ORACLE

Deprecation Overview

Stuart W. Marks aka “Dr Deprecator”
Project Lead, JDK Core Libraries
August 10, 2021

Deprecation: What and Why?

- Deprecation is a late stage in the life cycle of a feature
  - at some point a feature might become old, obsolescent, and disused
  - if features were added and not removed, the size and complexity of a system would grow without bound
- Old, obsolete features don't just sit there, bothering no one
  - they impose costs on the ongoing development and maintenance of a system
  - more code, docs, tests to be kept up to date
  - time to build, run tests, diagnose test failures, fix and maintain tests
  - surface area for security vulnerabilities
- Purposes of deprecation
  - notify developers of a future change (removal) and alert need for migration
  - collect feedback from community about migration, alternatives, etc.
- Deprecation can, but does not necessarily imply that the feature will be removed at some point

History of Deprecation in Java SE

- JDK 1.1 – 1997
  - javadoc tag `@deprecated` (“little-d deprecated”)
  - unusual: javac parses the contents of a *comment* and emits a classfile attribute
  - classfile attribute used for issuing warnings to consumers of the class
  - an early form of annotation
- Java SE 5.0 – 2004
  - annotations feature added to the Java language, along with several defined annotations
  - `the` `@Deprecated` annotation (“big-D deprecated”)
  - annotation written to class file, available at runtime via reflection
  - some overlap with javadoc’s `@deprecated` tag, but they serve distinct purposes

```java
/**
 * The {@code Compiler} class is provided to support Java-to-native-code
 * compilers and related services. By design, the {@code Compiler} class does
 * nothing; it serves as a placeholder for a JIT compiler implementation.
 * If no compiler is available, these methods do nothing.
 *
 * @deprecated JIT compilers and their technologies vary too widely to
 * be controlled effectively by a standardized interface. As such, many
 * JIT compiler implementations ignore this interface, and are instead
 * controllable by implementation-specific mechanisms such as command-line
 * options. This class is subject to removal in a future version of Java SE.
 *
 * @author  Frank Yellin
 * @since   1.0
 */
@Deprecated(since="9", forRemoval=true)
public final class Compiler  {
```

History of Deprecation in Java SE

- Unclear, contradictory definition and usage in Java SE
  - deprecation is for dangerous features that should not be used (Thread.suspend)
  - deprecation is for API cleanup (AWT Component hide/show/setVisible)
  - there was no official documents, clarification, or discussion on this topic
- Nothing much happened for about 13 years…
- This resulted in confusion in the Java developer community
  - “Don’t use that API; it’s deprecated so it might be removed in the future.”
  - “Don’t worry, (Sun|Oracle) has *never* removed anything from Java and never will.”
- Problem: lack of clarity over how developers should respond when encountering a deprecated API

JEP 277 – Enhanced Deprecation

- Delivered in Java 9 – 2017
  - authored by your speaker
  - added two annotation methods (attributes)

```
@Deprecated(forRemoval=true, since="9")
```

