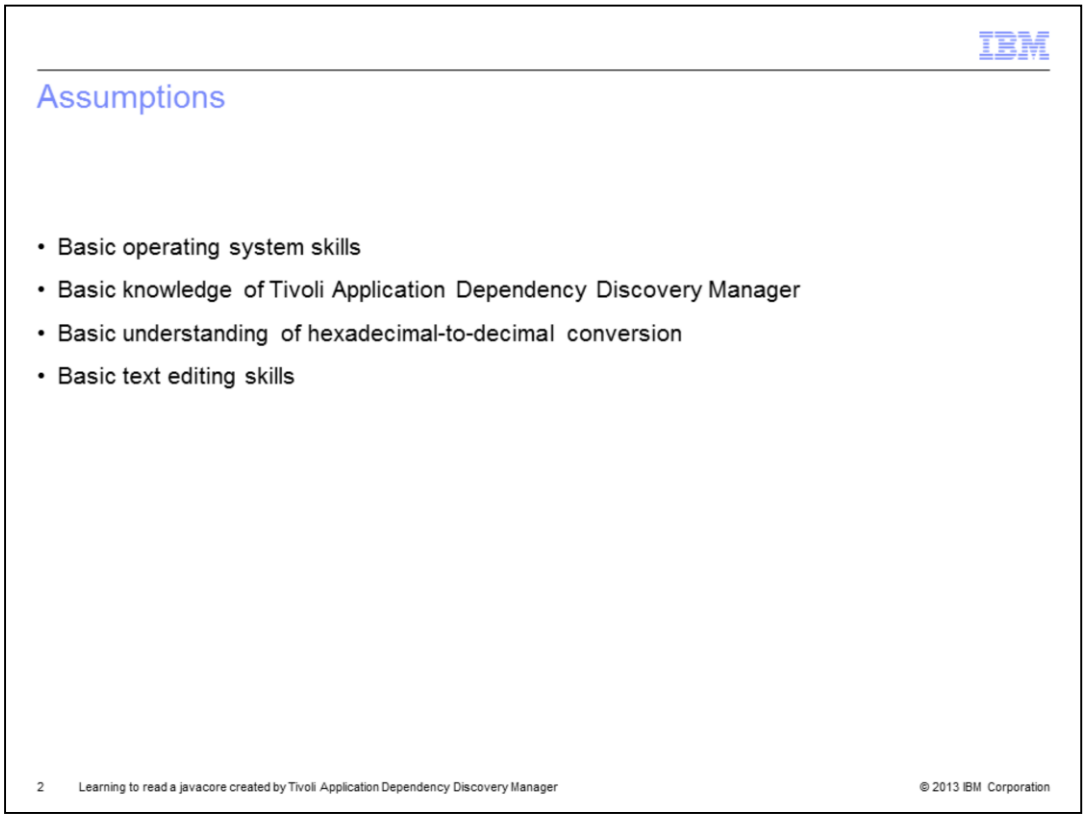
IBM

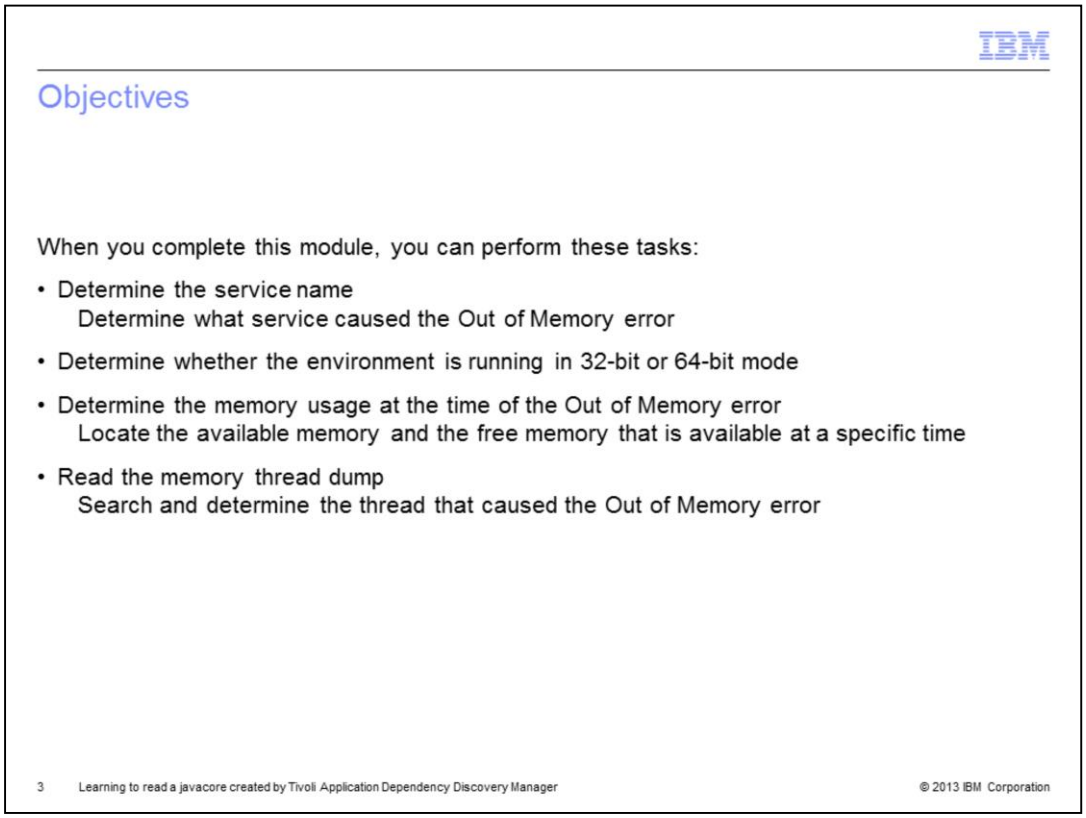
IBM Tivoli Application Dependency Discovery Manager 7.2.1

Learning to read a javacore created by Tivoli Application Dependency Discovery Manager

In this module, you learn how to read a Tivoli® Application Dependency Discovery Manager javacore file.



