Chris Bailey – IBM Java Serviceability and Cloud Integration Architect
11th April 2013

Java: Past Present and Future

History and direction of the Java language and platform

Important Disclaimers

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY.

WHILST EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED.

ALL PERFORMANCE DATA INCLUDED IN THIS PRESENTATION HAVE BEEN GATHERED IN A CONTROLLED ENVIRONMENT. YOUR OWN TEST RESULTS MAY VARY BASED ON HARDWARE, SOFTWARE OR INFRASTRUCTURE DIFFERENCES.

ALL DATA INCLUDED IN THIS PRESENTATION ARE MEANT TO BE USED ONLY AS A GUIDE.

IN ADDITION, THE INFORMATION CONTAINED IN THIS PRESENTATION IS BASED ON IBM’S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM, WITHOUT NOTICE.

IBM AND ITS AFFILIATED COMPANIES SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION.

NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, OR SHALL HAVE THE EFFECT OF:

- CREATING ANY WARRANT OR REPRESENTATION FROM IBM, ITS AFFILIATED COMPANIES OR ITS OR THEIR SUPPLIERS AND/OR LICENSORS

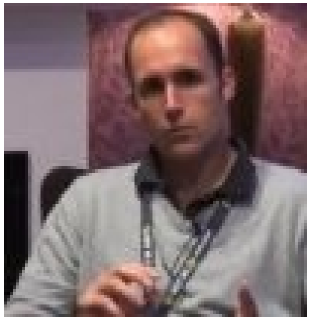
Introduction to the speaker

Chris Bailey

Java Serviceability and Cloud Integration Architect at IBM

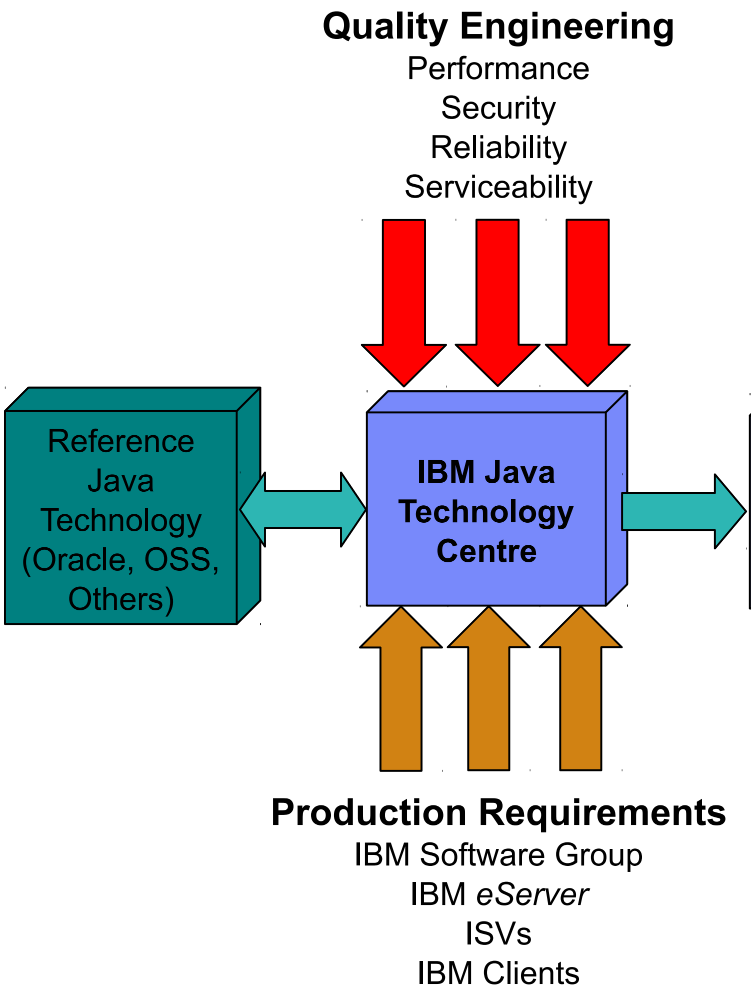
- 13 years experience developing and deploying Java SDKs

- Recent work focus:
  - Java integration into the cloud
  - Java monitoring, diagnostics and troubleshooting
  - Requirements gathering
  - Highly resilient and scalable deployments

