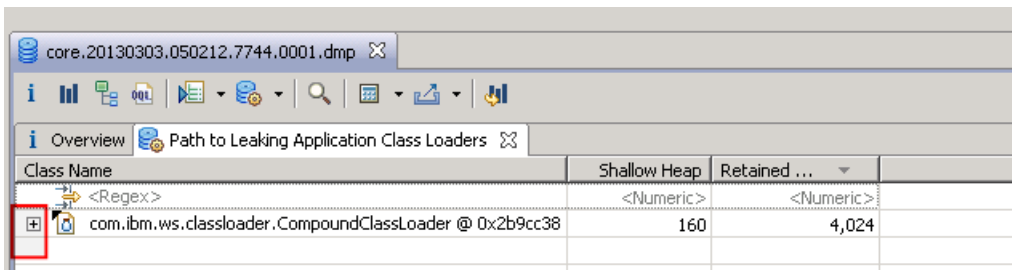
_____ On the “Getting Started Wizard” box, click **Cancel** to avoid generating the standard reports.



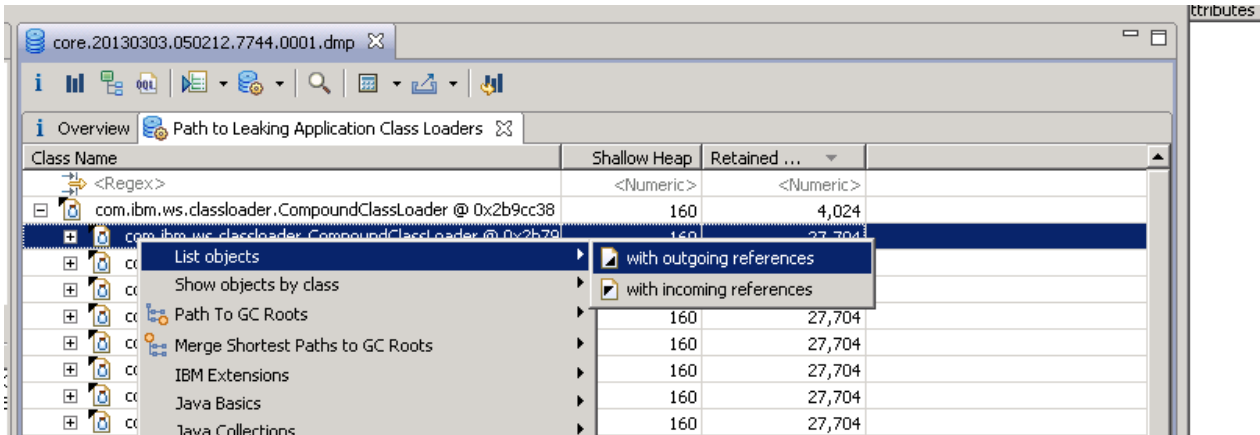
Next, Click the **reports** icon  and select **IBM Extensions->WebSphere Application Server->Class Loaders->Path to Leaking Application Class loader** as shown below.



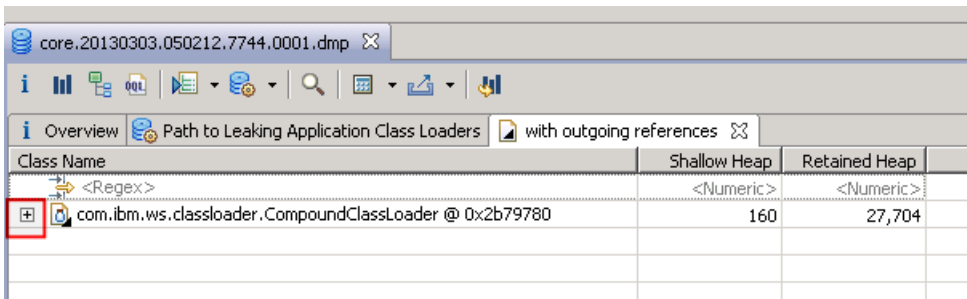
On the “Path to leaking Application Class Loaders” window, expand the classloader below by clicking on the “+” as shown below.



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




Expand the class loader references by clicking on the “+” as shown below.



The resulting objects loaded by this class loader relates to the plants sample class (com.ibm.websphere.samples.*) which gives a clear indication of the application that is leaking memory as shown below.

Class Name	Shallow Heap	Retained Heap
com.ibm.ws.classloader.CompoundClassLoader @ 0x2b79780	160	27,704
<class> class com.ibm.ws.classloader.CompoundClassLoader @ 0x12e0e	19,097	19,793
class com.ibm.websphere.samples.pbw.jpa.OrderItem @ 0x26251e0	414	430
class com.ibm.websphere.samples.pbw.jpa.OrderItem\$PK @ 0x2629a30	299	315
class com.ibm.websphere.samples.pbw.jpa.Inventory @ 0x2629b80	531	547
class com.ibm.websphere.samples.pbw.jpa.Order @ 0x2629be8	689	705
class com.ibm.websphere.samples.pbw.jpa.BackOrder @ 0x262c128	332	348
class com.ibm.websphere.samples.pbw.jpa.Customer @ 0x262ce50	312	328
class com.ibm.websphere.samples.pbw.jpa.Supplier @ 0x2630dd0	249	265
annotationCache java.util.Hashtable @ 0x2998538	48	6,208
packages java.util.Hashtable @ 0x2998568	48	128
packageSigners java.util.Hashtable @ 0x2998598	48	152
methodCache java.util.Hashtable @ 0x29985c8	48	104
fieldCache java.util.Hashtable @ 0x29985f8	48	8,264
constructorCache java.util.Hashtable @ 0x2998628	48	104
assertionLock java.lang.ClassLoader\$AssertionLock @ 0x2b83788	16	16
lazyInitLock java.lang.ClassLoader\$LazyInitLock @ 0x2b83798	16	16
pds java.util.HashMap @ 0x2b837a8	48	144
localClassPath java.lang.String @ 0x2b837d8 C:\IBM\WebSphere\App5	24	496
nativelibpaths java.lang.String[0] @ 0x2b837f0	16	16
libraryClassLoaders com.ibm.ws.classloader.CompoundClassLoader[0]	16	16
providers com.ibm.ws.classloader.SinglePathClassProvider[2] @ 0x2b83808	16	1,768
reloadableParents java.util.Vector @ 0x2b83820	24	72
resourceRequestCache java.util.Collections\$SynchronizedMap @ 0x2b83830	16	3,360
preDefinePlugins java.util.ArrayList @ 0x2b83848	24	72

Next, Click the reports icon  again and select **IBM Extensions->WebSphere Application Server->WAS Cache Analysis**.

