

3.– 6. September 2012
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Java SE 8 & Beyond

Dalibor Topic

ORACLE Deutschland B.V. & Co. KG



MOVING JAVA FORWARD

ORACLE™

Java SE 8 and Beyond

Dalibor Topić (@robilad)

Principal Product Manager, Java Platform Group

September 4th, 2012 - Herbstcampus

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.

Priorities for the Java Platforms



Grow Developer Base



Grow Adoption



Increase Competitiveness



Adapt to change



Evolving the Language

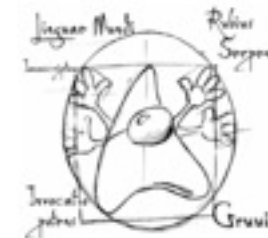
From “Evolving the Java Language” - JavaOne 2005

- Java language principles
 - Reading is more important than writing
 - Code should be a joy to read
 - The language should not hide what is happening
 - Code should do what it seems to do
 - Simplicity matters
 - Every “good” feature adds more “bad” weight
 - Sometimes it is best to leave things out
- One language: with the same meaning everywhere
 - No dialects
- We will evolve the Java language
 - But cautiously, with a long term view
 - “first do no harm”

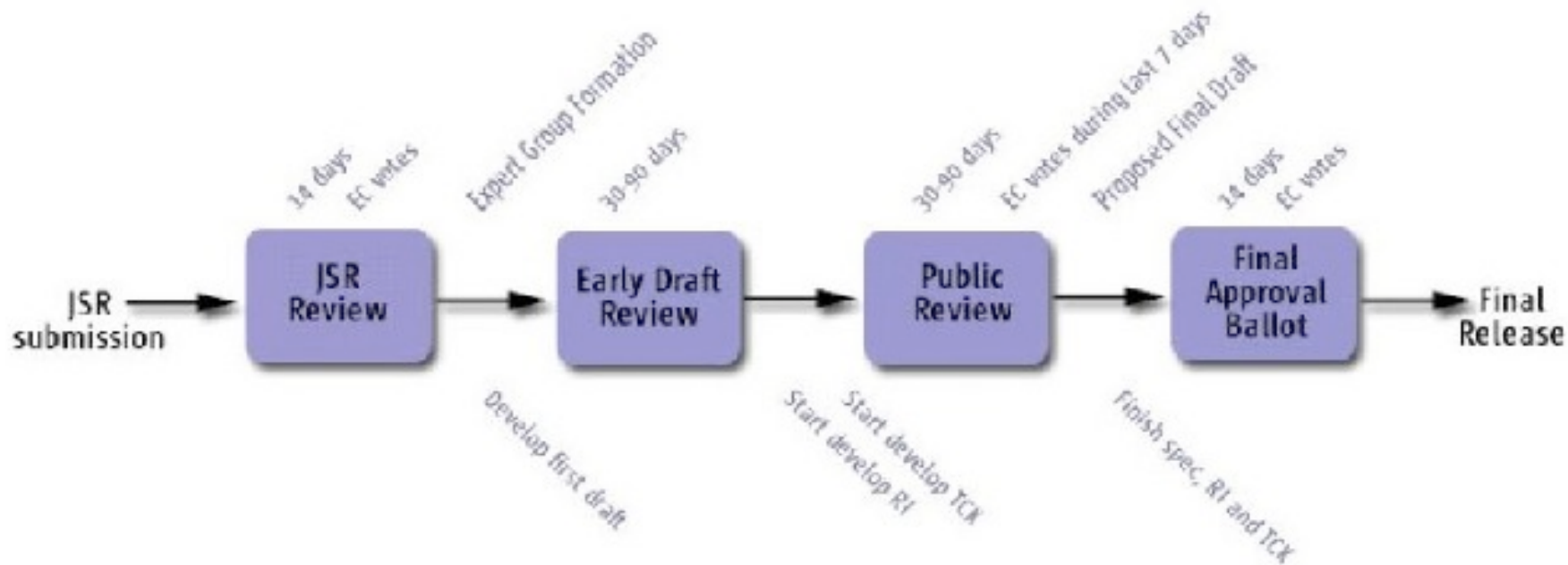
*also “Growing a Language” - Guy Steele 1999
“The Feel of Java” - James Gosling 1997*

Java SE 7 Release Contents

- Java Language
 - Project Coin (JSR-334)
- Class Libraries
 - NIO2 (JSR-203)
 - Fork-Join framework, ParallelArray (JSR-166y)
- Java Virtual Machine
 - The DaVinci Machine project (JSR-292)
 - InvokeDynamic bytecode
- Miscellaneous things
- JSR-336: Java SE 7 Release Contents



How Java Evolves and Adapts



JSR-348: JCP.next

Java SE 8 Platform Umbrella JSR (337)

This is the primary web page for [JSR 337](#), the Platform Umbrella JSR for Java SE 8.

Expert Group

- Kevin Bourrillion (Google)
- Andrew Haley (Red Hat)
- Steve Poole (IBM)
- Mark Reinhold (Oracle)

Schedule

- 2012/7 Expert Group formation
- 2012/9 Early Draft Review
- 2013/1 Public Review
- 2013/6 Proposed Final Draft
- 2013/8 Final Release

Mailing lists

There are three mailing lists:

- [java-se-8-spec-experts](#) is the Expert Group (EG) list. Only EG members may subscribe and post to this list, but the [archives](#) are public.
- [java-se-8-spec-observers](#) is for those who wish to monitor and, perhaps, discuss the EG's progress. Messages sent to the primary EG list are automatically forwarded to this list. Anyone may subscribe to this list, and any subscriber may post. EG members are under no obligation to follow the traffic on this list.
- [java-se-8-spec-comments](#) is for sending comments, suggestions, and other feedback directly to the EG. Only EG members may subscribe to this list, but anyone may post, and the [archives](#) are public. The EG will read all messages sent to this list.