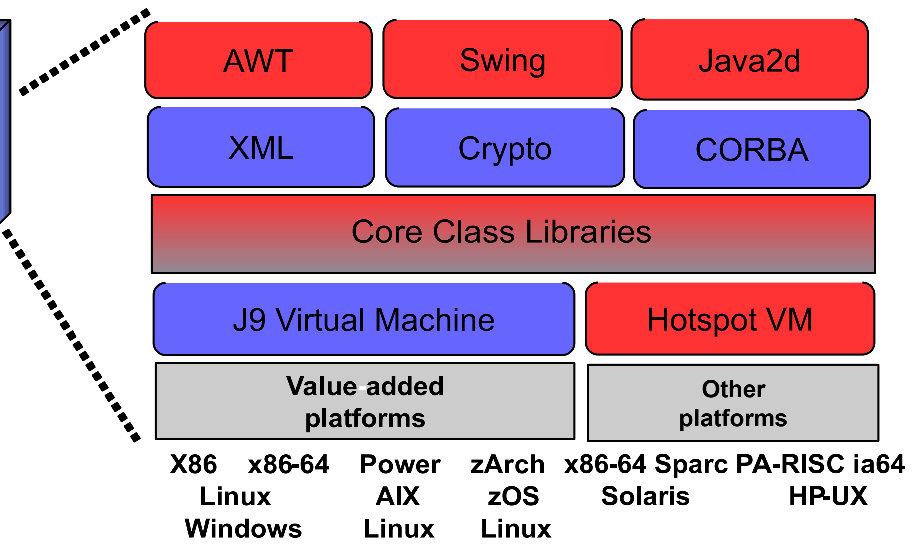


- Contact Details:
  - baileyc@uk.ibm.com
  - http://www.linkedin.com/in/chrisbaileyibm
  - http://www.slideshare.net/cnbailey/

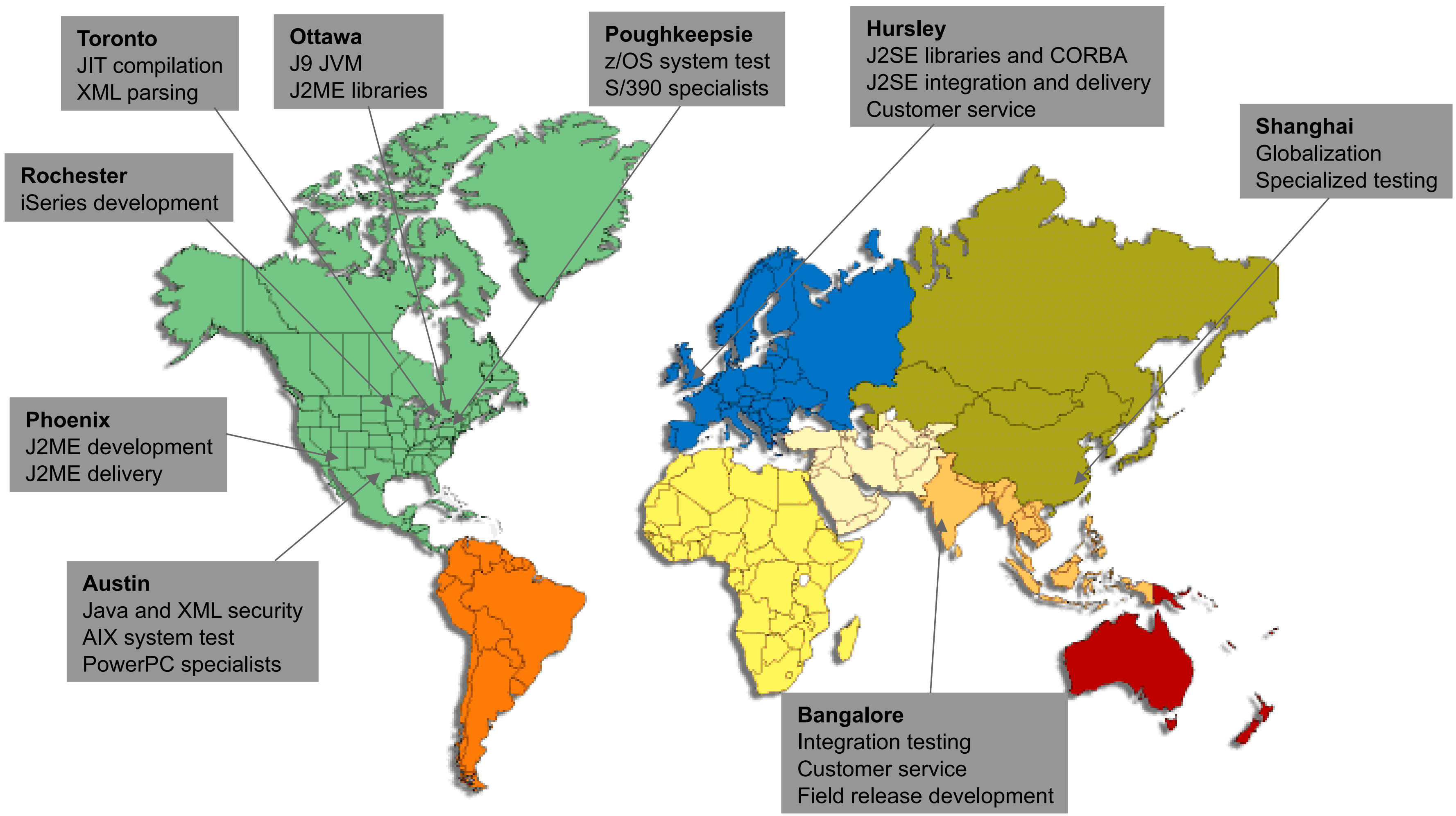
IBM and Java



- ✓ ***Listen to and act upon market requirements***
- ✓ ***World class service and support***
- ✓ ***Available on more platforms than any other vendor***
- ✓ ***Optimised for customers' key platforms***
- ✓ ***Write Once Run Anywhere***
- ✓ ***Embedded in IBM's broad middleware portfolio***



