POWER CATCH-UP
EVERYTHING PRACTICAL AND IMPORTANT IN JAVA 9 TO 13

Benjamin Schmid @bentolor <benjamin.schmid@exxcellent.de>
Public Support

Oracle Commercial Support

Oracle Extended Support

[Support Model: https://craftsmen.nl/java-11-is-here-to-stay/]
EVERYTHING-IS-SO-AWFUL™

1. Oracle Java 8 installations are dead!
   Unmaintained: No free & security- & bugfix updates.

2. Always **only 1 supported** Oracle Java

3. **Every 6 months** new major version;
   predecessor **expire the same day**

4. You must **always pay in production** for Oracle Java
LOVE, PEACE & HAPPINESS...

Starting with Java 11

- Oracle themselves releases OpenJDK (⚠ but only 6 month support cadence)
- OpenJDK *defacto identical* to Oracle JDK
- Numerous (new) OpenJDK-based options. Some with $free support
tl;dr: AdoptOpenJDK

Prebuilt OpenJDK Binaries for Free!

Java™ is the world's leading programming language and platform. AdoptOpenJDK open source initiative builds and distributes prebuilt binaries from Oracle™'s Java™ and a choice of either the OpenJDK HotSpot or OpenJDK OpenJ9 VM.

AdoptOpenJDK binaries and tools are open source, licensed and available for free.

Download for Linux x64

1. Choose a Version
   - OpenJDK 8 (Builds)
   - OpenJDK 11 (Builds)
   - OpenJDK 13 (Builds)

2. Choose a JVM
   - HotSpot
   - OpenJ9

Latest release [automatically update]

AGENDA

1. Java 9—13 Overview
2. Language
3. API
4. Tooling
5. Java 14 & beyond

👍 v12+ & Preview Features:
JAVA 9–13 IN A NUTSHELL
V9 (HUGE...)

- Java module (Jigsaw)
- API Improvements: Project Coin Milling, Stream, ...
- Unified JVM & Java Logging, ...
- Performance
  - Tools: jshell  jlink  -release, Multi-Release JARs

2017-03-23, 91 JEPs
V10

- **var**: Local-Variable Type Inference
- **Performance**: GC, Class-Data Sharing, Threads, ...
- **Experimental**: Graal AoT-Compiler

2018-03-20, 12 JEPs
V11 [LTS]

- New HTTP/2- & Websocket-Client
- New GCs: Low-latency ZGC, No-op Epsilon GC
- Ex-payware: Flight Recorder, ZGC, CDS, ...
  - Oracle JDK \equiv OpenJDK
- Dropped: Java EE, JavaFX, CORBA
- HTML5 Javadoc, Unicode 10, ...
- Updated Crypto: TLS 1.3, new Kex & Ciphers

2018-09-25, 17 JEPs
V12

- Low-pause GC „Shenandoah“
- Microbenchmark Suite
- Preview: Switch Expressions
- Performance improvements
  (CDS, G1 GC, Constants)

2019-03-19, 8 JEPs
V13

- Updated Preview: Switch-Expression
- Preview: Text Blocks
- Performance improvements (AppCDS, ZGC, Socket)

📅 2019-09-17, 5 JEPs
LANGUAGE
LOCAL VARIABLE TYPE INFERENCE

• New reserved Java type `var`
• "real type" inferred by compiler on first initialization
• can denote classes (Integer) & primitives (int)
• only available for local variables

```java
var primitiveVal = 5;       // int
doubleVal = 5d;            // double

final var sl = new ArrayList<String>();

var letters = "";
for (var s : sl) {
    letters += s.toLowerCase();
}

// var: enable annotation w/ type infer
(var x, @NotNull var y) -> x + y;
```
Type Inference — Bordercases

// var wont Compile; ①
// var wont Compile = null;
// var wont Compile = {-1, 1};

var myMap = new HashMap<>();
myMap.put(42, "The answer");
// var wont Compile = myMap.get(42).trim(); ②

