




ORACLE[®]

JVM Language Summit
Getting Started Guide

Cameron Purdy, VP of Development, Oracle

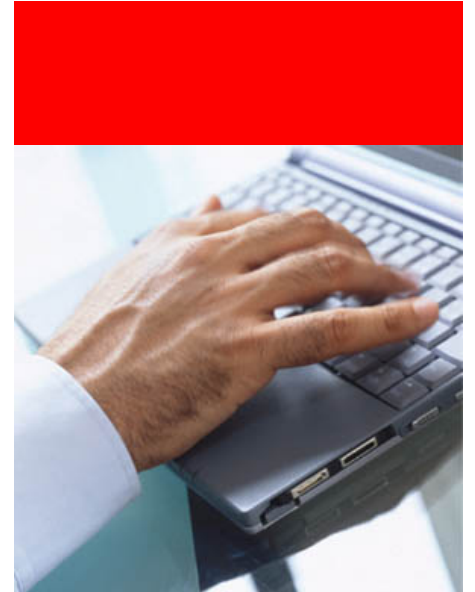


The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.



Agenda

- Introduction
- Obligatory Top-10 List
- Rambling
- Conclusion



About the Speaker: Cameron Purdy

- '97 – My first Java disassembler
- '98 – assembler/disassembler
- '98 – Java method compiler
- '99 – JavaScript method compiler
- '00 – AOP ClassLoader (TDE)
- '01 to present – Coherence
- '02 to present – JSR 107 :-o
- '04 – x86 on Azul (toy project)
- '05 to present – XVM (toy project)
- '11 – Java EE 7 (a *real* project)

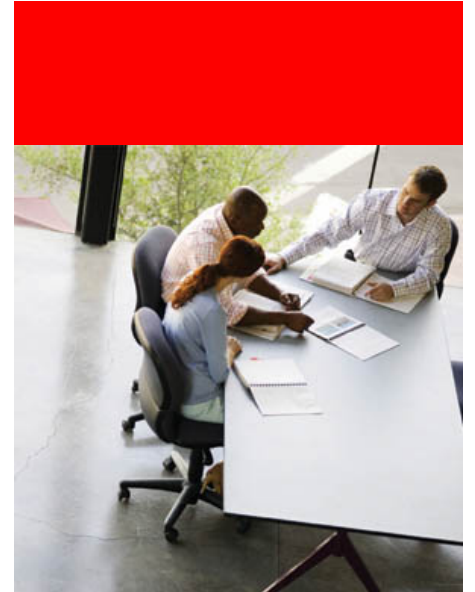




Disclaimer

- I don't work on the JVM
- I don't manage work on the JVM
- I don't write language implementations for the JVM
- I'm definitely out-of-date
- I can be wrong

Top 10 JVM erroneous zones





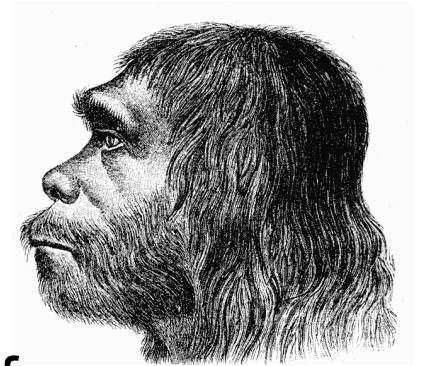
10. Immutability

It's time for a change

- How did we get so far without it?
 - String, intrinsic types & wrappers: Workable
 - Arrays, Date: Unusable in interfaces without immutability
 - What about objects that need to *become* immutable?
- Why not just swap the vtbl?
 - Methods that mutate fields just throw `IllegalStateException`
 - It would be easier if there were *properties* instead of fields
- So much can build on top of immutability
 - Escape-ability (compaction and concurrency optimizations)
 - Ability to safely define a class at runtime
 - “Primitive” types, i.e. no need for wrapper classes

9. Seriously? Primitive types?

In need of some evolution...

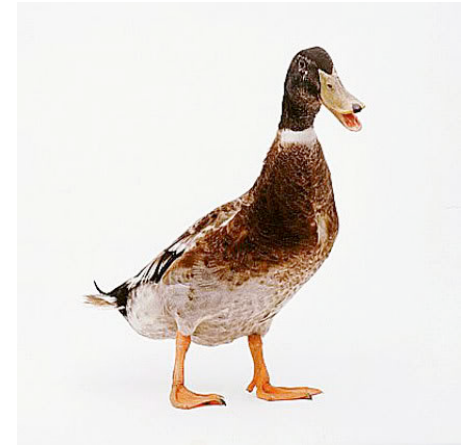


- Current primitive types have well-known interfaces
 - LADD, LCMP, LMUL, LDIV, ...
- Unfortunately, interface == implementation
 - We'll fix that one next
- Goal: Current code natively compiles the same way
 - Except perhaps with respect to auto-boxing
- Benefits
 - Reduction from two type systems to one
 - Single interface across `int` and `Integer`
 - Extensible through traits
 - Current op-codes become short-form for specific methods

8. Interface vs. Implementation

Why do I have to know what class it is?

