1) All of the given statements are true except: *"Strings are the primitive datatype to store characters"*. Strings do store characters, but they are not a primitive datatype, they are an object type.

2) Index numbers always start from 0. The length() method for Strings returns the number of characters in the String. Because the first character has an index of 0, in order to get the last index dynamically we can do the length() – 1. Again, the last index has to be one less than the length because the indexes start from 0.

3) The trim() method will delete any spaces from the beginning and end of the String. Any spaces found in the middle of a String between other characters are not removed, so after removing the extra spaces in the beginning and end of the String we get the total length of our String to be 16. We still have 2 space characters here between the words and those are included in the total length of the String.

4) Here we have to remember codes reads left to right. So, when we look at the 5 + 4 part, these are two numbers being added together given us a result of 9 then we are doing '+ a'. This part is not addition anymore because 'a' is a String, so the + operator is preforming concatenation and combining the two data into one String. The resulting String gives us each part concatenated together as 9123.

5) Remember indexes start from 0 and charAt method will give you a character back based on the index number you give, so charAt(4) gives us back the space character. Also notice we are storing this result into an int datatype which is okay because the char value is automatically converted to the ascii value of that given char. In the if statement that first condition is false because the character is not equal to 'a'. In the else if the condition is now true because the character is equal to the space character and because of this the statement in the else if block is executed giving us B.

6) The condition we are evaluating in the ternary statement becomes 3425 > 9000. This condition is false so we will go to the right side of the colon (:) to get the value that will be stored into the String 'result', which is patio. After that there is substring method used on the String, but it is not reassigned or stored anywhere, so the String value in 'result' is never changed after this line which means when we print the String in the next line it will be the same String that we got from the first line: patio. The important part here is that String is not mutable and not reassigning or storing to a new variable will not change the value of the String beyond the line any given method is executed.

7) We take the substring of our String from index 4 to index 17 and reassign that substring to our str variable. The resulting substring is "fox ran under". The next line which performs the toUpperCase() only modifies the String for that line and because it is not reassigned to the variable or stored somewhere else our String remains "fox ran under", all in lowercase letters. In the print statement we take our String and concatenate "ground" to it which gives us the final result of "fox ran underground".

8) Here we made another String variable which holds the substring of the first String (s) starting from index 5. The resulting substring, which is stored to variable 's2', is "ame was tied at 2-2". After this we use the indexOf method on both Strings. The indexOf method will take a String and give back the index where the given String can be found in the String we are looking through. The key point here is when the indexOf method does not find the String you are looking for anywhere in the String, it will give you back a negative number. The index1 value will be 4 because that is the index that "game" is found in the first String. The index2 value will be negative -1 because "game" cannot be found anywhere in the second String. Based on the if statement, because they are not equal (4 != 0) the else block will execute its statements which will print the value of index2 giving -1 as the result.

9) We are trying to take the substring of 1 to 8 from our String, but the String only has 7 characters, so when we try to do a substring including the index 7, which does not exist because the last index would be 6, our code will go out of bounds.

10) The boolean b is checking if our String (a) contains the character "i". Because the String does contain a lowercase i character, the boolean has a value of true. The boolean c is taking the substring of the String starting from 12 which gives us " go to the beach" and then check if that substring startsWith the characters "go". We need to pay close attention here because the substring we got by using index 12 begins with a space and when we use startsWith method we are checking if the String begins with that exact String, so the space that is in the String will cause the boolean to have a value of false because (go) is not the same as (go). Next we use the ternary is evaluate b && c which results in true && false. This statement will end up being false overall because both are not true. Because the condition is false the ternary will store the value on the right of the colon (:) which is don't go and that will be printed in the following line.

11) The keyword continue is used to skip the rest of the iteration, which means any code after this statement will not run for that iteration and it will go to the next iteration. The keyword break is used to exit the loop completely and resume execution after the loop. The given statement was true.

12) Here we have a standard for loop which begins at 0, will execute for 10 times and updates the number one at a time. Inside of the loop we have an if statement that checks if the 'i', which is the variable updated each iteration, is even. In other words, the number does not have any remainder when you divide the number of 2. When the condition is true, which is for every even number from 0 to 9 the loop will execute the continue statement and skip the rest of that iteration, which means no even numbers will ever print in this loop. In the case that the number is odd, the if statement will be false and the continue statement would not be executed, which mean that the odd numbers will always print giving us a result of 13579.

