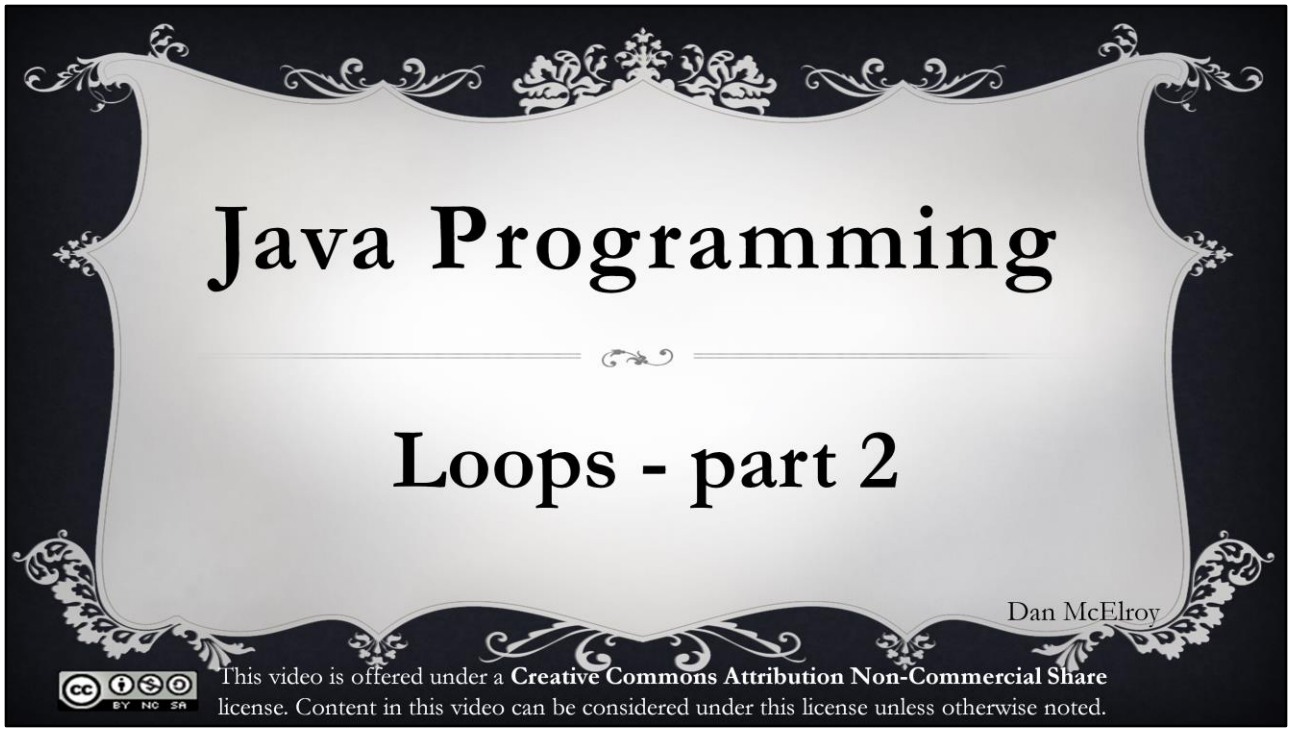
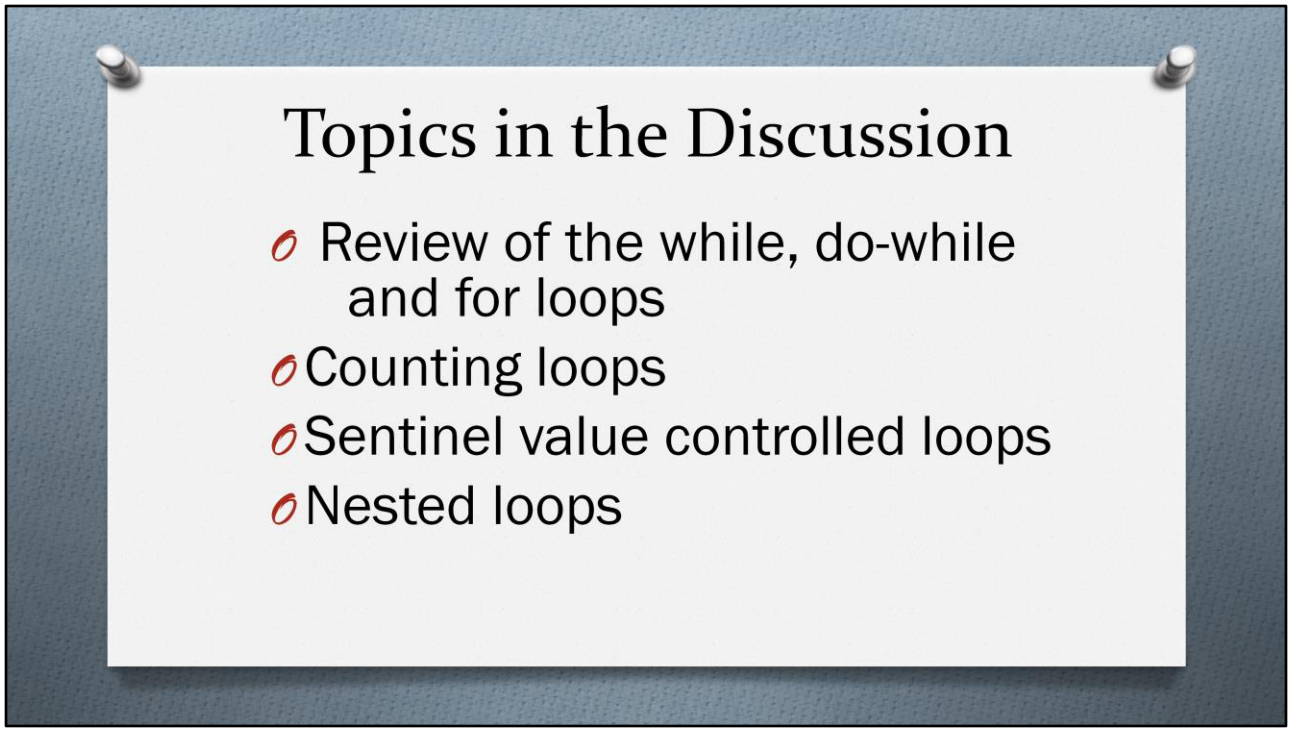


Hello programmers. A big giant welcome to more about Java Loops!