Source code

- Mercurial (6, 7, 7u, 8)
- Bundles (6, 7, 7u)

Groups

- (overview)
- 2D Graphics
- AWT
- Build
- Compiler
- Conformance
- Core Libraries
- Governing Board
- HotSpot
- Internationalization
- JMX
- Members
- Networking
- NetBeans Projects
- Porters
- Quality
- Security
- Serviceability

JDK 8

[« home](#) · [features](#) · [milestones](#) · [builds](#) »

The goal of this Project is to produce an open-source reference implementation of the Java SE 8 Platform, to be defined by JSR 337 in the Java Community Process.

Content

JDK 8 is the second part of [Plan B](#). The proposed release-driver features are the [Lambda](#) and [Jigsaw](#) Projects (though note that a proposal has been made to [defer Jigsaw to the next release](#)). Additional features proposed via the [JEP Process](#) will be included, but they must fit into the overall schedule required for the release drivers. Detailed information on the features funded and targeted to the release, so far, can be found on the [features](#) page.

Schedule

The overall development schedule is divided into a sequence of [milestone](#) cycles, with each feature targeted to a specific milestone.

2012/04/26	M1	
2012/06/14	M2	
2012/08/02	M3	
2012/09/13	M4	
2012/11/29	M5	
2013/01/31	M6	Feature Complete
2013/02/21	M7	Developer Preview
2013/07/05	M8	Final Release Candidate
2013/09/09	GA	General Availability

Further information on milestone content and the final phases of the release can be found on the [milestones](#) page. The [builds](#) and [integrations](#) page contains week-by-week scheduling information.

Early-access binaries

EA binaries built from the JDK 8 code base are available today from [Oracle](#).

Links to EA binaries from other vendors will be listed here when they become available.

Comments and questions to: jdk8-dash-dev@openjdk.java.net

JDK 8 Milestones

[« home](#) · [features](#) · [milestones](#) · [builds](#) »

The JDK 8 development schedule is divided into a sequence of milestone cycles, most six to eight weeks in length, with builds occurring roughly once each week. There will be no formal beta or early-access releases. Major features and other potentially-destabilizing changes will be targeted for integration early in a specific milestone.

Here is the milestone schedule, with the features targeted to each cycle:

- M1** 2012/04/26 (b36)
 - 117 Remove the Annotation-Processing Tool (apt)
- M2** 2012/06/14 (b43)
 - 133 Unicode 6.1
- M3** 2012/08/02 (b50)
 - 124 Enhance the Certificate Revocation-Checking API
 - 130 SHA-224 Message Digests
 - 131 PKCS#11 Crypto Provider for 64-bit Windows
- M4** 2012/09/13
 - 105 DocTree API
 - 121 Stronger Algorithms for Password-Based Encryption
 - 129 NSA Suite B Cryptographic Algorithms
- M5** 2012/11/29
 - 106 Add Javadoc to javax.tools
 - 110 New HTTP Client
 - 111 Additional Unicode Constructs for Regular Expressions
 - 112 Charset Implementation Improvements
 - 113 MS-SFU Kerberos 5 Extensions
 - 114 TLS Server Name Indication (SNI) Extension
 - 119 javax.lang.model Implementation Backed by Core Reflection
 - 122 Remove the Permanent Generation
 - 128 BCP 47 Locale Matching
 - 140 Limited doPrivileged
 - 153 Launch JavaFX Applications

M6	2013/01/31	<i>Feature Complete</i>
	101	Generalized Target-Type Inference
	104	Annotations on Java Types
	107	Bulk Data Operations for Collections
	108	Collections Enhancements from Third-Party Libraries
	109	Enhance Core Libraries with Lambda
	115	AEAD CipherSuites
	118	Access to Parameter Names at Runtime
	120	Repeating Annotations
	123	Configurable Secure Random-Number Generation
	126	Lambda Expressions and Virtual Extension Methods
	135	Base64 Encoding and Decoding
	156	G1 GC: Reduce need for full GCs
	160	Lambda-Form Representation for Method Handles
M7	2013/02/21	<i>Developer Preview</i>
	2013/03/18	<i>All Tests Run</i>
	2013/04/04	<i>Rampdown start</i>
	2013/05/02	<i>API/Interface Freeze</i>
	2013/05/16	<i>Zero Bug Bounce</i>
	2013/06/13	<i>Rampdown phase 2</i>
M8	2013/07/05	<i>Final Release Candidate</i>
GA	2013/09/09	<i>General Availability</i>

JDK 8

[Downloads](#)

[Feedback Forum](#)

[OpenJDK](#)

[Planet JDK](#)

JDK 8 Project

Building the next generation of the Java SE platform

Download JDK 8

- [JDK 8 snapshot release](#)
- [Source code](#) (instructions)

For details about JDK 8, please see the [JDK 8](#) and [Lambda](#) project pages.

JDK 8 Early Access Now Available!

Try it out today!