- String
 - ASCII? UTF-8? UTF-16? 32-bit code points?
- Array
 - Interface defined by op-codes (ALOAD, ASTORE, ARRAYLENGTH, etc.)
 - Current implementation is JVM intrinsic
 - Could be implemented by Java, e.g. `java.util.List`
- Duck typing: It's about the interface
 - All types *have* an interface; `interface = f(type)`
- Traits, Classes, Interfaces
 - Aren't they actually all the same thing?





7. Properties

When you can't get the same high from name-mangling

- Properties are like fields whose `getField` & `putField` can be intercepted & overridden
 - It's as if any property method implementation always has a `super()` provided by the JVM
 - If `super()` is called, then the underlying "field" must exist
- Not just `get()` and `set()`, e.g. atomics:
 - `SetIf`
 - `GetSet`
- For backwards compatibility, fields can become a simple (JVM-supplied) form of a property



6. Missing the most Obvious Intrinsic Types

$2.0 + 2.0 = 3.99999999999999999999794$

- Decimal
 - So what if the world's most popular platform for business applications doesn't support a decimal type?
 - IEEE 754-2008 a.k.a. IEEE 754r – defines decimal types!
- 64-bit is *sooooooooooooo* 2005 ...
 - Need 128-bit integer, binary floating point, decimal
- Binary String
- A working Date, Time & Date/Time
- Date & Time intervals
 - No more “long millis”?
- Unsigned types?

5. Wanted: A *Real* Runtime Model

Just because it's runtime doesn't mean it's not dev time



- Class as a first-class object
 - Ability to define classes, including properties and methods, at runtime
 - Modifying an existing class that already has instances could be a problem
- Access to Source and Byte Code at runtime
 - These should just be properties of a Method
 - Yet another reason why the JVM needs to include an assembler (and throw in the compiler while you're at it!)
- Access to execution information at runtime
 - Stack frames
 - Variables



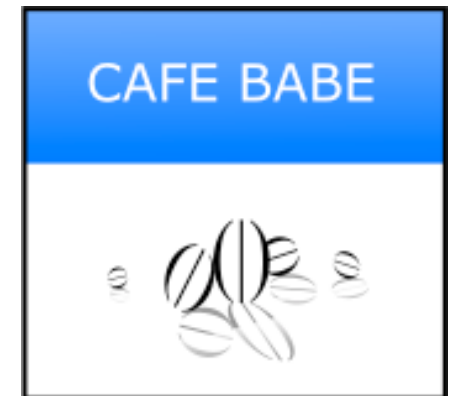
4. Let's change the constants

The more things change ...

- Binary, Date, Time, Date/Time, Date & Time Intervals
 - Basically anything that is an *intrinsic* type
- Arrays, Bags, Sets, Maps, Tuples, Structures
 - Simple recursive model of definition
- Runtime Model
 - Package, class, method, property, variable

3. Alternate ClassFile Format

It's time to stop coloring inside the lines



- 64KB limits? In 2011?
 - A marker could indicate an alternative encoding
 - e.g. All offsets would become 32-bit or variable-length
- Hierarchical
 - Why aren't inner classes, inner classes?
 - Why aren't anonymous classes within the containing method?
 - Why isn't a constant pool shared by all?
- Stack machine vs. Register machine
 - Once we acknowledge that LADD is a virtual invoke on the Long interface, what non-invocation ops are left?
 - Inputs are either a register or a constant; outputs are registers
 - `Invoke MethodConst Object Param* ReturnValue*`

2. Continuations

Still going ...

