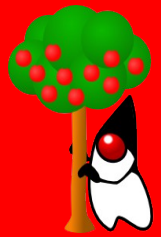
11-12-13



MADRIDJUG

To Java SE 8, and Beyond
(Plan B)

Francisco Morero Peyrona
EMEA Java Community Leader

javaHispano
www.javahispano.org

ORACLE

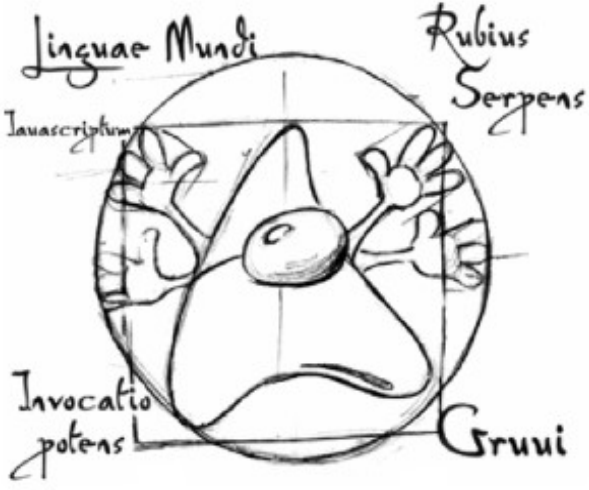
ORACLE®

1.0 Revolution 1.2 5.0 7 9

Evolution 1.1 1.3 1.4 6 8

1996 1997 1998 2000 2002 2004 2006 2010 ...2012 2020?

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

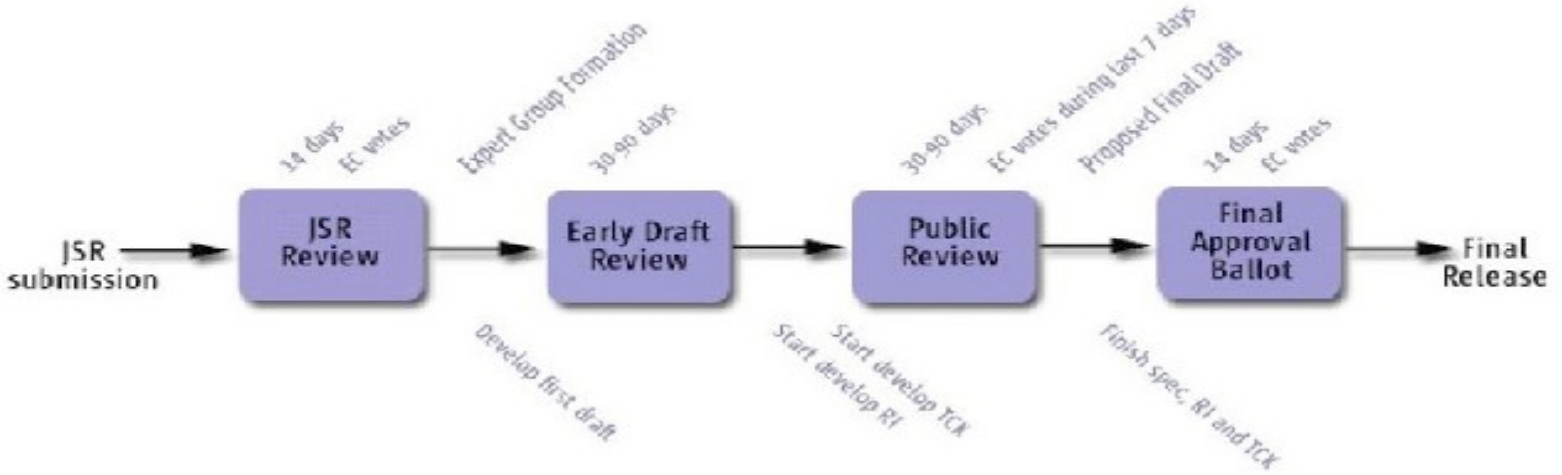
Java language principles

- 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
- Every "good" feature adds more "bad" weight
- Sometimes it is best to leave things out
- Simplicity matters

