Looping (Outsource: 5-1 – 5-2)

Making decisions is what makes programs seem smart; looping is what makes programs powerful. A **loop** is a structure that allows repeated execution of a block of statements. Within a looping structure, a boolean expression is evaluated. If it is true, then a block of statements, called the loop body, executes and then the boolean expression is evaluated again. As long as the expression is true, the statements in the loop body continue to execute. When the boolean evaluation is false, the loop ends. One execution of any loop is called an **iteration**.

The while Loop (Outsource: 5-4 – 5-8)

You can use a **while** loop to execute a body of statements continuously while some condition continues to be true. A while loop consists of the keyword while followed by a boolean expression within parentheses, followed by the body of the loop, which can be a single statement or a block of statements surrounded by curly brackets. There are three parts to every loop:

- Some variable, the **l**oop **c**ontrol **v**ariable (**LCV**), is initialized.
- The *loop control variable* is tested in the while statement.
- The body of the while statement must take some action that alters the value of the loop control variable.

For example, the following code causes *Hello World* to show up on the screen ten times:

```
Starting      Terminating
 Value           Value

int lcv = 0;                              // initialize the Loop Control Variable
while ( lcv < 10 )                        // test the LCV
{
    System.out.println("Hello World");
    lcv++;                                // alter the LCV
}
```

An **infinite loop** is a loop that never ends. The two most common causes of infinite loops are failing to alter the LCV and putting a semicolon at the end of the boolean expression. For example, the following program is an infinite loop that does nothing:

```
int lcv = 0;
while (lcv < 10 );       // Don't Do This
{
    System.out.println("Hello World");
    lcv ++;
}
```

The semicolon after the boolean expression **while (lcv < 10);** causes that statement to execute endlessly. This loop has an **empty body**, or a body with no statements in it. Remember, a semicolon means end of statement!

Shortcut Arithmetic Operators

It is very common to alter the value of a loop control variable by adding one to it, or *incrementing* the variable. Java provides several methods for incrementing and decrementing. For example the statement **x = x + 1;** adds one to the contents of the variable **x**. If **x** started out with the value of *0* it would now contain *1*. In Java, there are two shortcut methods of incrementing: prefix incrementing **++x** and postfix incrementing **x++.** The difference between the two is that postfix increments AFTER the statement that contains it is executed while prefix increments BEFORE the statement that contains the incrimination is executed. For example, the following program using postfix incrementing displays *0* on the console screen:

```
int x = 0;
System.out.println(x++);     // post-incrementation
```

The next program sample, using prefix incrementing, displays 1 on the console screen:

```
int x = 0;
System.out.println(++x);     // pre-incrementation
```

The prefix and postfix increment operators are **unary** operators because you use them with one value. Most arithmetic operators, like those used for addition and multiplication, are **binary** operators that operate on two values.

Besides using the prefix and postfix increment operators, you can use a prefix or postfix decrement operator (**--**) that reduces a variable's value by one. For example:

```
int x = 0;
System.out.println(x--);      // post-decrementation
```

Lab 08 - ASSIGNMENT

Lab 08A - 60 points

OBJECTIVE

WAP to print the integers from 1 through 20. The only method you will need for this program is the **main()** method.

HINT: Write a while statement that counts from 1 to 20. Print the loop control variable.

OUTPUT

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Lab 08B - 70 points

OBJECTIVE

WAP to print the following design using only one output statement that prints only one asterisk. I.E. **System.out.print(“*”) ;**

HINT: Write a while statement that counts from 1 to **n** where **n** is the value entered from the keyboard. Print a single asterisk in the loop.

FIELD SUMMARY

- **int stars** – the number of stars to be displayed.

METHOD SUMMARY

- **main** – instantiate an instance of this class. Make method calls to `input` and `output`.
- **input** – declare a `Scanner` object and read the number of stars to be displayed from the keyboard using an appropriate prompt.
- **output** – use a while loop to display the specified number of stars.

SAMPLE KEYBOARD INPUT

Enter the number of stars to display: **6**

SAMPLE OUTPUT

6 stars: ****

Lab 08C - 80 points

OBJECTIVE

WAP to output all the odd numbers between two values read from the keyboard in ascending order. Include appropriate prompts for the two input integers. Your program should be able to input the two integers in any order, i.e., smaller first or larger first.

HINT: Assign the *smallest* of the two numbers to the loop control variable. Math.min works great for that purpose. Declare another variable to use as the terminating value and assign it the *largest* value (Math.max). Use an if statement to print only the odd numbers. Odd numbers are numbers that cannot be evenly divided by 2.

FIELD SUMMARY

- **int a** – the first number.
- **int b** – the second number.

METHOD SUMMARY

- **main** – instantiate an instance of this class. Make method calls to `input` and `output`.
- **input** – declare a `Scanner` object and read two integers from the keyboard using appropriate prompts.
- **output** – use a while loop to display all the numbers from the smaller number to the larger number.

SAMPLE KEYBOARD INPUT

Enter the first number: **30**
Enter the second number: **4**

SAMPLE OUTPUT

```
All the odd numbers between 4 and 30:
5 7 9 11 13 15 17 19 21 23 25 27 29
```

Lab 08D - 90 points

OBJECTIVE

WAP to input an integer from the keyboard and output a multiplication table from one to ten.

HINT: The while statement should count from 1 to 10. Multiply the loop control variable times the value entered from the keyboard. Only one output statement is required.

SAMPLE KEYBOARD INPUT

```
Enter an integer: 8
```

SAMPLE OUTPUT

```
Multiplication table for 8's:
1 * 8 = 8
2 * 8 = 16
3 * 8 = 24
4 * 8 = 32
5 * 8 = 40
6 * 8 = 48
7 * 8 = 56
8 * 8 = 64
9 * 8 = 72
10 * 8 = 80
```

Lab 08E - 100 points

OBJECTIVE

WAP to input a name and a number from the keyboard. Output the name the specified number of times. The first time the name is displayed it should be flush against the left margin. Each succeeding name should be one space to the right of the name above it. This is simply a trick manipulating Strings inside a loop.

HINT: The while statement should count from 1 to **n** where **n** is the value entered from the keyboard. Use String concatenation to add a blank space to the front of the name each time the loop executes.

SAMPLE KEYBOARD INPUT

What is your name? **John**
How many times do you want your name to be displayed? **4**

SAMPLE OUTPUT

```
John displayed 4 times:
John
 John
  John
   John
```