Challenges updating your code to work with Java 9 Jigsaw

Uwe Schindler

Apache Lucene PMC & Apache Software Foundation Member

uschindler@apache.org

https://www.thetaphi.de, http://blog.thetaphi.de

@ThetaPh1

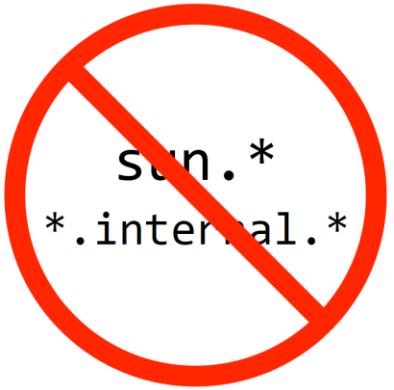
SD DataSolutions GmbH, Wätjenstr. 49, 28213 Bremen, Germany

Tel: +49 421 40889785-0, https://www.sd-datasolutions.de

What is this talk about?

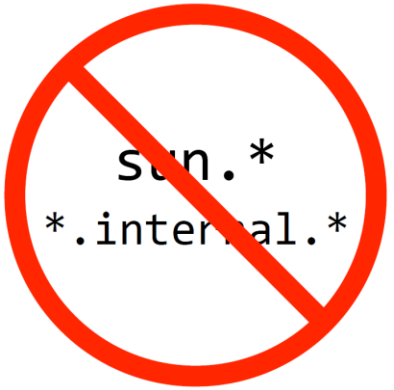
- **Migrating** your current project so it works with **Java 9 (Jigsaw)**
- Common **pitfalls** with Java 7 / Java 8 code, that just used to work
- *Not an introduction to the module system!*
- It does **not** show you how to “convert your project” to be a module

What changed in Jigsaw?
(module system)



- Strong encapsulation:
  - Code only sees classes from packages exported to your code
  - Private APIs are private – especially those in the JDK!

What changed in Jigsaw?
(module system)



- Strong encapsulation:
  - Code only sees classes from packages exported to your code
  - Private APIs are private – especially those in the JDK!
- *Your code behaves as if it will be executed with a security manager!* ☺

Examples

COMPILE TIME PROBLEMS

Direct use of invisible/removed APIs

- `sun.misc.BASE64Encoder /-Decoder`
- `sun.misc.Unsafe`
- `com.sun.javafx.*` (http://openjdk.java.net/jeps/253)

Direct use of invisible/removed APIs

- `sun.misc.BASE64Encoder /-Decoder`
- `sun.misc.Unsafe`
- `com.sun.javafx.*`
  (http://openjdk.java.net/jeps/253)

If compiled with older Java version it will result in `IllegalAccessError` on Java 9

Solution

Solution

```
1  import com.sun.security.sasl.Provider;
2
3  public class Test1 {
4    public static void main(String... args) {
5      new Provider();
6    }
7
8  }
```

Solution

```
1  import com.sun.security.sasl.Provider;
2
```

```
$ java Test1
Exception in thread "main" java.lang.IllegalAccessError: class Test1 (in unnamed module @0x64cee07)
cannot access class com.sun.security.sasl.Provider (in module java.security.sasl) because module
java.security.sasl does not export com.sun.security.sasl to unnamed module @0x64cee07
        at Test1.main(Test1.java:5)

$ _
```

```
7
8  }
```

Solution

- Scan your code with **jdeps** tool
  - Maven plugin available
  - Works only with **Java 8+**

