Java – MyStrBuilder

Purpose

This lab was designed to solidify your understanding of class design (constructors, overloading, overriding, accessors, mutators) and array processing.

Problem Description

The String, StringBuilder, and StringBuffer classes are provided by the Java library. You'll be coding your own String class based on the following UML diagram. You are **NOT** allowed to use any of Java's String classes. Doing so will result in a failing grade. No exceptions except in the tester. MyStrBuilder is a mutable class that allows for concatenation, deletion, and insertion of character arrays (Strings).

MyStrBuilder
-data: char[] -size: int
+MyStrBuilder() +MyStrBuilder(capacity: int) +MyStrBuilder(chars: char[]) +append(c: char) : void +append(n: int) : void +append(chars: char[]) : void +charAt(i: int) : char +capacity(): int +length(): int +getData(): char[] +substring(begin: int, end: int) : MyStrBuilder +substring(begin: int) : MyStrBuilder +toLowerCase() : MyStrBuilder +toUpperCase() : MyStrBuilder +equals(obj: Object) : boolean +indexOf(c: char) : int +reverse(): void +insert(offset: int, c: char): void +insert(offset: int, chars: char[]): void +delete(start: int, end: int) : void +delete(index: int) : void +replace(start: int, chars: char[]) : void

+ and - : indicates the method/field is public and private respectively

The methods will have the same functionality as the corresponding methods in the String and the StringBuilder class. You should consult the API and read the descriptions below.

You **should** test your code exhaustively outside of MyStrBuilder.java(use a tester/runner).

+MyStrBuilder() : Constructor that creates an empty MyStrBuilder with a capacity of 16 (16 empty elements). The size is zero. The default value for char is '\u0000' which represents null. This is not a printable character so depending on your system you may see a space or box if you try to print it.

+MyStrBuilder(capacity: int) : Constructor that creates an empty MyStrBuilder with an array capacity set to the provided argument.

+MyStrBuilder(chars: char[]) : Constructor that creates a MyStrBuilder whose value is initialized by the specified char array, plus an extra **16** empty elements trailing the array. Update size appropriately.

+append(c: char) : Appends the character to this string builder. If the builder is full prior to appending, create a new buffer that is 2 more than twice the capacity and copy the elements over. Given a buffer has a capacity of 16 doubling it would go from 16 to 34, from 34 to 70, 70 to 142, and so on.

+append(n: int) : Appends each digit of n to this string builder. It is possible the buffer doubles (plus 2) multiple times.

```
MyStringBuilder demo = new MyStringBuilder(2);     // capacity of 2

demo.append(Integer.MIN_VALUE);                    // -2_147_483_648
```

The demo buffer will now have a size of 11 and a capacity of 14 (2+2*2==6, then 2+2*6==14). Remember you're **not allowed** to use Strings or any similar classes and methods from the Java API. You must code this yourself (most challenging part of the project).

0	1	2	3	4	5	6	7	8	9	10	11	12	13
-	2	1	4	7	4	8	3	6	4	8			

+append(chars: char[]) : Appends the argument to this string builder. For example,

```
// create an empty builder, with default capacity of 16

MyStringBuilder msb = new MyStringBuilder();

// add 4 characters to the beginning

msb.append(new char[]{'J','a','v','a'});

// produces a MyStringBuilder with a size 4 and a capacity of 16.
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a												

```
msb.append(' ');  // add a space

msb.append(16); // add the number 16
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a		1	6									

char[] ary = {' ', 'i', 's', ' ', 't', 'h', 'e', ' ', 'l', 'a', 't', 'e', 's', 't', ' ', 'v', 'e', 'r', 's', 'i', 'o', 'n'};

msb.append(ary);

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
J	a	v	a		1	6		i	s		t	h	e		l	a	t	e	s	t		v	e	r	s	i	o	n					

Take note that the size is 29 and the capacity is 34. If the data buffer/builder is full and you have more characters to append, then you must create a new buffer that is 2 more than twice the capacity and copy the elements over. Only double the buffer if you have more characters to append and there are no empty slots. The buffer can be full.

+charAt(i: int) : return the character at the specified index. An index ranges from 0 to length() – 1 inclusive. If an invalid index is provided throw an exception with the following code:

```
throw new IndexOutOfBoundsException();
```

+capacity() : returns the capacity of the builder as an int

+length() : return the size (the number of characters that have been appended to the builder).