JVM Convergence



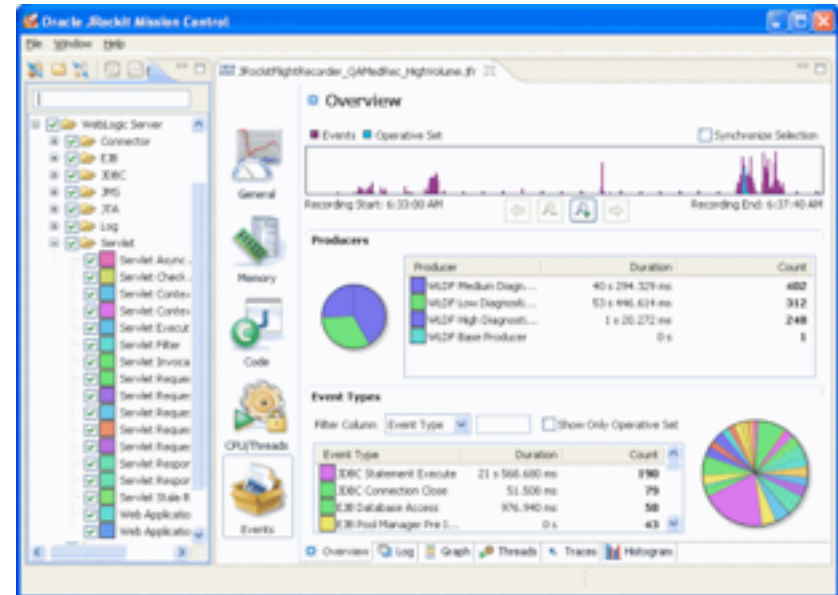
Oracle JRockit

The Definitive Guide

Develop and manage robust Java applications with Oracle's high-performance Java Virtual Machine

Foreword by Adam Westcott
Vice President of Development in the Oracle Fusion Middleware group

Marcus Hirt Marcus Lagergren **PACKET** enterprise™



JEP 122: Remove the Permanent Generation

<i>Author</i>	Jon Masamitsu
<i>Organization</i>	Oracle
<i>Created</i>	2010/8/15
<i>Updated</i>	2012/8/20
<i>Type</i>	Feature
<i>State</i>	Funded
<i>Component</i>	vm/gc
<i>Scope</i>	Impl
<i>RFE</i>	6964458
<i>Internal-refs</i>	Oracle:A360:682265
<i>Discussion</i>	hotspot dash dev at openjdk dot java dot net
<i>Start</i>	2010/Q3
<i>Effort</i>	XL
<i>Duration</i>	XL
i) <i>Reviewed-by</i>	Paul Hohensee
<i>Endorsed-by</i>	Paul Hohensee
<i>Funded-by</i>	Oracle
<i>Release</i>	8

Summary

Remove the permanent generation from the Hotspot JVM and thus the need to tune the size of the permanent generation.

Non-Goals

Extending Class Data Sharing to application classes. Reducing the memory needed for class metadata. Enabling asynchronous collection of class metadata.

Success Metrics

Class metadata, interned Strings and class static variables will be moved from the permanent generation to either the Java heap or native memory.

The code for the permanent generation in the Hotspot JVM will be removed.

Application startup and footprint will not regress more than 1% as measured by a yet-to-be-chosen set of benchmarks.

Motivation

This is part of the JRockit and Hotspot convergence effort. JRockit customers do not need to configure the permanent generation (since JRockit does not have a permanent generation) and are accustomed to not configuring the permanent generation.

[hsx/hotspot-gc/hotspot](#) / changeset

[summary](#) | [shortlog](#) | [changelog](#) | [tags](#) | [manifest](#) | [changeset](#) | [raw](#) | [bz2](#) | [zip](#) | [gz](#)

6964458: Reimplement class meta-data storage to use native memory [default](#) [tip](#)

author coleemp
 Sat Sep 01 13:25:18 2012 -0400 (45 hours ago)
changeset 3599 da91efe96a93
parent 3598 36d1d483d5d6

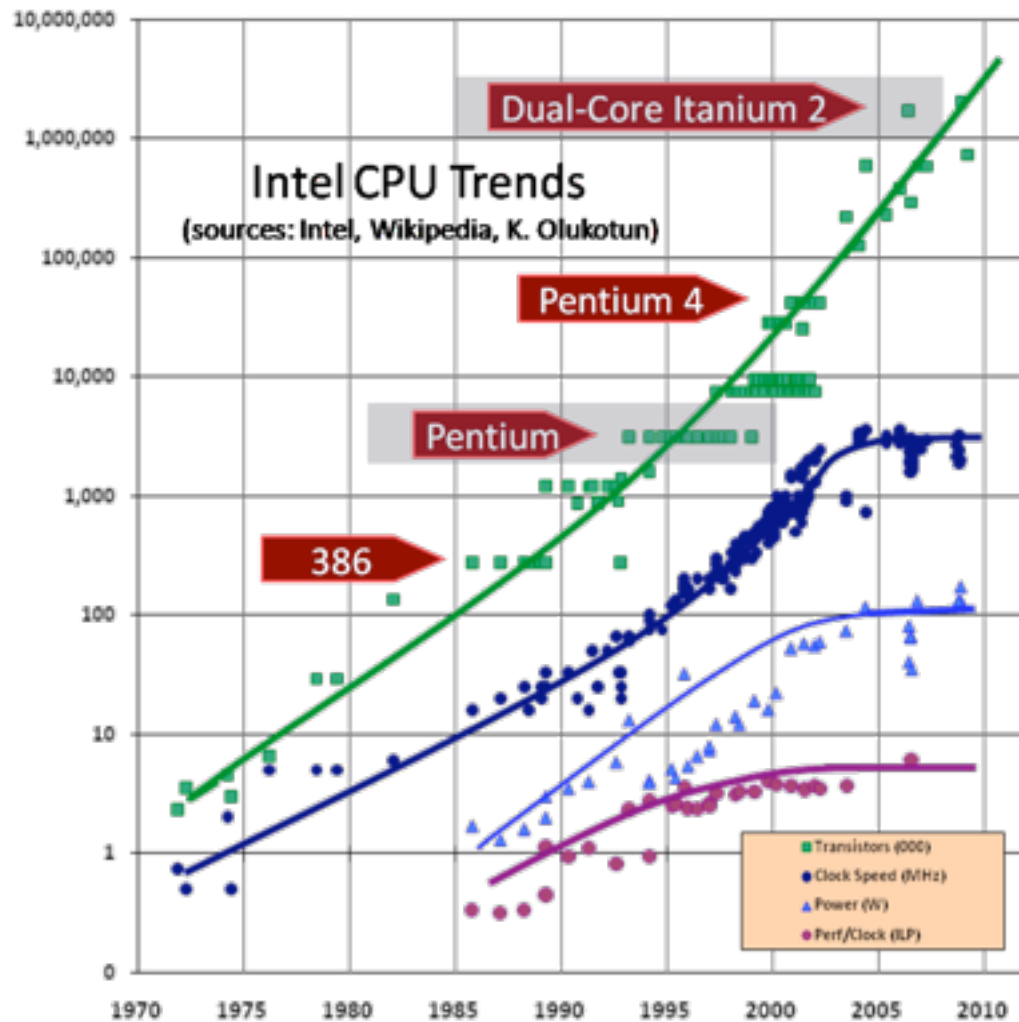
6964458: Reimplement class meta-data storage to use native memory