Some assumptions before you begin this module are that you have the skills listed here.



When you have completed this module, you can perform several tasks:

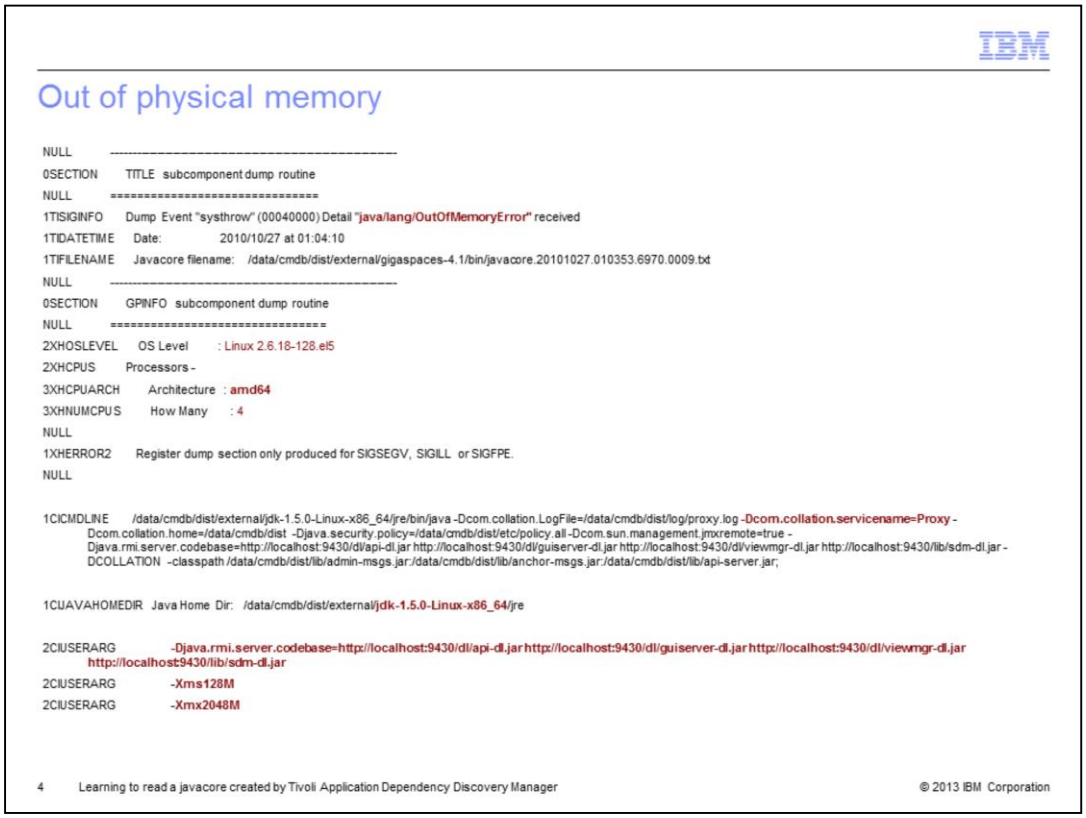
- Determine the service name and the service that is running at the time of the Out of Memory error.

