Multitenant Java

About me

- Iain Lewis
- QA Engineer, IBM Java Technology Center, IBM Hursley, U.K.
- 13 years experience developing, testing and deploying Java SDKs
- Currently testing Multitenancy on IBM Java 7 R1

- Contact info
  - iain_lewis@uk.ibm.com
- Presenting on behalf of the development team

This session will help you to:

- Understand what multitenancy is
- Understand why it is important
- Find out what it can do for you
- Discover IBM's multitenancy technology in Java 7 R1

Multitenancy == Simplification

- Multitenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations (tenants).
- Multitenancy is contrasted with a multi-instance architecture where separate software instances (or hardware systems) are set up for different client organizations.
- With a multitenant architecture, a software application is designed to virtually partition its data and configuration, and each client organization works with a customized virtual application instance.

Thanks to

Don't Repeat Yourself: Simplify to save time & $$$

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system"

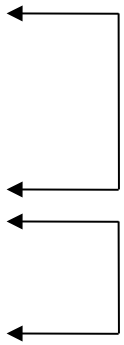
Pragmatic Programmer (Hunt & Thomas)

(or: copy-and-paste encourages problems)

http://www.instructables.com/id/How-To-Create-A-LEGO-Star-Wars-Clone-Army/

Background - Key Trends

- Share more, can do this at different levels in the stack:
  - Hardware partitioning
  - Hypervisor
  - Operating System
  - Containers
  - Runtime
  - Middleware
  - Application

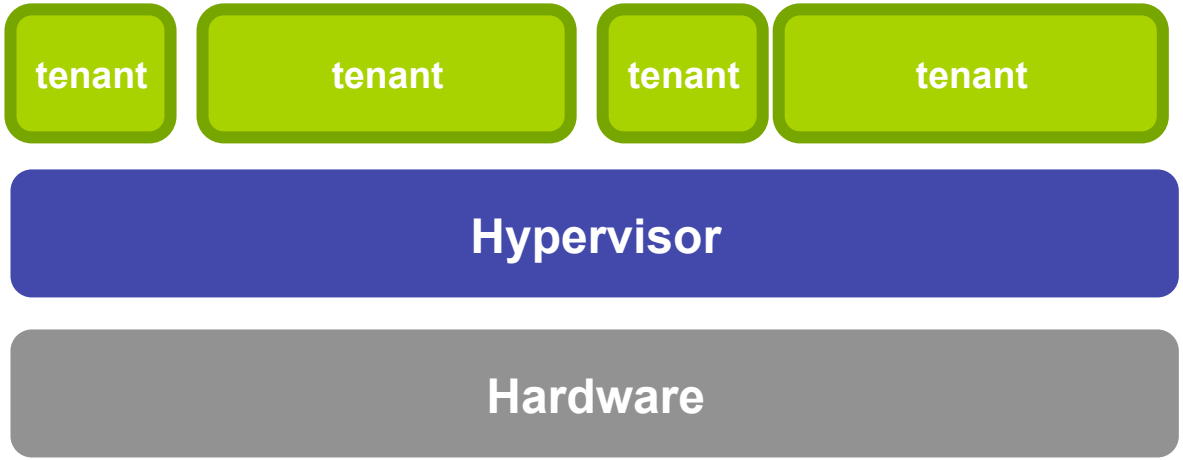


Virtualization – “make it appear as you have dedicated environment/container”

Multitenancy – “share an environment with more than 1 tenant”

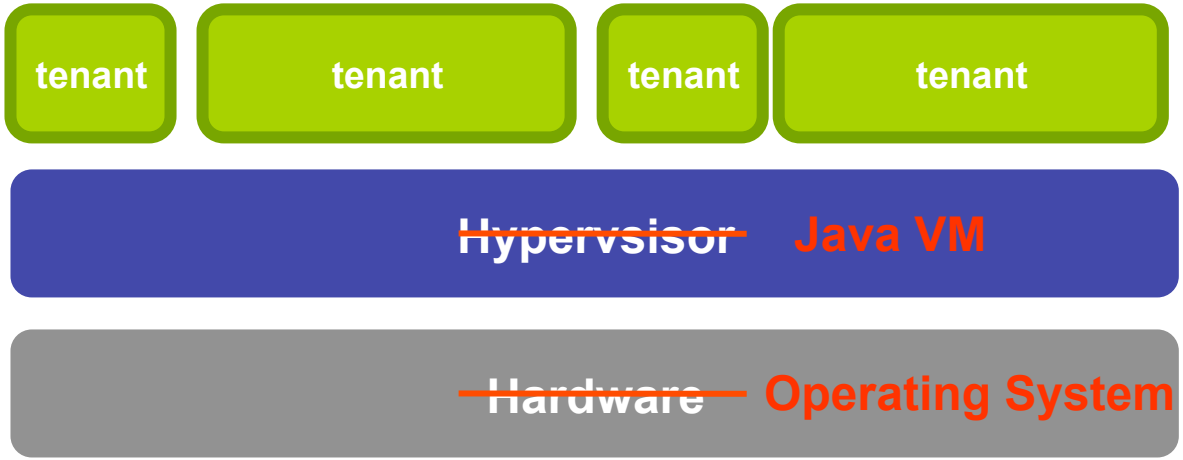
- Continual drive to improve
  - reduce overhead at each level (resource usage, startup)
  - improve isolation at each level
  - more sharing as we move down the stack
- Multitenant Java moves us down one layer in the stack