How Java Evolves and Adapts

Of the community, by the community, for the community

Java Community Process

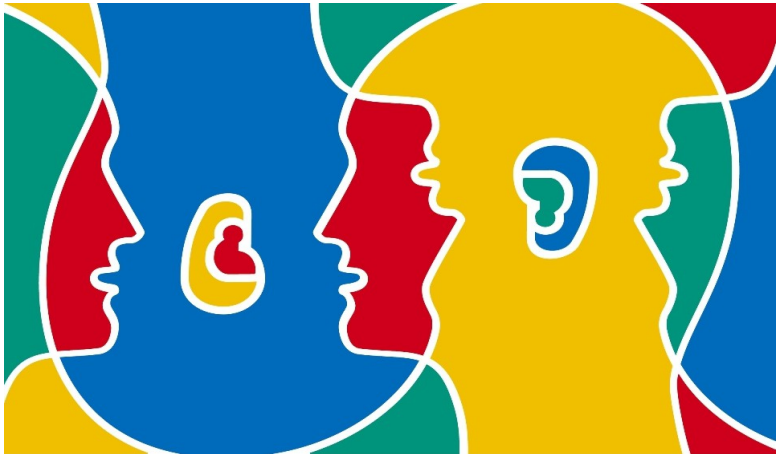



8

JDK 8 – Proposed Content

Theme	Description/Content
Project Jigsaw	• Module system for Java applications and for the Java platform
Project Lambda	• Closures and related features in the Java language (JSR 335) • Bulk parallel operations in Java collections APIs (filter/map/reduce)
Oracle JVM Convergence	• Complete migration of performance and serviceability features from JRockit, including Mission Control and the Flight Recorder
JavaFX 3.0	• Next generation Java client, Multi-touch
JavaScript	• Next-gen JavaScript-on-JVM engine (Project Nashorn) • JavaScript/Java interoperability on JVM
Device Support	• Camera, Location, Compass and Accelerometer
Developer Productivity	• Annotations onTypes (JSR 308), Minor language enhancements
API and Other Updates	• Enhancements to Security, Date/Time (JSR 310), Networking, Internationalization, Accessibility, Packaging/Installation

Language



Big Disclaimer

The syntax used in the following slides may change

Lambda Expressions I

Closures and Functional Programming

- Lambda expressions provide anonymous function types to Java
- They replace the use of single abstract method types (SAM)
- They are just instances of Runnable

Syntax

Argument List	Arrow Token	Body
(int x, int y)	->	x + y

Body can be:

(a) A single expression: body is evaluated and result value returned.

(b) Statement block: body is evaluated as a method body and “return” statement returns control to the caller.

Lambda Expressions II

Closures and Functional Programming

- Type of parameters can be explicitly declared or taken from context.
- Argument parenthesis are optional when there is only one and its type can be inferred.
- Argument List can be empty, if it is so, parenthesis must exist.
- If body has more that one statement, then curly braces are needed.

```
Runnable r = () ->
   { System.out.println( "I'm a Runnable!" ); };

r.run();



jbutton.addActionListener(
     e -> { System.out.println( "Clicked" ); } );
```

- `n -> n % 2 != 0;`
- `(char c) -> c == 'y';`
- `(x, y) -> x + y;`
- `(int a, int b) -> a * a + b * b;`
- `() -> { return 3.14 };`
- `(String s) ->`
  `{ System.out.println(s); };`

Lambda Expressions III

Closures and Functional Programming

```
 6  public class RunnableTest {
 7    public static void main(String[] args) {
 8
 9      System.out.println("=== RunnableTest ===");
10
11      // Anonymous Runnable
12      Runnable r1 = new Runnable(){
13
14        @Override
15        public void run(){
16          System.out.println("Hello world one!");
17        }
18      };
19
20      // Lambda Runnable
21      Runnable r2 = () -> System.out.println("Hello world two!");
22
23      // Run em!
24      r1.run();
25      r2.run();
26
27    }
28  }
```