- Know that the service name provides a starting point for deciding the next actions to take to resolve the issue.

- Determine whether the environment is running in 32-bit or 64-bit mode, which is important because each mode has different memory maximums. You can also find the reason for the Out of Memory error.

- Determine the amount of memory that is available in the environment. This information is important to know. The Out of Memory error might be caused by the system not having enough available memory to perform the request from Tivoli Application Dependency Discovery Manager.

When you read the thread memory dump, you can see the .jar that was running at the time of the Out of Memory error. You can investigate the Tivoli Application Dependency Discovery Manager logs to acquire the root cause of the issue.



This particular javacore is related to the environment running out of physical memory.

- The 1TISIGINFO line shows that the problem is an Out of Memory error.

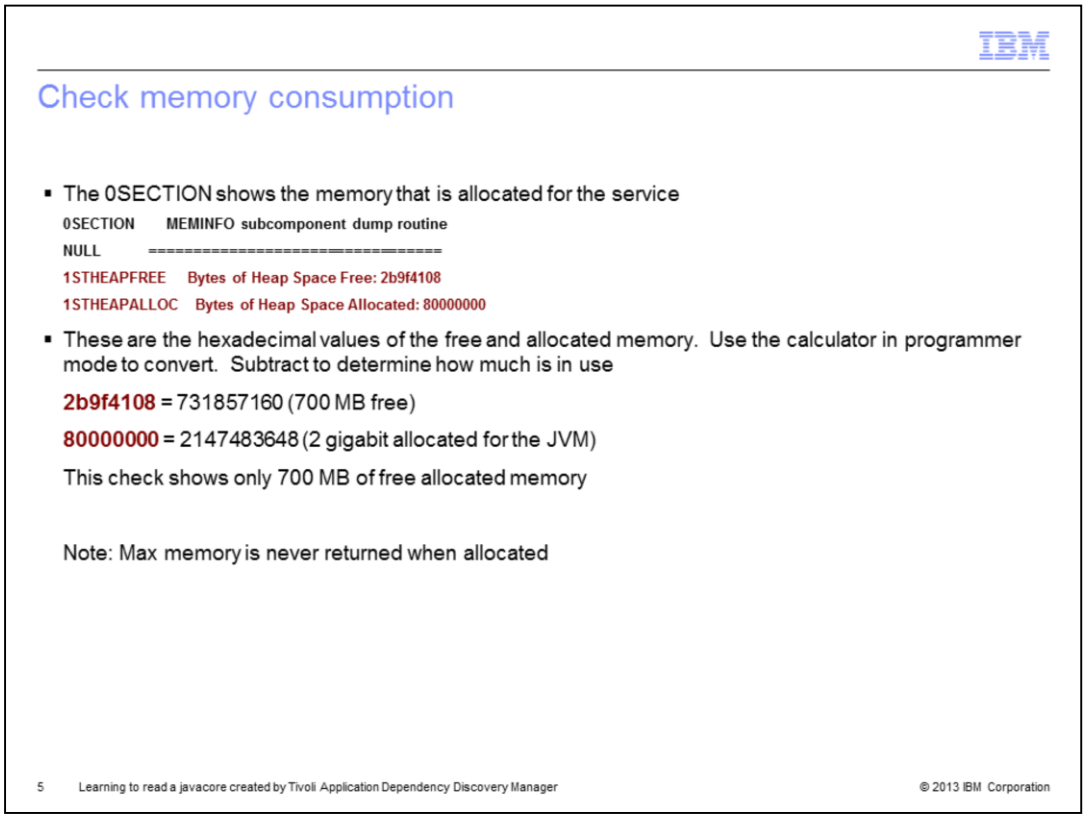
- The 2XHOSLEVEL shows the operating system of the machine; in this case, it is Linux®. You also see the base level and any patches that are applied.

