Iteration

Iteration, repetition, or looping, they all mean the same thing. When you want to repeat some portion of the program, you will use some form of iteration. Looping is a very common term to describe this.

The **while** Statement

The most basic looping construct is the **`while`** statement. The form is identical to the `if` statement, but uses the keyword while instead:

```
while( expression )
{
  statement(s)
}
```

Whereas the **`if`** statement caused *statement* to be executed exactly once if *expression* was true, the **`while`** statement causes *statement* to be executed repeatedly as long as *expression* remains true. If *expression* becomes false, then the repetition stops.

```
class Main
{
        public void main(String[] args)
        {
                int count = 5;
                int i = 0;

                while (i < count)  /* controlling expression  */
                {
                   i++;           /* body of the loop       */
                }
        }
}
```

Example with output: (and a review of expressions)

Version #1:

```
class Main
{
   public void main(String[] args)
   {
      int count = 5;
      int i = 0;

      while (i < count)
      {
         i++;
         System.out.println("i   is "  + i);
      }
   }
}
```

Version #2:

```
class Main
{
   public void main(String[] args)
   {
      int count = 5;
      int i = 0;

      while (i < count)
      {
         System.out.println("i   is "  + ++i);
      }
   }
}
```

Version #3:

```
class Main
{
   public void main(String[] args)
   {
      int count = 5;
      int i = 0;

      while (i ++< count)
      {
         System.out.println("i   is "  + i);
      }
   }
}
```

Output:
(for all 3 examples)

```
i is 1
i is 2
i is 3
i is 4
i is 5
```

Notes:

- The body of the loop is not guaranteed to execute at all.
- The controlling expression:
  - is executed once before the body of the loop is executed.
  - is executed again after each repetition of the body.
  - usually will eventually evaluate to false, to stop the repetition.
- If the controlling expression *never* evaluates to false, the while loop is called *an infinite loop*, because it never stops.

Infinite loop #1:

```
class Main
{
   public void main(String[] args)
   {
      int i = 1;
      while (i != 10)
      {
         i += 2;
      }
   }
}
```

Infinite loop #2:

```
class Main
{
   public void main(String[] args)
   {
      int i = 0;
      while (i < 10)
      {
            System.out.println("i is " + i);
      }
   }
}
```

Infinite loop #3:

```
class Main
{
   public void main(String[] args)
   {
      while (true)
      {
            System.out.println("This loop never ends...");
      }
   }
}
```

The `do` Statement

Also sometimes referred to as the **`do...while`** statement. The basic format is:

```
do
{
  statement(s)
}
while( expression );
```

The primary difference between the **`while`** statement and the **`do`** statement is that the body of the **`do`** statement is guaranteed to execute at least once. This is simply because the controlling expression is executed *after* the first iteration of the loop body:

Body executes 0 or more times	Body executes 1 or more times

`while ( expression )`
`{`
`  Statement(s)`
`}`
	`do`
	`{`
	`  Statement(s)`
	`}`
	`while ( expression );`

Example:

```
class Main
{
  public void main(String[] args)
  {
    int number;
    boolean choice = false;

    Scanner InputScanner = new Scanner(System.in);

    do
    {
      System.out.print("Enter a number: ");

      /* Get the users number */
      number = InputScanner.nextInt();
      System.out.println("You entered " + number);

      System.out.print("Enter another number? (true=yes,false=no) ");
      choice = InputScanner.nextBoolean();
    }
    while (choice);
  }
}
```

Sample Run:

```
Enter a number: 12
You entered 12
Enter another number? (true=yes,false=no) true
Enter a number: 12
You entered 12
Enter another number? (true=yes,false=no) true
Enter a number: 42
You entered 42
Enter another number? (true=yes,false=no) false
```

There really isn't much difference between **`while`** statement and the **`do`** statement. If you need the loop to execute at least once, then the **`do`** statement is the one to use.

The `for` Statement

Now we get to the most complex of the looping mechanisms: the **`for`** statement. The general form is:

```
for ( expression1 ; expression2 ; expression3 )
{
  statement(s)
}
```

The meaning of this is a little involved:

- First, evaluate *expression*$_1$. This is executed and evaluated exactly once at the beginning of the loop.
- Evaluate *expression*$_2$.
- If *expression*$_2$ is true, execute *statement(s)*. (If it's not true, jump out of the loop.)
- Evaluate *expression*$_3$.
- Goto step 2.

This process can be written using an equivalent **`while`** statement:

```
expression1;
while ( expression2 )
{
  statement(s)
  expression3;
}
```

This also means that any **`for`** loop can be written as a **`while`** loop and vice-versa.

