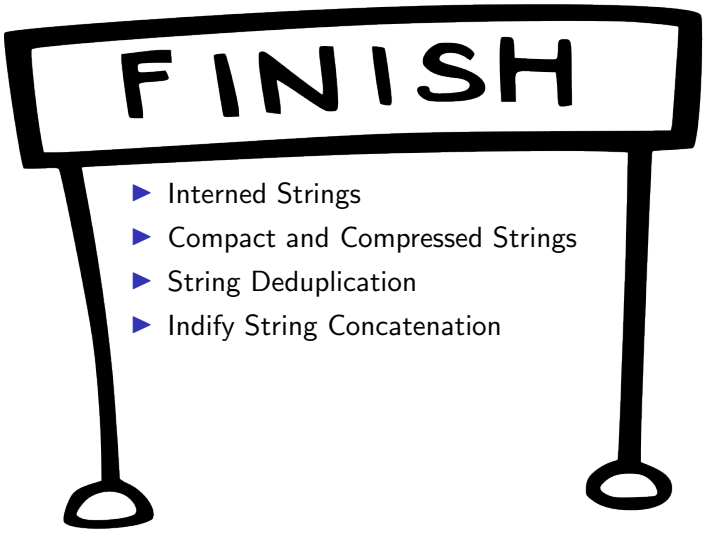
# What Every Java Programmer Should Know About Strings

Bernd Müller
Ostfalia


FINISH
Interned Strings
Compact and Compressed Strings
String Deduplication
Indify String Concatenation

# Speaker

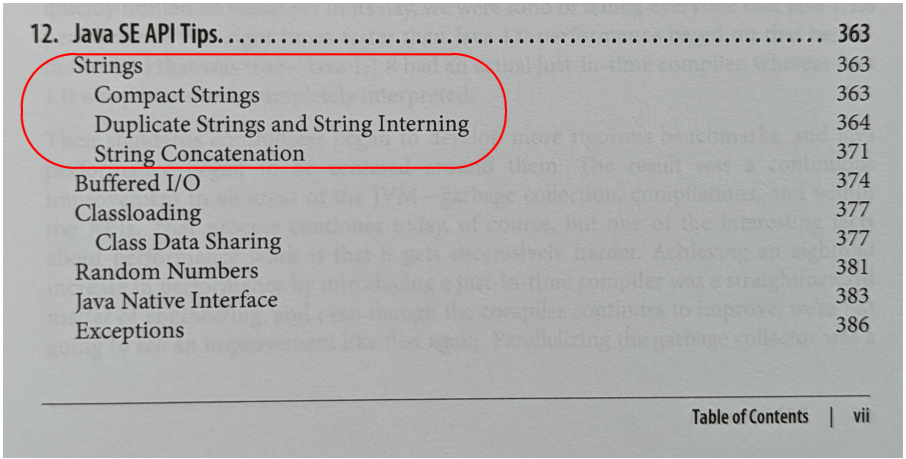
- Prof. Computer Science (Ostfalia, HS Braunschweig/Wolfenbüttel)
- Book author (JSF, JPA, Seam, ...)

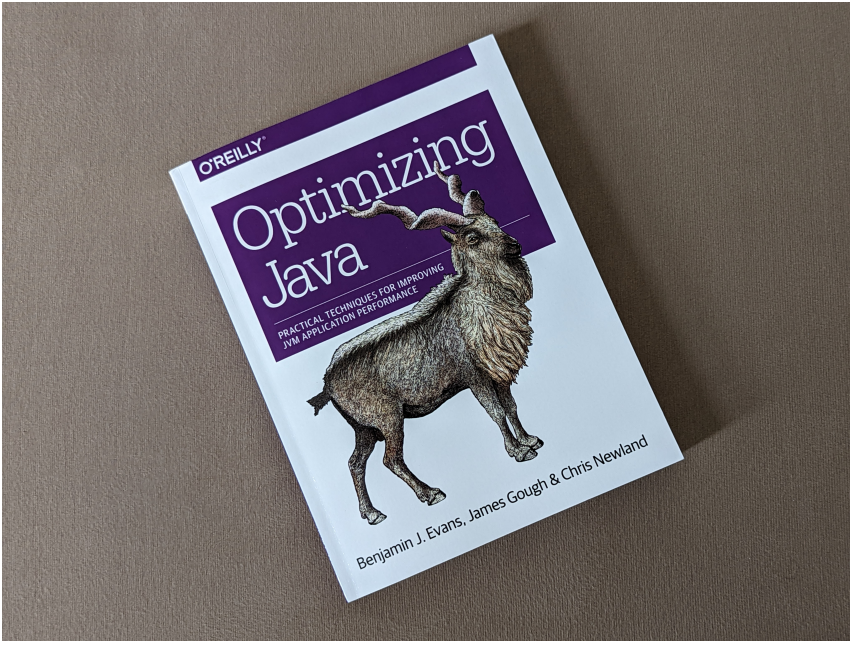

- Member EGs JSR 344 (JSF 2.2) und JSR 338 (JPA 2.1)
- Managing Director PMST GmbH
- JUG Ostfalen Co-Organizer
- bernd.mueller@ostfalia.de
- @berndmuller @berndmuller@fosstodon.org BerndMuller

