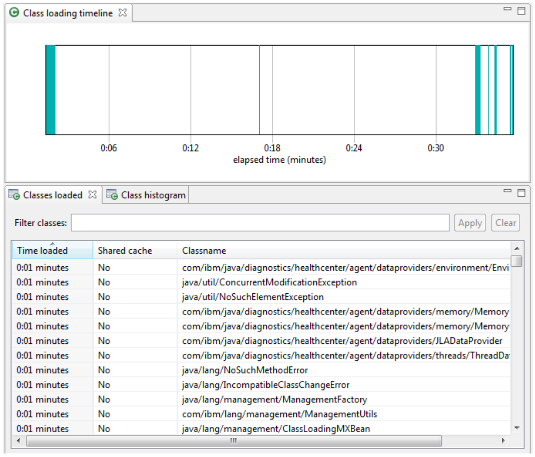
Java VM monitoring and the Health Center API

William Smith will.smith@uk.ibm.com

Health Center overview

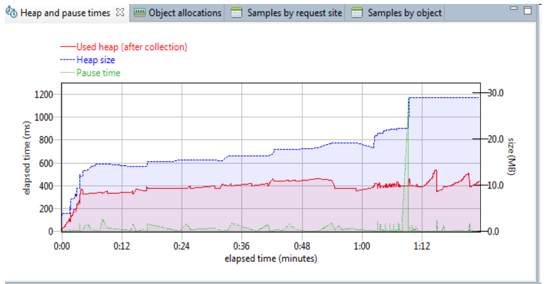
- What problem am I solving?
  - What is my JVM doing? Is everything OK?
  - Why is my application running slowly? Why is it not scaling?
  - Am I using the right options?
- Live monitoring tool with very low overhead (< 1%)
- Understand how your application is behaving
  - Monitor Class loading, File I/O, Environment settings, Garbage Collection, Method Profiling, Locking, Native memory use, Threads
- Diagnose potential problems, with recommendations
- Works at the JVM level
- Suitable for all Java applications
- Powerful API allowing embedding of Health Center into other applications

Health Center overview continued

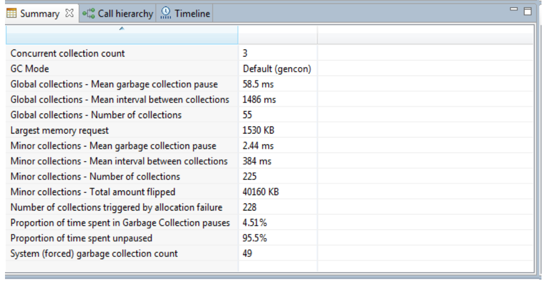


Time loaded	Shared cache	Classname
0:01 minutes	No	com/ibm/java/diagnostics/healthcenter/agent/dataproviders/environment/Envi
0:01 minutes	No	java/util/ConcurrentModificationException
0:01 minutes	No	java/util/NoSuchElementException
0:01 minutes	No	com/ibm/java/diagnostics/healthcenter/agent/dataproviders/memory/Memory
0:01 minutes	No	com/ibm/java/diagnostics/healthcenter/agent/dataproviders/memory/Memory
0:01 minutes	No	com/ibm/java/diagnostics/healthcenter/agent/dataproviders/JLADataProvider
0:01 minutes	No	com/ibm/java/diagnostics/healthcenter/agent/dataproviders/threads/ThreadDa
0:01 minutes	No	java/lang/NoSuchMethodError
0:01 minutes	No	java/lang/IncompatibleClassChangeError
0:01 minutes	No	java/lang/management/ManagementFactory
0:01 minutes	No	com/ibm/lang/management/ManagementUtils
0:01 minutes	No	java/lang/management/ClassLoadingMXBean

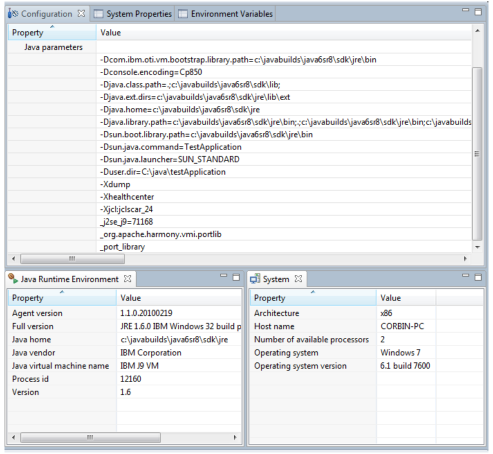
Class loading visualisation
- Shows all loaded classes
- Shows load time
- Identifies shared classes
- Live class histogram information



Garbage Collection visualisation
- Visualises heap usage and gc pause times over time
- Identifies memory leaks
- Suggests command-line and tuning parameters
- Same recommendation logic as GCMV



Concurrent collection count	3	
GC Mode	Default (gencon)	
Global collections - Mean garbage collection pause	58.5 ms	
Global collections - Mean interval between collections	1486 ms	
Global collections - Number of collections	55	
Largest memory request	1530 KB	
Minor collections - Mean garbage collection pause	2.44 ms	
Minor collections - Mean interval between collections	384 ms	
Minor collections - Number of collections	225	
Minor collections - Total amount flipped	40160 KB	
Number of collections triggered by allocation failure	228	
Proportion of time spent in Garbage Collection pauses	4.51%	
Proportion of time spent unpaused	95.5%	
System (forced) garbage collection count	49	



