Новинки 2024*

Или что там в Java 22

*Ни слова про Котлин 2

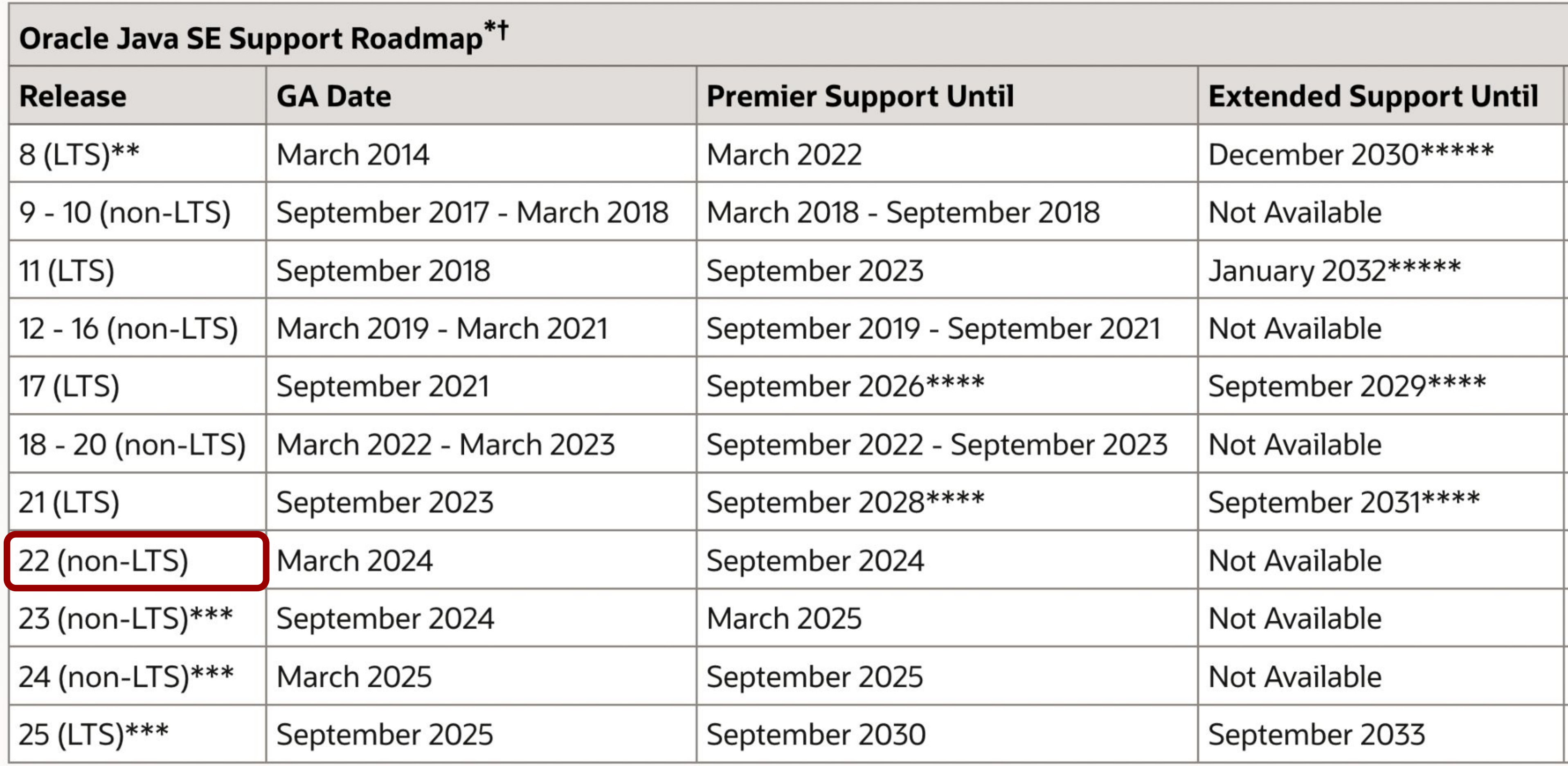
Андрей Кулешов
Positive Technologies

Владимир Воскресенский
Сбер

Андрей Зарубин
Росбанк

Java 22 - не LTS

Что это для нас значит?