# Motivation

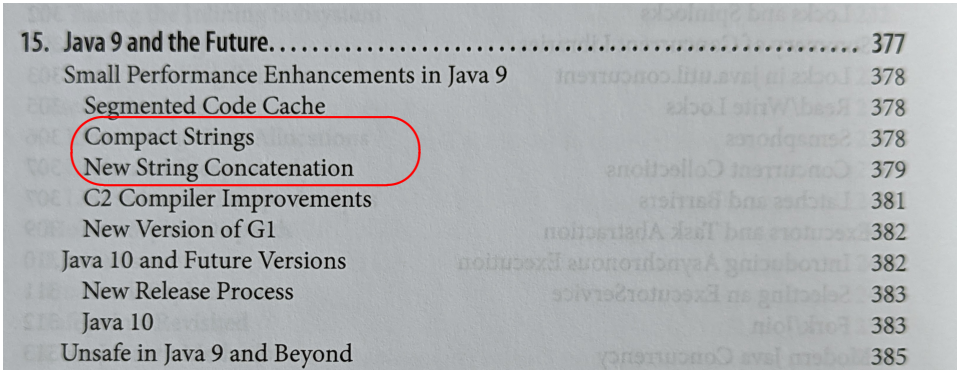

Second Edition
O'REILLY
Java Performance
In-Depth Advice for Tuning and Programming Java 8, 11, and Beyond
Scott Oaks

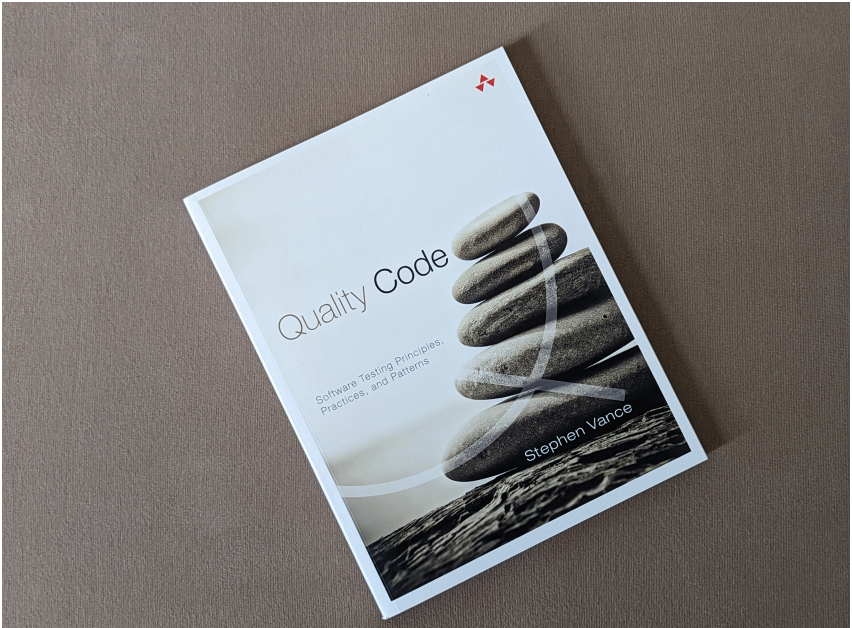
## Scott Oaks, Java Performance



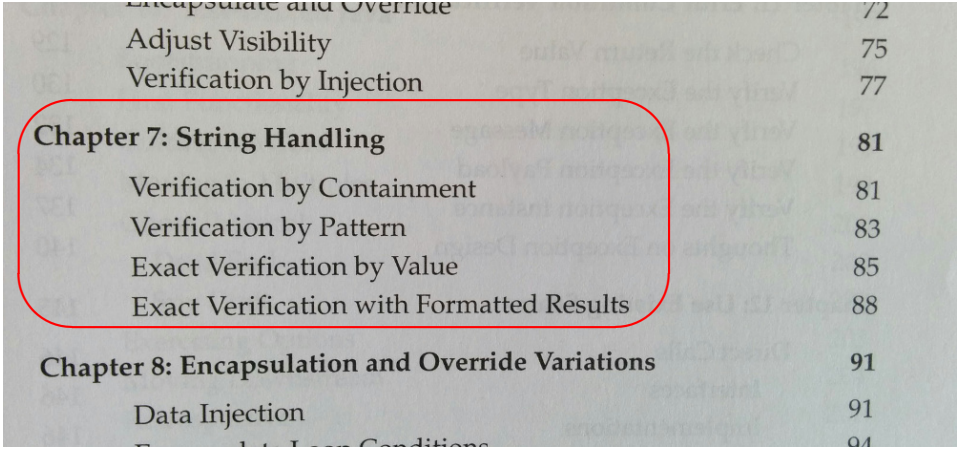

O'REILLY
Optimizing Java
PRACTICAL TECHNIQUES FOR IMPROVING JVM APPLICATION PERFORMANCE
Benjamin J. Evans, James Gough & Chris Newland

## Benjamin Evens et al., Optimizing Java




Quality Code
Software Testing Principles, Practices, and Patterns
Stephen Vance

## Stephen Vance, Quality Code



## Let's start with a quiz

How many bytes are used to store a single character inside a string ?

(a) 1 byte
(b) 2 bytes
(c) 4 bytes
(d) 2 or 4 bytes
(e) 1, 2 or 4 bytes

# String Pool and String Interning

## Java Language Specification

### 3.10.5 String Literals

. . .

Moreover, a string literal always refers to the same instance of class String. This is because string literals - or, more generally, strings that are the values of constant expressions (§15.29) - are "interned" so as to share unique instances, using the method String.intern (§12.5).

. . .

## Java Language Sepcification

**12.5 Creation of New Class Instances**

. . .

Loading of a class or interface that contains a string literal (§3.10.5) or a text block (§3.10.6) may create a new String object to denote the string represented by the string literal or text block. (This object creation will not occur if an instance of String denoting the same sequence of Unicode code points as the string represented by the string literal or text block has previously been interned.)