Topics in this discussion include:
A review of the while, do-while and for loops
Counting loops
Sentinel value controlled loops
Nested loops

while and **do-while** loops

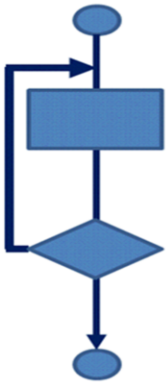
while

```
int i = 0;
while ( i < 5 ) {
   System.out.println (i);
   i++;
}
System.out.println ("Done“ );
```

```
0
1
2
3
4
Done
```

The test for looping is done at the top of the loop. Code may not be executed if the loop condition fails.

do . . . while



```
int i = 0;
do {
   System.out.println (i);
   i++;
} while ( i < 5 ) ;
System.out.println ("Done“ );
```

```
0
1
2
3
4
Done
```

The test for looping is done at the bottom of the loop. Code will be executed at least one time.

The while loop has its loop test condition at the top of the loop. It is possible that the body of the loop will not be executed if the test condition evaluates to a false.

The do-while loop has the loop test condition at the bottom of the loop. The body of the loop is guaranteed to execute at least one time.

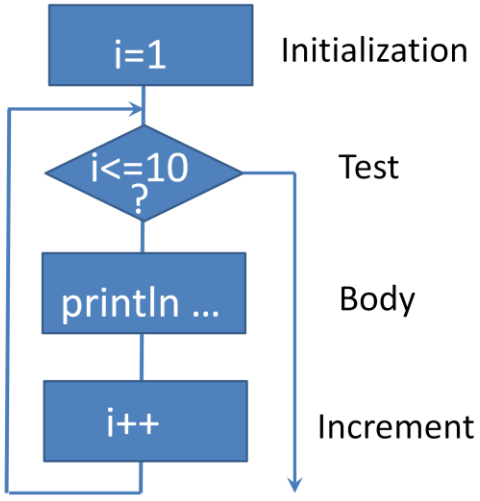
Both of these examples are counting loops. The number of times through the loop is determined before the loop even starts.

The **for** loop

The **for** loop in Java is a fancy **while** loop.

```
// display 1 through 10
int i;
i=1;
while (i<=10) {
  System.out.println(i);
  i++;
}
```

```
int i;
for (i=1; i<=10; i++) {
   System.out.println(i);
}
```

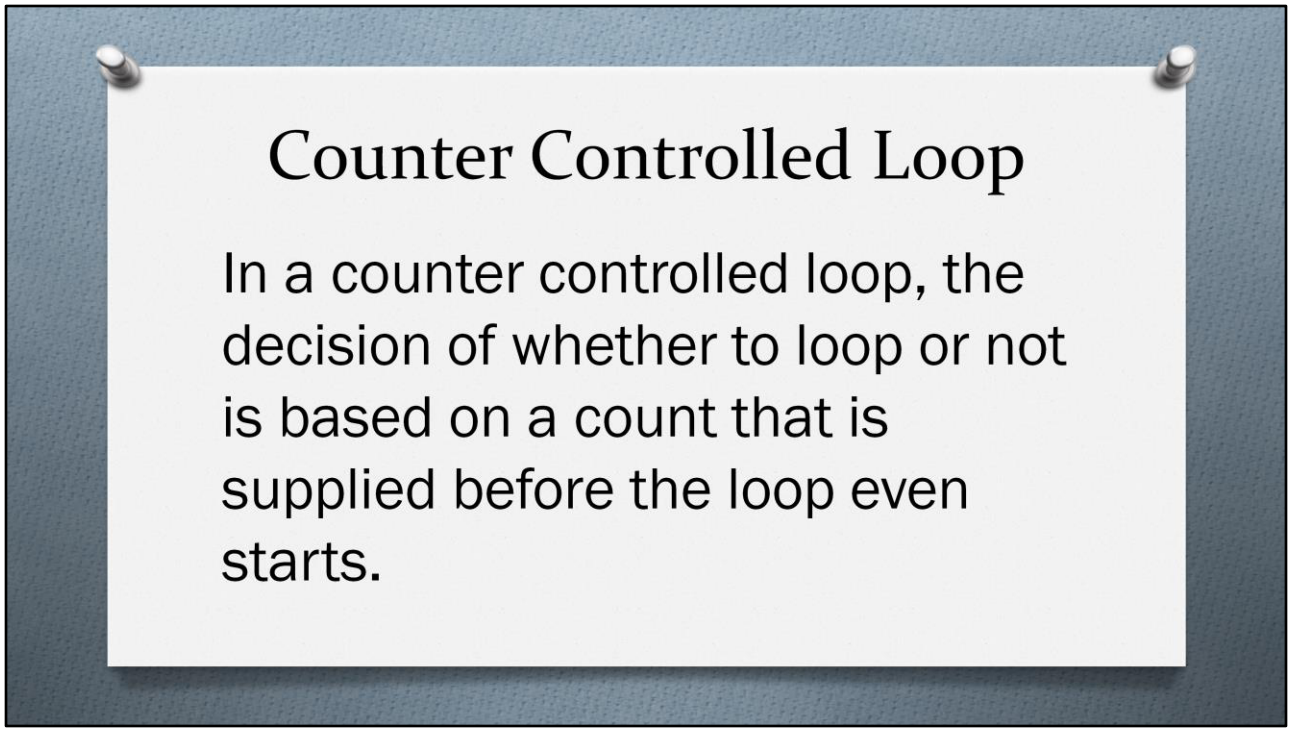


The for loop is just a fancy while loop. Although the initialization, loop control and increment are all placed in the for statement, these three pieces of code are executed at different times.

