# Programming Exercise 6:
Loops

**Purpose:** Introduction to while loops and for loops

**Background readings from textbook:** Liang, section 5.2-5.6

Due date for section 001: Monday, February 22 by 10 am
Due date for section 002: Wednesday, February 24 by 10 am

---

To this point of the semester, we have had to rerun our program every time we have new input. This is not user friendly. Another programming tool is called the loop (or iterative statement) that allows a program to repeat a set of code over and over either a set number of times (for instance, 10 times) or based on some condition (repeat until the user inputs 0). We can use these loops not only to "rerun" a program without having to run it again, but also to perform computations that require repeated calculations. In Java, there are three types of loops, the *while* loop, the *do-while* loop, and the *for* loop. This lab addresses the while loop and for loop.

## Part 1: While Loop Overview

A while loop consists of a condition (boolean expression) and a block of code, called the *loop body*. The structure of the while loop is

```
while(condition)
      body;
```

As with the if and if-else statement, if the number of instructions in the loop body is more than 1, we have to place the body inside { } as in

```
while(condition)
{
      body
}
```

Each instruction in the body ends in a ; but you do not put a ; after the condition or after either { }. The way the while loop works is this: test the condition and if true, do the loop body and repeat. That is, if the condition is true, the body executes and the condition is tested again and if still true, the body executes and the condition is tested again… The loop only terminates when the condition is false. At that point (whether the loop body executes once, ten times, ten million times or zero times), the first instruction after the loop executes. The while loop is a type of loop in which, if the condition is originally false, the body does not execute at all. On the other hand, if the condition never changes from true to false, the loop never terminates. This is called an *infinite loop*. The main difference between the while loop and the do-while loop is that the while loop is a *pre-test* loop, the condition is tested first while the do-while loop is a *post-test* loop, the condition is tested after the loop body executes. This means that in a do-while loop, the loop body will execute at least one time. In the while loop, the loop body does not necessarily have to execute depending on the condition.

One last comment. If the condition is true and the loop body executes, the condition is tested again. If the code in the loop body does not change the condition, then the condition remains true. If you are not careful, your loop body may never change the result of the condition and so you have an infinite loop. You must make sure that whatever test you have in your condition, the loop body can affect that test.

## Part 2: Example While Loops

```
int n = 0;
while( n < 10 )
{
       System.out.println( n );
       n++;
}
```

*The statement* ***n++*** *is equivalent to the statement* ***n = n + 1****.*

```
int sum = 0, value;
System.out.print(‘‘enter a value, 0 to exit: ’’);
value=in.nextInt();
while( value!= 0)
{
       sum=sum+value;
       System.out.print(‘‘enter next value, 0 to exit: ’’);
       value=in.nextInt();
}
```

*In this loop, the user controls whether to continue or not by inputting a number (0 to quit, non-0 to continue)*

```
int n = 1;
while( n < 1000 )
       n = n * 2;
System.out.println(‘‘The first power of 2 greater than 1000 is ’’ +
```

*Only 1 instruction in the loop body so we don’t need { }*

## Part 3: Common pitfalls.

1.
```
int n = 0;
while( n < 10 )
{
    System.out.print( n );
}
```

*Logic error!*

*Since n never changes in the loop body, we have an infinite loop.*

2.
```
int n = 0;
while( n < 10 );
{
    System.out.print( n );
    n++;
}
```

*Logic error!*

*Notice the ; at the end of the condition in the while loop. This while loop is interpreted as “while n is less than 10, do nothing”. So the print and n++ instructions are not reached.*

3.
```
int n = 0, y = 0, x = 1;
while( n = 0 )
{
   y = y + x;
   x++;
   if(x > 5) n = 1;
}
```

*Logic error!*

*Notice the condition says n = 0 instead of n == 0. The result is that n is set to 0 and this condition is always true. So no matter what happens inside the loop, we have an infinite loop.*

4.
```
int sum = 0, value;
System.out.print(``enter a value, 0 to exit:
``);
value=in.nextInt();
while( value!= 0)
{
       sum=sum+value;
`
```

*Logic error!*

*Compare this code to the second example under part 2. The difference is that in the loop body, we have removed the prompt and input statements so value does not change inside the loop body and thus, if value != 0 to begin with, it stays that way forever, an infinite loop! In fact this will lead to a run-time error in that eventually sum will overflow memory and the program terminates with an ArithmeticException because of the overflow.*

