String Builders

The StringBuilder Class

`String` objects are like `StringBuilder` 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. Prior to Java SE 9, if you need to concatenate a large number of strings, appending to a StringBuilder object may be more efficient. String concatenation has been optimized in Java SE 9, making concatenation more efficient than `StringBuilder` appending.

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.

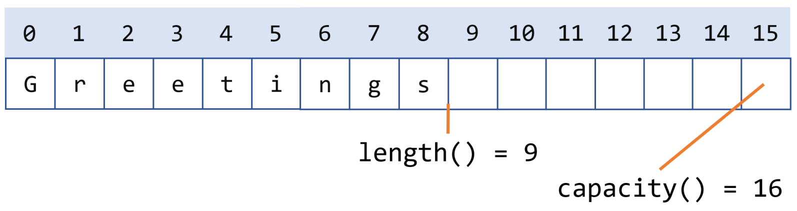
You can use the following constructors of the `StringBuilder` class:

- `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:



Length and capacity of a `StringBuilder`

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

- `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.

- You can append any primitive type or object to a string builder with an `append()` method. The data is converted to a string before the append operation takes place.
- The `delete(int start, int end)` method deletes the subsequence from `start` to `end - 1` (inclusive) in the `StringBuilder`'s char sequence.

- You can delete the `char` at index `index` with the method `deleteCharAt(int index)`.
- You can insert any primitive type or object at the given `offset` with one of the `insert(int offset)` methods. These methods take the element to be inserted as a second argument. The data is converted to a string before the insert operation takes place.
- You can replace characters with the methods `replace(int start, int end, String s)` and `setCharAt(int index, char c)`.
- You can reverse the sequence of characters in this string builder with the `reverse()` method.
- You can return a string that contains the character sequence in the builder with the `toString()` method.

> *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 string)` constructor.*

StringBuilder in Action

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];
        char[] charArray = new char[len];
```

```java

        // 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:

```java
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. Unless you absolutely need a thread-safe class, you do not need to use `StringBuffer`.*

Last update: *September 14, 2021*

Strings ← **String Builders** → Autoboxing and Unboxing

Home > Tutorials > Numbers and Strings > String Builders