

ORACLE®



ORACLE®

Java SE 7: The Platform Evolves

Dalibor Topić
Java F/OSS Ambassador

ORACLE®

Priorities for the Java Platforms

- ✓ Grow Developer Base
- ✓ Grow Adoption
- ✓ Increase Competitiveness
- ✓ Adapt to change



Java Communities

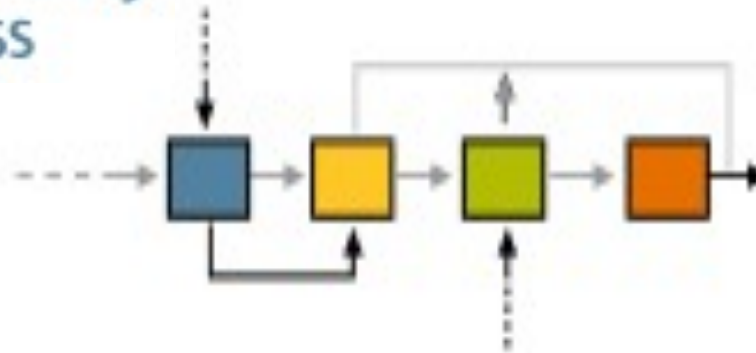


ORACLE

How Java Evolves and Adapts



Java
Community
Process



**Community Development of
Java Technology
Specifications**

ORACLE

JCP Reforms



- Developers' voice in the Executive Committee
 - SOUJava
 - Goldman Sachs
 - London JavaCommunity
 - Alex Terrazas
- JCP starting a program of reform
 - JSR 348: Towards a new version of the JCP



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”

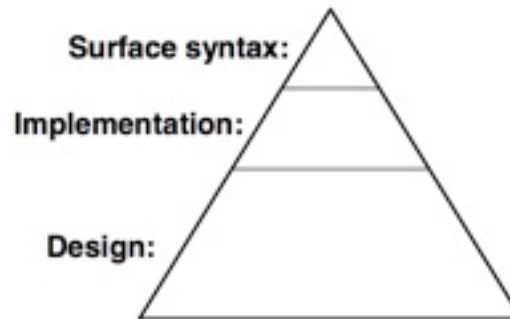
also “Growing a Language” - Guy Steele 1999

“The Feel of Java” - James Gosling 1997

ORACLE

So you want to change the language?

- Update the Java Language Spec.
- Compiler Implementation
- Essential library support
- Write tests
- Update the JVM Spec.
- Future language evolution
- Update the JVM and class file tools
- Update JNI
- Update the reflective APIs
- Update serialization
- Update javadoc output
- Kinds of compatibility



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



Small Language Changes Project Coin

coin, *n.* A piece of small change
coin, *v.* To create new language

Project Coin Constraints

- *Small* language changes
 - Small in specification, implementation, testing
 - No new keywords!
 - Wary of type system changes
- Coordinate with larger language changes
 - Project Lambda
 - Modularity
- One language, one javac

Better Integer Literal

- Binary literals

```
int mask = 0b101010101010;
```

- With underscores for clarity

```
int mask = 0b1010_1010_1010;
```

```
long big = 9_223_783_036_967_937L;
```

String Switch Statement

- Today case label includes integer constants and enum constants
- Strings are constants too (immutable)

Discriminating Strings Today

```
int monthNameToDays(String s, int year) {  
  
    if ("April".equals(s) || "June".equals(s) ||  
        "September".equals(s) || "November".equals(s))  
        return 30;  
  
    if ("January".equals(s) || "March".equals(s) ||  
        "May".equals(s) || "July".equals(s) ||  
        "August".equals(s) || "December".equals(s))  
        return 31;  
  
    if ("February".equals(s))  
        ...  
}
```

Strings in Switch Statements

```
int monthNameToDays(String s, int year) {  
    switch(s) {  
        case "April": case "June":  
        case "September": case "November":  
            return 30;  
  
        case "January": case "March":  
        case "May": case "July":  
        case "August": case "December":  
            return 31;  
  
        case "February":  
            ...  
        default:  
            ...  
    }  
}
```