Extension Methods

Bringing Multiple Inheritance to Java

- Provide a mechanism to add new methods to existing interfaces
- Without breaking backwards compatability
- Gives Java multiple inheritance

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

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

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

Annotations on Java Types

- Annotations can currently only be used on type declarations
  - Classes, methods, variable definitions
- Extension for places where types are used
  - e.g. Parameters
- Permits error detection by pluggable type checkers
  - e.g. null pointer errors, race conditions, etc

```
public void process(@notnull List data) {…}
```

Access to Parameter Names at Runtime

- Mechanism to retrieve parameter names of methods and constructors
  - At runtime via core reflection
- Improved code readability
  - Eliminate redundant annotations
- Improve IDE capabilities
  - Auto-generate template code

Small Things

- Repeating annotations
  - ➡ Multiple annotations with the same type applied to a single program element
- No more **`apt`** tool and associated API
  - ➡ Complete the transition to the JSR 269 implementation
- DocTree API
  - ➡ Provide access to the syntactic elements of a javadoc comment
- DocLint tool
  - ➡ Use DocTree API to identify basic errors in javadoc comments
- Javadoc support in **`javax.tools`**
  - ➡ Invoke javadoc tools from API as well as command line/exec

Library

Concurrency Updates

- Scalable update variables
  - `DoubleAccumulator`, `DoubleAdder`, etc
  - Multiple variables avoid update contention
  - Good for frequent updates, infrequent reads
- `ConcurrentHashMap` updates
  - Improved scanning support, key computation
- `ForkJoinPool` improvements
  - Completion based design for IO bound applications
  - Thread that is blocked hands work to thread that is running

Bulk Data Operations for Collections

- Adding .Net functionality
  - LINQ style processing
- Serial and parallel implementations
  - Generally expressed with Lambda statements
- Parallel implementation builds on Fork-Join framework

Date and Time APIs

- A new date, time, and calendar API for the Java SE platform
- Supports standard time concepts
  - Partial, duration, period, intervals
  - date, time, instant, and time-zone
- Initially provides a limited set of calendar systems and will be extensible to others
- Uses relevant standards, including ISO-8601, CLDR, and BCP47
- Based on an explicit time-scale with a connection to UTC

JDBC 4.2

Minor enhancements for usability and portability

- Add setter/update methods
  - **`ResultSet`**, **`PreparedStatement`** and **`CallableStatement`**
  - Support new data types such as those being defined in JSR 310
- REF_CURSOR support for **`CallableStatement`**
- Extended **`DatabaseMetaData.getIndexInfo`**
  - new columns for CARDINALITY and PAGES which return a long value
- New **`DatabaseMetaData`** method
  - **`getMaxLogicalLobSize`**
  - Return the logical maximum size for a LOB

Small (or perhaps not) things

- Enhance core libraries with Lambdas (not small thing)
- Parallel array sorting (improve ≈ x4)
- Base 64 Encoding and Decoding (no need of undocumented API)
- Charset implementation improvements
  - Reduced size of charsets
  - Improved performance of encoding/decoding
  - Reduced core-library memory usage
- Reduced object size, disable reflection compiler, internal table sizes
- Optimize **`java.text.DecimalFormat.format`** (improve x100 or x1000)
- Statically Linked JNI Libraries (needed for embedded applications)
- Handle frequent **`HashMap`** collisions with balanced trees

9...

Java SE 9 (and beyond…)

Interoperability	• Multi-language JVM • Improved Java/Native integration
Cloud	• Multi-tenancy support • Resource management
Ease of Use	• Self-tuning JVM • Language enhancements
Advanced Optimizations	• Unified type system • Data structure optimizations
Works Everywhere and with Everything	• Scale down to embedded, up to massive servers • Support for heterogenuous 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