- The 3XHCPUARCH shows the architecture of the machine. In this case, it is amd64 bit CPU.

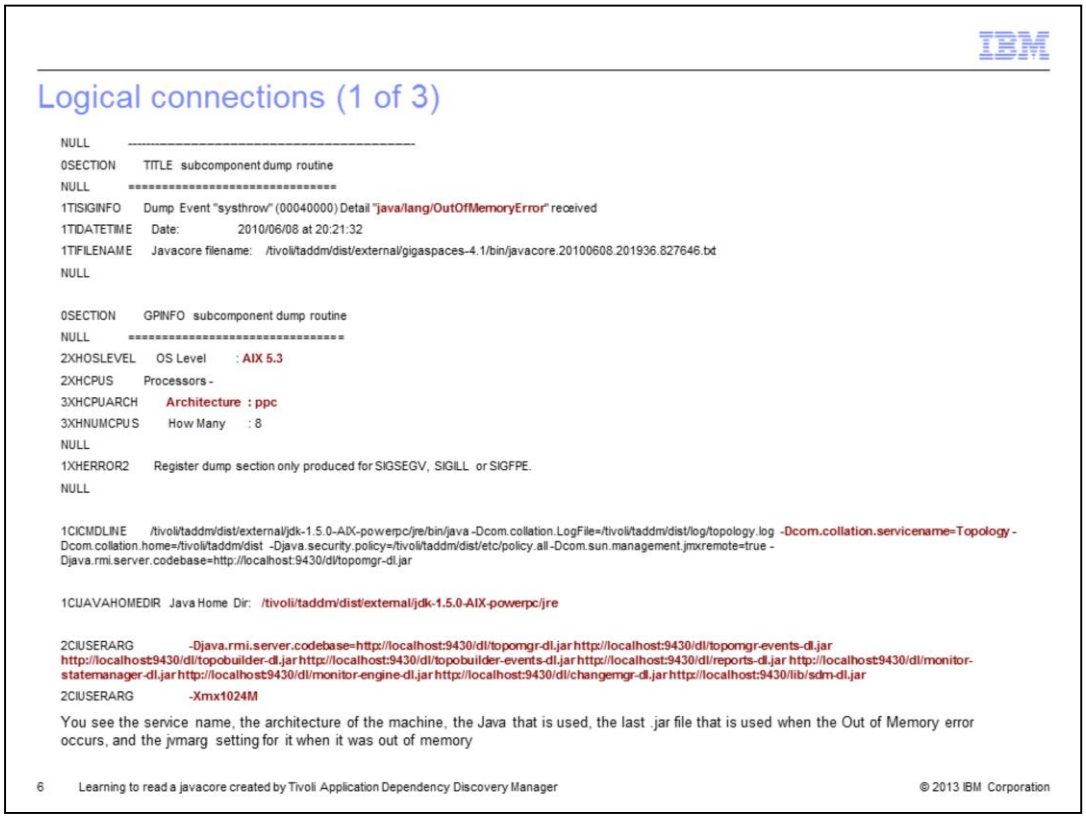
- The 3XHNUMCPUS shows the number of CPUs on the machine. In this case, it is four CPUs.

