



Java 1.8 (Java 8) Update

A VSI Just-In-Time Presentation ☺

Camiel Vanderhoeven & Brett Cameron

OpenVMS Boot Camp 2016 – 10014b

26-SEP-2016



Java 1.8 (Java 8) Update

This information contains forward looking statements and is provided solely for your convenience. While the information herein is based on our current best estimates, such information is subject to change without notice.

Agenda

- Java 8 port
 - Project outline
 - Project approach
 - Current status
- Java 7 new features
- Java 8 new features

Java 8 Port : Project Outline

- Joint HPE/VSI project
- Java 1.8 for HPE and VSI OpenVMS (8.4+, IA64)
- Starting point:
 - Java 1.6 port to OpenVMS
 - Java 1.8 port to HP-UX
- Updating a number of Java applications as part of the port: Tomcat, Axis2, Ant, Wsit, ...
- Probably around 90% complete (depending on how you measure)
- Field test planned for calendar Q4 2016
- Release planned for calendar Q1 2017

Java 8 Port : Project Outline

- Large amount of work
 - HotSpot:
 - 99% C++
 - 2,836 source files in 58 directories
 - 1,383,700 lines of code
 - 15% of code is very CPU/OS specific
 - ~2,500 code changes required
 - JDK:
 - 83% Java
 - 26,706 source files in 1,005 directories
 - 6,576,132 lines of code
 - ~1,200 code changes required

Java 8 Port : Project Outline

JDK	Java Language	Java Language						
	Tools & Tool APIs	java	javac	javadoc	jar	javap	jdeps	Scripting
		Security	Monitoring	JConsole	VisualVM	JMC	JFR	
		JPDA	JVM TI	IDL	RMI	Java DB	Deployment	
	Deployment	Internationalization			Web Services		Troubleshooting	
		Java Web Start			Applet / Java Plug-in			
		JavaFX						
		User Interface Toolkits	Swing		Java 2D	AWT	Accessibility	
	Drag and Drop		Input Methods	Image I/O	Print Service	Sound		
	Integration Libraries	IDL	JDBC	JNDI	RMI	RMI-IIOP	Scripting	
	JRE Other base Libraries	Beans	Security	Serialization		Extension Mechanism		
		JMX	XML JAXP		Networking	Override Mechanism		
		JNI	Date and Time	Input/Output		Internationalization		
	Lang and Util Base Libs	lang and util						
		Math	Collections	Ref Objects		Regular Expressions		
		Logging	Management	Instrumentation		Concurrency Utilities		
		Reflection	Versioning	Preferences API		JAR	Zip	
	Java Virtual Machine	Java HotSpot Client and Server VM						

Java 8 Port : Project Outline

- Scope of work:
 - Port : add VMS-specific functionality
 - Part merge from Java 6 code
 - Part new development
 - Java 6 was mixed 64-bit (HotSpot) / 32-bit (JDK)
 - Java 8 is pure 64-bit
 - Test and debug
 - Ad nauseam
 - Using real Java code: clojure, ActiveMQ, Tomcat, DaCapo, ...
 - Adapt Java-based applications we ship
 - Documentation

Java 8 Port : Project Approach

- Porting approach:
 - Different “rounds”
 - Round 1: Linux exploration
 - Before even seeing the Oracle/HP code
 - Determine the minimum set of changes required to turn Linux OpenJDK Java 6 into Java 8
 - End product: insight in the Java codebase, list of possible code changes to check for
 - Round 2: Applied insight
 - Selectively merge Java 6 code changes and add new code changes based on the Linux-based exploration. Harder changes left for later
 - End product: lots of “easy changes” out of the way

Java 8 Port : Project Approach

- Porting approach:
 - Round 3: hack-a-thon / codefest
 - Debug-driven development
 - Finding and fixing bugs
 - Non-systematic, based on programmers' instincts
 - End result: somewhat functional Java8, but crash-prone

This can't be good...

Iteration 1: $\text{SQRT}(4.0) = 2.0$

Iteration 2: $\text{SQRT}(4.0) = 2.0$

...

Iteration 10028: $\text{SQRT}(4.0) = 2.0$

Iteration 10029: $\text{SQRT}(4.0) = 2.0$

Iteration 10030: $\text{SQRT}(4.0) = 29.8562832991$

Iteration 10031: $\text{SQRT}(4.0) = 29.8562832991$

Iteration 10032: $\text{SQRT}(4.0) = 29.8562832991$

Iteration 10033: $\text{SQRT}(4.0) = 29.8562832991$

...

