CSc 1301: PRINCIPLES OF COMPUTER SCIENCE

Lab 10 – Nested For loop practice and exercise with solutions

**Reviewing materials**

*public class NestingLoops*

*{*

*public static void main( String[] args )*

*{*

*// this is #1 - I'll call it "CN"*

*for ( char c='A'; c <= 'E'; c++ )*

*{*

*for ( int n=1; n <= 3; n++ )*

*{*

*System.out.println( c + " " + n );*

*}*

*}*

*System.out.println("\n");*

*// this is #2 - I'll call it "AB"*

*for ( int a=1; a <= 3; a++ )*

*{*

*for ( int b=1; b <= 3; b++ )*

*{*

*System.out.print( a + "-" + b + " " );*

*}*

*// \* You will add a line of code here.*

*}*

*System.out.println("\n");*

*}*

*}*

This series of examples works on developing a multiplication table, say, multiplying numbers from 1 to 12 by themselves. One needs doubly-nested for loops, with variables (say, i and j) each going from 1 to 12. The structure looks like this:

**for (int i = 1; i <= 12; i++)**

**for (int j = 1; j <= 12; j++)**

**// print the value of i\*j**

The first immediate problem has to do with skipping to a new line. That needs to come at the end of each row. Putting that in and enclosing the whole in a main function and a class, gives the following Java code to display the various products:

**// TimesTable.java: print a times table**

**public class TimesTable0**

**{**

**public static void main (String[] args) {**

**// print main table**

**for (int i = 1; i <= 12; i++) {**

**System.out.print(i + ":");**

**for (int j = 1; j <= 12; j++) {**

**System.out.print(i\*j + " ");**

**}**

**System.out.println();**

**}**

**} // end of main**

**}**

Output:

**1:1 2 3 4 5 6 7 8 9 10 11 12**

**2:2 4 6 8 10 12 14 16 18 20 22 24**

**3:3 6 9 12 15 18 21 24 27 30 33 36**

**4:4 8 12 16 20 24 28 32 36 40 44 48**

**5:5 10 15 20 25 30 35 40 45 50 55 60**

**6:6 12 18 24 30 36 42 48 54 60 66 72**

**7:7 14 21 28 35 42 49 56 63 70 77 84**

**8:8 16 24 32 40 48 56 64 72 80 88 96**

**9:9 18 27 36 45 54 63 72 81 90 99 108**

**10:10 20 30 40 50 60 70 80 90 100 110 120**

**11:11 22 33 44 55 66 77 88 99 110 121 132**

**12:12 24 36 48 60 72 84 96 108 120 132 144**

**Observations :**

**The columns are not lined up, and the table is almost useless. We need to arrange to print each product in 3 columns, whether they need three or not. One way to do this is to print an extra two blanks for single-digit numbers and to print one extra blank for double-digit numbers.**

**// TimesTable2.java: print a times table**

**public class TimesTable2**

**{**

**public static void main (String[] args) {**

**int mult;**

**// print main table**

**for (int i = 1; i <= 12; i++) {**

**System.out.print(i);**

**if (i < 10) System.out.print(" :");**

**else System.out.print(":");**

**for (int j = 1; j <= 12; j++) {**

**mult = i\*j;**

**System.out.print(" " + mult);**

**if (mult < 10) System.out.print(" ");**

**else if (mult < 100) System.out.print(" ");**

**}**

**System.out.println();**

**}**

**} // end of main**

**}**

Output (after several tries with altering the above code):

**1 : 1 2 3 4 5 6 7 8 9 10 11 12**

**2 : 2 4 6 8 10 12 14 16 18 20 22 24**

**3 : 3 6 9 12 15 18 21 24 27 30 33 36**

**4 : 4 8 12 16 20 24 28 32 36 40 44 48**

**5 : 5 10 15 20 25 30 35 40 45 50 55 60**

**6 : 6 12 18 24 30 36 42 48 54 60 66 72**

**7 : 7 14 21 28 35 42 49 56 63 70 77 84**

**8 : 8 16 24 32 40 48 56 64 72 80 88 96**

**9 : 9 18 27 36 45 54 63 72 81 90 99 108**

**10: 10 20 30 40 50 60 70 80 90 100 110 120**

**11: 11 22 33 44 55 66 77 88 99 110 121 132**

**12: 12 24 36 48 60 72 84 96 108 120 132 144**

**Observations :**

This is ugly. It looks ugly because the numbers are arranged in a row to the left (left-justified).

Try it again, to get the numbers right justified.

**// TimesTable1.java: print a times table**

**public class TimesTable1**

**{**

**public static void main (String[] args) {**

**int mult;**

**// print main table**

**for (int i = 1; i <= 12; i++) {**

**if (i < 10) System.out.print(" ");**

**System.out.print(i + ":");**

**for (int j = 1; j <= 12; j++) {**

**mult = i\*j;**

**if (mult < 10) System.out.print(" ");**

**else if (mult < 100) System.out.print(" ");**

**System.out.print(mult + " ");**

**}**

**System.out.println();**

**}**

**} // end of main**

**}**

Output:

**1: 1 2 3 4 5 6 7 8 9 10 11 12**

**2: 2 4 6 8 10 12 14 16 18 20 22 24**

**3: 3 6 9 12 15 18 21 24 27 30 33 36**

**4: 4 8 12 16 20 24 28 32 36 40 44 48**

**5: 5 10 15 20 25 30 35 40 45 50 55 60**

**6: 6 12 18 24 30 36 42 48 54 60 66 72**

**7: 7 14 21 28 35 42 49 56 63 70 77 84**

**8: 8 16 24 32 40 48 56 64 72 80 88 96**

**9: 9 18 27 36 45 54 63 72 81 90 99 108**

**10: 10 20 30 40 50 60 70 80 90 100 110 120**

**11: 11 22 33 44 55 66 77 88 99 110 121 132**

**12: 12 24 36 48 60 72 84 96 108 120 132 144**

It still doesn't look like a regular table, because the columns aren't labeled. Here is a fancier version. It also uses a separate function to insert extra blanks.

