COUNTING LOOPS AND ACCUMULATORS

Two very important looping idioms are counting loops and accumulators.

A counting loop uses a variable, called the loop control variable, to keep count of how many cycles it’s been through. To fashion such a loop:

1. Think up a short, easy name for your loop control variable.
2. Before the loop begins, declare the loop control variable and initialize it to zero.
3. During each loop cycle, increment the loop control variable.

Example
This code fragment illustrates a loop that cycles once for each value of \( k \) from 0, 1, 2, etc. \( k \) is the loop control variable.

```c
int k = 0;
while ( . . . )
{
    . . .
    k++;
}
```

A definite counting loop cycles a definite number of times. Such a loop must stop when the loop control variable reaches the desired number of cycles.

Examples
Each of these definite counting loops cycles \( n \) times.

```c
int k = 0;
while ( k < n )
{
    . . .
    k++;
}
```

```c
int k = 1;
while ( k <= n )
{
    . . .
    k++;
}
```

```c
for (int i=0; i<n; i++)
{
    . . .
}
```

```c
for (int j=1; j<=n; j++)
{
    . . .
}
```
An **indefinite counting loop** cycles an unknown number of times, stopping when some criterion is met that doesn’t involve the loop control variable. The purpose of such a loop is to count how many times the loop will cycle.

**Example**

How many times can we halve the value of \( n \) until 1 is reached? The following loop determines this by halving \( n \) each loop cycle and counting the number of such cycles. Since \( n \) is an integer, any fraction from dividing an odd number by 2 is truncated; this guarantees that \( n \) will eventually become exactly 1.

```java
1 int n, count;
2 . . .
3 count = 0;
4 while ( n > 1 )
5 {
6   n /= 2;
7   count++;
8 }
9 System.out.println( count + " time(s)" );
```

**Example**

How many digits are in the number \( n \)? We can solve this by counting the rightmost digit and removing it, continuing until all digits have been removed so that \( n \) is 0. A post-test loop is used because \( n = 0 \) has 1 digit.

```java
1 int n, count;
2 . . .
3 n = Math.abs( n ); // remove any - sign
4 count = 0;
5 do
6 {
7   count++; // count rightmost digit
8   n /= 10; // remove it
9 } while ( n > 0 );
10 System.out.println( count + " digit(s)" );
```
An *accumulator* is a variable that the program uses to calculate a sum or product of a series of values. A computer program does this by having a loop that adds or multiplies each successive value onto the accumulator. To visualize how this works try adding some numbers in your head, say 12, 7, 9 and 10. Most people do it by keeping track of the accumulated total, like this:

To fashion a loop with an accumulator:

1. Think up a variable name to use as your accumulator.
2. Before the loop begins, declare the accumulator and initialize it to zero.
3. During each loop cycle, add the next value you wish to sum onto the accumulator.

**Example**
This code fragment illustrates a loop that calculates a sum.

```java
double sum = 0.0;  // initialize the accumulator
while ( . . . )
{
    // obtain the next value
    . . .
    sum += value;  // add the value to the accumulator
}
```
Mathematical Series Example
For our first programming example, we’ll compute a mathematical series (google *math series*). This is a straightforward application of the definite counting loop and accumulator idioms. Here’s the program specification:

- **input** a positive integer \( n \)
- **calculate** \( 1+2+\ldots+n \)
- **output** the formula and the answer

For example, if \( n \) is 5, the output must read something like:

\[
1+2+3+4+5 = 15
\]

This problem requires a definite counting loop that counts from 1 to \( n \) by 1s. So, I start with a definite counting loop idiom using a Java `for` statement.

```
input n
for ( k=1; k<=n; k++ )
{
    . . .
}
```

This loop executes \( n \) times. On the first cycle, \( k=1 \); on the second, \( k=2 \); on the third, \( k=3 \); and so on. To calculate the sum, I introduce an accumulator (called `sum`) and add \( k \) onto it in each loop cycle.

```
input n
sum = 0;
for ( k=1; k<=n; k++ )
{
    sum += k;
    . . .
}
```
To the basic looping structure, I add the output operations and my algorithm is complete.

The Java program is given below.

```java
// Calculate 1 + 2 + ... + n
import java.util.Scanner;

public class Gauss1 {
    public static void main( String args[] ) {
        // build scanner, prompt user and get input
        Scanner scanner = new Scanner( System.in );
        System.out.print( "Enter n: " );
        int n = scanner.nextInt();
        // calculate sum
        int sum = 0; // start sum at 0
        for ( int k = 1; k <= n; k++ ) {
            sum += k; // add k onto sum
            System.out.print( "+" + k ); // nice output
        }
        // output final answer
        System.out.println( " = " + sum );
    }
}
```
**String Accumulator Example**

Our second example shows how to accumulate data within a string object by appending to it during each cycle of a loop.

The problem is the same as the previous example, except that the input and output must be done using dialog boxes.

This program uses, more or less, the same definite counting loop as the previous example.

But to display the formula within a dialog box after the loop has quit, we must append each term of the formula to the output string as the loop cycles.

To do this, create a `String` object and, before the loop, initialize it to the null string. On each loop cycle, append `k` onto the string.
The coded Java program is shown below.

```java
// Calculate 1 + 2 + ... + n
import static javax.swing.JOptionPane.*;

public class Gauss2
{
    public static void main( String args [] )
    {
        // input n
        String prompt = "Enter n: ";
        String input = showInputDialog( prompt );
        int n = Integer.parseInt( input );
        // calculate sum
        String output = ""; // start output str
        int sum = 0;        // start sum
        for ( int k = 1; k <= n; k++ )
        {
            sum += k; // add k onto sum
            output += "+" + k; // add k onto output
        }
        // output final answer
        output += " = " + sum;
        showMessageDialog( null, output );
    }
}
```
Programming Exercises

1. Write a Java application to print the children's song “There Were 10 in a Bed.” The first two verses are:

   There were 10 in the bed
   And the little one said
   "Roll over, roll over"
   So they all rolled over
   And one fell out

   There were 9 in the bed
   And the little one said
   "Roll over, roll over"
   So they all rolled over
   And one fell out

   The song continues through 8, 7, 6, 5, 4, 3 and 2. The last verse concludes:

   There was 1 in the bed
   And the little one said
   "Good Night!"

2. Write a Java application that reads a positive integer $n$ and uses a loop to calculate $1^2 + 2^2 + \ldots + n^2$. Print the result.

3. Write a Java application that reads a positive integer $n$, calculates the sum of the first $n$ odd integers and prints both the sum and the formula. For example:

   ![Input and Message Window]
4. Write a Java application that reads a positive integer $n$, calculates the product $1 \cdot 2 \cdot \ldots \cdot n$ and prints both the product and the formula. For example,

<table>
<thead>
<tr>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Image.md" alt="Image" /></td>
</tr>
</tbody>
</table>

6. Write a Java application that calculates and prints $s = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \ldots + \frac{1}{2^n}$.