The initialization occurs before the loop even starts.
The test is done at the top before the body of the loop.
The increment happens after everything in the body has executed.

Here is a comparison between the for loop and the while loop when used to go through the loop a specified number of times.

The variable i is declared as an integer. i is initialized to 1 before the loop even starts. The first thing that happens in the loop is the test to see if the body of the loop is to be executed. If the test evaluates to true, then the body is executed. The last thing inside the loop is the increment, i++



In a counter controlled loop, the decision of whether to loop or not is based on a count that is supplied before the loop even starts.

Sentinel Value Controlled Loop

In a sentinel value controlled loop, the decision of whether to loop or not to loop is determined by data input from the user from within the loop.

In a sentinel value controlled loop, the decision of whether to loop or not to loop is determined by data input from the user from within the loop.

Nested Loops

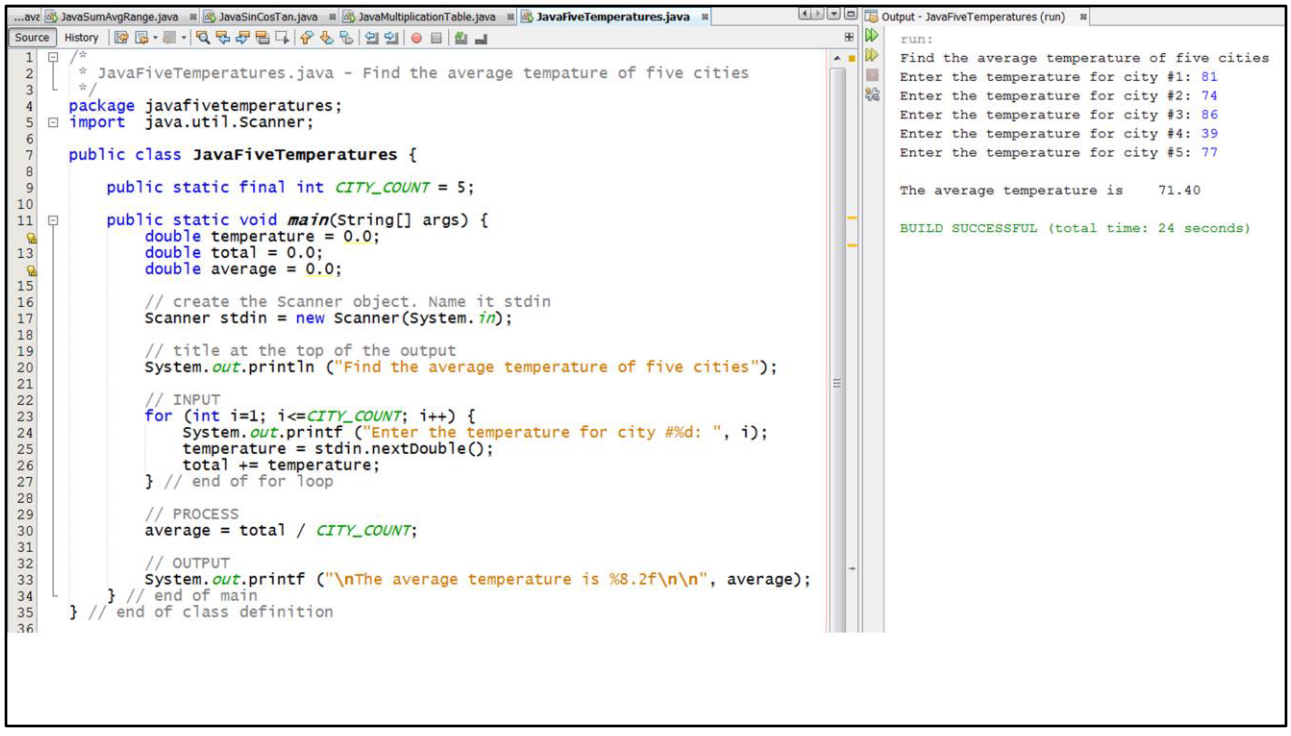
A nested loop is a second loop that occurs within the body of the first loop.

A nested loop is a second loop that occurs within the body of the first loop.

Counter Controlled Loop Examples

1. Find the average temperature of five cities
2. Display a trigonometry table for sine, cosine and tangent from 0 to 90 degrees in 3 degree increments

Here are two examples of counter controlled loops.
1. Find the average temperature of five cities
2. Display a trigonometry table for sine, cosine and tangent from 0 to 90 degrees in 3 degree increments



```
/*
 * JavaFiveTemperatures.java - Find the average tempature of five cities
 */
package javafivetemperatures;
import  java.util.Scanner;

public class JavaFiveTemperatures {

    public static final int CITY_COUNT = 5;

    public static void main(String[] args) {
        double temperature = 0.0;
        double total = 0.0;
        double average = 0.0;

        // create the Scanner object. Name it stdin
        Scanner stdin = new Scanner(System.in);

        // title at the top of the output
        System.out.println ("Find the average temperature of five cities");

        // INPUT
        for (int i=1; i<=CITY_COUNT; i++) {
            System.out.printf ("Enter the temperature for city #%d: ", i);
            temperature = stdin.nextDouble();
            total += temperature;
        } // end of for loop

        // PROCESS
        average = total / CITY_COUNT;

        // OUTPUT
        System.out.printf ("\nThe average temperature is %8.2f\n\n", average);
    } // end of main
} // end of class definition
```

