**Name:**

**Advanced Programming in Java**

**Lab Exercise 10/2/2019**

**Arrays**

Read Lesson 18 in your textbook and answer the following questions.

1. Write code that will declare an integer array called *sgt* having 800 elements.

2. double []dfw = new double[21];

System.out.println( dfw.length );

3. For the code in #1 above, write a for-loop that will cycle through all the elements of

*double sgt[]* and store the square root of the index of each element in that element.

4. Assuming *rtl* is the name of an array, what’s wrong with this code?

double rtl\_len = rtl.length( );

5. On **one** line of code, both declare a character array called *cr* and initialize its elements

to be ‘a’, ‘b’, ‘c’, ‘d’, and ‘e’.

6. Refer to #5 above. What is the value of *cr.length* ?

7. Write code that will print the sum of the squares of the all elements of the *ref* integer

array.

8. What’s wrong with the following code?

for (int k = 2; k < homer.length; k++)

{

homer[k +1] = k;

}

9. Fill in the blanks below to enable us to pass a *double* array called *dbx* to a method

called *heroWorship*. Within the method, the array should be called *vb*.

boolean bbc = heroWorship( );

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

public boolean heroWorship( \_\_\_\_\_\_\_\_\_ ) //signature of method

10. Assume the five values an integer array *adc* contains are: 34, 56, -102, 18, and 5.

What is the value of *adc[1]* ?

11. Using the *adc* array from #10 above, what would be the value of *adc[3] + adc[4]*?

12. Using the *adc* array from #10 above, what would be the value of *adc[5]* ?

13. Describe what the following code segment does:

for (int j = 0; j < b.length; j++)

b[j] = Math.abs( b[j] );

14. For the *int* array *c = {1, 2, 3, 4}*, what would be the output of the following code?

String ss = “>>>”;

int len = ss.length( );

for (int j = 0; j < len; j++)

ss+= c[j];

System.out.println(ss);

15. Write a loop that locates the first occurrence of a negative integer in an array, *pg*.

When the loop is finished, the variable *indx* should contain the index of the negative

number or the length of the array if there were no negative numbers in the array.

16. String wc = “Whooping crane”;

String sp[] = wc.split(“oo”);

for(int j = 0; j < sp.length; j++)

{

System.out.println(sp[j]);

}

17. List the elements of *String [ ]sArray = “fee fi fo ”.split(“*\\*s+”);*.

18. List the elements of *String [ ]sp = “One two”.split(“Q”);*.

19. Using the *split* method, write code that will count all of the occurrences of “th”

(without regard to upper or lower case) in “The best THERE is is Barth”.

When completing the projects, attach your program source code to this sheet and turn in.

**Project… Array of Hope**

This project called *ArrayOfHope* will consist of just one class, *Tester*, that in turn, has just one method, *main*. The *main* method will use two *for*-loops:

• The first loop will produce an integer count from 65 to 90 (notice these are the ASCII

codes for characters A…Z) and initialize the elements of the character array *ch[ ]* with

the characters corresponding to the ASCII codes being generated by the loop. This will

fill the *ch[ ]* array as follows: ch[0] = ‘A’, ch[1] = ‘B’, …, ch[25] = ‘Z’.

• The second loop will print the 26 elements of the *ch[ ]* array with one comma followed

by one space between adjacent characters as follows:

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

**Project… Count ’em Right**

At the bottom of page 18-4 we were left with the dilemma of how to use the *split* method to count multiple occurrences of a regular expression when that expression is at the **end** of the *String* to be searched. It is also interesting to see what happens when there is a delimiter at the **beginning** of the *String*. Following is an example of the anomalies caused by having a delimiter at either the beginning or end of the *String*:

String s = “cHello good cbuddyc”;

int sp[] = s.split(“c”);

One would normally think that this would produce two elements for the *sp* array (sp[0] =

“Hello good ” and sp[1] = “buddy”). This is **not** the case. In fact, it produces three

elements (sp[0] = “”, sp[1] = “Hello good ”, and sp[2] = “buddy”). This anomaly of an

empty *String* occurs when a delimiter is at the beginning of the *String*, but strangely

enough, not when a delimiter is at the end.

Fortunately, for the sake of counting delimiters, one at the beginning of a **String** is

automatically handled since sp[0] = “”. A delimiter at the end of the *String* is easily

handled with the concatenation of extra characters to the end of the *String*.

Our project will be to count all occurrences of the letter *s* followed by the letter *a*. Case will be ignored, and it will be permissible to have **any** amount (including none) of white space between the *s* and the *a*.

Call your project *CountEmRight* and create just one class, *Tester*. In the *main* method, do the following:

• Create a loop that asks for *String* input (a sentence).

• Release from the loop if the input *String* is “EXIT”.

• So as to ignore case, convert the input *String* into an uppercase version of itself.

• Concatenate some “harmless” *String* to the end of the input *String*. By “harmless” it is

meant that it should not contain any occurrences of the delimiter expression. This is the

**real secret** to this project… to get any occurrence of the sought after expression off the

end and into the “interior” of the *String*.

• Use the *split* method to produce a *String* array (call it *sp*). Then use *sp.length –1* to

count the number of occurrences.

A typical run will appear as follows:

Type in a sentence and press ENTER. His initials are SA

There are 2 occurrences.

Type in a sentence and press ENTER. Sad but true, their teams are

better.

There are 2 occurrences.

Type in a sentence and press ENTER. S a sa ssa s a

There are 4 occurrences.

Type in a sentence and press ENTER. exit