What's new in the JVM in Java SE 8

Gil Tene, CTO & co-Founder, Azul Systems

AZUL SYSTEMS

About me: Gil Tene

- co-founder, CTO
  @Azul Systems
- Have been working on JVMs and on "think different" GC approaches since 2002
- Created Pauseless & C4 core GC algorithms (Tene, Wolf)
- A Long history building Virtual & Physical Machines, Operating Systems, Enterprise apps, etc...
- JCP EC Member...

* working on real-world trash compaction issues, circa 2004

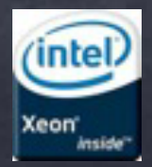
AZUL SYSTEMS

About Azul

- We make scalable Virtual Machines
- Have built "whatever it takes to get job done" since 2002
- 3 generations of custom SMP Multi-core HW (Vega)
- **Zing: Pure software for commodity x86**
- Known for Low Latency, Consistent execution, and Large data set excellence

About Azul

- We also do other things with JVMs...
- Zulu: A commercialized, supported version of OpenJDK
- Long term, commercial support for Java SE 6, Java SE 7, (and 8 as it comes out)
- Free & Open Source, with changes contributed back to OpenJDK
- Windows & Linux . . .
- Azure & EC2 . . .



AZUL SYSTEMS

This talk will focus on the JVM

- Java SE 8 brings many changes
- JDK
- JVM
- Specific implementation considerations (OpenJDK, etc.)
- We will focus on the JVM, with some notes about major pushes in specific implementations

Semantic changes for the JVM

- Not that many...
- Lambda Expressions, Virtual Extension Methods (JEP126)
- Parameter Reflection (JEP118)
- Type annotation (JSR 308)
- Class file format changes (version 52.0)

Lambda Expressions

- Lambda Expressions, Virtual Extension Methods
- JDK: Parallel Collections, Stream API

- Semantic Implications for the JVM:
  - Leverages JSR292 (already there in Java SE 7)
  - New (in Java SE 8) Virtual Extension Methods support Default Methods capability in interfaces

Lambda Expressions: Performance

- JDK and generic user code will now make heavy use of JSR292, InvokeDynamic, Method Handles
- Performance of Lambda expressions and MethodHandles becomes critical
- Specific (OpenJDK, Java SE 8 RI) Implementation note:
- Drove a new Lambda-Form implementation of Method Handle
  - Lambda-Form based implementation shifts much of the previously hard-wired JVM MethodHandle implementation to generated Java code that is then JIT-compiled and optimized at runtime.
- Drove changes to JIT inlining policies and heuristics
  - "Late inlining" added to support the deep inlining needed for performant Lamda-Form execution

Some Implementation-specific notes

- OpenJDK 8 (Java SE 8 RI) brings some additional implementation-specific changes:
- E.g. Enhanced Verification Errors (JEP136)
- E.g. Improved Intrinsic support for atomics and fences
  - Unsafe adds intrinsic support for getAndAddX, getAndSetX.
  - JDK Atomic APIS implementations replace CAS-loops for some common atomic operations with single call (translates to single instruction on e.g. x86).
  - New Unsafe intrinsics for ordering/fencing (JEP171). No (pure, safe) Java APIs exposed yet.
- E.g. Leveraging CPU instructions for AES cryptography (JEP164)

Some Implementation-specific notes (Cont.)

- Some internally useful performance-affecting annotations
- E.g. (internal, for trusted code) support for @Stable
  - enables contant folding for lazily evaluated variables
  - improves performance in trusted JDK code
    - "Dangerous" to expose to non-trusted code...
- E.g. (internal, for trusted code) support for @Contended (JEP142)
  - Facilitates cache-layout for contended fields
  - Helps avoid false-sharing bottlenecks in concurrent code
  - Currently available only in JDK (security concerns)
  - Will hopefully be made more widely available in the future

Some Implementation-specific notes (cont.): PermGen

- Removal of PermGen, replaced with MetaSpace (JEP122)
- Major change in JVM-internal handling of long lived, class-related information
- Intended to reduce GC-related and fixed-size issues historically associated with PermGen
- New MetaSpace is separate from Java heap. Still garbage collected but with specific sematics
- Expected to alleviate issues around running out of PermGen space and thrashing GC when PermGen come under pressure due to class loading activity.
- New class loading pressure expected due to Lamda-Form use is a potential driver
- Note: This is specific to OpenJDK and the RI, and is not a Java SE 8 feature item. Other JVMs (e.g. Zing) that provide an elastic, configuration-free, and pause-free PermGen will not need the new MetaSpace.

Summary

- Java SE 8 includes major changes to the language & libraries
- The majority of these changes are in the JDK (library code, written in Java) and in the surrounding tools (e.g. javac)
- The JVM is changing in small increments
  - Semantic changes are mostly around default methods in interfaces, and type annotations (both drive class file change)
- JVM implementations (including OpenJDK 8, the Java SE 8 RI):
  - Performance changes are driven by expectation for heavy use of JSR292
  - Many of the internal performance changes are not specific to Java SE 8, and use it mostly as a release as a release vehicle
  - Expect significant additional performance and footprint work (around Lambda-Forms, etc.) to appear in updates over time.