Java 8 Port : Project Approach

- Porting approach:
 - Round 3: hack-a-thon / codefest
 - Debug-driven development
 - Finding and fixing bugs
 - Non-systematic, based on programmers' instincts
 - End result: somewhat functional Java8, but crash-prone
 - Round 4: Systematic drudgery
 - Individually reviewing every single VMS-specific line of code in the Java 6 and Java 8 code bases
 - Making reasoned corrections to the code
 - End result: functional Java8, but still with some omissions and bugs

Java 8 Port : Project Approach

- Porting approach:
 - Round 5: Debugging
 - Back to debugging, still some nasty bugs out there
 - Add more applications to the test mix to find more bugs
 - End result: nearly bug-free Java 8, but with some missing bits
 - Round 6: Filling in the blanks
 - Add VMS support to new Java features (like the re-coded graphics system)
 - End result: field-test-ready Java8
 - After field test: probably another round of debugging

Java 8 Port : Current Status

- In the middle of round 5, debugging / round 6, filling in the blanks
- X-Windows / Motif integration

Most of it is working...

The screenshot shows the Apache Manager web interface. At the top, there's the Apache Software Foundation logo and the Apache cat logo. The main heading is "Server Status". Below it, there are navigation links: "Applications", "HTML Manager Help", "Manager Help", and "Complete Server Status".

The "Server Information" section contains the following table:

Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture	Hostname	IP Address
Apache Tomcat/8.5.4	1.8.0.03-vms-rc1	Oracle Corporation	OpenVMS	XE0V-B4N	ia64	exec14	10.10.116.196

The "JVM" section shows memory usage: memory: 88.22 MB Total memory: 140.50 MB Max memory: 196.00 MB. Below this is a table of memory pools:

Memory Pool	Type	Initial	Total	Maximum	Used
PS Eden Space	Heap memory	55.00 MB	28.50 MB	28.50 MB	27.02 MB (94%)
PS Old Gen	Heap memory	147.00 MB	89.50 MB	147.00 MB	21.98 MB (14%)
PS Survivor Space	Heap memory	9.00 MB	22.50 MB	22.50 MB	3.29 MB (14%)
Code Cache	Non-heap memory	2.00 MB	3.12 MB	256.00 MB	3.03 MB (1%)
Metaspace	Non-heap memory	0.00 MB	24.50 MB	-0.00 MB	23.78 MB

The "ajp-nio-8009" section shows thread and request statistics:

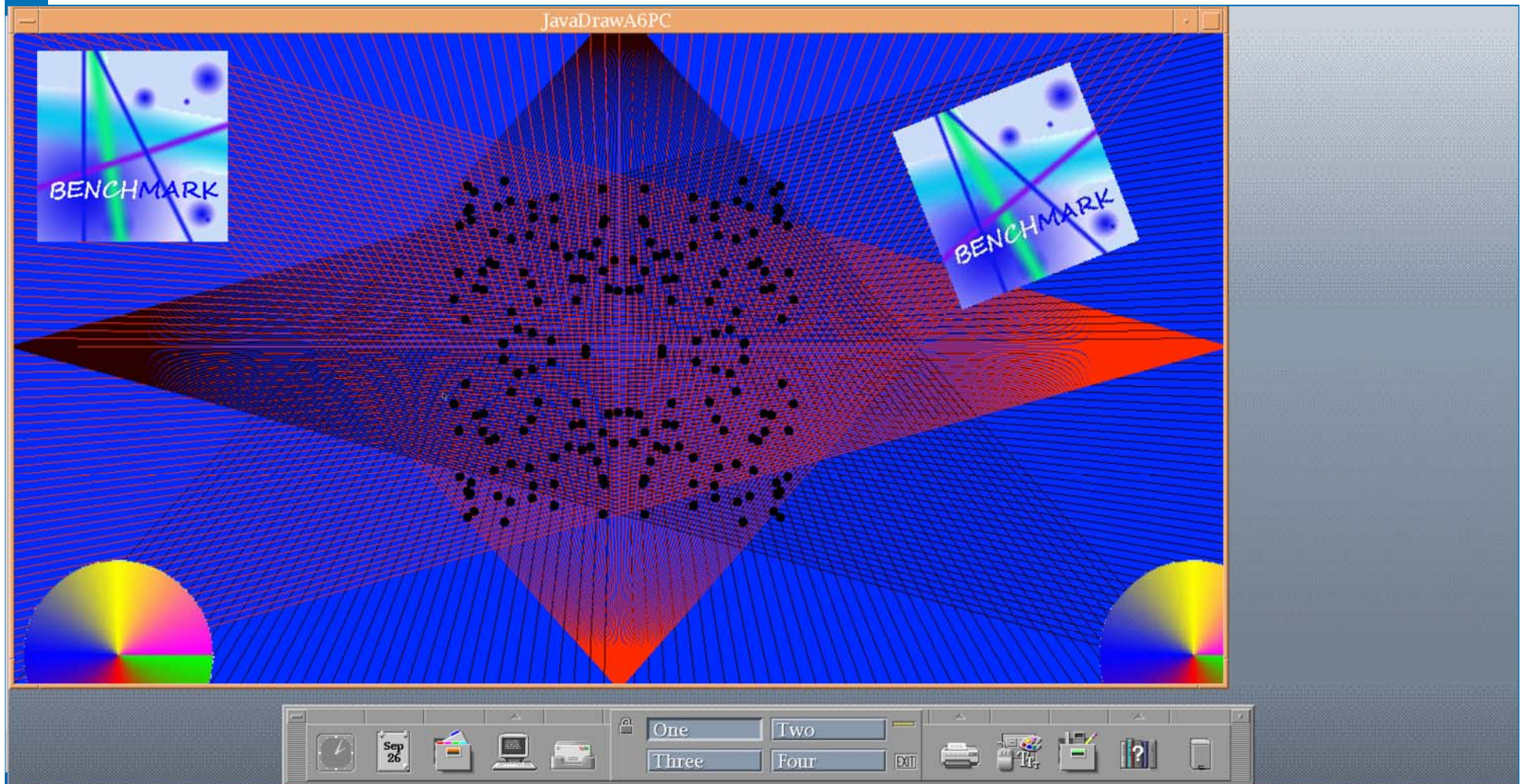
threads: 200 Current thread count: 0 Current thread busy: 0 Kept alive sockets count: 0
 processing time: 0 ms Processing time: 0.0 s Request count: 0 Error count: 0 Bytes received: 0.00 MB Bytes sent: 0.00 MB