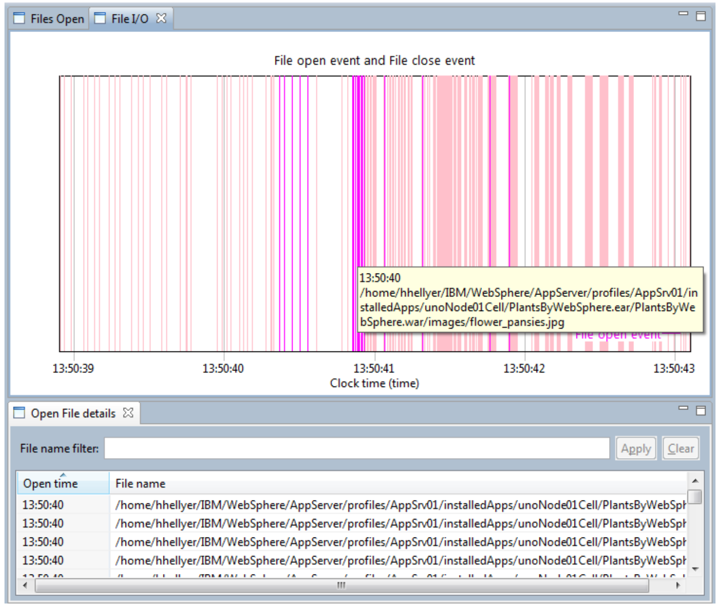
Property	Value
Java parameters	
	-Dcom.ibm.oti.vm.bootstrap.library.path=c:\javabuilds\java6sr8\sdk\jre\bin
	-Dconsole.encoding=Cp850
	-Djava.class.path=.;c:\javabuilds\java6sr8\sdk\lib;
	-Djava.ext.dirs=c:\javabuilds\java6sr8\sdk\jre\lib\ext
	-Djava.home=c:\javabuilds\java6sr8\sdk\jre
	-Djava.library.path=c:\javabuilds\java6sr8\sdk\jre\bin;.;c:\javabuilds\java6sr8\sdk\jre\bin;c:\javabuilds
	-Dsun.boot.library.path=c:\javabuilds\java6sr8\sdk\jre\bin
	-Dsun.java.command=TestApplication
	-Dsun.java.launcher=SUN_STANDARD
	-Duser.dir=C:\java\testApplication
	-Xdump
	-Xhealthcenter
	-Xjcl:jclscar_24
	_j2se_j9=71168
	_org.apache.harmony.vmi.portlib
	_port_library

Property	Value
Agent version	1.1.0.20100219
Full version	JRE 1.6.0 IBM Windows 32 build p
Java home	c:\javabuilds\java6sr8\sdk\jre
Java vendor	IBM Corporation
Java virtual machine name	IBM J9 VM
Process id	12160
Version	1.6

Property	Value
Architecture	x86
Host name	CORBIN-PC
Number of available processors	2
Operating system	Windows 7
Operating system version	6.1 build 7600

Environment reporting
- Detects invalid Java options
- Detects options which may hurt performance or serviceability
- Useful for remote diagnosis of configuration-related problems

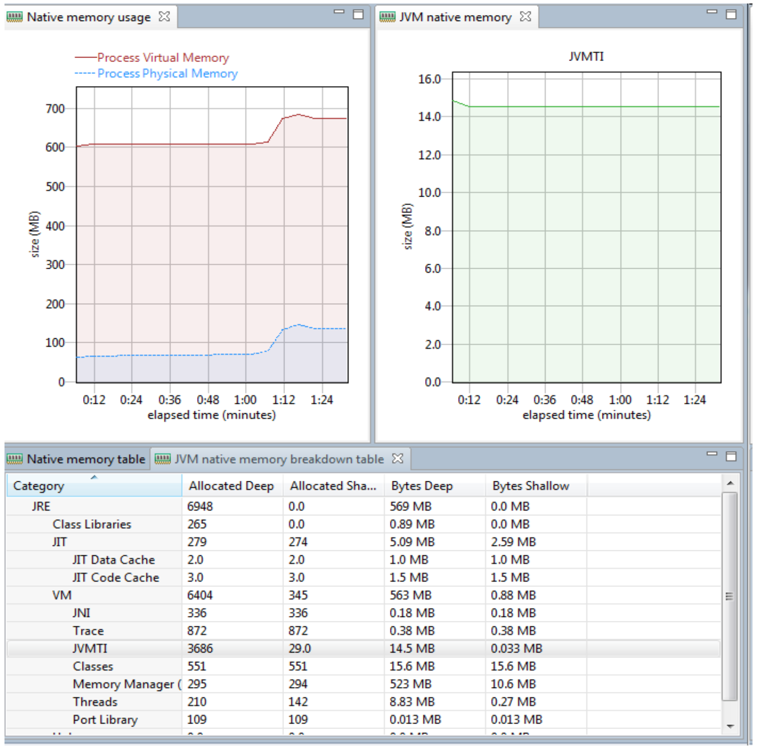
Health Center overview continued



13:50:40
/home/hhellyer/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/unoNode01Cell/PlantsByWebSphere.ear/PlantsByWebSphere.war/images/flower_pansies.jpg