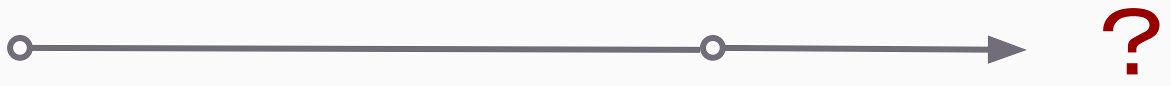
Oracle Java SE Support Roadmap*†

Release	GA Date	Premier Support Until	Extended Support Until
8 (LTS)**	March 2014	March 2022	December 2030*****
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available
11 (LTS)	September 2018	September 2023	January 2032*****
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available
17 (LTS)	September 2021	September 2026****	September 2029****
18 - 20 (non-LTS)	March 2022 - March 2023	September 2022 - September 2023	Not Available
21 (LTS)	September 2023	September 2028****	September 2031****
22 (non-LTS)	March 2024	September 2024	Not Available
23 (non-LTS)***	September 2024	March 2025	Not Available
24 (non-LTS)***	March 2025	September 2025	Not Available
25 (LTS)***	September 2025	September 2030	September 2033

Из горячего🔥

Что для вас стало самым интересным?

“Скандал” со String Templates



JEP 430:
String Templates
(Preview)
Java 21

JEP 459:
String Templates
(Second Preview)
Java 22

JEP 430 - JEP 459

String interpolation is dangerous (c)

```
C#              $"{x} plus {y} equals {x + y}"
Visual Basic    $"{x} plus {y} equals {x + y}"
Python          f"{x} plus {y} equals {x + y}"
Scala           s"$x plus $y equals ${x + y}"
Groovy          "$x plus $y equals ${x + y}"
Kotlin          "$x plus $y equals ${x + y}"
JavaScript      `${x} plus ${y} equals ${x + y}`
Ruby            "#{x} plus #{y} equals #{x + y}"
Swift           "\(x) plus \(y) equals \(x + y)"
```

“Скандал” со String Templates

JEP 430:
String Templates
(Preview)
Java 21

JEP 459:
String Templates
(Second Preview)
Java 22

JEP 430 - JEP 459

Can we do better? (c)

java.lang.StringTemplate → java.lang.StringTemplate.Processor

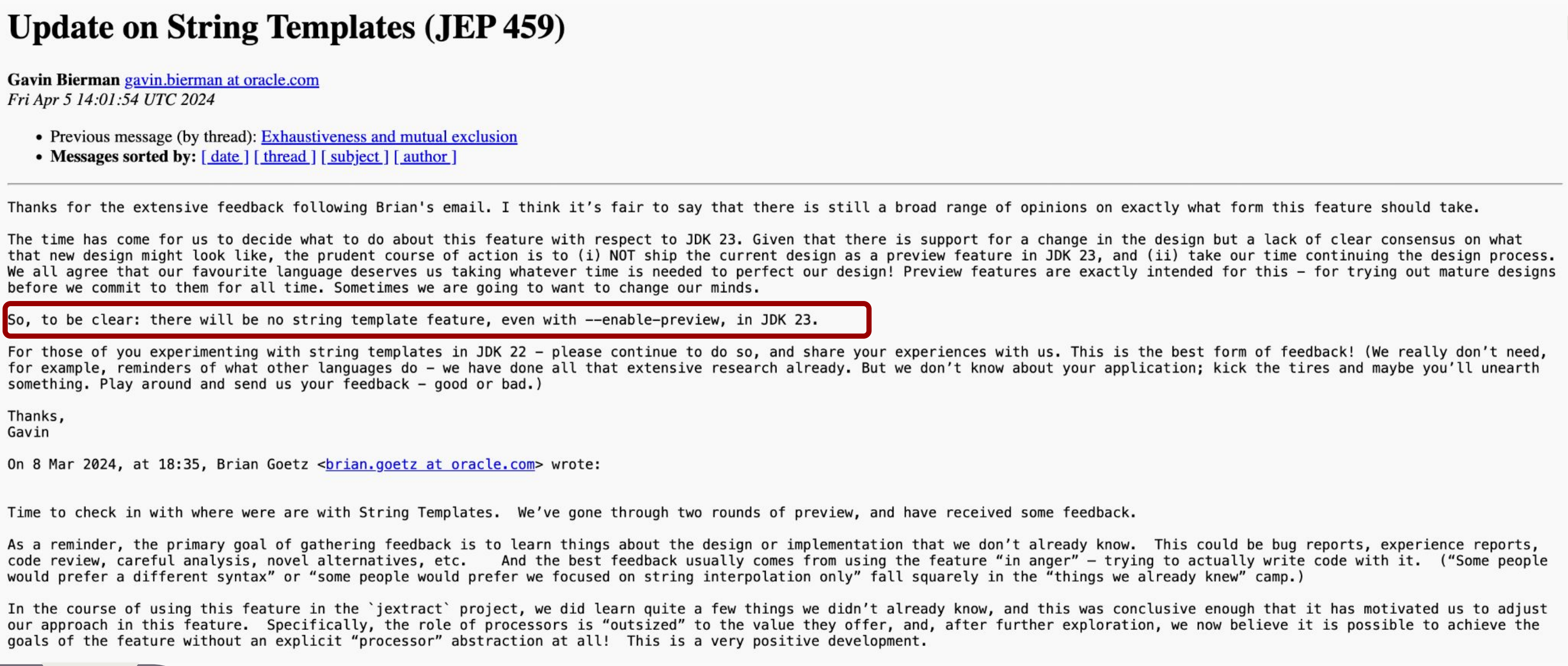
RAW template processor

FMT template processor

```
FMT."%-12s\{zone[0].name}  %7.2f\{zone[0].width}"
```