Below this is a table for request statistics:

Stage	Time	B Sent	B Recv	Client (Forwarded)	Client (Actual)	VHost	Request
-------	------	--------	--------	--------------------	-----------------	-------	---------

Legend: parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

Most of it is working...



Java 7 new features

- Diamond Operator

- Java 6:

```
Map<String, List<Trade>> trades = new  
    TreeMap<String, List<Trade>> ();
```
- Java 7:

```
Map<String, List<Trade>> trades = new TreeMap<> ();
```

- Strings in switch statement

- Java 6: Series of if / else if statements
- Java 7: switch statement with strings as case labels

- Numeric literals with underscores

- Java 6:

```
int million = 1000000;
```
- Java 7:

```
int million = 1_000_000;
```


Java 7 new features

- Automatic resource cleanup

- Java 6:

```
try {
    FileOutputStream fos = new
FileOutputStream("movies.txt");
    DataOutputStream dos = new DataOutputStream(fos);
    dos.writeUTF("Java 7 Block Buster");
} catch(IOException e) {
    e.printStackTrace();
} finally {
    dos.close();
    fos.close();
}
```
- Java 7:

```
try (FileOutputStream fos = new
FileOutputStream("movies.txt");
    DataOutputStream dos = new DataOutputStream(fos))
{
    dos.writeUTF("Java 7 Block Buster");
} catch(IOException e) {
    e.printStackTrace();
}
```

Java 7 new features

- Multi-exception catch

- Java 6: `try {`

- `...`
 - `} catch (ExceptionTypeA e) {`
 - `e.printStackTrace();`
 - `} catch (ExceptionTypeB e) {`
 - `e.printStackTrace();`
 - `}`

- Java 7: `try {`

- `...`
 - `} catch (ExceptionTypeA | ExceptionTypeB e) {`
 - `e.printStackTrace();`
 - `}`

Java 7 new features

- New file system I/O (NIO 2.0)
 - More predictable across platforms
 - WatchService for file change notifications
- Easy “Fork and Join” parallelization
 - Using ForkJoinPool and ForkJoinTask
- Dynamic run-time function invocation
 - Not used by the Java language, but useful for dynamically-typed languages running on the JVM.

Java 8 new features

- Default method implementation in interface
 - Interface methods can now have a “default” implementation
 - If a class implements two interfaces that both provide a default implementation of the same method, the class has to override this method

Java 8 new features

- Functional interfaces and lambda expressions
 - Functional interface: interface with a single abstract method

- Java 7:

```
Runnable r = new Runnable() {  
    @Override  
    public void run() {  
        System.out.println("My Runnable");  
    }  
};
```

- Java 8:

```
Runnable r = () -> {  
    System.out.println("My Runnable");  
}
```

- Or even:

```
Runnable r = () -> System.out.println("My Runnable");
```

Java 8 new features

- **forEach** function

- **Java 7:**

```
Iterator<Integer> it = myList.iterator();
while (it.hasNext()) {
    Integer i = it.next();
    System.out.println(i);
}
```

- **Java 8:**

```
myList.forEach(new Consumer<Integer>() {
    public void accept(Integer i) {
        System.out.println(i);
    }
});
```

- **Consumer is a functional interface, so with Lambda:**

```
myList.forEach(i -> System.out.println(i));
```

Java 8 new features

- Stream API for Big Data

- Parallel (or sequential) processing of collections

```
Stream<Integer> dataSet = myList.parallelStream();  
Stream<Integer> underTen = myList.filter(i -> i<10);  
underTen.forEach(i -> System.out.println(i));
```

- New Time API

- Standardized approach to using time and date on Java

- Improvements in Collection, Concurrency, and IO API's

Credits

- Some Java 7 code examples taken from O'Reilly Publishing / Madhusudhan Konda