Hardware Virtualization



- **Hypervisors run multiple applications side-by-side safely**
  - Examples: VMware, kvm, PowerVM, zVM
- **Advantages**
  - Capture idle CPU cycles
  - Automatic de-duplication (RAM)
  - Ability to meter and shift resource toward demand
  - No need to change applications (tenants)

JVM Virtualization



- ~~Hypervisor~~s JVMs can run multiple applications side-by-side safely
- **Advantages**
  - Capture idle CPU cycles
  - Automatic de-duplication (ability to share Java artifacts)
  - Ability to meter and shift resource toward demand
  - No need to change tenant applications

JVM Multitenancy: What is it ?

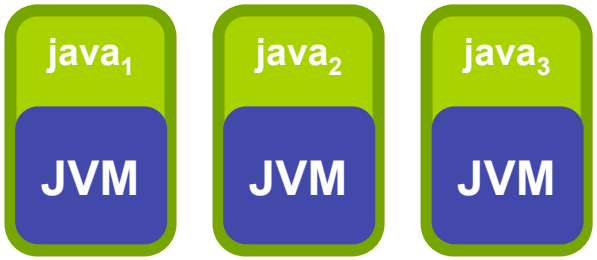
- What we're building: basically a 'virtual JVM'
  - Transparent multitenancy for 100% pure Java applications
  - Opt-in via –Xmt option
  - Shared JVM (javad) process hosts all tenants with in/out/err redirection to launcher
  - JVM-enforced resource controls on Heap, Threads, I/O, and CPU
  - Will behave exactly like a dedicated JVM, only smaller

JVM Multitenancy: What do I get?

- Tech preview in the Java 7.1 release:
  - Full platform evaluation Linux-x86, z/OS, AIX, zLinux, pLinux
- Download from:
  - https://www.ibm.com/developerworks/java/jdk/linux/download.html
- Performance Goals (work in progress)
  - Lower memory usage (classes and JVM internal data structures are shared)
  - 10x density improvement on “Hello World” style applications
  - 3x density improvement on larger OSGi applications (Liberty)
  - Less than 10% throughput degradation on TradeLite
  - Quicker startups: JVM is already up and running
  - Better performance from JIT'd code

Multitenancy: Basics – How does it work

- A standard `java` invocation creates a dedicated (non-shared) JVM in each process



Cost of Dedicated JVM

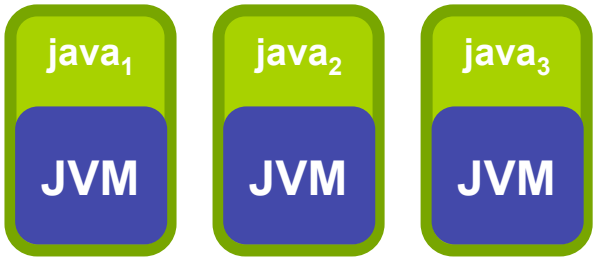
Java Heap consumes 100's of MB of memory
- –Heap objects cannot be shared between JVMs

Just-in-Time Compiler consumes 10's of MB of memory
- –Generated code is private and big
- –Generated code is expensive to produce
  - Steals time from application

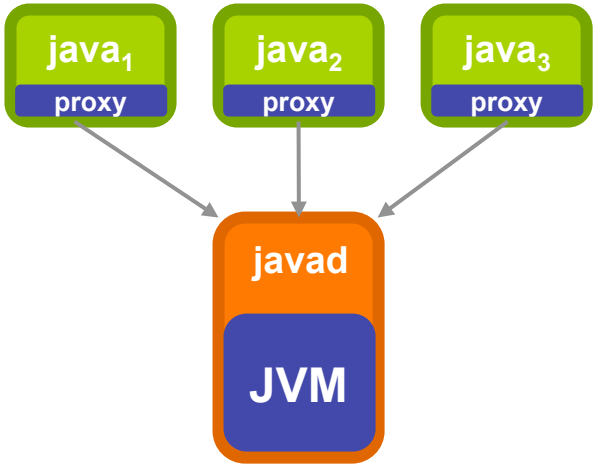
JVM Control structures and threads
- – Heap control structures
- – Multiple compilation threads
- – Multiple GC helper threads

Multitenancy: Basics – How does it work

- A standard **`java`** invocation creates a dedicated (non-shared) JVM in each process



- IBM's Multitenant JVM puts a lightweight 'proxy' JVM in each **`java`** invocation. The 'proxy' knows how to communicate with a shared JVM daemon called **`javad`**.



