Exam 3 – Offline Version – Solutions

1. What does short circuit evaluation mean in the following code?

```
i) if (a < b && c != d)
```

   a. if a < b is false it evaluates c != d – See explanation of choice (b) below.
   b. if a < b is false it doesn't evaluate c != d - Both sides of the && must be true for the entire if statement to pass, so the (c != d) does not need to be evaluated if (a < b) is false. This is called short circuit evaluation.
   c. if c != d is false it evaluates a < b – Short-circuit evaluation happens from left to right.
   d. if c != d is false it doesn't evaluate a < b – Short-circuit evaluation happens from left to right.
   e. if a < b is true it doesn't evaluate c != d – If (a < b) is true then the entire expression can still be true or false, so (c !=d) still needs to be evaluated.

2. Which if statement below tests if the variable letter holds the char value #?
   a. if (letter == #) – This choice is missing single-quotes.
   b. if (letter == "#") – This choice uses double-quotes instead of single-quotes. We need single-quotes to express char values.
   c. if (letter == '#') – Single-quotes are used to express char values, unlike Strings which use double-quotes.
   d. if (letter >= '#') – The question is asking to check equality.
   e. if (letter = '#') – To check for equality, use the equals operator (==), not the variable setting operator (=).

3. Consider the following code:

```
int a [] = {2, 6, 8, 10, 12, 14, 16, 18};

int sum = 0;

for (int i = 0; i < a.length; i++) {

      if (a[i]%3 == 0)

            sum += a[i];

}

System.out.println(sum);
```

   What is output?

   a. 20 – See explanation of choice (d) below.
   b. 26 – See explanation of choice (d) below.
   c. 28 – See explanation of choice (d) below.
   d. 36 – This code adds together all of the multiples of 3 in the array. 6+12+18=36.
   e. 38 – See explanation of choice (d) above.

4. What is output by the following code?

```
for (int i = 0; i < 5; i++)

      System.out.print(i + " ");
```

a. 0 1 2 3 4 5 – See explanation of choice (b) below.
b. **0 1 2 3 4 – This code prints out the numbers 0 through 4 inclusive. 0 is included because that is the initial value of i. 5 is not included because the loop exits when the i variable reaches 5.**
c. 1 2 3 4 – See explanation of choice (b) above.
d. 1 2 3 4 5 – See explanation of choice (b) above.
e. 1 2 3 4 5 6 – See explanation of choice (b) above.

5. Which of the following would print the numbers, 32 54 76 98?

I.
```
for (int t = 32; t <= 100; t += 22)
      System.out.print(t + " ");
```

II.
```
int t = 10;
while (t < 90) {
      t += 22;
      System.out.print(t + " ");
}
```

III.
```
int t = 32;
while (t < 100) {
      t += 22;
      System.out.print(t + " ");
}
```

a. I only – II is also correct.
b. II only – I is also correct.
c. III only – III is incorrect because the initial value, 32, does not get printed. 22 is added to 32 before any printing takes place.
d. **I and II only – I is correct because the initial value of t, 32, gets printed, then 22 is added and the result is printed on each pass through the loop until t is greater than 100. II is correct because the value of t when it is first printed is 32, then 22 is added and the result is printed on each pass through the while loop. When t reaches 98, the result gets printed before the loop condition gets checked again and the loop exits.**
e. I, II and III – III is incorrect.

6. Assume that x and y are boolean variables and have been properly initialized.

(x && y) && !(x || y)

Which of the following best describes the result of evaluating the expression above?

a. Always true – See explanation of choice (b) below.
b. **Always false – The left-hand expression (x && y) is only true when both x and y are true. The right-hand expression !(x || y) is only true when both x and y are false. These two expressions are joined by an &&, but any time the left-hand expression is true the right-hand expression will be false and vice versa, so the entire expression is always false.**
c. true only when x is true and y is true – See explanation of choice (b) above.
d. true only when x and y have the same value – See explanation of choice (b) above.

e. true only when x and y have different values – See explanation of choice (b) above.

