Nested Loops

A nested loop simply means that a one loop, usually referred to as the “inner loop,” is located within the code block of another loop, usually called the “outer loop.”
While this concept is simple enough, the interactions between the inner and outer loops can lead to very complex behavior!

In JAVA, each loop can be either a “while” or a “for” loop.

A simple example looks like this:

Program:

```java
public class NestedLoopsExample1 {
   public static void main(String[] args) {
      for (int outer=1; outer<=3; outer++) {
         for (int inner=1; inner<=3; inner++) {
            System.out.println("outer="+outer+", inner="+inner);
         }
      }
   }
}
```

Console Output:

```
outer=1, inner=1
outer=1, inner=2
outer=1, inner=3
outer=2, inner=1
outer=2, inner=2
outer=2, inner=3
outer=3, inner=1
outer=3, inner=2
outer=3, inner=3
```

Looking at the console output closely, you might notice that every combination of outer and inner values is represented once. In other words, for each value of outer from 1 to 3, inner also has the value of 1 to 3. This means that the inner loop iterates a total of 9 times. This is true because the inner loop iterates 3 times for each time the outer loop iterates and the outer loop iterates 3 times!

It is important to be able to recognize when it makes sense to use nested loops to solve a problem. For example, if you were trying to get the following console output, then the program on the right would work:

Console Output:

```
123
123
123
```

Program:

```java
public class NestedLoopsExample2 {
   public static void main(String[] args) {
      System.out.println("123");
      System.out.println("123");
      System.out.println("123");
   }
}
```

But is this code efficient?

What will happen if the pattern we want changes, to have 99 lines with the numbers 1 through 10 in them?

Using a loop will add flexibility.

The same program could be written using a loop as:

Program:

```java
public class NestedLoopsExample3 {
   public static void main(String[] args) {
      for (int a=1; a<=3; a++) {
         System.out.println("123");
      }
   }
}
```

Console Output:

```
123
123
123
```

In this version of the program it would now be easy to have 99 lines instead of just three. Simply changing the for loop condition to a<=99 is all it would take!

However, it still does not provide the flexibility to easily change the range of numbers displayed on each line. In order to get that flexibility we need to use a “nested” loop!

Since we want both the number of rows to be easily changed and the digits represented in each row to also be easily changed, we will replace the simple `System.out.println("123");` with a loop that prints the digits we want for each row:

Program:

```
public class NestedLoopsExample4 {
   public static void main(String[] args) {
      for (int a=1; a<=3; a++) {
         for (int b=1; b<=3; b++) {
            System.out.print(a);
         }
         System.out.println();
      }
   }
}
```

Console Output:

```
123
123
123
```

This version of the program makes it easy to change both the number of rows to output and the digits printed on each row.

To get 1 through 10 output 99 times would only require changing the outer loop's conditional expression to `a<=99` and the inner loops conditional expression to `b<=10`.

Furthermore, if the desired console output was more complex, the relative values of the outer and inner loops can be used. For example, taking advantage of both the inner and outer loop variables can generate the pattern below:

Console Output:

```
123
234
345
```

Program:

```
public class NestedLoopsExample5 {
   public static void main(String[] args) {
      for (int a=0; a<=2; a++) {
         for (int b=1; b<=3; b++) {
            System.out.print(a+b);
         }
         System.out.println();
      }
   }
}
```

Notice that two lines of code have been changed.

First the outer loop variable is initialized to `0` instead of `1` and the loop condition has been changed to `a<=2`.

Secondly, the console output has been altered so that the current values of the variables `a` and `b` are added together!

After deciding to use a nested loop, figuring out the correct initial values to use for the loop variable, the loop condition, and the loop variable modifier for an outer and inner loop is the real challenge.

If the desired pattern is:

```
987
654
321
```

Then first decide upon the outer loop's initial value. Since the rows start 9, 6, and 3, and there are three rows, it makes sense to start the outer loop variable with a value of 9, and then reduce it by 3 on each iteration while it is greater than or equal to 3:

`for (int r=9; r>=3; r=r-3).`

Now it is time to decide what values to use for the inner loop. In this case, first it is clear that the loop must iterate 3 times. Next an initial value for the inner loop needs to be chosen. If this value is going to be combined with the outer variable's value, then the inner loop's initial value must be 0 so the first digit of each line maintains the pattern. Only the amount to modify the inner loop variable by remains. Having this value decrease by 1 each iteration allows the two loop variable's values to be added to get the desired output:

`for (int c=0; c>=-2; c--).`

```
public class NestedLoopsExample6 {
   public static void main(String[] args) {
      for (int r=9; r>=3; r=r-3) {
         for (int c=0; c>=-2; c--) {
            System.out.print(r+c);
         }
         System.out.println();
      }
   }
}
```