- **`javad`** is launched and shuts down automatically
- no changes required to the application
- **`javad`** process is where aggressive sharing of runtime artifacts happens

Cost of Virtualized JVM

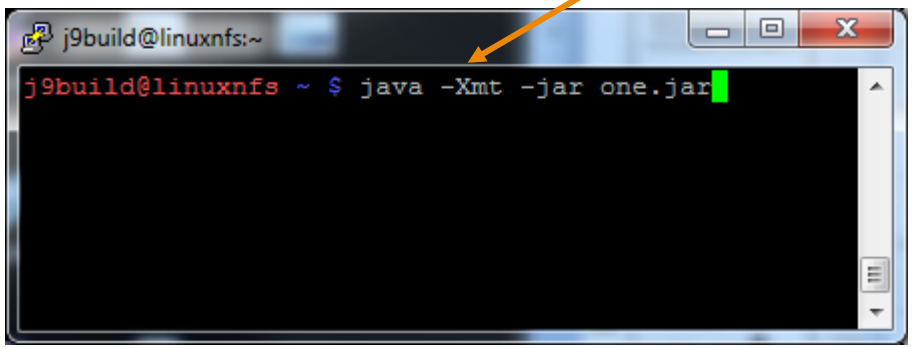
- 1 Heap – no duplication of control structures, helper threads
- 1 JIT – code is compiled once, less memory is used, less CPU used by the JIT
- Shared Classes – Java SDK classes are shared, less memory consumed
- Quicker JVM startup – when a tenant connects, the JVM is already running
- Per tenant instance data is not shared – isolated on the heap by the JVM
- Performance hit for virtualization

Multitenancy: Getting started

- **Multitenancy is an opt-in feature** of IBM JDKs for Java 8 (tech preview 7.1 4Q2013)
  - Just add the `-Xmt` command-line option to opt-in
  - Enables a model very similar to JSR-121: Isolates but doesn't require any new API
- **Daemon startup and communications is handled automatically** by the 'java' launcher
  - One daemon per user to keep permissions aligned between launcher & daemon
  - Launcher:daemon rendezvous accomplished using advertisement files
- **Standard in / out / error streams are connected to daemon**
  - e.g. System.out.println() in the daemon works as expected
  - JVM will multicast messages like dump events to all connected tenants
- **Most standard JVM options are used as-is**
  - -classpath / -jar entries
  - -Dname=value system properties
- **Select JVM options are mapped to tenant-specific values**
  - -Xmx applies to the tenant being launched
  - See documentation for details
- **Daemon-wide options are stored in JAVA_HOME/bin/javad.options file**
- **Documentation available at:**
  - **http://www.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.aix.71.doc/diag/preface/changes_71/overview_mt_evaluation.html**

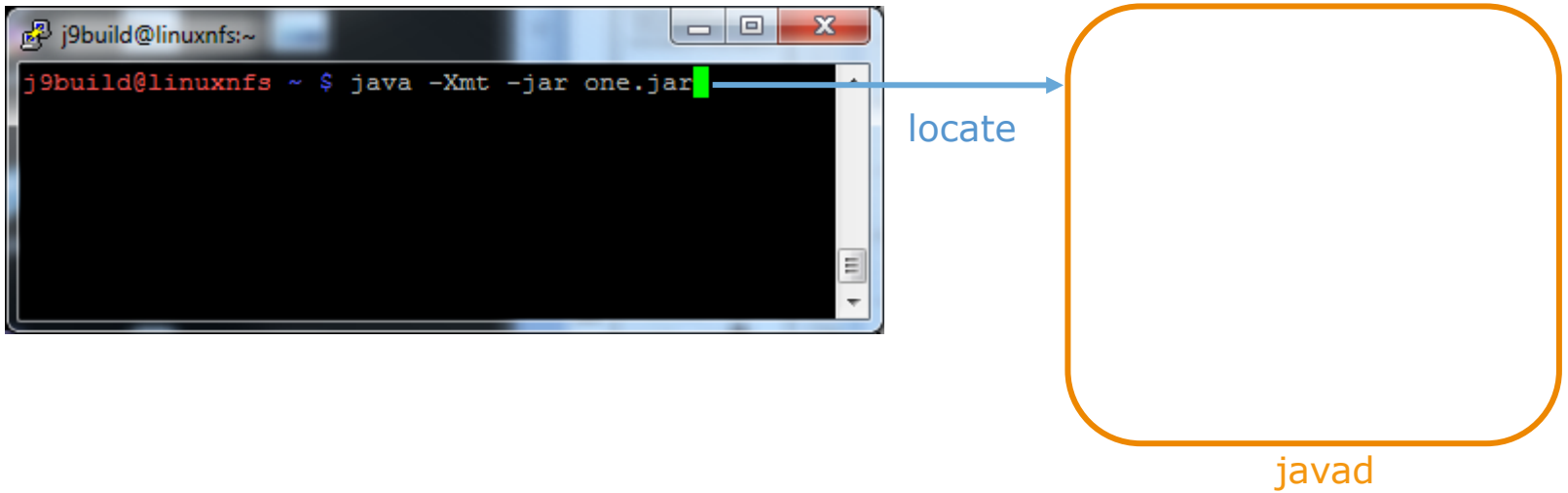
Multitenant JDK: Launch your application