Solution

- Scan your code with **jdeps** tool
  - Maven plugin available
  - Works only with **Java 8+**
- *Alternative:* **ForbiddenAPIs**
  - https://github.com/policeman-tools/forbidden-apis
  - `jdk-non-portable` or `jdk-internal-*` signatures
  - Maven/Gradle/Ant plugin for Java 6+

```
$ javac Test1.java
Test1.java:1: warning: Provider is internal proprietary API and may be removed in a future release
import com.sun.security.sasl.Provider;
                           ^
Test1.java:5: warning: Provider is internal proprietary API and may be removed in a future release
    new Provider();
        ^
2 warnings

$ jdeps -cp . -jdkinternals
. -> [...]/rt.jar
   Test1 (.)
      -> com.sun.security.sasl.Provider                     JDK internal API (rt.jar)

Warning: JDK internal APIs are unsupported and private to JDK implementation that are
subject to be removed or changed incompatibly and could break your application.
Please modify your code to eliminate dependency on any JDK internal APIs.
For the most recent update on JDK internal API replacements, please check:
https://wiki.openjdk.java.net/display/JDK8/Java+Dependency+Analysis+Tool

$ java -jar forbiddenapis-2.2.jar -d . -b jdk-non-portable
Scanning for classes to check...
Reading bundled API signatures: jdk-non-portable
Loading classes to check...
Scanning classes for violations...
ERROR: Forbidden class/interface use: com.sun.security.sasl.Provider [non-portable or internal runtime class]
ERROR:   in Test1 (Test1.java:5)
ERROR: Scanned 1 (and 6 related) class file(s) for forbidden API invocations (in 0.03s), 1 error(s).
ERROR: Check for forbidden API calls failed, see log.

$ _
```

Solution

- Scan your code with **jdeps** tool
  - Maven plugin available
  - Works only with **Java 8+**
- *Alternative:* **ForbiddenAPIs**
  - https://github.com/policeman-tools/forbidden-apis
  - `jdk-non-portable` or `jdk-internal-*` signatures
  - Maven/Gradle/Ant plugin for Java 6+
- Won't help if reflection was used!

Examples

REFLECTION

Reflection “hacks”

- Clever people use reflection to access private / internal Java APIs:

Reflection “hacks”

- Clever people use reflection to access private / internal Java APIs:
  - No compile-time dependency on Oracle JDK
  - Sometimes needed to access private members, e.g. “`sun.misc.Unsafe`” instance

Reflection “hacks”

Reflection “hacks”

- **Downside:** Static analysis can’t help

Reflection “hacks”

- **Downside:** Static analysis can’t help
- **Much worse:** No correct error handling *(if APIs are missing/incompatible)!*

Brokeness around the world

Brokeness around the world

- People use `setAccessible()` everywhere to break into internal APIs

Brokeness around the world

- People use `setAccessible()` everywhere to break into internal APIs
  - Almost no library does this correct

Brokeness around the world

- People use `setAccessible()` everywhere to break into internal APIs
  - Almost no library does this correct
- People don’t wrap with `AccessController.doPrivileged()`

Brokeness around the world

- People forget to add correct try/catch:

Brokeness around the world

- People forget to add correct try/catch:
  - `e.printStackTrace()`
  - `throw new RuntimeException(e)`

Brokeness around the world

- People forget to add correct try/catch:
  - `e.printStackTrace()`
  - `throw new RuntimeException(e)`
  - **…inside static initializers!**

Brokeness around the world

- People forget to add correct try/catch:
  - `e.printStackTrace()`
  - `throw new RuntimeException(e)`
  - **…inside static initializers!**
- No alternative solution:

Brokeness around the world

- People forget to add correct try/catch:
  - `e.printStackTrace()`
  - `throw new RuntimeException(e)`
  - **…inside static initializers!**
- No alternative solution:
  - static initializer breaks
  - `NoClassDefFoundError` forever!

What's wrong with Jigsaw?

#ReflectiveAccessToNonExportedTypes

#AwkwardStrongEncapsulation

- New since build 148 of Java 9
- Prevents reflective access to any class from Java runtime

What's wrong with Jigsaw?

#AwkwardStrongEncapsulation: A non-public element of an exported package can still be accessed via the `AccessibleObject::setAccessible` method of the core reflection API. The only way to strongly encapsulate such an element is to move it to a non-exported package. This makes it awkward, at best, to encapsulate the internals of a package that defines a public API.

What's wrong with Jigsaw?

#ReflectiveAccessToNonExportedTypes

#AwkwardStrongEncapsulation

- New since build 148 of Java 9
- Prevents reflective access to any class from Java runtime

What does no longer work?