. . .

Execution of a string concatenation operator + (§15.18.1) that is not part of a constant expression (§15.29) always creates a new String object to represent the result. String concatenation operators may also create temporary wrapper objects for a value of a primitive type.

. . .

## Summing up

- All interned strings are stored in a string memory pool
- When a class gets loaded check if string already in the pool
- If so use it, if not insert it
- Result: each string literal is a singleton inside JVM
- Also summed up in JavaDoc of `String.intern()`

# Demo Time

Sleepy from slides, we are !

## Performance Tip from Scott Oaks [Java Performance]

"On the topic of interning strings, what about using the `intern()` method to make the programm run faster, since interned strings can be compared via the == operator? That is a popular thought, though in most cases it turns out to by a myth. The String.equals() method is pretty fast."

## Performance Tip from Scott Oaks [Java Performance] (cont'd)

"Comparing strings via the == operator is undeniably faster, but the cost of interning the string must also be taken into consideration."

## Performance Tip from Scott Oaks [Java Performance] (cont'd)

"Like most optimizations, interning strings shouldn't be done arbitrarily, but it can be effective if there are lots of duplicate strings occupying a significant portion of the heap."

## String Pool Details

- Hash table in native memory, strings itself on heap

## String Pool Details

- Hash table in native memory, strings itself on heap
- Fixed size (1009 buckets $<$ 7. Then 60,013. Since Java 11 65,536 )

## String Pool Details

- Hash table in native memory, strings itself on heap
- Fixed size (1009 buckets $< 7$. Then 60,013. Since Java 11 65,536 )
- $< 6$ in PermGen, $\geq 7$ Heap. OOME PermGen space or OOME heap space

## String Pool Details

- Hash table in native memory, strings itself on heap
- Fixed size (1009 buckets $<$ 7. Then 60,013. Since Java 11 65,536 )
- $<$ 6 in PermGen, $\geq$ 7 Heap. OOME PermGen space or OOME heap space
- If you want to optimize:
  - `jmap -heap <process-id>`
  - `-XX:+PrintStringTableStatistics`
  - `-XX:StringTableSize=<value>`

## Off topic: Mistakes can happen ...

```
/** The offset is the first index of the storage that is used. */
private final int offset;
/** The count is the number of characters in the String. */
private final int count;
```

- used for example in `substring(begin, end)`

## Off topic: Mistakes can happen ...

```
/** The offset is the first index of the storage that is used. */
private final int offset;
/** The count is the number of characters in the String. */
private final int count;
```

- used for example in `substring(begin, end)`
- JDK-4637640 : Memory leak due to `String.substring()` implementation
- `offset` and `count` removed with Java 7

# Compact Strings (JEP 254)

## Some History

- Java started 1995 as an Internet Language
- Therefore Unicode Standard, 16 bit `char` type
- Class `String` internally:

```
/** The value is used for character storage. */
private final char value[];
```

## Some History

- Java started 1995 as an Internet Language
- Therefore Unicode Standard, 16 bit `char` type
- Class `String` internally:

```
/** The value is used for character storage. */
private final char value[];
```

- With Java 5 Unicode 4.0 used: sometimes requires 32 bit

## Some History (cont'd)

- So called supplementary characters represented as pair of `char` values for characters greater than U+FFFF. Documented in class `Character`

## Some History (cont'd)

- So called supplementary characters represented as pair of `char` values for characters greater than U+FFFF. Documented in class `Character`
- E.g. JavaDoc of `String.length()` changed from Java 5

  *Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string*

  to Java 6

  *Returns the length of this string. The length is equal to the number of Unicode code units in the string*

## Some History (cont'd)

- So called supplementary characters represented as pair of `char` values for characters greater than U+FFFF. Documented in class `Character`
- E.g. JavaDoc of `String.length()` changed from Java 5

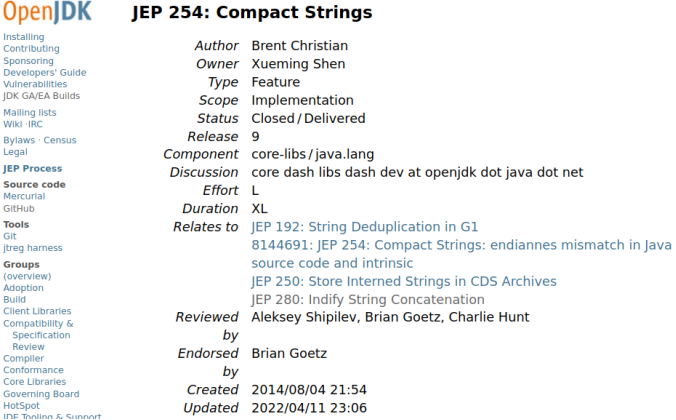
  *Returns the length of this string. The length is equal to the number of 16-bit Unicode characters in the string*

  to Java 6

  *Returns the length of this string. The length is equal to the number of Unicode code units in the string*