Simplifying Generics

- Pre-generics

```
List strList = new ArrayList();
```

Simplifying Generics

- Pre-generics

```
List strList = new ArrayList();
```

- With Generics

```
List<String> strList = new ArrayList<String>();
```

Simplifying Generics

- Pre-generics

```
List strList = new ArrayList();
```

- With Generics

```
List<String> strList = new ArrayList<String>();  
List<Map<String, List<String>> strList =  
    new ArrayList<Map<String, List<String>>();
```

Diamond Operator

- Pre-generics

```
List strList = new ArrayList();
```

- With Generics

```
List<String> strList = new ArrayList<String>();  
List<Map<String, List<String>> strList =  
    new ArrayList<Map<String, List<String>>();
```

- With diamond (<>) compiler infers type

```
List<String> strList = new ArrayList<>();  
List<Map<String, List<String>> strList =  
    new ArrayList<>();
```

Copying a File

```
InputStream in = new FileInputStream(src);  
OutputStream out = new FileOutputStream(dest);
```

```
byte[] buf = new byte[8192];  
int n;
```

```
while (n = in.read(buf)) >= 0)  
    out.write(buf, 0, n);
```

Copying a File (Better, but wrong)

```
InputStream in = new FileInputStream(src);
OutputStream out = new FileOutputStream(dest);

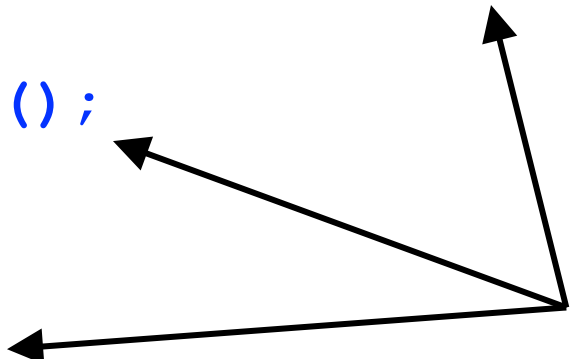
try {
    byte[] buf = new byte[8192];
    int n;
    while (n = in.read(buf)) >= 0)
        out.write(buf, 0, n);
} finally {
    in.close();
    out.close();
}
```

Copying a File (Correct, but complex)

```
InputStream in = new FileInputStream(src);
try {
    OutputStream out = new FileOutputStream(dest);
    try {
        byte[] buf = new byte[8192];
        int n;
        while (n = in.read(buf) >= 0)
            out.write(buf, 0, n);
    } finally {
        out.close();
    }
} finally {
    in.close();
}
```

Copying a File (Correct, but complex)

```
InputStream in = new FileInputStream(src);
try {
    OutputStream out = new FileOutputStream(dest);
    try {
        byte[] buf = new byte[8192];
        int n;
        while (n = in.read(buf)) >= 0)
            out.write(buf, 0, n);
    } finally {
        out.close();
    }
} finally {
    in.close();
}
```



Exception thrown from potentially three places. Details of first two could be lost

Automatic Resource Management

```
try (InputStream in = new FileInputStream(src) ,
     OutputStream out = new FileOutputStream(dest) )
{
    byte[] buf = new byte[8192];
    int n;
    while (n = in.read(buf)) >= 0)
        out.write(buf, 0, n);
}
```

The Details

- Compiler desugars try-with-resources into nested try-finally blocks with variables to track exception state
- Suppressed exceptions are recorded for posterity using a new facility of Throwable
- API support in JDK 7
 - New superinterface `java.lang.AutoCloseable`
 - All `AutoCloseable` and by extension `java.io.Closeable` types useable with try-with-resources
 - anything with a `void close()` method is a candidate
 - JDBC 4.1 refitted as `AutoCloseable` too

More Informative Backtraces

```
java.io.IOException
  at Suppress.write(Suppress.java:19)
  at Suppress.main(Suppress.java:8)
  Suppressed: java.io.IOException
    at Suppress.close(Suppress.java:24)
    at Suppress.main(Suppress.java:9)
  Suppressed: java.io.IOException
    at Suppress.close(Suppress.java:24)
    at Suppress.main(Suppress.java:9)
```

Varargs Warnings

```
class Test {  
    public static void main(String... args) {  
        List<List<String>> monthsInTwoLanguages =  
            Arrays.asList(Arrays.asList("January",  
                                         "February"),  
                        Arrays.asList("Gennaio",  
                                      "Febbraio" ));  
    }  
}
```

```
Test.java:7: warning:  
[unchecked] unchecked generic array creation  
for varargs parameter of type List<String>[]  
    Arrays.asList(Arrays.asList("January",  
                               ^  
1 warning
```