Open time	File name
13:50:40	/home/hhellyer/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/unoNode01Cell/PlantsByWebSph
13:50:40	/home/hhellyer/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/unoNode01Cell/PlantsByWebSph
13:50:40	/home/hhellyer/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/unoNode01Cell/PlantsByWebSph
13:50:40	/home/hhellyer/IBM/WebSphere/AppServer/profiles/AppSrv01/installedApps/unoNode01Cell/PlantsByWebSph

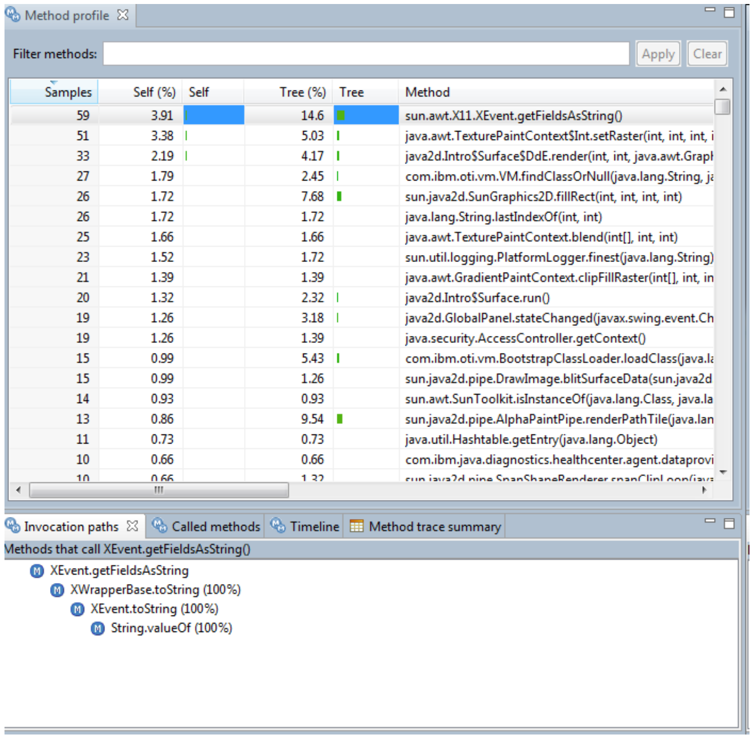
I/O
- •Monitor application file open/close events as they occur
- •Lists currently open files



Category	Allocated Deep	Allocated Sha...	Bytes Deep	Bytes Shallow
JRE	6948	0.0	569 MB	0.0 MB
Class Libraries	265	0.0	0.89 MB	0.0 MB
JIT	279	274	5.09 MB	2.59 MB
JIT Data Cache	2.0	2.0	1.0 MB	1.0 MB
JIT Code Cache	3.0	3.0	1.5 MB	1.5 MB
VM	6404	345	563 MB	0.88 MB
JNI	336	336	0.18 MB	0.18 MB
Trace	872	872	0.38 MB	0.38 MB
JVMTI	3686	29.0	14.5 MB	0.033 MB
Classes	551	551	15.6 MB	15.6 MB
Memory Manager (	295	294	523 MB	10.6 MB
Threads	210	142	8.83 MB	0.27 MB
Port Library	109	109	0.013 MB	0.013 MB

Native Memory
- •Detect native memory leaks in application
- •Determine if external forces are using more memory
- •Memory counters showing which parts of the JVM are using the most native memory

