Local variable

A *local variable* is a variable that is declared within the body of a method.

The method given to the right uses a local variable `t` of type **`int`**. This is not the best method to return the larger of two values, but it illustrates that the declaration of a local variable can be placed essentially anywhere where a statement can be placed.

```
/** Return the max of x and y. */
public static int max(int x, int y) {
    // ensure that x is the max
    if (x < y) {
        int t= x;
        x= y;
        y= t;
    }
}
```

The declaration of a local variable has the form:

<type> <variable-name>;

but it can be combined with an assignment statement:

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

Local variables are not initialized

You know that fields that are not explicitly assigned a value are given a default value, depending on their type. For example, suppose field `f` is declared as follows:

```
public int f;
```

If the constructor does not store a value in `f` when an object is created, `f` will contain 0.

This is not the case for local variables: they are uninitialized. Silly method `zero` to the right won't even compile! It has a syntax error because it can be determined that the reference to `m` in the return statement is illegal because `m` does not have a value. Go ahead —copy method `zero` into a Java class somewhere and see what happens!

```
/** Return 0. */
public int zero() {
    int m;
    return m;
}
```

Scope of a local variable

The scope of a local variable —the places it can be used— consists of its declaration to the end of the block in which it is declared. In the example of method `max` above, the scope of `t` is the whole block

```
{ int t= x; x= y; y= t; }
```

Suppose the if-statement in method `max` was written as in the box to the right. The reference to `t` in the declaration-assignment to `b` is syntactically wrong because that reference to `t` is not within the scope of local variable `t`, so the program would not compile.

```
if (x < y) {
    int b= t;
    int t= x;
    x= y;
    y= t;
}
```

When space for a local variable is allocated and deallocated

As you will learn when you study how a method call is executed, space for all parameters and local variables is created *before* the method body is executed and space is deallocated *after* execution of the method body ends.

The method to the right has parameter `b`, which is (a pointer to) an array, and two local variables. Space for `b`, `k`, and `t` is allocated *before* the method body is executed. In particular, space for local variable `t` is created *once* only, even though it is declared in the body of the loop.

```
/** Return true iff b contains a 0 */
public boolean z(int[] b) {
    for (int k= 0; k < b.length; k++) {
        int t= b[k];
        if (t == 0) return true;
    }
    return false;
}
```

Principle: declare a local variable as close to its first use as possible

Beginning programmers have the tendency to place local variable declarations at the top of the method body, as shown to the right. *Fight this tendency*! Logically, it makes no sense to declare `t` there. Why should the reader have to know about it at that place? Logically, `t` should be declared within the then-part of the if-statement, as done at the top of this page.

```
/** Return the max of x and y. */
public static int max(int x, int y) {
    int t;
    // ensure that x is the max
    if (x < y) {
        t= x;
        x= y;
        y= t;
    }
```

Follow this principle: *Declare a local variable as close to its first use as possible*!