STRING METHODS

The Java language distinguishes between two fundamental types of variables. The "primitive" variables we have covered so far include `int`, `double`, and `boolean`. Another variable type includes `String`. But String is different than the other types we have learned about because a String is a JAVA OBJECT with some special rules. One aspect of working with Objects is that they have methods that allow you to retrieve information about them (often referred to as "ACCESSOR" or "GETTER" methods) as well as methods that let you manipulate them (often referred to as "SETTER" methods).

The String class includes only getter methods because it is a special kind of object. A String in JAVA is IMMUTABLE. An immutable object is an object that, once created cannot be altered!

This document will cover primarily only those String methods that are in the AP Subset. Other String methods may be useful, so it is worthwhile to investigate them all, but your focus should be the use of the methods covered here.

When declaring a String one usually gives it an initial value like other primitive variables:

```
String name="Mr. Braskin";
```

You can also give a String an "EMPTY" value using the declaration:

```
String str="";
```

Several methods can be used to gather information about a `String`. The three methods you must know for the AP Exam are:

`.lenght()` – This method returns an integer value equal to the number of characters in the `String`.

`.equals(String anotherString)` – This method returns the `boolean` value `true` if each character in `anotherString` is the same as each character in the `String` you are testing.

`.indexOf(String anotherString)` – This method returns an integer value representing the point in the `String` you are testing where `anotherString` starts. If `anotherString` isn't part of the String you are testing, then it returns `-1`.

`.substring(int from)` – This method returns a String that is the characters from the index `from` to the end of the string.

`.substring(int from, int to)` – This method returns a String that is the characters from the index `from` up to but not including the character at index `to`.

Below are samples of the first three methods applied to `name` and `str`:

Method Name	Variable Name	
	`name`	`str`
`.length()`	11	0
`.indexOf("Braskin")`	4	-1
`.indexOf("Aaron")`	-1	-1
`.equals(str)`	false	true

There is also one primary method for returning a portion of a `String`. There are two versions of the method because it is an OVERLOADED method. An overloaded method may be called using different combinations of parameters but the same name.

This method returns a portion of the String you are using the method upon beginning with the character given as an integer where the 1st character is 0 and the last character is `.length()-1`.

If we think of a `String` as having an index for each character, `name` looks like this:

Character Index:	0	1	2	3	4	5	6	7	8	9	10	`name.length()=11`
String name:	M	r	.		B	r	a	s	k	i	n	

The .substring(characterIndex) method returns a String with the first character from characterIndex, until the end of the String. So:

Method call	String Result
name.substring(0)	Mr. Braskin
name.substring(4)	Braskin
name.substring(9)	in
name.substring(11)	"" (SPECIAL NOTE: returns a String with nothing in it, like empty.)
name.substring(12)	java.lang.StringIndexOutOfBoundsException: String index out of range: -1

The other form of the substring method lets one specify the beginIndex and endIndex. The beginIndex works just like the other version, but the endIndex character is NOT returned. Look at the charts below for examples using name:

Character Index:	0	1	2	3	4	5	6	7	8	9	10	name.length()=11
String name:	M	r	.		B	r	a	s	k	i	n	

Using the substring method on name we can see:

substring method	result
name.substring(0,0)	"" (SPECIAL NOTE: returns a String with nothing in it, like str.)
name.substring(0,1)	M
name.substring(0,3)	Mr.
name.substring(4,11)	Braskin
name.substring(6,8)	as
name.substring(0,name.length())	Mr. Braskin
name.substring(4, 3)	java.lang.StringIndexOutOfBoundsException: String index out of range: -1

Now let us write a simple program that takes a String variable and outputs it to the console one character at a time, replacing vowels with "*":

```
01 public class ReplaceLetters {
02    public static void main(String[] args) {
03       String name="The quick brown fox jumps over the lazy dog.";
04       for (int letterIndex=0; letterIndex<name.length(); letterIndex++) {
05          String l=name.substring(letterIndex, letterIndex+1);
06          if (letter.equals("a") || letter.equals("e") || letter.equals("i")
--                || letter.equals("o") || letter.equals("u")) {
07             letter="*";
08          }
09          System.out.print(letter);
10       }
11    }
12 }
```

Enter the program above and see if you get the following console output:

```
Th* q**ck br*wn f*x j*mps *v*r th* lazy d*g.
```

Looking at the code:

- Line 03: Assigns the String name a value.
- Line 04: Sets up a loop that iterates letterIndex from 0 to the length of the String name.
- Line 05: Uses the substring() method and the current value of the iterator to assign the String letter the value of the character at letterIndex in the String name.
- Line 06: Compares letter to each vowel using the equals() method and if any of them match:
  - Line 07: Assigns the literal String value "*" to letter.
- Line 09: Outputs letter to the console.