Simple examples to print the numbers 1 through 10:

for loop #1:

```
class Main
{
   public void main(String[] args)
   {
      for(int i = 1; i <= 10; i++)
      {
            System.out.println(i);
      }
   }
}
```

for loop #2:

```
class Main
{
   public void main(String[] args)
   {
      for(int i = 0; i < 10; i++)
      {
            System.out.println(i + 1);
      }
   }
}
```

while loop:

```
class Main
{
   public void main(String[] args)
   {
      int i = 1;
      while (i <= 10)
      {
         System.out.println( i++);
      }
   }
}
```

The `for` loops above show the typical ways in which they are used. The variable *i* is sometimes called the *loop control variable* (or simply the *counter*) because it controls when the loop continues or stops. The three expressions generally

1. *expression*$_1$ - initializes the loop variable (or counter)
2. *expression*$_2$ - compares the counter with some value
3. *expression*$_3$ - modifies the counter (usually add/subtract 1)

These are just typical uses of the expressions. You can do practically anything with those expressions.

<table>
<tr>
<td><b>Count to 20 by 2</b></td>
<td><pre>
class Main
{
   public void main(String[] args)
   {
      for(int i = 2; i <= 20; i+=2)
      {
         System.out.println(i);
      }
   }
}
</pre></td>
<td><pre>
class Main
{
   public void main(String[] args)
   {
      int i = 2;
      while(i <= 20)
      {
         System.out.println(i);
         i+=2;
      }
   }
}
</pre></td>
</tr>
<tr>
<td><b><u>Output:</u></b></td>
<td colspan="2"><pre>
2
4
6
8
10
12
14
16
18
20
</pre></td>
</tr>
</table>

<table>
<tr>
<td><b>Count down from 30 by 12</b></td>
<td><pre>
class Main
{
   public void main(String[] args)
   {
      for(int i = 30; i >= 12; i -= 3)
      {
         System.out.println(i);
      }
   }
}
</pre></td>
<td><pre>
class Main
{
   public void main(String[] args)
   {
      int i = 30;
      while(i >= 12)
      {
         System.out.println(i);
         i -= 3;
      }
   }
}
</pre></td>
</tr>
<tr>
<td><b>Output:</b></td>
<td colspan="2"><pre>
30
27
24
21
18
15
12
</pre></td>
</tr>
</table>

Squares of 1 to 10	class Main
	{
	   public void main(String[] args)
	   {
	      for(int i = 1; i <= 10; i++)
	      {
	         System.out.println(i * i);
	      }
	   }
	}
		class Main
		{
		   public void main(String[] args)
		   {
		      int i = 1;
		      while(i <= 10)
		      {
		         System.out.println(i * i);
		         i++;
		      }
		   }
		}

Output:	**1** **4** **9** **16** **25** **36** **49** **64** **81** **100**	

More On The Looping

Note that any or all of the expressions in the **`for`** loop can be omitted:

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

Of course, this is nothing but a strange-looking while loop now.

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

You can even omit the second expression, but this would lead to an infinite loop, since the default *empty* expression is true!

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

If you want to exit from the loop prematurely, you can use the **`break`** statement:

breaking out of infinite `for`	**breaking out of infinite `while`**

i = 1;
for (; ;)
{
  System.out.println(i++);
  if (i > 10)
  {
    break;
  }
}
	i = 1;
	while (true)
	{
	  System.out.println(i++);
	  if (i > 10)
	  {
	    break;
	  }
	}

The ***break*** statement can be used in any of the looping mechanisms as well as the switch statement.

You can also have multiple expressions in between the semicolons:

Using a `for` loop

```
for (int i = 0, j = 0; i < 16 || j < 3; i +=2, j++)
{
   System.out.format("%d * %d = %d%n", i, j, i * j);
}
```

Using a `while` loop

```
i = 0;
j = 0;
while (i < 16 || j < 3)
{
   System.out.format("%d * %d = %d%n",  i, j, i * j);
   i += 2;
   j++;
}
```

Output:

```
0 * 0 = 0
2 * 1 = 2
4 * 2 = 8
6 * 3 = 18
8 * 4 = 32
10 * 5 = 50
12 * 6 = 72
14 * 7 = 98
```

The **`continue`** statement is similar to the **`break`** statement in that it causes the loop to deviate from its prescribed course. The difference is subtle, but very important.

`break` statement

```
for (/* expressions */)
{
  /* first statement in loop   */
  /* second statement in loop */
  /* etc...                    */

  break;

  /* last statement in loop    */
}
[break jumps to here]
/* first statement after loop */
```

`continue` statement

```
for (/* expressions */)
{
  /* first statement in loop   */
  /* second statement in loop */
  /* etc...                    */

  continue;

  /* last statement in loop    */
  [continue jumps to here]
}
/* first statement after loop */
```

This prints the even numbers from 2 to 20:

using `for`	using `while`	using `while`

for (i = 2; i <= 20; i++)
{
   if ((i % 2) == 1)
   {
     continue;
   }
   System.out.println(i);
}
	i = 2;
	while (i <= 20)
	{
	   if ((i % 2) == 1)
	   {
	     i++;
	     continue;
	   }
	   System.out.println(i++);
	}
		i = 2;
		while (i <= 20)
		{
		   if ((i++ % 2) == 1)
		   {
		     continue;
		   }
		   System.out.println(i - 1);
		}

Output:	**2** **4** **6** **8** **10** **12** **14** **16** **18** **20**	