

ORACLE



JEP 500: Prepare to Make Final Mean Final

Alex Buckley

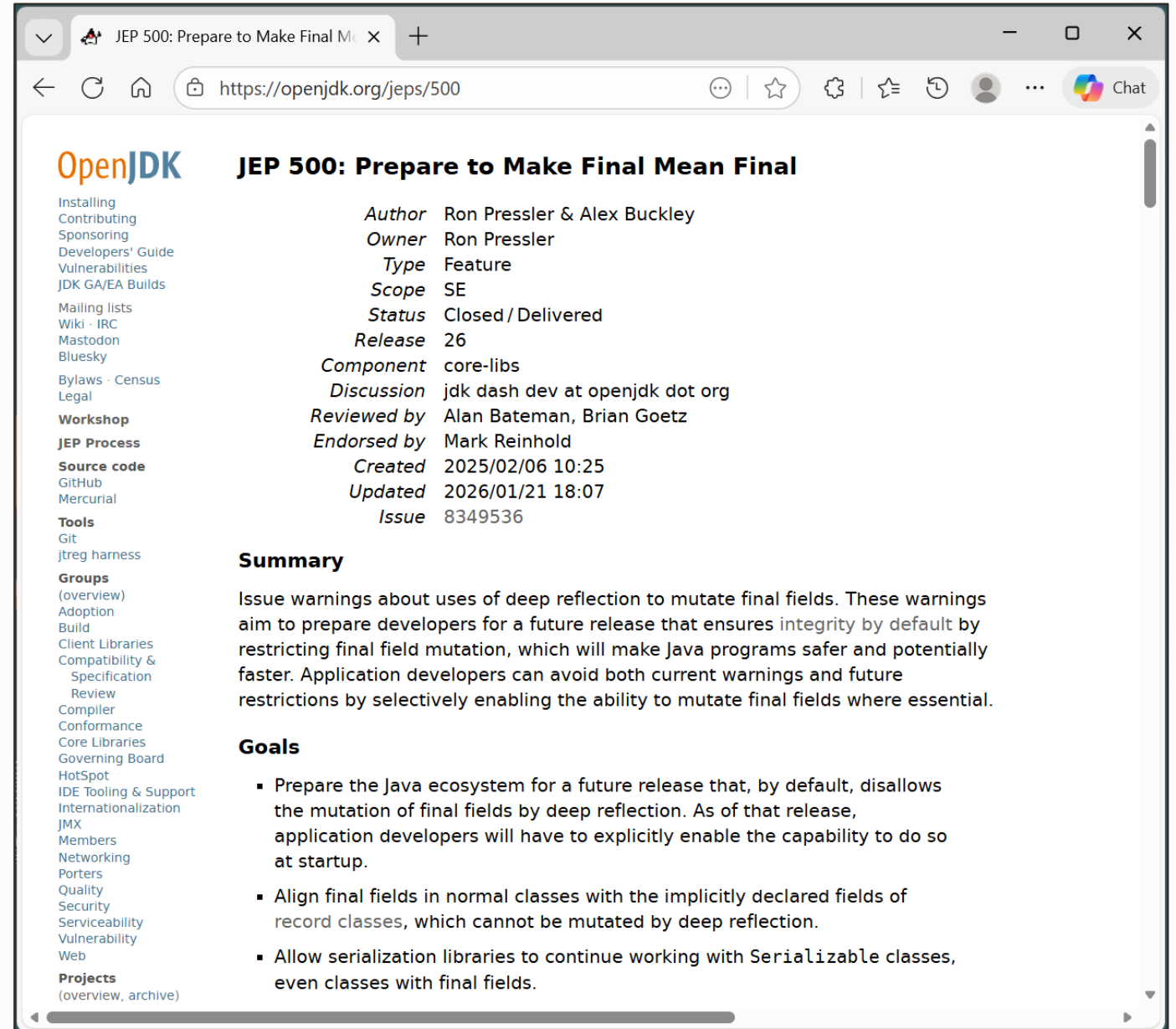
Java Platform Group

February 2026



At a high level

- Delivered in JDK 26
- A change to the Java SE Platform, not just the JDK
- Critical for *Integrity by Default*
- For compatibility reasons, the first of three JEPs about final-field mutation

A screenshot of a web browser displaying the OpenJDK JEP 500 page. The browser's address bar shows the URL "https://openjdk.org/jeps/500". The page features the OpenJDK logo on the left and a sidebar with various links. The main content area is titled "JEP 500: Prepare to Make Final Mean Final" and includes a table of metadata, a summary, and a list of goals.