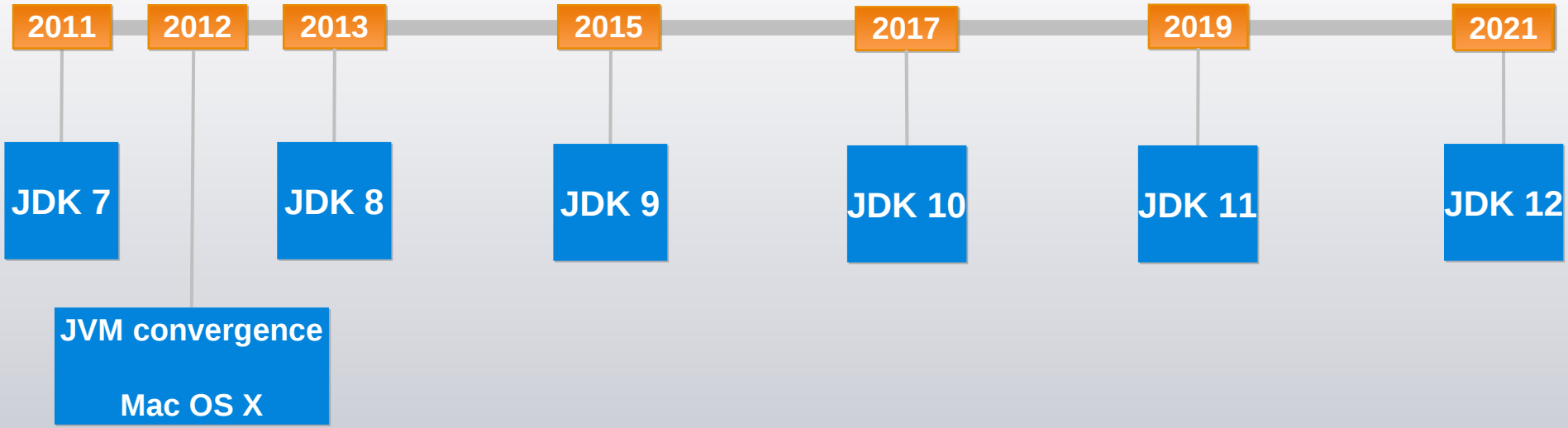
Vision: Integration

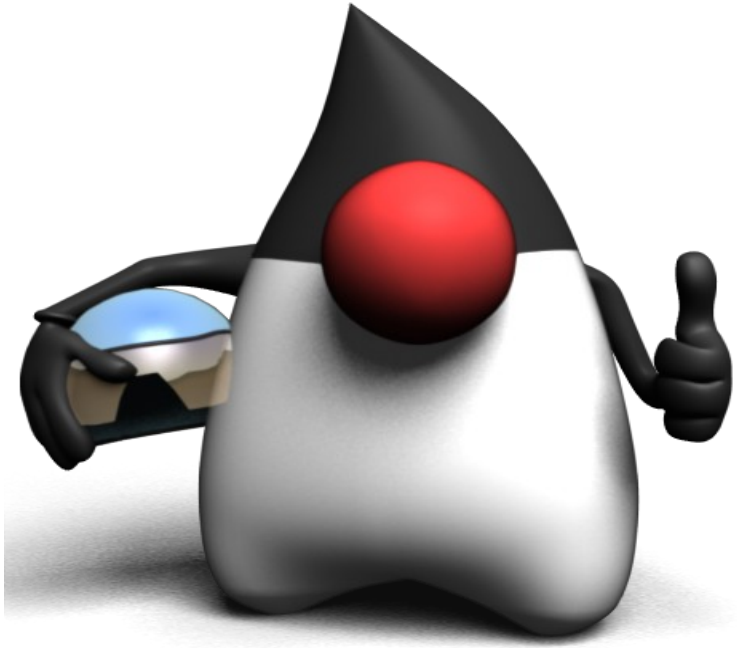
- Modern device support (JDK 8+)
  - Multitouch (JDK 8+)
  - Location (JDK 8+)
  - Sensors – compass, accelerometer, temperature, pressure, ... (JDK 8+)
- Heterogenous compute models (JDK 9+)
  - Java language support for GPU, FPGA, offload engines, remote PL/SQL...

The Path Forward (JDK 10)

- 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

Java SE from JDK 7 to JDK 12





Try not: do or do not, there is no try.

ORACLE®