What does no longer work?

- `Class.forName()`
  - on non-exported packages

What does no longer work?

- `Class.forName()`
  - on non-exported packages
- `AccessibleObject.setAccessible(true)`
  - on any *public* runtime class

What does no longer work?

- `Class.forName()`
  - on non-exported packages
- `AccessibleObject.setAccessible(true)`
  - on any *public* runtime class
- Some exceptions:
  - `sun.misc.Unsafe`

Problems

- No tool to detect reflective access to private APIs with earlier Java versions during testing/compilation

- Forbidden-APIs can disallow `AccessibleObject::setAccessible`

What can I do?

SOLUTIONS

Run tests with `SecurityManager`!

(Apache Lucene, Apache Solr, Elasticsearch)

Howto: Important patterns!

- Add fallbacks for private APIs (try…catch in static initializers)
- Catch `SecurityException` **AND** `RuntimeException`

Howto: Important patterns!

- Add fallbacks for private APIs (try…catch in static initializers)
- Catch `SecurityException` **AND** `RuntimeException`
  - `InaccessibleObjectException` extends `RuntimeException` ☹

Howto: Important patterns!

- **Don’t fail in static initializers** if you have no workaround!
  - Save error details while trying to initialize your private API hacks (Unsafe & Co.)
  - Use `AccessController#doPrivileged`
  - If consumer of your library calls a method using the hack, throw useful exception

Early binding using MethodHandles

- MethodHandles are bound early
  - like javac is compiling and type-checking a method call
- MethodHandles can be used to add "programming logic" with if/then/else
  - `MethodHandles.guardWithTest()` & Co.
- No linkage errors possible at call time

EXAMPLE

APACHE LUCENE'S MAPPEDBYTEBUFFER UNMAPPING

The Generics Policeman Blog

MMAP

2012-07-18

Use Lucene's M...ap...ctory o...bit...orms, ...lease!

Don't be afraid – S...ne...ari...ca...to...om...n misunderstandings

Since version 3.1, **Apache Lucene** and **Solr** use `MMapDirectory` by ...ult on 64bi... ...dow and Solaris systems; since version 3.3 also for 64bit Linux systems. This change lead ... ...me confusion among Lucene and Solr users, because suddenly their systems started to behave differently than in previous versions. On the Lucene and Solr mailing lists a lot of posts arrived from users asking why their Java installation is suddenly consuming three times their physical memory or system administrators complaining about heavy resource usage. Also consultants were starting to tell people that they should **not** use `MMapDirectory` and change their solrconfig.xml to work instead with slow `SimpleFSDirectory` or `NIOFSDirectory` (which is much slower on Windows, caused by a JVM bug #6265734). From the point of view of the Lucene committers, who carefully decided that using `MMapDirectory` is the best for those platforms, this is rather annoying, because they know, that Lucene/Solr can work with much better performance than before. Common misinformation about the background of this change causes suboptimal installations of this great search engine everywhere.

In this blog post, I will try to explain the basic operating system facts regarding virtual memory handling in the kernel and how this can be used to largely improve performance of Lucene