OpenJDK

JEP 500: Prepare to Make Final Mean Final

Author	Ron Pressler & Alex Buckley
Owner	Ron Pressler
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	26
Component	core-libs
Discussion	jdk dash dev at openjdk dot org
Reviewed by	Alan Bateman, Brian Goetz
Endorsed by	Mark Reinhold
Created	2025/02/06 10:25
Updated	2026/01/21 18:07
Issue	8349536

Summary

Issue warnings about uses of deep reflection to mutate final fields. These warnings aim to prepare developers for a future release that ensures *integrity by default* by restricting final field mutation, which will make Java programs safer and potentially faster. Application developers can avoid both current warnings and future restrictions by selectively enabling the ability to mutate final fields where essential.

Goals

- Prepare the Java ecosystem for a future release that, by default, disallows the mutation of final fields by deep reflection. As of that release, application developers will have to explicitly enable the capability to do so at startup.
- Align final fields in normal classes with the implicitly declared fields of record classes, which cannot be mutated by deep reflection.
- Allow serialization libraries to continue working with Serializable classes, even classes with final fields.

<https://openjdk.org/jeps/500>



1. The Problem

What do you mean, “Make Final Mean Final” ?

- Final fields represent immutable state.
- The expectation that a final field cannot be reassigned in far-flung parts of the program is crucial for correctness and performance.
- Unfortunately, the expectation that a final field cannot be reassigned is **false**.
- Deep reflection strikes again!
- “Final Does Not Mean Final”
- Prevents JIT optimization for trusted fields within this run of the JVM, and across runs (Leyden)

```
// A normal class with a final field
class C {
    final int x;
    C() { x = 100; }
}

// 1. Perform deep reflection over the final field in C
java.lang.reflect.Field f = C.class.getDeclaredField("x");
f.setAccessible(true);      // Make C's final field mutable

// 2. Create an instance of C
C obj = new C();
System.out.println(obj.x);  // Prints 100

// 3. Mutate the final field in the object
f.set(obj, 200);
System.out.println(obj.x);  // Prints 200
f.set(obj, 300);
System.out.println(obj.x);  // Prints 300
```

The culprit: Serialization

It might seem absurd that the Java Platform includes an API that undermines the meaning of the `final` keyword. Final fields have played a crucial role in the [Java Memory Model](#) since JDK 5; in particular, their immutability underpins the [safe initialization of objects](#) in multi-threaded code. Unfortunately, their immutability also conflicts with the operation of serialization libraries that mutate fields to initialize objects during deserialization. This use case was sufficiently important to justify [changing the reflection API](#) in JDK 5 so that it could be used to mutate final fields.

In retrospect, offering such unconstrained functionality was a poor choice because it sacrificed integrity. When we introduced [hidden classes](#) in JDK 15 and [record classes](#) in JDK 16, we constrained deep reflection to [disallow the mutation of final fields in hidden and record classes](#). We constrained deep reflection further when we [strongly encapsulated JDK internals](#) in JDK 17. In JDK 24, we started the process to remove methods in `sun.misc.Unsafe` that, like deep reflection, allow the mutation of final fields.

2. The Solution

JEP 500: Prepare to Make Final Mean Final

Description

In JDK 5 and later releases, you can mutate final fields via deep reflection, i.e., the `setAccessible` and `set` methods of `java.lang.reflect.Field`. In JDK 26, we will restrict deep reflection so that, by default, mutating a final field causes a warning to be issued at run time. It will not be possible to avoid the warning simply by using `--add-opens` to enable the deep reflection of classes with final fields.

We refer to restrictions on mutating final fields as *final field restrictions*. We will strengthen the effect of final field restrictions over time. Rather than issue warnings, a future JDK release will, by default, throw exceptions when Java code uses deep reflection to mutate final fields. This will enable the Java Platform and applications running on it to have *integrity by default*. The warning we introduce here is intended to prepare developers for that future.

java.lang.reflect.Field::set requires *final field mutation* to be enabled

This specification is not final and is subject to change. Use is subject to [license terms](#).

