Strings in Java – String Methods

The only operator that can be applied to String objects is *concatenation* (+) for combining one or more strings.

Java also provides many methods with the String class to allow us to manipulate text. These are some of the most common and useful, but this is not a comprehensive listing. You can find many references to the full String class if you wish to explore it in greater depth.

`length()`

In many ways, strings in Java are like arrays. One example is that they both have lengths associated with them, but the concepts are not identical.

With an array, the length is a property of the array. For the array a, the length is given by a.length.

For a string, the length is determined by the *instance method* length, which is part of the String class. The length method returns the number of characters in the string.

```
String s = "Sergiy";
System.out.println(s.length());       // outputs the value 6
```

`public char charAt (int i)`

This method will return the character located at the specified position in the string. Like an array, positions in a string start numbering at zero. Specifying a position outside of the available characters in a string will cause an exception to be thrown.

For example, given

```
String s = "Jasper";
```

then

(a) `s.charAt(0)` will return 'J'
(b) `s.charAt(5)` will return 'r'
(c) `s.charAt(6)` will throw an exception because 6 is out of bounds for the string

`public int indexOf (char c)`
`public int indexOf (char c, int i)`

This method allows us to locate a particular character in a string. If `c` is a character in the string, the method will return the index of the first (leftmost) occurrence of c. If `c` is not in the string, it will return `-1`.

The method is also overloaded. The more complicated version also accepts an integer argument. When it searches for the character, `c`, it starts looking at index `i`, which means from index `0` to index `(i-1)` is ignored.

Given

```
String s = "Toronto";
```

then

(a) `s.indexOf('T')` will return 0
(b) `s.indexOf('t')` will return 5
(c) `s.indexOf('o',3)` will return 3 (ignoring the first 'o' at 1)
(d) `s.indexOf('r',3)` will return -1 (since the only 'r', at 2, is ignored)

`public String substring (int start)`
`public String substring (int start, int pastEnd)`

Similar to `charAt`, which extracts characters, the `substring` method extracts part of a string from a string. The value of `start` must be greater than or equal to zero, and less than or equal to the length of the string.

The overloaded method adds an additional parameter, `pastEnd`, which specifies the substring should go from `start` to the character *one before* `pastEnd` (which is the reason for the odd name – `pastEnd` rather than simply `end`).

Given

```
String s = "Brian Auyeung";
```

then

(a) `s.substring(3)` will return "an Auyeung"
(b) `s.substring(2,5)` will return "ian"
(c) `s.substring(20)` will throw an exception as our string index is out of bounds

`trim()`

This method removes any leading or trailing whitespace from a string. Since we cannot change a string, this method creates a new string object in memory and returns a `String` reference to that object.

Whitespace includes blank spaces, tabs, new-lines, carriage returns, and form feed characters.

Given

```
s = "       Lots      of  extra       blanks          ";
```

then

```
s.trim();
```

will return the string `" Lots      of  extra       blanks"`. Notice the leading and trailing whitespace has been removed, but the whitespace between characters is unaffected.

`public String toUpperCase()`
`public String toLowerCase()`
`boolean equalsIgnoreCase(String other)`

When comparing strings, upper case and lower case letters are considered to be different. To compare strings for the same letters, without worrying about the upper/lower case issue, we can convert them to either all upper or lower case. There is also a method to specifically check for equality while ignoring case.

Given

```
String s = "Bart & Lisa";
```

then

(a) `s.toLowerCase()` will return `"bart & lisa"`
(b) `s.toUpperCase()` will return `"BART & LISA"`
(c) `s.equalsIgnoreCase("BART & lisa")` will return `true`

`valueOf`

Values of any type can be converted to string (which is how the `print` and `println` methods work). To convert an object to a string, we can use the object's `toString` method. For any primitive data types, we need to use the `valueOf` method from the `String` class.

(a) `String.valueOf(123)` returns `"123"`
(b) `String.valueOf('a')` returns `"a"`
(c) `String.valueOf(2.5)` returns `"2.5"`

Chaining Instance Methods

Java allows us to chain instance methods to produce compound effects, which can be particularly useful when working with strings.

Given an expression in the form

```
<object>.<method1>.<method2>
```

Java will first execute using `<object>` as its implicit object parameter. It will then execute `<method2>` using the value returned by the first operation.

For example, given `s` with the value `"abc"`, then the expression

```
s.toUpperCase().charAt(1)
```

will return the value `'B'`.

`s.toUpperCase()` returns the value `"ABC"`.
`"ABC".charAt(1)` returns the value `'B'`.

Note that these operations do *not* change the value of the string `s`.

Exercises

1. Given the declaration
   ```
   String s = "Computer Science";
   ```
   find the value of each expression.
   (a) `s.charAt(3)`
   (b) `s.length()`
   (c) `s.charAt(0)`
   (d) `s.substring(8)`
   (e) `s.indexOf('a')`
   (f) `s.substring(1,4)`
   (g) `s.substring(1)`
   (h) `s.indexOf('m',4)`
   (i) `s.charAt(4)`
   (j) `s.substring(4,5)`

2. State the value of each expression.
   (a) `String.valueOf(2*4)`
   (b) `String.valueOf(27%7)`
   (c) `String.valueOf((char)('A' + 4))`
   (d) `String.valueOf(3 < 4 && 5 < 6)`

3. Assuming that the string `name` has the value "Buckaroo Banzai", find the value of each expression.
   (a) `name.toLowerCase().indexOf('a')`
   (b) `name.toUpperCase().charAt(5)`
   (c) `name.substring(3).indexOf('i')`
   (d) `name.substring(2).toUpperCase()`
   (e) `name.toUpperCase().indexOf('A',1)`
   (f) `name.substring(name.indexOf(' ')+1).length()`

4. Complete the definition of the method count so that it returns the number of occurrences of the character `c` in the string `s`.
   ```
   public static int count (char c, String s)
   ```

5. Complete the definition of the method replace so that it returns a string in which all occurrences of `oldChar` in the string `s` are replaced by `newChar`.
   ```
   public static String replace (String s, char oldChar, char newChar)
   ```