Class Name	Shallow Heap	Retained Heap
<Numeric>	<Numeric>	<Numeric>
	160	27,672
	21,834	22,514
annotationCache	48	104
packages	8	8
packageSigners	8	8
methodCache	48	144
fieldCache	24	496
constructorCache	16	16
assertionLock	16	16
lazyInitLock	16	1,768
pds	24	72
localClassPath	16	3,360
nativelibpaths	24	72
libraryClassLoaders	24	120
providers	398	414
reloadableParents	283	299
resourceRequestCache	515	531
preDefinePlugins	673	689
name	316	332
class com.ibm.websphere.samples.pbw.jpa.OrderItem @	296	312
class com.ibm.websphere.samples.pbw.jpa.OrderItem\$PK	160	4,008
class com.ibm.websphere.samples.pbw.jpa.Inventory @		
class com.ibm.websphere.samples.pbw.jpa.Order @		
class com.ibm.websphere.samples.pbw.jpa.BackOrder @		
class com.ibm.websphere.samples.pbw.jpa.Customer @		
parent, parent		
Total: 25 of 26 entries		

WebSphere Cache Analysis

▼ WAS Caches Found

Unknown Column: address	Unknown Column: cacheName	Unknown Column: currentCacheSize	Unknown Column: cacheSizeLimit	Unknown Column: diskOffload	Unknown Column: shallowHeap	Unknown Column: retainedHeap
0x2158c60	com.ibm.ws.wsssecurity.sctClientCacheMap	0	2000	disabled	328	15584
0x1d2e738	com.ibm.workplace/ExtensionRegistryCache	12	5000	disabled	328	43808
0x1d28828	com.ibm.ws.wsssecurity.sctServiceCacheMap	0	2000	disabled	328	15584
Σ Total: 3 entries						

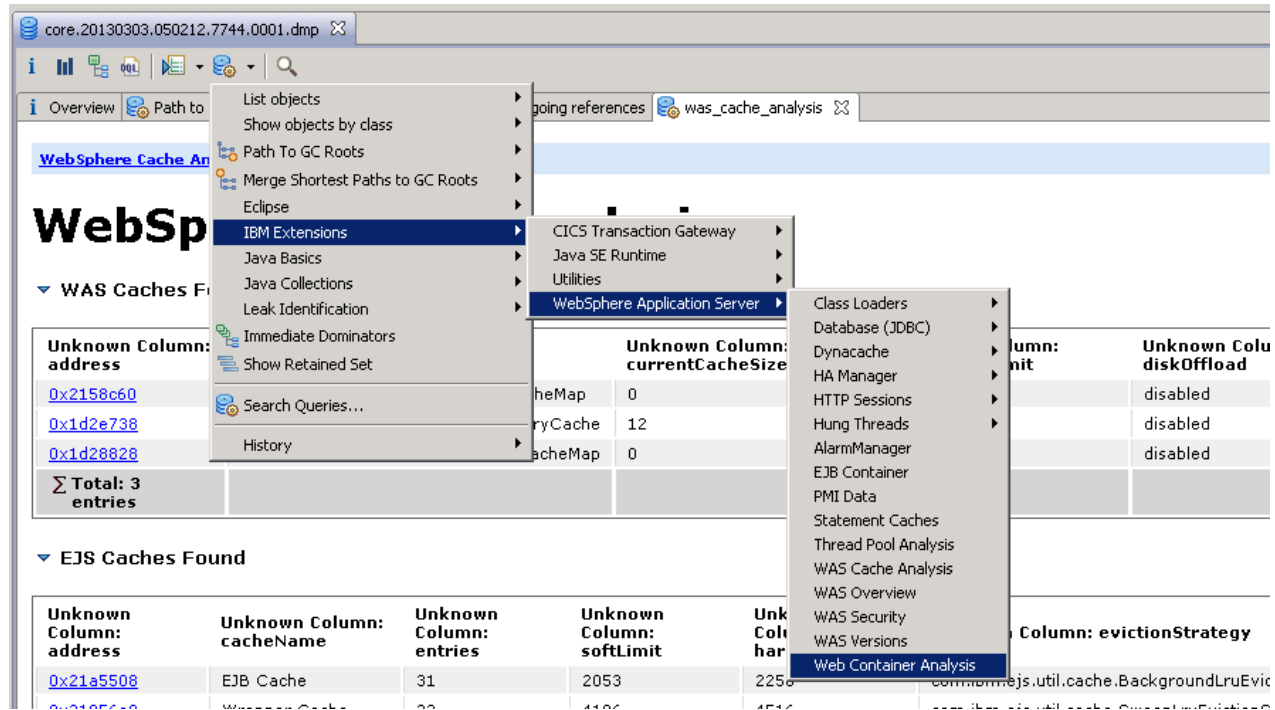
▼ EJS Caches Found

Unknown Column: address	Unknown Column: cacheName	Unknown Column: entries	Unknown Column: softLimit	Unknown Column: hardLimit	Unknown Column: evictionStrategy	Unknown Column: shallowHeap	Unknown Column: retainedHeap
0x21a5508	EJB Cache	31	2053	2258	com.ibm.ejs.util.cache.BackgroundLruEvictionStrategy	64	170064
0x21856e8	Wrapper Cache	33	4106	4516	com.ibm.ejs.util.cache.SweepLruEvictionStrategy	64	114792
Σ Total: 2 entries							