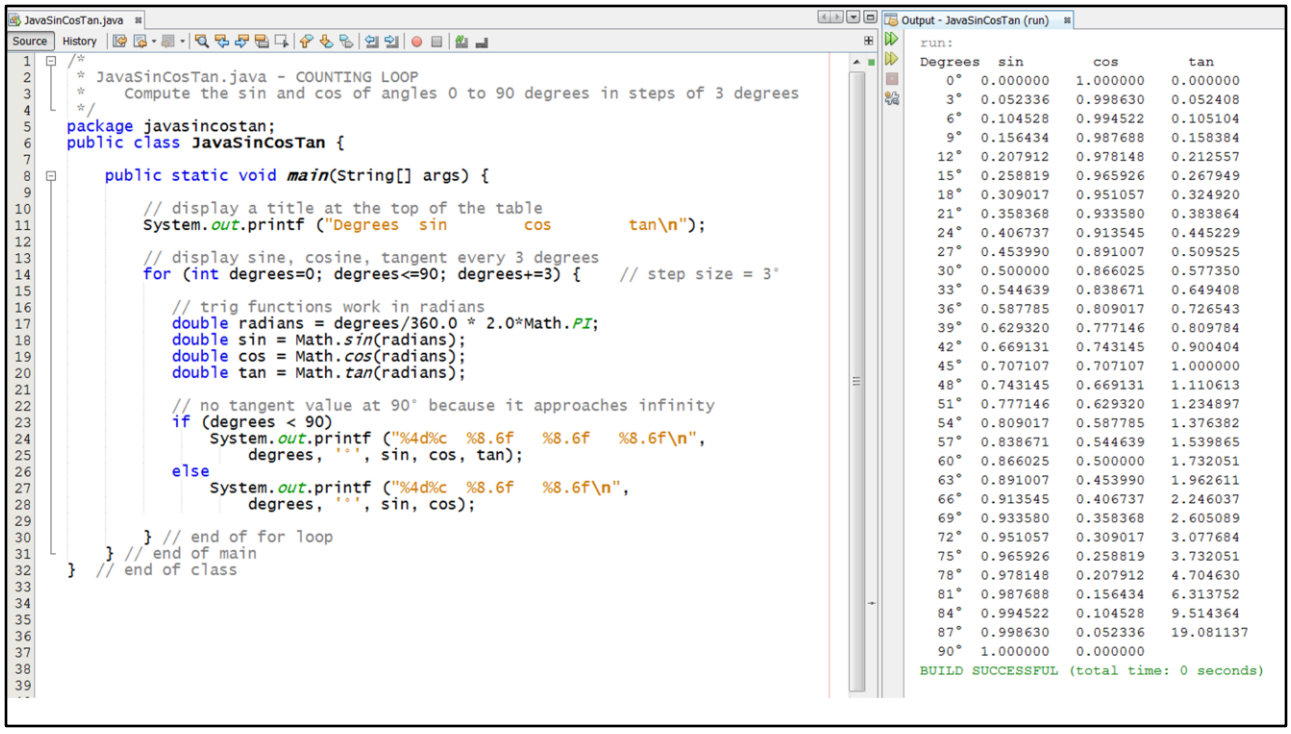
```
run:
Find the average temperature of five cities
Enter the temperature for city #1: 81
Enter the temperature for city #2: 74
Enter the temperature for city #3: 86
Enter the temperature for city #4: 39
Enter the temperature for city #5: 77

The average temperature is    71.40

BUILD SUCCESSFUL (total time: 24 seconds)
```

This program reads temperatures from five cities, and then computes and displays the average temperature.

In this example, the count of the number of cities is defined with a constant at the top of the program.

public static final int CITY_COUNT = 5;

Instead of using a constant, the program could have also asked the user how many cities there were in the list, read that value from the keyboard and then used it as the count. In either case, the number of times through the loop is known at the start of the loop. That is why it is called a counting loop.



