ORACLE®

JavaOne™

ORACLE®

Saving the Future From the Past

Innovations in Deprecation

Stuart Marks
aka "Dr Deprecator"
Oracle Java Platform Group

Twitter: @DrDeprecator

JavaOne ORACLE

Safe Harbor Statement

Java SE Evolution

- API life cycle: additions and removals
  - over the past 20 years, huge amounts added, vanishingly little removed
  - officially, all parts of specification have equal normative force
  - in fact, not all parts of the platform are equally valuable
  - how to remove bad ideas, correct mistakes, remove obsolete stuff?
- "Deprecation" concept introduced early
  - but there was very little follow through
- A platform that never removes anything is in an unhealthy state
- Tweet comments, questions, feedback to this hashtag: #JavaDeprecation

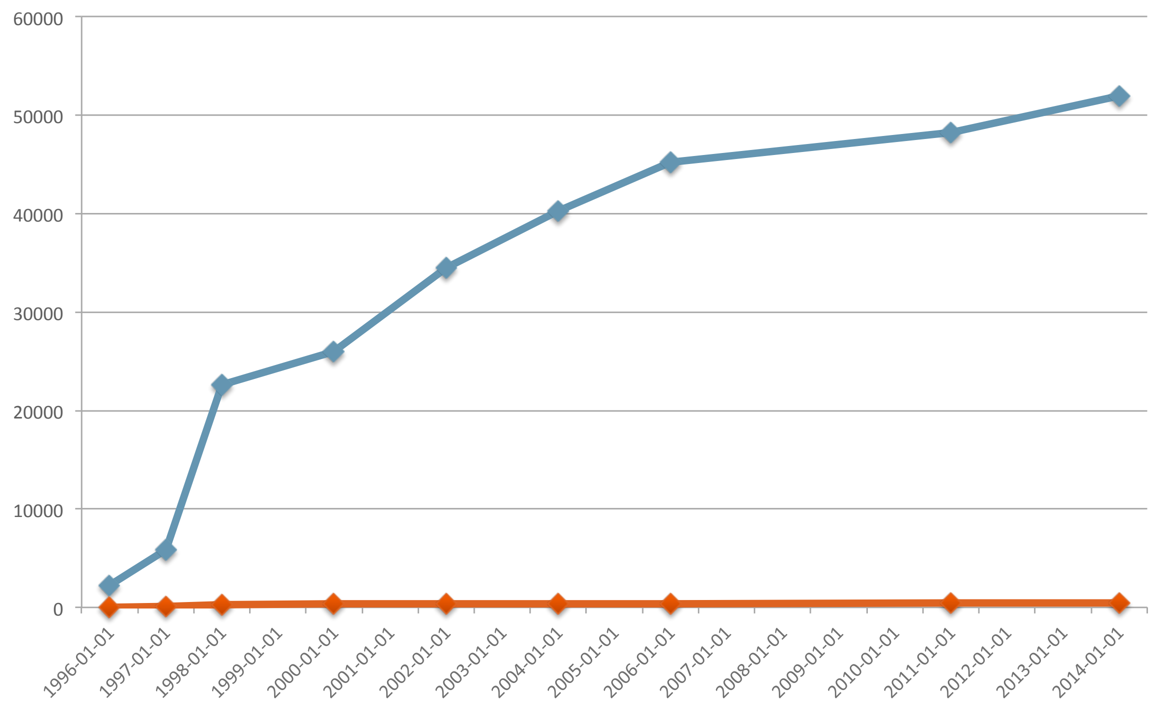
Library API Size Statistics

- Consider the “features” of an API
  - types: classes, interfaces, annotation types, exceptions
  - members: methods, fields, annotation elements
- Statistics collection based on standardized packages
  - java.**
  - javax.**
  - org.**
- Internal packages, classes, members not considered