Summary: Remove PermGen, allocate meta-data in metaspace linked to class loaders, rewrite GC walking, rewrite and rename metadata to be C++ classes

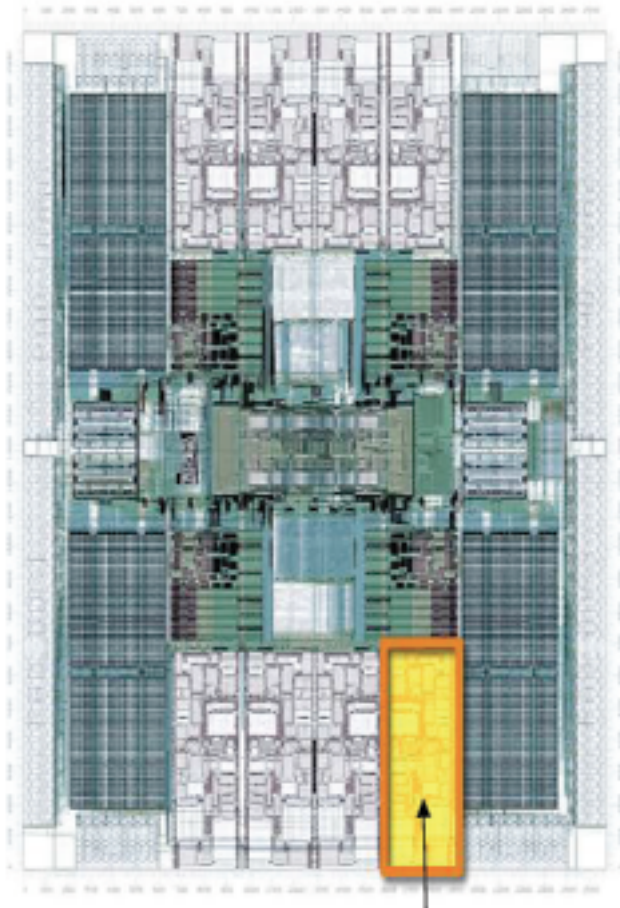
Reviewed-by: jmasa, stefank, never, coleemp, kvn, brutisso, mgerdin, dholmes, jrose, twisti, roland

Contributed-by: jmasa <jon.masamitsu@oracle.com>, stefank <stefan.karfsson@oracle.com>, mgerdin <mikael.gerdin@oracle.com>, never <tom.rodriguez@oracle.com>

The (Performance) Free Lunch Is Over



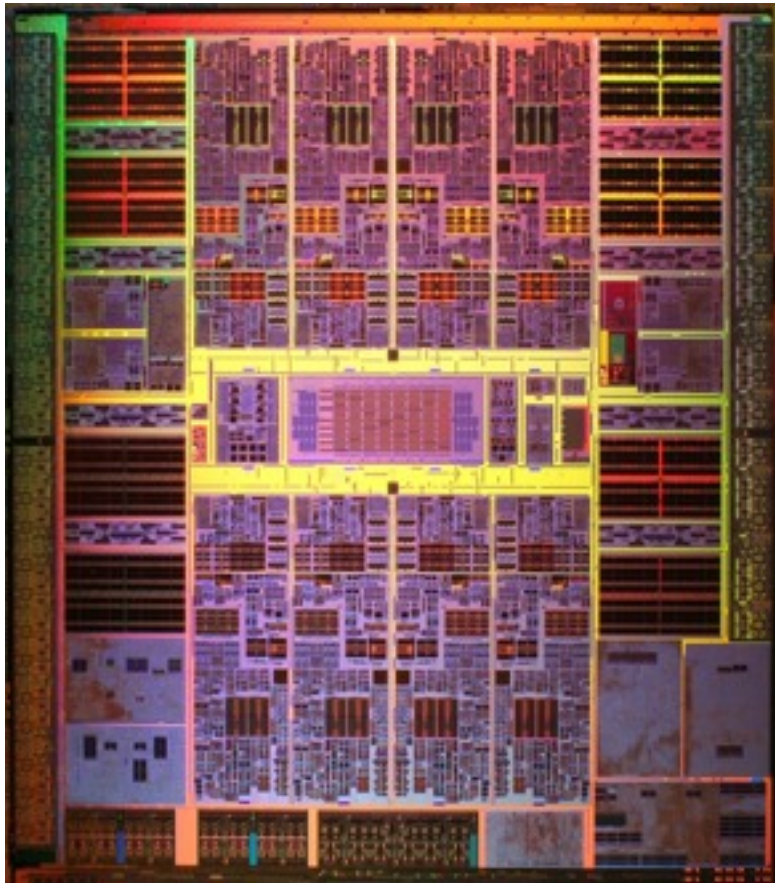




UltraSPARC-Core

SPARC T1 (2005)

