Repetition using Java – Simple `for` Statements

The following is a practical example to illustrate the `while statement`. Depending upon the condition at the beginning of the `while` loop, the code inside the loop may be executed zero, once, or many times.

```
while (there are lumps in the gravy)
  give the gravy a stir
end while
```

We have compared this to the `do statement`, where the code inside the loop will be executed *at least* once, and possibly many times.

```
do
  give the gravy a stir
while (there are lumps in the gravy)
```

The third, and last, type of loop in Java uses the for statement, and it is a counted loop. In terms of our gravy example, it might look something like

```
stir the gravy 5 times
```

The For Statement – A Counted Loop

In a more Java-like form, we might write

```
for (int count = 1; count <= 5; count++)
{
     <stir the sauce>;
}
```

It is always possible to rewrite the for statement using a while statement, as follows.

```
int count = 1;
while (count <= 5)
{
     <stir the sauce>;
     count++;
}
```

The `for` statement actually contains three very important components, which are hopefully illustrated by the equivalent `while` statement:

1. **Initialization** – the variable used to count the loop is set to an initial value
2. **Boolean Expression** – controls the loop by testing the variable
3. **Modifier** – changes the variable (usually up or down by one)

All of the same steps occur in the `for` loop. The counting variable is declared and initialized (`count = 1`), the boolean condition is checked (`count <= 5`), and the counter is *incremented* for each iteration of the loop. The main differences are (a) how compact the code is, and (b) that the structures makes it obvious that it is a counting loop.

Another change in the code worth noting is *how* we incremented the counter. The "++" at the end of `count++` is called the **increment operator**, and it increases the value of an integer variable by one. There is also a **decrement operator**, which reduces the value of an integer variable by one.

`count++`	is equivalent to	`count = count + 1`
`count--`	is equivalent to	`count = count - 1`

When counting through a loop, you may also see many examples where the counting starts at zero, instead of one. Our examples from above would be slightly modified:

The for loop, starting at zero, to stir the sauce 5 times. How have we changed the code?

```
for (int count = 0; count < 5; count++)
{
      <stir the sauce>;
}
```

The while loop, starting at zero, to stir the sauce 5 times. How have we changed the code?

```
int count = 0;
while (count < 5)
{
      <stir the sauce>;
      count++;
}
```

In both examples, starting at zero or starting at one, the loop executes five times. This is accomplished through the use of "less than" (<) vs "less than or equal to" (<=) to control when we stop each loop.

1, 2, 3, 4, 5 – the loop starts at one and continues while less than or equal to five

0, 1, 2, 3, 4 – the loop starts at zero and continues while less than five (which is four)

Condensed (Short Form) Mathematical Operations

Programmers can be so concerned with efficiency that they develop shorter ways, or a short form, for writing some common operations. Consider the following mathematical operations:

Operation	**Long Form**	**Short Form**
Increment (by 1)	`x = x + 1`	`x++`
Decrement (by 1)	`x = x - 1`	`x--`
Addition	`x = x + 5`	`x += 5`
Subtraction	`x = x - 2`	`x -= 2`
Multiplication	`x = x * 5`	`x *= 5`
Division	`x = x / 10`	`x /= 10`
Modulo (remainder)	`x = x % 3`	`x %= 3`

Repetition using Java – Simple `for` Statements

Example 1 – The following fragment finds the sum of the numbers from 1 to 100,
$1+2+3+\cdots+99+100$

```
int sum = 0;
for (int i = 1; i <= 100; i++)
{
     sum = sum + i;   // for each value of i, add it to sum
}
System.out.println("The sum from 1 to 100 is " + sum);
```

Example 2 – Similar to the example above, we allow the user to specify the ending value, giving the sum of integers from 1 to N, $1+2+3+\cdots+N$

```
int sum = 0;
int n = In.getInt();
for (int i = 1; i <= n; i++)
{
     sum = sum + i;
}
System.out.println("The sum from 1 to " + n " is " + sum);
```

Example 3 – This fragment prints the sum of squares from 1 to 100: $1^2+2^2+3^2+\cdots+99^2+100^2$

```
int sumSquares = 0;
for (int i = 1; i <= 100; i++)
{
     sumSquares += i*i;
}
System.out.println("The sum of squares from 1 to 100 is "
                   + sumSquares);
```

Example 4 – This program fragment prints the values of the squares ($y=x^2$), but only for every *third* integer less than 20 (i.e., 1, 4, 7, 10, 13, 16, 19). Notice that although our condition stops at 20, the last number whose square is printed is 19, as the next number, 22, would fail the condition.

```
for (int x = 1; x < 20; x += 3)
{
     System.out.println("x = " + x + "  x^2: " + (x*x));
}
```

Exercises

1. What does each statement print?

a)
```
for (int i = -3; i <= 3; i++)
{
     System.out.print("*");
}
```

b)
```
for (int countDown = 5; countDown > 0; countDown--)
{
     System.out.println(countDown + " seconds");
}
```

c)
```
for (char letter = 'P'; letter <= 'S'; letter++)
{
     System.out.println("Give me a " + letter);
}
```

d)
```
for (int i = 2; i < 100; i *= i)
{
     System.out.println(i);
}
```

2. Write statements that will print a table of values for the expression $y=2x+5$ for the indicated values of x.

a) $x=6,5,4,\ldots,0$

b) $x=0,3,6,\ldots,30$

c) $x=-15,-10,-5,\ldots,15$

d) $x=1,2,4,8,\ldots,1024$

3. Write a fragment that uses a for statement to perform the indicated action.

a) Set the double variable sum to the value of
$$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{1000}$$

b) Set the double variable sum to the value of
$$\sqrt{100}+\sqrt{200}+\sqrt{300}+\cdots+\sqrt{5000}$$

c) Set the long variable product to the value of
$$1\times2\times3\times\cdots\times20$$

d) Set the int variable total to the value of
$$(-12)^3+(-11)^3+(-10)^3+\cdots+(20)^3$$

e) Set the double variable sum to the value of
$$1+\sqrt{2}+\sqrt[3]{3}+\sqrt[4]{4}+\cdots+\sqrt[25]{25}$$

4. Write a program that reads a positive integer n and then prints an "n-times table" containing values up to $n\times n$. For example, if the program reads the value 5, it should print
```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
```

Repetition using Java – Simple `for` Statements

Solutions

1. You can test these using the *interactions pane* in DrJava, or write a short program for each.

2. a) `for (int x = 6; x >= 0; x--)`

   b) `for (int x = 0; x <= 30; x+=3)`

   c) `for (int x = -15; x <= 15; x+=5)`

   d) `for (int x = 1; x <= 1024; x*=2)`

3. a)
```
double sum = 0;
for (int i = 1; i <= 1000; i++)
{
     sum += 1.0/i;
}
```

   b)
```
double sum = 0;
for (int i = 100; i <= 5000; i += 100)
{
     sum += Math.sqrt(i);
}
```

   c)
```
long product = 1;
for (int i = 1; i <= 20; i++)
{
     product *= i;
}
```

   d)
```
long total = 0;
for (int i = -12; i <= 20; i++)
{
     total += Math.pow(i,3);
}
```

   e)
```
double sum = 0;
for (int i = 1; i <= 25; i++)
{
     sum += Math.pow(i,(1.0/i));
}
```

4.
```
class ForLoopExercise4
{
  public static void main(String [] args)
  {
    int n;  // for the n-times table
    System.out.println("Which times table do you want?");
    n = In.getInt();
    for (int i = 1; i <= n; i++)
    {
      System.out.println(i + " x " + n + " = " + i*n);
    }
  }
}
```