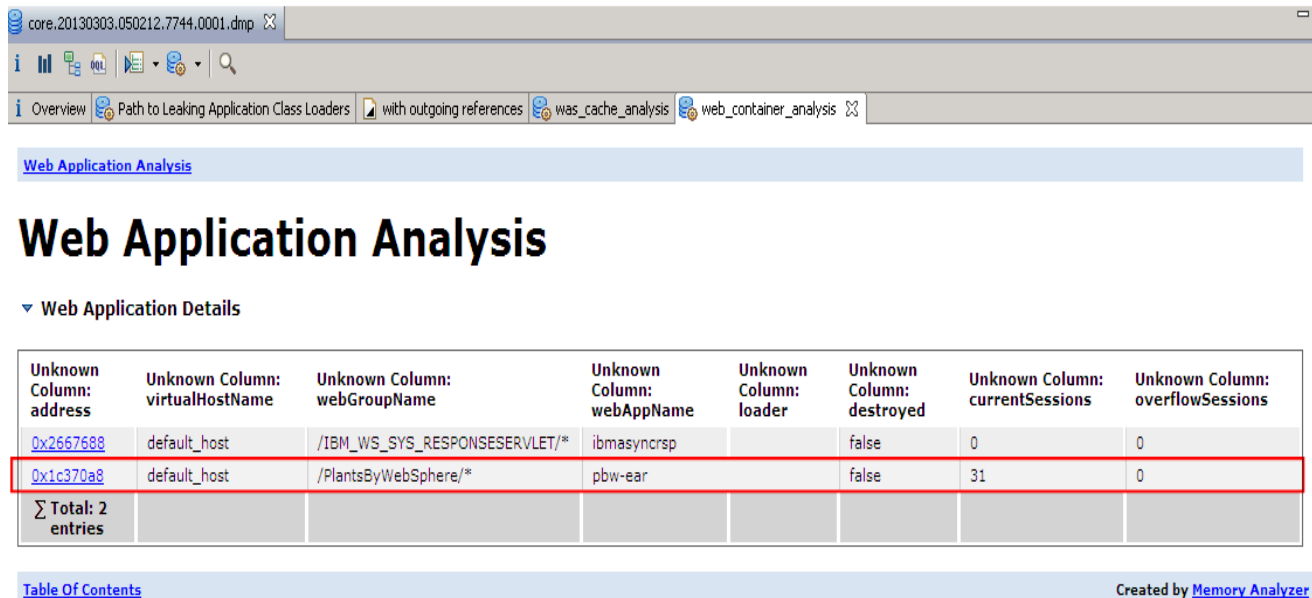
[Table Of Contents](#) Created by [Memory Analyzer](#)


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 off loaded to disk. For this lab, there is no action required.

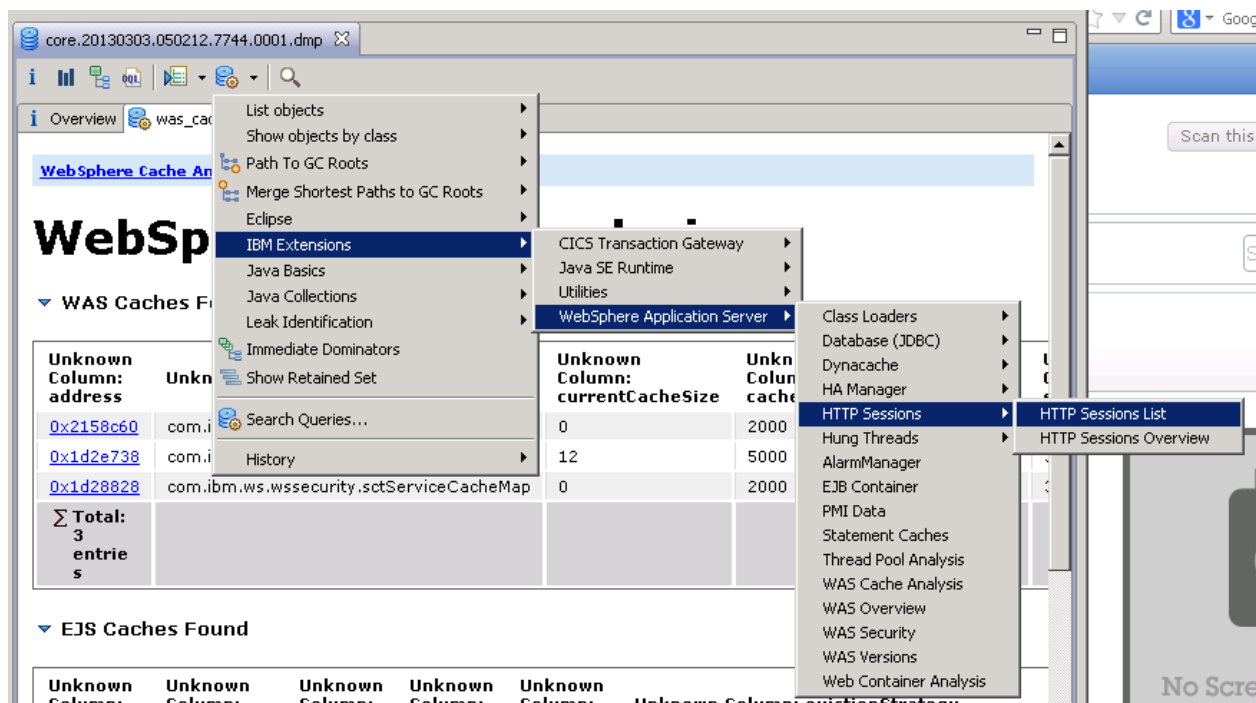
Click the  and select **IBM Extensions->WebSphere Application Server->Web Container Analysis**.



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.