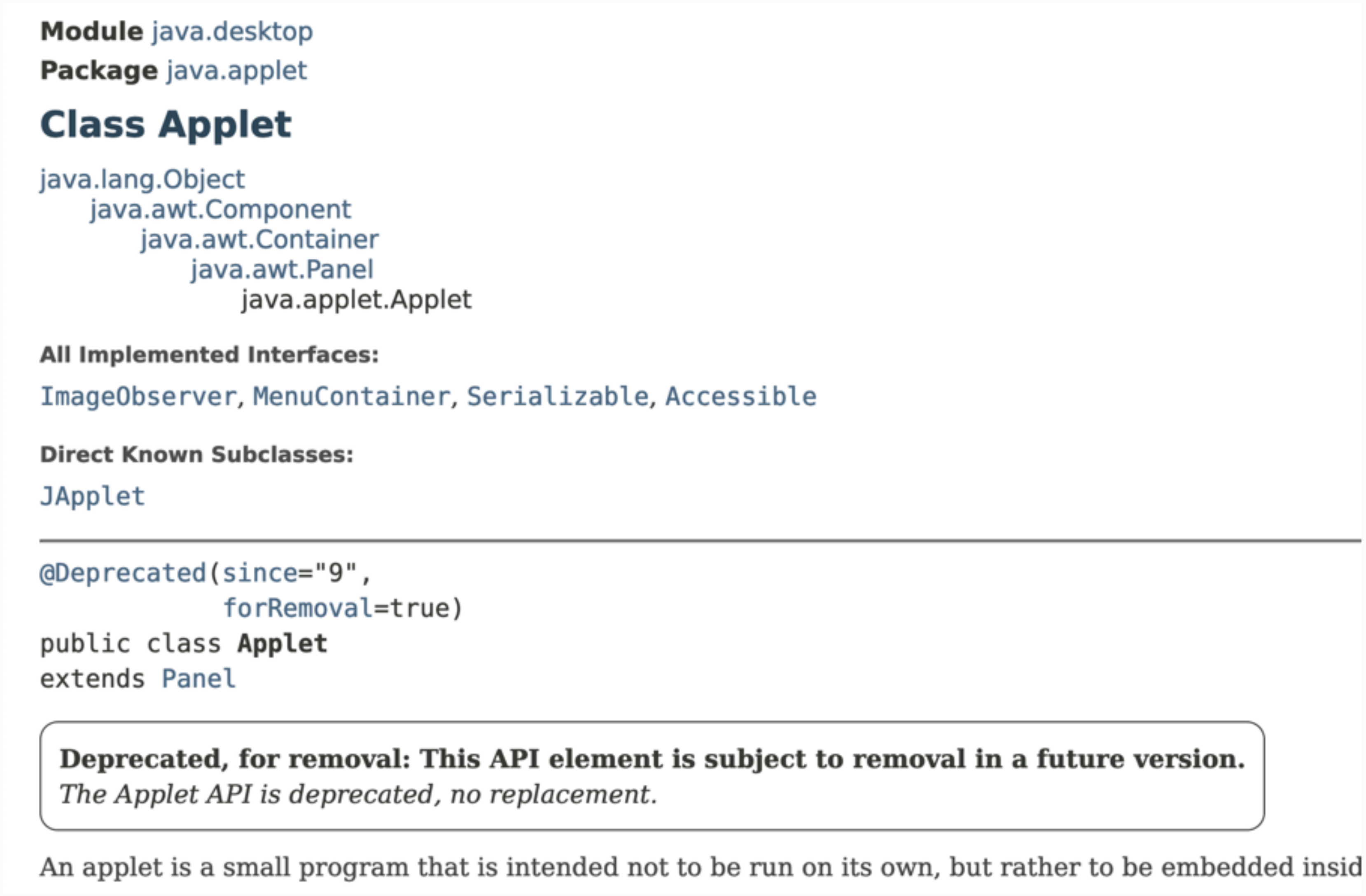
“Yes, we really intend to remove this.”

just informational

- Most work in JEP 277 is conceptual
  - filled in holes, clarified, strengthened existing concepts
  - added a new JLS mandatory warning: *removal warning*
  - vocabulary: “ordinary” versus “terminal” deprecation
  - message: certain things indeed will be removed from Java SE

Effects of Deprecation

- Compile-time warning
  - ordinary deprecations generate “deprecation” warnings
  - terminal deprecations generate “removal” warnings (new in Java 9)
  - warnings can be controlled from the javac command line or with the @SuppressWarnings annotation at the point of use
- Highlighted text emitted in javadoc
- API maintainers should provide additional information in the @deprecated javadoc tag
  - replacements (if any); rationale; references
- Annotation recorded in class file
  - available for static analysis (jdeprscan tool)
  - also available at runtime, via reflection



Module java.desktop
Package java.applet

Class Applet

```
java.lang.Object
    java.awt.Component
        java.awt.Container
            java.awt.Panel
                java.applet.Applet
```

All Implemented Interfaces:

`ImageObserver`, `MenuContainer`, `Serializable`, `Accessible`

Direct Known Subclasses:

`JApplet`

```
@Deprecated(since="9",
            forRemoval=true)
public class Applet
extends Panel
```

Deprecated, for removal: This API element is subject to removal in a future version.
The Applet API is deprecated, no replacement.