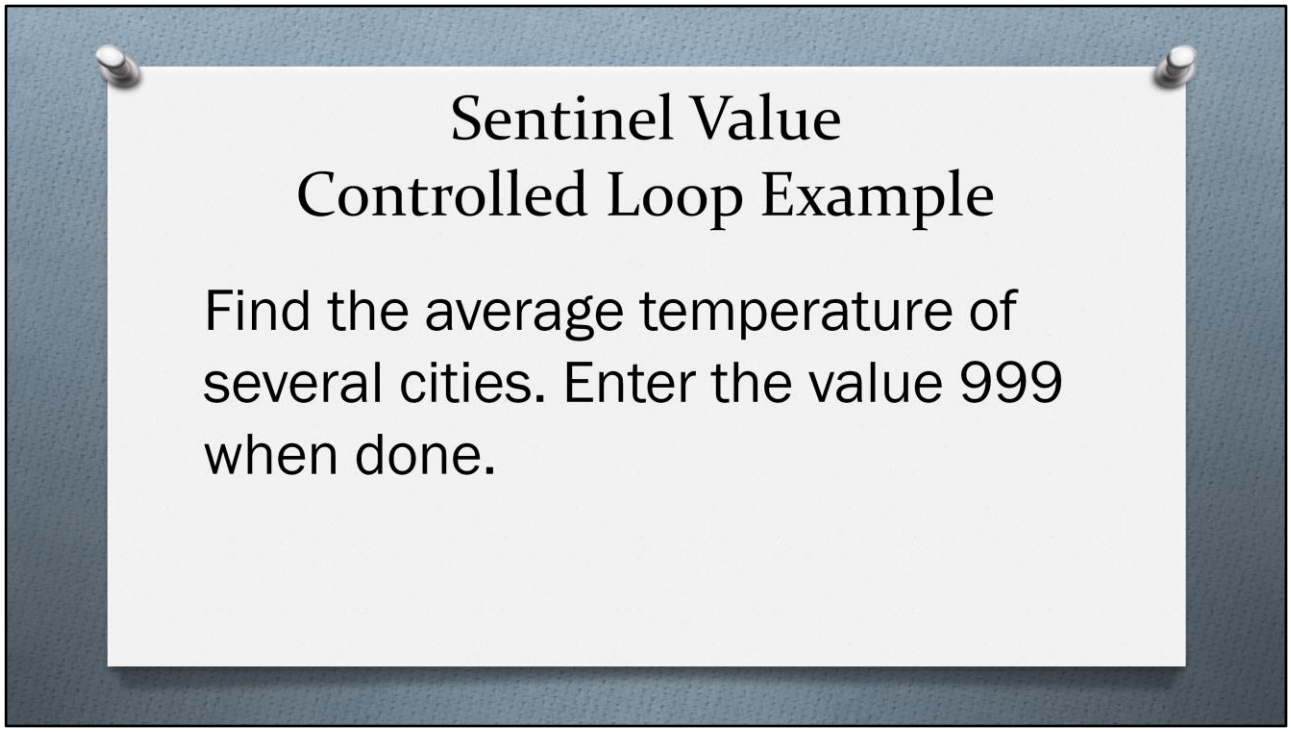
```
/*
 * JavaSinCosTan.java - COUNTING LOOP
 *    Compute the sin and cos of angles 0 to 90 degrees in steps of 3 degrees
 */
package javasincostan;
public class JavaSinCosTan {

    public static void main(String[] args) {

        // display a title at the top of the table
        System.out.printf ("Degrees  sin        cos        tan\n");

        // display sine, cosine, tangent every 3 degrees
        for (int degrees=0; degrees<=90; degrees+=3) {    // step size = 3°

           // trig functions work in radians
           double radians = degrees/360.0 * 2.0*Math.PI;
           double sin = Math.sin(radians);
           double cos = Math.cos(radians);
           double tan = Math.tan(radians);

           // no tangent value at 90° because it approaches infinity
           if (degrees < 90)
               System.out.printf ("%4d%c  %8.6f   %8.6f   %8.6f\n",
                   degrees, '°', sin, cos, tan);
           else
               System.out.printf ("%4d%c  %8.6f   %8.6f\n",
                   degrees, '°', sin, cos);

        } // end of for loop
    } // end of main
}  // end of class
```

```
run:
Degrees  sin        cos        tan
   0°  0.000000   1.000000   0.000000
   3°  0.052336   0.998630   0.052408
   6°  0.104528   0.994522   0.105104
   9°  0.156434   0.987688   0.158384
  12°  0.207912   0.978148   0.212557
  15°  0.258819   0.965926   0.267949
  18°  0.309017   0.951057   0.324920
  21°  0.358368   0.933580   0.383864
  24°  0.406737   0.913545   0.445229
  27°  0.453990   0.891007   0.509525
  30°  0.500000   0.866025   0.577350
  33°  0.544639   0.838671   0.649408
  36°  0.587785   0.809017   0.726543
  39°  0.629320   0.777146   0.809784
  42°  0.669131   0.743145   0.900404
  45°  0.707107   0.707107   1.000000
  48°  0.743145   0.669131   1.110613
  51°  0.777146   0.629320   1.234897
  54°  0.809017   0.587785   1.376382
  57°  0.838671   0.544639   1.539865
  60°  0.866025   0.500000   1.732051
  63°  0.891007   0.453990   1.962611
  66°  0.913545   0.406737   2.246037
  69°  0.933580   0.358368   2.605089
  72°  0.951057   0.309017   3.077684
  75°  0.965926   0.258819   3.732051
  78°  0.978148   0.207912   4.704630
  81°  0.987688   0.156434   6.313752
  84°  0.994522   0.104528   9.514364
  87°  0.998630   0.052336   19.081137
  90°  1.000000   0.000000
BUILD SUCCESSFUL (total time: 0 seconds)
```

Here is another example of a counting loop. The program displays the trig tables for sine, cosine and tangent from 0 to 90 degrees in increments of 3 degrees. Look at the increment within the for loop statement. It has degrees+=3 Any valid C or C++ statement can be used within the three parts of the for loop, initialization, test and increment. Just make sure that the test condition evaluates to true or false, and make sure that it will eventually evaluate to false to cause the loop to end.

One thing to remember is that radians are typically used for trig functions in computer programs. There are 2pi radians that make up a full circle, just as there are 360 degrees in a full circle. To convert degrees to radians, multiply by 0.01745329251994329576923690768489. That number is too hard to remember. It is easier to take the ratio of degrees to 360 and multiply it by 2 pi. Java has a constant already defined for pi. It is Math.PI

Once the degrees have been converted to radians, use Java's Math functions to find the values for sine, cosine and tan. The functions are called: sin, cos and tan. Display these values on the screen. Since the tangent of 90 degrees approaches infinity, the value for the tangent at 90 degrees is not displayed.



Remember the definition of a sentinel value controlled loop. With a sentinel value controlled loop, the decision of whether to loop or not to loop is determined by data input from the user from within the loop.

Here is an example of using a sentinel value controlled loop. Find the average temperature of several cities. Enter the value 999 when done.