Varargs Warnings Revised

- New mandatory compiler warning at suspect varargs method declarations
- By applying an annotation at the declaration, warnings at the declaration *and call sites* can be suppressed
- `@SuppressWarnings(value = "unchecked")`
- `@SafeVarargs`

Exceptions Galore

```
try {  
    ...  
} catch (ClassNotFoundException cnfe) {  
    doSomethingClever(cnfe);  
    throw cnfe;  
} catch (InstantiationException ie) {  
    log(ie);  
    throw ie;  
} catch (NoSuchMethodException nsme) {  
    log(nsme);  
    throw nsme;  
} catch (InvocationTargetException ite) {  
    log(ite);  
    throw ite;  
}
```

Multi-Catch

```
try {  
    ...  
} catch (ClassCastException e) {  
    doSomethingClever(e);  
    throw e;  
} catch (InstantiationException |  
         NoSuchMethodException |  
         InvocationTargetException e) {  
    log(e);  
    throw e;  
}
```

The background of the slide consists of several horizontal wooden shelves against a wood-grain wall. The shelves are empty and evenly spaced.

Library Changes

New I/O 2 (NIO2) Libraries

JSR 203

- Original Java I/O APIs presented challenges for developers
 - Not designed to be extensible
 - Many methods do not throw exceptions as expected
 - **rename ()** method works inconsistently
 - Developers want greater access to file metadata
- Java NIO2 solves these problems

ORACLE

Java NIO2 Features

- **Path** is a replacement for **File**
 - Biggest impact on developers
- Better directory support
 - `list()` method can stream via iterator
 - Entries can be filtered using regular expressions in API
- Symbolic link support
- **java.nio.file.Filesystem**
 - interface to a filesystem (FAT, ZFS, Zip archive, network, etc)
- **java.nio.file.attribute** package
 - Access to file metadata

Path Class

- Equivalent of `java.io.File` in the new API
 - Immutable
- Have methods to access and manipulate `Path`
- Few ways to create a `Path`
 - From `Paths` and `FileSystem`

```
//Make a reference to the path
```

```
Path home = Paths.get("/home/fred");
```

```
//Resolve tmp from /home/fred -> /home/fred/tmp
```

```
Path tmpPath = home.resolve("tmp");
```

```
//Create a relative path from tmp -> ..
```

```
Path relativePath = tmpPath.relativeTo(home)
```

```
File file = relativePath.toFile();
```

File Operation – Copy, Move

- File copy is really easy

- With fine grain control

```
Path src = Paths.get("/home/fred/readme.txt");  
Path dst = Paths.get("/home/fred/copy_readme.txt");  
  
Files.copy(src, dst,  
           StandardCopyOption.COPY_ATTRIBUTES,  
           StandardCopyOption.REPLACE_EXISTING);
```

- File move is supported

- Optional atomic move supported

```
Path src = Paths.get("/home/fred/readme.txt");  
Path dst = Paths.get("/home/fred/readme.1st");  
  
Files.move(src, dst, StandardCopyOption.ATOMIC_MOVE);
```

Directories

- `DirectoryStream` iterate over entries
 - Scales to large directories
 - Uses less resources
 - Smooth out response time for remote file systems
 - Implements **`Iterable`** and **`Closeable`** for productivity
- Filtering support
 - Build-in support for glob, regex and custom filters

```
Path srcPath = Paths.get("/home/fred/src");  
try (DirectoryStream<Path> dir =  
    srcPath.newDirectoryStream("*.*java")) {  
    for (Path file: dir)  
        System.out.println(file.getName());  
}
```

Concurrency APIs

- JSR166y
 - Update to JSR166x which was an update to JSR166
- Adds a lightweight task framework
 - Also referred to as Fork/Join
- **Phaser**
 - Barrier similar to **CyclicBarrier** and **CountDownLatch**
- **TransferQueue** interface
 - Extension to **BlockingQueue**
 - Implemented by **LinkedTransferQueue**