$$8 \times 4 = 32$$

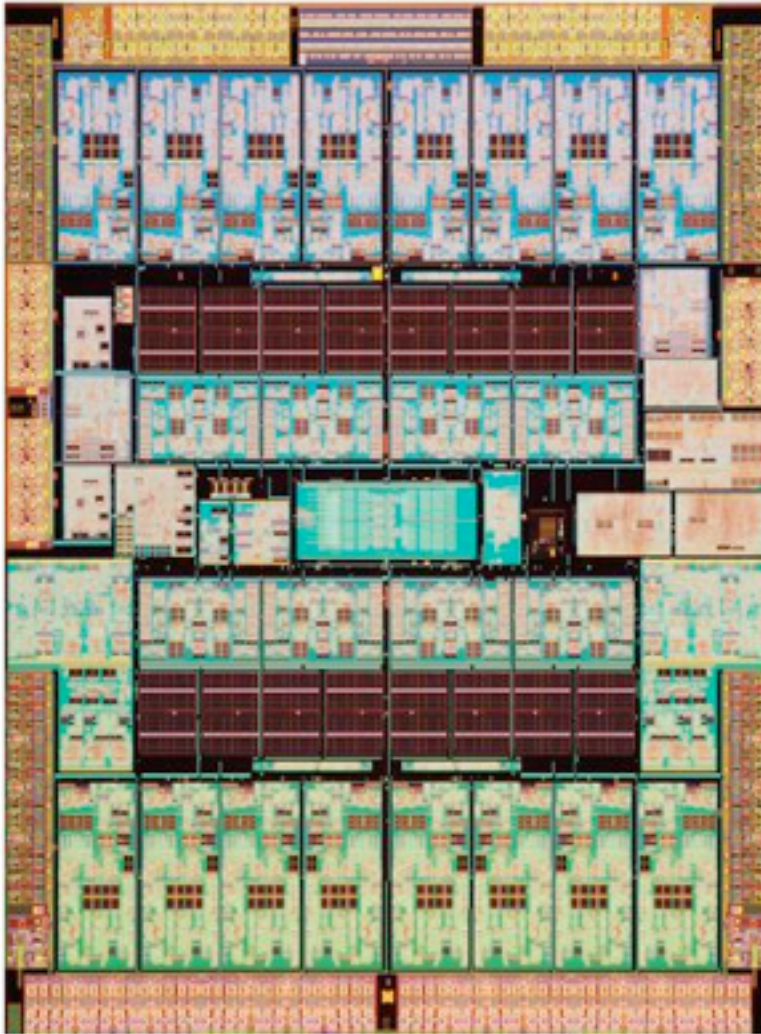


SPARC T1 (2005)

$$8 \times 4 = 32$$

SPARC T2 (2007)

$$8 \times 8 = 64$$



SPARC T1 (2005)

$$8 \times 4 = 32$$

SPARC T2 (2007)

$$8 \times 8 = 64$$

SPARC T3 (2011)

$$16 \times 8 = 128$$

Big Disclaimer

The syntax used in the following slides may change

Caveat emptor

```
class Student {  
    String name;  
    int gradYear;  
    double score;  
}
```

```
Collection<Student> students = ...;
```

```
Collection<Student> students = ...;

double max = Double.MIN_VALUE;

for (Student s : students) {
    if (s.gradYear == 2011)
        max = Math.max(max, s.score);
}
```

```
Collection<Student> students = ...;

double max = Double.MIN_VALUE;

for (Student s : students) {
    if (s.gradYear == 2011)
        max = Math.max(max, s.score);
}
```



```
Collection<Student> students = ...;
```

```
max = students.filter(new Predicate<Student>() {  
    public boolean op(Student s) {  
        return s.gradYear == 2011;  
    }  
}).map(new Extractor<Student, Double>() {  
    public Double extract(Student s) {  
        return s.score;  
    }  
}).reduce(0.0, new Reducer<Double, Double>() {  
    public Double reduce(Double max, Double score) {  
        return Math.max(max, score);  
    }  
});
```

Inner Classes Are Imperfect Closures

- Bulky syntax
- Unable to capture non-final local variables
- Transparency issues
 - Meaning of return, break, continue, this
- No non-local control flow operators

Single Abstract Method (SAM) Types

- Lots of examples in the Java APIs
 - Runnable, Callable, EventHandler, Comparator

```
foo.doSomething(new CallbackHandler() {  
    public void callback(Context c) {  
        System.out.println(c.v());  
    }  
});
```

- Noise:Work ratio is 5:1
- Lambda expressions grow out of the idea of making callback objects easier



```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)  
              .map((Student s) -> s.score)
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                       (Double max, Double score) ->
                         Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```



```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
               .filter(s -> s.gradYear == 2011)
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
               .filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
               .filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
               .filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
Collection<Student> students = ...;
```

```
max = students.filter((Student s) -> s.gradYear == 2011)
               .map((Student s) -> s.score)
               .reduce(0.0,
                      (Double max, Double score) ->
                        Math.max(max, score));
```

```
max = students.filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
max = students.parallel()
               .filter(s -> s.gradYear == 2011)
               .map(s -> s.score)
               .reduce(0.0, Math::max);
```

```
Collection<Student> students = ...;
```

```
double max = // Lambda expressions  
    students.filter(Students s -> s.gradYear == 2010})  
        .map(Students s -> s.score )  
        .reduce(0.0, Math::max);
```

```
Collection<Student> students = ...;
```

```
double max = // Lambda expressions  
    students.filter(Students s -> s.gradYear == 2010})  
        .map(Students s -> s.score )  
        .reduce(0.0, Math::max);
```