// var wont Compile = String::toUpperCase; ③

var myPredicate = new IntPredicate() {
    public boolean test(int value) {
        return value > 0;
    }
};
// myPredicate = (int i) -> (i % 2) == 0; ④

① Type must be inferrable - no late init, null or array literals
② Type inferred at declaration time: Map<Object, Object>
③ Method references must be explicit
④ Anonymous Types are types-per-instance; therefore no reassignment
Type Inference: Style Guide

Introduction

Java SE 10 introduced type inference for local variables. Previously, all local variable declarations required an explicit (manifest) type on the left-hand side. With type inference, the explicit type can be replaced by the reserved type name var for local variable declarations that have initializers. The type of the variable is inferred from the type of the initializer.

There is a certain amount of controversy over this feature. Some welcome the conclusion it enables; others fear that it deprives readers of important type information, impairing readability. And both groups are right. It can make code more readable by eliminating redundant information, and it can also make code less readable by eliding useful information. Another group worries that it will be overused, resulting in more bad Java code being written. This is also true, but it’s also likely to result in more good Java code being written. Like all features, it must be used with judgment. There’s no blanket rule for when it should and shouldn’t be used.

Local variable declarations do not exist in isolation; the surrounding code can affect or even overwhelm the effects of using var. The goal of this document is to examine the impact that surrounding code has on var declarations, to explain some of the tradeoffs, and to provide guidelines for the effective use of var.

Principles

P1. Reading code is more important than writing code.

Code is read much more often than it is written. Further, when writing code, we usually have the whole context in our head, and take our time, when reading code,
MILLING PROJECT COIN

Try-with-resources now support „effectively final“ variables

```java
var inputStream = new FileInputStream(".gitignore");
try (inputStream) {
    ... }
```

Private methods in Interfaces

```java
interface Version {
    byte[] digits();
    default String text() {
        return text(digits());
    }
    private String text(byte[] version) {
        ... }
}
```

Allows default methods share common code
Unlock Compilation
javac --release xx --enable-preview

Unlock Execution
java --enable-preview ...

xx must exactly match used JDK version
```java
enum Direction {N, S, W, E}

String switchExpressionJDK8(Direction way) {
    String result;
    switch (way) {
        case N:
            result = "Up";
            break;
        case S:
            result = "Down";
            break;
        case E:
        case W:
            result = "Somewhere left or right";
            break;
        default:
            throw new IllegalStateException("Huh?: " + way);
    }
    return result;
}
```
String switchExpressionPreview13(Direction way) {  
    return switch (way) {  
        case N -> "Up";  
        case S -> { yield "Down"; }  
        case E, W -> "Somewhere left or right";  
        // default -> "Foo"  
    };  
}

1. `switch` can be used as expression
2. `->` instead of `: -> no break; necessary!
3. Lambdas can be used to. For *expressions* they must `yield` a value
4. `default` can be ommitted if a) no expression or b) `enum` with every value handled
TEXT BLOCKS

Java 8

```java
Object obj = engine.eval(
  "function hello() {
  \n  print("Hi, world!\n\n  
  
  \n  \n  }\n  \n  \n  \n  hello();\n  
  
  
  
}

```

Java 13

```java
Object obj = engine.eval(""
  function hello() {
    print("Hi, world!\n  
  
  
  
  }\n  
  hello();

```

- Leading space-character indentation removed up to the most-left character in the block including the closing ""
- Normalizes line-endings to \n and removes all trailing white spaces
JIGSAW

Modules bundle together one or more packages and offer stronger encapsulation than jars

- Allows **scaled-down runtime** → IoT & Container
- **Stronger:** `public` invisible to other modules
- *Sad:* No module versioning
Metadata: /module-info.java:

```java
module com.mysql.jdbc {
  requires java.sql; // Module dependency
  exports com.mysql.jdbc; // Export module package → enables access
  provides java.sql.Driver // SPI: Provide service instance
    with com.mysql.jdbc.Driver;
}
```

Jigsaw uses modules & .jmod instead of Classpath & .jar

```bash
javac -mp modulepath ...
java -mp modulepath -m modulename/moduleclass
```
It is too early to say whether modules will achieve widespread use outside of the JDK itself. In the meantime, it seems best to avoid them unless you have a compelling need.