- The 1CICMDLINE shows the servicename that caused the Out of Memory error. In this case, it is Proxy service.

- The 2CIUSERARG shows the jvmargs. Be sure to note the –Xms and –Xmx settings.



The 0SECTION shows the memory information, in hexadecimal. In this case, there is not enough allocated memory. There is only 700 MB of memory free, and that is not enough to complete the action that is requested by the proxy. Confirm that additional memory is available on the server, and increase the max (Xmx) memory that is allocated for the proxy service. Note that max memory is never given back after it is allocated. Keep that in mind when you allocate memory.



This particular javacore is related to logical connections.

- The 1TISIGINFO shows the Out of Memory error.

- The 2XHOSLEVEL shows the operating system of the machine. In this case, it is AIX® 5.3.

- The 3XHCPUARCH shows the architecture of the machine. In this case, it is a Power PC architecture.

- The 3XHNUMCPUS shows the number of CPUs of the machine. In this case, it is eight CPUs.

- The 1CICMDLINE shows the service name that is causing the Out of Memory error. In this case, it is the Topology Service.

- The 2CIUSERARG shows the Java arguments that are used.

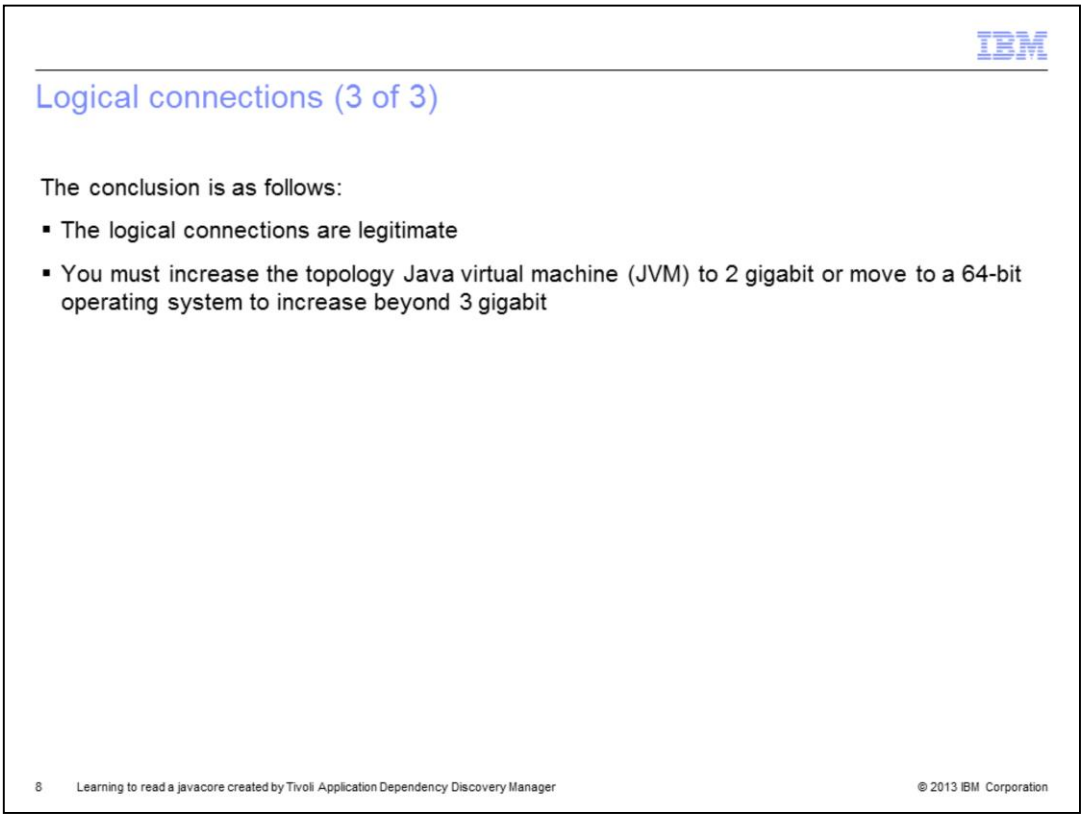
The 3XMTHREADINFO shows that TopologyBuilderEngineThread is the point of failure.