- Recap: 1 byte suffices mostly (Latin-1), sometimes 2, sometimes 4 bytes

# JEP 254: Compact Strings



OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

Bylaws · Census
Legal

**JEP Process**

**Source code**
Mercurial
GitHub

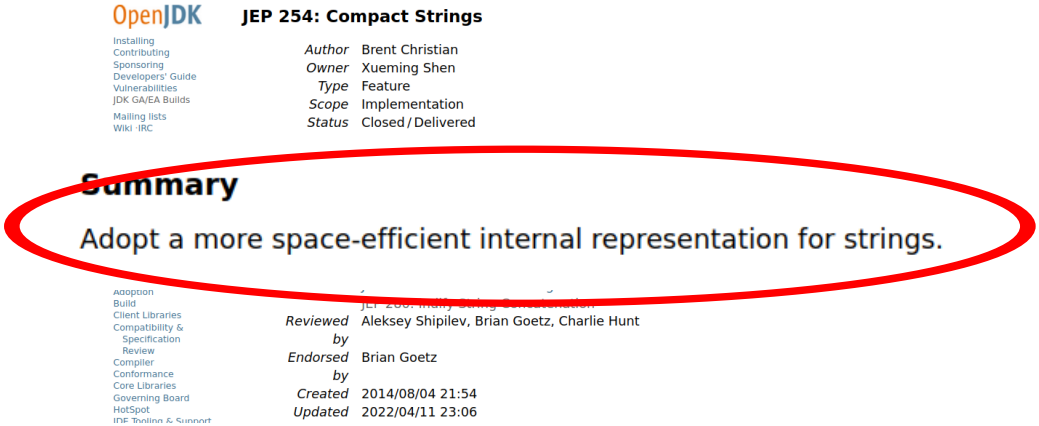
**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support

**JEP 254: Compact Strings**

| | |
|---|---|
| *Author* | Brent Christian |
| *Owner* | Xueming Shen |
| *Type* | Feature |
| *Scope* | Implementation |
| *Status* | Closed / Delivered |
| *Release* | 9 |
| *Component* | core-libs / java.lang |
| *Discussion* | core dash libs dash dev at openjdk dot java dot net |
| *Effort* | L |
| *Duration* | XL |
| *Relates to* | JEP 192: String Deduplication in G1<br>8144691: JEP 254: Compact Strings: endiannes mismatch in Java source code and intrinsic<br>JEP 250: Store Interned Strings in CDS Archives<br>JEP 280: Indify String Concatenation |
| *Reviewed by* | Aleksey Shipilev, Brian Goetz, Charlie Hunt |
| *Endorsed by* | Brian Goetz |
| *Created* | 2014/08/04 21:54 |
| *Updated* | 2022/04/11 23:06 |

# JEP 254: Compact Strings



OpenJDK

**JEP 254: Compact Strings**

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

| | |
|---|---|
| *Author* | Brent Christian |
| *Owner* | Xueming Shen |
| *Type* | Feature |
| *Scope* | Implementation |
| *Status* | Closed / Delivered |

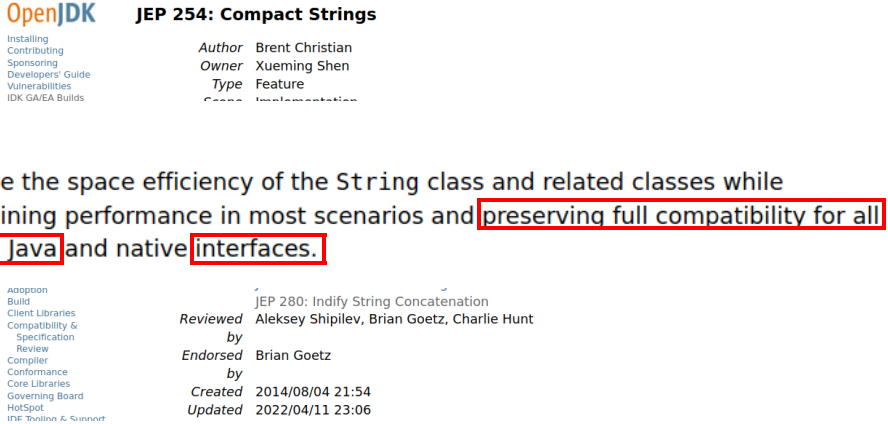
## Summary

Adopt a more space-efficient internal representation for strings.

Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support

| | |
|---|---|
| *Reviewed by* | Aleksey Shipilev, Brian Goetz, Charlie Hunt |
| *Endorsed by* | Brian Goetz |
| *Created* | 2014/08/04 21:54 |
| *Updated* | 2022/04/11 23:06 |

# JEP 254: Compact Strings



OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
IDK GA/EA Builds

**JEP 254: Compact Strings**

| | |
|---|---|
| *Author* | Brent Christian |
| *Owner* | Xueming Shen |
| *Type* | Feature |

**Goals**

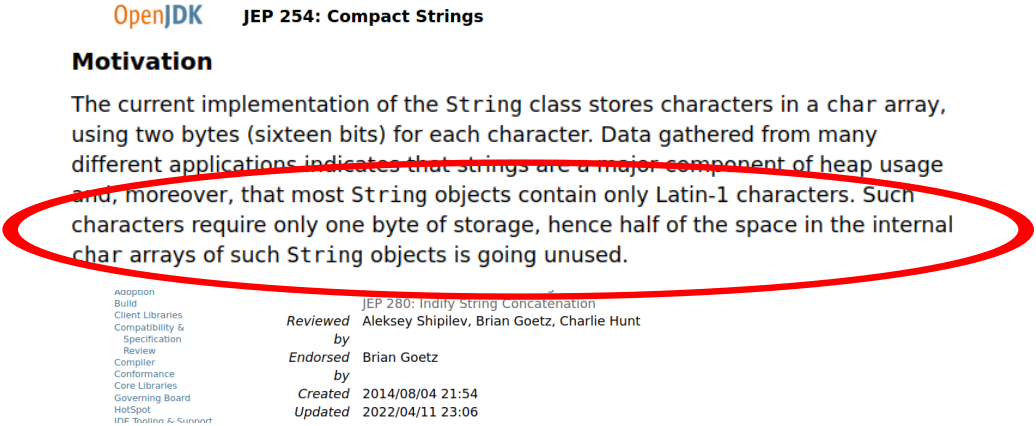
Improve the space efficiency of the `String` class and related classes while maintaining performance in most scenarios and preserving full compatibility for all related Java and native interfaces.

Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support

JEP 280: Indify String Concatenation

| | |
|---|---|
| *Reviewed by* | Aleksey Shipilev, Brian Goetz, Charlie Hunt |
| *Endorsed by* | Brian Goetz |
| *Created* | 2014/08/04 21:54 |
| *Updated* | 2022/04/11 23:06 |

# JEP 254: Compact Strings



OpenJDK **JEP 254: Compact Strings**

## Motivation

The current implementation of the `String` class stores characters in a `char` array, using two bytes (sixteen bits) for each character. Data gathered from many different applications indicates that strings are a major component of heap usage and, moreover, that most `String` objects contain only Latin-1 characters. Such characters require only one byte of storage, hence half of the space in the internal `char` arrays of such `String` objects is going unused.

Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support

JEP 280: Indify String Concatenation

| | |
|---|---|
| *Reviewed by* | Aleksey Shipilev, Brian Goetz, Charlie Hunt |
| *Endorsed by* | Brian Goetz |
| *Created* | 2014/08/04 21:54 |
| *Updated* | 2022/04/11 23:06 |

## With Java 9 it happened

```
private final char value[];
```

## With Java 9 it happened

```
private final char value[];
```

changed to

```
private final byte[] value;

/**
* The identifier of the encoding used to encode the bytes in
* {@code value}. The supported values in this implementation are
* LATIN1, UTF16
*/
private final byte coder;
```

## A Masterpiece of Software Engineering

No API changes in class String from Java 8 to 9

## Usage

- Nothing to do. Default since Java 9
- VM parameter: `-XX:-CompactStrings`

# Demo Time

Sleepy from slides, we are !

## But things can also go wrong . . .

- JDK 6 introduced *Compressed Strings*
- Similar idea but different implementation (two variants)
- Decision on JVM level: `-XX:+UseCompressedStrings`
- But . . .

## But things can also go wrong . . .

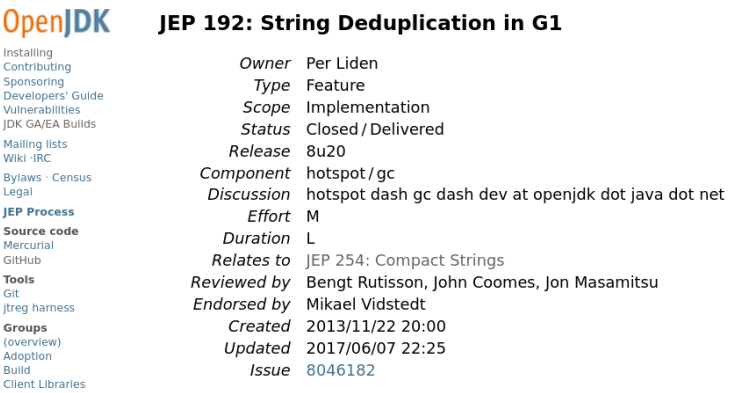
- JDK 6 introduced *Compressed Strings*
- Similar idea but different implementation (two variants)
- Decision on JVM level: `-XX:+UseCompressedStrings`
- But . . .
- Aleksey Shipilv: "*UseCompressedStrings was really the experimental feature, that was ultimately limited by design, error-prone, and hard to maintain.*"

## But things can also go wrong . . .

- JDK 6 introduced *Compressed Strings*
- Similar idea but different implementation (two variants)
- Decision on JVM level: `-XX:+UseCompressedStrings`
- But . . .
- Aleksey Shipilv: "*UseCompressedStrings was really the experimental feature, that was ultimately limited by design, error-prone, and hard to maintain.*"
- Revoked with JDK 7

# String Deduplication in G1 (JEP 192)

# JEP 192: String Deduplication in G1



OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

Bylaws · Census
Legal

**JEP Process**

**Source code**
Mercurial
GitHub

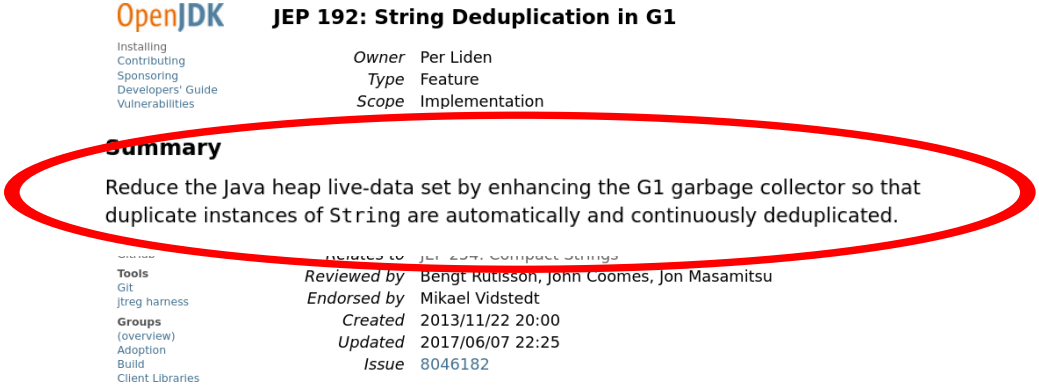
**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries