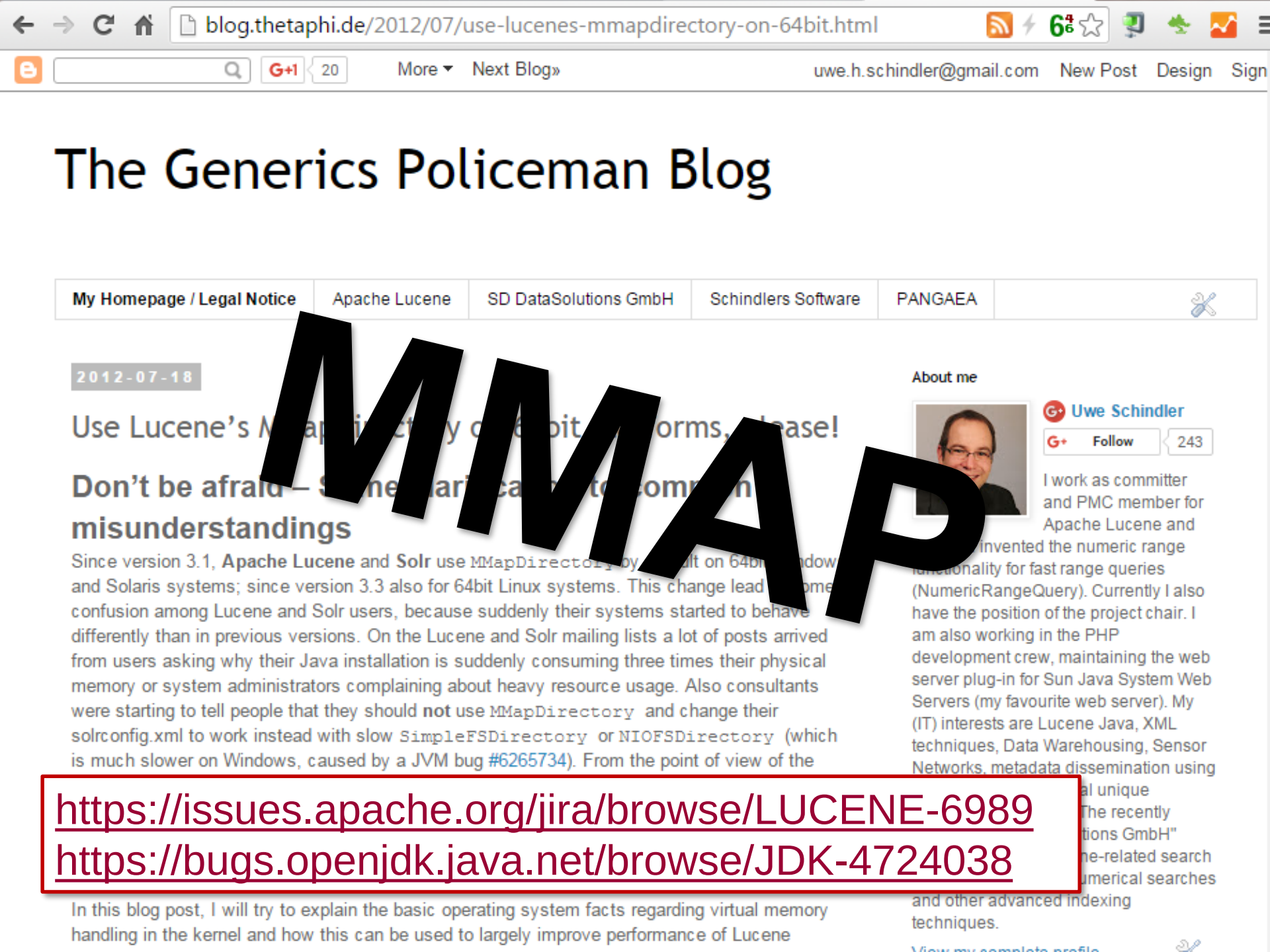
About me

Uwe Schindler

G+ Follow 243

I work as committer and PMC member for Apache Lucene and ... invented the numeric range ...functionality for fast range queries (NumericRangeQuery). Currently I also have the position of the project chair. I am also working in the PHP development crew, maintaining the web server plug-in for Sun Java System Web Servers (my favourite web server). My (IT) interests are Lucene Java, XML techniques, Data Warehousing, Sensor Networks, metadata dissemination using global standards, global unique identifiers like DOIs,... The recently founded "SD DataSolutions GmbH" offers support for Lucene-related search solutions, especially numerical searches and other advanced indexing techniques.