*Effective Java: Third Edition (2018)*
— Joshua Bloch
API
IMMUTABLE COLLECTION FACTORIES

List/Set/Map.of()

/* Comment sections would break ... */
List<Integer> listOfNumbers = List.of(1, 2, 3, 4, 5/*, null*/);
Set<Integer> setOfNumbers = Set.of(1, 2, 3, 4, 5/*, 1*/);

Map<String, String> mapOfString = Map.of("key1", "value1", "key2", "value2");

Map<String, String> moreMaps = Map.ofEntries(
        Map.entry("key1", "value1"),
        Map.entry("key2", "value2"),
        Map.entry("key1", "value3")
    );

Produces immutable collections. All methods fail on null values. Set.of() and Map.of() break on duplicate (key) entries.
**Immutable Collection Copy**

List/Set/Map.copyOf()

```java
var listCopy = List.copyOf(originalList); // independent, immutable copy
var setCopy = Set.copyOf(originalList);  // copy to other types
var mapCopy = Map.copyOf(...);

assert 3 == listCopy.size();  // ["a", "b", "a"]
assert 2 == setCopy.size();   // ["a", "b"]

// listCopy.add("z") // → UnsupportedOperationException
```

Real copies – independent from source vs. Collections.unmodifiable()
```java
var txtPath = Path.of("src", "test", "resources", "sample.txt");
Path newPath = Path.of("src/test/resources/newfile.txt");

String fileContent = Files.readString(txtPath);
// → "Nostrum iure ullam."

Files.writeString(newPath, "Nostrum lore.", StandardOpenOption.CREATE);

long firstMismatch = Files.mismatch(txtPath, newPath); // → 8 | JDK12
```

1. Directly create `Path`-instances
2. Hassle-free reading a file content into a string
3. ...or writing into files
4. Find position of first mismatching bytes of two Files
TRANSFORM & GENERATE STREAMS

Modify with `dropWhile() & takeWhile()`

```java
var stream = Stream.of(-1, 0, 1, 21, 42);
stream.dropWhile(i -> i < 2)       // → [ 21, 42 ]
stream.takeWhile(i -> i < 2)       // → [ -1, 0, 1 ]
```

Redirect with `transferTo()`

```java
new ByteArrayInputStream(buf)
   .transferTo(System.out);  // InputStream → OutputStream
```

Generate with `iterate() & ofNullable()`

```java
// Stream<T> iterate(seedValue, nextTest, applyFunc)
Stream.iterate(2, i -> i < 999, i -> i*i);       // → [2, 4, 16, 256]
Stream.ofNullable(null);                         // [ ]
Stream.ofNullable("Hi");                       // [ "Hi" ]
```
COLLECTORS

Dispatch a stream to two consuming Collectors with `teeing()`

```java
double rms(Stream<Integer> numStream) {
    return rms = numStream.collect(
        Collectors.teeing(
            Collectors.summingDouble(i -> i * i), // → sum
            Collectors.counting(), // → n
            (sum, n) -> Math.sqrt(sum / n) // Join into final result
        )
    );
}
```

More new Collectors goodness...