World Wide IBM Java Development Team



Java in the '90s

Java 1.0

- AWT
- WebRunner browser

1.0

Java 1.1

- Inner Classes
- JavaBeans
- JDBC
- RMI
- Reflection

1.1

J2SE 1.2

- Swing
- Browser plug-in
- RMI
- Java Collections

1.2

1991 1992 1993 1994 1995 1996 1997 1998 1999

1

- Sun for Green" lo consume
- James work on

1992

- Green releas PDA
  - Includes O

1993

- Green beco FirstPerson s
  - looking at i TV
- WWW from

1994

- Oak used t WebRunner
  - moving ob browser
-
- Netscape N released

1995

- Oak becomes Java
- Java released on a small website
- WebRunner demoed at TED
- Java adopted by Netscape

1997

- #2 programming language
- 400,000 developers

1999

- Announcement of ME, SE and EE releases

Java in the 2Ks

J2SE 1.3
- RMI-IIOP (CORBA)
- JNDI
- JPDA
- Proxy classes

J2SE 1.4
- Assertions
- Regular Expressions
- Logging API
- New I/O APIs (NIO)
- XML/XSLT
- WebStart

J2SE 5
- New Language features:
  - Autoboxing
  - Enumerated types
  - Generics
  - Meta Data

Java SE6
- Performance improvements
- Improved UI
- Client WebServices Support
- JConsole monitoring
- Collection framework enhancements

1.3 1.4 1.5 1.6

2000 2001 2002 2003 2004 2005 2006 2007 2008

2(
- Java W
announc
•
•Chris Ba

2001
- IBM starts Eclipse project

2003
- Java on 55
desktops

2004
- Java downloa
7 million
- 4 million deve
- 1.5 billion dev
- 550 Java Use
Groups

2005
- 4.5million d
- 2.5 billion d

2006
- Java releas
GPL

2007
- 6 million developers
- 5.5 billion devices

Java in the 2Ks

Java SE7

- Small language changes (coin)
- Improved IO APIs (NIO2)
- Invoke Dynamic
- Concurrency framework

Java SE8

- More small language changes
- Date and Time API
- Annotations on Java Types
- Lambda Expressions
- Compact profiles

Java SE9?

- Java Platform Module System?
- Cloud and Virtualization?
- Struts 2.0 / Packed Objects?

1.7 1.8 1.9

2009 2010 2011 2012 2013 2014 2015 2016 2017

20

- Oracle a acquisitio

2010

- Oracle acquires Sun

2012

- 9 million developers
- 1 billion Java downloads per year

The Present: Java SE7

- Java Specification Request (JSR) 337
  - Java Specification Request that pulls together the set of changes proposed for Java SE
  - Meta-document describing the themes of the release and operating rules
  -
- Main JSRs included:
  - JSR 334: Small Enhancements to the Java Programming Language (OpenJDK Project Coin)
  - JSR 203: More New I/O APIs for the Java Platform ("NIO.2")
  - JSR 292: Supporting Dynamically Typed Languages on the Java Platform
  - As well as JSR 166y: further work to collections and concurrency
  -
- Number of smaller enhancements:
  - Thread-safe concurrent class loaders
  - Unicode 6.0
  - Enhanced locale support (IETF BCP 47 and UTR 35)
  - TLS 1.2
  - Elliptic-curve cryptography
  - JDBC 4.1
  - Translucent and shaped windows
  - Heavyweight/lightweight component mixing
  - Swing Nimbus look-and-feel
  - Swing JLayer component

Project Coin: "A bunch of small change(s)"

- **Strings in switch**

```
switch(myString) {
      case "one": <do something>; break;
      case "red": <do something else>; break;
      default: <do something generic>;
}
```

-

- **Improved Type Inference for Generic Instance Creation (diamond):**

```
Map<String,MyType> foo = new HashMap<String,MyType>();
```

Becomes

```
Map<String,MyType> foo = new HashMap<>();
```

-

- **An omnibus proposal for better integral literals**
  - Allow binary literals: (`0b10011010`)
  - Allow underscores in numbers: (`34_409_066`)
  -

- **Simplified Varargs Methods Warnings**
  - Adds @SafeVarargs annotation to remove warnings on safe varargs method declarations and invocations.
    - Acknowledges subtleties of varargs and generics implementation.
    - Only applies to final methods (can't vouch for overriders!)
  - Reduces unavoidable (and scary!) warnings from many APIs.