An applet is a small program that is intended not to be run on its own, but rather to be embedded insid

Deprecation is a Java SE Specification Change

- Annotations are part of class, method, field declarations
  - they are as much a part of the specification as class name, method name, parameter types, return types, etc.
  - applies to addition or removal of the annotation
  - applies to a change in an attribute (e.g., change forRemoval from false to true)
- Deprecations go through the same process as other APIs
  - specification changes are code, so they go through code review
  - all specification changes go through CSR process
  - significant changes have JEPs filed for them
- All spec changes (with or without JEPs) are fed into the Java SE JSR specification

Deprecation Policy

- Some policy covered in JEP 277
  - terminal deprecation must appear in a Java SE release *before* the API element is removed from Java SE
- Typically, we've followed an "at least one year" (two six-month Java SE releases) policy
- Small, safe things can have a shorter notice period
  - no-arg constructors of java.lang.reflect.Modifier and java.lang.invoke.ConstantBootstraps
  - CSR JDK-8230724 terminal deprecation (Java 14)
  - CSR JDK-8235548 removal (Java 15)
- More-significant things benefit from a longer notice period and JEPs
  - CMS GC terminally deprecated in Java 9 (JEP 291), removed in Java 14 (JEP 363)
  - Nashorn terminally deprecated in Java 11 (JEP 335), removed in Java 15 (JEP 372)

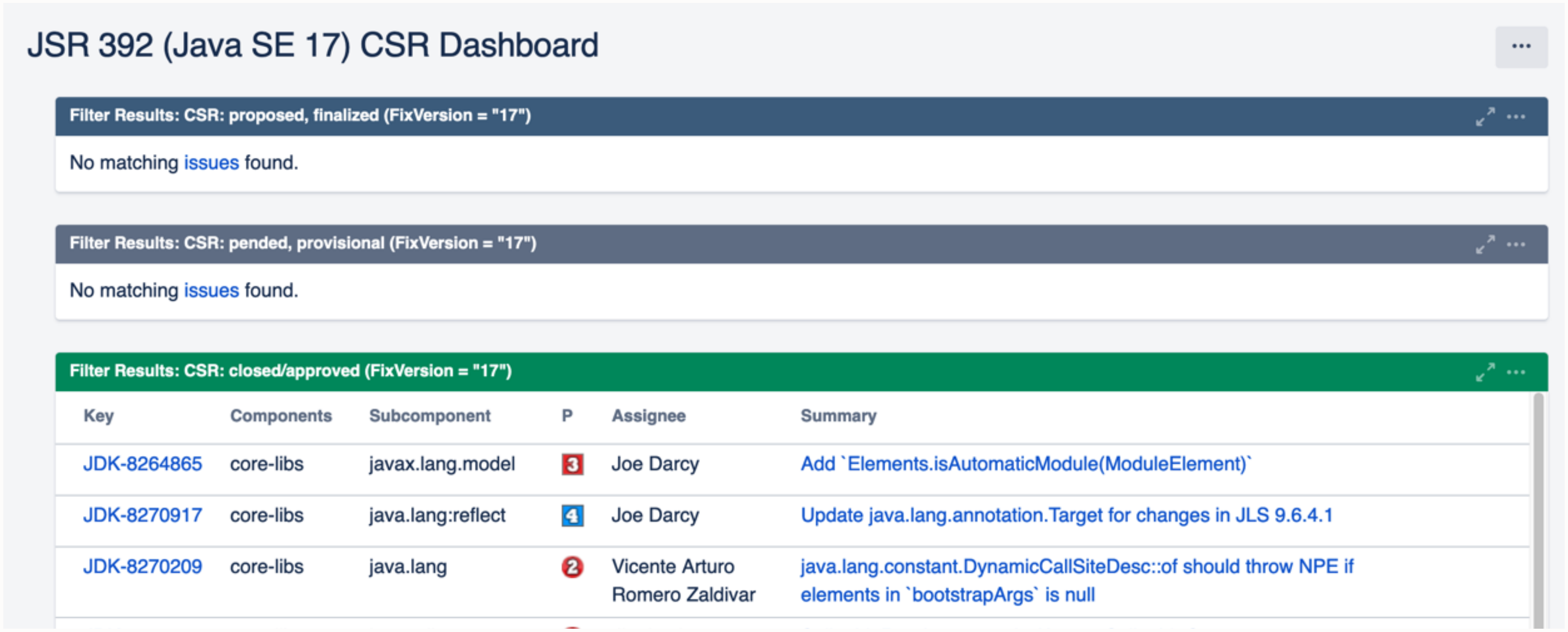
Deprecation Policy Questions

- Who decides…
  - whether a feature should be deprecated?
  - when a feature should be deprecated?
  - when a feature should be removed?
  - whether there should be some kind of replacement for the deprecated feature?
  - etc.
- Answer: the feature’s maintainer
  - most decisions are aspects of API design, which is also up to the maintainer of the feature
  - the right decision depends highly on context
  - deprecation/removal occurs infrequently across diverse features of different granularity
  - difficult to generalize any policies
  - community comments and customer feedback are a big part of deprecation decisions
  - Dr Deprecator does not make all deprecation decisions (though he is often consulted)