Total APIs vs Deprecated APIs

Release	Date	# Features	# Deprecated	%
1.0	1996-01-23	2,215	0	0.00%
1.1	1997-02-19	5,859	136	2.32%
1.2	1998-12-08	22,595	269	1.19%
1.3	2000-05-08	25,967	327	1.26%
1.4	2002-02-06	34,510	349	1.01%
5	2004-09-30	40,289	390	0.97%
6	2006-12-11	45,263	409	0.90%
7	2011-07-28	48,215	421	0.87%
8	2014-03-18	51,980	440	0.85%

Total APIs vs Deprecated APIs



History of Deprecation in Java SE

- Concept introduced in JDK 1.1, October 1996
- Only a year after 1.0
- Clearly, early API designers knew there would be pressure to evolve
- New javadoc tag introduced: “@deprecated”
- Annotation type “@Deprecated” intro’d with annotations in Java SE 5
  - September 2004
  - semantics substantially the same as the @deprecated javadoc tag

What is Deprecation, Anyway?

- *Java Language Specification* sec. 9.6.4.6 covers warnings for @Deprecated
  - warnings are issued unless...
  - use is within an element that is itself deprecated, or
  - use is within an element that has @SuppressWarnings("deprecation"), or
  - use and declaration are both within same outermost class
- The java.lang.@Deprecated annotation type
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Deprecated.html

    "A program element annotated @Deprecated is one that programmers are discouraged from using, typically because it is dangerous, or because a better alternative exists. Compilers warn when a deprecated program element is used or overridden in non-deprecated code."

Deprecation Documentation

- List of Deprecated APIs
  - http://docs.oracle.com/javase/8/docs/api/deprecated-list.html
- Javadoc tool: search for "@deprecated"
  - http://docs.oracle.com/javase/8/docs/technotes/tools/unix/javadoc.html
  - mostly tells how to use the @deprecated tag in documentation
- *How and When to Deprecate APIs*
  - http://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/deprecation/deprecation.html
  - rather vague definition, but mostly describes use of the @deprecated javadoc tag

Deprecation: Original Intent

- *@deprecated: Mechanical Help for **Abandoning** Old APIs* (John Rose)
  - appeared in JDK 1.1 – 1.4 documentation bundles
  - still available via “Java Archive”
- Select quotes
  - “Java 1.1 introduces many new APIs, some of which ***supersede*** older ones.”
  - “Valid reasons for wishing one’s users to migrate to the new API include:
    - the old API is insecure, buggy, or highly inefficient
    - the old API is going away in a future release
    - the old API encourages very bad coding practices”
- Note use of the words “abandon” and “supersede”

Fundamental Meaning of Deprecation

Notification to developers that they should migrate their code away from the deprecated API.

Effects of Deprecating an API

- Compiler warnings
- IDE support
- Documentation – javadoc output

Compiler Warnings Issued by javac

```
$ javac MyClass.java
Note: /Users/smarks/MyClass.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.


$ javac -Xlint:deprecation MyClass.java
/Users/smarks/MyClass.java:19: warning: [deprecation] destroy() in Thread has
been deprecated
        thread.destroy();
```