View my complete profile


blog.thetaphi.de/2012/07/use-lucenes-mmapdirectory-on-64bit.html
G+1 20
More
Next Blog»
uwe.h.schindler@gmail.com
New Post
Design
Sign
The Generics Policeman Blog
My Homepage / Legal Notice
Apache Lucene
SD DataSolutions GmbH
Schindlers Software
PANGAEA
MMAP
2012-07-18
Use Lucene's M
ap
ct
y
bit
orms,
ease!
Don't be afraid –
misunderstandings
Since version 3.1, Apache Lucene and Solr use MMapDirectory by
lt on 64bi
ndow
and Solaris systems; since version 3.3 also for 64bit Linux systems. This change lead
me
confusion among Lucene and Solr users, because suddenly their systems started to behave
differently than in previous versions. On the Lucene and Solr mailing lists a lot of posts arrived
from users asking why their Java installation is suddenly consuming three times their physical
memory or system administrators complaining about heavy resource usage. Also consultants
were starting to tell people that they should not use MMapDirectory and change their
solrconfig.xml to work instead with slow SimpleFSDirectory or NIOFSDirectory (which
is much slower on Windows, caused by a JVM bug #6265734). From the point of view of the
https://issues.apache.org/jira/browse/LUCENE-6989
https://bugs.openjdk.java.net/browse/JDK-4724038
In this blog post, I will try to explain the basic operating system facts regarding virtual memory
handling in the kernel and how this can be used to largely improve performance of Lucene
About me
Uwe Schindler
Follow
243
I work as committer
and PMC member for
Apache Lucene and
invented the numeric range
functionality for fast range queries
(NumericRangeQuery). Currently I also
have the position of the project chair. I
am also working in the PHP
development crew, maintaining the web
server plug-in for Sun Java System Web
Servers (my favourite web server). My
(IT) interests are Lucene Java, XML
techniques, Data Warehousing, Sensor
Networks, metadata dissemination using
al unique
The recently
tions GmbH"
e-related search
umerical searches
and other advanced indexing
techniques.

```
// *** sun.misc.Cleaner unmapping (Java 8) ***
final Class<?> directBufferClass = Class.forName("java.nio.DirectByteBuffer");

final Method m = directBufferClass.getMethod("cleaner");
m.setAccessible(true);
final MethodHandle directBufferCleanerMethod = lookup.unreflect(m);
final Class<?> cleanerClass = directBufferCleanerMethod.type().returnType();

/* "Compile" a MH that basically is equivalent to the following code:
 * void unmapper(ByteBuffer byteBuffer) {
 *   sun.misc.Cleaner cleaner = ((java.nio.DirectByteBuffer) byteBuffer).cleaner();
 *   if (Objects.nonNull(cleaner)) {
 *     cleaner.clean();
 *   } else {
 *     noop(cleaner); // the noop is needed because MethodHandles#guardWithTest always needs ELSE
 *   }
 * }
 */
final MethodHandle cleanMethod = lookup.findVirtual(cleanerClass, "clean", methodType(void.class));
final MethodHandle nonNullTest = lookup.findStatic(Objects.class, "nonNull", methodType(boolean.class, Object.class))
    .asType(methodType(boolean.class, cleanerClass));
final MethodHandle noop = dropArguments(constant(Void.class, null).asType(methodType(void.class)), 0, cleanerClass);
final MethodHandle unmapper = filterReturnValue(directBufferCleanerMethod, guardWithTest(nonNullTest, cleanMethod, noop))
    .asType(methodType(void.class, ByteBuffer.class));
return newBufferCleaner(directBufferClass, unmapper);
```

```
// *** sun.misc.Cleaner unmapping (Java 8) ***
final Class<?> directBufferClass = Class.forName("java.nio.DirectByteBuffer");

final Method m = directBufferClass.getMethod("cleaner");
m.setAccessible(true);
final MethodHandle directBufferCleanerMethod = lookup.unreflect(m);
final Class<?> cleanerClass = directBufferCleanerMethod.type().returnType();

/* "Compile" a MH that basically is equivalent to the following code:
 * void unmapper(ByteBuffer byteBuffer) {