+getData() : accessor method that returns a char[]. THIS METHOD WILL BE USED FOR TESTING - DO NOT FORGET TO ADD IT.

+substring(begin: int, end: int) : returns a MyStrBuilder object with a size of (end – begin) and a capacity 16 greater than size where begin is inclusive and end is exclusive (identical to Java's String class). If begin (begin < 0) or end (end > size) are invalid indices or the difference is less than zero, throw an IndexOutOfBoundsException as outlined above in charAt(). This method **does not** modify the current buffer.

```
MyStringBuilder msb = new MyStringBuilder(4);          // capacity 4

msb.append(new char[]{'J','a','v','a'});               // capacity 4 and size 4

MyStringBuilder msbSub = msb.substring(1,3);           // capacity 18 and size 2 (msbSub)

// msb buffer depicted below
```

0	1	2	3
J	a	v	a

// msbSub buffer depicted below

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	v																

+substring(begin: int) : returns a MyStrBuilder object from inclusive to the last character in the buffer. Throw an exception if begin is invalid. This behavior is identical to the overloaded method above. Do not modify the current buffer.

+toLowerCase() : returns a MyStrBuilder object with the same size and a capacity 16 greater than size but where all upper case letters have been converted to lower case. Do not modify the buffer.

```
MyStringBuilder msb = new MyStringBuilder(10);          // capacity 10 and size 0

msb.append(new char[]{'J','a','v','a'});                // capacity 10 and size 4

MyStringBuilder msbLower = msb.toLowerCase();           // capacity 20 and size 4 (msbLower)
```

+toUpperCase() : returns a MyStrBuilder object with the same size and a capacity 16 greater than size but where all lower-case letters have been converted to upper case. Do not modify the buffer.

+equals(obj: Object) : this method is being overridden from the parent class Object. First type cast the argument into a MyStrBuilder.

```
MyStrBuilder other = (MyStrBuilder)other;
```

Now check if they have the same sequence of characters. The capacity is irrelevant, but the size and all non-null characters must be identical to return true. Return false otherwise.

```
// create two MyStringBuilders

MyStringBuilder msb1 = new MyStringBuilder();      // capacity 16

msb1.append(new char[]{'J','a','v','a'});

MyStringBuilder msb2 = new MyStringBuilder(10);   // capacity 10

msb2.append(new char[]{'J','a','v','a'});

out.println(msb1.equals(msb2));   // outs true, both have size 4 and Java in the buffer

msb1.append(4);

out.println(msb1.equals(msb2));   // outs false
```

+indexOf(c: char) : returns the index of the first occurrence of the character in the character sequence represented by this object, or -1 if the character does not occur (identical to Java's String class).

+reverse(): Reverses the sequence of characters in this string builder.

```
MyStringBuilder msb = new MyStringBuilder();

msb.append(new char[]{'J','a','v','a'});
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a												

```
msb.reverse();          // notice this modifies the buffer, and the return type is void
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

a	v	a	J												

+insert(offset: int, c: char): Inserts the char argument into this sequence at the specified offset. The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence or a IndexOutOfBoundsException will occur. If the buffer is full, create a new buffer that is 2 more than twice the capacity as outlined in append.

MyStringBuilder msb = new MyStringBuilder();

msb.append(new char[]{'J','a','v','a'})

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a												

msb.insert(1, 'J');

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	J	a	v	a											

+insert(offset: int, chars: char[]): Inserts the char character sequence into this sequence at the specified offset. The offset argument must be greater than or equal to 0, and less than or equal to the length of this sequence or a IndexOutOfBoundsException will occur. Provided the offset is valid, double (plus 2) the buffer as many times as needed to insert the char sequence.

MyStringBuilder msb = new MyStringBuilder();

msb.append(new char[]{'K','n','o','w', ' ','p','o','w', 'e','r'});

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K	n	o	w		p	o	w	e	r						

msb.insert(4, new char[]{'l', 'e', 'd', 'g', 'e', ' ', 'i', 's'});

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
K	n	o	w	l	e	d	g	e		i	s		p	o	w	e	r																

+delete(index: int) : deletes the character at the specified index. Throw an IndexOutOfBoundsException if the index is invalid. Delete removes the character so each character after the index will have to be shifted 1 to the left. Capacity doesn't change.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
J	a	v	a												

MyStringBuilder msb = new MyStringBuilder(new char[]{'J','a','v','a'});

msb.delete(2);

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

J	a	a													

+delete(start: int, end: int) : deletes the values from the buffer from the specified range where start is inclusive, and end is exclusive. Start and end must be valid [0, size] and start must be less than or equal to end, otherwise throw an exception. Capacity doesn't change.