Project Coin: “A bunch of small change(s)”

- **Multi-Catch**
  - Developers often want to catch 2 exceptions the same way, but can't in Java 6:

```
try {
    ...
} catch(Exception a) {
    handle(a);
} catch(Error b) {
    handle(b);
}
```

  The following now works:

```
try {
    ...
} catch(Exception|Error a) {
    handle(a);
}
```

- **Automatic Resource Management:**
  - Dealing with all possible failure paths is hard. Closing resources is hard.

```
try(
    InputStream inFile = new FileInputStream(aFileName);
    OutputStream outFile = new FileOutputStream(aFileName)
) {
    byte[] buf = new byte[BUF_SIZE];
    int readBytes;
    while ((readBytes = inFile.read(buf)) >= 0)
        inFile.write(buf, readBytes);
}
```

New I/O 2

- **Asynchronous I/O**
  - Enable significant control over how I/O operations are handled, enabling better scaling.
  - Socket & file classes available.
  - 2 approaches to completion notification:
    - `j.u.c.Future`
    - `CompletionHandler` interface (completed() & failed() calls).
  - Flexible thread pooling strategies, including custom ones.
  -

- **Improved File System API**
  - Address long-standing usability issues & boilerplate
    - User-level modelling of more file system concepts like symlinks
    - File attributes modelled to represent FS-specific attributes (eg: owner, permissions...)
    - DirectoryStream iterates through directories
    - Allows glob, regex or custom filtering.
    - Recursive walks now provided, modelled on Visitor pattern.
  - Model entirely artificial file systems much like Windows® Explorer extensions
  - File Change Notification

Concurrency and Collections Updates

- **Fork-join Framework**
  - Very good at 'divide and conquer' problems
  - Specific model for parallel computation acceleration, significantly more efficient than normal Thread or Executor -base synchronization models.
  - Implements work stealing for lopsided work breakdowns
  -

- **Transfer Queue**
  - A form of `BlockingQueue` but differs in that it provides a recorded delivery service.
  - Thread inserting an object into a `TransferQueue` will return only after the object has been removed from queue by another thread.

-

- **Phaser**
  - Very flexible synchronization barrier that builds on `CyclicBarrier` from Java 5
  - Provides ability to change the number of registered parties dynamically.

The Future: Java SE8

- Java Specification Request (JSR) 337
  - Java Specification Request that pulls together the set of changes proposed for Java SE
  - Meta-document describing the themes of the release and operating rules
- Release drivers
  - Lambda expressions
    - Java language changes to support multicore programming
    - Corresponding changes to collections APIs to exploit parallelisation
  - Virtual extension methods
    - Language constructs designed for library evolution
    - Enhancements to existing interfaces to provide new functionality
- Component JSR specifications and Java Enhancement Proposals (JEPs)
  - A list of ~34 significant items being targetted at Java 8 GA
  - JSRs are developed in conjunction with Java 8, and incorporated
  - Each JEP is at least two weeks platform development work
- Features
  - Any number of smaller pieces of work that don't warrant a JEP

JSR 335 – Lambda expressions

- Currently anonymous inner classes are used for passing context (poorly):

```
final State myState = ...
model.addEventListener(new Listener() {
  public void eventCallback(Event e) {
    if (myState.isActive() && e.isInteresting()) {
       ...
    };
  }
});
```

  - Bulky syntax and confusion surrounding the meaning of names and “this”
  - Inflexible class-loading and instance-creation semantics, often leading to 'class leaking'
  - Inability to capture non-final local variables

- Often used with:
  - java.lang.Runnable
  - java.security.PrivilegedAction
  - java.io.FileFilter
  - java.beans.PropertyChangeListener
  - ...etc

JSR 335 – Lambda expressions

- Lambda expressions in Java 8 have a simple syntax

```
() -> Integer.SIZE;
(int x, int y) -> x + y
(String s, int x) -> { x+=2; System.out.println(s); return x;}
```

- No new level of lexical scoping, so variable names and 'this' are identical to enclosing environment
- Think of them as “anonymous methods”
  - No need for the class definition infrastructure
-
- The Java 8 compiler will allow references to 'effectively final' variables even if they are not marked final
  - compiler data flow determines that the value is not being modified by the lambda expression
-
- Use Lambda expressions to enhance the class libraries
  - Simplify existing APIs that use inner classes
  - Enhance collections APIs to do internal iteration
  - Introduce stream processing of data
  -

Lambdas simplify existing APIs

- Think of them as “anonymous methods”
  – No need for the class definition infrastructure

- The Java 8 compiler will allow references to 'effectively final' variables even if they are not marked final
  – compiler data flow determines that the value is not being modified by the lambda expression
  –

```
final State myState = ...
model.addEventListener(new Listener() {
  public void eventCallback(Event e) {
    if (myState.isActive() && e.isInteresting()) {
       ...
    };
  }
});
```

```
State myState = …
Listener ear = (Event e) -> {
  if (myState.isActive() && e.isInteresting()) {
     ...
  };
};
model.addEventListener(ear);
```

Lambdas enable internal operations

- Lambdas allow the control flow for operations on data to reside near the data
- e.g. internal iteration
  - New methods on collections that accept Lambda expressions as operations on them
  - Allows collections to decide how to iterate over elements
    - Laziness, out-of-order execution, parallelism
  -

```
for (MyType element : myCollection) {
  element.reset();
};
```

```
myCollection.forEach(element - > {element.reset();});
```