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



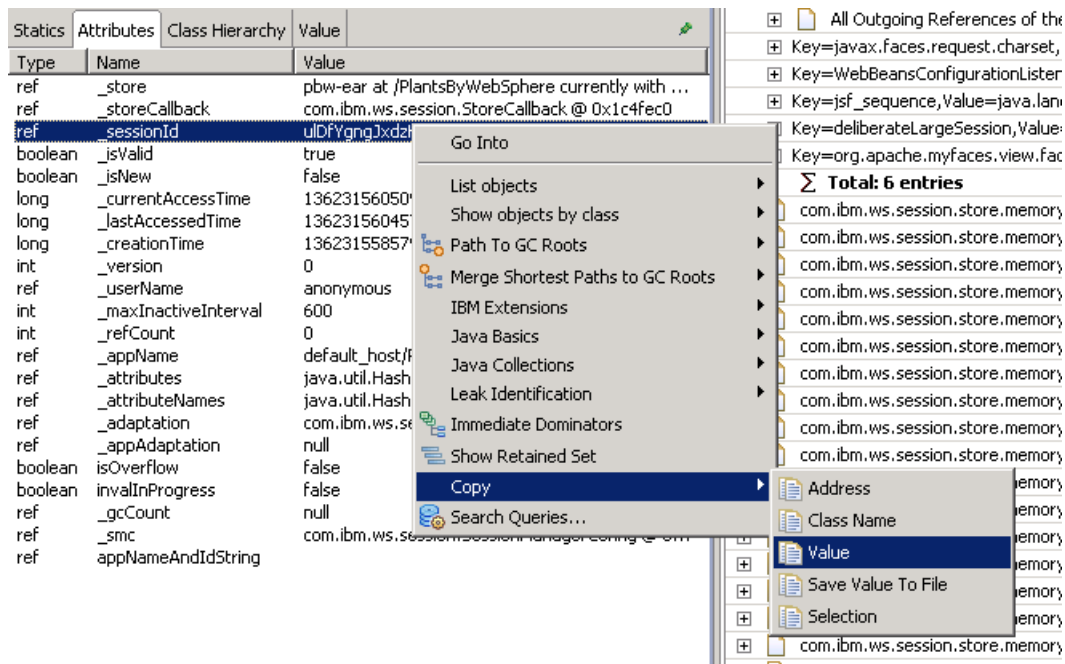
Ensure the table of sessions is ordered by “Retained Heap”. **(Click on the Retained Heap Column to order it as shown below)** Notice how some or most sessions for PlantsByWebSphere are around 23k. while others are ranging from 9K to 24k.

Class Name	Shallow Heap	Retained ...	AppName	SessionID
<Regex>	<Numeric>	<Numeric>	<Regex>	<Regex>
com.ibm.ws.session.store.memory.MemorySession @ 0x3580a80 for memory store pbw-	112	9,912	default_host/PlantsByWebSphere	oTrm4G35OpTWY7-26
com.ibm.ws.session.store.memory.MemorySession @ 0x72bce88 for memory store pbw-	112	16,448	default_host/PlantsByWebSphere	QQxCH3F0185xFZqD:

Class Name	Shallow Heap	Retained ...	AppName	SessionID
<Regex>	<Numeric>	<Numeric>	<Regex>	<Regex>
com.ibm.ws.session.store.memory.MemorySession @ 0x3580a80 for memory store pbw-i	112	9,912	default_host/PlantsByWebSphere	oTrm4G35Op
com.ibm.ws.session.store.memory.MemorySession @ 0x72bce88 for memory store pbw-i	112	16,448	default_host/PlantsByWebSphere	QQxCH3FO1
com.ibm.ws.session.store.memory.MemorySession @ 0x72bb850 for memory store pbw-i	112	17,128	default_host/PlantsByWebSphere	6Coa3Tqkah
com.ibm.ws.session.store.memory.MemorySession @ 0x403f320 for memory store pbw-i	112	23,920	default_host/PlantsByWebSphere	ulDFYngJxd
com.ibm.ws.session.store.memory.MemorySession @ 0x3d2c1e8 for memory store pbw-i	112	23,928	default_host/PlantsByWebSphere	TyeQA0Jv0i
com.ibm.ws.session.store.memory.MemorySession @ 0x4aa09b8 for memory store pbw-i	112	23,936	default_host/PlantsByWebSphere	OcblxydKlUE
com.ibm.ws.session.store.memory.MemorySession @ 0x35828e8 for memory store pbw-i	112	23,936	default_host/PlantsByWebSphere	sR116Ald-Iun
com.ibm.ws.session.store.memory.MemorySession @ 0x4aa0930 for memory store pbw-i	112	23,944	default_host/PlantsByWebSphere	H3_r_LBIqsi
com.ibm.ws.session.store.memory.MemorySession @ 0x324b958 for memory store pbw-i	112	23,944	default_host/PlantsByWebSphere	8XBjJhn7cw
com.ibm.ws.session.store.memory.MemorySession @ 0x30e7890 for memory store pbw-i	112	23,944	default_host/PlantsByWebSphere	zmMvtCpSSn
com.ibm.ws.session.store.memory.MemorySession @ 0x2d910f0 for memory store pbw-i	112	23,944	default_host/PlantsByWebSphere	iHFQuOPMMc
com.ibm.ws.session.store.memory.MemorySession @ 0x4fd82c8 for memory store pbw-i	112	23,952	default_host/PlantsByWebSphere	oBD87-YM_E
com.ibm.ws.session.store.memory.MemorySession @ 0x3582958 for memory store pbw-i	112	23,952	default_host/PlantsByWebSphere	eWNf7gd1d
com.ibm.ws.session.store.memory.MemorySession @ 0x39f52b8 for memory store pbw-i	112	23,960	default_host/PlantsByWebSphere	XjD4623-Dxt
com.ibm.ws.session.store.memory.MemorySession @ 0x324b9c8 for memory store pbw-i	112	23,960	default_host/PlantsByWebSphere	CYzg8ppWt
com.ibm.ws.session.store.memory.MemorySession @ 0x2d91080 for memory store pbw-i	112	23,960	default_host/PlantsByWebSphere	uyWl9wRRaf
com.ibm.ws.session.store.memory.MemorySession @ 0x60f1170 for memory store pbw-i	112	23,968	default_host/PlantsByWebSphere	5GRB8Y006i
com.ibm.ws.session.store.memory.MemorySession @ 0x5a62d10 for memory store pbw-i	112	23,968	default_host/PlantsByWebSphere	Cm2dHesirYz
com.ibm.ws.session.store.memory.MemorySession @ 0x347caa8 for memory store pbw-i	112	23,968	default_host/PlantsByWebSphere	xB_bILFUQqi
com.ibm.ws.session.store.memory.MemorySession @ 0x3250078 for memory store pbw-i	112	23,968	default_host/PlantsByWebSphere	KMUDH5sil_c
com.ibm.ws.session.store.memory.MemorySession @ 0x30e7918 for memory store pbw-i	112	23,968	default_host/PlantsByWebSphere	6_npVsglsDK
com.ibm.ws.session.store.memory.MemorySession @ 0x60f10e8 for memory store pbw-i	112	23,976	default_host/PlantsByWebSphere	4V8ui_vinVW
com.ibm.ws.session.store.memory.MemorySession @ 0x5a62d80 for memory store pbw-i	112	23,976	default_host/PlantsByWebSphere	hAKk71U_Tz
com.ibm.ws.session.store.memory.MemorySession @ 0x4fd8ec8 for memory store pbw-i	112	23,976	default_host/PlantsByWebSphere	-lxq38lbGmk7
com.ibm.ws.session.store.memory.MemorySession @ 0x392d720 for memory store pbw-i	112	23,976	default_host/PlantsByWebSphere	bVdVTLSPR7
com.ibm.ws.session.store.memory.MemorySession @ 0x44c1fd0 for memory store pbw-i	112	23,992	default_host/PlantsByWebSphere	iWTKzD?VnF