- Opt-in to multitenancy by adding **`-Xmt`**



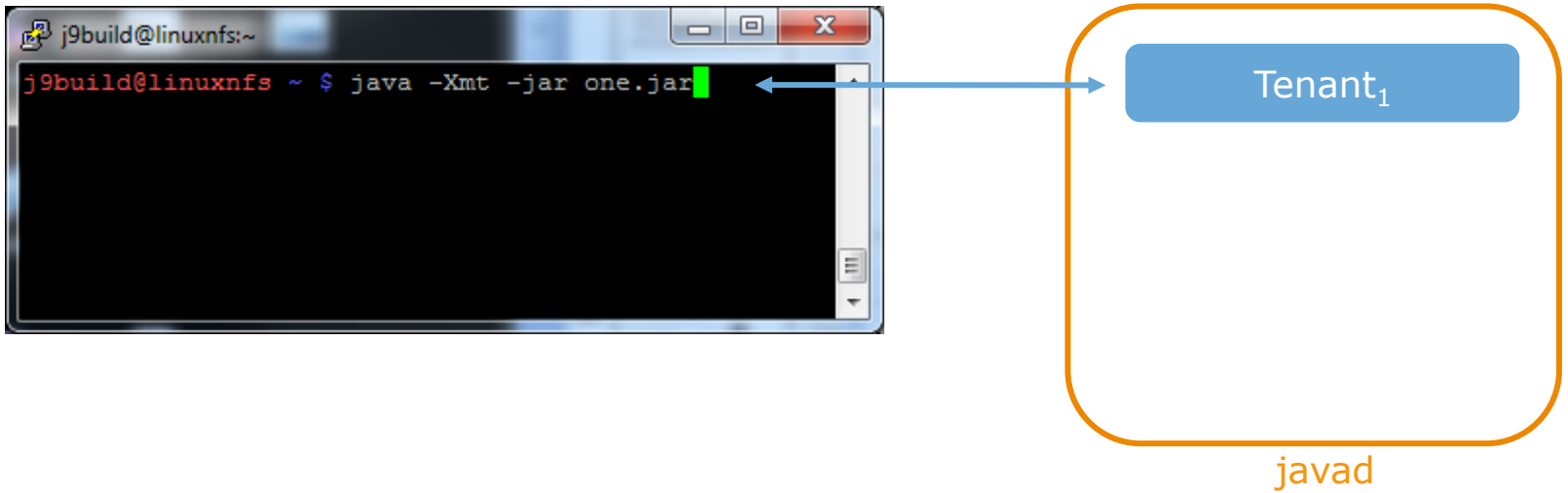
Multitenant JDK: Register with `javad` daemon