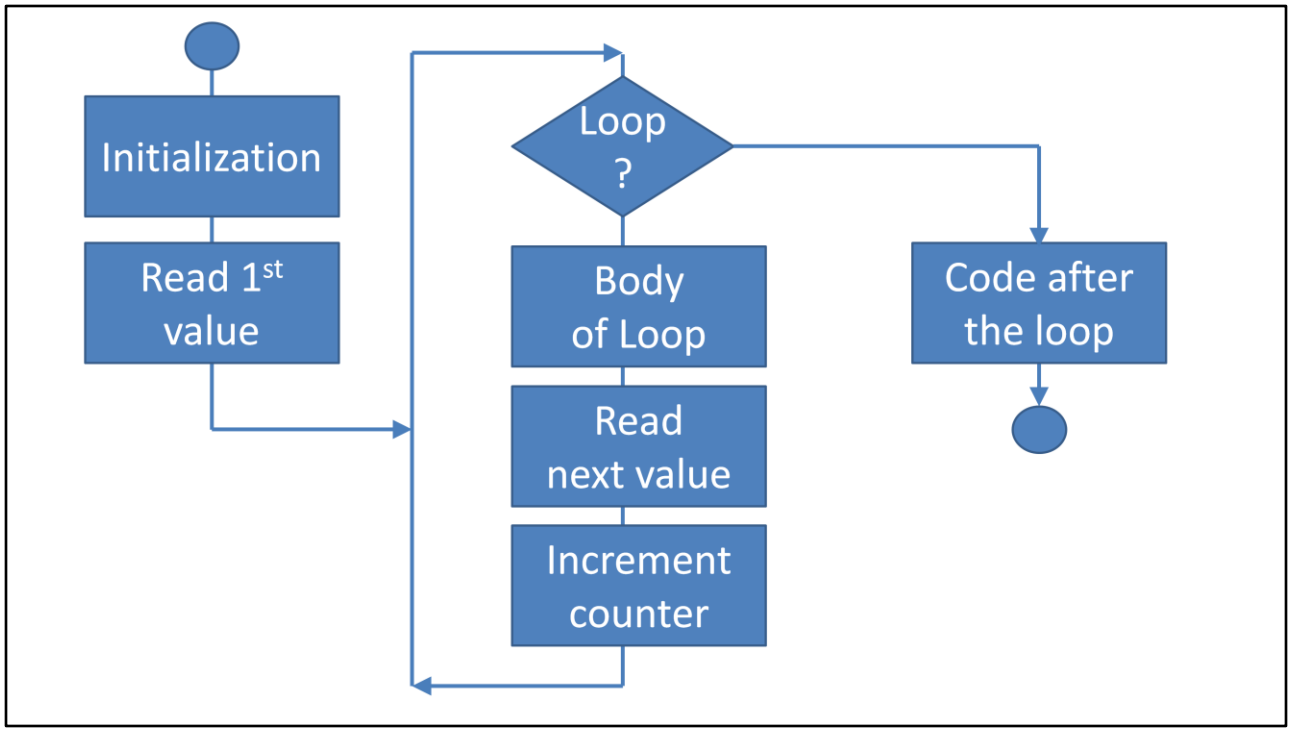
One of the most important things to consider when creating the code for a sentinel value controlled loop is to make sure that the 'stop' value that the user needs to input is both a legal value for the data type, AND a value that would not be used as a valid input. Zero and negative one are not a good choices when inputting temperatures because it is possible to get values less than zero. Once you see many programs, you will often see 999 or 9999 used to end a list of data values. Once we start working with data on a disk, we can use the end-of-file to indicate that we have reached the end of data. The end-of-file can also be indicated on a keyboard by typing Ctrl+D on a PC or Ctrl+Z on a Mac or Linux systems.

Sentinel Value Controlled Loop

- The 'stop' value must be out of range for normal input data. Use something like 0, -1, 999, etc.
- Read the first value before the loop even starts
- Inside the loop, process the input value
- At the bottom of the loop, read the next value
- Increment the loop counter if needed

One of the most important things to consider when creating the code for a sentinel value controlled loop is to make sure that the 'stop' value that the user needs to input is both a legal value for the data type, AND a value that would not be used as a valid input. Zero and negative one are not a good choices when inputting temperatures because it is possible to get values less than zero. Once you see many programs, you will often see 999 or 9999 used to end a list of data values. Once we start working with data on a disk, we can use the end-of-file to indicate that we have reached the end of data. The end-of-file can also be indicated on a keyboard by typing Ctrl+D on a PC or Ctrl+Z on a Mac or Linux systems.

The sentinel value controlled loop reads the first value before the loop even starts. Once inside the loop, that value is processed and any additional processing takes place. Although it may look weird, the next value is read at the bottom of the loop and the loop counter is incremented.