Let's Highlight and then expand one of the **23k** session objects. (For this example, the first 23k in the ordered list was picked as shown below)

Class Name	Shallow Heap	Retained ...	AppName	SessionID
<Regex>	<Numeric>	<Numeric>	<Regex>	<Regex>
com.ibm.ws.session.store.memory.MemorySession @ 0x3580a80 for memory store pbw-i	112	9,912	default_host/PlantsByWebSphere	oTrm4G35OpTwy7-26XF
com.ibm.ws.session.store.memory.MemorySession @ 0x72bce88 for memory store pbw-i	112	16,448	default_host/PlantsByWebSphere	QQxCH3FO185xFZqJxKE
com.ibm.ws.session.store.memory.MemorySession @ 0x72bb850 for memory store pbw-i	112	17,128	default_host/PlantsByWebSphere	6Coa3TqkahZImXkCNvst
com.ibm.ws.session.store.memory.MemorySession @ 0x403f320 for memory store pbw-i	112	23,920	default_host/PlantsByWebSphere	ulDFYngJxdzHLnmXpw8l
com.ibm.ws.session.store.memory.MemorySession @ 0x3d2c1e8 for memory store pbw-i	112	23,928	default_host/PlantsByWebSphere	TyeQA0Jv0cW6_vjKXVt
com.ibm.ws.session.store.memory.MemorySession @ 0x4aa09b8 for memory store pbw-i	112	23,936	default_host/PlantsByWebSphere	OcblxydKlUEupmQFct6D
com.ibm.ws.session.store.memory.MemorySession @ 0x35828e8 for memory store pbw-i	112	23,936	default_host/PlantsByWebSphere	sR116Ald-IunE1azRnnt5c
com.ibm.ws.session.store.memory.MemorySession @ 0x4aa0930 for memory store pbw-i	112	23,944	default_host/PlantsByWebSphere	H3_r_LBITncDM vk5350I

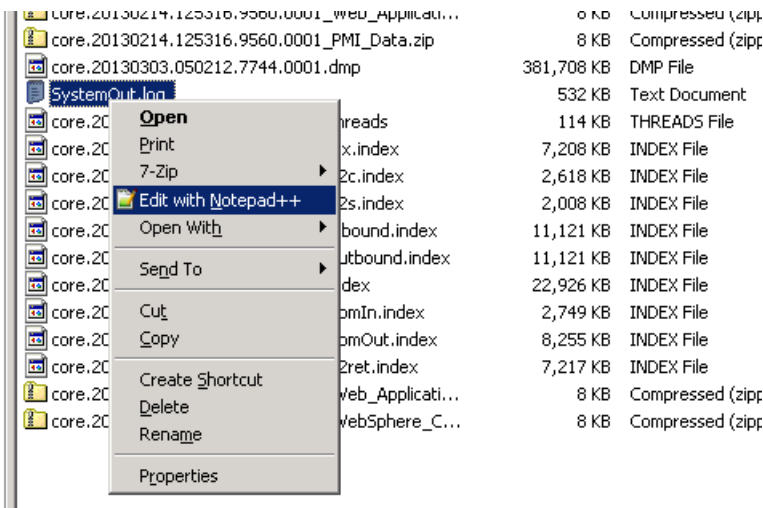


A possible way to relate this unusually large session to the application code is to search the log files for the session ID. If the application uses this in its logging, it may be possible to determine what the user did to cause the large session.

Note:

This lab provides a SystemOut.log file that correlates with the time the system dump was generated. This will be used to search for more details about the large session size.

Launch Windows Explorer. Navigate to and right click file: **E:\Impact Lab Files\IBM Extensions\SystemOut.log**, then select **Edit with Notepad++** as shown below.

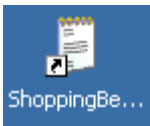


_____ After Notepad++ opens the systemOut.log file. Select “**Ctrl + F**” or Click **Search >Find** and search the file for the Session ID you identified or copied earlier from the core dump. You should be able to copy the id from your clipboard into the search window. It should then reveal a log statement that gives a clue about the application’s actions as shown below.

```
[3/3/13 4:59:58:718 PST] 0000009e SystemOut      O performProductDetail : itemID=F0018
[3/3/13 4:59:58:734 PST] 0000009e SystemOut      O ==> STARTING DELIBERATE LARGE SESSION for ID=ulDfYgngJxdzHLnmXpw8Pw6
[3/3/13 4:59:58:734 PST] 0000009e SystemOut      O ==> ENDING DELIBERATE LARGE SESSION
[3/3/13 4:59:59:171 PST] 00000098 SystemOut      O performProductDetail : itemID=F0015
[3/3/13 4:59:59:718 PST] 0000009e SystemOut      O performProductDetail : itemID=F0015
```

Optional Steps:

_____ Double click the desktop shortcut as shown below for **ShoppingBean.java**



_____ Click **Edit->Find** 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.

```
private void deliberateLargeSession(ExternalContext externalContext) {
    // -----
    // User clicked on the White Poinsettia, let's make the session for this user
    // blooming big.
    // -----
    System.out.println("=> STARTING DELIBERATE LARGE SESSION for ID="+((HttpSession)externalContext.getSession(true)).getId());

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

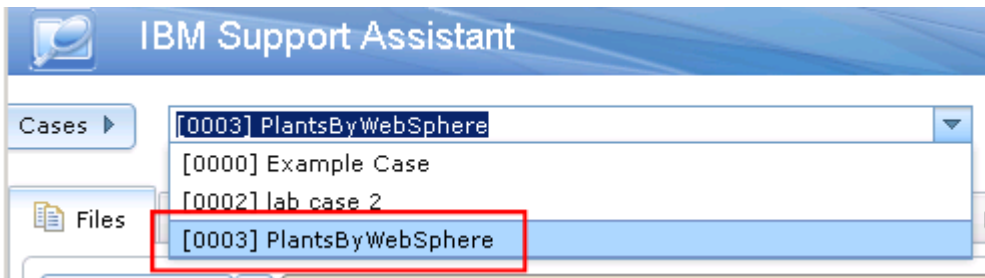
    System.out.println("=> ENDING DELIBERATE LARGE SESSION");
}
```

Part 9: (Optional) Using the WebSphere Application Server Configuration Visualizer

This part of the lab as the title indicates is optional. However, this is available for you to do as time permits.

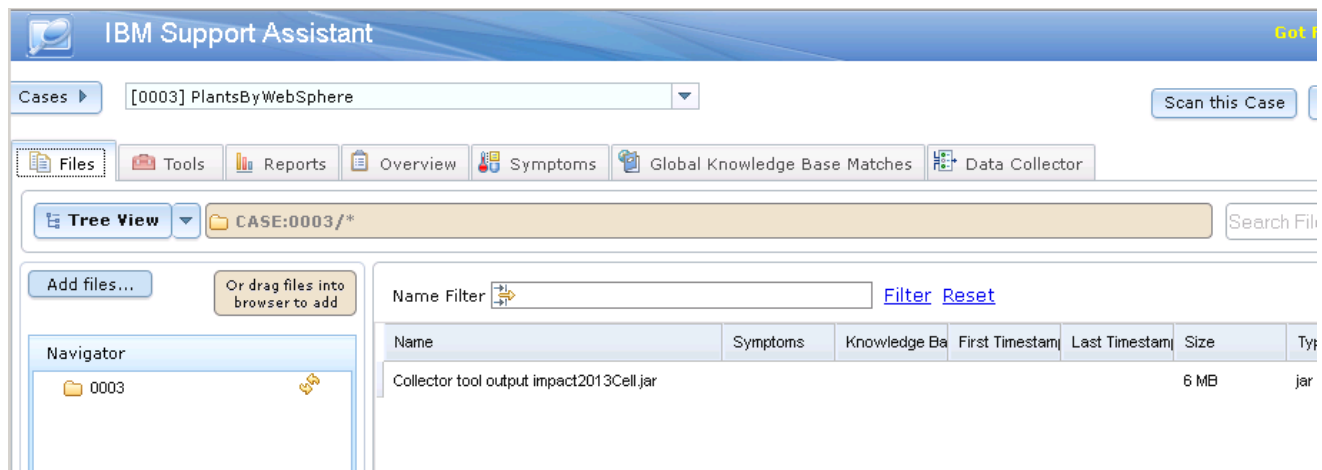
_____ Launch the browser and use the bookmarks to load the ISA web interface.

_____ To demonstrate one of the report generator tools, first switch to the “**PlantsByWebSphere**” case that has been prepared for this lab, and click the **Files** tab.

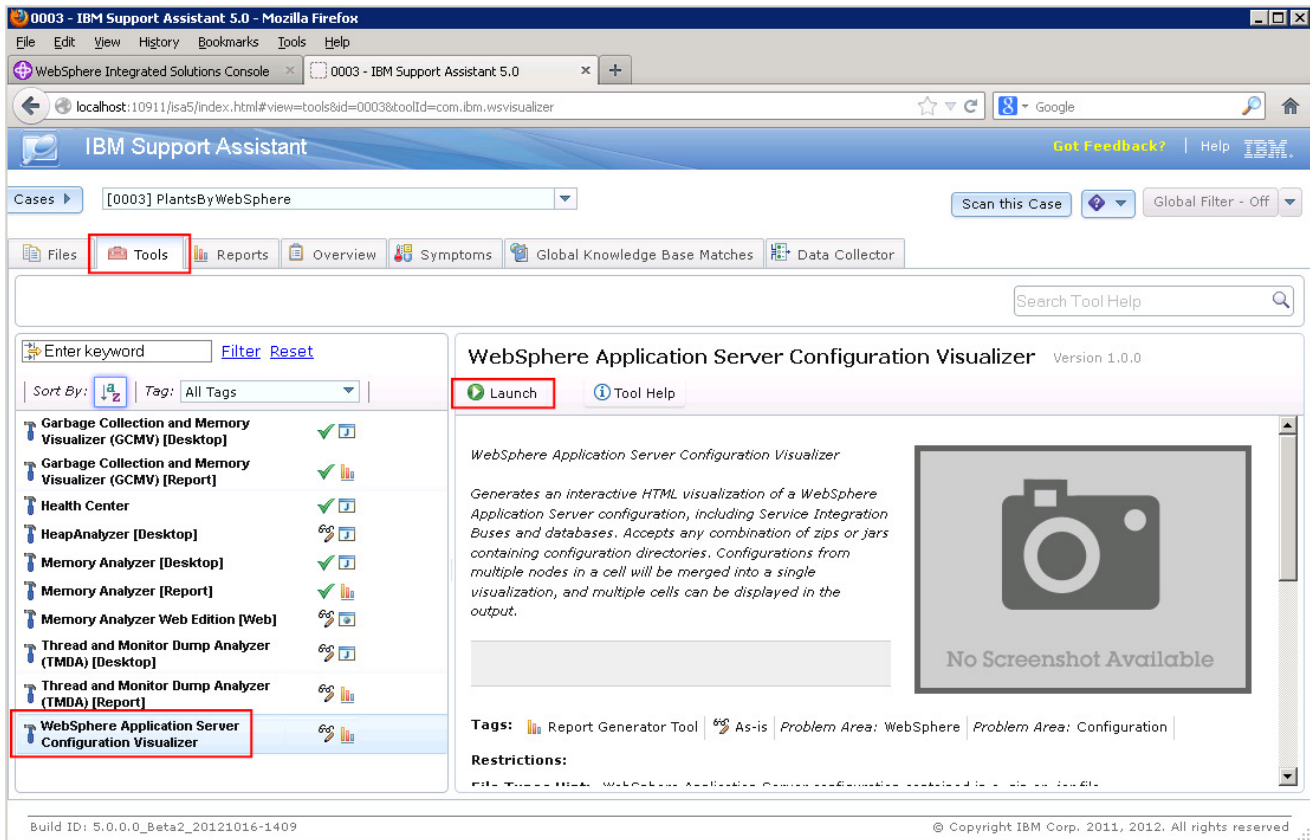


Note:

The case contains file “**Collector tool output impact2013Cell.jar**” as shown below. This is the output from the WebSphere collector tool which collects information about the WebSphere Application Server configuration – it is frequently requested by IBM support when investigating PMRs.



_____ To demonstrate one of the report generator tools, return to the Tools tab and select **WebSphere Application Server Configuration Visualizer** as shown below. Then press the launch button.



_____ Press the Browse button from the above window to select the input files.

Problem Analysis
✕

Run WebSphere Application Server Configuration Visualizer (Version 1.0.0)

Input Files and Folders *

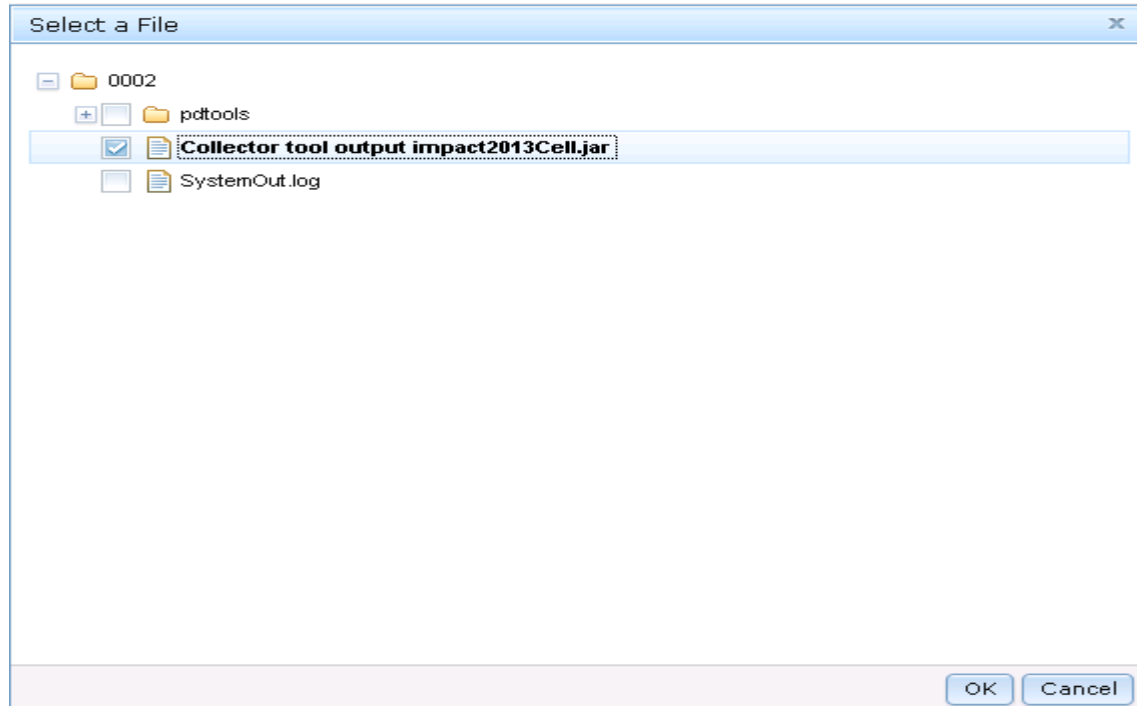
Browse

Parameters

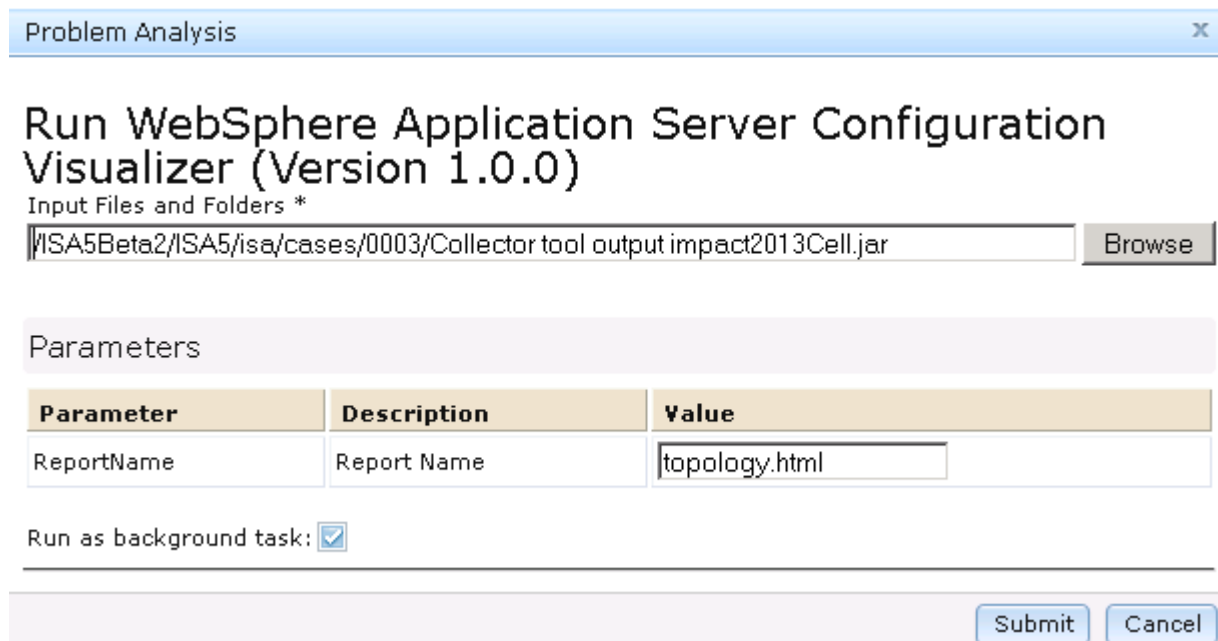
Parameter	Description	Value
ReportName	Report Name	<input style="width: 80%;" type="text" value="topology.html"/>