Health Center overview continued



Method profile

Filter methods: Apply Clear

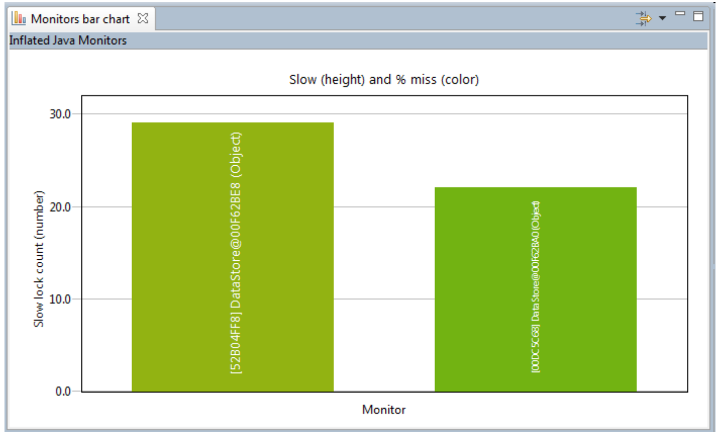
Samples	Self (%)	Self	Tree (%)	Tree	Method
59	3.91	▮	14.6	▮	sun.awt.X11.XEvent.getFieldsAsString()
51	3.38	▮	5.03	▮	java.awt.TexturePaintContext$Int.setRaster(int, int, int, i
33	2.19	▮	4.17	▮	java2d.Intro$Surface$DdE.render(int, int, java.awt.Grapl
27	1.79		2.45	▮	com.ibm.oti.vm.VM.findClassOrNull(java.lang.String, ja
26	1.72		7.68	▮	sun.java2d.SunGraphics2D.fillRect(int, int, int, int)
26	1.72		1.72		java.lang.String.lastIndexOf(int, int)
25	1.66		1.66		java.awt.TexturePaintContext.blend(int[], int, int)
23	1.52		1.72		sun.util.logging.PlatformLogger.finest(java.lang.String)
21	1.39		1.39		java.awt.GradientPaintContext.clipFillRaster(int[], int, in
20	1.32		2.32	▮	java2d.Intro$Surface.run()
19	1.26		3.18	▮	java2d.GlobalPanel.stateChanged(javax.swing.event.Ch
19	1.26		1.39		java.security.AccessController.getContext()
15	0.99		5.43	▮	com.ibm.oti.vm.BootstrapClassLoader.loadClass(java.la
15	0.99		1.26		sun.java2d.pipe.DrawImage.blitSurfaceData(sun.java2d
14	0.93		0.93		sun.awt.SunToolkit.isInstanceOf(java.lang.Class, java.la
13	0.86		9.54	▮	sun.java2d.pipe.AlphaPaintPipe.renderPathTile(java.lan
11	0.73		0.73		java.util.Hashtable.getEntry(java.lang.Object)
10	0.66		0.66		com.ibm.java.diagnostics.healthcenter.agent.dataprovi
10	0.66		1.32		sun.java2d.pipe.SpanShapeRenderer.spanClipLoop(java

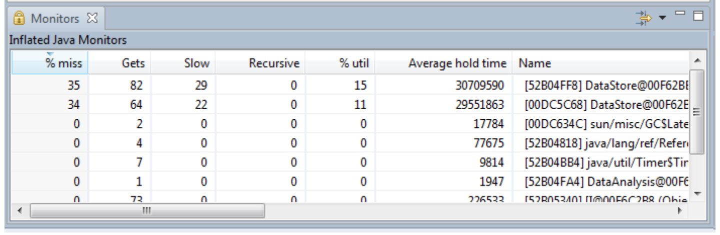
Invocation paths | Called methods | Timeline | Method trace summary

Methods that call XEvent.getFieldsAsString()

```
XEvent.getFieldsAsString
  XWrapperBase.toString (100%)
    XEvent.toString (100%)
      String.valueOf (100%)
```

Method Profiling
- Always-on profiling offers insight into application activity
- Identifies the hottest methods in an application
- Full call stacks to identify where methods are being called from and what methods they call
- No byte code instrumentation, no recompiling





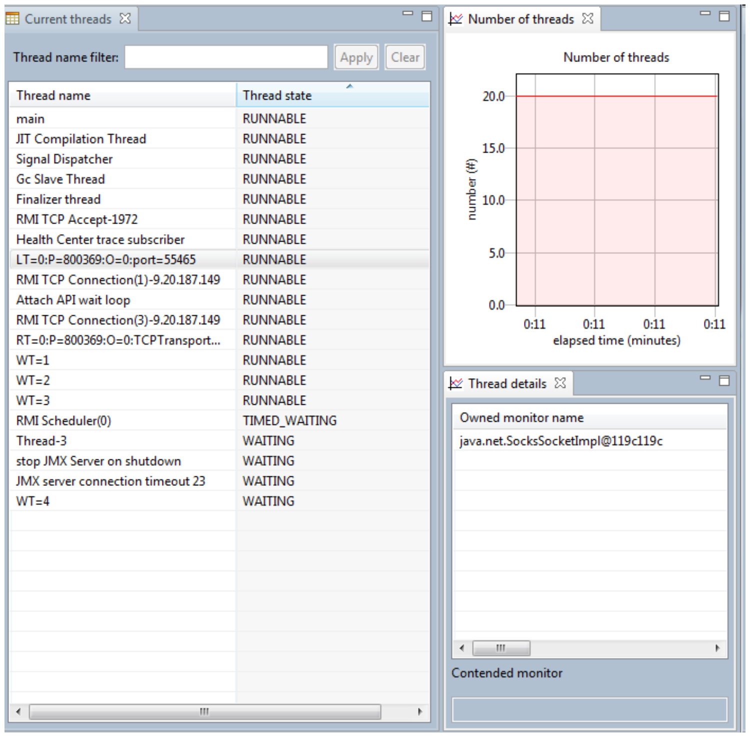
Monitors

Inflated Java Monitors

% miss	Gets	Slow	Recursive	% util	Average hold time	Name
35	82	29	0	15	30709590	[52B04FF8] DataStore@00F62BE
34	64	22	0	11	29551863	[00DC5C68] DataStore@00F62E
0	2	0	0	0	17784	[00DC634C] sun/misc/GC$Late
0	4	0	0	0	77675	[52B04818] java/lang/ref/Refer
0	7	0	0	0	9814	[52B04BB4] java/util/Timer$Tir
0	1	0	0	0	1947	[52B04FA4] DataAnalysis@00F6
0	73	0	0	0	226533	[52B05340] [I@00F6C2B8 (Obje