- JVM will locate/start daemon automatically



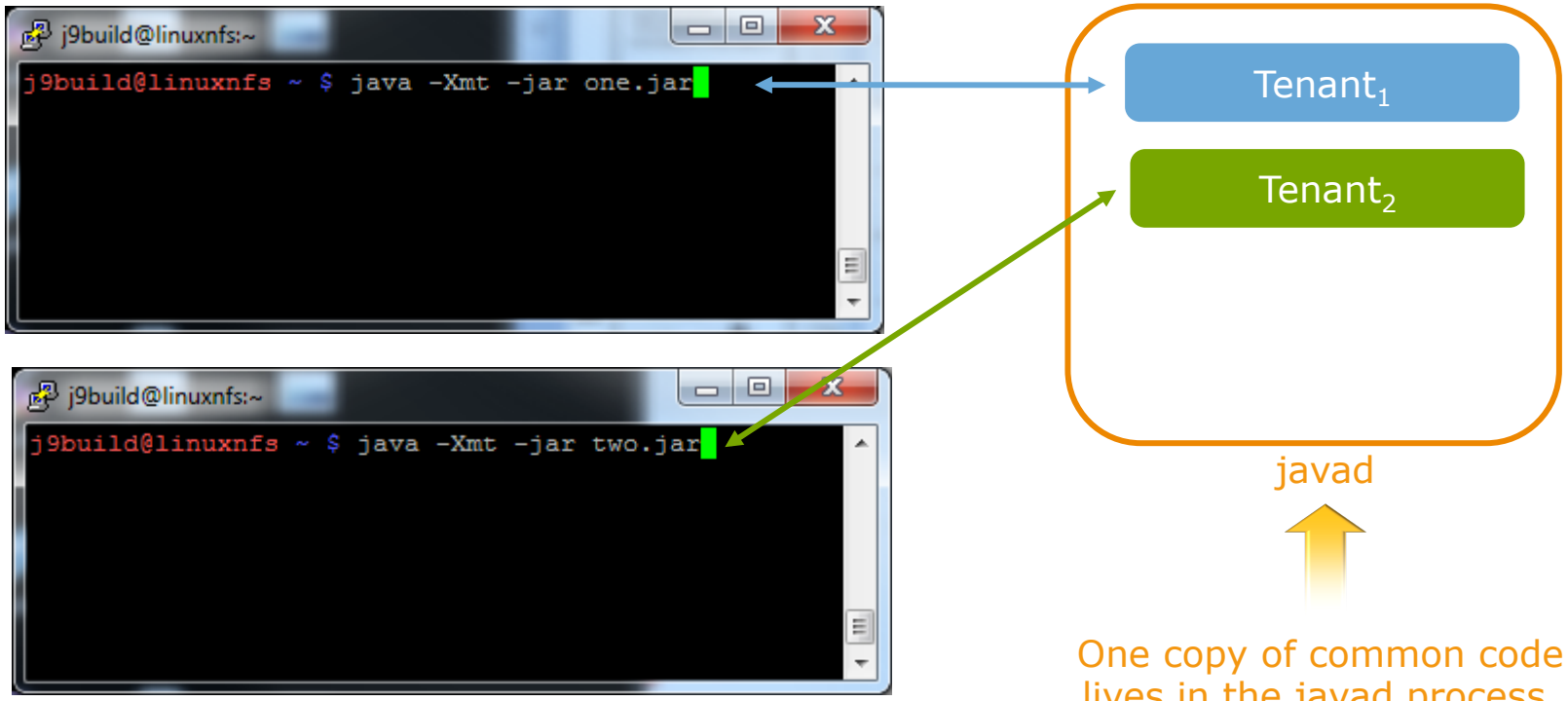
Multitenant JDK: Create a new tenant

- New tenant created inside the **`javad`** daemon



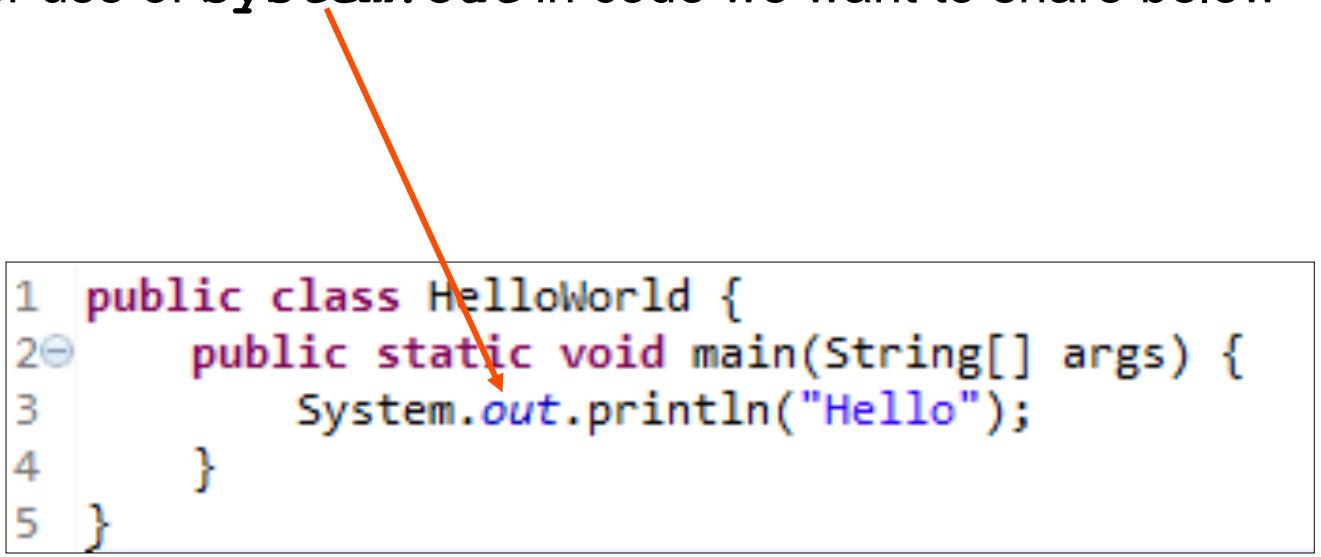
Multitenant JDK: Create a second tenant

- New tenant created inside the **`javad`** daemon



A peek under the covers: Separating State

- Static Variables are a problem for sharing
- Consider use of **`System.out`** in code we want to share below



```
1  public class HelloWorld {
2      public static void main(String[] args) {
3          System.out.println("Hello");
4      }
5  }
```

A peek under the covers: Separating State

```
1  public class HelloWorld {
2      public static void main(String[] args) {
3          System.out.println("Hello");
4      }
5  }
```

```
HelloWorld
// access flags 0x9
public static main([Ljava/lang/String;)V
 L0
  LINENUMBER 3 L0
  GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
  LDC "Hello"
  INVOKEVIRTUAL java/io/PrintStream.println(Ljava/lang/String;)V
 L1
  LINENUMBER 4 L1
  RETURN
```

`getstatic` does 2 things

1. Triggers class initialization on first contact
   - Notable: Each 'tenant' needs to do this
2. Resolves a name (`out`) to a storage location and reads from it
   - Notable : Each tenant needs dedicated storage

Multitenancy: Controlling Resource Consumption

- An opportunity!
- Problem – multiple applications on a single server, one misbehaved app can break all the rest
- Allow the JVM to limit resource consumption of your tenants

Multitenancy: Controlling Resource Consumption

- The second key feature for safe multitenancy is **resource control**
  - Based on JSR-284 for resource configuration management
  - Internally uses a token-bucket algorithm commonly applied to network traffic shaping
- Resources that can be throttled:
  - **CPU** & **Threads**
  - **Heap** memory consumption
  - Disk and Network **I/O**

Multitenancy: Controlling Resource Consumption

- Throttling is controlled using a new **`-Xlimit`** command-line option
