Iteration Statements

while

The **while** loop repeats a statement or block while its controlling expression is true. Here is the general form:

```
while(condition) {
    // body of loop
}
```

The *condition* can be any Boolean expression. The body of the loop will be executed as long as the conditional expression is true. When*condition* becomes false, control passes to the next line of code immediately following the loop.

Since the **while** loop evaluates its conditional expression at the top of the loop, the body of the loop will not execute even once if the condition is false to begin with.

do-while

The **do-while** loop always executes its body at least once, because its conditional expression is at the bottom of the loop. Here is the general form:

```
do {
  // body of loop
} while
```

Each iteration of the **do-while** loop executes the body of the loop and then evaluates the conditional expression. If this expression is true, the loop will repeat. Otherwise, the loop terminates. The *condition* must a Boolean expression.

for

There are two forms of the **for** loop. The traditional form and the **for-each** form. Here is the general form of the traditional **for** statement:

```
for (initialization; condition; iteration) {
    //body of loop
}
```

When the loop first starts, the *initialization* portion of the loop is executed. Generally this is an expression that sets the value of the *loop control variable*, which acts as a counter that controls the loop.

Next, *condition* is evaluated . This must be a Boolean expression. It usually test the loop control variable against a target value. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.

Next, the *iteration* portion of the loop is executed. This is usually an expression that increments or decrements the loop control variable. The loop iterates, first evaluating the conditional expression, then executing the body of the loop, and then executing the iteration expression each pass. This process repeates until the controlling expression is false.

You can declare a variable inside a **for** loop, there is one important thing to remember: the scope of that variable ends when the **for** statement does.

Java allows you to include multiple statements in both the initialization and iteration portions of the **for**. Each statement is separated from the next by a comma.

```
for(int a = 1, int b = 4; a < b; a++,b--)
```

for-each

A **for-each** style loop is designed to cycle through a collection of objects, such as an array, in strictly sequential way, from start to finish. The general form of the **for-each** version of the **for**

```
for (type iterator-variable :  collection) statement-block
```

Here, *type* specifies the type and *iterator-variable* specifies the name of an *iteration variable* that will receive the elements from a collection, one at a time, from beginning to end. The collection being cycled through is specified by *collection*. With each iteration of the loop, the next element in the collection have been obtained.

Because the iteration variable receives values from the collection, *type* must be the same (or compatible with) the elements stored in the collection. Thus, when iterating over arrays, *type* must be compatible with the elementtype of the array.

There is one important point to understand about the **for-each** style loop. Its iteration variable is "read-only" as it relates to the underlying array. An assignment to the iteration variable has no eefect on the underlying array. In other words, you can't change the contents of the array by assigning the iteration variable a new value.