13) In this question the important part is to check the boolean condition in the while loop for each iteration. In the beginning our number has a value of 5, so 5 < 100 is true, so we will run the next iteration which will execute the code: number += number. This code will essentially always double your number and store that number in our variable. Because there is nothing after this statement the while loop goes to check the condition again and continue these steps until the while loop condition is false. Flow will be: (10 < 100) → number = 20 | (20 < 100) → number = 40 | (40 < 100) → number = 80 | (80 < 100) → number = 160. (160 < 100) → false. Now that the condition is false the loop will stop, and the print statement will give the value of number at that point: 160. Another thing to keep in mind is the iteration which checks 80 < 100 is true so the code runs that iteration which is how the number gets doubled to 160.

14) Do while loop always has at least one execution of code inside of the loop. In this first guaranteed execution the charAt method is reading the character of index 0 from the *word* String, so it will print out the first character 'j'. The *a* variable is incremented by 1 becoming 1. Then the while condition is checked and since the *a* is not bigger than the length of *word* the statement is false so the loop does not keep executing. Only 'j' was printed here.

15) The main focus here is that we are always printing the first character in the String, but after printing we are also changing the String every iteration by each third characters. We do this change with substring by the 3rd index in each iteration. So essentially, we are taking every 3rd character in our String, until we get to the end and when we get to the end, we know we have checked every character because the String will be empty at that point and the loop will stop executing. Result we get is: T o miw in
In the last iteration the String was "ng." and taking the substring of 3 on that makes the String empty.

16) Small details are important. This question does not require any analysis. Right away you should check the termination condition of your loop. The condition (index <= 0) is false from the beginning. The last index is not less than or equal to 0, which means the loop will never run at all, so when we print the str 'word' at the end, it will print nothing.

17) The loop in this code snippet has a variable 'i' which start at 0, the termination condition is to run as long as "i' is less than or equal to the length of the String. Also, the loop is updating the 'i' value by 2 every iteration. We use charAt() method to read the character at the index of 'i' in every iteration which will give us: cbre, but because of the termination condition after printing the e character the 'i' will update to 8 and because 8 is equal to the length of the String the condition will be true and the loop will run one more time, but when it runs for this iteration using the charAt() method will give the StringIndexOutOfBoundsException because the index 8 does not exist in the given String. Because we are printing everything in one line with the print method, we get cbreStringIndexOutOfBoundsException in the console.

18) No trick in this question, we need to keep track of everything during each iteration. The key here is about the updates we do in the while condition and the print statement. Also, it is not really important to know the length of the String because the loop will never get that far. (read top left, top right, bottom left, bottom right)

```
n = 0

while(n++ < input.length())
        (0 < input.length())
n = 1
System.out.print(input.charAt(++n))
n = 2  | prints 'd' (char at index 2)
```

```
while(n++ < input.length())
        (2 < input.length())
n = 3
System.out.print(input.charAt(++n))
n = 4 | prints 'y' (char at index 4)
```

```
while(n++ < input.length())
        (4 < input.length())
n = 5
System.out.print(input.charAt(++n))
n = 7 | prints 'i' (char at index 6)
```

```
while(n++ < input.length())
        (7 < input.length())
n = 8
```

If (n == 8) is true so the loop continues to next iteration, printing nothing

```
while(n++ < input.length())
        (8 < input.length())
n = 9
```

else if (n == 9) is now true which mean the loop will break having only printed 3 characters: dyi

19) The inputs given will be stored into the variables: num1 will be 2, num2 will be 6 and iterate will be 5. This means that the loop will run for 5 times total. The if condition given checks if the number each iterations (j) is divisible by 3 evenly by checking if there is no remainder. Whenever 'j' is evenly divisibly by 3 the iteration will execute the continue statement and skip any code found afterwards, but for the rest of the numbers the continue statement will not be executed and num1 + num2 will be executed and stored into the variable 'total'. The key here is to determine how many times the loop will iterate and add the values of num1 and num2 into total. The loop will continue when j is 0 and 3, so the adding will happen for j = 1, 2, 4. Because we will add the numbers num1 and num2 for 3 iterations we will be adding (2 + 6) 8 to 'total' for 3 times which is how we will get the final 'total' to be 24.

20) In this question we have a nested loop and we will need to determine how many times our variable 'count' gets incremented by 1 based on both loop in combination with the continue and break statements.

{ a = 0 | 0 < 4 } { b = a + 1 == 1 | 1 < 5 → count = 1; b = 2 | 2 < 5 → count = 2; b = 3 | 3 < 5 → count = 3; if (b ==3) break executed stopping inner loop. }
{a = 1 | 1 < 4 } { b = a + 1 == 2 | 2 < 5 → count = 4; b = 3 | 3 < 5 → count = 5; if (b ==3) break executed stopping inner loop. }
{a = 2 | 2 < 4 } { b= a + 1 == 3 | 3 < 5 → count = 6; if (b ==3) break executed stopping inner loop. }
{a = 3 | 3 < 4} if(a == 3) continue statement run, a = 4 | 4 < 4 → condition is false to the outer loop stops
In the end the variable count only went up to 6 based on this nested loop.