- Create immutable Stream copies with `toUnmodifiableList/set/Map()`
- Unpack nested values using `flatMap()`
- Filter out unwanted values using `filtering()`
`Predicate::not`

```java
stream.filter(s -> !s.isEmpty)
```

```java
stream.filter(Predicate.not(String::isEmpty))
```

```java
stream.filter(not(String::isEmpty))
```
New `ifPresentOrElse()`, `or()`, `stream()` and `isPresent()`

```java
var maybeInt = Optional.ofNullable( (Math.random() < 0.5) ? 42 : null );

maybeInt.ifPresentOrElse(
    (n) -> System.out.println(n),
    () -> System.out.println("Nada")
);  // ... do either this or that

// Optional empty? → Build a new one on-the-fly...
var triedAgain = maybeInt.or( () -> Optional.of(-1) );

// Provide a stream → [] or [42]
Stream<Integer> intStream = maybeInt.stream();

// Yay! — complements `isPresent()`
assert maybeInt.isPresent() == !maybeInt.isEmpty();
```
Process sleeper = new ProcessBuilder("sleep", "10s").start();
ProcessHandle sleepHandle = sleeper.toHandle();

// Perform Runnables on process exit
sleepHandle.onExit().thenRun(
    () -> out.println("\`sleep\` process exited") );
out.println(sleepHandle.info()); // → [user: ben, cmd: /bin/sleep, args: [10s], startTime:..., totalTime: ...]

// Find ...
ProcessHandle jvm = ProcessHandle.current(); // → JVM,
Stream<ProcessHandle> all = ProcessHandle.allProcesses(); // → all processes
jvm.children().forEach(out::println);

// Kill ...
sleepHandle.destroy();
ENHANCED DEPRECIATION POLICY

@Deprecated now wears a forRemoval flag

@Deprecated(since="1.2", forRemoval=true)
public final synchronized void stop(Throwable obj) {
    throw new UnsupportedOperationException();
}

jdeprscan: Scan Jars for deprecated usages

$ jdeprscan commons-math3-3.6.1.jar
Jar file commons-math3-3.6.1.jar:
class org/apache/commons/math3/fraction/BigFraction uses deprecated method java/math/BigDecimal::divide(Ljava/math/BigDecimal;II)...;
class org/apache/commons/math3/util/MathUtils uses deprecated method java/lang/Double::<init>(D)V
class org/apache/commons/math3/util/Precision uses deprecated method java/math/BigDecimal::setScale(II)Ljava/math/BigDecimal;

can filter for deprecated methods marked for removal
```
HttpClient client = HttpClient.newBuilder()
    .version(HttpClient.Version.HTTP_2)
    .connectTimeout(Duration.ofSeconds(3)).build();

URI uri = URI.create("https://www.excellent.de/");
HttpRequest req = HttpRequest.newBuilder(uri)
    .header("Useragent", "MyDemo").GET().build();

var future = client.sendAsync(req, HttpResponse.BodyHandlers.ofString())
    .thenApply(HttpResponse::body)
    .thenAccept(System.out::println);

var expensiveOperation = Math.pow(12345, 9876); // meanwhile do sth. else...
future.get(); // wait for completion of the background request
```

1. Fluent API and Builder Pattern
2. Standardized HTTP client capable of HTTP/2 and WebSocket connections!
3. Start asynchronously HTTP requests in the background
REACTIVE STREAMS

java.util.concurrent.Flow

JDK Standard of https://www.reactive-streams.org for asynchronous stream processing with non-blocking backpressure
DEPRECATION OF `finalize()`

**Motivation**

Finalizers are inherently problematic and their use can lead to performance issues, deadlocks, hangs, and other problematic behavior.
DEPRECATION OF `finalize()`

**Motivation**

Finalizers are inherently problematic and their use can lead to performance issues, deadlocks, hangs, and other problematic behavior.

No `forRemoval=true` yet — probably for long!

New way: Register `Cleaner` services for your objects. They will receive `PhantomReference` to objects which are no longer reachable.
New APIs in Java 13

Comparing 13.0a.25-open with 8.0.201-oracle.