STR template processor

```
STR."\{x} + \{y} = \{x + y}";
```



Update on String Templates (JEP 459)

Gavin Bierman gavin.bierman at oracle.com
Fri Apr 5 14:01:54 UTC 2024

- Previous message (by thread): Exhaustiveness and mutual exclusion
- **Messages sorted by:** [date] [thread] [subject] [author]

Thanks for the extensive feedback following Brian's email. I think it's fair to say that there is still a broad range of opinions on exactly what form this feature should take.

The time has come for us to decide what to do about this feature with respect to JDK 23. Given that there is support for a change in the design but a lack of clear consensus on what that new design might look like, the prudent course of action is to (i) NOT ship the current design as a preview feature in JDK 23, and (ii) take our time continuing the design process. We all agree that our favourite language deserves us taking whatever time is needed to perfect our design! Preview features are exactly intended for this – for trying out mature designs before we commit to them for all time. Sometimes we are going to want to change our minds.

So, to be clear: there will be no string template feature, even with --enable-preview, in JDK 23.

For those of you experimenting with string templates in JDK 22 – please continue to do so, and share your experiences with us. This is the best form of feedback! (We really don't need, for example, reminders of what other languages do – we have done all that extensive research already. But we don't know about your application; kick the tires and maybe you'll unearth something. Play around and send us your feedback – good or bad.)