Run as background task:

Submit
Cancel

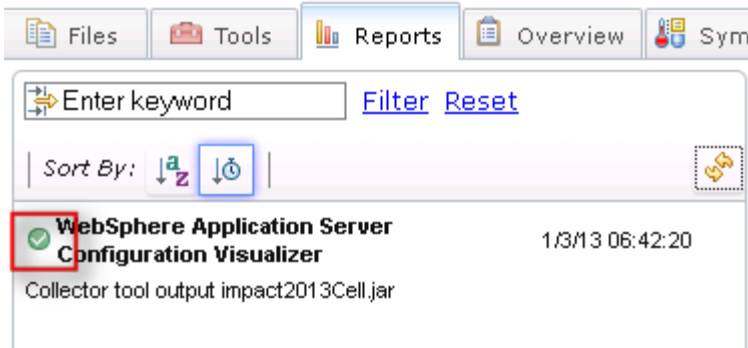
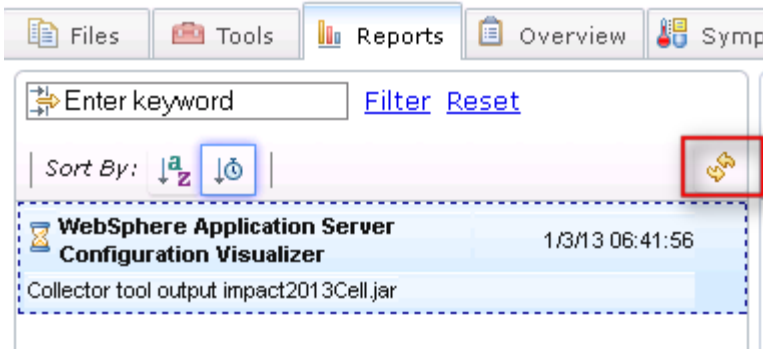


_____ Select the file **Collector tool output impact2013Cell.jar** as shown above and press OK.

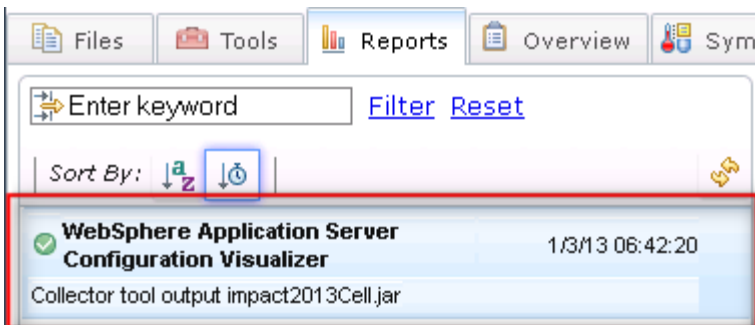


_____ Click **Submit**

Click on the **Reports** tab, then click the **Refresh** icon a few times until the report is complete, as indicated by a green tick icon.



Select the completed report.



Note:

Take a look at the report in the right hand window. This visualizes the topology of WebSphere Application Server. For example, in this lab the topology consists of a single cell containing two nodes – one containing the deployment manager, and the second containing two application servers. Those servers are part of a cluster called “**PlantsByWebSphereCluster**”. In addition, there are several datasources defined in the configuration.

_____ Try clicking on any of the topology components, the report is interactive and will pop up dialogues with more detailed information.

WebSphere Application Server Configuration Visualizer

The screenshot displays the WebSphere Application Server Configuration Visualizer interface. The main area shows a topology diagram for the 'impact2013Cell'. It includes components like 'impact2013CellManager' (containing 'dmgr'), 'impact2013Node' (containing 'nodeagent' and 'webserver'), and a cluster named 'PlantsByWebSphereCluster'. Below the topology, there is a 'Databases' section with several entries:

- DBA: \${USER_INSTALL_ROOT}/databases/EJBTimers/\${SERVER}/EJB...
- DBB: \${WAS_INSTALL_ROOT}/derby/PLANTSDB
- DBC: \${USER_INSTALL_ROOT}/OTIS
- DBD: PLANTSDB
- DBE: \${APP_INSTALL_ROOT}/\${CELL}/DefaultApplication.ear/Def...

A detailed view window for 'server1' is open, showing the following configuration details:

- type: Application server
- name: server1
- clusterName: PlantsByWebSphereCluster
- node: impact2013Node
- Core group: DefaultCoreGroup
- serverType: APPLICATION_SERVER
- version: 2.0
- DynamicCache:**
 - enable: ExternalCacheGroup
 - name: ExternalCacheGroupMe
 - true: EsInvalidator adapterBeanName com.ibm.websphere.ser...

At the bottom left, there is a link: [Show all detail for browser search](#)

Reference Links

- IBM Support Assistant Information and Downloads:
 - <http://www-01.ibm.com/software/support/isa/>
 - [ISA 5: http://www-01.ibm.com/software/support/isa/teamserver.html](http://www-01.ibm.com/software/support/isa/teamserver.html)
- Health Center
 - <http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/>
- Configuring a controller to start on a specific node or deployment manager
 - <http://www-01.ibm.com/support/docview.wss?uid=swg21425281>
- Verify Java SDK version shipped with IBM WebSphere Application Server fix packs:
 - <http://www-01.ibm.com/support/docview.wss?uid=swg27005002>
- WAS 8.5 Infocenter:
 - http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-dist&topic=welcome_nd
- What's new in WebSphere Application Server 8.0 & 8.5
 - http://www.ibm.com/developerworks/websphere/techjournal/1106_alcott/1106_alcott.html
 - http://www.ibm.com/developerworks/websphere/techjournal/1206_alcott/1206_alcott.html