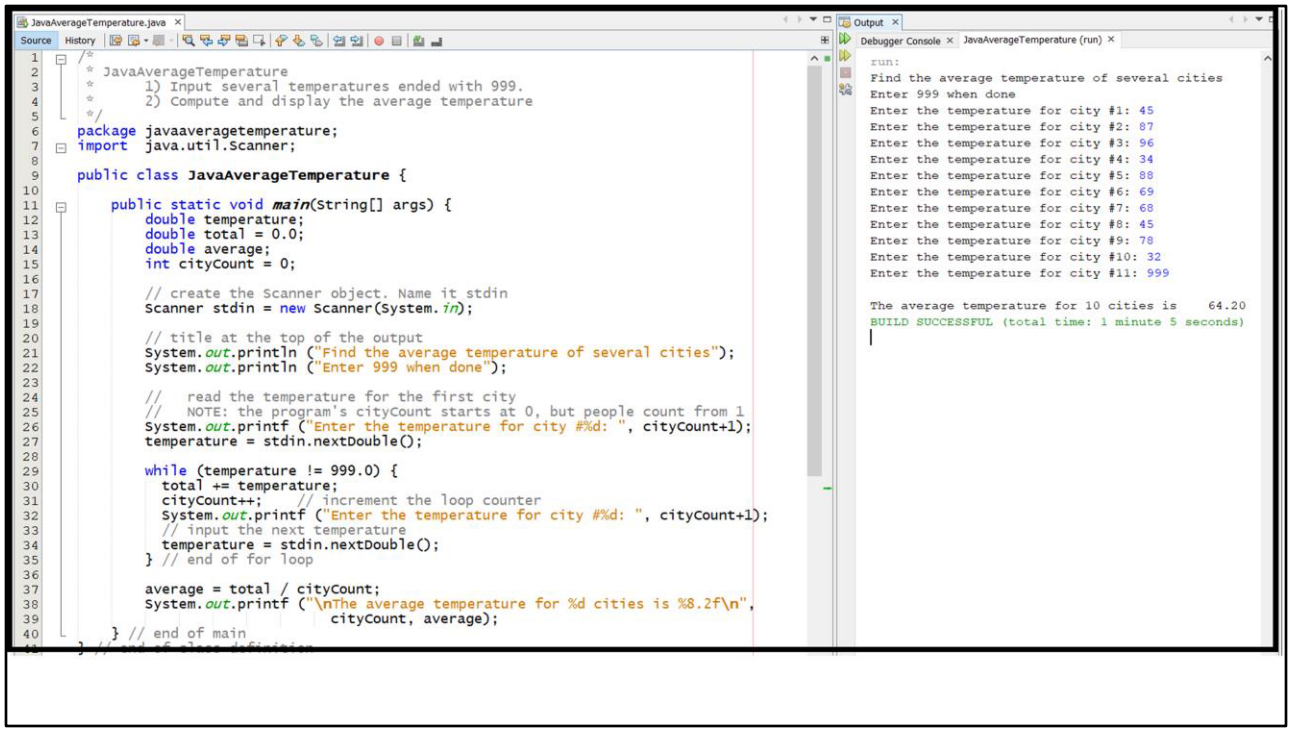


Here is a flowchart for a sentinel value controlled loop. During the initialization, the loop counter is set to 0 and any other required variables are declared and initialized as needed.

The first value is read, including a prompt message if needed. It is important that the data value be read before deciding on whether to do the body of the loop. This is the reason that the input is not done at the top of the loop. Maybe there is no data to be processed and the loop ends right away. The other thing to consider is that when the 'stop' value is read, the loop needs to end without processing it.

If we read the input value at the top of the loop and always processed that value before deciding on whether or not to end the loop, in the example of using 999 as the 'stop' value, the 999 would be considered as one of the temperatures and we would count this as one of the cities before ending the loop. Our average would definitely be thrown off by a huge amount.

We also want to end the loop if 999 is entered as the first value to indicate that there was no data.



```
/*
 * JavaAverageTemperature
 *      1) Input several temperatures ended with 999.
 *      2) Compute and display the average temperature
 */
package javaaveragetemperature;
import  java.util.Scanner;

public class JavaAverageTemperature {

    public static void main(String[] args) {
        double temperature;
        double total = 0.0;
        double average;
        int cityCount = 0;

        // create the Scanner object. Name it stdin
        Scanner stdin = new Scanner(System.in);

        // title at the top of the output
        System.out.println ("Find the average temperature of several cities");
        System.out.println ("Enter 999 when done");

        //   read the temperature for the first city
        //   NOTE: the program's cityCount starts at 0, but people count from 1
        System.out.printf ("Enter the temperature for city #%d: ", cityCount+1);
        temperature = stdin.nextDouble();

        while (temperature != 999.0) {
          total += temperature;
          cityCount++;    // increment the loop counter
          System.out.printf ("Enter the temperature for city #%d: ", cityCount+1);
          // input the next temperature
          temperature = stdin.nextDouble();
        } // end of for loop

        average = total / cityCount;
        System.out.printf ("\nThe average temperature for %d cities is %8.2f\n",
                           cityCount, average);
    } // end of main
} // end of class definition
```

```
run:
Find the average temperature of several cities
Enter 999 when done
Enter the temperature for city #1: 45
Enter the temperature for city #2: 87
Enter the temperature for city #3: 96
Enter the temperature for city #4: 34
Enter the temperature for city #5: 88
Enter the temperature for city #6: 69
Enter the temperature for city #7: 68
Enter the temperature for city #8: 45
Enter the temperature for city #9: 78
Enter the temperature for city #10: 32
Enter the temperature for city #11: 999

The average temperature for 10 cities is    64.20
BUILD SUCCESSFUL (total time: 1 minute 5 seconds)
```

Here is the code and a sample output. The cityCount is initialized to 0 and the first temperature is read before the loop starts. The while statement decides whether or not a 999 was entered. If not, then the body of the loop is executed. We add the input temperature to the total in the body of the loop. At the end of the body, we read another temperature and increment the cityCount. We need the cityCount when determining the average temperature.

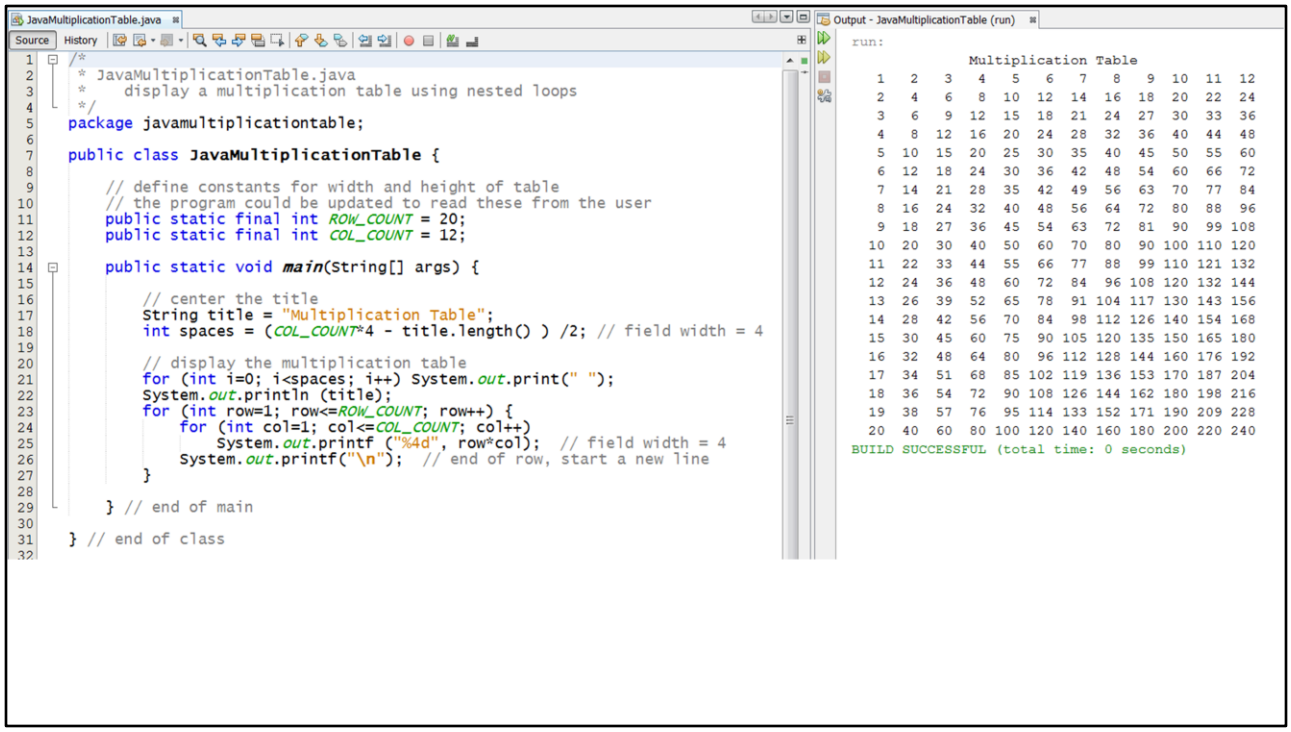
After the loop ends, we can compute the average and display the city count and the average temperature.