**JEP 192: String Deduplication in G1**

| | |
|---|---|
| *Owner* | Per Liden |
| *Type* | Feature |
| *Scope* | Implementation |
| *Status* | Closed / Delivered |
| *Release* | 8u20 |
| *Component* | hotspot / gc |
| *Discussion* | hotspot dash gc dash dev at openjdk dot java dot net |
| *Effort* | M |
| *Duration* | L |
| *Relates to* | JEP 254: Compact Strings |
| *Reviewed by* | Bengt Rutisson, John Coomes, Jon Masamitsu |
| *Endorsed by* | Mikael Vidstedt |
| *Created* | 2013/11/22 20:00 |
| *Updated* | 2017/06/07 22:25 |
| *Issue* | 8046182 |

# JEP 192: String Deduplication in G1



OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities

**JEP 192: String Deduplication in G1**

| | |
|---|---|
| *Owner* | Per Liden |
| *Type* | Feature |
| *Scope* | Implementation |

**Summary**

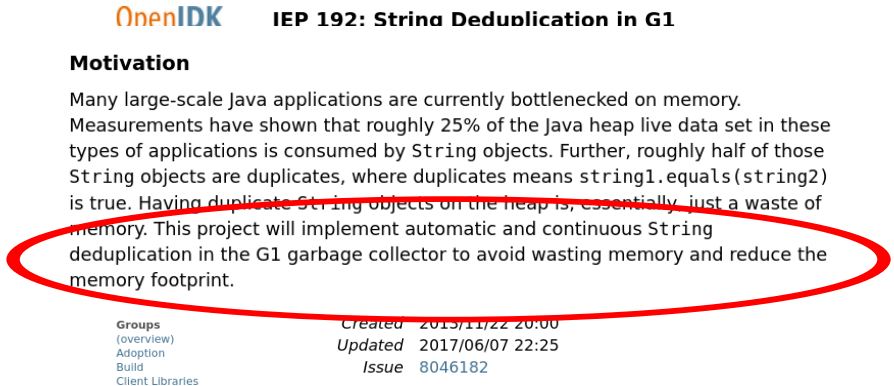
Reduce the Java heap live-data set by enhancing the G1 garbage collector so that duplicate instances of `String` are automatically and continuously deduplicated.

**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries

| | |
|---|---|
| *Reviewed by* | Bengt Rutisson, John Coomes, Jon Masamitsu |
| *Endorsed by* | Mikael Vidstedt |
| *Created* | 2013/11/22 20:00 |
| *Updated* | 2017/06/07 22:25 |
| *Issue* | 8046182 |

# JEP 192: String Deduplication in G1



OpenJDK **JEP 192: String Deduplication in G1**

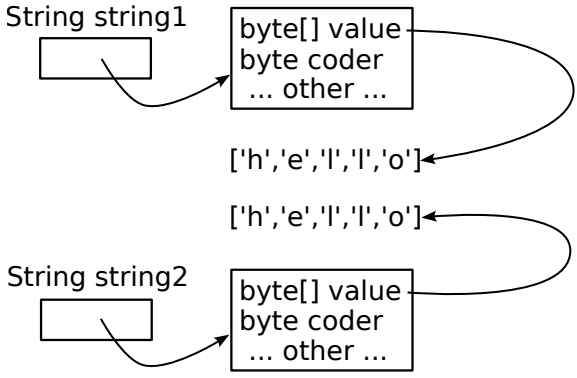
**Motivation**

Many large-scale Java applications are currently bottlenecked on memory. Measurements have shown that roughly 25% of the Java heap live data set in these types of applications is consumed by `String` objects. Further, roughly half of those `String` objects are duplicates, where duplicates means `string1.equals(string2)` is true. Having duplicate `String` objects on the heap is, essentially, just a waste of memory. This project will implement automatic and continuous `String` deduplication in the G1 garbage collector to avoid wasting memory and reduce the memory footprint.

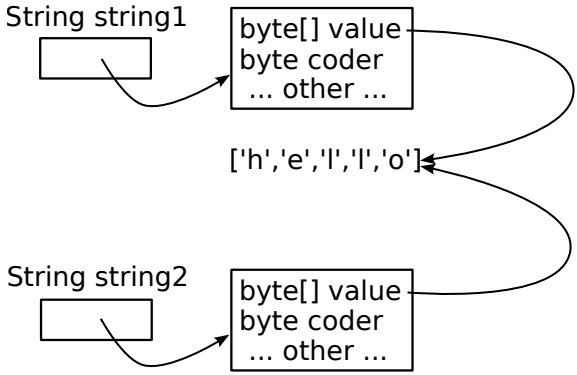
**Groups**
(overview)
Adoption
Build
Client Libraries

*Created* 2013/11/22 20:00
*Updated* 2017/06/07 22:25
*Issue* 8046182

## How does it work ?



## After String Deduplication



## Usage

- Only works with G1 garbage collector
- VM parameter: `-XX:+UseStringDeduplication`
- Available since Java 8 update 20
- `-XX:+PrintStringDeduplicationStatistics` removed with Java 9
- Use `-Xlog:stringdedup*=debug` instead

# Demo Time

Sleepy from slides, we are !

## Case Study

- Article G1: from garbage collector to waste management consultant
- Case study with Eclipse IDE
- Result: Decrease heap usage by about 10%

# Indify String Concatenation (JEP 280)

Wiktionary
The free dictionary

Main Page
Community portal
Preferences
Requested entries
Recent changes
Random entry
Help
Glossary
Donations
Contact us

Tools
What links here

Not logged in Talk Contributions Preferences Create account Log in

Entry Discussion Citations | Read Edit History | Search Wiktionary

# indify

## English [ edit ]

### Verb [ edit ]

**indify** (*third-person singular simple present* **indifies**, *present participle* **indifying**, *simple past and past participle* **indified**)

1. (*programming*, *JAVA*) Change a particular functionality to use invokedynamic calls.