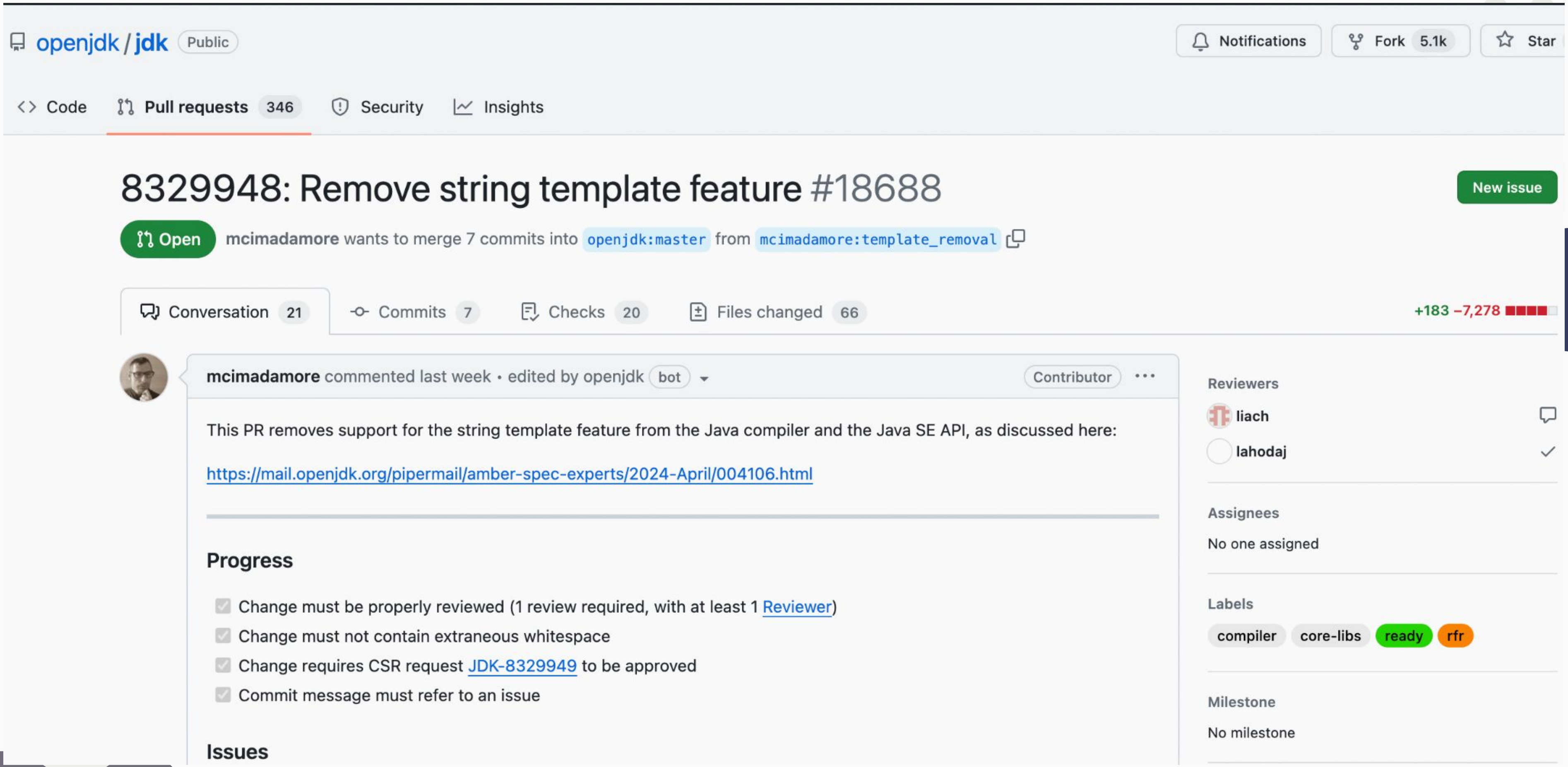
Thanks,
Gavin

On 8 Mar 2024, at 18:35, Brian Goetz <brian.goetz at oracle.com> wrote:

Time to check in with where were are with String Templates. We've gone through two rounds of preview, and have received some feedback.

As a reminder, the primary goal of gathering feedback is to learn things about the design or implementation that we don't already know. This could be bug reports, experience reports, code review, careful analysis, novel alternatives, etc.   And the best feedback usually comes from using the feature "in anger" – trying to actually write code with it. ("Some people would prefer a different syntax" or "some people would prefer we focused on string interpolation only" fall squarely in the "things we already knew" camp.)

In the course of using this feature in the `jextract` project, we did learn quite a few things we didn't already know, and this was conclusive enough that it has motivated us to adjust our approach in this feature. Specifically, the role of processors is "outsized" to the value they offer, and, after further exploration, we now believe it is possible to achieve the goals of the feature without an explicit "processor" abstraction at all! This is a very positive development.



openjdk / jdk Public

Notifications Fork 5.1k Star

Code Pull requests 346 Security Insights

8329948: Remove string template feature #18688

New issue

Open mcimadamore wants to merge 7 commits into `openjdk:master` from `mcimadamore:template_removal`

Conversation 21 Commits 7 Checks 20 Files changed 66 +183 −7,278

mcimadamore commented last week • edited by openjdk bot Contributor

This PR removes support for the string template feature from the Java compiler and the Java SE API, as discussed here:

https://mail.openjdk.org/pipermail/amber-spec-experts/2024-April/004106.html

Progress

- [x] Change must be properly reviewed (1 review required, with at least 1 Reviewer)
- [x] Change must not contain extraneous whitespace
- [x] Change requires CSR request JDK-8329949 to be approved
- [x] Commit message must refer to an issue