Nested Loop Example

Display a multiplication table of rows and columns with a title at the top of the table

A nested loop has one loop inside another loop. Displaying a multiplication table is a great example for nested loops.

The outer loop is used for each line. Another loop is used within the body of the first loop to display a column of numbers. At the end of the line, the cursor is moved down to the next line and another column of numbers is displayed.

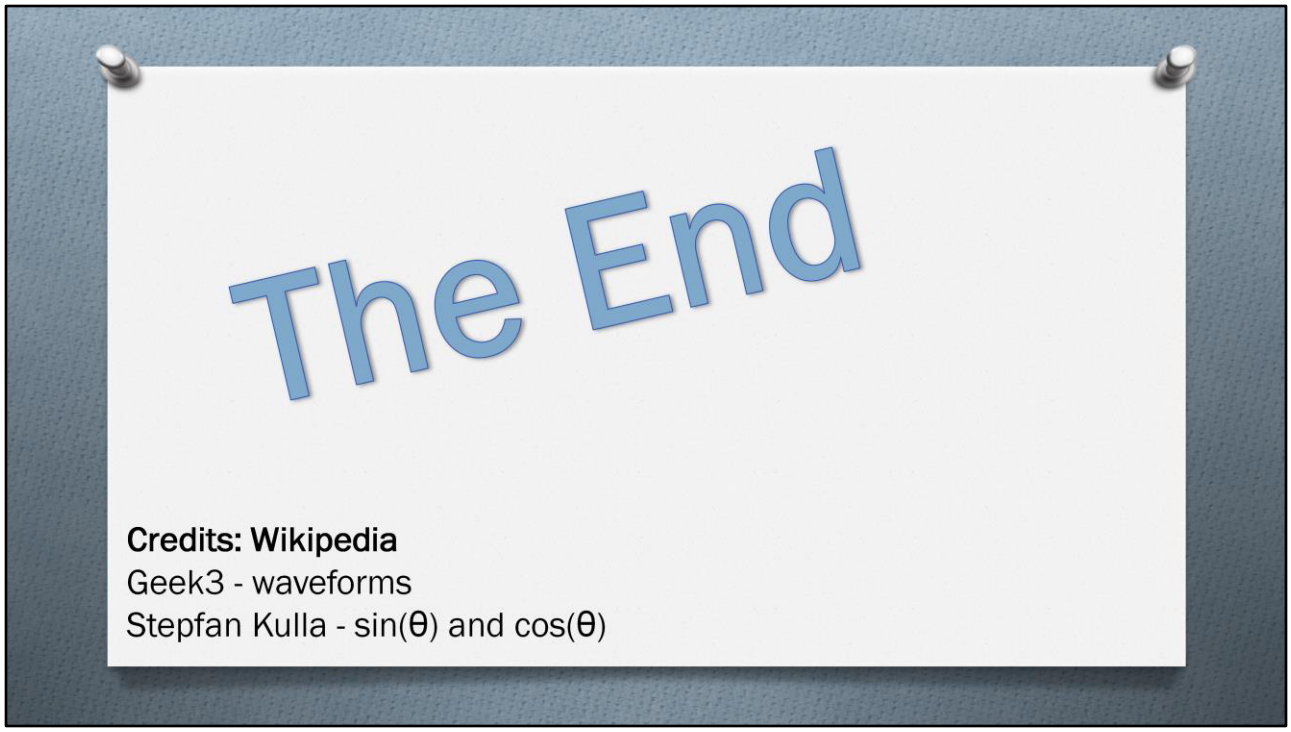


Use good names for the variables, such as row and col to indicate the row (line) or column for each value that will be displayed. This program uses constants at the top of the program to define the number of rows and columns that will be displayed. The program could be easily modified to input these numbers from the user's keyboard instead of using constants.

It is important to know what the largest value that will be displayed so that the columns can be lined up together. A field width of 4 will give room for a space character and three digits for each value being displayed. That is why the printf statement has ("%4d", row*col).

There is also a title at the top of the multiplication table that is centered above the table. To center the title above the table, we need to know the width of the table and the width of the title. The width of the table is the number of columns * 4 because there are 4 positions on the screen used for each number and its spaces. The width of the title can be determined by the length of the string. title.length()

The width of the table divided by 2 would start the title at the mid-point position of the table. We want the title centered. This can be done by subtracting the width of the title divided by 2. Since both values are divided by 2, it is just as easy to do (COL_COUNT*4 - title.length()) /2;



Loops are very powerful. The more you use them, the better you will become. You will find loops in almost every program except the tiniest of them all.

Bye bye