Why String Class is made Immutable or Final in Java - 5 Reasons

There is hardly any Java Interview, where no questions are asked from String, and *Why String is Immutable in Java* is I think most popular one. This question is also asked as *Why String class is made final in Java* or simply, *Why String is final*. In order to answer these questions, Java programmer must have a solid understanding of How String works, what are special features of this class and some key fundamentals. The String class is a God class in Java, It has got special features which is not available to other classes e.g. `String` literals are stored in pools, You can concatenate strings using + operator. Given its importance in Java programming, Java designer has made it final, which means you can not extend `java.lang.String` class, this also helps to make `String` object Immutable.

Now coming to questions, Why String is Immutable in Java? of course it should be related to benefits, advantages. Now let's think what are those advantages or features, which drives this decision. I don't know if there is any official document from Oracle or Sun previously, which can throw some light on this decision.

Though I remember reading somewhere, that once asked to James Gosling, creator of Java about making String class final, he has said something along security. It's been argued that making a class final seriously limits its ability to evolve or extend and James has made comment that, classes which are key to Java's Security commitment are made final, so that no one can change its behaviour and game with Java platform.

- string - join
- string - substring
- string - split
- string - palindrome
- string - reverse words
- string - byte array
- string - to enum
- string - compare
- string - empty
- string - stringbuffer
- string - duplicate
- string - immutable
- string - split regex
- string - remove whitespace
- string - toLowerCase
- string - reverse

Categories

- core java (451)
- core java interview question answer (124)
- programming (123)
- Java collection tutorial (111)
- Coding Problems (74)
- interview questions (65)
- data structure and algorithm (53)
- Java 8 (51)
- String (51)
- error and exception (46)
- object oriented programming (41)
- books (39)
- ArrayList (36)
- array (33)

5 Reasons of Why String is final or Immutable in Java

Though true reasons of why String class was made final is best known to Java designers, and apart from that hint on security by James Gosling, I think following reasons also suggest Why String is Final or Immutable in Java.

1) String Pool

Java designer knows that `String` is going to be most used data type in all kind of Java applications and that's why they wanted to optimize from start. One of key step on that direction was idea of storing String literals in String pool. Goal was to reduce temporary `String` object by sharing them and in order to share, they must have to be from Immutable class. You can not share a mutable object with two parties which are unknown to each other. Let's take an hypothetical example, where two reference variable is pointing to same `String` object:

```
String s1 = "Java";
String s2 = "Java";
```

Now if `s1` changes the object from `"Java"` to `"C++"`, reference variable also got value `s2="C++"`, which it doesn't even know about it. By making String immutable, this sharing of String literal was possible. In short, key idea of String pool can not be implemented without making String final or Immutable in Java.

2) Security

Java has clear goal in terms of providing a secure environment at every level of service and String is critical in those whole security stuff. String has been widely used as parameter for many Java classes, e.g. for opening network connection, you can pass host and port as String, for reading files in Java you can pass path of files and directory as String and for opening database connection, you can pass database URL as String. If String was not immutable, a user might have granted to access a particular file in system, but after authentication he can change the PATH to something else, this could cause serious security issues. Similarly, while connecting to database or any other machine in network, mutating String value can pose security threats. Mutable strings could also cause security problem in Reflection as well, as the parameters are strings.

3) Use of String in Class Loading Mechanism

Another reason for making String final or Immutable was driven by the fact that it was heavily used in class loading mechanism. As String been not Immutable, an attacker can take advantage of this fact

- 10 Books Java Programmers Should Rea
- 10 C Fra
- 10 F in 2
- The day
- 10 J Questions
- Top 10 Android Interview Questions for Java Programmers
- 10 Books Every Programmer Should Read
- 5 Great Books to Learn Java 8

Read more

Books and Resources

- Best Book to Learn Java in 2017
- 5 Books to Learn Spring MVC and Core in 2017
- 2 books to learn Hibernate in 2017
- 12 Advanced Java Programming Books for Experienced Programmers
- 5 Free JavaScript Books to download
- 5 Books to Improve Your Coding Skill
- Books Every Programmer Should Read
- Top 10 Computer Algorithm Books
- 10 Free Java Programming Books
- 5 Books to Learn Java 8 Better
- Books Every Programmer Should Read
- Top 10 Computer Algorithm Books

- coding (32)
- Java Multithreading Tutorial (31)
- spring framework (30)
- date and time (23)
- sql (22)
- java io tutorial (21)
- SQL interview Question (20)
- J2EE (18)
- JDBC (18)
- Linux (18)
- java (18)
- JavaScript (16)
- database (16)
- Eclipse (15)
- Java programming Tutorial (15)
- JSP (14)
- Servlet (14)
- thread interview questions (13)
- OCAJP (11)
- binary tree (10)
- Unix (9)
- design pattern (8)
- Java Interview Question (7)
- OCPJP (7)
- java concurrency tutorial (7)
- JSON (6)
- homework (6)
- jQuery (6)
- Hibernate (4)
- Web Service (4)
- Hibernate interview Question (3)
- Java 5 tutorial (3)
- Struts (3)
- debugging (3)
- thread (3)
- Java Enum (2)

and a request to load standard Java classes e.g. `java.io.Reader` can be changed to malicious class `com.unknown.DataStolenReader`. By keeping String final and immutable, we can at least be sure that JVM is loading correct classes.