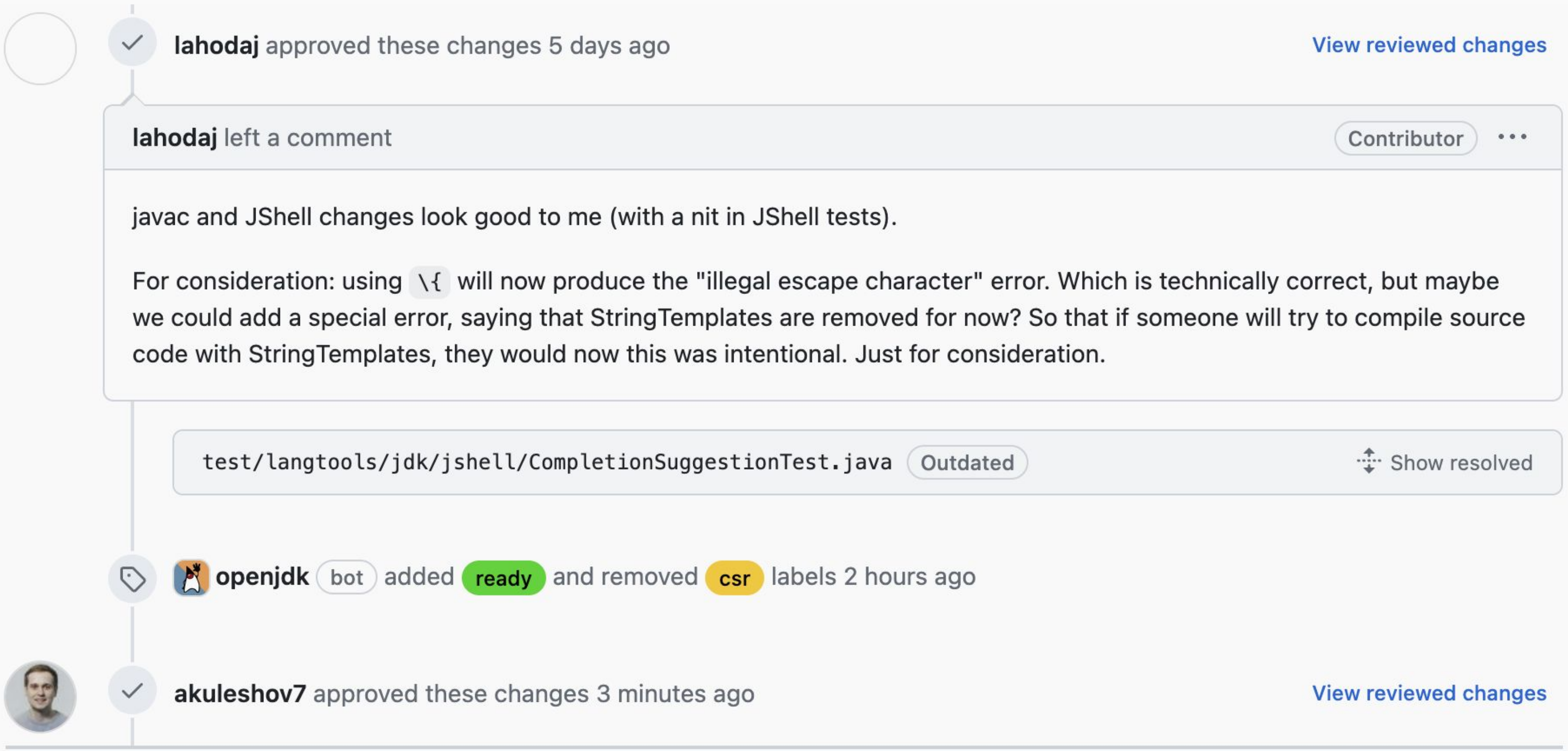
Issues

Reviewers
liach
lahodaj ✓

Assignees
No one assigned

Labels
compiler core-libs ready rfr

Milestone
No milestone



lahodaj approved these changes 5 days ago

View reviewed changes

lahodaj left a comment

Contributor

javac and JShell changes look good to me (with a nit in JShell tests).

For consideration: using `\{` will now produce the "illegal escape character" error. Which is technically correct, but maybe we could add a special error, saying that StringTemplates are removed for now? So that if someone will try to compile source code with StringTemplates, they would now this was intentional. Just for consideration.

`test/langtools/jdk/jshell/CompletionSuggestionTest.java` Outdated

Show resolved

openjdk bot added ready and removed csr labels 2 hours ago

akuleshov7 approved these changes 3 minutes ago

View reviewed changes

JEP 458

Launch Multi-File Source-Code Programs

```
root
├─ lib
│    └─ audience.jar
├─ Main.java
└─ CallFromMain.java
```

```
$ java -cp "lib/*" Main.java
```

✅

```
root
├─ lib
│   └─ audience.jar
├─ Main.java
└─ CallFromMain.java
```

```
$ java -cp "lib/*" Main.java
```

✅

Вот так вот не надо

JEP 456

Unnamed Variables & Patterns

Уже было в Java 21: preview

```
for (Person _ : persons) ++count;
```

Уже было в Java 21: preview

```java
try {
    ...
} catch (IllegalArgumentException _) {
    System.out.println("Ooops");
}
```