Fork Join Framework

- Goal is to take advantage of multiple processor
- Designed for task that can be broken down into smaller pieces
 - Eg. Fibonacci number $\text{fib}(10) = \text{fib}(9) + \text{fib}(8)$
- Typical algorithm that uses fork join

if I can manage the task

perform the task

else

fork task into x number of smaller/similar task

join the results

Key Classes

- **ForkJoinPool**
 - Executor service for running **ForkJoinTask**
- **ForkJoinTask**
 - The base class for forkjoin task
- **RecursiveAction**
 - A subclass of **ForkJoinTask**
 - A recursive resultless task
 - Implements **compute ()** abstract method to perform calculation
- **RecursiveTask**
 - Similar to **RecursiveAction** but returns a result

ForkJoin Example – Fibonacci

```
public class Fibonacci extends RecursiveTask<Integer> {
    private final int number;
    public Fibonacci(int n) { number = n; }

    @Override protected Integer compute() {
        switch (number) {
            case 0: return (0);
            case 1: return (1);
            default:
                Fibonacci f1 = new Fibonacci(number - 1);
                Fibonacci f2 = new Fibonacci(number - 2);
                f1.fork(); f2.fork();
                return (f1.join() + f2.join());
        }
    }
}
```

ForkJoin Example – Fibonacci

```
ForkJoinPool pool = new ForkJoinPool();  
Fibonacci r = new Fibonacci(10);  
pool.submit(r);
```

```
while (!r.isDone()) {  
    //Do some work  
    ...  
}
```

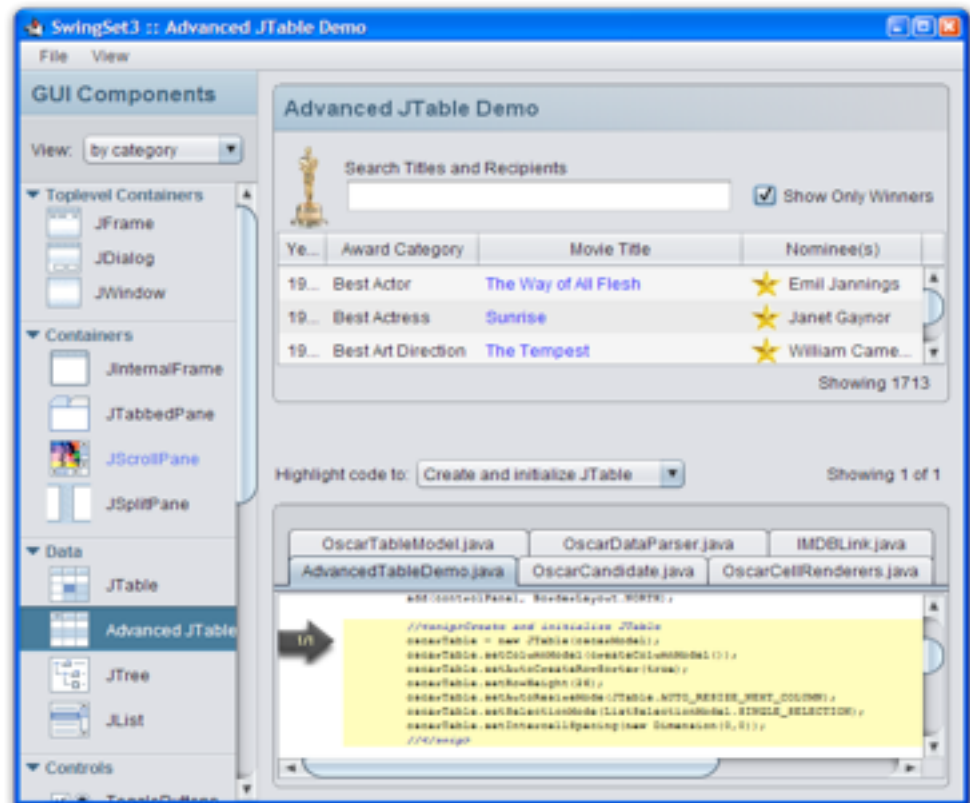
```
System.out.println("Result of fib(10) = "  
    + r.get());
```

Client Libraries

- Nimbus Look and Feel
- Platform APIs for shaped and translucent windows
- JLayer (formerly from Swing labs)
- Optimised 2D rendering

