INITIALIZATION IN DECLARATIONS

When a variable declaration is executed, memory is allocated for the variable. This memory must be initialized to contain some definite value before the variable can be used in an expression. In the case of a local variable, the declaration is often followed closely by an assignment statement that does the initialization. For example,

```
int count;    // Declare a variable named count.
count = 0;    // Give count its initial value.
```

However, the truth about declaration statements is that it is legal to include the initialization of the variable in the declaration statement. The two statements above can therefore be abbreviated as

```
int count = 0;  // Declare count and give it an initial value.
```

The computer still executes this statement in two steps: Declare the variable `count`, then assign the value 0 to the newly created variable. The initial value does not have to be a constant. It can be any expression. It is legal to initialize several variables in one declaration statement. For example,

```
char firstInitial = 'D', secondInitial = 'E';

int x, y = 1;   // OK, but only y has been initialized!

int N = 3, M = N+2;  // OK, N is initialized
                     //        before its value is used.
```

This feature is especially common in `for` loops, since it makes it possible to declare a loop control variable at the same point in the loop where it is initialized. Since the loop control variable generally has nothing to do with the rest of the program outside

the loop, it's reasonable to have its declaration in the part of the program where it's actually used. For example:

```
for ( int i = 0;  i < 10;  i++ ) {
   System.out.println(i);
}
```

Again, you should remember that this is simply an abbreviation for the following, where I've added an extra pair of braces to show that `i` is considered to be local to the `for` statement and no longer exists after the `for` loop ends:

```
{
   int i;
   for ( i = 0;  i < 10;  i++ ) {
      System.out.println(i);
   }
}
```

(You might recall, by the way, that for "for-each" loops, the special type of `for` statement that is used with enumerated types, declaring the variable in the `for` is **required**. See Subsection 3.4.4.)

A member variable can also be initialized at the point where it is declared, just as for a local variable. For example:

```
public class Bank {
   static double interestRate = 0.05;
   static int maxWithdrawal = 200;
     .
     .  // More variables and subroutines.
     .
}
```

A static member variable is created as soon as the class is loaded by the Java interpreter, and the initialization is also done at that time. In the case of member

variables, this is not simply an abbreviation for a declaration followed by an assignment statement. Declaration statements are the only type of statement that can occur outside of a subroutine. Assignment statements cannot, so the following is illegal:

```
public class Bank {
   static double interestRate;
   interestRate = 0.05;  // ILLEGAL:
   .                     //    Can't be outside a subroutine!:
   .
   .
```

Because of this, declarations of member variables often include initial values. In fact, as mentioned in Subsection 4.2.4, if no initial value is provided for a member variable, then a default initial value is used. For example, when declaring an integer member variable, `count`, "`static int count;`" is equivalent to "`static int count = 0;`".

Source : http://math.hws.edu/javanotes/c4/s7.html