### Anagrams [ edit ]

- nidify

Categories: English lemmas | English verbs | en:Programming

# JEP 280: Indify String Concatenation

OpenJDK

Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds

Mailing lists
Wiki · IRC

Bylaws · Census
Legal

**JEP Process**

**Source code**
Mercurial
GitHub

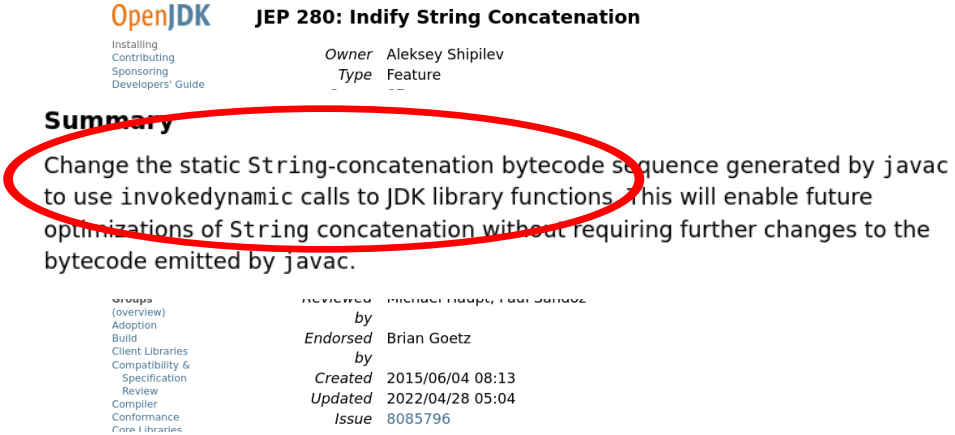
**Tools**
Git
jtreg harness

**Groups**
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries

**JEP 280: Indify String Concatenation**

| | |
|---|---|
| *Owner* | Aleksey Shipilev |
| *Type* | Feature |
| *Scope* | SE |
| *Status* | Closed / Delivered |
| *Release* | 9 |
| *Component* | tools / javac |
| *Discussion* | core dash libs dash dev at openjdk dot java dot net, compiler dash dev at openjdk dot java dot net, hotspot dash dev at openjdk dot java dot net |
| *Effort* | M |
| *Duration* | M |
| *Relates to* | JEP 254: Compact Strings |
| *Reviewed by* | Michael Haupt, Paul Sandoz |
| *Endorsed by* | Brian Goetz |
| *Created* | 2015/06/04 08:13 |
| *Updated* | 2022/04/28 05:04 |
| *Issue* | 8085796 |

# JEP 280: Indify String Concatenation



OpenJDK **JEP 280: Indify String Concatenation**

Installing
Contributing
Sponsoring
Developers' Guide

*Owner* Aleksey Shipilev
*Type* Feature

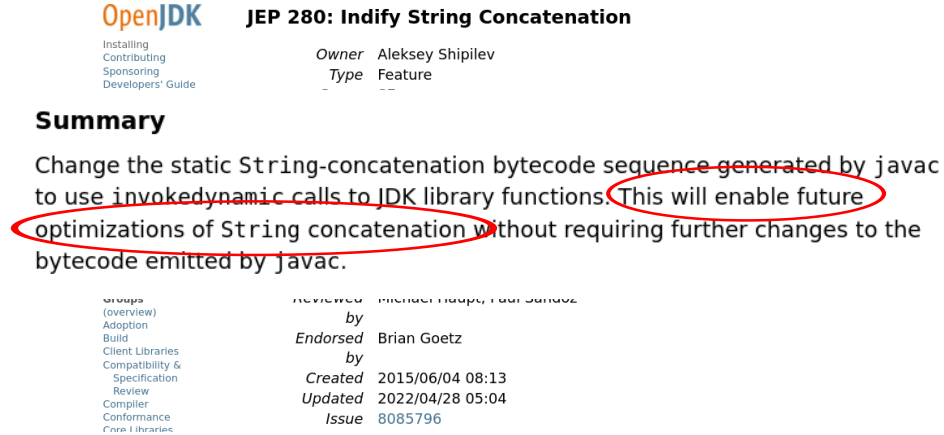
**Summary**

Change the static `String`-concatenation bytecode sequence generated by `javac` to use `invokedynamic` calls to JDK library functions. This will enable future optimizations of `String` concatenation without requiring further changes to the bytecode emitted by `javac`.

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility &
Specification
Review
Compiler
Conformance
Core Libraries

*Reviewed by* Michael Haupt, Paul Sandoz
*Endorsed by* Brian Goetz
*Created* 2015/06/04 08:13
*Updated* 2022/04/28 05:04
*Issue* 8085796

# JEP 280: Indify String Concatenation



OpenJDK **JEP 280: Indify String Concatenation**

Installing
Contributing
Sponsoring
Developers' Guide

*Owner* Aleksey Shipilev
*Type* Feature

**Summary**

Change the static `String`-concatenation bytecode sequence generated by `javac` to use `invokedynamic` calls to JDK library functions. This will enable future optimizations of `String` concatenation without requiring further changes to the bytecode emitted by `javac`.