```
interface Collection<T> {  
    int add(T t);  
    int size();  
    void clear();  
    ...  
}
```



```
Collection<Student> students = ...;
```

```
double max = // Lambda expressions  
    students.filter(Students s -> s.gradYear == 2010})  
        .map(Students s -> s.score )  
        .reduce(0.0, Math::max);
```

```
interface Collection<T> {  
    int add(T t);  
    int size();  
    void clear();  
    ...  
}
```

How to extend an interface in Java SE 8

tells us this
method
extends the
interface

```
public interface Set<T> extends Collection<T>
{
    public int size();

    ...    // The rest of the existing Set methods

    public extension T reduce(Reducer<T> r)
        default Collections.<T>setReducer;
}
```

**Implementation to use if none
exists for the implementing class**

How to extend an interface in Java SE 8

tells us this
method
extends the
interface

```
public interface Set<T> extends Collection<T>
{
    public int size();
    ... // The rest of the existing Set methods

    public extension T reduce(Reducer<T> r)
        default Collections.<T>setReducer;
}
```

Implementation to use if none
exists for the implementing class

```
Collection<Student> students = ...;
```

```
double max =           // Lambda expressions  
    students.filter(Students s -> s.gradYear == 2010)  
        .map(Students s -> s.score )  
        .reduce(0.0, Math#max);
```

```
interface Collection<T> { // Default methods  
    extension Collection<E> filter(Predicate<T> p)  
        default Collections.<T>filter;  
  
    extension <V> Collection<V> map(Extractor<T,V> e)  
        default Collections.<T>map;  
  
    extension <V> V reduce()  
        default Collections.<V>reduce;  
}
```

JDK 8

- Downloads
- Feedback Forum
- OpenJDK
- Planet JDK

Java™ Platform, Standard Edition 8 Early Access with Lambda Support

This page provides an Early Access of OpenJDK with [Lambda](#) (JSR 335) support. The Lambda project aims to support programming in a multicore environment by adding closures and related features to the Java language

For documentations and other details, please see the [Lambda project page](#).

Please note:

The Lambda project has used source files that are not yet available in JDK8; therefore, these early access builds are created using the latest OpenJDK 7 source repository. This project will merge into OpenJDK 8 when the source files are available.

These bundles are meant to allow developers to try the Lambda features without making their own compilations. If you are looking for the latest JDK 8 builds, please download from [here](#).

License Agreement:

You must accept the [Pre-Production Software Evaluation Agreement for Java SE](#) to download this software.

Accept License Agreement | Decline License Agreement

Downloads (b50)

Platforms		JDK
Windows		zip (md5) 82 MB
Windows x64		zip (md5) 76 MB
Solaris SPARC	32-bit	tar.gz (md5) 333 MB
	64-bit*	tar.gz (md5) 492 MB
Solaris	x86	tar.gz (md5) 338 MB
	x64*	tar.gz (md5) 495 MB
Linux		tar.gz (md5) 80 MB
Linux x64		tar.gz (md5) 137 MB
Mac OS X		tar.gz (md5) 66 MB

*Note: Solaris 64-bit requires users to first install the 32-bit version.

There's not a moment to lose!

Mark Reinhold's Blog

Project Jigsaw: Late for the train

2012/07/17 08:58:00 -07:00

The aim of [Project Jigsaw](#) is to design and implement a standard module system for the Java SE Platform, and to apply that system to the Platform itself and to the JDK.

Jigsaw is currently slated for Java 8. The [proposed development schedule](#) for Java 8 expects work on major features to be finished by May 2013, in preparation for a final release around September. Steady progress is being made, but some significant technical challenges remain. There is, more importantly, not enough time left for the broad evaluation, review, and feedback which such a profound change to the Platform demands.

I therefore propose to defer Project Jigsaw to the next release, Java 9. In order to increase the predictability of all future Java SE releases, I further propose to aim explicitly for a regular two-year release cycle going forward.

There's not a moment to lose!

Mark Reinhold's Blog

Project Jigsaw: Late for the train: The Q&A

2012/08/24 08:52:12 -07:00

I recently proposed, to the Java community in general and to the [SE 8 \(JSR 337\) Expert Group](#) in particular, to defer [Project Jigsaw](#) from Java 8 to Java 9. I also proposed to aim explicitly for a regular two-year release cycle going forward. Herewith a summary of the key questions I've seen in reaction to these proposals, along with answers.

Making the decision

Q Has the [Java SE 8 Expert Group](#) decided whether to defer the addition of a module system and the modularization of the Platform to Java SE 9?

A No, it has not yet decided.

Q By when do you expect the EG to make this decision?

A In the next month or so.

Q How can I make sure my voice is heard?

A The EG will consider all relevant input from the wider community. If you have a prominent blog, column, or other communication channel then there's a good chance that we've [already seen your opinion](#). If not, you're welcome to send it to the [Java SE 8 Comments List](#), which is the EG's official feedback channel.

Q What's the overall tone of the feedback you've received?

A The feedback has been about evenly divided as to whether Java 8 should be delayed for Jigsaw, Jigsaw should be deferred to Java 9, or some other, usually less-realistic, option should be taken.