4) Multithreading Benefits

Since Concurrency and Multi-threading was Java's key offering, it made lot of sense to think about thread-safety of String objects. Since it was expected that String will be used widely, making it Immutable means no external synchronization, means much cleaner code involving sharing of String between multiple threads. This single feature, makes already complicate, confusing and error prone concurrency coding much easier. Because String is immutable and we just share it between threads, it result in more readable code.

5) Optimization and Performance

Now when you make a class Immutable, you know in advance that, this class is not going to change once created. This guarantee open path for many performance optimization e.g. caching. String itself know that, I am not going to change, so String cache its hashcode. It even calculate hashcode lazily and once created, just cache it. In simple world, when you first call hashCode() method of any String object, it calculate hash code and all subsequent call to `hashCode()` returns already calculated, cached value. This results in good performance gain, given String is heavily used in hash based Maps e.g. Hashtable and HashMap. Caching of hashcode was not possible without making it immutable and final, as it depends upon content of String itself.

Pros and Cons of String being Immutable or Final in Java

Apart from above benefits, there is one more advantage that you can count due to String being final in Java. It's one of the most popular object to be used as key in hash based collections e.g. HashMap and Hashtable. Though immutability is not an absolute requirement for `HashMap` keys, its much more safe to use Immutable object as key than mutable ones, because if state of mutable object is changed during its stay inside HashMap, it would be impossible to retrieve it back, given it's `equals()` and `hashCode()` method depends upon the changed attribute. If a class is Immutable, there is no risk of changing its state, when it is stored inside hash based collections. Another significant benefits, which I have already highlighted is its thread-safety. Since String is immutable, you can safely share it between threads without worrying about external synchronization. It makes concurrent code more readable and less error prone.

- garbage collection (2)
- xml (2)

Despite all these advantages, Immutability also has some disadvantages, e.g. it doesn't come without cost. Since String is immutable, it generates lots of temporary use and throw object, which creates pressure for Garbage collector. Java designer has already thought about it and storing String literals in pool is their solution to reduce String garbage. It does help, but you have to be careful to create String without using constructor e.g. `new String()` will not pick object from String pool. Also on average Java application generates too much garbage. Also storing Strings in pool has a hidden risk associated with it. String pool is located in PermGen Space of Java Heap, which is very limited as compared to Java Heap. Having too many String literals will quickly fill this space, resulting in java.lang.OutOfMemoryError: PermGen Space. Thankfully, Java language programmers has realized this problem and from Java 7 onwards, they have moved String pool to normal heap space, which is much much larger than `PermGen` space. There is another disadvantage of making String final, as it limits its extensibility. Now, you just can not extend String to provide more functionality, though more general cases its hardly needed, still its limitation for those who wants to extend `java.lang.String` class.

These 5 reasons definitely gives an hint that **Why String class has been made Final and Immutable in Java**. Of-course it's decision of Java designers but looks like above points contributes to take them this decision. Due to similar reasons wrapper classes like `Integer`, `Long`, `Double` and `Float` are also immutable and Final

Further Learning

Data Structures and Algorithms: Deep Dive Using Java
Java Fundamentals: The Java Language
Complete Java Masterclass

Blog Archive

- ► 2020 (208)
- ► 2019 (148)
- ► 2018 (32)
- ► 2017 (57)
- ► 2016 (109)
- ► 2015 (87)
- ▼ 2014 (46)
  - ► December (6)
  - ► November (4)
  - ► October (2)

► September (4)
► August (3)
► July (1)
► June (3)
► May (3)
► April (4)
► March (4)
► February (5)
▼ January (7)
How HashSet Internally Works in Java
Java Regular Expression to Check If String contain...
How to Fix java.lang.OufOfMemoryError: Direct Buff...
Why String Class is made Immutable or Final in Jav...
10 points about Thread in Java
java.lang.UnsatisfiedLinkError: Library not found ...
6 Difference between LinkedHashSet vs TreeSet vs H...
► 2013 (41)
► 2012 (59)

Posted by javin paul

Labels: core java, core java interview question answer, String

16 comments:

Anonymous January 15, 2014 at 2:31 AM

#3 looks bogus to me, if you have the ability to replace String with an evil twin you might as well do it directly for the IO classes to get your DataStolenReader into the jvm

Reply

Anonymous March 1, 2014 at 3:42 AM

#2 - If thats true, StringBuffer, StringBuilder etc.. should also be made immutable. We use these classes to constrct a text and probably convert it to String and use it.