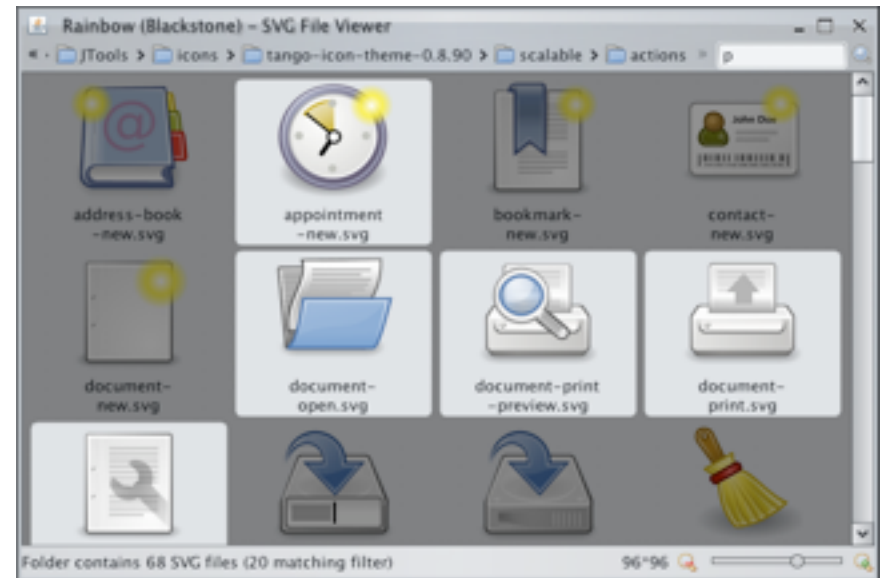
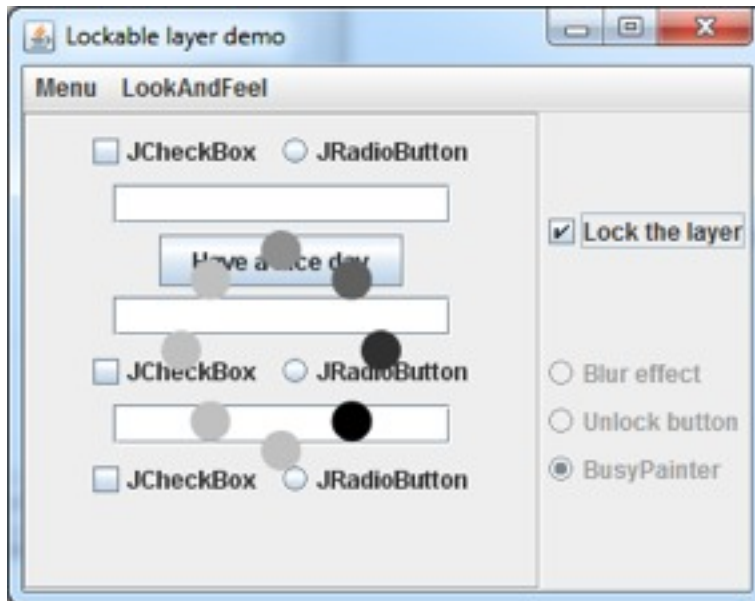
Nimbus Look and Feel

- Better than Metal for cross platform look-and-feel
- Introduced in Java SE 6u10, now part of Swing
- Not the default L&F



