5. Programming – Variable Scope

***Note**: some example programs are from, or based on, examples from Java for Everyone (C Horstmann), the course text.*

We have seen a number of examples of programs – and parts of programs – where we first declare some variables and then do something to them. Some programming languages insist you do that, but Java is a bit more flexible.

- You must declare variables in Java before you use them.
- But you don't have to declare them all at the start of your code.

For example, let's look at some code that prints out the average of a group of numbers – assume that we have already calculated the total. We want to avoid dividing by zero and might write:

```
int average = 0;//Give it a value or compiler complains
if (count == 0) {
     System.out.println("Zero data items");
} else {
     average = total/count;
     System.out.println("Average: " + average);
}
```

But we only use the variable average inside the else part – so why not declare it there?

```
if (count == 0) {
     System.out.println("Zero data items");
} else {
     int average = total/count;
     System.out.println("Average: " + average);
}
```

Notice it's shorter and a bit more readable – we don't have to look back up the code to find the declaration (for example to see if it's an `int` or a `double`).

KEY POINT: Declare Variables Where You Use Them

It's good practice to declare variables within the block (i.e. in the {…}) that you use them.

One thing to worry about though is if we want to use the value stored in average later on. Suppose we try to do this:

```
if (count == 0) {
     System.out.println("Zero data items");
} else {
     int average = total/count;
     System.out.println("Average: " + average);
}

int saveTheAverage = average;
```

If we compile this we get:

```
Average.java:13: error: cannot find symbol
     int saveTheAverage = average;
                          ^
  symbol:   variable average
```

Java is unable to find the variable average – and the reason is that it is only visible within the block it's declared in – in this case, as shown below in red:

```
if (count == 0) {
     System.out.println("Zero data items");
} else {
     int average = total/count;
     System.out.println("Average: " + average);
}

int saveTheAverage = average;
```

In this particular case, if we *really* want to use the variable `average` outside the else block, we *have* to declare it outside.

The Scope of a Variable

We use the term *scope* to describe the 'visibility' of a variable – in general, *the scope of a variable is the block it is declared in and any blocks within that.* For example, taking the case above and extending it:

```
if (count == 0) {
     System.out.println("Zero data items");
} else {
     int average = total/count;
     System.out.println("Average: " + average);
     if (average > 10) {
          System.out.println("An average length of "+
               average + " is big for lemons.");
     }
}
```

There are three different scope blocks here – green, red and blue. The variable `average` is declared in the red block and so is not visible to the green block. The blue block is inside the red block, so we can still access and use `average` within it.

Style Issue – One Variable or Two? (or More…)

One case that comes up fairly often is a case like this:

```
double tax = 0.0;
if (salary > 40000) {
     tax = salary * 0.4;
     System.out.println("Your tax bill is: " + tax);
} else {
     tax = salary * 0.2;
     System.out.println("Your tax bill is: " + tax);
}
```

At first sight it looks like we have to declare tax outside the blocks inside the if-else – *but we use them to store separate and unrelated values*. So we can actually do this:

```
if (salary > 40000) {
     double tax = salary * 0.4;
     System.out.println("Your tax bill is: " + tax);
} else {
     double tax = salary * 0.2;
     System.out.println("Your tax bill is: " + tax);
}
```

As well as being a bit shorter, this moves the variable declarations closer to the code that uses them, which is good.

More than One Variable – Same Name!

Notice that in the example above *we have two variables with the same name*. **This is completely fine – *provided they are in unrelated blocks.*** In this case they are.

KEY POINT: Think Carefully Before Using the Same Name for Variables

Just because you can do something does not mean you should – so you need to think if using the same name is going to be confusing. I don't think it is in this case, because they both refer to the same thing (income tax). I would be cautious about using the same name if they did not refer to the same thing (even if it was only different *kinds* of tax). You need to use your judgment and decide if the choices you make improve your code and it's readability.

Same Name, Nested Blocks – Not Allowed

One thing you cannot do in Java though is this:

```
if (salary > 40000) {
     double tax = salary * 0.4;
     System.out.println("Your tax bill is: " + tax);
     if (salary > 100000) {
          double tax = salary * 0.5;
          System.out.println("You're rich – extra tax: "
               + tax);
     }
}
```

You'll get a message something like this:

```
Tax.java:9: error: variable tax is already defined in
method main(String[])
               double tax = salary * 0.5;
```

Some languages *do* permit this – the inner variable 'hides' the outer one while the inner block is being executed. But the designers of Java thought it was too confusing and did not allow it (I agree).