---

**DragSourceContext**
- DragSourceContext(DragGestureEvent, Cursor, Image, Point, Transferable, DragSource...) - added
- DragSourceContext(DragSourceContextPeer, DragGestureEvent, Cursor, Image, Point,...) - removed

**DragSource**
- createDragSourceContext(DragGestureEvent, Cursor, Image, Point, Transferable, Dr... - added
- createDragSourceContext(DragSourceContextPeer, DragGestureEvent, Cursor, Image, ...) - removed

**DropTargetContext**
- activeClassName(DropTargetContextPeer) - removed
- removeNotify() - removed

**DropTarget**
- activeClassName() - removed
- activeClassName(ComponentPeer) - removed
- removeNotify() - removed
- removeNotify(ComponentPeer) - removed

---

**JDK API CHANGES**

JDK API Changes Java Almanac

---

**FocusEvent**
- FocusEvent(Component, int, boolean, Component, FocusEvent.Cause) - added

**java.awt.font**
- TextLayout.equals(Object) - removed
- TextLayout.hashCode() - removed

**NumericShaper.Range**
- MYANMAR_TAI_LAING - added
- SINDHIA - added

---

**java.awt.geom**
- Path2D.Double.trimToSize() - added
- Path2D.Float.trimToSize() - added

---

Event more detailed API comparison

Source: Java Almanac (Marc R. Hoffmann)
Willkommen zu fish, der freundlichen interaktiven Shell

ben@neptun $ ~/Development/jdk-13/bin/jshell
| Welcome to JShell -- Version 13-ea |
| For an introduction type: /help intro |

jshell> "HI"
$1 ==> "HI"

jshell> var hi = $1.toCharArray()
toLowerCase() toUpperCase() toString()

jshell> var hi = $1.toLowerCase()
hi ==> "hi"

jshell> String twice (String s) {
    ...
    return s + s;
    ...
} created method twice(String)

jshell> var hihi = twice(hi)
hihi ==> "hihi"

jshell>
$ jshell
| Welcome to JShell -- Version 13-ea
| For an introduction type: /help intro

jshell> "HI"
$1 ==> "HI"

jshell> var hi = $1.toCharArray().toLowerCase().toString()
hi ==> "hi"

jshell> String twice(String s) {
...
return s + s;
}...
| created method twice(String)

jshell> var hihi = twice(hi)
hihi ==> "hihi"

jshell> /vars
| String $1 = "HI"

jshell> /methods
| String twice(String)
jshell> /edit twice
| modified method twice(String)
jshell> /list
  1 : "HI"
  2 : var hi = $1.toLowerCase();
  4 : var hihi = twice(hi);
  5 : String twice(String s) {
          return "Double Time: " + s + s;
      }

jshell> /4
var hihi = twice(hi);
hihi ==> "Double Time: hihi"

jshell> /save twice twice.jsh
Directly execute Java source in File hello (no .java-Suffix!)

```java
#!/usr/bin/java --source 11

public class SingleClassJava {
    public static void main(String[] args) {
        System.out.println("Hello "+ args[0] + ". Executable .java - Yay!");
    }
}
```

```
$ chmod +x hello
$ ./hello world
Hello world. Executable .java – Yay!
```

or launch any single-class Java file:

```
$ java SingleClassJava.java world
```
JVM LOGGING & LOG REROUTING

Fine-grained, easy-to-configure JVM Logging

```
$ java -Xlog:os="debug,gc*=debug:stdout:pid,level,tags" SingleClassJava.java

[30911][debug][os] Initial active processor count set to 4
[30911][info ][os] Use of CLOCK_MONOTONIC is supported
[30911][debug][gc,ergo,heap ] Attempt heap expansion (allocate archive regions). Total size: ...
[30911][info ][gc,cds  ] Mark closed archive regions in map: [0x00000007bff0000, ...]
```

JVM Log rerouting

Allows applications to provide custom logger implementations to be used by platform classes.
FLIGHT RECORDER

- 📈 Production- Profiling & Monitoring
- Former Oracle payware
  Open-sourced with Java 11
- Flight recording start
  on new & running java
- Rules → Alerts

http://jdk.java.net/jmc/
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MICROSECONDS)
@State(Scope.Thread)
public class BenchmarkIntegers {
  @Param("500")
  private int size;
  private String[] strings;