- “Continuations refer to a functional programming technique that allows you to save the current execution state of a thread, with the possibility of suspending and later resuming execution.” - Artima
- Various implementations
 - Javaflow
 - RIFE
 - Jetty
 - Scala (using exceptions?)
- Could leverage a stronger runtime model (execution frames) combined with Thread or Throwable support



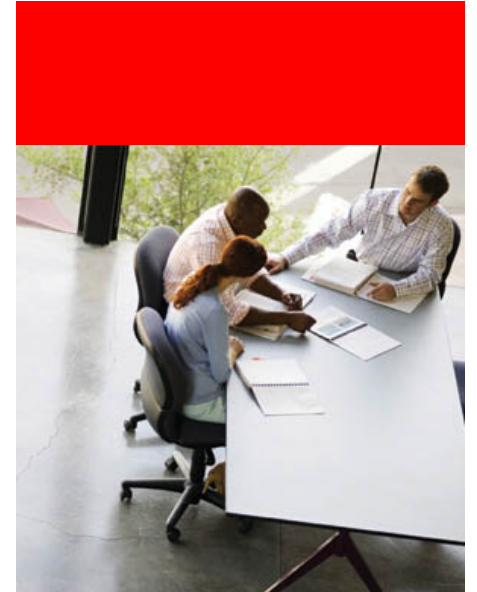
1. Tail Recursion / Tail Call Optimization

... because I promised Guy Steele I'd mention it



- Tail Recursion is the property of a method that has recursion as its final operation before returning; the last thing the method does is call itself.
- Tail Call Optimization is used when the last thing a method does before it returns is call another method: it does a “goto” to the start of the second method, letting it re-use the first routine's stack frame.
- Tail Recursion implemented by Scala
 - As a `while` loop?
- JVM support desired by Fortress & many others
 - Impact on stack traces?

Left-Overs





Separate but equal != equal

It works for primitive types ...

- Immutability introduces a new possibility for “==”
 - Can have two copies of the *same* object (mt-compaction)
 - “Pointer” no longer equals “reference”
 - We need to ask ourselves what “==” actually means – it’s comparing a *value* in a system of *objects*
- Reference equality was an excellent engineering decision ... in 1995
 - Performance-wise: Free
 - But for objects that *are* values, it’s broken, e.g. auto-boxing
- Graphs represent nightmare scenarios
 - All references are of one type; there is no *ownership semantic*



Why is `new` non-virtual?

Non-virtuals: Useless to resist; you will be assimilated

- Who knows how to “new”?
 - Remember: Interface == Class
 - The context that the code is running within provides the correct implementation
 - The context is the name resolver and the “factory”
- Example: Inner classes
 - Base: “new Inner()”
 - Sub: “class Inner extends Base.Inner”
- Could be coupled with dependency injection
 - Context is the classloader, the thread, the `new`-ing method, ...
 - Unification of runtime namespace / implementation resolution and DI helps decouple interface from implementation



Cross-Cutting Concerns

- Traits *implement* some *interface* to explain their capabilities
- Traits *extend* some *interface* to explain their requirements
- Traits can extend *implementation*, e.g. other traits
- Trait binding can be explicit by consumer or provider, or implicit
 - e.g. By a `with` keyword
 - A trait can be incorporated into a class at compile time
 - A trait can be bound automatically by the Execution Context
- Example cross-cutting concern: Serialization



Concurrency Control

What was beautiful in '95 is ugly in '11

- Swap out vtbl for escaped vs. non-escaped object
 - No-cost thread safety when an object hasn't escaped
- `synchronized` must die (*we have >2 CPUs!*)
 - If *actual* deadlock is possible, then we've failed in our design
 - Why don't we have `DeadlockError` today? Backwards compatibility?!?
- Objects that aren't thread-safe shouldn't be allowed to escape!
 - Safety and predictability outweigh the right to stupidity
- Simplified thread safety for mutable objects
 - CAS – fine-grained, single-operation transactions
 - STM – coarse-grained, more expensive transactions



Garbage Collection


Sooner or later, you have to pay the piper

- Unescaped objects are managed locally to a thread
 - Thread local GC can be done with complete concurrency
- Escaped objects can also be collected concurrently
 - Immutable objects enable concurrency by permitting multiple copies to exist (concurrent compact)
 - STM *may* allow for concurrent mark, sweep and compact of mutable objects – *concurrent with execution!*
- Finally an answer for `System.gc()`
 - Collect the thread-local heap
 - Concurrently collect the global heap – using just this thread

Isolation, Resource Management & Multi-Tenancy

Didn't you see the cloud picture on the first slide?

- We've got JSR-284: Isolates
- We need isolates-lite
 - `doNonPrivileged()`
 - A “ring 3” approach
- All resources need to be “managed”
 - Disk – virtualizable with NIO2 (but not runtime configurable!)
 - CPU – we already embed checkpoints for GC
 - RAM – must be able to limit allocation
 - Network – must be able to limit visibility, bandwidth utilization



The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

ORACLE®