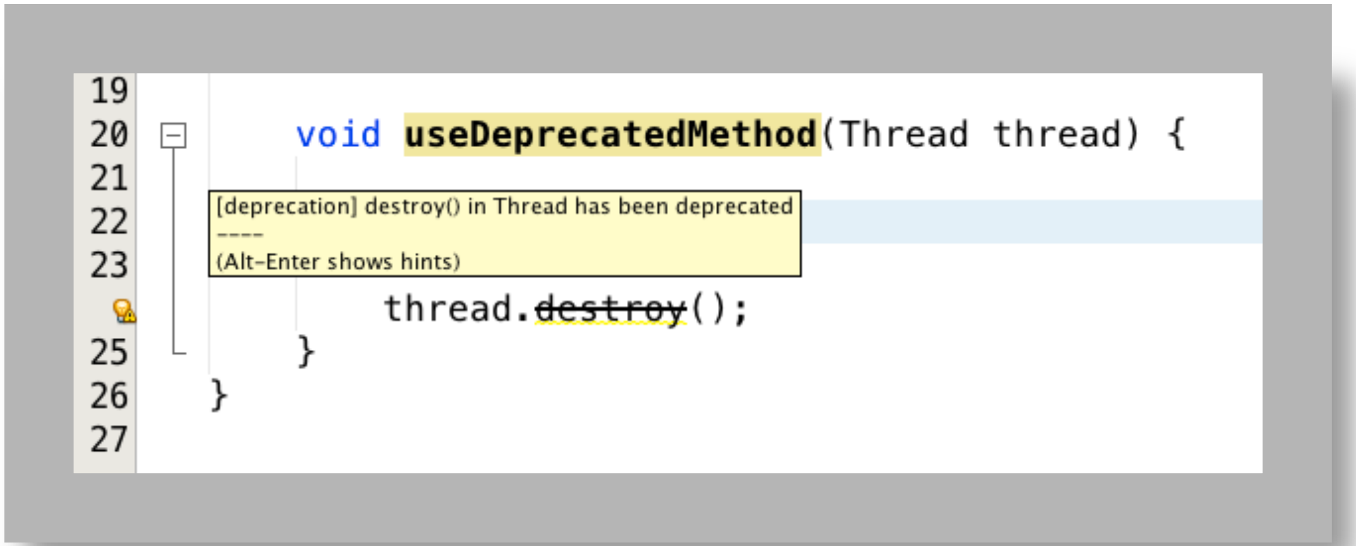
Rules for Deprecation Warnings

```
public class DeprecationTest {

    @Deprecated
    static class A {
        static A a = new A();                   // no warning; usage is within
    }                                           // the deprecated element itself

    static class B {
        A a = new A();                          // no warning; usage is within
    }                                           // the same top-level class
}
```

Rules for Deprecation Warnings

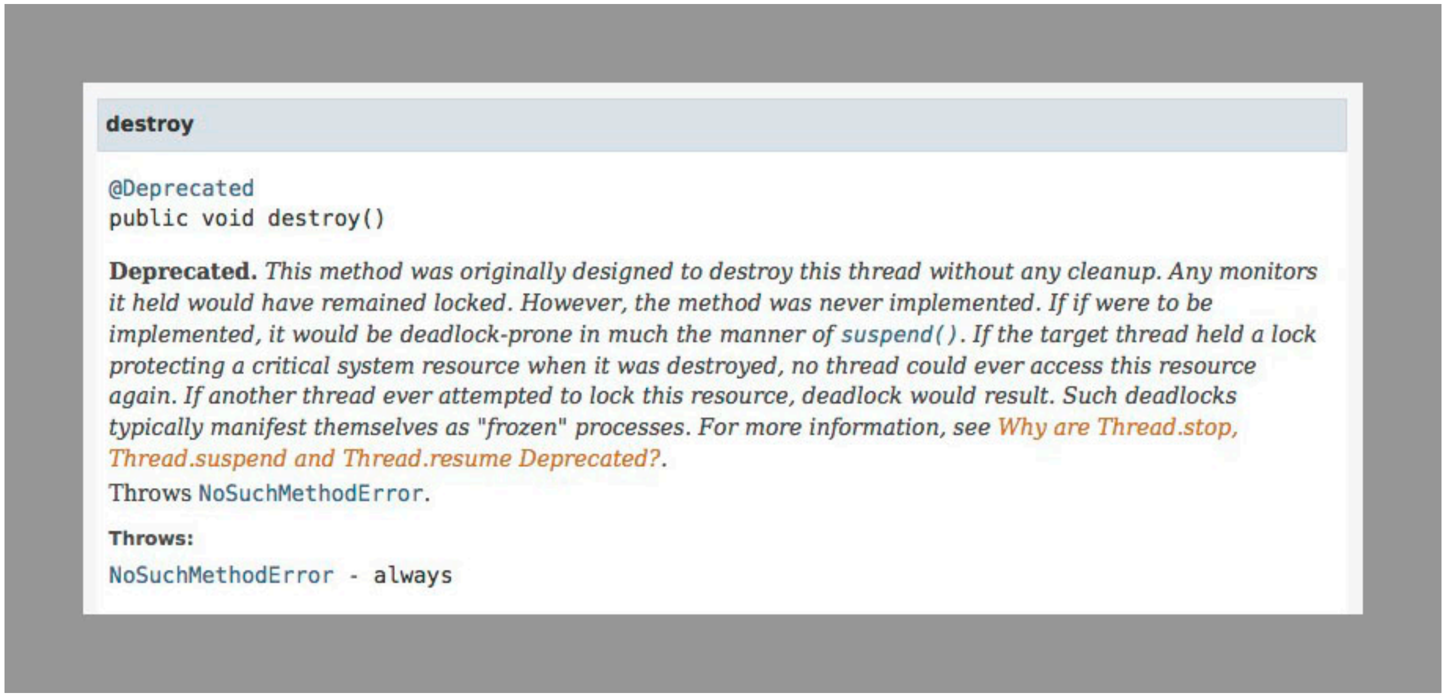
```
// top-level classes in the same or in a different file

class C {
    DeprecationTest.A a = new DeprecationTest.A();    // warning emitted here
}

@SuppressWarnings("deprecation")
class D {
    DeprecationTest.A a = new DeprecationTest.A();    // no warning, because
}                                                     // warnings suppressed
```