Lambdas enable stream operations

- The operations on data structures can now be pipelined into a stream
- Streams can re-order and optimize lambda operations
-
- Ask your collection for a stream, describe operations, gather results.
  - Intermediate operations on streams produce new streams
  - End with a terminal operations
-
- Intermediate operations can be lazy, terminal operations will be eager
  -

```
Stream<MyType> stream = myCollection.stream()
     .filter(element -> element.length() == 0)
     .forEach(element -> { element.reset(); });

Set<MyType> emptyTypes = stream.into(new HashSet());
```

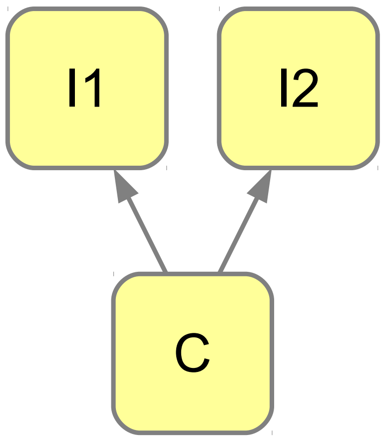
Virtual extension methods

- Lambda and stream operations are useful on existing collection types
- Need some way to extend well established data structures
  - ie creating parallel hierarchy of similar structures
  - Retaining compatibility
- Adding a new method to an existing interface is binary compatible
  - But disenfranchises implementers
- Enhance language to provide default implementations in interfaces
  - Interface declarations contain code, or references to code, to run if classes do not provide an implementation
  -

```
public interface Set<T> extends Collection<T> {
     public boolean add(E e);
     public void clear();
     ...
     public void forEach(Block<T> blk)
          default Collections.<T>setForEach;
}
```

Virtual extension methods

- Types can implement multiple interfaces
- Interfaces can implement methods
- Most specific method implementation is chosen
  - If the class implements it, then use that
  - Otherwise chooses the most specific interface with a default implementation
  - If there is a conflict, then you get a compile/link time error




```
public interface I1 {
     default void run() { ... }
}
public interface I2 {
     default void run() { ... }
}

public class C implements I1, I2 {
     public static void main(String[] args) {
          new C().run();
}
```

Error!

JSR 308: Annotations on Java Types

- Extending the scope of annotations as introduced in Java 6
  – Annotate type usage, not just type declaration
  – Carried in class files for robust development time checking
- Allows for pluggable extensions to Java language type checking
  – Strengthen and refine the built-in type system
  – Type annotations can be written before any type, e.g. @NonNull String
- Software quality and security
  – Null pointer errors, side effects on immutable data, race conditions, information leakage, non-internationalized strings, etc.
- Checkers framework use additional information
  – Non-prescriptive use of annotations allows for varied tooling
  – Expect to see variety of coding tools use annotations for developer feedback

```
List<@NonNull String> strings;

myGraph = (@Immutable Graph) tmpGraph;

class UnmodifiableList<T> implements @Readonly List<@Readonly T> { ... }

@Tainted String entry;
```

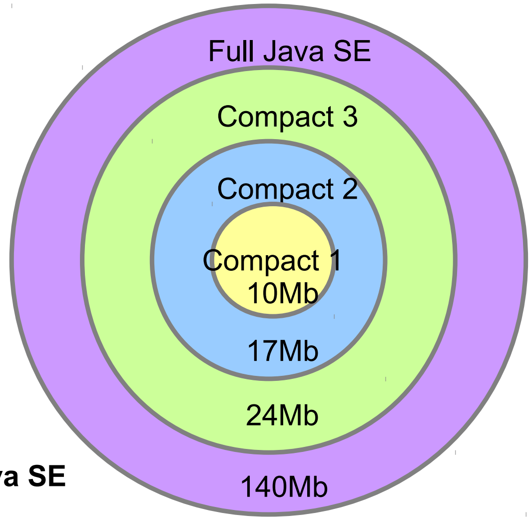
JSR 310: Date and Time API

- A new, modern date and time API for Java
- Current date and time types are split across multiple packages
  - java.util, java.sql, java.text, etc.
- API could be improved in a number of ways...
  - java.util.Date is actually a timestamp
  - Based on years from 1900 onwards
  - Calendar instances cannot be converted to simple date formatted strings
  - Etc
- JSR-310 is a top to bottom review of the date and time handling in Java
  - Use relevant standards, including ISO-8601, CLDR, and BCP47
  - Types represent point in time, duration, and localization

```
java.time
    main API for dates, times, instants, and durations
java.time.calendar
    Support for Hijrah, Japanese, Minguo, Thai Buddest calendar systems
java.time.format
    Provides classes to print and parse dates and times
java.time.temporal
    Expands on the base package for more powerful use cases
java.time.zone
    Support for time-zones and their rules
```

Profiles and Stripped Implementations

- Profiles are a subset of the full Java SE APIs
  - Applications can specify the minimal profile they require.
  - Development tools (e.g. javac) are enhanced to understand profile boundaries
  - Runtime tools (e.g. java) are enhanced to understand profile requirements
- Stripped Implementations remove unused elements of the runtime



Compact 1

java.lang
java.io
java.nio
java.text
java.math
java.net
javax.net
java.util
java.util.logging
java.security
javax.crypto
javax.security