Java Lock analysis
- Always-on lock monitoring
- Quickly allows the usage of all locks to be profiled
- Helps to identify points of contention in the application that are preventing scaling

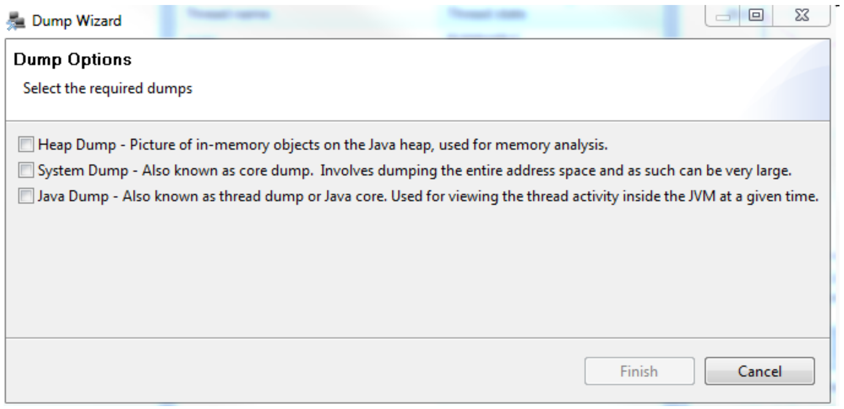
Health Center overview continued



Thread name	Thread state
main	RUNNABLE
JIT Compilation Thread	RUNNABLE
Signal Dispatcher	RUNNABLE
Gc Slave Thread	RUNNABLE
Finalizer thread	RUNNABLE
RMI TCP Accept-1972	RUNNABLE
Health Center trace subscriber	RUNNABLE
LT=0:P=800369:O=0:port=55465	RUNNABLE
RMI TCP Connection(1)-9.20.187.149	RUNNABLE
Attach API wait loop	RUNNABLE
RMI TCP Connection(3)-9.20.187.149	RUNNABLE
RT=0:P=800369:O=0:TCPTransport...	RUNNABLE
WT=1	RUNNABLE
WT=2	RUNNABLE
WT=3	RUNNABLE
RMI Scheduler(0)	TIMED_WAITING
Thread-3	WAITING
stop JMX Server on shutdown	WAITING
JMX server connection timeout 23	WAITING
WT=4	WAITING

Owned monitor name
java.net.SocksSocketImpl@119c119c

Threads view
- List of current threads and states
- Deadlock detection and analysis
- Number of threads over time
- See contended monitors



Dump Options

Select the required dumps

- ☐ Heap Dump - Picture of in-memory objects on the Java heap, used for memory analysis.
- ☐ System Dump - Also known as core dump. Involves dumping the entire address space and as such can be very large.
- ☐ Java Dump - Also known as thread dump or Java core. Used for viewing the thread activity inside the JVM at a given time.

Live control of application
- Trigger dumps
- Enable verbosegc collection

Health Center installation

- The tool is provided in two parts:
  - An agent that collects data from a running application.
  - An Eclipse-based client that connects to the agent.
- The Agent ships with the following Java SDK versions:
  - Java 5sr9 and upwards
  - Java 6sr3 and upwards
- The latest version of the agent is always available from within the Health Center Client
  - Recommended to always update to the latest version of the agent
- Agent package unzips over the JRE directory of the Java installation you are using to run the application

How to enable an application for monitoring

- Full instructions are provided within the help shipped with the Health Center Client but in most cases as simple as :

  For Java 5 SR10 and later, or Java 6 SR5 and later (including Java 7)

  ```
  java –Xhealthcenter HelloWorld
  ```

  (can be used in production)

  For 5 SR9 and earlier, or Java 6 SR4 and earlier

  ```
  java –agentlib:healthcenter –Xtrace:output=healthcenter.out HelloWorld
  ```