The 4XESTACKTRACE shows the code that failed. In this case, it is highlighted in red and an asterisk has been added to the trace to help you locate the line.

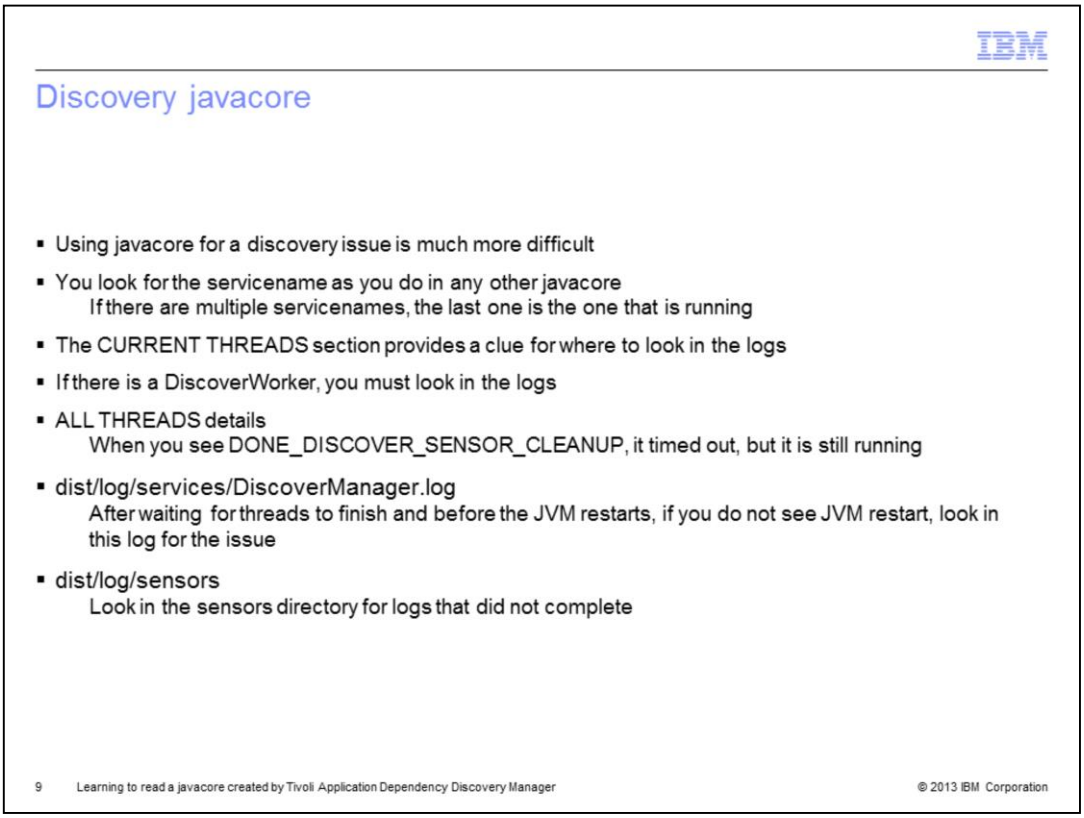
Note that the TopologyBuilderAgent is single-threaded.

Current threads are not typically what cause the issue. It is typically in the All Threads Details.

Anything ending in jdo is automatically generated dynamically or listed as (Compiled Code). This type of code requires a code change because it is compiled.



The reason for the logical connection Out of Memory error is that the logical connections were legitimate, but the allocated memory did not allow for the number of logical connections. Increase the Topology JVM Xmx setting. You must have a 64-bit architecture if you have to increase memory beyond 3 gigabit.



Discovery out-of-memory is more difficult to diagnose. You can find the service name. However, if you see multiple service names, the last one is the one that is running at the time of the Out of Memory error.