+

Compact 2

java.sql
javax.sql
javax.xml
org.w3c.dom
org.xml.sax
java.rmi
javax.rmi
javax.transaction

+

Compact 3

java.lang.management
javax.management
javax.naming
javax.sql.roweset
javax.security.auth.kerberos
org.ietf.jgss
javax.script
javax.xml.crypto
java.util.prefs
javax.security.sasl
javax.security.acl
java.lang.instrument
javax.annotation.processing
javax.lang.model
javax.lang.model.element
javax.lang.model.type
javax.lang.model.util
javax.tools

+

Full Java SE

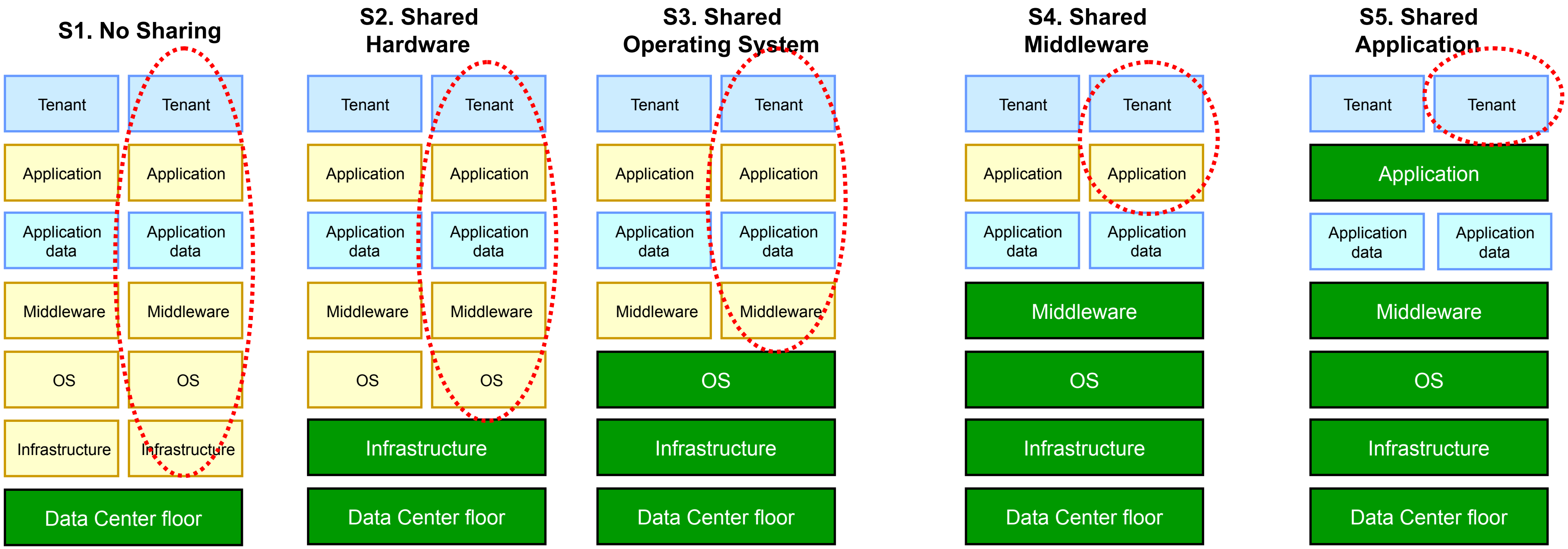
java.applet
java.awt
java.beans
javax.activity
javax.rmi.CORBA
org.omg
javax.accessibility
javax.imagio
javax.print
javax.sound
javax.swing
javax.activation
javax.jws
javax.xml.bind
javax.xml.soap
javax.xml.ws
javax.annotation

And Beyond...

- Modularity and reduced footprints
- Data access and platform integration
- Cloud and Multi-Tenancy
- Other Languages on the JVM

Cloud and Multi-Tenancy

The amount of work required for setting up a new tenant depends on where the "multi-tenancy point" sits within the technology stack. The higher the multi-tenancy point, the less effort is required for setting up a new tenant