Уже было в Java 21: preview

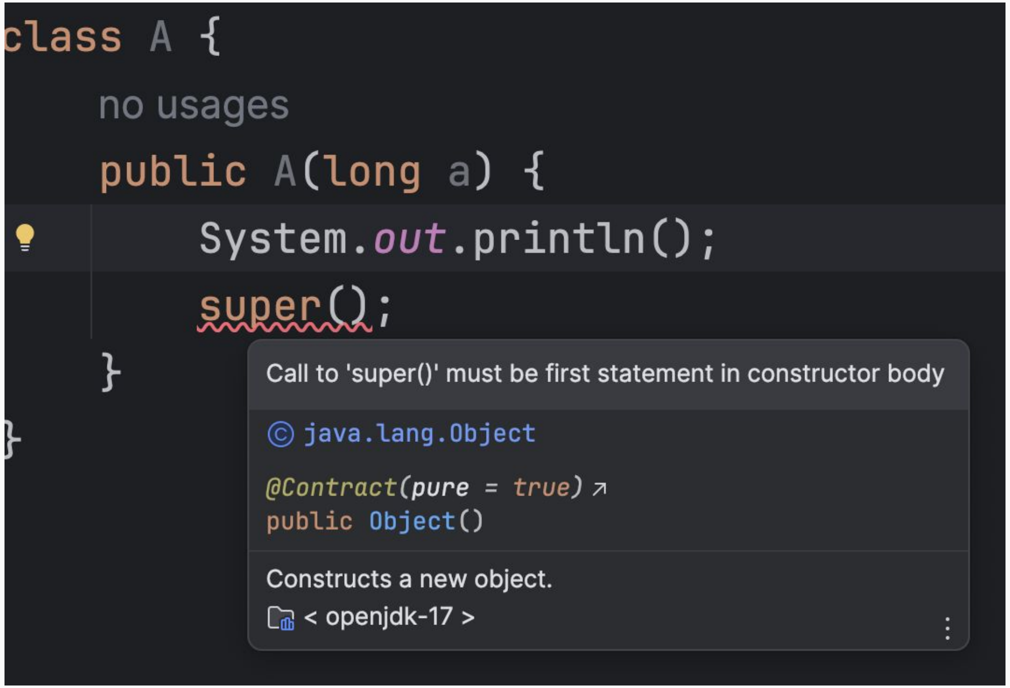
```
switch (myObject) {
   case Obj(Type1 _), Obj(Type2 _) → foo();
   case Obj(Type3 _)               → bar();
   case Obj(_)                     → other();
}
```

JEP 447

Statements before super(...)

(Preview)

Теперь можно писать код в конструкторе перед явным вызовом super() или this()



```
class A {
    no usages
    public A(long a) {
        System.out.println();
        super();
    }
}
```

Call to 'super()' must be first statement in constructor body

```
java.lang.Object

@Contract(pure = true)
public Object()
```

Constructs a new object.

< openjdk-17 >

JEP 463

Implicitly Declared Classes and
Instance Main Methods
(Second Preview)

Ушли на второе preview из Java 21 с изменениями. Main можно повсюду!

```java
class HelloWorld {
    void main() {
        System.out.println("Hello, World!");
    }
}
```

```java
void main() {
    System.out.println("Hello, World!");
}
```

Отличия от Java 21:

Unnamed class has proven **to be a distraction**. We have adopted a simpler approach: A source file without an enclosing class declaration is said to implicitly declare a class with a name chosen by the host system. Such implicitly declared classes **behave like normal top-level classes** and require no additional tooling, library, or runtime support.

If there is a candidate main method with a String[] parameter then we invoke that method; otherwise we invoke a candidate main method with no parameters. There is no ambiguity here because a class cannot declare a static method and an instance method of the same name and signature.

JEP 462

Structured Concurrency

(Second Preview)

Второй раунд после Java 21 без изменений

StructuredTaskScope

fork() join()

```
try (var scope = new StructuredTaskScope.ShutdownOnFailure()) {
   Supplier<T1> t1 = scope.fork(() -> doSmth1());
   Supplier<T2> t2 = scope.fork(() -> doSmth2());

   scope.join()
           .throwIfFailed();
}
```

JEP 464

Scoped Values

(Second Preview)

И снова второй раунд после Java 21 без изменений

$$\begin{tikzcd}\texttt{ScopedValue} \arrow[r, leftrightarrow, "?"] & \texttt{ThreadLocal}\end{tikzcd}$$

~~set()~~
where()
run()

set()
get()
remove()

