



## Java Foundations

# Working with Character Data

## Copyright

Introduction to Java Programming by Eric Matthews is licensed under a Creative Commons Attribution 3.0 Unported License. This license allows you to

- Copy, distribute and transmit the work
- Adapt the work
- Make commercial use of the work

Under the following conditions:

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work

With the understanding that:

**Waiver** — Any of the above conditions can be waived if you get permission from the copyright holder.

**Public Domain** — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

**Other Rights** — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

**Notice** — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to one or more of the following web pages.

[www.DeveloperGeekResources.com](http://www.DeveloperGeekResources.com)

[www.EducationAnytime.com](http://www.EducationAnytime.com)

Table of Contents

*WORKING WITH STRINGS IN JAVA*.....4

## WORKING WITH STRINGS IN JAVA

In another lesson I introduced you to using the String class which has a number of useful methods for string manipulation. However, for gathering data from a UI, and for scenarios in large systems the String class is not typically used. Instead the StringBuffer class is used. Why?

This is because the String class is immutable. This means that once you set the data in an instanced string class that it cannot be changed.

Wait a minute you argue. I can change data in a String and it works just fine. For example,

```
String s = new String("Foo");  
s = "Bar";
```

...works just fine. What are you talking about?

True you can change the data in your object named "s". However, each time you do so the object needs to be re-instanced! Consider the following example:

**Source:** string\_stringbuffer1.java

```
public class string_stringbuffer1 {  
  
    public static void main(String[] args) {  
        String s = new String("foobar string");  
        StringBuffer sb = new StringBuffer("foobar stringbuffer");  
  
        System.out.println("Instance and set initial value for String and StringBuffer  
classes \n");  
        System.out.println("        String hashCode: " + s.hashCode() +  
                            "\n StringBuffer hashCode: " + sb.hashCode() + "\n");  
        s = "wth? s";  
        sb.append("wth? sb;");  
        System.out.println("Modify String and StringBuffer class values \n");  
        System.out.println("        String hashCode: " + s.hashCode() +  
                            "\n StringBuffer hashCode: " + sb.hashCode() + "\n");  
        s = "yada? s";  
        sb.append("yada? sb;");  
        System.out.println("Again, modify String and StringBuffer class values \n");  
        System.out.println("        String hashCode: " + s.hashCode() +  
                            "\n StringBuffer hashCode: " + sb.hashCode());  
    }  
}
```

```

C:\javatoycode>java string_stringbuffer1
Instance and set initial value for String and StringBuffer classes
    String hashCode: 1230171652
    StringBuffer hashCode: 17523401
Modify String and StringBuffer class values
    String hashCode: -777809977
    StringBuffer hashCode: 17523401
Again, modify String and StringBuffer class values
    String hashCode: -1408873171
    StringBuffer hashCode: 17523401
C:\javatoycode>

```

Without getting too much into the technical details, the data for Java objects gets stored into an associative array. All Java classes inherit from the Java class named Object. The Object class has a method named hashCode() that allows us to return the key where the data for an object is stored. As you can see in the StringBuffer class the hash key stays the same when I modify the data. However, each time I change the data for my String class I end up with a new hash key. This is what is meant when we say that String is an immutable class. There is certainly overhead involved in deleting and creating a new hash key with each update.

**Source:** string\_stringbuffer2.java

```

import java.util.Calendar;
public class string_stringbuffer2 {

    public static void main(String[] args) {

        String s = new String("foobar string");
        StringBuffer sb = new StringBuffer("foobar stringbuffer");

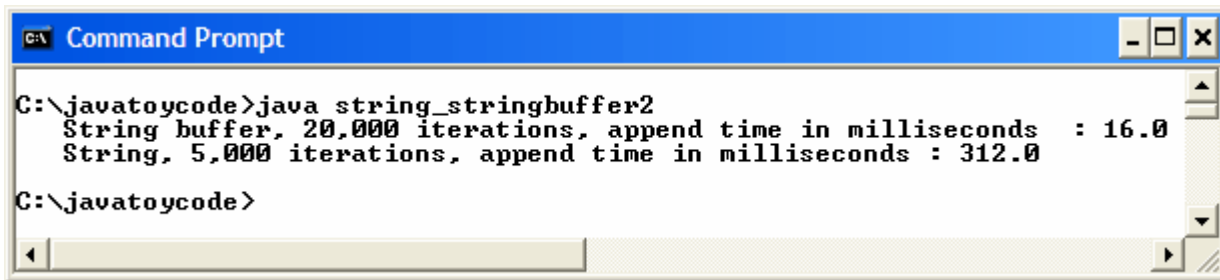
        double T1a = 0;
        double T1b = 0;
        double T1t = 0;

        double T2a = 0;
        double T2b = 0;
        double T2t = 0;

        Calendar cal3 = Calendar.getInstance();
        T1a = (double)cal3.getTimeInMillis();
        for (int i=1; i<20000; i++) {
            sb.append(i);
        }
        Calendar cal4 = Calendar.getInstance();
        T1b = (double)cal4.getTimeInMillis();
        T1t = (T1b - T1a);
        System.out.println("String buffer, 20,000 iterations, append time in
milliseconds: " + T1t);

```

```
        Calendar cal = Calendar.getInstance();
        T2a = (double)cal.getTimeInMillis();
        for (int i=1; i<5000; i++) {
            s += i;
        }
        Calendar cal2 = Calendar.getInstance();
        T2b = (double)cal2.getTimeInMillis();
        T2t = (T2b - T2a);
        System.out.println("String, 5,000 iterations, append time in
milliseconds: " + T2t);
    }
}
```



The following program represents a rather crude attempt to demonstrate the performance differences using the String and the StringBuffer classes. There are many factors involving performance. The goal here is not to address these factors, rather it is to point out the dramatic performance gain in using StringBuffer instead of String. In fact, I needed to ratchet up the iterations to 20,000 to even get the StringBuffer to register a value using my crude millisecond timer hack.

Now covering this subject can be a slippery slope. There are so many factors involved in performance. One has to take many runtime and development factors into account when deciding which class to use and why. The point I wish to make here is not to stop using the String class. For one, this class has many useful methods like Trim and Substring which the StringBuffer class does not have.