  - General form is: **`-Xlimit:`**`<resource_name>=<min_limit>-<max_limit>`
  - `<min_limit>`: Specifies the minimum amount of the resource that must be available for the tenant to start. This value is optional.
  - `<max_limit>`: Specifies the maximum amount of the resource that the tenant is allowed to use.
- Examples:
  - **`-Xlimit:cpu=10-30`**
    - requires a 10% share of the processor to start and limits processor consumption to 30%.
  - **`-Xlimit:threads=5-20`**
    - requires a minimum reservation of five threads and an upper limit of 20
  - **`-Xmx20m`**
    - Limit heap consumption to 20 megabytes

Multitenancy – When and Where?

- Advantages
- Disadvantages
- Possible use cases

Multitenancy Sweet spot

- **How low can you go?**
  - Simple ('Hello World') applications showing per-tenant sizes of ~170 KB of heap
  - This equates to a **5-6x** more applications running on the same hardware
- **Performance**
  - Target is 10% overhead, still a work in progress
- **Second-run start-up times are significantly better**
  - Faster because the JVM is already up and running
- **Application Sweet spot:**
  - **One of:**
    - Relatively large class:heap ratio (JRuby and other JVM languages)
    - Require fast startup: run-and-done / batch
    - Workloads with varying busy:idle cycles – MT JDK is better at shifting resource between tenants
  - **100% pure Java Code**

Multitenancy: Caveats & Limitations

- **Main Limitations of the MT Model**

  JNI Natives
  - The operating system allows the shared JVM process to load only one copy of a shared library. Only native libraries present on the bootclasspath of the JVM usable.

  JVMTI
  - Because debugging and profiling activities impact all tenants that share the JVM daemon process, these features are not supported in the multitenant JVM process model. Note: we do have per-tenant -javaagent: support.

  GUI programs
  - Libraries such as the Standard Widget Toolkit (SWT) are not supported in the multitenant JVM process model because the libraries maintain a global state in the native layer.