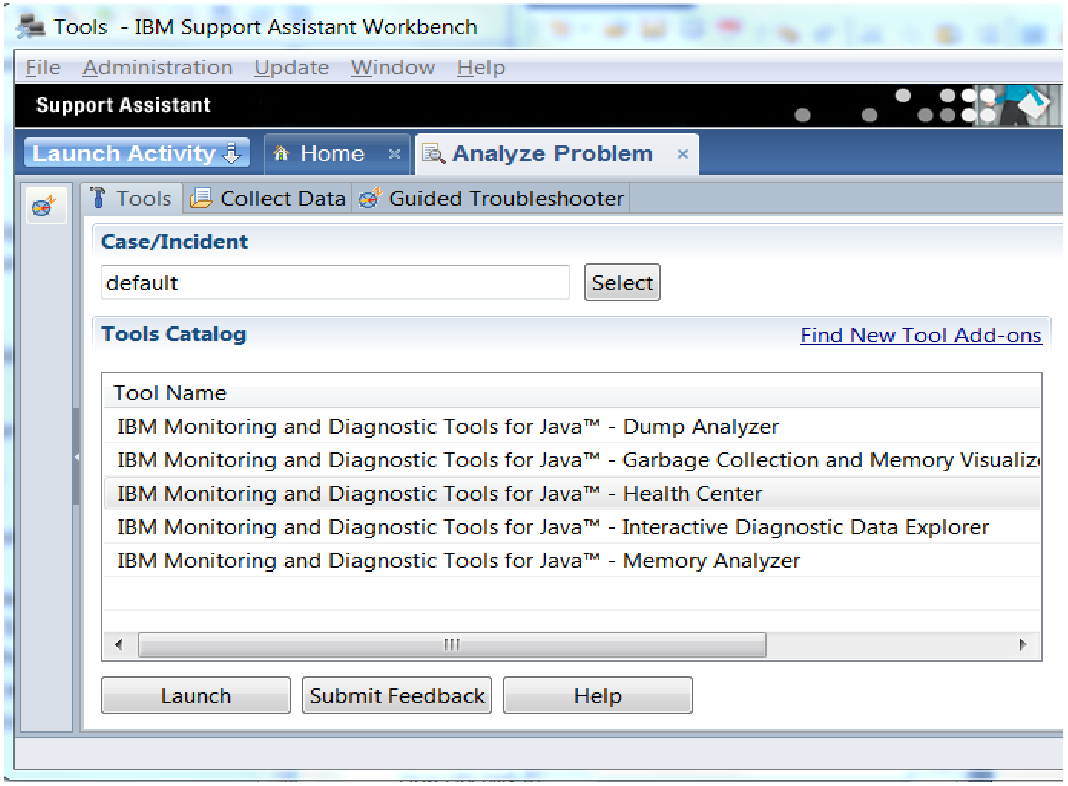
  (not recommended for use in a production environment)

How to install the client

- Download and install IBM Support Assistant Workbench 4.1
  - http://www.ibm.com/software/support/isa/workbench.html
  - An Eclipse based tool
  - You select the IBM support plugins you want
  - In the workbench, select Update > Find New... > Tools Add-ons
  - Expand JVM-based Tools
  - Select "IBM Monitoring and Diagnostics Tools for Java – Health Center"
  - Click Next, accept the license
  - Click Next, confirm the tool selected, Click Install
  - The Eclipse update mechanism will install the Health Center plug in into IBM Support Assistant

How to launch the client

- In IBM Support Assistant go to the Home tab
- Click Analyze Problem
- Select Health Center in the Tools Catalog, click Launch