JEP 460

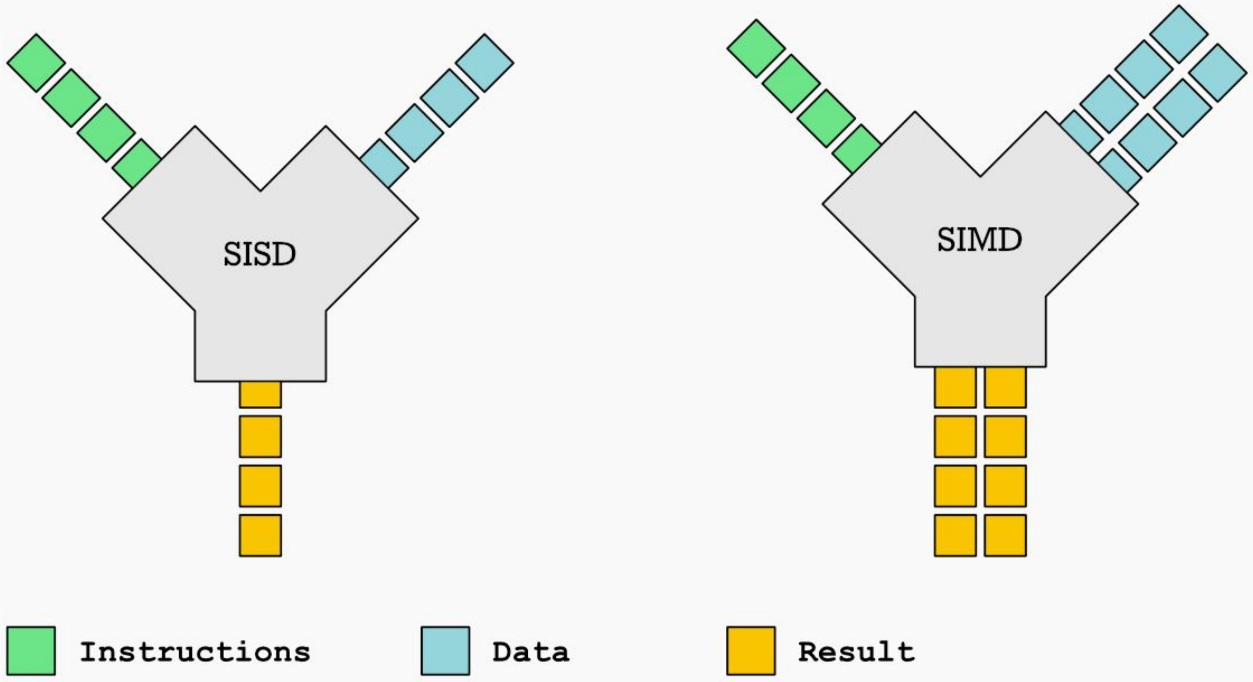
Vector API

(**SEVENTH** Preview)

Project Panama: Interconnecting JVM and native code

Foreign Function & Memory API: JEP-424

Vector API: JEP-426


SISD
SIMD
Instructions
Data
Result

Чего ждем? Ожидаем Valhalla и value classes

```
abstract class Vector<E>
```

```
void scalarComputation(float[] a, float[] b, float[] c) {
   for (int i = 0; i < a.length; i++) {
       c[i] = (a[i] * a[i] + b[i] * b[i]) * -1.0f;
   }
}
```

```
void vectorComputation(float[] a, float[] b, float[] c) {
   for (int i = 0; i < a.length; i += SPECIES.length()) {
       // VectorMask<Float>  m;
       var m = SPECIES.indexInRange(i, a.length);
       // FloatVector va, vb, vc;
       var va = FloatVector.fromArray(SPECIES, a, i, m);
       var vb = FloatVector.fromArray(SPECIES, b, i, m);
       var vc = va.mul(va)
                  .add(vb.mul(vb))
                  .neg();
       vc.intoArray(c, i, m);
   }
}
```

```
0.43%  / │  0x000000113d43890: vmovdqu 0x10(%r8,%rbx,4),%ymm0
  7.38%  │ │  0x000000113d43897: vmovdqu 0x10(%r10,%rbx,4),%ymm1
  8.70%  │ │  0x000000113d4389e: vmulps %ymm0,%ymm0,%ymm0
  5.60%  │ │  0x000000113d438a2: vmulps %ymm1,%ymm1,%ymm1
 13.16%  │ │  0x000000113d438a6: vaddps %ymm0,%ymm1,%ymm0
 21.86%  │ │  0x000000113d438aa: vxorps
-0x7ad76b2(%rip),%ymm0,%ymm0
  7.66%  │ │  0x000000113d438b2: vmovdqu %ymm0,0x10(%r9,%rbx,4)
 26.20%  │ │  0x000000113d438b9: add    $0x8,%ebx
  6.44%  │ │  0x000000113d438bc: cmp    %r11d,%ebx
           \ │  0x000000113d438bf: jl     0x000000113d43890
```

JEP 454

Foreign Function & Memory API

Project Panama: Interconnecting JVM and native code

Foreign Function & Memory API: JEP-424

Vector API: JEP-426

Стабильно…

Хочется интероп через FFM API вместо JNI

- Работа с внешней памятью: `MemorySegment`, `MemoryAddress`, `SegmentAllocator`
- Управление памятью: `MemoryLayout`, `MemoryHandles`, `MemoryAccess`
- УправлениЕ жизненным циклом ресурсов `ResourceScope`
- Вызов внешних функций `SymbolLookup`, `CLinker`

Стабильно...

