Jenny Hamer

Chapter 22: Swing Handling

:: Exercises 1-9, p. 839-940 ::

1. Write a statement that displays the substring of a string **s** consisting of all but the last character of s.

String s = "bananas";

int lenS = s.length();

String newS = "";

for (int count = 0; count < lenS-1; count++) {

newS += s.charAt(count);

}

System.out.println("method 1: \noriginal: " + s + "\nsubstring of s: "

+ newS);

String substringS = s.substring(0, lenS-1);

System.out.println("method 2: \noriginal: " + s + "\nsubstring of s: "

+ substringS);

1. Assume **s** is a string and consider the following statement:

boolean b = s.isEmpty();

find the equivalent statement (or set thereof) using **length()** instead of **isEmpty()**.

public static boolean emptyString(String s) {

return s.length() <= 0;

}

1. What happens if you call **charAt()** and the value you pass in for the index is out of range?

A null pointer error is returned (NullPointerError).

1. Suppose you want to create on string that consists of two lines of text, each on a separate line. You cannot use a statement such as:

String s = “This is the first line.

This is the second line”;

because this statement will generate a compiler error. How can you do it?

* One can simply use the “+” operand, which concatenates strings, or the built-in method string1.concat(String string2);

1. 2 + 2 + “ME” is compiled and returned as “4ME”.
2. Suppose you want to test whether a string variable **s** contains the string “abcdef”. Is it sufficient to call s.startsWith(”abc”) and s.endsWith(“def”) and then see whether both method calls return true?

No, it is not sufficient, because those letters could be positioned at any location, therefore checking only the beginning portion and ending portions is arbitrary. It is better to check using indexOf(*substring, index*), which will return the index, iff true, or false otherwise.

1. In the **UseTrim** example in the chapter that demonstrated how to use the **trim()** method, the **if-else-if** ladder repeatedly tested the equality of two string using the **equals()** method. Why didn’t the ladder repeatedly test the equality using “==”?

When testing for equality with strings, given that the data type is immutable, using “==” will check for equality between pointers or the location references (ie. where the string data is being stored in memory), whereas the **.equals()** method returns **true** if the invoking object contains the string specified by its parameter (and false otherwise).

1. Explain the difference between **startsWith(substring, index)** and **indexOf(substring, index)**.

startsWith(…) begins searching for the given *substring*beginning at the specified *index*, and scans through the end of the invoking string. On the other hand, indexOf(…) begins searching for the given *substring* until just before it reaches the specified *index* (ie. up to, but not including the character at that index).

1. Suppose you have a string **s**  containing some English text that always uses male gender pronouns (e.g. he, his, him) and you want to change it to use female pronouns. The following program reads a string that is input by the user and uses the **replace()** method to attempt such a conversion. It will correctly convert sentences such as “He went to his house.” Into “She went to her house.” Unfortunately, this program doesn’t work correctly for other English sentences. Find a sentence in which the program doesn’t correctly convert.

The string “he goes to his house.” converts to “he goes to her house.”, where it should be “she goes to her house.” This **replace()** method is case and white-space sensitive.