Lab 3 Conditionals and Loops - Solution

Prelab Exercises

1. Rewrite each condition below in valid Java syntax (give a *boolean* expression):
   a. x > y > z

      Ans: `x > y && y > z`

   b. *x* and *y* are both less than 0

      Ans: `x < 0 && y < 0`

   c. neither *x* nor *y* is less than 0

      Ans: `x >= 0 && y >= 0`

   d. *x* is equal to *y* but not equal to *z*

      Ans: `x == y && x != z`

2. Suppose GPA is a variable containing the grade point average of a student. Suppose the goal of a program is to let a student know if he/she made the very great distinction list (the GPA must be 3.75 or above). Write an if... else... statement that prints out the appropriate message (either "Congratulations—you made the very great distinction" or "Sorry you didn't make Very Great Distinction ").

```
if(GPA >= 3.75){
   System.out.println("Congratulations—you made it to Very Great Distinction");
}else {
   System.out.println("Sorry you didn't make Very Great Distinction");
}
```

PreLab Exercises

In a while loop, execution of a set of statements (the body of the loop) continues until the boolean expression controlling the loop (the condition) becomes false. As for an if statement, the condition must be enclosed in parentheses. For example, the loop below prints the numbers from 1 to to LIMIT:

```
final int LIMIT = 100; // setup
int count = 1;
while (count <= LIMIT) // condition
{ // body
   System.out.println(count); // -- perform task
   count = count + 1; // -- update condition
}
```

There are three parts to a loop:

- The setup, or initialization. This comes before the actual loop, and is where variables are initialized in preparation for the first time through the loop.
- The condition, which is the boolean expression that controls the loop. This expression is evaluated each time through the loop. If it evaluates to true, the body of the loop is executed, and then the condition is evaluated again; if it evaluates to false, the loop terminates.
- The body of the loop. The body typically needs to do two things:
    - Do some work toward the task that the loop is trying to accomplish. This might involve printing, calculation, input and output, method calls—this code can be arbitrarily complex.
    - Update the condition. Something has to happen inside the loop so that the condition will eventually be false— otherwise the loop will go on forever (an infinite loop). This code can also be complex, but often it simply involves incrementing a counter or reading in a new value.

  Sometimes doing the work and updating the condition are related. For example, in the loop above, the print statement is doing work, while the statement that increments count is both doing work (since the loop's task is to print the values of count) and updating the condition (since the loop stops when count hits a certain value).

The loop above is an example of a count-controlled loop, that is, a loop that contains a counter (a variable that increases or decreases by a fixed value—usually 1—each time through the loop) and that stops when the counter reaches a certain value. Not all loops with counters are count-controlled; consider the example below, which determines how many even numbers must be added together, starting at 2, to reach or exceed a given limit.

```
final int LIMIT = 16;
```

```
int count = 1;
int sum = 0;
int nextVal = 2;
while (sum < LIMIT) {
   sum = sum + nextVal;
   nextVal = nextVal + 2;
   count = count + 1;
}
System.out.println("Had to add together " + (count-1) + " even
numbers" + "to reach value " + LIMIT + ". Sum is " + sum);
```

Note that although this loop counts how many times the body is executed, the condition does not depend on the value of count.

Not all loops have counters. For example, if the task in the loop above were simply to add together even numbers until the sum reached a certain limit and then print the sum (as opposed to printing the number of things added together), there would be no need for the counter. Similarly, the loop below sums integers input by the user and prints the sum; it contains no counter.

```
int sum = 0; //setup
String keepGoing = "y";
int nextVal;
while (keepGoing.equals("y") || keepGoing.equals("Y")) {
   System.out.print("Enter the next integer: "); //do work
   nextVal = scan.nextInt();
   sum = sum + nextVal;
   System.out.println("Type y or Y to keep going"); //update condition
   keepGoing = scan.next();
}
System.out.println("The sum of your integers is " + sum);
```

Exercises

1. In the first loop above, the println statement comes before the value of count is incremented. What would happen if you reversed the order of these statements so that count was incremented before its value was printed? Would the loop still print the same values? Explain.

   Ans: Currently the loop prints values from 1 to 100, if we modify it accounting to this instruction it will print 2 to 101

2. Consider the second loop above.
    a. Trace this loop, that is, in the table next to the code show values for variables nextVal, sum and count at each iteration. Then show what the code prints.
    b. Note that when the loop terminates, the number of even numbers added together before reaching the limit is count-1, not count. How could you modify the code so that when the loop terminates, the number of things added together is simply count?

3. Write a while loop that will print "I love computer science!!" 100 times. Is this loop count-controlled?

    Answer:

```
count = 0;
while (count < 100) {
   System.out.println("I love computer science!!");
   count = count + 1;
}
```

    Yes, the loop count-controlled

4. Add a counter to the third example loop above (the one that reads and sums integers input by the user). After the loop, print the number of integers read as well as the sum. Just note your changes on the example code. Is your loop now count-controlled?

    Answer:

```
int sum = 0; //setup
String keepGoing = "y";
int nextVal, count = 0;
while (keepGoing.equals("y") || keepGoing.equals("Y")) {
   System.out.print("Enter the next integer: ");
   nextVal = scan.nextInt();
   sum = sum + nextVal;
   count++;                  // Change
   System.out.println("Type y or Y to keep going");
   keepGoing = scan.next();
}
System.out.println("Number of integers: " + count); // Change
System.out.println("The sum of your integers is " + sum);
```

No, the loop is still not count-controlled

5. The code below is supposed to print the integers from 10 to 1 backwards. What is wrong with it? (Hint: there are two problems!) Correct the code so it does the right thing.

```
count = 10;
while (count >= 0) {
   System.out.println(count);
   count = count + 1;
}
```

Answer:

I. Since count >=0, the loop runs down to zero should run to count >0

II. Count is incremented, it should be decremented

```
count = 10;
while (count > 0) {
   System.out.println(count);
   count = count - 1;
}
```