**Part 4:. A Full Example: Summing User Input**

```
int sum = 0, value;
Scanner input = new Scanner(System.in);
System.out.print(``Enter an integer value (0 to quit): ’’);
value = input.nextInt();
while (value != 0)
{
       sum += value;
       System.out.print(``Enter next value (0 to quit): ’’);
       value = input.nextInt();
}
System.out.println(``The sum of the entered values is ’’ + sum);
```

*The statement*
***sum += value;***
*is equivalent to the statement*
***sum = sum + value;***

The above code adds to the earlier example. The user inputs an initial value and if it is not 0, we enter the loop body. The value is added to sum and then we ask the user for the next value. If that value is not 0, this repeats. Once the user enters 0, the loop terminates and we reach the next instruction after the loop, the output statement which outputs the sum.

**Part 5: While Loop Exercise**

Enter a full program using the code from part 4 (you will have to add any import statements, class header, etc). Make sure that you understand how it works and add appropriate comments. Save, compile and run it and test it under several situations (input three non-zero values, input one non-zero value, input 0 initially, input both positive and negative numbers). When done, add the following enhancements.

(1) In addition to displaying the sum of the entered values, display the *number* of non-zero values that were entered. You will need to add another variable, call it `count`, and since any non-zero will execute the loop body, all you need to do is `count++;` in the loop body to count yet another non-zero input. Output count after you output the sum.
(2) If at least one non-zero value was entered, then also display the average of the non-zero values. This will require that you test to make sure count is not zero after the loop and if it is not, then compute and output the average. If count is 0, output an error message

because there were no inputs (other than 0) which would result in sum and count being 0 and a computation of 0/0 which itself would cause a division by zero ArithmeticException error.

(3) Add instructions to also count and display the number of user input values that are odd. You can test whether *value* is odd by testing that *value % 2* is equal to 1. This will require yet another variable to count the number of odd inputs. Initialize it to 0, and add an if statement in the loop body to test if value is odd and increment this variable. Finally, add an output statement after the loop to output the number of odd values input.

(4) Add another variable which will be input prior to the loop which asks the user for a target number. With another if statement in the loop body, test if the input value equals this target and if so, add one to another counter variable. After the loop, output both the target value and the number of times the user entered target.

(5) Add another variable, max, initialized to zero, to store the largest value entered. In the loop body, determine if the new input, value, is greater than max (`value > max`) and if so, set max to this new value. Output the max value as part of your output.

Let's assume you run the program entering 10 for target followed by this list of inputs: 5 10 11 1 3 8 10 4 1 6 7 10 7 0. The output of your program might look like this. Notice that 0 will not be counted as part of the input.

```
The sum of the input values is 83
The number of inputs is 13
The average of the input values is 6.38
The number of odd values input is 7
The value 10 was input 3 times
The maximum value entered was 11
```

**Part 6: For Loop Overview**

The **for** loop is known as a counting controlled loop. Rather than deciding whether to repeat or not on a condition, for loops are used to count some number of iterations. The Java for loop is far more flexible than a typical counting loop though and can also serve as a conditional loop like the while loop. For now, we will only use it as a counting loop. Whereas the while loop contains the condition and the body, the for loop has two additional parts called the *initialization* and *increment*. The form of the for loop is

```
for(initialization; condition; increment)
     body;
```

As with the while loop, if the body consists of more than one instruction, it must be placed in { }. The initialization is used to initialize the variable(s) that will be used in the condition and increment part. Such a variable is called the *loop variable*. The increment is used to alter that variable, often incrementing it by 1 but not necessarily. Here are some example loops (without the bodies) to illustrate what they look like.

```
for(i=0;i<10;i++)       // iterate from 0 to 9
for(i=10;i>=0;i--)      // iterate from 10 down to 0
for(i=a;i<b;i++)        // iterate from a to b-1 where a and b are int variables
```

You can initialize and increment more than one loop variable, as shown in the following.

```
for(i=0,j=n;i<j;i++,j=j-i)
```

## Part 7: A for loop Example

The example that we saw in Part 2 is very conveniently expressed as a for loop:

```
for (int n = 0; n < 10; n++)
{
      System.out.println( n );
}
```

This loop starts by setting n to 0. Next, the loop tests n < 10. If true, the loop body executes (n is output) and the incrementing step executes, n is incrementing from 0 to 1. Since n is still less than 10, the loop body executes followed by incrementing n to 2. This continues until n is incremented to 10. At that point, n < 10 is false, so the loop terminates. This code will output 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 on separate lines. Why doesn't it output 10? Because once n is equal to 10, n < 10 is false. So we never do the loop body when n = 10. How can we make it output 10? Change the condition from n < 10 to n < 11 or n <= 10. Why do we start at 0? This has something to do with how Java handles characters in Strings and elements in arrays. We'll hold from understanding this now and revisit it later in the semester.

