Local variables

A *local variable* is a variable that is declared within a method body. The program you see has two different local variables, both named `temp`.

The syntax of a local variable declaration is:

> *<type> <variable-name>* `;`

and it can be an initializing declaration:

> *<type> <variable-name>*`=` *<expression>*`;`

The initializing declaration allows us to write a sequence of statements to swap two variables as follows:

```
int temp= x;  x= z;  z=  temp;
```

We limit our remarks on local variables to the three issues that you see on your monitor:

1. When a local variable is created and destroyed.
2. The scope of a local variable.
3. Naming a local variable.

When a local variable is created and destroyed

Here is a call on method `m`, with three arguments. In executing this call, all local variables are created before execution of the method body. Note that there are two declarations for variable `temp`, They are different variables, and that is why you see two variables with that name. Second, the local variables remain in existence as long as the method body is being executed. Third, destroyed when execution of the method body terminates.

Knowing when local variables are created —before execution of the method body begins— will help you answer questions about local variables that will arise later on. So please memorize when local variables are created.

The scope of a local variable

The scope of a local variable is the place in which it can be referenced. This includes all statements in the block in which it is declared, starting with the statement following its declaration and going to the end of the block.

For example, in this function, the scope of `s` begins at the declaration of `k` and ends with the return statement. And the scope of `k` begins with the for-loop and ends with the return statement. This means that one cannot refer to `k` in a statement before its declaration.

Note that the scope of a variable declared in the initialization of a for-loop consists of the rest of the for-loop, but not the statement following the for-loop. In the example shown here, `k` is declared within the initialization of the loop, so it cannot be used in following the loop.

Guidelines for naming a local variable

Now let's talk about local-variable names and parameter names. Generally, methods in OO programming are usually short. In fact, they generally take less than a page, and almost always less than two pages.

The brevity of a method means that parameter declarations as well as the specification of the method, which describes what each parameter is for, can always be seen when reading the method body. Therefore, there is no need for long parameter names.

One will sometimes see a method like this with long parameter names —the programmer is trying to put the meaning of the parameters into their names. But instead of clarifying and simplifying, this change has complicated. How much harder the program is to read! Doesn't it look awkward? And there is no need for this complication; the short names, with no meaning, are OK because the specification and the other comments gives all the meaning, and they are readily seen because the method is short.

In the same way, if local variables are sufficiently described in comments, and if the method body is short, short names are to be preferred for local variables.