JLayer component

Easy enrichment for Swing components



JLayer component

The universal decorator

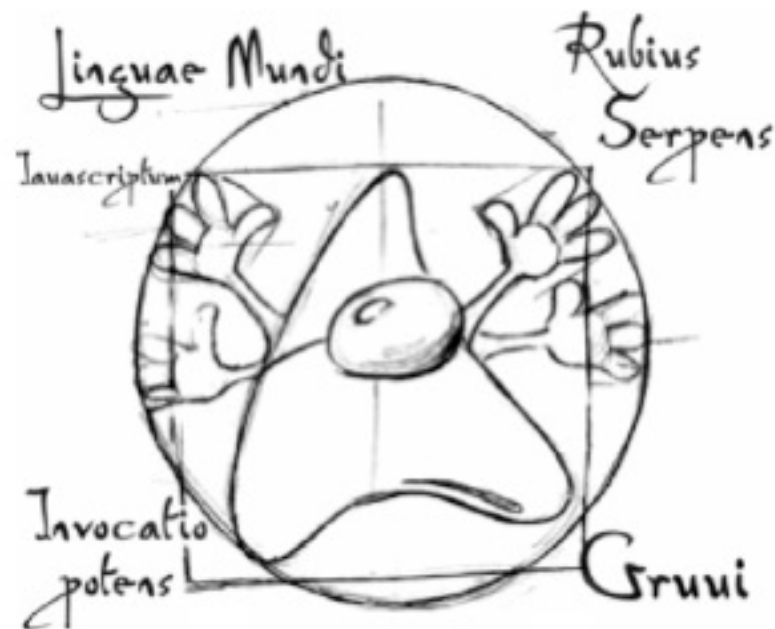
- Transparent decorator for a Swing component
- Controls the painting of its subcomponents
- Catches all input and focus events for the whole hierarchy

```
// wrap your component with JLayer
JLayer<JPanel> layer = new JLayer<JPanel>(panel);

// custom ui provides all extra functionality
layer.setUI(myLayerUI);

// add the layer as usual component
frame.add(layer);
```

The DaVinci Machine Project (JSR-292) (A multi-language renaissance for the JVM)



ORACLE

Languages Like Virtual Machines

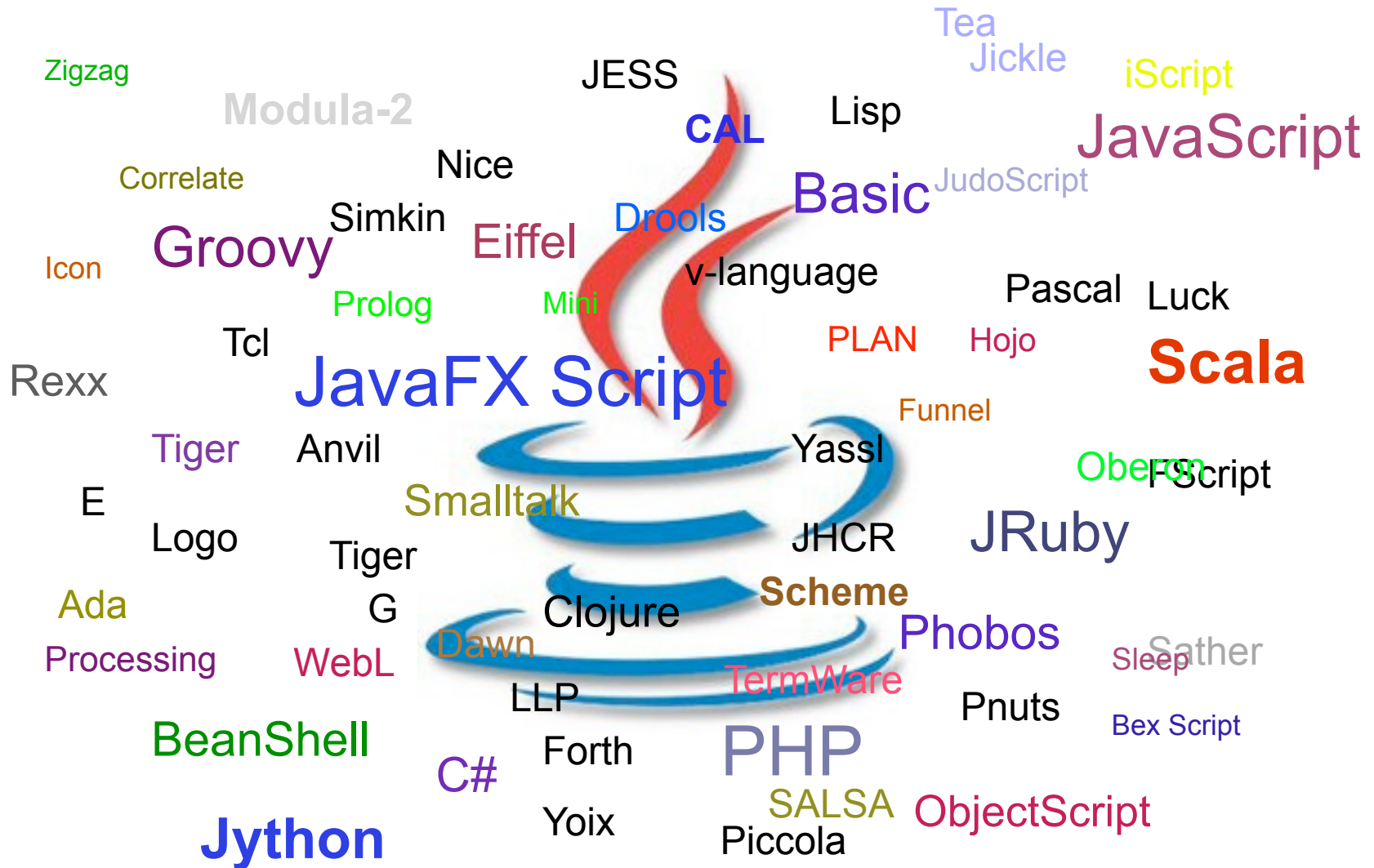
- Programming languages need runtime support
 - Memory management / Garbage collection
 - Concurrency control
 - Security
 - Reflection
 - Debugging integration
 - Standard libraries
- Compiler writers have to build these from scratch
- Targeting a VM allows reuse of infrastructure