+replace(start: int, chars: char[]) : replaces the characters at the specified index with the character sequence. Start must be [0, size] or throw an exception. It's permissible for the char sequence to be longer than the buffer can accommodate at which point the buffer will be increased by 2 more than twice the capacity.

Style

As always, use good style in your classes and methods. Appropriately use control structures like loops and if/else statements. Avoid redundancy using techniques such as methods, loops, and factoring common code out of if/else statements. Properly use indentation, good variable names, and types. Do not have any lines of code longer than 80 characters.

Avoid Redundancy

Create "helper" method(s) to capture repeated code. Provided all extra methods you create are private (so outside code cannot call them), you can have additional methods in your class beyond those specified here. If you find that multiple methods in your class do similar things, you should create helper method(s) to capture the common code.

Commenting

You should comment your code with a heading at the top of your class with your name, section, and a description of the overall program. All method headers should be commented as well as all complex sections of code. Make sure you describe complex methods inside methods. Comments should explain each method's behavior, parameters, return values, and assumptions made by your code, as appropriate. Write descriptive comments that explain error cases, and details of the behavior that would be important to the client. Your comments should be written in your own words and not taken verbatim from this document.

Data Fields

Properly encapsulate your objects by making data fields private. Avoid unnecessary fields; use fields to store important data of your objects but not to store temporary values only used in one place. Fields should always be initialized inside a constructor or method, never at declaration.

Development Strategy

Code the easiest methods first and write test cases as you go. append(int) and insert() are the hardest so I recommend doing them last. Do not wait till the end to test your code. Code a method and test it. Testing is very time consuming so plan for it accordingly. If a method isn't working, try placing a debug println() statement in your method. You may wish to write a print method inside your tester that outputs the contents (size, capacity, the String and array) of the MyStrBuilder. Arrays.equals() may be used for testing equality of two buffers.

System.arraycopy() and Arrays.copyOf() may be used to copy elements or you can utilize a loop. You may wish to decompose your test cases by writing static methods inside of MyStrBuilderTester.java and invoking them from main.

```
public static void testAppend(){
    MyStrBuilder test01 = new MyStrBuilder(4);
    test01.print();
    test01.append("abc".toCharArray());
    test01.print();
    test01.append("d".toCharArray());
    test01.print();
    test01.append("efgh".toCharArray());
    test01.print();
    // add a barrage of test cases testing the append method
}
```

You are permitted but not required to use java.lang.String in the tester program. Given

```
char[] chars = {'1','2','3'};
```

You can create a String from a character array by:

```
String str = new String(chars);
```

Create a character array from a String by:

```
char[] ary = str.toCharArray();
```

And copy the elements of an array by invoking the copyOf method from the Arrays utility class:

```
MyStrBuilder instance = new MyStrBuilder("Test".toCharArray());
char[] copy = Arrays.copyOf(instance.getData(), instance.length());
```

Remember **no String class** of any kind is allowed in MyStrBuilder. Only the tester.

The characters beyond size and up to capacity are irrelevant. Any character in those indices is acceptable. Arrays.fill() may be used to fill the buffer if this makes testing and debugging easier for you. The grader will check the capacity and the contents from zero inclusive to size exclusive. The characters beyond size will not be evaluated.

Visualize the process prior to writing any code by using pen and paper and drawing the buffer with the indices. It helps.

Grading

85% of your grade will be based on implementing the methods/constructors as described. Each method will have multiple test cases and each test case will be graded as right or wrong. Code that does not compile will lose all 85 of these points. Be sure to remove or comment out all extraneous output from your code. Output should only be done in the tester. None of the methods from the MyStrBuilder.java API print anything to the console. Extraneous output will incur a letter grade penalty (minus 10% overall). Please remove or comment out any package declarations prior to submission (if you have them). The remaining 15% of your grade is based on comments, style, code eloquence, and structure as outlined in this document.