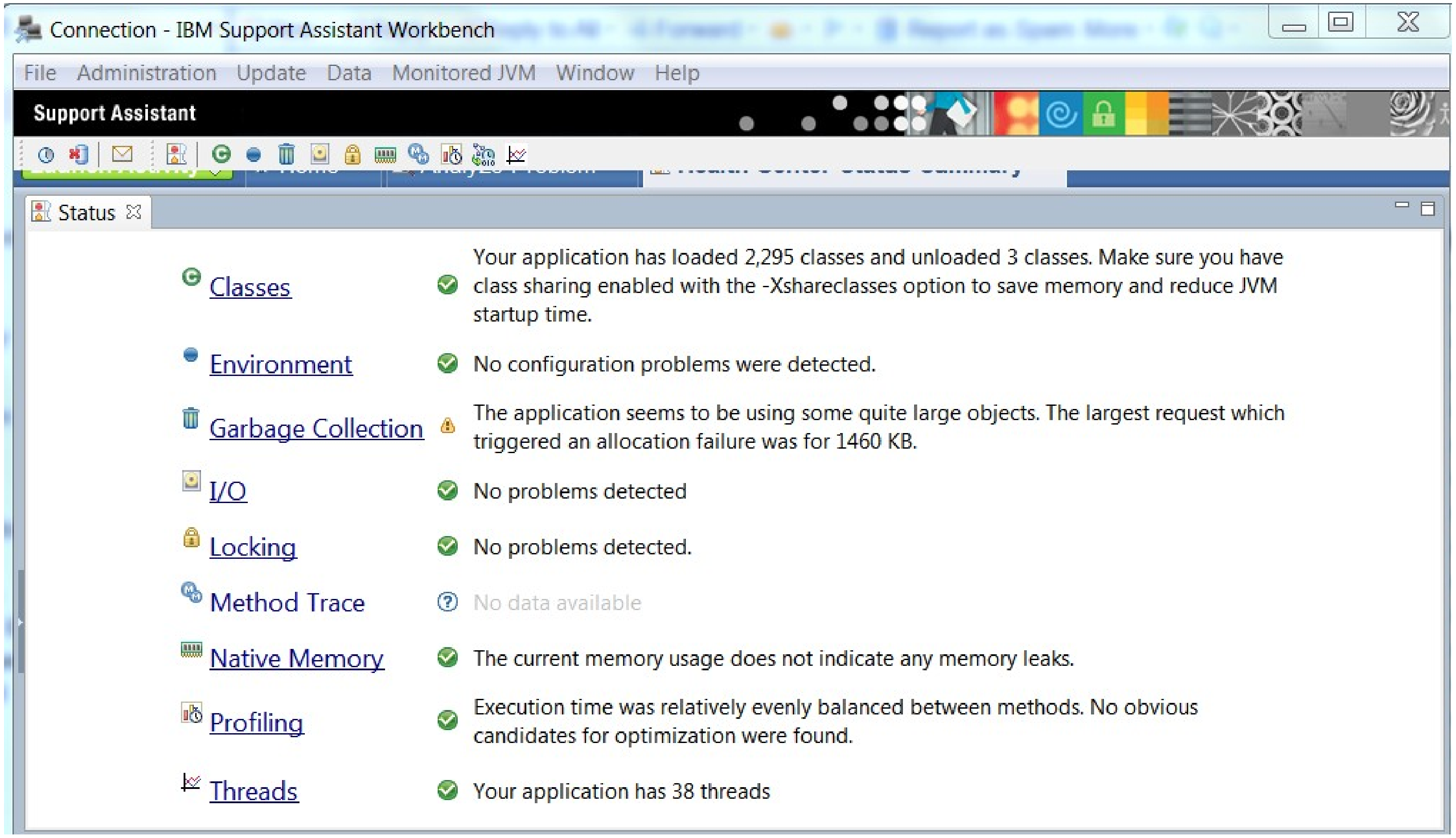
Case/Incident

default

Tools Catalog

Tool Name
IBM Monitoring and Diagnostic Tools for Java™ - Dump Analyzer
IBM Monitoring and Diagnostic Tools for Java™ - Garbage Collection and Memory Visualiz
IBM Monitoring and Diagnostic Tools for Java™ - Health Center
IBM Monitoring and Diagnostic Tools for Java™ - Interactive Diagnostic Data Explorer
IBM Monitoring and Diagnostic Tools for Java™ - Memory Analyzer

IBM Health Center Demonstration


Connection - IBM Support Assistant Workbench
File Administration Update Data Monitored JVM Window Help
Support Assistant
Status

Classes	✓	Your application has loaded 2,295 classes and unloaded 3 classes. Make sure you have class sharing enabled with the -Xshareclasses option to save memory and reduce JVM startup time.
Environment	✓	No configuration problems were detected.
Garbage Collection	⚠	The application seems to be using some quite large objects. The largest request which triggered an allocation failure was for 1460 KB.
I/O	✓	No problems detected
Locking	✓	No problems detected.
Method Trace	?	No data available
Native Memory	✓	The current memory usage does not indicate any memory leaks.
Profiling	✓	Execution time was relatively evenly balanced between methods. No obvious candidates for optimization were found.
Threads	✓	Your application has 38 threads

Access Health Center data with the API

- The 2.1 release of Health Center contains a powerful API. The API allows Java™ developers to embed Health Center in their applications

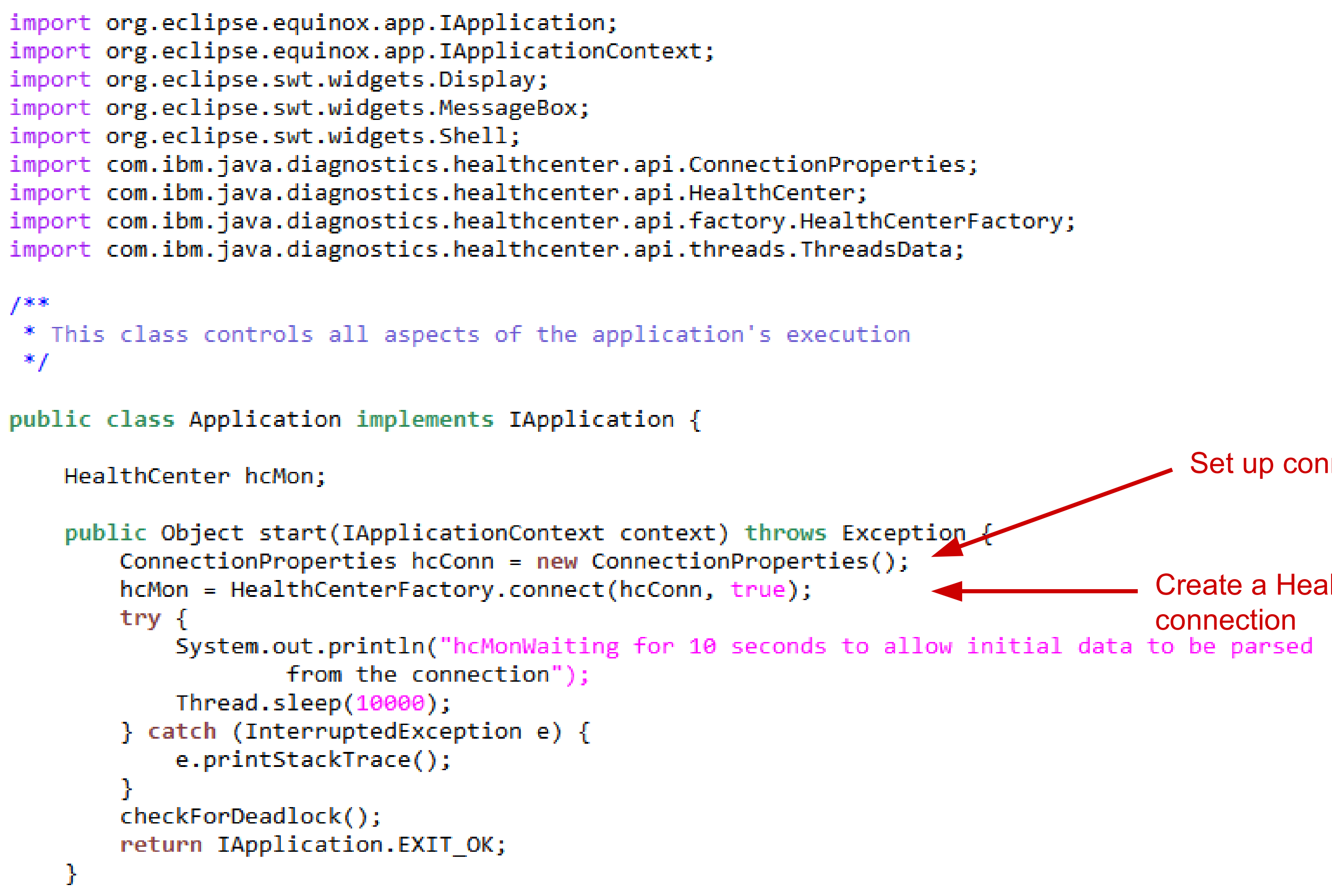
- With a few lines of code, you can embed the monitoring power of Health Center in your own Eclipse based application and harness its monitoring power to troubleshoot problems

```
// Create the connection object:
ConnectionProperties conn1 = new ConnectionProperties("localhost", 1973);
// Connect to the Health Center agent, using the previous connection
// settings:
HealthCenter hcObject = HealthCenterFactory.connect(conn1, true);
// Get garbage collection data and print:
GCData gcData = hcObject.getGCData();
System.out.println("GC Mode is " + gcData.getGCMode().toString());
```