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility &
Specification
Review
Compiler
Conformance
Core Libraries

*Reviewed by* Michael Haupt, Paul Sandoz
*Endorsed by* Brian Goetz
*Created* 2015/06/04 08:13
*Updated* 2022/04/28 05:04
*Issue* 8085796

## What they did

- Replace StringBuilder#append() chains with
  - invokedynamic
  - bootstrap methods
  - Class `java.lang.invoke.StringConcatFactory`
- Please verify yourself with `javap -l -s -verbose <class>`

**Module** java.base
**Package** java.lang.invoke

# Class StringConcatFactory

java.lang.Object
    java.lang.invoke.StringConcatFactory

---

```
public final class StringConcatFactory
extends Object
```

Methods to facilitate the creation of String concatenation methods, that can be used to efficiently concatenate a known number of arguments of known types, possibly after type adaptation and partial evaluation of arguments. These methods are typically used as *bootstrap methods* for `invokedynamic` call sites, to support the *string concatenation* feature of the Java Programming Language.

Indirect access to the behavior specified by the provided `MethodHandle` proceeds in order through two phases:

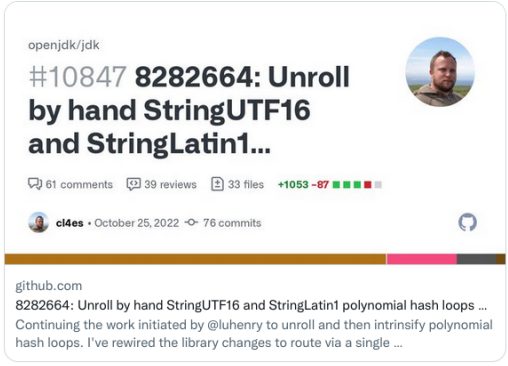
1. *Linkage* occurs when the methods in this class are invoked. They take as arguments a method type describing the concatenated arguments count and types, and optionally the String *recipe*, plus the constants that participate in the String concatenation. The details on accepted recipe shapes are described further below. Linkage may involve dynamically loading a new class that implements the expected concatenation behavior. The `CallSite` holds the `MethodHandle` pointing to the exact concatenation method. The concatenation methods may be shared among different `CallSite`s, e.g. if linkage methods produce them as pure functions.
2. *Invocation* occurs when a generated concatenation method is invoked with the exact dynamic arguments. This may occur many times for a single concatenation method. The method referenced by the behavior `MethodHandle` is invoked with the

# It will always go on . . .

**Ismael Juma**
@ijuma

A significant `String.hashCode()` performance improvement from @cl4es has been merged - a micro-benchmarks involving a string with 10k characters shows a 6x(!) improvement. The improvement is not as extreme (but still impressive) for smaller strings.

Tweet übersetzen




1:31 vorm. · 18. Jan. 2023 · **6.444** Mal angezeigt

# 294 String.format() 3x faster in Java 17

Author: Dr Heinz M. Kabutz | Date: 2021-10-29 | Java Version: 17 | Category: Performance

**Abstract:** One of the most convenient ways of constructing complex Strings is with String.format(). It used to be excessively slow, but in Java 17 is about 3x faster. In this newsletter we discover what the difference is and where it will help you. Also when you should use format() instead of the plain String addition with +.

Welcome to the 294th edition of **The Java(tm) Specialists' Newsletter**. We had a lovely run in the rain today, followed by a dip in the sea, clocking in at 21.6 degrees celsius. That is bathwater for someone from Bantry Bay! I remember the water in Cape Town being so cold that our breath misted as my brother and I contemplated how crazy we were to spearfish in single-digit water temperatures - and that was in summer.

**javaspecialists.teachable.com:** Please visit our new self-study course catalog to see how you can upskill your Java knowledge.

## String.format() 3x faster in Java 17

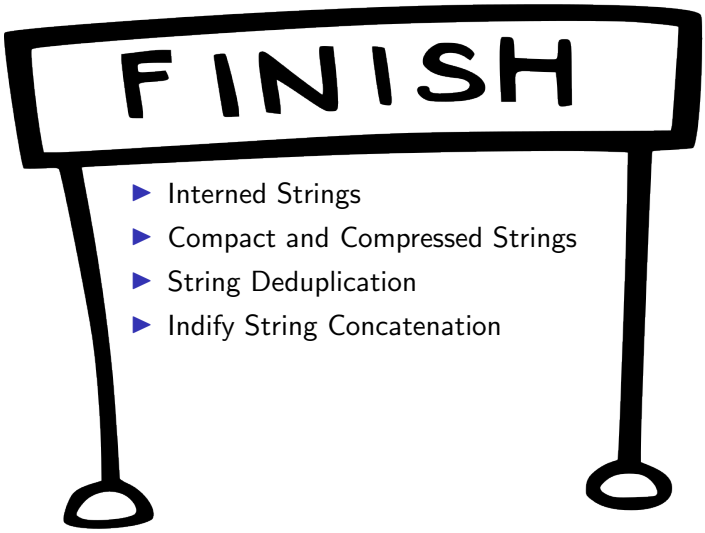
A few years ago, my friend Dmitry Vyazelenko and I submitted a talk to JavaOne, where we spoke for about an hour about the humble `java.lang.String`. We have since spoken about this fundamental class at Devoxx, Geecon, Geekout, JAX, Voxxed Days, GOTO, and various JUGs around the world. Who would have thought that we could easily fill an hour with a talk about `java.lang.String`?

I would usually start the talk by showing a quiz. Which method is the fastest at appending Strings?

```
public class StringAppendingQuiz {
  public String appendPlain(String question,
                            String answer1,
```



## About the Author


FINISH
Interned Strings
Compact and Compressed Strings
String Deduplication
Indify String Concatenation

## Slides and Code

https://github.com/BerndMuller/strings-jcon-2023