 *   sun.misc.Cleaner cleaner = ((java.nio.DirectByteBuffer) byteBuffer).cleaner();
 *   if (Objects.nonNull(cleaner)) {
 *     cleaner.clean();
 *   } else {
 *     noop(cleaner); // the noop is needed because MethodHandles#guardWithTest always needs ELSE
 *   }
 * }
 */
final MethodHandle cleanMethod = lookup.findVirtual(cleanerClass, "clean", methodType(void.class));
final MethodHandle nonNullTest = lookup.findStatic(Objects.class, "nonNull", methodType(boolean.class, Object.class))
    .asType(methodType(boolean.class, cleanerClass));
final MethodHandle noop = dropArguments(constant(Void.class, null).asType(methodType(void.class)), 0, cleanerClass);
final MethodHandle unmapper = filterReturnValue(directBufferCleanerMethod, guardWithTest(nonNullTest, cleanMethod, noop))
    .asType(methodType(void.class, ByteBuffer.class));
return newBufferCleaner(directBufferClass, unmapper);
```

```
// *** sun.misc.Cleaner unmapping (Java 8) ***
final Class<?> directBufferClass = Class.forName("java.nio.DirectByteBuffer");

final Method m = directBufferClass.getMethod("cleaner");
m.setAccessible(true);
final MethodHandle directBufferCleanerMethod = lookup.unreflect(m);
final Class<?> cleanerClass = directBufferCleanerMethod.type().returnType();

/* "Compile" a MH that basically is equivalent to the following code:
 * void unmapper(ByteBuffer byteBuffer) {
 *   sun.misc.Cleaner cleaner = ((java.nio.DirectByteBuffer) byteBuffer).cleaner();
 *   if (Objects.nonNull(cleaner)) {
 *     cleaner.clean();
 *   } else {
 *     noop(cleaner); // the noop is needed because MethodHandles#guardWithTest always needs ELSE
 *   }
 * }
 */
final MethodHandle cleanMethod = lookup.findVirtual(cleanerClass, "clean", methodType(void.class));
final MethodHandle nonNullTest = lookup.findStatic(Objects.class, "nonNull", methodType(boolean.class, Object.class))
    .asType(methodType(boolean.class, cleanerClass));
final MethodHandle noop = dropArguments(constant(Void.class, null).asType(methodType(void.class)), 0, cleanerClass);
final MethodHandle unmapper = filterReturnValue(directBufferCleanerMethod, guardWithTest(nonNullTest, cleanMethod, noop))
    .asType(methodType(void.class, ByteBuffer.class));
return newBufferCleaner(directBufferClass, unmapper);
```

```
  // *** sun.misc.Unsafe unmapping (Java 9+) ***
  final Class<?> unsafeClass = Class.forName("sun.misc.Unsafe");
  // first check if Unsafe has the right method, otherwise we can give up
  // without doing any security critical stuff:
  final MethodHandle unmapper = lookup.findVirtual(unsafeClass, "invokeCleaner",
      methodType(void.class, ByteBuffer.class));
  // fetch the unsafe instance and bind it to the virtual MH:
  final Field f = unsafeClass.getDeclaredField("theUnsafe");
  f.setAccessible(true);
  final Object theUnsafe = f.get(null);
  return newBufferCleaner(ByteBuffer.class, unmapper.bindTo(theUnsafe));
} catch (SecurityException se) {
  // rethrow to report errors correctly (we need to catch it here, as we also catch RuntimeException below!):
  throw se;
} catch (ReflectiveOperationException | RuntimeException e) {
  // *** sun.misc.Cleaner unmapping (Java 8) ***
  final Class<?> directBufferClass = Class.forName("java.nio.DirectByteBuffer");
```

```
        .asType(methodType(void.class, ByteBuffer.class));
      return newBufferCleaner(directBufferClass, unmapper);
    }
  } catch (SecurityException se) {
    return "Unmapping is not supported, because not all required permissions are given to the Lucene JAR f
        " [Please grant at least the following permissions: RuntimePermission(\"accessClassInPackage.sun.m
        " and ReflectPermission(\"suppressAccessChecks\")]";
  } catch (ReflectiveOperationException | RuntimeException e) {
    return "Unmapping is not supported on this platform, because internal Java APIs are not compatible wit
  }
}
```

Thank You!

THE APACHE™ SOFTWARE FOUNDATION
http://www.apache.org/

Lucene