What I Learned This Month:
~~Some~~ Most days I Just Don't Understand Java
Scott Chapman
American Electric Power

We had a major Java application go live in Websphere on the mainframe recently. I'm happy to say that it's running very nicely on the z10 zAAPs and we routinely see >95% of the CPU time for the Java application on the zAAPs. It seems like Websphere Application Server (WAS) on z makes some good sense, depending of course on the license cost since WAS is licensed on your general-purpose CPU capacity. However, this has given me the "opportunity" to be involved in debugging three "interesting" Java behaviors that frankly don't make sense to me, even though I'm sure they made sense to somebody at some point in time.

One interesting thing we ran into was that most (all?) JVMs by default cache all DNS requests for the life of the JVM. Once your app has made a connection to a server, the same IP address will be used for that server from that point forward. So if you have multiple servers listed in a generic DNS entry thinking that your requests will round-robin between those addresses, perhaps it's not the case (that's been the results we've seen). Or if your server goes down and the DNS gets updated to point to a new server, then any connecting JVM needs to be recycled to get that new address. That seems BAD to me: "Broken As Designed". But supposedly it's a security option to help avoid certain DNS spoofing attacks. In an enterprise environment, I'm not sure it makes a lot of sense. The "fix" for this is to either change the security property "networkaddress.cache.ttl" or set that dynamically in your application code with java.security.Security.setProperty ("networkaddress.cache.ttl",0). Zero means "never cache", -1 is the default and means "cache forever" and positive numbers are the duration that the results will be cached. Choose wisely!

Another interesting result we ran into was an excessive number of SQL statements failing with -301, which means a host variable has an invalid data type. This is not something that you would expect to see hundreds of per hour in production, especially with no apparent negative impact on the application. After some digging by the application folks, they discovered that the JDBC driver they were using (I don't know which one it was) was trapping the -301 and automatically changing the variable from an integer to the expected character value and then resubmitting the corrected SQL statement transparently to the application. So DB2 was having to handle twice as many update statements as it should have had to because it was failing the first execution. Whoever wrote that driver probably thought that sounded like a really polite and nice thing to do. But if it hadn't been so polite, the coding error would have been caught in development and not resulted in many unnecessary calls from the application to DB2. The application developer eventually found this in some detailed JDBC log file. So perhaps another lesson is to double check all your log files for errors before moving into production!

Finally, since I have ample spare zAAP capacity, I was working on an application that would collect some performance statistics over time. Being the memory-conscientious guy that I am, I set out to determine how much heap space each observation would require. After all, we always ask our vendors for heap space estimates. As it turns out, the conclusion that I came to after a fair bit of research is that you can't determine how much memory your objects are going to consume! There are ways to generate some estimates, but it appears that it's nowhere nearly as simple as an 8 character string simply consumes 16 bytes: although that's not a bad estimate, there's additional per-object overhead. This would seem to make memory capacity planning for Java applications very difficult—but then we all already knew that. Still, it's disappointing.

In exploring the memory issues, it has become obvious to me that how the JVM manages the heap and the resulting performance and CPU consumption may vary significantly based on your run time options and exactly how the application runs. But that's a topic for a much longer discussion—I'm contemplating a CMG '11 paper based on my research.
So that's what I think I learned this month. If you think I didn't learn my lesson correctly, please email me at sachapman@aep.com and let me know!