

## Prerequisites

- The IBM Heap Analyzer .jar file:  
<https://public.dhe.ibm.com/software/websphere/appserv/support/tools/HeapAnalyzer/ha457.jar>
- Java JDK v1.8 or higher (Oracle JDK or OpenJDK are both fine) installed and configured (be sure that 'java' and 'javac' can both be run from the command line)

## Introduction

In this demo, we will create an obvious memory leak in a Java application and use the IBM Heap Analyzer tool to determine the source of the leak.

The application itself creates a leak by populating an unbounded Java ArrayList with 10mb byte array objects every 100ms.

## Instructions

The first step is to acquire the source code, build it, and run it.

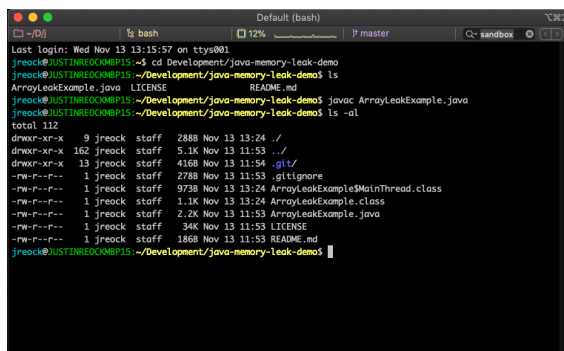
### Build and Run

Clone the repository: <https://github.com/jreock/java-memory-leak-demo>

Inside the repository root, you'll see a file called ArrayLeakExample.java. This is the file we will use to create the leak. Build it with javac:

[inside the repository root folder]  
javac ArrayLeakExample.java

That will create the necessary .class files:



```
Default (bash)
~/D/
jreock@JUSTINIEOONBP15:~$ cd Development/java-memory-leak-demo
jreock@JUSTINIEOONBP15:~/Development/java-memory-leak-demo$ ls
ArrayLeakExample.java  LICENSE  README.md
jreock@JUSTINIEOONBP15:~/Development/java-memory-leak-demo$ javac ArrayLeakExample.java
jreock@JUSTINIEOONBP15:~/Development/java-memory-leak-demo$ ls -al
total 112
drwxr-xr-x  9 jreock staff 2888 Nov 13 13:24 ./
drwxr-xr-x 162 jreock staff 5.1K Nov 13 11:53 ../
drwxr-xr-x 13 jreock staff 4168 Nov 13 11:54 .git/
-rw-r--r--  1 jreock staff 2788 Nov 13 11:53 .gitignore
-rw-r--r--  1 jreock staff 9738 Nov 13 13:24 ArrayLeakExample$MainThread.class
-rw-r--r--  1 jreock staff 1.1K Nov 13 13:24 ArrayLeakExample.class
-rw-r--r--  1 jreock staff 2.2K Nov 13 11:53 ArrayLeakExample.java
-rw-r--r--  1 jreock staff 34K Nov 13 11:53 LICENSE
-rw-r--r--  1 jreock staff 1868 Nov 13 11:53 README.md
jreock@JUSTINIEOONBP15:~/Development/java-memory-leak-demo$
```

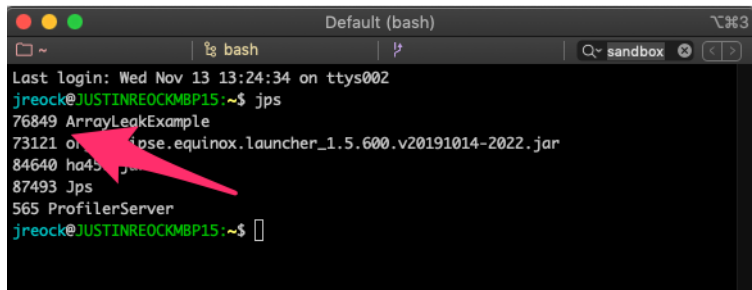
```
java -cp . ArrayLeakExample
```

Eventually, you will Heap out of Memory errors occurring, but, you don't need to wait for that to start the diagnosis.

## Diagnosis

A screenshot of a macOS desktop environment. On the left, a Terminal window titled "Default (java)" shows the command `java -jar hsa.jar` being executed. The output indicates that IBM HeapAnalyzer version 4.5.7 started successfully, inspected the class `com.ibm.jinwoo.heap.plugin.DOMParserPlugin`, loaded the plugin `com.ibm.jinwoo.heap.plugin.DOMParserPlugin`, and displayed details about the DOM4J v1.6.1 instances in the tree view. On the right, the IBM HeapAnalyzer application window is open, displaying its splash screen which includes the title bar "IBM HeapAnalyzer", menu items "File Analysis View Window Help", and a toolbar. The main area features a blue abstract graphic and the text "HeapAnalyzer Version 4.5.7" along with copyright information. At the bottom of the application window, there is a "Console" tab.