JVM Specification

“The Java virtual machine knows nothing about the Java programming language, only of a particular binary format, the class file format.”

1.2 The Java Virtual Machine Spec.

Languages Running on the JVM



InvokeDynamic Bytecode

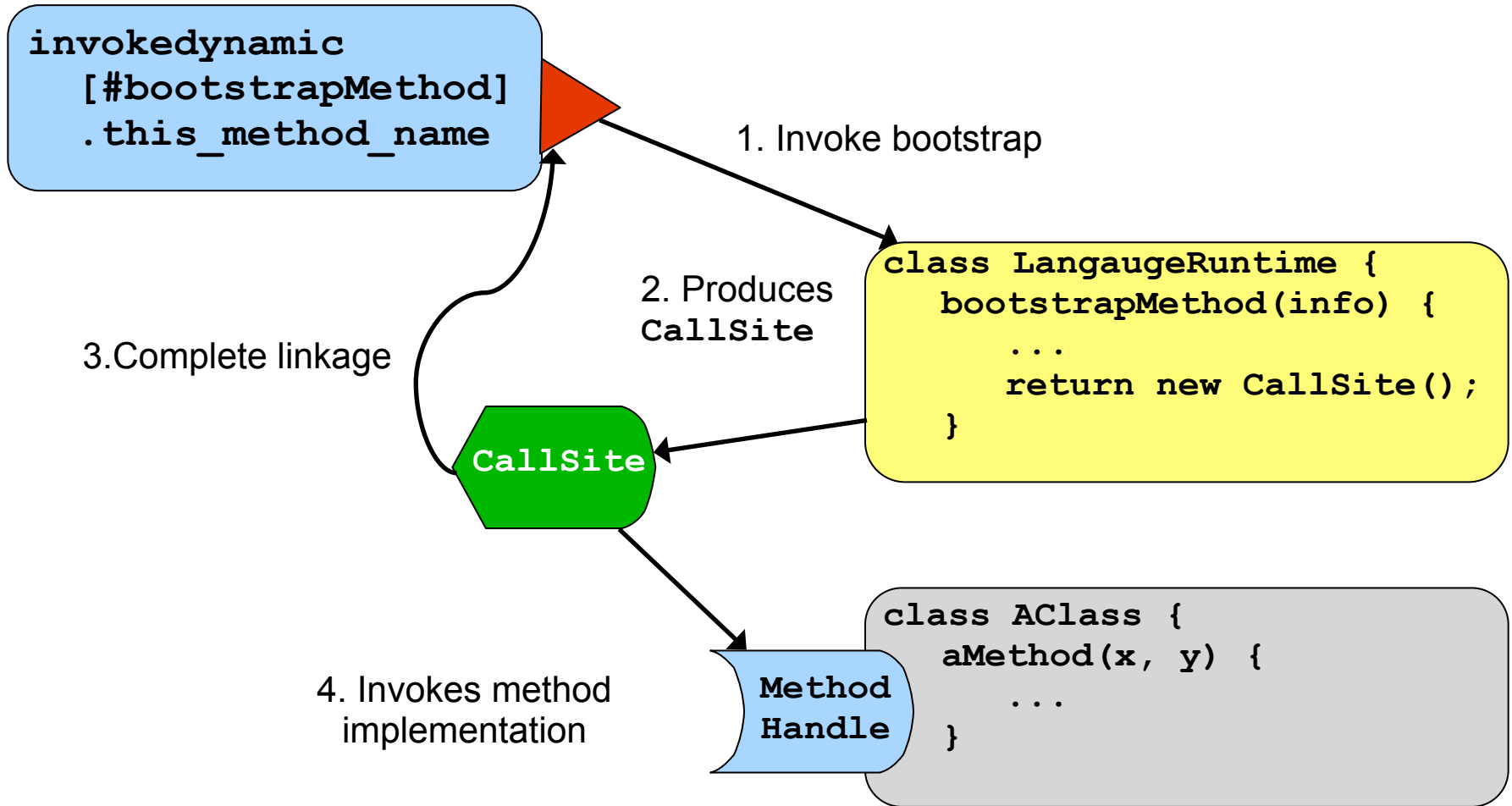
- JVM currently has four ways to invoke method
 - `Invokevirtual`, `Invokeinterface`, `Invokestatic`, `Invokespecial`
- All require full method signature data
- `InvokeDynamic` will use method handle
 - Effectively an indirect pointer to the method
- When dynamic method is first called bootstrap code determines method and creates handle
- Subsequent calls simply reference defined handle
- Type changes force a re-compute of the method location and an update to the handle
 - Method call changes are invisible to calling code