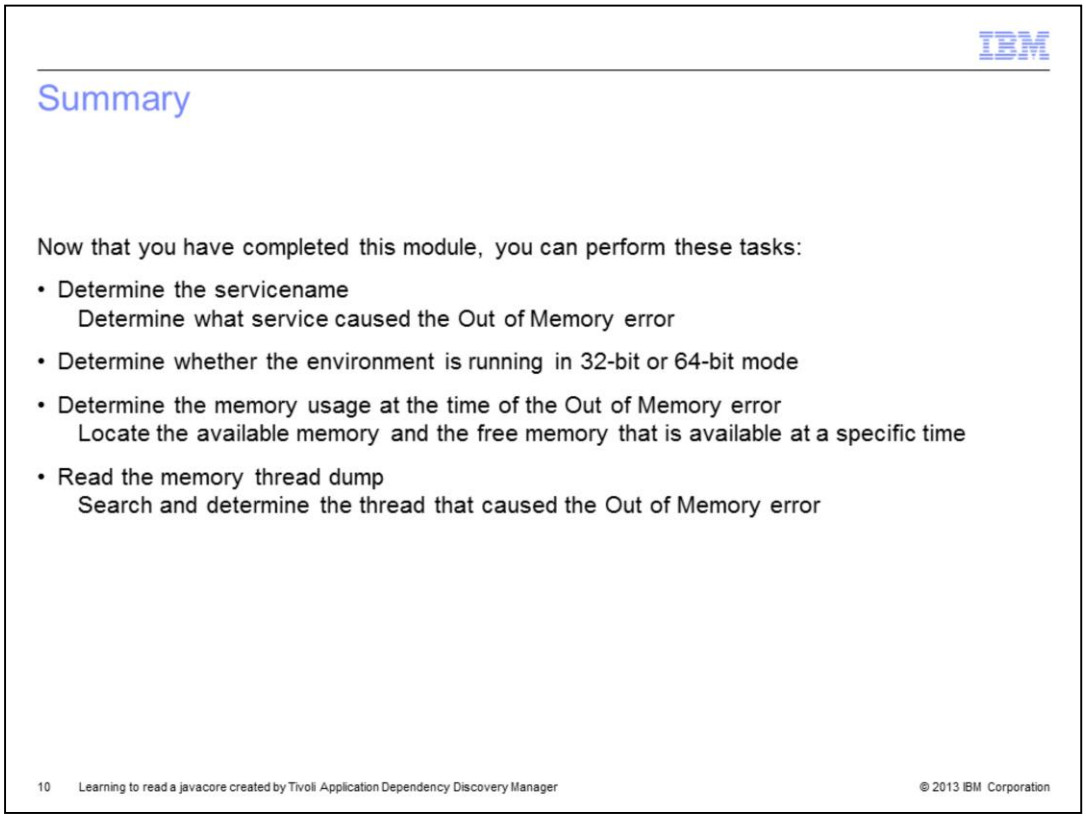
The current thread at the bottom of the javacore provides a clue to where to look in the dist/log log files. Search on 4XE and look for a heading of Current threads.

If there is a DiscoverWorker, then look in the dist/log log files.

In the All Threads Details, look for DONE_DISCOVER_SENSOR_CLEANUP. This is where it timed out, but is still running.

DiscoverManager.log shows everything after waiting for the threads to finish the message and before the JVM restarts. If JVM does not restart, then look for the issue in the dist/log/sensors directory.

In the dist/log/sensors directory, look for sensors that did not complete.



Now that you completed this module, you can determine the servicename and the service that is running at the time the Out of Memory error occurred.

You know whether the environment is running in 32-bit or 64-bit mode, which is important because each mode has different memory maximums. You know the amount of memory allocated to the service.

Reading the thread memory dump provides the .jar file that is running at the time of the Out of Memory error. You can investigate the Tivoli Application Dependency Discovery Manager logs when needed to acquire the root cause of the issue.

Trademarks, disclaimer, and copyright information