```
$ java org.planetjdk.aggregator.Main
```



```
$ java -cp $APPHOME/lib/jdom-1.0.jar:  
$APPHOME/lib/jaxen-1.0.jar:  
$APPHOME/lib/saxpath-1.0.jar:  
$APPHOME/lib/rome.jar-1.0.jar:  
$APPHOME/lib/rome-fetcher-1.0.jar:  
$APPHOME/lib/joda-time-1.6.jar:  
$APPHOME/lib/tagsoup-1.2.jar:  
org.planetjtdk.aggregator.Main
```

```
$ java -cp $APPHOME/lib/jdom-1.0.jar:  
$APPHOME/lib/jaxen-1.0.jar:  
$APPHOME/lib/saxpath-1.0.jar:  
$APPHOME/lib/rome.jar-1.0.jar:  
$APPHOME/lib/rome-fetcher-1.0.jar:  
$APPHOME/lib/joda-time-1.6.jar:  
$APPHOME/lib/tagsoup-1.2.jar:  
org.planetjtdk.aggregator.Main
```

module-info.java

```
module org.planetjtk.aggregator @ 1.0 {  
    requires jdom @ 1.0;  
    requires tagsoup @ 1.2;  
    requires rome @ 1.0;  
    requires rome-fetcher @ 1.0;  
    requires joda-time @ 1.6;  
    requires jaxp @ 1.4.4;  
    class org.openjdk.aggregator.Main;  
}
```

classpath



classpath

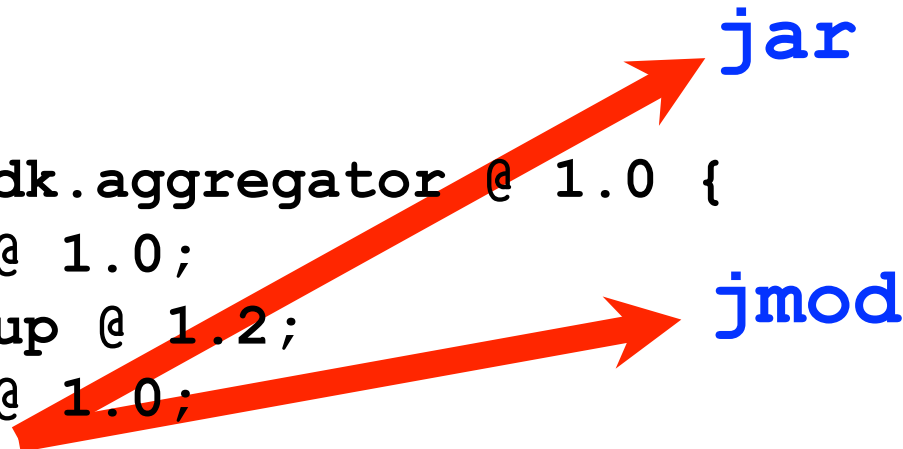


```
module org.planetjtdk.aggregator @ 1.0 {
    requires jdom @ 1.0;
    requires tagsoup @ 1.2;
    requires rome @ 1.0;
    requires rome-fetcher @ 1.0;
    requires joda-time @ 1.6;
    requires jaxp @ 1.4.4;
    class org.openjdk.aggregator.Main;
}
```

```
module org.planetjdk.aggregator @ 1.0 {  
    requires jdom @ 1.0;  
    requires tagsoup @ 1.2;  
    requires rome @ 1.0;  
    requires rome-fetcher @ 1.0;  
    requires joda-time @ 1.6;  
    requires jaxp @ 1.4.4;  
    class org.openjdk.aggregator.Main;  
}
```

jar

jmod




```
module org.planetjdk.aggregator @ 1.0 {
  requires jdom @ 1.0;
  requires tagsoup @ 1.2;
  requires rome @ 1.0;
  requires rome-fetcher @ 1.0;
  requires joda-time @ 1.6;
  requires jaxp @ 1.4.4;
  class org.openjdk.aggregator.Main;
}
```

jar

jmod

rpm

deb

mvn