CallSite and MethodHandle

- **invokedynamic** linked to a **CallSite**
 - **CallSite** can be linked or unlinked
 - **CallSite** holder of **MethodHandle**
- **MethodHandle** is a directly executable reference to an underlying method, constructor, field
 - Can transform arguments and return type
 - Transformation – conversion, insertion, deletion, substitution

invokedynamic Illustrated

`this[method_name](x, y)`



Miscellaneous Things

- Security
 - Elliptic curve cryptography
 - TLS 1.2
- JAXP 1.4.4
- JAX-WS 2.2
- JAXB 2.2
- ClassLoader architecture changes
- `close ()` for `URLClassLoader`
- Javadoc support for CSS

JDK 7 Platform Support

- Windows x86
 - Server 2008, Server 2008 R2, 7 & 8 (when it GAs)
 - Windows Vista, XP
- Linux x86
 - Oracle Linux 5.5+, 6.x
 - Red Hat Enterprise Linux 5.5+, 6.x
 - SuSE Linux Enterprise Server 10.x, 11.x
 - Ubuntu Linux 10.04 LTS, 11.04
- Solaris x86/SPARC
 - Solaris 10.9+, 11.x
- **Apple OSX x86**
 - **will be supported post-GA, detailed plan TBD**

Note: JDK 7 should run on pretty much any Windows/Linux/Solaris. These configurations are the ones primarily tested by Oracle, and for which we provide commercial support.

JVM Convergence – Forward looking

Project “HotRockit”

JDK 7 GA – 07/11

- Hotspot 21
- Java SE 7 Support
- Rebranding
- Improved JMX Agent
- Command line servicability tool (jrcmd)

--- Premium ---

- Improved JRockit Mission Control Console support

JDK 7u2

- Hotspot 22
- Performance
- Enable large heaps with reasonable latencies

JDK 7uX

- Hotspot 23
- More performance
- Improved command line servicability (jcmd)
- Enable large heaps with consistent reasonable latencies
- No PermGen

--- Premium ---

- Complete JRockit Flight Recorder Support

JDK 8 GA

- Hotspot24
- Java SE 8 Support
- All performance features from JRockit ported
- All servicability features from JRockit ported
 - Compiler controls
 - Verbose logging
- Premium ---
- JRockit Mission Control Memleak Tool Support
- Soft Real Time GC

ORACLE

Java SE 8



Project Jigsaw (JSR-294)

Modularising the Java Platform



Project Lambda (JSR 335)

Closures and lambda expressions

Better support for multi-core processors




More Project Coin

Small Language Changes

ORACLE

Conclusions

- Java SE 7
 - Incremental changes
 - Evolutionary, not revolutionary
 - Good solid set of features to make developers life easier
- Java SE 8
 - Major new features: Modularisation and Closures
 - More smaller features to be defined
- Java continues to grow and adapt to the changing world of IT



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



ORACLE