7. Consider the following boolean statement:

```
!( x > y && w == z )
```

Which of the following will produce the same result?

a. x <= y && w == z – Bringing the ! operator inside the expression flips the && to an || and (w==z) to (w!=z).
b. x >= y || w != z – Bringing the ! operator inside the expression flips (x > y) to (x <= y).
c. x <= y || w != z – We can bring the ! operator inside of the expression to flip (x > y) to (x <= y), flip (w==z) to (w!=z), and flip the && operator to a || operator.
d. x <= y && w != z – Bringing the ! operator inside the expression flips the && to an ||.
e. x < y && w != z - Bringing the ! operator inside the expression flips (x > y) to (x <= y).

8. What is 0 0 1 1 1 0 0 0 in base ten?
a. 54 b. 55 **c. 56** d. 57 e. 58

Moving right-to-left in binary digits,

$0*2^0+0*2^1+0*2^2+1*2^3+1*2^4+1*2^5+0*2^6+0*2^7 = 8+16+32 = 56$

9. What is 67 in binary?
a. 0 1 0 0 0 0 1 0 – See explanation of choice (b).
b. 0 1 0 0 0 0 1 1 – $67=64+3 = 64+2+1 = 2^6+2^1+2^0 = 01000011$
c. 0 1 0 0 0 1 1 0 – See explanation of choice (b).
d. 0 1 0 0 0 1 1 1 – See explanation of choice (b).
e. 0 1 0 0 1 1 1 0 – See explanation of choice (b).

10. Consider the following code:

```
String q = "power";

String r = "brown";

System.out.println(q.charAt( r.indexOf('n')));
```

What is output?

a. 2 b. 3 c. e **d. r** e. w

The character 'n' is at index 4 in String r, since we start counting the index from 0. The character in q that is at index 4 is 'r'.

11. Consider the following code:

```
String words[] = {"avalanche", "budget", "cannot", "center", "outside",
"meaning", "clear", "furniture", "deep", "piccolo", "friendly",
"poison"};

int c = 0;

for (int i = 0; i < words.length; i++) {

      if (words[i].substring(3).indexOf('o') >= 0)

            c++;

}

System.out.println(c);
```

What is output?

a. 0 – See explanation of choice (d).
b. 1 – See explanation of choice (d).
c. 2 – See explanation of choice (d).
d. **3 – This loop counts the number of words in the String array where 'o' appears in the substring beginning at index 3. This is true for the three Strings "cannot," "piccolo," and "poison."**
e. None of the above – See explanation of choice (d).

12. Consider the following code intended to implement a linear search:

```
int [] d = /* Assume array is correctly initialized */;

int num = /* Input from the keyboard */;

int found = -1;

for (int i = 0; i < d.length; i++) {

      if ( d[i] == num )

            /* Missing Code */

}
```

Which could be used to replace /* Missing Code */ so that the code works as intended?

a. found = 1; - This will tell us that we did find the value we are searching for, but will not store the index of that value as needed.
b. **found = i; - In a linear search, we need to find the index in the array where the value we are searching for is stored. The missing code runs when we have found the value, so we need to store the index of that value in the found variable.**

c. i = found; - There is no need to alter the loop variable.
d. i = -1; - There is no need to alter the loop variable.
e. d[i] = found; - There is no need to alter the values in the array.

13. Consider the following code:

```
String w = "Rapunzel";

for (int i = 0; i < w.length(); i++) {

      System.out.print(w.charAt(i) + " ");

      if (i%4 == 3)

            System.out.println();

}
```

What is output?

a. R a p u
n z e l – The code prints each character in the String separated by spaces, but adds an extra newline when i%4==3, or when the remainder after dividing i by 4 is equal to 3. This is true when i==3, so the line break happens after the character at index 3 is printed.
b. R a p
n z e l - See explanation of choice (a).
c. R a p u n z e l - See explanation of choice (a).
d. u l - See explanation of choice (a).
e. R a p n z e - See explanation of choice (a).