Хочется интероп через FFM API вместо JNI

- Работа с внешней памятью: MemorySegment, MemoryAddress, SegmentAllocator
- Управление памятью: MemoryLayout, MemoryHandles, MemoryAccess
- УправлениЕ жизненным циклом ресурсов ResourceScope
- Вызов внешних функций SymbolLookup, CLinker

```
size_t strlen(const char *s);
```

```
MethodHandle strlen = CLinker.getInstance().downcallHandle(
        CLinker.systemLookup().lookup("strlen").get(),
        MethodType.methodType(long.class, MemoryAddress.class),
        FunctionDescriptor.of(C_LONG, C_POINTER)
);

MemorySegment str = CLinker.toCString("Hello", newImplicitScope());
long len = strlen.invokeExact(str.address());  // 5
```

JEP 461

Stream Gatherers (Preview)

Новые конструкции, к которым мы привыкли в других языках

- **fold**
- mapConcurrent
- scan
- windowFixed
- windowSliding

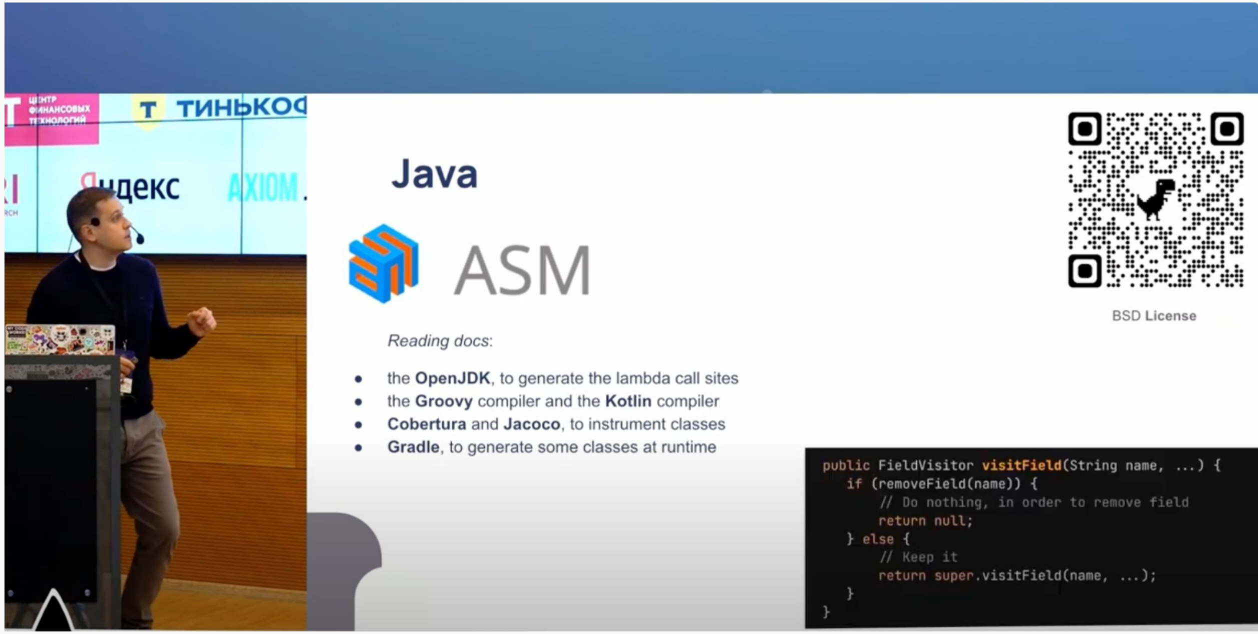
```
Stream::gather(Gatherer)
```

JEP 457

Class-File API (Preview)

Убийца ASM и SOOT

Lean into the language — In **2002**, the **visitor approach used by ASM seemed clever**, and was surely more pleasant to use than what came before. However, the Java programming language has **improved tremendously** since then — with the introduction of **lambdas, records, sealed classes, and pattern matching** — and the Java Platform now has a standard API for describing class-file constants (`java.lang.constant`). We can use these features to design an **API that is more flexible** and pleasant to use, less verbose, and less error-prone.



Java

ASM

BSD License

Reading docs:

- the **OpenJDK**, to generate the lambda call sites
- the **Groovy** compiler and the **Kotlin** compiler
- **Cobertura** and **Jacoco**, to instrument classes
- **Gradle**, to generate some classes at runtime

```
public FieldVisitor visitField(String name, ...) {
    if (removeField(name)) {
        // Do nothing, in order to remove field
        return null;
    } else {
        // Keep it
        return super.visitField(name, ...);
    }
}
```

SnowOne 2024

JEP 423

Region Pinning for G1

Summary

Reduce latency by implementing region pinning in G1, so that garbage collection need not be disabled during Java Native Interface (JNI) critical regions.

Goals

- No stalling of threads due to JNI critical regions.
- No additional latency to start a garbage collection due to JNI critical regions.
- No regressions in GC pause times when no JNI critical regions are active.
- Minimal regressions in GC pause times when JNI critical regions are active.

Общая дискуссия и вопросы

@ZARANDR