– Full list available at:
http://www-01.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.aix.71.doc/user/mt_limitations.html

Use Cases

- **MT-UC1 – Small Application Consolidation**
- MT-UC2 – Run and Done
- **MT-UC3 – Resource Time Sharing**
- MT-UC4 – Resource Limiting based on SLA
- **MT-UC5 – Resource Limiting for Safety**
- MT-UC6 – Memory Cost Sensitive Environments
- **MT-UC7 – Health Monitoring and Recovery**

MT-UC1 – Small Application Consolidation

- Key attributes
  - Customer has multiple small applications
  - Non EE deployment OR more isolation needed between applications than provided by EE deployment OR need per application Middleware instances (ex liberty) due to management/operational requirements
  - Application memory/CPU overhead low compared to JVM overhead
- MT Benefit:
  - Lower total footprint/memory requirements
    - Limit overhead to that of 1 JVM versus many
    - Limit heap to 1 shared head heap

MT-UC2 – Run and Done

- Key attributes
  - Short running application with multiple invocations
  - Startup/Shutdown dominates run time
  - Need Isolation between invocations
  - Examples: Ant scripts, compilation with javac, IMS, Z Batch, JRuby scripts, Jython scripts etc.
- MT Benefit:
  - Faster startup/shutdown
    - Avoid full JVM startup/shutdown for each invocation

MT-UC3 – Resource Time Sharing

- Key attributes
  - Customer has multiple applications that have load at different times
  - Non EE deployment OR more isolation needed between applications than provided by EE deployment OR need per application Middleware instances (ex liberty) due to management/operational requirements
- MT Benefit:
  - Lower total footprint/memory requirements
    - Limit overhead to that of 1 JVM versus many
    - Shared heap sized to match concurrent maximum instead of sum of all application maximums

MT-UC4 – Resource Limiting based on SLA

- Key attributes
  - Customer has multiple applications sharing same OS instance
  - Some applications have higher SLA levels than others
- MT Benefit:
  - Able to control CPU, Network IO , File IO resource usage to favor application with higher SLA

MT-UC5 – Resource Limiting for Safety

- Key attributes
  - Customer has multiple applications sharing same OS instance
  - Some applications un-trusted or buggy, concern they will affect performance of other applications.
- MT Benefit:
  - Able to control CPU, Network IO , File IO resource usage to limit maximum impact of “runaway” application

MT-UC6 – Memory Cost Sensitive Environments

- Key attributes
  - Memory is costly for environment (e.g. Legacy hardware)
  - Shares attributes of one of earlier use cases
- MT Benefits
  - Footprint savings more compelling than in other environments.

MT-UC7 – Health monitoring and recovery

- Key attributes
  - Health monitoring/recovery runs in JVM with application
  - Application failures should not affect health monitoring (ex OOM on app)
- MT Benefits
  - Ability to ensure minimum amount of memory available to health monitoring/recovery components

Demo - Scenario

- JVM Health monitoring
  - Want heartbeat to track “liveness” of server
  - Need this to run reliably as long as application is still running
  - Requires some memory and cpu to generate heartbeat
  - Simulate in demo with thread that prints out “heartbeat” at 2 second interval
- Application Transactions
  - Transactions submitted from external system
  - Use variable amount of cpu depending on request
  - If transaction uses too much memory it can starve heartbeat thread
  - Simulate with thread(s) that uses as much cpu as they can