JCP Notification

- All specification changes tracked by CSRs
  - CSR = Compatibility and Specification Review
  - Java 17 Dashboard: https://bugs.openjdk.java.net/secure/Dashboard.jspa?selectPageId=19801
- Significant specification changes *also* have JEPs
  - JSR Expert Group + EC notified of JEP status changes
  - JEP 411 (Security Manager) => Candidate, April 2021 email
  - JEP 411 (Security Manager) => Proposed To Target, May 2021 email
- Published draft JSR specifications include terminal deprecations and removals
  - http://cr.openjdk.java.net/~iris/se/17/latestSpec/

CSR Dashboard


JSR 392 (Java SE 17) CSR Dashboard

Filter Results: CSR: proposed, finalized (FixVersion = "17")

No matching issues found.

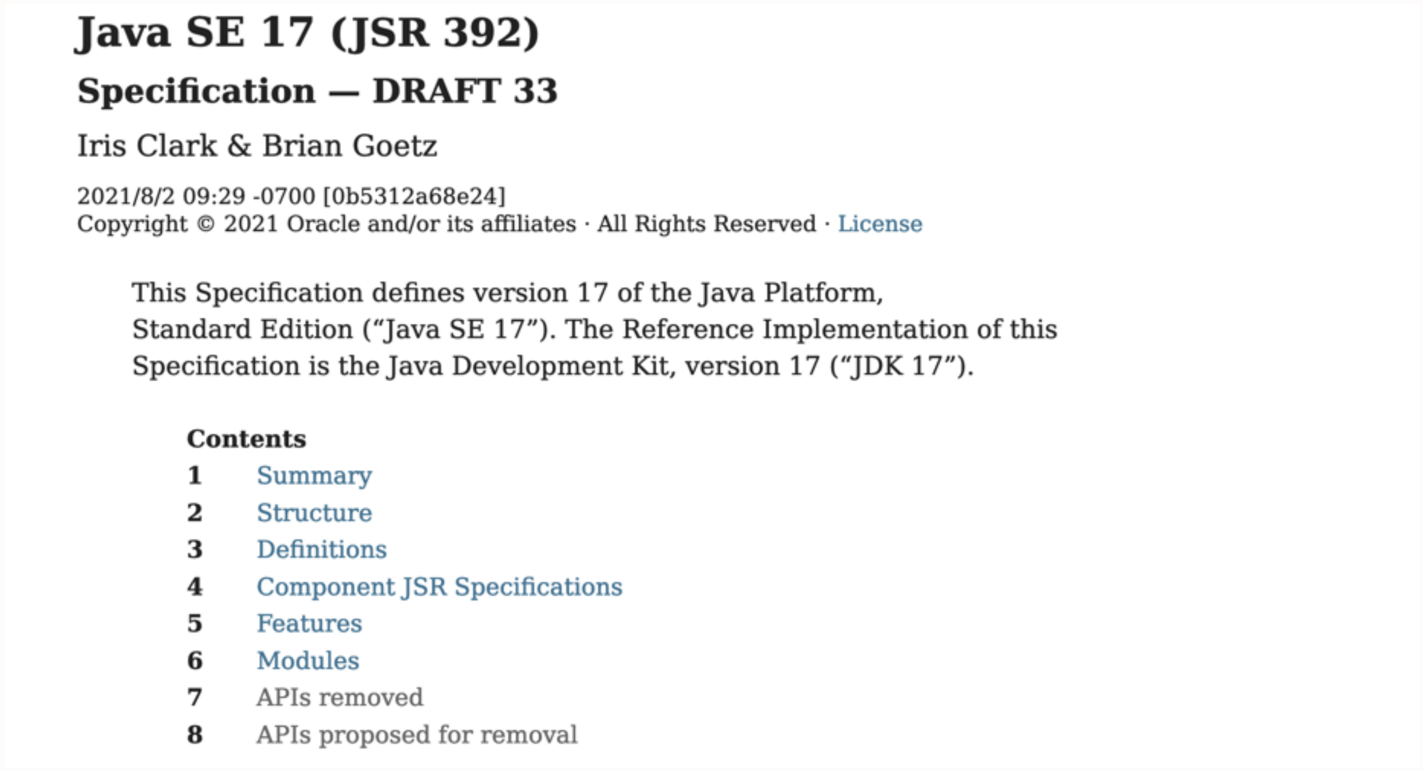
Filter Results: CSR: pended, provisional (FixVersion = "17")

No matching issues found.