Deprecated's Appearance in NetBeans



```
    void useDeprecatedMethod(Thread thread) {
[deprecation] destroy() in Thread has been deprecated
----
(Alt-Enter shows hints)
        thread.~~destroy~~();
    }
}
```

Javadoc for Thread.destroy()



destroy

```
@Deprecated
public void destroy()
```

Deprecated. *This method was originally designed to destroy this thread without any cleanup. Any monitors it held would have remained locked. However, the method was never implemented. If if were to be implemented, it would be deadlock-prone in much the manner of `suspend()`. If the target thread held a lock protecting a critical system resource when it was destroyed, no thread could ever access this resource again. If another thread ever attempted to lock this resource, deadlock would result. Such deadlocks typically manifest themselves as "frozen" processes. For more information, see Why are Thread.stop, Thread.suspend and Thread.resume Deprecated?.*

Throws `NoSuchMethodError`.

Throws:

`NoSuchMethodError` - always

Ideal API Life Cycle

- Introduce new API
- API in active use
- New API introduced, old API deprecated
  - deprecation notifies developers via warnings, documentation, etc.
- Developers migrate from old API to new API
- Old API removed

Actual API Life Cycle

- Introduce new API
- API in active use
- New API introduced, old API deprecated
  - deprecation notifies developers via warnings, documentation, etc.
  - *everybody ignores this*
- Developers migrate from old API to new API
  - *nobody migrates anything*
- Old API removed
  - *virtually no APIs have been removed from the JDK (maybe JDK 9?)*

JDK Bears Some Responsibility For This

- Many things deprecated for many different reasons
  - caused lots of confusion
- No follow through for many years
  - JDK internal use of deprecated APIs not removed
  - some progress in Java 8 with warnings cleanup
  - virtually no APIs ever removed
- Consequences
  - allowed confusion to fester
  - no incentive or impetus for developers to migrate

Why Has Nothing Ever Been Removed?

- Removal represents the ultimate incompatibility
- Source incompatible
  - compilation fails, “cannot find symbol”
- Binary incompatible
  - NoSuchMethodError, NoClassDefFoundError
- Behaviorally incompatible
  - it used to work, now it doesn’t work anymore!
- Documentation difficulty
  - where do you document something that’s been removed?