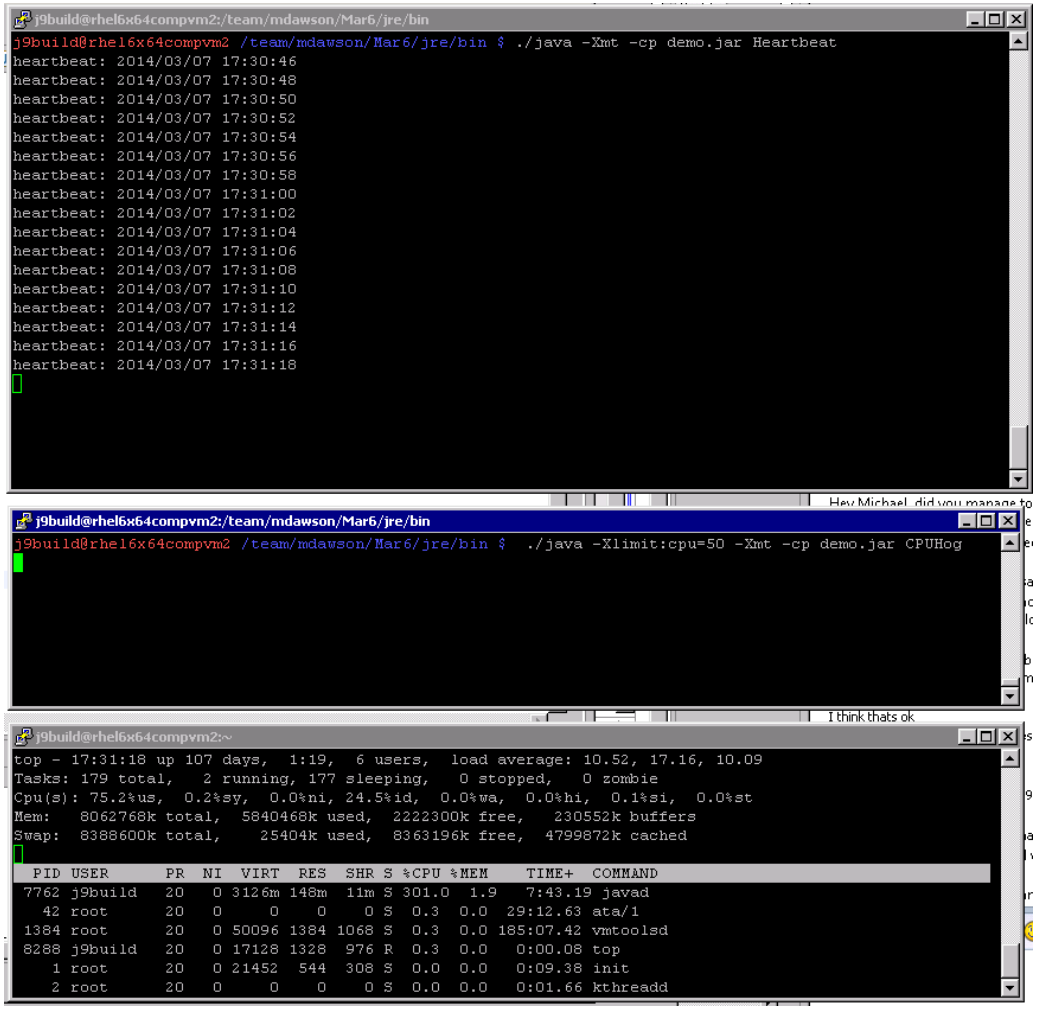
Demo – what happens today

- Run HeartbeatAndCPUHog
- ./java -cp demo.jar HeartbeatAndCPUHog
- Shows running both heartbeat thread and transaction in regular jvm
- Note that times between heartbeat messages stretch out once cpu hog starts

```
j9build@rhel6x64compvm2 /team/mdawson/Mar6/jre/bin $ cat javad.options^C
j9build@rhel6x64compvm2 /team/mdawson/Mar6/jre/bin $ ./java -cp demo.jar HeartbeatAndCPUHog
heartbeat: 2014/03/07 17:23:26
heartbeat: 2014/03/07 17:23:28
heartbeat: 2014/03/07 17:23:30
heartbeat: 2014/03/07 17:23:32
heartbeat: 2014/03/07 17:23:34
CPU Hog starting
heartbeat: 2014/03/07 17:23:36
heartbeat: 2014/03/07 17:23:44
heartbeat: 2014/03/07 17:23:53
heartbeat: 2014/03/07 17:24:03
heartbeat: 2014/03/07 17:24:12
heartbeat: 2014/03/07 17:24:22
heartbeat: 2014/03/07 17:24:33
```

Demo with MT

- ./java -Xmt -cp demo.jar Heartbeat in one window
- ./java -Xlimit:cpu=50 -Xmt -cp demo.jar CPUHog in another window
- Top in third window
- Note that heartbeat remains consistent even after hog starts
- Top shows that cpu varies, but average looks to be around 50%
- You might have to play with limit depending on your machine as heartbeat task does use reasonable amount of cpu to show affect.



IBM Knowledge Center

The new home for IBM product documentation

- A one-stop shop for IBM technical publications; replaces Information Centers
- Create collections that apply to you
- Collaborate with IBM and your colleagues; share comments and rate pages

Want to learn more about IBM Knowledge Center?

- Come and visit the stand
- Look for representatives wearing these badges

IBM Knowledge Center collections that might interest you:

IBM SDK, Java Technology Edition	http://ibm.biz/javasdkdocs
IBM Monitoring and Diagnostic Tools for Java	http://ibm.biz/javasdktoolsdocs
IBM WebSphere Real Time	http://ibm.biz/wrtdocs

Key Links and Contacts

- Download
  - https://www.ibm.com/developerworks/java/jdk/linux/download.html
- Documentation
  - http://www.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.aix.71.doc/diag/preface/changes_71/overview_mt_evaluation.html
- **Contacts for feedback**
  - **Iain Lewis (iain_lewis@uk.ibm.com)**
  - **Michael Dawson (michael_dawson@ca.ibm.com)**