Getting started with the Health Center API

- Detailed steps with screen shots in online articles (see final slide)
- Online articles have code samples to get started with
- Download and install Eclipse 3.4 or above from eclipse.org
- Use the Health Center update site to install the API into Eclipse
- Create a new Rich Client Platform (RCP) project
- Add the Health Center API plugin to the build path of the project
- Start coding to the API

Coding Example: Deadlock Detection



```java
import org.eclipse.equinox.app.IApplication;
import org.eclipse.equinox.app.IApplicationContext;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;
import com.ibm.java.diagnostics.healthcenter.api.ConnectionProperties;
import com.ibm.java.diagnostics.healthcenter.api.HealthCenter;
import com.ibm.java.diagnostics.healthcenter.api.factory.HealthCenterFactory;
import com.ibm.java.diagnostics.healthcenter.api.threads.ThreadsData;

/**
 * This class controls all aspects of the application's execution
 */

public class Application implements IApplication {

    HealthCenter hcMon;

    public Object start(IApplicationContext context) throws Exception {
        ConnectionProperties hcConn = new ConnectionProperties();
        hcMon = HealthCenterFactory.connect(hcConn, true);
        try {
            System.out.println("hcMonWaiting for 10 seconds to allow initial data to be parsed
                    from the connection");
            Thread.sleep(10000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        checkForDeadlock();
        return IApplication.EXIT_OK;
    }
```

Set up connection properties

Create a Health Centre connection

Coding Example continued: Deadlock Detection

```java
public void checkForDeadlock() {
    while (!detectDeadlock()) {
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

private boolean detectDeadlock() {
    ThreadsData hcthreadsData = hcMon.getThreadsData();
    if (hcthreadsData == null) {
        System.out.println("No threads yet");
    } else {
        if (hcthreadsData.deadlockDetected()) {
            Display display = new Display();
            Shell shell = new Shell(display);
            MessageBox mb = new MessageBox(shell);
            String deadlockMessage = new String();
            String[] hcThreadsRec = hcthreadsData
                .getCriticalRecommendations();
            for (String rec : hcThreadsRec) {
                deadlockMessage = deadlockMessage + rec + "\n";
            }
            mb.setMessage(deadlockMessage);
            mb.setText("Deadlock detected");
            mb.open();
            display.dispose();
            return true;
        }
    }

    return false;
}
```

Request Threads data

Check for a thread deadlock

Access the threads recommendations

Display the deadlock detected message

Advanced options for using Health Center

- Headless mode for data collection without connecting the GUI
  - Useful for scenarios where firewall blocks connection
  - Configurable to limit disk space used
  - Timed collections
  - Interval based collections
  - Started with

    `-Xhealthcenter:level=headless`
  - Output: .hcd data files. Open in GUI client or with API.

- Late Attach enabled
- Automated javacore creation

Quick contacts

- YouTube videos for a quick introduction to the tools
- @IBM_JTC Twitter feed
- Email javatool@uk.ibm.com for tools support

Where to find more information

- IBM Monitoring and Diagnostic Tools for Java™ on developerWorks
  http://www.ibm.com/developerworks/java/jdk/tools/

- http://tinyurl.com/IBMJavaTools

- Health Center API documentation online (it's also in the client help menu)

- Health Center API articles
  - Monitor a Java application with the Health Center API parts 1 and 2
    - http://www.ibm.com/developerworks/library/j-healthcareapi1/index.html
    - http://www.ibm.com/developerworks/library/j-healthcareapi2/index.html

- IBM Support Assistant (ISA) Workbench

  http://www.ibm.com/software/support/isa/workbench.html