  @Setup
  public void setup() {
    Random r = new Random(0);
    strings = new String[size];
    for (int i = 0; i < size; i++) {
      strings[i] = "" + (r.nextInt(10000) - 5000);
    }
  }

  @Benchmark
  public void parseInt(Blackhole bh) {
    for (String s : strings) {
      bh.consume(Integer.parseInt(s));
    }
  }
}

- Based on jmh
- Avoids JIT Falltrips
- Start via Maven Archetype
Most interesting Part of Java 9 / Jigsaw!

Minimize size with **jlink** Linker

Build optimized, slim run-time image for modular Java application with minimal JRE

List package dependencies with **jdeps**

```
jdeps app.jar
```

+ better **Container-Awareness**

Support of **Docker** CPU and memory limits.
PACKAGING

Cross compilation with `javac --release N`.
Conveniently compile for older Java with `--release`.
Replaces: `javac -source N -target N -bootclasspath rtN.jar`.

Packaging: Multi-release JAR

JAR Files can now contain multiple, version-specific versions classes.
JAVA 14 & BEYOND
PROJECT METROPOLIS

Automatic transformation of interpreters to compiler

GraalVM™

Embeddable in native or managed applications

OpenJDK
node
Oracle Database
standalone

Kotlin
Java
JS
Ruby
R
Python
Scala
GRAALVM — POLYGLOT VM

- Ahead-of time compiler (AoT)
- Generic VM for different languages
  - JVM (Java, Kotlin, Scala, ...)
  - LLVM (C, C++) → native
  - Java Script, Python, Ruby, R

- Shared Runtime → Zero overhead for language interop
- Native executables (GRAALVM on SubstrateVM)
  - Much smaller startup & memory

AoT can be used today:
java -XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI -XX:+UseJVMCICompiler
SIMPLYFING SYNTAX: AMBER

Pattern matching for `instanceof`

```java
if (obj instanceof String s) {
    return s.contains("abc")
}
```

Unnamed lambda parameters

```java
(i, _) -> String.valueOf(i);
```

Data Classes & Sealed Types

```java
record Point(int x, int y) {
}
```
VALUE TYPES: VALHALLA

JVM infrastructure for working with immutable and reference-free objects, for efficient by-value computation with non-primitive types.

Value Object: Immutable & Final

```java
value class Rectangle { int width, height; }
```

$t(L1/L2 \text{ Cache}) \ll t(\text{RAM})$: \rightarrow Similar performance to primitives

Benefits of immutables: Functional-style computations, shareability, ...

OpenJDK Valhalla
Lightweight concurrency

- Continuations
- Fibers (JVM „threads“)
- Tail-call elimination

Similar to cooperative multitasking or Kotlin’s Coroutines...
2016: "Java is doomed! It evolves so slowly! Every good idea takes ages to be available!"

2018: "Java is doomed! It evolves too quickly! Nobody will be able to keep pace!"
RESOURCES

References
- JDK API Changes Java Almanac
- VM Options Explorer for HotSpot, Graal, OpenJ9, Xing

Articles
- Java is Still Free 2.0.3
- Oracle Blog: Oracle JDK
- Getting Started with javac, Part 2 & 3

OpenJDK Suppliers
- AdoptOpenJDK
- Red Hat
- Amazon
- Corretto
- Alibaba
- Azul
- Zulu
- SAP
- IBM
- BellSoft
- jClarity

Tools
- GraalVM
- Flight Recorder/Mission Control Center (EAP)

Contact
- Twitter: @bentolor
- Email: benjamin.schmid@excellent.de

Sources
- GitHub Repo with Source Code Examples & Slide Sources

Image & Video Source
- Background image: Flowers, Relaxing at the beach, Coffee Beans, Woman studying, White keyboard, Jackhammer, Plasma, Light, Industrial Site


These slides with all links