Documentation

The Java™ Tutorials

Trail: Learning the Java Language
Lesson: Numbers and Strings
Section: Strings

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

The StringBuilder Class

`StringBuilder` objects are like `String` objects, except that they can be modified. Internally, these objects are treated like variable-length arrays that contain a sequence of characters. At any point, the length and content of the sequence can be changed through method invocations.

Strings should always be used unless string builders offer an advantage in terms of simpler code (see the sample program at the end of this section) or better performance. For example, if you need to concatenate a large number of strings, appending to a `StringBuilder` object is more efficient.

Length and Capacity

The `StringBuilder` class, like the `String` class, has a `length()` method that returns the length of the character sequence in the builder.

Unlike strings, every string builder also has a *capacity*, the number of character spaces that have been allocated. The capacity, which is returned by the `capacity()` method, is always greater than or equal to the length (usually greater than) and will automatically expand as necessary to accommodate additions to the string builder.

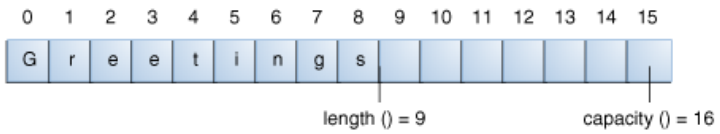
`StringBuilder` Constructors

Constructor	Description
`StringBuilder()`	Creates an empty string builder with a capacity of 16 (16 empty elements).
`StringBuilder(CharSequence cs)`	Constructs a string builder containing the same characters as the specified `CharSequence`, plus an extra 16 empty elements trailing the `CharSequence`.
`StringBuilder(int initCapacity)`	Creates an empty string builder with the specified initial capacity.
`StringBuilder(String s)`	Creates a string builder whose value is initialized by the specified string, plus an extra 16 empty elements trailing the string.

For example, the following code

```
// creates empty builder, capacity 16
StringBuilder sb = new StringBuilder();
// adds 9 character string at beginning
sb.append("Greetings");
```

will produce a string builder with a length of 9 and a capacity of 16:



The `StringBuilder` class has some methods related to length and capacity that the `String` class does not have:

Length and Capacity Methods

Method	Description
`void setLength(int newLength)`	Sets the length of the character sequence. If `newLength` is less than `length()`, the last characters in the character sequence are truncated.

	If newLength is greater than length(), null characters are added at the end of the character sequence.
void ensureCapacity(int minCapacity)	Ensures that the capacity is at least equal to the specified minimum.

A number of operations (for example, append(), insert(), or setLength()) can increase the length of the character sequence in the string builder so that the resultant length() would be greater than the current capacity(). When this happens, the capacity is automatically increased.

StringBuilder Operations

The principal operations on a StringBuilder that are not available in String are the append() and insert() methods, which are overloaded so as to accept data of any type. Each converts its argument to a string and then appends or inserts the characters of that string to the character sequence in the string builder. The append method always adds these characters at the end of the existing character sequence, while the insert method adds the characters at a specified point.

Here are a number of the methods of the StringBuilder class.

Various StringBuilder Methods

Method	Description
StringBuilder append(boolean b) StringBuilder append(char c) StringBuilder append(char[] str) StringBuilder append(char[] str, int offset, int len) StringBuilder append(double d) StringBuilder append(float f) StringBuilder append(int i) StringBuilder append(long lng) StringBuilder append(Object obj) StringBuilder append(String s)	Appends the argument to this string builder. The data is converted to a string before the append operation takes place.
StringBuilder delete(int start, int end) StringBuilder deleteCharAt(int index)	The first method deletes the subsequence from start to end-1 (inclusive) in the StringBuilder's char sequence. The second method deletes the character located at index.
StringBuilder insert(int offset, boolean b) StringBuilder insert(int offset, char c) StringBuilder insert(int offset, char[] str) StringBuilder insert(int index, char[] str, int offset, int len) StringBuilder insert(int offset, double d) StringBuilder insert(int offset, float f) StringBuilder insert(int offset, int i) StringBuilder insert(int offset, long lng) StringBuilder insert(int offset, Object obj) StringBuilder insert(int offset, String s)	Inserts the second argument into the string builder. The first integer argument indicates the index before which the data is to be inserted. The data is converted to a string before the insert operation takes place.
StringBuilder replace(int start, int end, String s) void setCharAt(int index, char c)	Replaces the specified character(s) in this string builder.
StringBuilder reverse()	Reverses the sequence of characters in this string builder.
String toString()	Returns a string that contains the character sequence in the builder.

Note: You can use any String method on a StringBuilder object by first converting the string builder to a string with the toString() method of the StringBuilder class. Then convert the string back into a string builder using the StringBuilder(String str) constructor.

An Example

The StringDemo program that was listed in the section titled "Strings" is an example of a program that would be more efficient if a StringBuilder were used instead of a String.

StringDemo reversed a palindrome. Here, once again, is its listing:

```
public class StringDemo {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";
        int len = palindrome.length();
        char[] tempCharArray = new char[len];
```

```
            // put original string in an
            // array of chars
            for (int i = 0; i < len; i++) {
                tempCharArray[i] =
                    palindrome.charAt(i);
            }

            // reverse array of chars
            for (int j = 0; j < len; j++) {
                charArray[j] =
                    tempCharArray[len - 1 - j];
            }

            String reversePalindrome =
                new String(charArray);
            System.out.println(reversePalindrome);
        }
    }
```

Running the program produces this output:

```
doT saw I was toD
```

To accomplish the string reversal, the program converts the string to an array of characters (first `for` loop), reverses the array into a second array (second `for` loop), and then converts back to a string.

If you convert the `palindrome` string to a string builder, you can use the `reverse()` method in the `StringBuilder` class. It makes the code simpler and easier to read:

```
public class StringBuilderDemo {
    public static void main(String[] args) {
        String palindrome = "Dot saw I was Tod";

        StringBuilder sb = new StringBuilder(palindrome);

        sb.reverse();  // reverse it

        System.out.println(sb);
    }
}
```

Running this program produces the same output:

```
doT saw I was toD
```

Note that `println()` prints a string builder, as in:

```
System.out.println(sb);
```

because `sb.toString()` is called implicitly, as it is with any other object in a `println()` invocation.

Note: There is also a `StringBuffer` class that is *exactly* the same as the `StringBuilder` class, except that it is thread-safe by virtue of having its methods synchronized. Threads will be discussed in the lesson on concurrency.

Previous page: Comparing Strings and Portions of Strings
Next page: Summary of Characters and Strings