Difference between StringBuilder and StringBuffer in Java [Answer]

If you are in a hurry and heading straight to interview then I won't take much of your time, In a couple of words, the main difference between StringBuffer and StringBuilder is between four parameters, `synchronization`, `speed`, `thread-safety`, and `availability`. `StringBuffer` is synchronized and that's why thread-safe, but `StringBuilder` is not synchronized, not thread-safe and that's why fast. Regarding availability, `StringBuffer` is available from Java 1.0 while `StringBuilder` was added in Java 5.

Now we can take a breath, and can continue with rest of this article. In Java, there are three classes to deal with String data type, `String`, `StringBuffer` and `StringBuilder`. All of three belongs to `java.lang` package, which is automatically imported into every Java program thus you don't need to do any import for using `StringBuilder` and `StringBuffer`.

Out of these three, String is immutable in Java, while `StringBuilder` and `StringBuffer` are mutable version of String. Some one with little bit of common sense will think that, why do one need three classes for same functionality, why can't `String` just do what `StringBuilder` and `StringBuffer` does.

This definitely would have required keeping String mutable, which is not favourable to Java designers for many reasons, like Security, String Pool, and Performance. So designers of Java API has introduced a pattern of providing a mutable companion class for immutable main class.

Difference between StringBuffer and StringBuilder in Java

Though both of these classes are a mutable versions of String class, there is one significant difference between them, which makes this question interesting and favourite among Java interviewers. More often than not, I see this question coming up in telephonic round of Java interviews or even a couple of rounds of a preliminary face-to-face interview.

1) StringBuffer is Synchronized and StringBuilder is Not

Yes, this is the common answer from a majority of Java developers to this question, which is true, but when asked to elaborate this answer, most of them fail miserably. In order to elaborate, you must know what is meant by synchronized and what changes StringBuilder make to achieve that.

In this case it's pretty straightforward. If you open code of `StringBuffer.java` and `StringBuilder.java` , you can do this in Eclipse IDE by pressing `Ctrl + T` and typing `StringBuffer` for former and `StringBuilder` for later.

This is one of the several useful Eclipse short-cuts discussed in that article. Almost all the methods of StringBuffer e.g. `length()`, `capacity()`, `ensureCapacity()` , `setLength()`, `charAt()` and all

overloaded version of `append()` methods are synchronized, as shown below :

```
public StringBuffer(CharSequence seq) {
        this(seq.length() + 16);
        append(seq);
    }

    public synchronized int length() {
        return count;
    }

    public synchronized int capacity() {
        return value.length;
    }


    public synchronized void ensureCapacity(int minimumCapacity) {
        if (minimumCapacity > value.length) {
            expandCapacity(minimumCapacity);
        }
    }

    /**
     * @since      1.5
     */
    public synchronized void trimToSize() {
        super.trimToSize();

    }
```

This is done to protect instance variable like `count`, `value` (character array which holds the content of StringBuffer) being corrupted by exposing to multiple threads. In fact, every method which touches these variables are made synchronized, which is of course absolutely needed if you want to make `StringBuffer` thread-safe, but as a side effect, this makes `StringBuffer` instances slower.

On the other hand, if you look at `StringBuider` class, you will not find any single synchronized method. If you don't believe me do a search for a synchronized word in `StringBuilder.java` file. All corresponding methods in this class are non-synchronized.

```
public StringBuilder append(char[] str, int offset, int len) {
        super.append(str, offset, len);
        return this;
    }

    public StringBuilder append(boolean b) {
        super.append(b);
        return this;
    }

    public StringBuilder append(char c) {
        super.append(c);
        return this;
    }

    public StringBuilder append(int i) {
        super.append(i);
        return this;
    }

    public StringBuilder append(long lng) {
        super.append(lng);
        return this;
    }

    public StringBuilder append(float f) {
        super.append(f);
        return this;
    }
```

One more thing to note about StringBuffer and StringBuilder is that they inherit from a

common parent class, `AbstractStringBuilder`. This is an abstract class, which also implements `Appendable` and `CharSequence` interface. This was introduced in Java 1.5 to bring `StringBuffer` and `StringBuilder` into the same type hierarchy.

This class represents a mutable sequence of characters. At any point in time, it contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls e.g. `append()`.

StringBuffer overrides almost every method to add synchronized keyword on it, but apparently doesn't use @Override annotation. I really found this amusing because annotation was also introduced on same release but they didn't they could have added those in `StringBuffer` class to make it more readable.

2) Another *difference between StringBuffer and StringBuilder* is that former is `thread-safe` and later is not. You can share instance of StringBuffer between multiple thread without any external synchronization, but its not safe to share instance of `StringBuilder` like that. If you ever need to use a mutable String between multiple thread, go for `StringBuffer`, for all local use, which is within inside one thread, use StringBuilder.

3) Third difference between `StringBuilder` and `StringBuffer` also stem from same fact, because `StringBuffer` is synchronized and thread-safe its bound to be slower than its non-synchronized, non-thread-safe counterpart StringBuilder, because of time taken to acquire and release lock while calling methods of `StringBuffer`.

In short, if you have good knowledge of Java programming language, you can actually deduce all three differences from just one fact that, *StringBuffer is synchronized while StringBuilder is not*. In fact, that will impress interviewer more, because apart from answering the question it also demonstrate your sound knowledge of Java programming language. Always take this kind of opportunities and demonstrate your talent and deep understanding of subject during interviews. It's a proven strategy to do well on interviews.

StringBuider vs StringBuffer

In Summary, the following are the notable difference between StringBuffer and StringBuilder in Java

1) StringBuilder is a non synchronized version of `StringBuffer` class. Methods in `StringBuilder` e.g. all overloaded version of `append()` method is not synchronized.

2) StringBuilder is definitely faster than StringBuffer because of no overhead of acquiring and releasing locks associated with synchronized methods.

3) StringBuffer is thread-safe and StringBuilder is not. You can not share the Instances of the StringBuilder class between multiple threads. If such synchronization is required then it is better to use the StringBuffer class.

4) StringBuffer is an old class, its there in JDK from very first release, while StringBuilder is relatively newer class, introduced much later in the release of JDK 1.5

5) Another interesting fact to know about both of this class is that, when you do `String` concatenation using `+` operator, Java internally convert that call to corresponding `StringBuilder` append() method class. For example "one" + "two" + "three" will be converted to new `StringBuilder().append("one").append("two").append("three")`. Only problem is that it initializes `StringBuilder` with default capacity, which means expensive array copy operation when `StringBuilder` get resized.

That's all about the **difference between StringBuffer and StringBuilder in Java**. You just can't afford to miss this question. The good thing about this question is that it provides you an opportunity to show your knowledge about how synchronization, thread-safety, and speed are related to each other. It also demonstrates your sound knowledge of Java API and how much you know about its evolution in every version, but the most important thing is to learn when to use StringBuilder and StringBuffer class.

Since every Java program makes extensive use of both immutable and mutable String, you must know the difference between `StringBuilder` and `StringBuffer` to make right

use of them. By default, you should always use StringBuilder and only consider or StringBuffer when you see that a mutable string must be shared between multiple threads