OVERVIEW **CLASS** USE TREE PREVIEW NEW DEPRECATED INDEX SEARCH HELP ⚙️

Java SE 26 & JDK 26
DRAFT 26-ea+33-2879

java.base > java.lang.reflect > Field

Search documentation (type /)

set

```
public void set(Object obj,
               Object value)
    throws IllegalArgumentException,
           IllegalAccessException
```

Sets the field represented by this `Field` object on the specified object argument to the specified new value. The new value is automatically unwrapped if the underlying field has a primitive type.

The operation proceeds as follows:

If the underlying field is static, the `obj` argument is ignored; it may be null.

Otherwise the underlying field is an instance field. If the specified object argument is null, the method throws a `NullPointerException`. If the specified object argument is not an instance of the class or interface declaring the underlying field, the method throws an `IllegalArgumentException`.

If this `Field` object is enforcing Java language access control, and the underlying field is inaccessible, the method throws an `IllegalAccessException`.

If the underlying field is final, this `Field` object has *write* access if and only if all of the following conditions are true, where `D` is the field's [declaring class](#):

- `setAccessible(true)` has succeeded for this `Field` object.
- [final field mutation is enabled](#) for the caller's module.
- At least one of the following conditions holds:
 - a. `D` and the caller class are in the same module.
 - b. The field is public and `D` is public in a package that the module containing `D` exports to at least the caller's module.
 - c. `D` is in a package that is [open](#) to the caller's module.
- `D` is not a [record class](#).
- `D` is not a [hidden class](#).
- The field is non-static.

java.lang.reflect.Field::set require

Field (Java SE 26 & JDK 26 [build ...])

https://download.java.net/java/early_access/jdk26/docs/api/java.base/java/lang/reflect/Field.html#set

This specification is not final and is subject to change. Use is subject to change.

OVERVIEW CLASS USE TREE PREVIEW NEW DEPRECATED INDEX SEARCH HELP

java.base > java.lang.reflect > Field

set

```
public void set(Object obj,
               Object value)
    throws IllegalArgumentException,
           IllegalAccessException
```

Sets the field represented by this `Field` object on the specified object argument to the specified value. If the underlying field has a primitive type, the value must be a primitive value.

The operation proceeds as follows:

If the underlying field is static, the `obj` argument is ignored; it may be null.

Otherwise the underlying field is an instance field. If the specified object argument is null, the method throws `NullPointerException`. If the object argument is not an instance of the class or interface declaring the underlying field, the method throws `IllegalArgumentException`.

If this `Field` object is enforcing Java language access control, and the underlying field is inaccessible, the method throws `IllegalAccessException`.

If the underlying field is final, this `Field` object has write access if and only if all of the following conditions are met:

- `setAccessible(true)` has succeeded for this `Field` object.
- final field mutation is enabled for the caller's module.
- At least one of the following conditions holds:
 - a. `D` and the caller class are in the same module.
 - b. The field is public and `D` is public in a package that the module containing `D` exports.
 - c. `D` is in a package that is open to the caller's module.
- `D` is not a record class.
- `D` is not a hidden class.
- The field is non-static.

Java SE 26 (JSR 401)

https://cr.openjdk.org/~iris/se/26/...

Integrity of final fields

Various methods in the Java SE API provide *write access* to final fields at run time. This means that Java code can mutate the value of a final field after the field has been initialized in a constructor, an instance initializer, or the field's declaration. Only the values of final instance fields, not final static fields, can be mutated in this way.

The methods that provide write access, known as *mutation methods*, are:

- `java.lang.reflect.Field.set`
- `java.lang.reflect.Field.setBoolean`
- `java.lang.reflect.Field.setByte`
- `java.lang.reflect.Field.setChar`
- `java.lang.reflect.Field.setShort`
- `java.lang.reflect.Field.setInt`
- `java.lang.reflect.Field.setLong`
- `java.lang.reflect.Field.setFloat`
- `java.lang.reflect.Field.setDouble`
- `java.lang.invoke.MethodHandles.Lookup.unreflectSetter`

The use of mutation methods to alter the values of final fields is a legacy practice. It is strongly inadvisable because it undermines the correctness of programs written in expectation of final fields being immutable.

In Java SE 26 and later, when a mutation method is invoked, one of the preconditions for providing its caller with write access is that *final field mutation* is enabled for the caller. If final field mutation is disabled for the caller, then write access is not provided and the mutation method throws `IllegalAccessException`.