Run in the same data center floor space with no sharing

Sharing servers storage, networks in a data center

Hypervisors (e.g. KVM, VMWare) are used to virtualize the hardware

Multiple applications sharing the same middleware

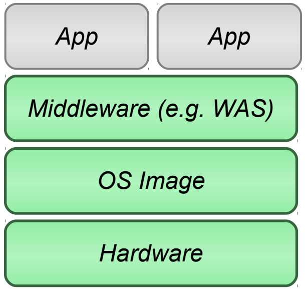
Sharing the same application

Lower Density
Larger Footprint
Slower Startup

Higher Density
Smaller Footprint
Faster Startup

Shared Middleware

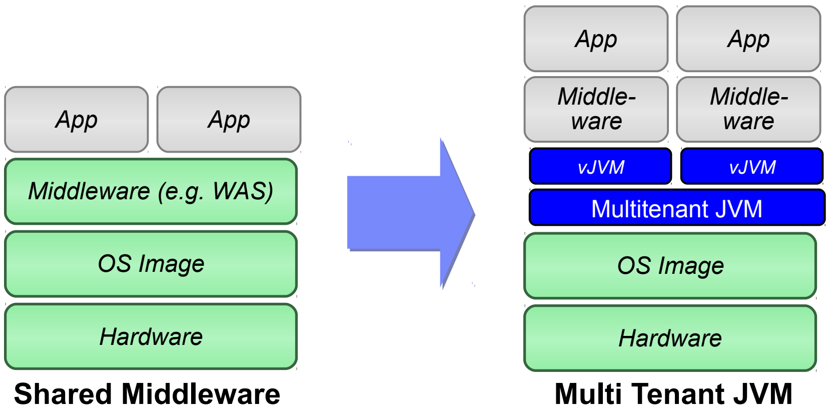
- Greater Density: S4. Shared Middleware
  - Reduces memory footprint
  - Reduces startup time
- Middleware has to be enabled for sharing
  - Avoid resource collisions:
  - URLs, cache contents, static variables, etc
- Shared middleware introduces sharing problems:
  - Resource collisions: URLs, cache contents, static variables, etc
  - Resource sharing: CPU, Memory, I/O
  - Reduced security



S4. Shared Middleware

Shared Middleware

- Multi-Tenant JDK:
  – JVM-level virtualization vs. middleware virtulization
  – JVM allows for multiple isolated middleware/application instances
-
- Solved isolation problems:
  – Middleware stack appears dedicated
- JVM, Class and Compiled Code sharing
- Resource control provided by JVM
  – CPU, Memory, Network, I/O
  –

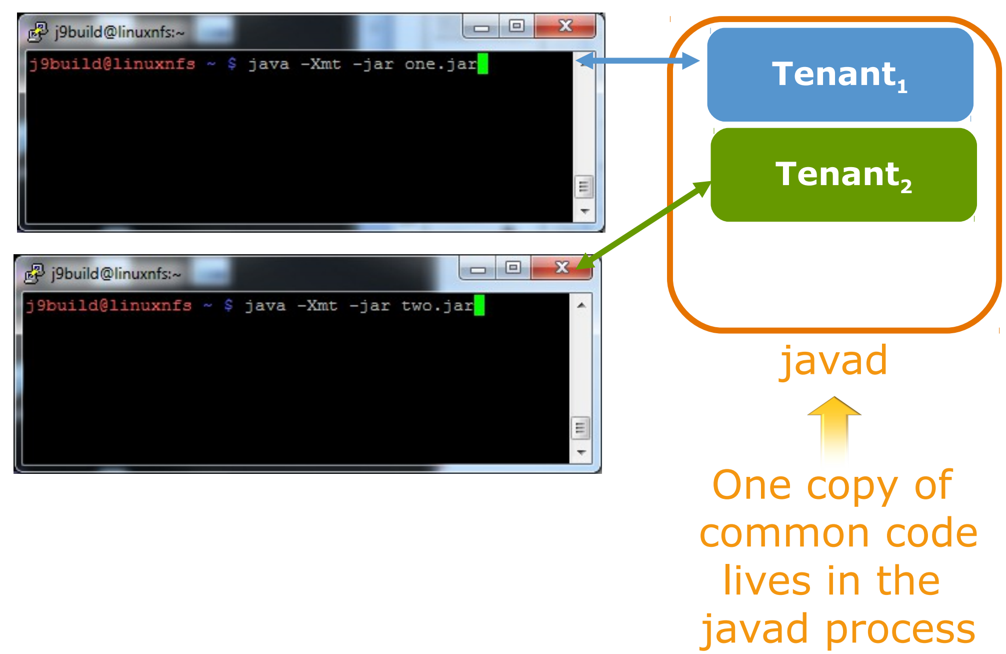


Operating System	Mechanism	File System Isolation	Disk Quotas	I/O Rate Limiting	Memory Limits	CPU Quotas	Network Isolation	Live Migration
AIX	LPAR	Y	Y	Y	Y	Y	Y	Y
	WPAR	Y	Y	Y	Y	Y	Y	Y
	MT-JDK	NY	NY	Y	Y	Y	NY	N
Linux	VMware ESX	Y	Y	Y	Y	Y	Y	Y
	Open VZ	Y	Y	Y	Y	Y	Y	Y
	MT-JDK	NY	NY	Y	Y	Y	NY	N

Generic middleware sharing via the JVM

- Multi-Tenant JDK for High Density Java:
  - Multiple applications run in one daemon JVM
  - Applications appear as if they are running in a dedicated JVM
- Provides Isolation and Resource Controls:
  - CPU, Heap Memory, Threads, Files, Network

- Running the Multi-Tenant JDK:
  - Opt-in by adding `-Xmt`
  - JVM will locate/start daemon
  - Tenant created inside `javad` daemon
- Adding a second tenant:
  - Opt-in by adding `-Xmt`
  - JVM will locate/start daemon
  - Tenant created inside `javad` daemon



- One copy of common code
- Most runtime structures shared

Other Languages on the JDK

- In the good old days…. Enterprise apps written all in Java
  – All layers (Presentation logic, business logic, persistence/O-R tier, XML processing, everything)
- Today, applications are written in many languages
  – Java, JavaScript, Scala, Ruby, Python…
- With many frameworks
  – JEE, Node.js, Rails, Django, etc