```
module org.planetjdk.aggregator @ 1.0 {  
  requires jdom @ 1.0;  
  requires tagsoup @ 1.2;  
  requires rome @ 1.0;  
  requires rome-fetcher @ 1.0;  
  requires joda-time @ 1.6;  
  requires jaxp @ 1.4.4;  
  class org.openjdk.aggregator.Main;  
}
```

jar

jmod

rpm

deb

JDK 8

- Downloads
- Feedback Forum
- OpenJDK
- Planet JDK

Java™ Platform, Standard Edition 8 Early Access with Project Jigsaw

This page provides an Early Access of OpenJDK with Project Jigsaw support. The goal of Project Jigsaw is to design and implement a standard module system for the Java SE Platform, and to apply that system to the Platform itself and to the JDK.

For documentations and other details, please see the [Project Jigsaw page](#).

These bundles are meant to allow developers to try out Project Jigsaw without needing to build it from sources. If you are looking for the latest JDK 8 builds, please download from [here](#).

License Agreement:

You must accept the [Pre-Production Software Evaluation Agreement for Java SE](#) to download this software.

Accept License Agreement | Decline License Agreement

JDK modules image: This download is equivalent to the normal JDK download except that all components are pre-installed as modules. Note that the runtime no longer contains a "jre" directory, and rt.jar and tools.jar no longer exist.

JDK base image + jmod packages: This download contains a minimal "base" runtime and a directory of jmod packages with the JDK modules. The jmod packages can be installed directly via the "jmod install" command, or added to a file or http based module repository and installed automatically when required.

Downloads (b42):

See [Quick Start Guide](#) to get started.

See [Release Notes](#) for known issues.

Platforms	JDK modules image	JDK base image + jmod packages
Windows	zip (md5)	zip (md5)
	52 MB	55 MB
Windows x64	zip (md5)	zip (md5)
	46 MB	44 MB
Solaris SPARC	tar.gz (md5)	tar.gz (md5)
	310 MB	566 MB
Solaris SPARCv9	tar.gz (md5)	tar.gz (md5)
	468 MB	719 MB
Solaris x86	tar.gz (md5)	tar.gz (md5)
	315 MB	578 MB
Solaris x64	tar.gz (md5)	tar.gz (md5)
	330 MB	591 MB
Linux	tar.gz (md5)	tar.gz (md5)
	57 MB	63 MB
Linux x64	tar.gz (md5)	tar.gz (md5)
	115 MB	178 MB
Mac OS X	tar.gz (md5)	tar.gz (md5)
	43 MB	32 MB

JDK 8 – Proposed Content

Theme	Description/Content
Project Jigsaw	<ul style="list-style-type: none">• Module system for Java applications and for the Java platform
Project Lambda	<ul style="list-style-type: none">• Closures and related features in the Java language (JSR 335)• Bulk parallel operations in Java collections APIs (filter/map/reduce)
Oracle JVM Convergence	<ul style="list-style-type: none">• Complete migration of performance and serviceability features from JRockit, including Mission Control and the Flight Recorder
JavaFX 3.0	<ul style="list-style-type: none">• Next generation Java client, Multi-touch
JavaScript	<ul style="list-style-type: none">• Next-gen JavaScript-on-JVM engine (Project Nashorn)• JavaScript/Java interoperability on JVM
Device Support	<ul style="list-style-type: none">• Camera, Location, Compass and Accelerometer
Developer Productivity	<ul style="list-style-type: none">• Annotations onTypes (JSR 308), Minor language enhancements
API and Other Updates	<ul style="list-style-type: none">• Enhancements to Security, Date/Time (JSR 310), Networking, Internationalization, Accessibility, Packaging/Installation

Additional Disclaimers

- Some *ideas* for the Java Platform are shown on the following slides
- Large R&D effort required
- Content and timing highly speculative
- Some things will turn out to be bad ideas
- New ideas will be added
- Java's future is bright (in our humble opinion)!

Java SE 9 (and beyond...)

Interoperability	<ul style="list-style-type: none">• Multi-language JVM• Improved Java/Native integration
Cloud	<ul style="list-style-type: none">• Multi-tenancy support• Resource management
Ease of Use	<ul style="list-style-type: none">• Self-tuning JVM• Language enhancements
Advanced Optimizations	<ul style="list-style-type: none">• Unified type system• Data structure optimizations
Works Everywhere and with Everything	<ul style="list-style-type: none">• Scale down to embedded, up to massive servers• Support for heterogeneous compute models

Vision: Interoperability

- Improved support for non-Java languages
 - Invokedynamic (done)
 - Java/JavaScript interop (in progress – JDK 8)
 - Meta-object protocol (JDK 9)
 - Long list of JVM optimizations (JDK 9+)
- Java/Native
 - Calls between Java and Native without JNI boilerplate (JDK 9)

Vision: Cloud

- Multi-tenancy (JDK 8+)
 - Improved sharing between JVMs in same OS
 - Per-thread/threadgroup resource tracking/management
- Hypervisor aware JVM (JDK 9+)
 - Co-operative memory page sharing
 - Co-operative lifecycle, migration

Vision: Language Features

- Large data support (JDK 9)
 - Large arrays (64 bit support)
- Unified type system (JDK 10+)
 - No more primitives, make everything objects
- Other type reification (JDK 10+)
 - True generics
 - Function types
- Data structure optimizations (JDK 10+)
 - Structs, multi-dimensional arrays, etc
 - Close last(?) performance gap to low-level languages

Vision: Integration

- Modern device support (JDK 8+)
 - Multitouch (JDK 8)
 - Location (JDK 8)
 - Sensors – compass, accelerometer, temperature, pressure, ... (JDK 8+)
- Heterogenous compute models (JDK 9+)
 - Java language support for GPU, FPGA, offload engines, remote PL/SQL...

The Path Forward

- Open development
 - Prototyping and R&D in OpenJDK
 - Cooperate with partners, academia, greater community
- Work on next JDK, future features in parallel
- 2-year cycle for Java SE releases

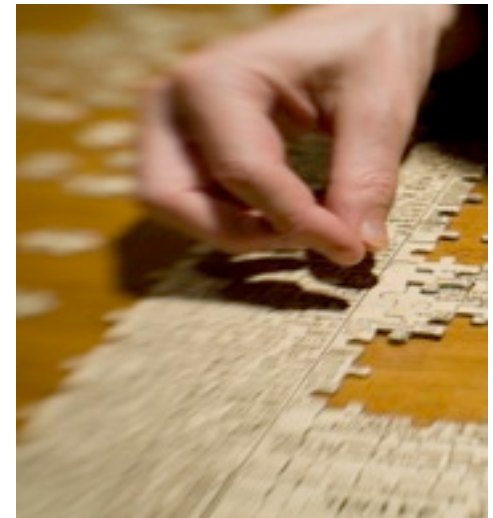
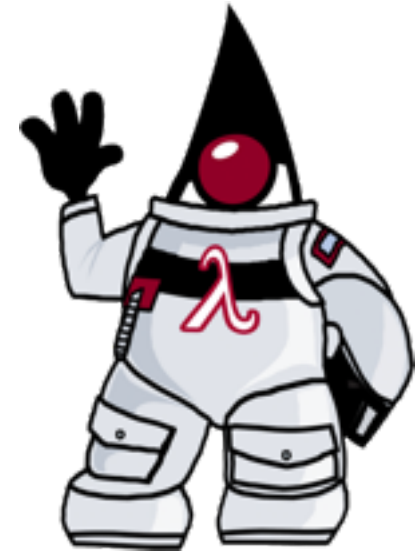
Conclusions

- The Java platform will continue to evolve
- Java SE 8 will add some nice, big features
- Expect to see more in Java SE 9 and beyond
- Java is not the new Cobol

Further Information

- Project Lambda
 - openjdk.java.net/projects/lambda

- Project Jigsaw
 - openjdk.java.net/projects/jigsaw



Q&A

3.– 6. September 2012
in Nürnberg



Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Dalibor Topic

ORACLE Deutschland B.V. & Co. KG