An Implementation of this Specification must:

- disable final field mutation for all code by default.
- provide a means to invoke its run-time system with final field mutation enabled for code identified to the run-time system, and disabled for all other code.

Demo: Enable final field mutation

```
C:\Users\Alex Buckley>java -version
openjdk version "26-ea" 2026-03-17
OpenJDK Runtime Environment (build 26-ea+33-2879)
OpenJDK 64-Bit Server VM (build 26-ea+33-2879, mixed mode, sharing)

C:\Users\Alex Buckley>java Test
100
WARNING: Final field x in class Test$C has been mutated reflectively by class Test in unnamed module @4e0e2f2a
(file:/C:/Users/Alex%20Buckley/)
WARNING: Use --enable-final-field-mutation=ALL-UNNAMED to avoid a warning
WARNING: Mutating final fields will be blocked in a future release unless final field mutation is enabled
200
300

C:\Users\Alex Buckley>java --enable-final-field-mutation=ALL-UNNAMED Test
100
200
300
```

3. The Impact

Recommendation for serialization libraries: `sun.reflect.ReflectionFactory`

When final field restrictions are strengthened in a future JDK release, serialization libraries will no longer be able to use deep reflection out-of-the-box. Rather than ask users to enable final field mutation on the command line, maintainers of serialization libraries should serialize and deserialize objects using the `sun.reflect.ReflectionFactory` API, which is supported for this purpose. This API allows a serialization library to obtain a `method handle` to special code that initializes an object by assigning to its instance fields directly, including final fields. This code, which is dynamically generated by the JDK, gives the serialization library the same powers as the JDK's own serialization facilities; it is not necessary to enable final field mutation by the module of the serialization library.

Recommendation for other libraries: Avoid mutating final fields altogether

Some libraries and frameworks for dependency injection, unit testing, and mocking **use deep reflection** to manipulate objects, including by mutating final fields. The maintainers of such components should only ask users to enable final field mutation as a last resort. Instead, maintainers should **find architectural approaches** that avoid the need to mutate final fields or access private fields altogether. For example, most dependency injection frameworks now forbid the injection of final fields, and all discourage it, instead recommending constructor injection.

4. The User Experience

At a high level: Final-field mutation follows the playbook for *Integrity by Default*

Most of the unsafe APIs in the Java Platform were in use for years before they were deemed unsafe, so it is not practical to restrict them without notice: applications would fail unexpectedly. In addition, configuring the Java runtime to allow the use of unsafe APIs by libraries, frameworks, and tools is not part of the traditional developer experience. To alert application developers to the need to configure the Java runtime, we restrict the use of a pre-existing unsafe API in a gradual fashion:

<https://openjdk.org/jeps/8305968#Adapting-to-restrictions-on-unsafe-APIs>

Java	Feature	Short-term default	Long-term policy
9	JEP 261 Module System	<code>--illegal-access=permit</code> (warn)	<code>--add-opens</code> <code>--add-exports</code>
24	JEP 472 Prepare to Restrict JNI	<code>--illegal-native-access=warn</code>	<code>--enable-native-access</code>
26	JEP 500 Prepare to Make Final Mean Final	<code>--illegal-final-field-mutation=warn</code>	<code>--enable-final-field-mutation</code>



JDK 26 defaults to `--illegal-final-field-mutation=warn`

```
C:\Users\Alex Buckley>java Test
100
WARNING: Final field x in class Test$C has been mutated reflectively by class Test in unnamed module @4e0e2f2a
(file:/C:/Users/Alex%20Buckley/)
WARNING: Use --enable-final-field-mutation=ALL-UNNAMED to avoid a warning
WARNING: Mutating final fields will be blocked in a future release unless final field mutation is enabled
200
300
```

```
C:\Users\Alex Buckley>java --illegal-final-field-mutation=warn Test
100
WARNING: Final field x in class Test$C has been mutated reflectively by class Test in unnamed module @4e0e2f2a
(file:/C:/Users/Alex%20Buckley/)
WARNING: Use --enable-final-field-mutation=ALL-UNNAMED to avoid a warning
WARNING: Mutating final fields will be blocked in a future release unless final field mutation is enabled
200
300
```



