Java API spec for class String

A string is sequences of characters. In Java, we write String literals, like "abc", with double quotes and not single quotes surrounding the list of characters.

Here is what the Java API webpage for class String looks like. The description tells us that the class represents character strings. Note that a String immutable; it cannot be changed. We'll talk more about that later, when we get into OO more.

A simple notation for characters and substrings

This more detailed picture of a string shows you that, in Java, the characters of a string are numbered 0, 1, 2, and so on.

When talking about a string s, we often use the notation *s*[*k*] to denote the character at index *k*. This notation is useful, even if it is not Java notation. In the same way, *s*[*h*..*k*] denotes the substring of *s* that contains the characters *s*[*h*], *s*[*h*+1], …, *s*[*k*]; *s*[*h*..] denotes the substring of *s* starting at *h* and going to the end; and *s*[*h*..*h*-1] denotes the empty string starting at position h.

String functions

At this point, we introduce the major functions used on Strings. First, we show the notation for a call. For a String variable *s*, the call of a function like *length* is the name *s*, a period, the function name, and the arguments of the call, in parentheses —*length* takes no arguments:

s.length()

If you are new to OO programming, this may seem strange, but it will become clearer very soon.

Below is a list of frequently used string functions. We don't read them to you, you can read them yourself. We do note that *s.substring*(*h*, *k*) does *not* contain character *s*[*k*].

s.length() returns the number of characters in the *s*.

s.charAt(*h*) returns the characters at position *h*, or *s*[*h*] in our notation.

s.equals(*string*) is true if and only if *s* and the other *string* contains the same sequence of characters.

s.substring(*h*, k) returns a new String that contains *s*[*h*..*k*-1]. Note that *s*[*k*] is *not* included.

s.substring(*h*) returns a new String contains all characters from *s*[*h*] onward, or *s*[*h*..].

s.indexOf(*string*) returns the first position where the *string* occurs in *s*, or -1 if the string is not in *i*.

s.trim() returns a new String that is I but with blanks removed from the beginning and the end.

There are more functions, shown below, but they are less frequently used.

indexOf	*lastIndexOf*	*contains*
startsWith	*startsWith*	*compareTo*
replace	*toLowerCase*	*toUpperCase*

Developing a function

We now develop a function to show you how we tend to develop code, thinking in terms of English, not Java, and translating into Java when we know what is to be done. The silly function we develop is designed to use many of the String functions

```
/**String s contains a list of at least one name.
   Adjacent names are separated by one or more space chars.
   If there is only one name, return it; otherwise,
    return the first name but with its first character placed at the end and with all chars in upper case. */
public static string getName(String s)
```

For example, f(" Abe ") is "Abe " and f("Abe Lincoln ") is "EAB".

To start, let's remove the spaces at the beginning and end of *s*. Is there a function that can help us do this? Yes! *trim*(). But be careful, the call *s.trim*() does not *change s* but produces a new String with surrounding spaces removed, so we have to assign the result to *s*.

Now we have to determine whether there is only one name. That's the case if there are no space characters, or blanks, in *s*. If there are at least two names, there is a blank after the first name. So let's look for the first blank. We use function *indexOf* and store the result in a new local variable *k*.

Now, if k is –1, s contains no blank, so we can return s.

Otherwise, we have to construct a string consisting of the last character of the first name, which is just before the blank character s[k], catenated with everything that comes before, s[0..k-2], but all in upper case. Now let's translate this into java. First comes s[k-1] —we use function charAt. Next comes the substring s[0..k-2] —we use function substring, but note that the second parameter is k–1 and not k–2 because of the way substring is defined. Now, we have to make all those characters into upper case —we use instance function toUpperCase. We are done.

Note that the code works if the first name is a single character. That's because s[0..-1] and s.substring(0, 0) are the empty string "".

```
/**String s contains a list of at least one name.
   Adjacent names separated by one or more space characters.
   If there is only one name, return it; otherwise,
    return the first name but with its last character placed at the beginning and all chars in upper case. */
public static string getName(String s)
   // Remove spaces at beginning and end
   s= trim();

   // Store in k the index of the first blank in s (-1 if none)
   int k= s.indexOf(' ');
   if (k == -1) return s.charAt(0);

   // return s[k–1] + s[0..k–2], all in upper case
   return (s.charAt(k–1)  + s.substring(0, k–1)).toUpperCase();

}
```