**// TimesTable.java: print a times table**

**public class TimesTable**

**{**

**public static void main (String[] args) {**

**// print header row**

**System.out.print(" |");**

**for (int i = 1; i <= 12; i++)**

**System.out.print(pad(i) + " ");**

**System.out.println();**

**// print separator**

**System.out.print("---+");**

**for (int i = 1; i <= 12; i++)**

**System.out.print("-----");**

**System.out.println();**

**// print main table**

**for (int i = 1; i <= 12; i++) {**

**System.out.print(pad(i) + "|");**

**for (int j = 1; j <= 12; j++) {**

**System.out.print(pad(i\*j) + " ");**

**}**

**System.out.println();**

**}**

**} // end of main**

**// pad: add blanks to make it 3 long**

**public static String pad(int x) {**

**String s = new String();**

**if (x < 10) s = " " + x;**

**else if (x < 100) s = " " + x;**

**else s = "" + x;**

**return s;**

**}**

**}**

Output:

**| 1 2 3 4 5 6 7 8 9 10 11 12**

**---+------------------------------------------------**

**1| 1 2 3 4 5 6 7 8 9 10 11 12**

**2| 2 4 6 8 10 12 14 16 18 20 22 24**

**3| 3 6 9 12 15 18 21 24 27 30 33 36**

**4| 4 8 12 16 20 24 28 32 36 40 44 48**

**5| 5 10 15 20 25 30 35 40 45 50 55 60**

**6| 6 12 18 24 30 36 42 48 54 60 66 72**

**7| 7 14 21 28 35 42 49 56 63 70 77 84**

**8| 8 16 24 32 40 48 56 64 72 80 88 96**

**9| 9 18 27 36 45 54 63 72 81 90 99 108**

**10| 10 20 30 40 50 60 70 80 90 100 110 120**

**11| 11 22 33 44 55 66 77 88 99 110 121 132**

**12| 12 24 36 48 60 72 84 96 108 120 132 144**

One more little change will allow a multiplecation table of any size:

**// TimesTableN.java: print a times table**

**public class TimesTableN**

**{**

**public static void main (String[] args) {**

**// Make table size a variable**

**final int N = 16;**

**// print header row**

**System.out.print(" |");**

**for (int i = 1; i <= N; i++)**

**System.out.print(pad(i) + " ");**

**System.out.println();**

**// print separator**

**System.out.print("---+");**

**for (int i = 1; i <= N; i++)**

**System.out.print("----");**

**System.out.println();**

**// print main table**

**for (int i = 1; i <= N; i++) {**

**System.out.print(pad(i) + "|");**

**for (int j = 1; j <= N; j++) {**

**System.out.print(pad(i\*j) + " ");**

**}**

**System.out.println();**

**}**

**} // end of main**

**// pad: add blanks to make it 3 long**

**public static String pad(int x) {**

**String s = new String();**

**if (x < 10) s = " " + x;**

**else if (x < 100) s = " " + x;**

**else s = "" + x;**

**return s;**

**}**

**}**

Output:

**| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16**

**---+----------------------------------------------------------------**

**1| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16**

**2| 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32**

**3| 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48**

**4| 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64**

**5| 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80**

**6| 6 12 18 24 30 36 42 48 54 60 66 72 78 84 90 96**

**7| 7 14 21 28 35 42 49 56 63 70 77 84 91 98 105 112**

**8| 8 16 24 32 40 48 56 64 72 80 88 96 104 112 120 128**

**9| 9 18 27 36 45 54 63 72 81 90 99 108 117 126 135 144**

**10| 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160**

**11| 11 22 33 44 55 66 77 88 99 110 121 132 143 154 165 176**

**12| 12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192**

**13| 13 26 39 52 65 78 91 104 117 130 143 156 169 182 195 208**

**14| 14 28 42 56 70 84 98 112 126 140 154 168 182 196 210 224**

**15| 15 30 45 60 75 90 105 120 135 150 165 180 195 210 225 240**

**16| 16 32 48 64 80 96 112 128 144 160 176 192 208 224 240 256**

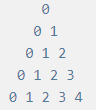
**Assignment**

**Assignments need to finish and submit for grade (submit lab10.java)**

You must use a Nested for and start for Loop – feel free to modify for

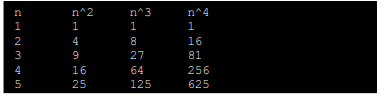
For Loop Exercise Part a

Write a method used a nested for loop called nesty\_MeNumbers that will produce the exact numbers.



For Loop Exercise Part b

You are to write a method called powers() 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.



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.

**Hint\*\*\*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 c: write a method called q3()

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. Expected output:

A picture containing table

Description automatically generated