Declaring local variables where they belong, logically speaking

We discuss the placement of local variable declarations. Generally, the declarations should go where they belong, logically speaking, and this usually means placing them as close to their first use as possible. Let's look at an example that illustrates this.

Here is a procedure to sort an array segment `b[0..n]` using an algorithm called *selection sort*. We have not discussed arrays in this course, but you have seen them in another programming language. In the first parameter declaration, you see how to declare `b` to be a one-dimensional array.

Just after the declaration of `temp and` j is a comment that states the invariant of the main loop —this is simply a picture that holds before and after each iteration of the loop, and we display the information as a picture down below. The picture says that the segment `b[0..k-1]` is sorted and that all of its elements are at most those in the second segment, that's what we mean by the signs <= and >=.

Take a look at the repetend of the outer loop. It sets `j` to the index of the minimum value in `b[k..n]` and then swaps `b[j]` and `b[k]`, so that `b[k]` contains the smallest value in the second segment `b[k..n]`. Since `b[k]` is the smallest value of that segment, and it is at least all the elements in the first segment, incrementing `k` in this segment keeps the picture true.

Now, why are `temp` and j declared at the beginning? What is their meaning? When you first read the procedure body, aren't you confused? In fact, these variables have no meaning at this point, and their appearance at the beginning of the body is logically incorrect. Their appearance at the beginning gives the impression that they have meaning from iteration to iteration, which they don't. The declarations don't belong at the beginning.

Consider variable `temp`. It is used only in swapping `b[j]` and `b[k]`, so `temp` can be declared in the first assignment.

Consider variable `j`. Where is it used? Only in the repetend of the outer loop: it will contain the index of the minimum of `b[k..n],` and then it will participate in a swap. So we can declare `j` at its first assignment.

Note that `temp` is declared within the *statement-comment* "Swap `b[j]` and `b[k]`", because that is where it is used, that is where it belongs. And, the declaration is as close to its first use as possible.

Similarly, `j` is declared where it belongs; it is a local variable of the repetend of the outer loop, and it is as close to its first use as possible.

Are you worried about making the procedure less efficient because the declaration is "processed" many times? Don't be worried. Remember, a local variable is created before execution of the method body, just once. So we have not made the procedure less efficient. Moreover, the procedure is easier to understand.