14. Consider the following code, intended to count the number of times that a given string appears in an array:

```
String n[] = /* Assume array initialized correctly */;

int c = 0;

String name = /* Input from the keyboard */;

for (int i = 0; i < n.length; i++) {

      if (/* Missing Code */ )

            c ++;

}

System.out.println(name + " appeared " + c  + " times in the array.");
```

What could be used to replace /* Missing Code */ so that the code works as intended?

a. n.equals(name) – n is an array, so we need to index into the array to compare values individually.
b. name.equals(n) – n is an array, so we need to index into the array to compare values individually.
c. **n[i].equals(name) – The loop should check one value in the array on each pass through the loop, and compare its value to the name variable using the String.equals() method.**
d. i.equals(name) – i is an int variable used to track which array index we are looking at on each pass through the loop.
e. ! n[i].equals(name) – This would count the number of Strings in the array that are not equal to the given String.

15. The following is intended to count the number of times a score less than 70 is found in an array of test scores:

```
int [] d = /* Assume array is correctly initialized */;

int failing = 0;

for (int i = 0; i < d.length; i++) {

      if (/* Missing Code */)

            failing++;

}

System.out.println("Number of failing scores: " + failing);
```

Which of the following could replace /* Missing Code */ so that the code works as intended?

a. d != 70 – d is an array, so we need to index into the array to check values individually. Also this would count grades above 70 as well as below 70.
b. failing < 70 – The failing variable is counting the number of failing grades, so it shouldn't be compared to the grade cutoff.
c. d < 70 - d is an array, so we need to index into the array to check values individually.
d. **d[i] < 70 - The loop should check one value in the array on each pass through the loop, and check whether that value is less than 70 (failing).**
e. i < 70 – i is the loop variable that tracks which index in the array we are examining on each pass through the loop. We need to use it to index into the array d.

16. What does the String method substring() do?
a. **Returns a portion of the String. – String.substring(int beginIndex) returns a smaller version of the String that starts at the parameter beginIndex.**
b. Tests two String objects for equality. – String.equals() is used to check two Strings for equality.
c. Compares this String with a second String for greater than, equal to, or less than. – String.compareTo() is used to compare two Strings in this way.
d. Returns the character at a certain location as a char value. – String.charAt() does this.
e. Returns the length of a String. – String.length() returns the length of the String.

17. What does the following code do?

```
String w3 = "aardvark";

System.out.println(w3.charAt(w3.length() - 2));
```

a. Prints the second letter in the String. – The index is counted backwards from the end of the String.
b. **Prints the second to last letter in the String. – w3.length() – 1 is the index of the last letter in the String since we start counting the index from 0. The second-to-last letter is therefore w3.length()-2.**
c. Prints the last letter in the String. - See explanation for choice (b).
d. Prints the first letter in the String. - The index is counted backwards from the end of the String.
e. Causes an index out of bounds error. – Since the String is more than two characters long, we do not get an index out of bounds error.

18. Consider the following code:

```
String major = "Computer Science";
```

What is returned by the method call major.charAt(2)?

a. 'C' b. 'o' **c. 'm'** d. "C" e. "Science"

We start counting the index from 0, so the character at index 2 is 'm.'

19. Consider the following code:

```
int list [] = new int [30];
```

The index of the first value is _____ and the last index is _____.

a. **0, 29 – We start counting the index from 0, so the index of the first value is 0 and the index of the last value is list.length-1==29.**
b. 0, 30 – See explanation to choice (a).
c. 1, 29 – See explanation to choice (a).
d. 1, 30 – See explanation to choice (a).
e. 1, 31 – See explanation to choice (a).

20. When should you use a for loop instead of a while loop?
a. When you are doing repeated calculations. – Either loop can be used for this.
b. When you do not know how many times a loop will repeat. – A while loop is designed for this.
c. **When you have an exact starting and stopping point. – Both types of loop could be used in this case, but a for loop is often easier to work with when you know how many times the loop will repeat and can easily express the starting and stopping point.**
d. When working with numbers. – Either loop can be used for this.
e. When doing user input. – Either loop can be used for this.

