**CSCI 2302**

**Object-Oriented Thinking Chapter**

**StringBuilder Lab**

Contents

[Intro 1](#_Toc95048877)

[Learning Goals 1](#_Toc95048878)

[Notes 1](#_Toc95048879)

[When to use what 1](#_Toc95048880)

[The difference of length & capacity 2](#_Toc95048881)

[StringBuilder Operations 3](#_Toc95048882)

[Task 4](#_Toc95048883)

[Submit 6](#_Toc95048884)

Intro: Java’s StringBuilder class differs from Java’s String class in that this class is mutable. It is similar by having a length method that returns how many characters in that StringBuilder, but it also has a capacity method that returns how much memory is available for that StringBuilder. This extra memory space allows for the manipulation. As the StringBuilder’s content grows, the capacity grows. The Java StringBuilder class has over 30 methods for manipulation; there are methods that allow content to be appended to the end of the StringBuilder, methods that allow the characters to be inserted, replaced, or deleted within the StringBuilder, and a method that allows the StringBuilder content to be reversed.

Learning Goals: To add to the Java knowledge and apply this knowledge by instantiating StringBuilder objects and utilizing some of the methods within this class.

Notes:

StringBuilder & StringBuffer are more flexible than String because they can be modified.

* Can insert
* Can append
* Can delete characters
* Can replace characters

StringBuilder is more efficient if it is accessed by just a single task

StringBuffer is synchronized (only one task is allowed to execute methods) = used in concurrent programming – multiple tasks running concurrently = multithreading & parallel

When to use what – a String, a StringBuilder, or a StringBuffer

* String – when the object is not to be changed \*\* it is more efficient (in memory storage)
  + If the string is modified over & over then unused objects are left in memory waiting for the garbage collector
* StringBuilder – when a single program is using the string
  + Are not thread safe
* StringBuffer – when there is multithreading / parallel access to the string

the methods (except setCharAt) do 2 things

1. Changes the content of the string builder/buffer
2. Returns the reference of the string builder/buffer

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.

\*\*\* Internally a string builder is an array of characters

### The difference of length & capacity

* Length = the actual size of the string stored in the builder
* Capacity = the current size of the builder
  + The builder's capacity is automatically increased if more characters are added to exceed its capacity
  + The size of the array
    - If the size of the array *– the capacity -* is exceeded, the array is replaced with a new array
      * New array = 2 \* (the previous array size + 1)

|  |  |
| --- | --- |
| **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

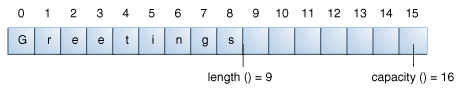
// creates empty builder, capacity 16

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.

Examples:

// capacity and length of StringBuilders

// default StringBuilder instantiated

StringBuilder sb = new StringBuilder();

System.out.println("sb constructed is " + sb);

// sb's capacity is 16 - default capacity size

System.out.println("sb's capacity is " + sb.capacity());

// sb's length is 0 - empty

System.out.println("sb's length is " + sb.length());

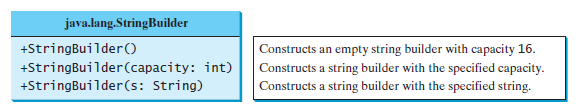
Sb.append("Adding to the StringBuilder");

Task: Complete the steps outlined below in a file named YourMySFAusername\_StringBuilderLab.java.

1. Declare and assign a String the value of *affect*.
2. Use the following method to display the memory location of the affect string: System.identityHashCode(stringIdentifier) to the screen.



1. Reassign the value to *effect* and display the memory location to the screen again.
2. Instantiate a StringBuilder object with the value of *affect*.



1. Use the following method to display the memory location of the StringBuilder affect: System.identityHashCode(stringBuilderIdentifier) to the screen.
2. Use the replace or the setCharAt method to modify affect to *effect* and display the memory location to the screen again.







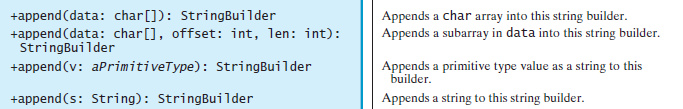
1. Display how many characters are in the StringBuilder to the screen.



1. Use the capacity method to see how much memory is currently allocated to the StringBuilder, display the results to the screen.



1. Use the append method to add the following: *is the result of change*.





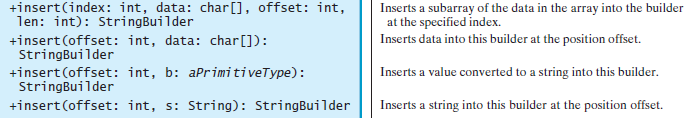
1. Use the length and capacity methods again and display the results to the screen.
2. Use the trimToSize method and repeat the length and capacity methods displaying the results to the screen.



1. Use the toString method on the StringBuilder and display the memory location of the string.



1. Instantiate a StringBuilder with the value of *Stephen*.
2. Use the insert method to insert *n F. Austi* at index 6.



Submit: Submit your YourMySFAusername\_StringBuilderLab.java file in the Dropbox in Brightspace by D2L.