Now you will need to acquire a heap dump. The easiest way is from the command line using the jmap tool which ships with the JDK. First, acquire the PID of your running process. The jps tool can be used to do this easily:

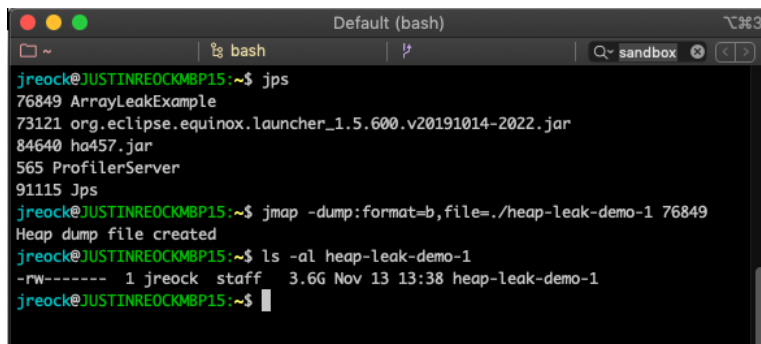


```
Default (bash)
~
Last login: Wed Nov 13 13:24:34 on ttys002
jreock@JUSTINREOCKMBP15:~$ jps
76849 ArrayLeakExample
73121 org.eclipse.equinox.launcher_1.5.600.v20191014-2022.jar
84640 ha457.jar
87493 Jps
565 ProfilerServer
jreock@JUSTINREOCKMBP15:~$
```

Then, run:

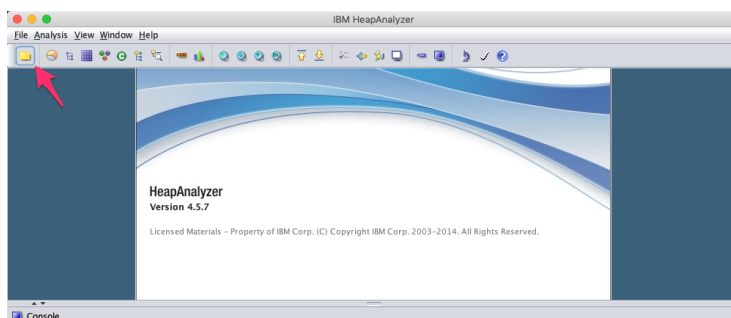
```
jmap -dump:format=b,file=./heap-leak-demo-1 [PID]
```

Where [PID] is the PID you just identified. It will take a few moments but will produce a (probably large) file which is a dump of your entire Java heap.



```
Default (bash)
~
jreock@JUSTINREOCKMBP15:~$ jps
76849 ArrayLeakExample
73121 org.eclipse.equinox.launcher_1.5.600.v20191014-2022.jar
84640 ha457.jar
565 ProfilerServer
91115 Jps
jreock@JUSTINREOCKMBP15:~$ jmap -dump:format=b,file=./heap-leak-demo-1 76849
Heap dump file created
jreock@JUSTINREOCKMBP15:~$ ls -al heap-leak-demo-1
-rw-r----- 1 jreock staff 3.6G Nov 13 13:38 heap-leak-demo-1
jreock@JUSTINREOCKMBP15:~$
```

Open this file in the Heap Analyzer:



Select Leak Suspects, and we can see from the object tree and from the leak suspects tab that we are suspicious of an ArrayList containing byte[] objects of 10mb in size.

## Validation

Finally, we can “validate” in the code that we have a leaky array. By searching for ArrayLists and looking at the way we populate the array, it becomes clear that we are never cleaning up the byte[] objects:

```
/**
 * Implement the thread as a while loop that repeats every 100ms and calls the
 * populateArray() method
 */
static final class MainThread extends Thread {
    @Override
    public void run() {
        while (running) {
            try {
                populateArray();
            } catch (Throwable ex) {
                ex.printStackTrace();
            }
            try {
                Thread.sleep(100);
            } catch (InterruptedException ex) {
                System.out.println("Bye!");
                running = false;
            }
        }
    }

    /**
     * Here's where the magic happens, add a 10mb byte array to the leaky ArrayList
     * object
     */
    static void populateArray() {
        final byte leakerByte[] = new byte[1024 * 1024 * 10];
        leakyArr.add(leakerByte);
    }
}
```

## Wrap Up

This has been an example, albeit simplified, of a real support scenario that the IBM TSS team and OpenLogic deal with all the time. The IBM Heap Analysis tool is one of many resources the team has at their disposal to deliver solutions to our customers.