Notice Currently Given Is Insufficient

- Compile-time only!
  - If you don’t recompile, you don’t see warnings
- You might see a warning in the documentation
  - But you can’t tell whether your code is impacted
  - And nothing tells you to look in the documentation either

Dr. Deprecator's Prescriptions

- Refine vocabulary of reasons for which "@Deprecated" is used today
  - recategorize or un-deprecate existing APIs as necessary
  - remove APIs judiciously after clear notification has been given
- Develop runtime warning system
- Develop tooling for static analysis
- Reorganize documentation to separate deprecated APIs
- Work with IDEs
  - dissuade developers from using deprecated APIs in the first place
  - automated refactoring to migrate away from deprecated APIs

New @Deprecated Annotation Members

- Reason enum
  - UNSPECIFIED, CONDEMNED, DANGEROUS, OBSOLETE, SUPERSEDED, UNIMPLEMENTED, EXPERIMENTAL
- Reason value(s)
- Replacement link(s)
- "Since" release tag

Deprecation Reasons

- UNSPECIFIED – the default, basically no reason
- CONDEMNED – to be removed in a future release
- DANGEROUS – may incur risk of data loss, deadlock, etc.
- OBSOLETE – retired, no longer useful
- SUPERSEDED – replaced with another API
- UNIMPLEMENTED – does nothing, don't bother calling
- EXPERIMENTAL – no long term support, removal/replacement likely soon

Deprecation Reasons

- DANGEROUS doesn’t necessarily imply CONDEMNED
  - “dangerous” implies risk
  - tradeoff of risk vs. benefit sometimes better left to caller
- CONDEMNED: when should something be removed?
  - need clearer notice of when something is to be removed
  - removal reduces customer’s flexibility making migration decisions
  - JDK should be a bit more aggressive about removing stuff
  - ... but only a bit

Deprecation Examples: String.getBytes() – Current

```
// String.java

/**
 * ...
 * @deprecated  This method does not properly convert characters into
 * bytes.  As of JDK 1.1, the preferred way to do this is via the
 * {@link #getBytes()} method, which uses the platform's default charset.
 * ...
 */
@Deprecated
public void getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin) {
    ...
}
```

Deprecation Examples: String.getBytes() – Future

```
// String.java

/**
 * ...
 * @deprecated  This method does not properly convert characters into
 * bytes.  As of JDK 1.1, the preferred way to do this is via the
 * {@link #getBytes()} method, which uses the platform's default charset.
 * ...
 */
@Deprecated(value={DANGEROUS,SUPERSEDED},
            replacement="String#getBytes()",
            since="1.1")
public void getBytes(int srcBegin, int srcEnd, byte[] dst, int dstBegin) {
    ...
}
```

More Examples

- Add @Deprecated(SUPERSEDED) to
  - Vector, Hashtable, Stack, Dictionary, Enumeration, Observer, Observable, Timer
  - Boolean() constructor
- Add @Deprecated(DANGEROUS) to
  - Thread.suspend(), Thread.resume(), Thread.stop()
- Add @Deprecated(UNIMPLEMENTED,CONDEMNED) to
  - Thread.destroy(), Thread.stop(Throwable)
- Remove @Deprecated from
  - AWT Component.show(), hide()
  - this is just a simple API rename; no need for code to migrate

Runtime Deprecation Warnings

- The @Deprecated annotation currently only warns at compile time
  - consider a binary that compiled cleanly in the past (no deprecation usage)
  - APIs were deprecated in a newer Java version
  - old binary runs on newer Java version, now uses deprecated APIs
  - no warnings!