--illegal-
final-field-
mutation=debug

```
C:\Users\Alex Buckley>java --illegal-final-field-mutation=debug Test
100
Final field x in class Test$C has been mutated reflectively by class Test in unnamed module @c387f44 (file:/C:/Users/Alex%20Buckley/)
    at java.base/java.lang.reflect.Field.postSetFinal(Field.java:1534)
    at java.base/java.lang.reflect.Field.setFinal(Field.java:1449)
    at java.base/java.lang.reflect.Field.set(Field.java:909)
    at Test.main(Test.java:19)
200
Final field x in class Test$C has been mutated reflectively by class Test in unnamed module @c387f44 (file:/C:/Users/Alex%20Buckley/)
    at java.base/java.lang.reflect.Field.postSetFinal(Field.java:1534)
    at java.base/java.lang.reflect.Field.setFinal(Field.java:1449)
    at java.base/java.lang.reflect.Field.set(Field.java:909)
    at Test.main(Test.java:21)
300
```

--illegal-
final-field-
mutation=deny

```
C:\Users\Alex Buckley>java --illegal-final-field-mutation=deny Test
100
java.lang.IllegalAccessException: class Test (in unnamed module @4e0e2f2a) cannot set final field Test$C.x (in unnamed module @4e0e2f2a), unnamed module @4e0e2f2a is not allowed to mutate final fields
    at java.base/java.lang.reflect.Field.preSetFinal(Field.java:1504)
    at java.base/java.lang.reflect.Field.setFinal(Field.java:1447)
    at java.base/java.lang.reflect.Field.set(Field.java:909)
    at Test.main(Test.java:19)
```

--illegal-
final-field-
mutation=allow

```
C:\Users\Alex Buckley>java --illegal-final-field-mutation=allow Test
100
200
300
```

A Change for *Integrity by Default* Occurs in Three Phases

Strong Encapsulation

Phase	Java	Feature	Short-term default
1	9	JEP 261 Module System	<code>--illegal-access=permit</code> (warn)
2	16	JEP 396 Strongly Encapsulate JDK Internals by Default	<code>--illegal-access=deny</code>
3	17	JEP 403 Strongly Encapsulate JDK Internals	<code>--illegal-access</code> is ignored

Final Field Restrictions

Phase	Java	Feature	Short-term default
1	26	JEP 500 Prepare to Make Final Mean Final	<code>--illegal-final-field-mutation=warn</code>
2	YY	JEP YYY Make Final Mean Final by Default	<code>--illegal-final-field-mutation=deny</code>
3	ZZ	JEP ZZZ Make Final Mean Final	<code>--illegal-final-field-mutation</code> is ignored

A Change for *Integrity by Default*: Phase 1 of 3 (Java 26 / JEP 500)



`--illegal-final-field-mutation=allow`

`--illegal-final-field-mutation=warn`

`--illegal-final-field-mutation=debug`

`--illegal-final-field-mutation=deny`

DEFAULT

A Change for *Integrity by Default*: Phase 2 of 3 (Java YY / JEP YYY)