Why do we have int in parentheses? We can handle our loop variable in two ways. We can declare it somewhere before the loop or we can declare it in the loop. Declaring it in the loop means that the variable is only declared for the contents of the loop. For instance, in the following code we get a syntax error in the last instruction because n is not declared in that location of the program. Had we done `int n;` before the loop, we would be ok. Why then do we declare n in the loop? A lot of programmers do it out of convenience or laziness. They wait until they write a loop to decide that they need n, and so they declare it was they write the loop itself.

```
for (int n = 0; n < 10; n++)
{
      System.out.println( n );
}
System.out.print(``Enter an integer number:   '');
n=in.nextInt();
```

## Part 8: Common Pitfalls

The following pitfalls are directly analogous to those that we saw earlier for **while** loops.

1.
```
for (int n = 0; n < 10; )
{
   System.out.print( n );
}
```

*Logic error!*

*This is an infinite loop since variable* n *is never updated; the boolean expression is always true.*

Note that you can leave the incrementing portion empty but it is risky because you can get an infinite loop. You can also leave the initialization portion empty as in

```
for( ; n < 10; n++)
```

2.
```
for (int n = 0; n < 10; n++);
{
      System.out.println( n );
}
```

*Logic error!*

*Also an infinite loop since the extra semicolon at the end of the for statement (first line) is interpreted as the loop body.*

3.

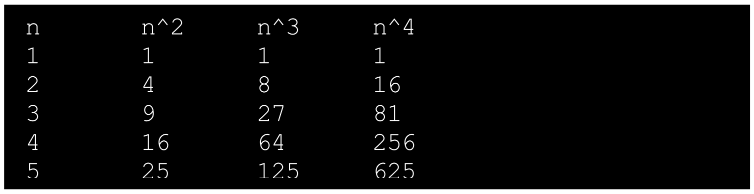
```
for (int n = 0; m < 10; n++)
{
      System.out.println( n );
}
```

*Logic error!*

*This is possibly an infinite loop in that you are comparing the wrong variable – the loop is using n, you are comparing m.*

**Part 9: For Loop Exercise**

You are to write a program that generates the second, third, and fourth powers of a list of whole numbers from 1 to n where n is input by the user. Write a Java program to do this. First, ask the user for the largest of the whole numbers to use (n). Second, output column headers (see below, n, n^2, n^3, n^4). Then, use a for loop to iterate from 1 to n, computing each of that loop variable to the second power, third power and fourth power. Assuming your loop variable is called i, you can do this either as i*i or Math.pow(i,2). To output the values in nice columns as shown below, separate each output with a tab (`"\t"`). This is similar to using \n for a new line.



```
n       n^2     n^3     n^4
1       1       1       1
2       4       8       16
3       9       27      81
4       16      64      256
5       25      125     625
```

Test your program with different input values such as 5, 10, 1, 0 and -1. For 0 and -1, you should get no output at all. If you wrote your loop correctly, that will be the case.

**Debugging tip:** If you get incorrect results when running code that contains a loop, place some print statements inside the loop to print the values of any variables that you think might be involved in the errors. This helps you check what your loop is actually doing.

**Part 10: One More Program**

Write one additional program which itself contains two loops. You will have to decide which loop type to use and how to write each one. Both loops will have a loop body that consists of a computation and an output.

- Loop 1: sum up all of the values from 1 to 10, outputting the sum as you go. For instance, it will output 1, 3 (1+2), 6 (1+2+3), 10 (1+2+3+4), etc (on separate lines).
- Loop 2: sum up all of the values starting at 1 and going until the sum is greater than 100, again outputting results as you go.

You will have to decide what type of loops to use and how to write each, and what other variables and instructions are needed in your program.

**Part 11: Submitting Your Assignment**

Print or email all 3 programs (parts 5, 9 and 10). You do not have to submit any output.