JVM Support for Other Languages

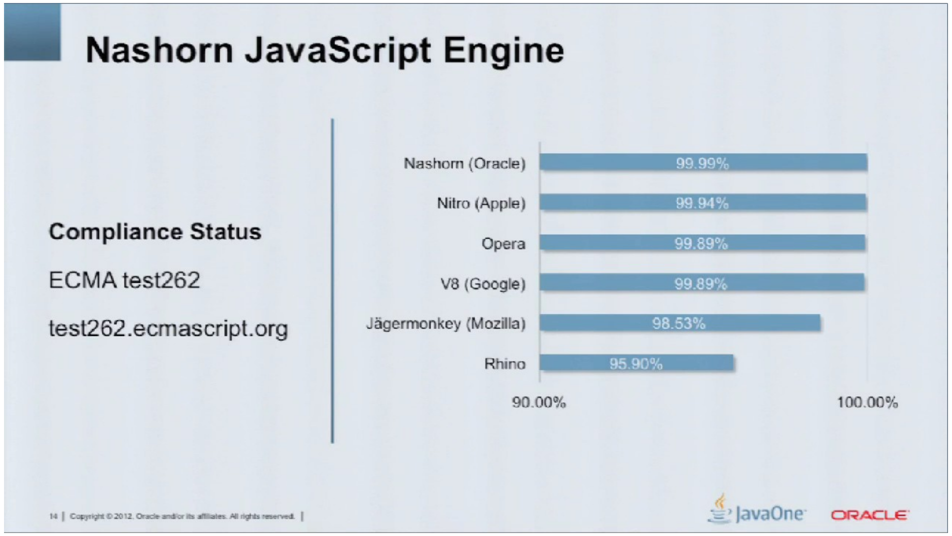
- Java has been reacting to this change for a while
  - Evolving efficient multi-language support in the JVM
    - Java 7 and JSR 292
  - Interop APIs for many languages
    - JSR 353 (JSON)
  - Support for many existing languages
    - Jython, JRuby, PHP-J
  - A rich ecosystem of great new languages
    - Scale, X10, Clojure
-
- Java will be PART of solutions going forward but not the entire solution in many cases
-
- Developers should learn about new languages and where they are appropriate to use

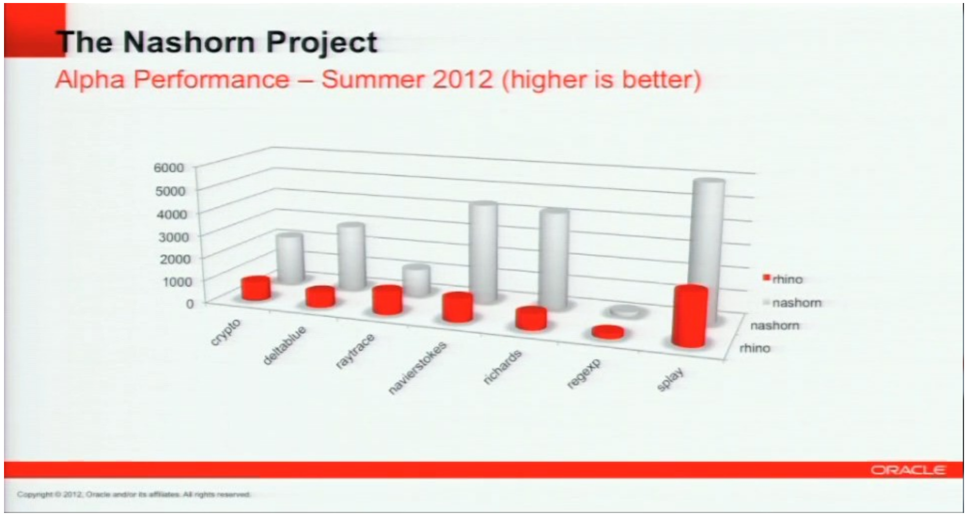
JSR 292: invokeDynamic

- Most new languages we might want to run on the JVM are dynamically typed
- Java is a statically typed language
  –
- However: at the bytecode level Java is more loosely typed
  – Variables on the operand stack are typed only in terms of primitive type or object reference.
  – Method invocation has strong typing enforcement
    • Methods are invoked with full signature including parameters and return types
-
- JSR 292 decouples method lookup and method dispatch
  – Get away from being purely Java (the language) centric.
- Approach: Introduce a new bytecode that executes a given method directly, and provides the ability at runtime to rewire what method that is.
  – Include a model for building up mutators (add a parameter, drop a parameter, etc..)
  – Ensure the JIT can efficiently exploit these constructs to ensure efficient code generation.

Project Nashorn

- JavaScript engine on the JDK released to OpenJDK in November 2012
  - JavaScript applications running on the JVM
  - Allows the embedding of JavaScript into Java applications
- Greater JavaScipt compliance and performance that previous “Rhino” engine:





Slides from Oracle presentations at JavaOne 2012

- Ongoing project to port Node.js onto Nashorn: Node.jar

Copyright and Trademarks