Filter Results: CSR: closed/approved (FixVersion = "17")

Key	Components	Subcomponent	P	Assignee	Summary
JDK-8264865	core-libs	javax.lang.model	3	Joe Darcy	Add `Elements.isAutomaticModule(ModuleElement)`
JDK-8270917	core-libs	java.lang:reflect	4	Joe Darcy	Update java.lang.annotation.Target for changes in JLS 9.6.4.1
JDK-8270209	core-libs	java.lang	2	Vicente Arturo Romero Zaldivar	java.lang.constant.DynamicCallSiteDesc::of should throw NPE if elements in `bootstrapArgs` is null

Java SE 17 Specification Draft



Java SE 17 (JSR 392)

Specification — DRAFT 33

Iris Clark & Brian Goetz

2021/8/2 09:29 -0700 [0b5312a68e24]

This Specification defines version 17 of the Java Platform, Standard Edition ("Java SE 17"). The Reference Implementation of this Specification is the Java Development Kit, version 17 ("JDK 17").

Contents

7 APIs removed

The following APIs were removed from the Java SE Platform by this Platform Specification. The number in parentheses indicates the Java SE Platform Specification which first proposed its removal.

Packages

java.rmi.activation (15)

Classes

java.rmi.activation.Activatable (15)
java.rmi.activation.ActivationDesc (15)
java.rmi.activation.ActivationGroup (15)
java.rmi.activation.ActivationGroup_Stub (15)
java.rmi.activation.ActivationGroupDesc (15)
java.rmi.activation.ActivationGroupID (15)
java.rmi.activation.ActivationID (15)
java.rmi.activation.ActivationInstantiator (15)
java.rmi.activation.ActivationMonitor (15)
java.rmi.activation.ActivationSystem (15)
java.rmi.activation.Activator (15)

Exceptions

java.rmi.activation.ActivateFailedException (15)
java.rmi.activation.ActivationException (15)
java.rmi.activation.UnknownGroupException (15)
java.rmi.activation.UnknownObjectException (15)

Constructors

java.net.URLDecoder.<init>() (16)

8 APIs proposed for removal

The following APIs were proposed for removal from the Java SE Platform by the Platform Specifications for Java SE 9, Java SE 10, Java SE 13, Java SE 14, and Java SE 16. They are not removed in this release of the Java SE Platform. They continue to be eligible for removal in a future release.

Classes

java.lang.Compiler (9)
java.security.Certificate (10)
java.security.Identity (10)
java.security.IdentityScope (10)
java.security.Signer (10)
javax.security.cert.Certificate (13)
javax.security.cert.X509Certificate (13)

Public Notification

- JEP publication (for significant deprecations or removals)
- mailing list announcement, CSR, code review
- Twitter
- blog posts
- podcast episodes
- conferences and user group talks
- EA builds
- direct contact with library and tool maintainers
- Quality Outreach Program
- ~~full-page ad in *New York Times*~~

Notification is never good enough, since some people are always surprised....

Conclusion

- Summary
  - keeping the platform healthy
  - new features introduced, obsolete features retired
  - most of this isn’t about technology; it’s communication with developers
- Links
  - Java 17 CSR dashboard
    - https://bugs.openjdk.java.net/secure/Dashboard.jspa?selectPageId=19801
  - JSR spec drafts, sections 7 and 8
    - http://cr.openjdk.java.net/~iris/se/17/latestSpec/
  - JEP 277 Enhanced Deprecation
    - https://openjdk.java.net/jeps/277
  - JEP index (JEP 0) – search on page for “deprecate” or “remove”
    - https://openjdk.java.net/jeps/0

Thank you!

Stuart Marks (aka Dr Deprecator)
stuart.marks@oracle.com

ORACLE