21) False, the array's size is fixed upon creation. The size of the array must always be given in the beginning, either in an empty array with the size declared, or by giving the initial elements with {} brackets. Either way the size of the array is defined right away, and it can never be changed. The size is not flexible ever.

22) Make sure to know what a valid declaration for an array is. Valid ones include: int [] a = {1,2,3}, String s [] = new String[4], boolean [] bool = {true, true, false, true}, and char chars [] new char[26]. All the other lines are invalid because they do not follow the syntax rules.

Note: To properly print an array and see the elements we have to use the Arrays.toString() method, otherwise the memory location will be printed rather than the actual element values.

23) The key here is reading and writing information from and to the array. We need to just analysis the code line by line to determine have happens to our array. The index numbers are important to pay attention to. In the beginning we define a new array with 5 elements ([0, 0, 0, 0, 0]). We also have a variable 'a' that is initially 5. nums[2] = a will store 5 to the index 2 ([0, 0, 5, 0, 0]). nums[0] = a * 2 will evaluate to 5 *2, so 10 is stored to the first index of the array ([10, 0, 5, 0, 0]). nums[4] = nums[1] * a will evaluate to 0 * 5 → 0 will stay as the last index of the array ([10, 0, 5 0, 0]). nums[1] = nums[2] will read the element in index 2 and store that value to the index 1. ([10, 5, 5, 0, 0]). Finally, we do nums[3] = nums[a-3]. 'a' is 3 at this point, so a-3 (3-3) evaluates to 0. This means we will read the element in the first index and store it into the 3rd index. Printing the final array: ([10, 5, 5, 5, 0]).

24) In this question the important part is on line 4 where we assign a new array to our 'doub' reference. What this does is created a brand-new array with a size of 4, but at that point all the elements are 0. The first array we had in which we stored 1.0 and 42.1 is lost because our reference 'doub' is now point to a different array. So, after that line we add 17.2 to index 1 and the length of the array into the last index. The first and third index remain 0. The final array is [0.0, 17.2, 0.0, 4.0].

25) This question was similar to one you saw on a past quiz, but with a different result. The int 'a' will store the value of the length of the array but when the following line of int b executes the code will go out of bounds. The reason is because there is no index 5 in our array (we know the last index is the length -1 → 5 -1 = last index is 4, and if we try to read an element with an index that does not exist, we will get ArrayIndexOutOfBoundsException.

26) Here we have a byte array with values 1,2,3. We are using a for loop to take each element in the array and multiplying 2 to each element. That result of the multiplication is then stored back into the array at the same index it was red from. The last point to mention is the calculation results in an int datatype so in order to store it as a byte for the array we need to cast the value to byte first. Final array value is [2, 4, 6].

27) In this code snippet we have two arrays, one is the given starting Strings, and the other for the results. We use a for each loop to read each element from the 'words' array. After reading each word we also look at the length of the String element and concentrate the those two together. We take that combination and store it onto the 2nd array 'other' using the 'index' variable that we have also declared. In the end the array that prints is each String element we had in the 'words' array but with the length of each the String at the end. The 'other' array elements: [one3, two3, three5, four4]

28) Here are two arrays. One is an int array with values and the other is an empty boolean array with the same size as the int array. The int array is looped through and boolean elements is changed to true when the number is even. Thing to keep in mind is the default value of the boolean array elements is false. These are the two arrays in comparison after going through the loop and assigning true to the even number positions:

arr	1	13	12	5	24	7	9	10
bArr	false	false	true	false	true	false	false	true

29) This question involves reading each element of an array, using a loop and some if conditions to execute some code that will change a variable 'total' during each iteration. The variable total will change ever iteration based on if the element is not divisible by 2, so we just need to go through each element/ each iteration one at a time.

nums array	14	1	84	97	1243	46
Code run	Total += 10	Total += 5	Total += 10	Total += 5	Total += 5	Total += 10
Total value	10	15	25	30	35	**45**

30) In this code snippet we are looping through every element of the String array 'things' with a for each loop in which every element in the array will be referred to as 's'. We take each element from the array and use a switch statement to print out some numbers each iteration. The switch value that is read each iteration is always the character at the index of 1, so for each element we are running the switch statement for every 2nd character of the words. Again, for switch statements keep in mind whenever a case matches the code will continue to execute until a break is executed or it reaches the end of the switch statement. Any cases that do not match will execute the default case.

[house] o → 45
[shed] h →1
[slide] l → (nothing)
[zebra] e → default → 45
[park] a → 23
[garden] a → 23

Final result would have been: 451452323