Reply

Replies

Anonymous May 14, 2015 at 9:34 AM

please refer this
http://stackoverflow.com/questions/47605/string-concatenation-concat-vs-operator
operator + is treated different from .concate method.

Reply

Anonymous January 1, 2015 at 8:40 PM

One advantage of making String immutable is for saving memory. When your program grows the number of String instances it creates also grows and if you don't cache String constants you end up with lots and lots of String in your heap space. By caching and sharing String constants JVM reduces lots of memory for real world Java applications.

Reply

Unknown April 19, 2015 at 12:21 AM

good reason.thanks a lot

Reply

Anonymous September 25, 2016 at 8:42 AM

Can you please elaborate 1st Reason?

Reply

Unknown October 3, 2016 at 9:04 AM

Thanks, good explanation..

Reply

Kenny January 4, 2017 at 8:36 AM

I like your simple language to complex matters!

Reply

Anonymous January 16, 2017 at 8:46 PM

Can you please elaborate 1st Reason?

Reply

Anonymous January 25, 2017 at 1:14 AM

The first point, second paragraph is contradictory with the actual implementation,

Now if s1 changes the object from "Java" to "C++", reference variable also got value s2="C++", which it doesn't even know about it. By making String immutable, this sharing of String literal was possible. In short, key idea of String pool can not be implemented without making String final or Immutable in Java.

S2 will not have value of C++. the reference mapping of S1 is removed from "Java" and mapped to new location with value of C++.

Reply

Replies

Unknown March 4, 2017 at 6:15 AM

No, it is not contradictory.
What he meant is, if the Strings are mutable, by changing Java to C++, both S1 and S2 will affect.
And with immutability in place, the point you said, "S2 will not have value of C++. the reference mapping of S1 is removed from "Java" and mapped to new location with value of C++" holds good.

Reply

Anonymous May 10, 2017 at 2:15 AM

Let's take an hypothetical example, where two reference variable is pointing to same String object:

String s1 = "Java";
String s2 = "Java";

Now if s1 changes the object from "Java" to "C++", reference variable also got value s2="C++", which it doesn't even know about it.

TAKING about this example it that really possible. plz provide the code.

Reply

Replies

javin paul May 10, 2017 at 3:08 AM

It's not possible now because String is Immutable but if it was Mutable and then String was shared from pool then it was possible.

Reply

Unknown May 20, 2017 at 1:07 AM

Can someone please more elaborate on point 5

Reply

Unknown September 25, 2017 at 8:27 AM

Could someone please explain this properly example with a program.

Reply

Sheriff January 27, 2018 at 12:57 AM

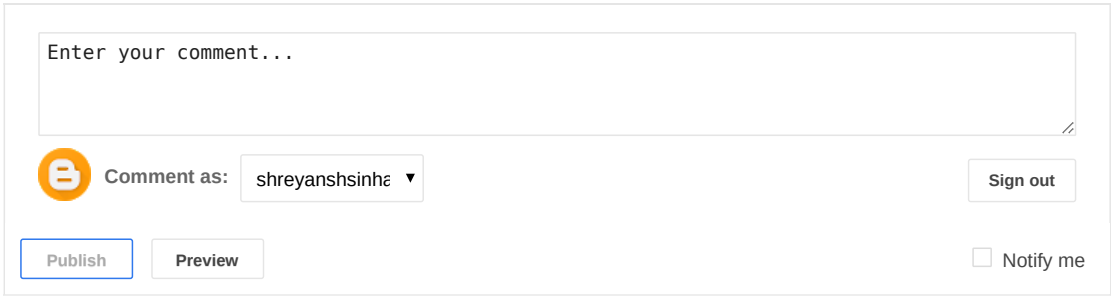
I am trying to understand point #2. Since String ref. variable will be used across the code. String objects are not mutable, but we can assign new malicious value to String reference Variable and hence that new value will be used across the code. e.g.
public class MyApi {
final String myUrl;
public MyApi(String urlString) {
// Verify that urlString points to an approved server
if (!checkApprovedUrl(urlString)) throw new IllegalArgumentException();

```
myUrl = urlString;
}
}
```

In the above code suppose some new value is assigned to urlString say after the if condition and which will be used across the code, thus compromising security. Please explain.

Thanks.

Reply



Newer Post Home Older Post

Subscribe to: Post Comments (Atom)