`--illegal-final-field-mutation=warn`



`--illegal-final-field-mutation=debug`



`--illegal-final-field-mutation=deny`

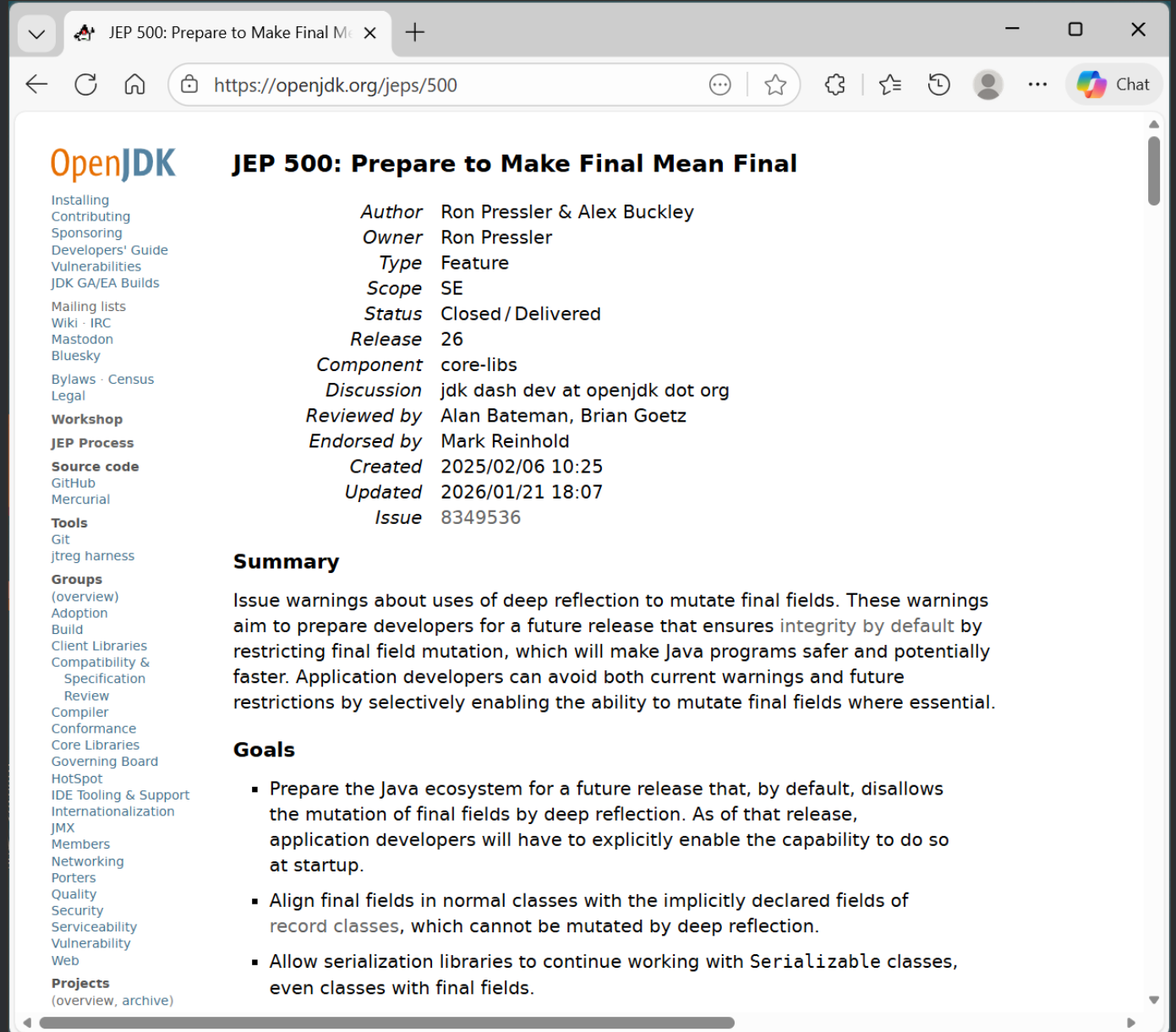
DEFAULT

A Change for *Integrity by Default*: Phase 3 of 3 (Java ZZ / JEP ZZZ)



`--illegal-final-field-mutation=...` is obsolete and ignored

Thank You



The screenshot shows a web browser window with the URL <https://openjdk.org/jeps/500>. The page title is "JEP 500: Prepare to Make Final Mean Final". The left sidebar contains a navigation menu with categories: Installing, Contributing, Sponsoring, Developers' Guide, Vulnerabilities, JDK GA/EA Builds, Mailing lists, Wiki - IRC, Mastodon, Bluesky, Bylaws - Census, Legal, Workshop, JEP Process, Source code, Tools, Groups, and Projects. The main content area displays the JEP details in a table format, followed by a Summary and Goals section.

Author	Ron Pressler & Alex Buckley
Owner	Ron Pressler
Type	Feature
Scope	SE
Status	Closed / Delivered
Release	26
Component	core-libs
Discussion	jdk dash dev at openjdk dot org
Reviewed by	Alan Bateman, Brian Goetz
Endorsed by	Mark Reinhold
Created	2025/02/06 10:25
Updated	2026/01/21 18:07
Issue	8349536

Summary

Issue warnings about uses of deep reflection to mutate final fields. These warnings aim to prepare developers for a future release that ensures integrity by default by restricting final field mutation, which will make Java programs safer and potentially faster. Application developers can avoid both current warnings and future restrictions by selectively enabling the ability to mutate final fields where essential.

Goals

- Prepare the Java ecosystem for a future release that, by default, disallows the mutation of final fields by deep reflection. As of that release, application developers will have to explicitly enable the capability to do so at startup.
- Align final fields in normal classes with the implicitly declared fields of record classes, which cannot be mutated by deep reflection.
- Allow serialization libraries to continue working with Serializable classes, even classes with final fields.