Runtime Deprecation Warnings

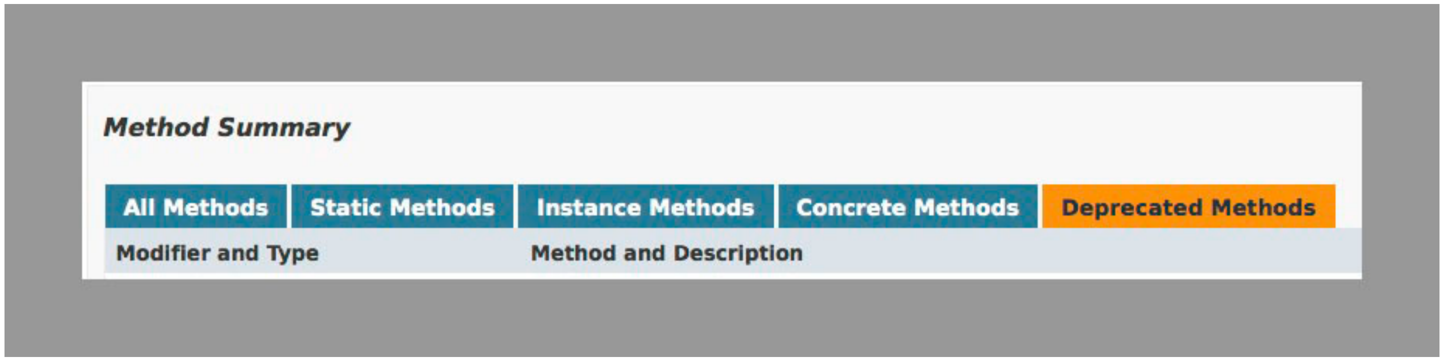
- Enable deprecation warnings at runtime
  - possible “-verbose:deprecation” command line option to enable warnings
  - issue warning upon load of a deprecated class
  - issue warning upon (first) call to deprecated method
  - sets flag checkable by library code
    - e.g., java.util.Arrays.useLegacyMergeSort
- Note: @Deprecated annotation already has RUNTIME retention
  - enables use by reflective code at runtime

Static Checking for Deprecation Usage

- A static analysis tool could do this:
  - analyze a JDK class library and find all @Deprecated APIs
  - examine a jar file (or module) for usages of such APIs
  - issue a report on deprecation usage
- Similar to “jdeps” tool for examining modular dependencies

Potential Future Javadoc Updates

- There's already a Deprecated Methods tab:



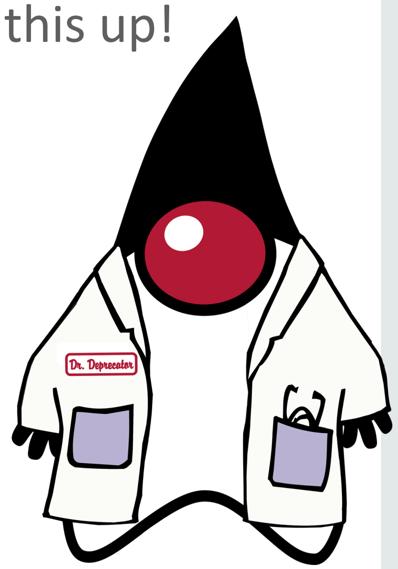
- Remove deprecated methods from "All Methods" ?

Potential IDE Enhancements

- Remove deprecated APIs from code completion by default
- Enable javac -Xlint:deprecation by default
- Add refactoring rules to migrate code away from deprecated APIs
  - many subtleties
  - Vector thread-safe, ArrayList is not
  - Vector.elements() is not fail-fast
  - ArrayList.iterator() is fail-fast

Conclusion

- This is all about controlled ***API evolution***
- Enable developers to migrate away from deprecated APIs effectively
- @Deprecated is currently confusing and inconsistent – clean this up!
- "When are you going to remove all the deprecated APIs?"
  - the Doctor's preferred tool is a scalpel, not an axe
  - only few things truly deserve removal; mark them clearly
- Deprecation JEP draft, in progress:
  - https://bugs.openjdk.java.net/browse/JDK-8065614
- See me on Twitter, I'm **@DrDeprecator**



JavaOne™

ORACLE®

ORACLE®