Local Variables and Scope

In the past couple of videos, we've looked at many of Java's flow statements, the **switch** statement, the **for** statement, the **while** statement, as well as the **do while** statement.

In previous videos, we covered the **if then else** statement.

All of these statements may, and probably will, have their own code blocks.

We've talked about code blocks quite a bit, but we haven't really talked about variables declared locally to many of these code blocks.

Local Variable

A local variable is called local, because it is available for use by the code block in which it was declared.

It is also available to code blocks that are contained by a declaring block.

```
{    // Starts on outer block - for example a method block
     int firstVariable = 5;
     int secondVariable = 10;

     if (firstVariable > 0) {    // flow statement block starts inner block

          // Inner block code has access to outer block's variables
          System.out.println(secondVariable);
     }
}
```

Scope

Scope describes the accessibility of a variable.

'In scope' means the variable can be used by an executing block or any nested blocks.

'Out of scope' means the variable is no longer available.

When are Local Variables in Scope?

Local variables are always in scope, in the block they are declared.

They are also in scope for any nested blocks, or blocks contained within the outer block.

When are Local Variables out of Scope?

Local variables are always out of scope, for outer blocks, or the containing blocks they are declared in.

Let's look at an example:

```
public static void aMethod(boolean aBoolean) {

    if (aBoolean) {
        int myCounter = 10;          // myCounter is local to this if block
    }
    System.out.println(myCounter);  //myCounter is out of scope here
}
```

Cannot resolve symbol 'myCounter'

Bring 'int myCounter' into scope Alt+Shift+Enter More actions... Alt+Enter

Scope Best Practices

It is considered best practice:

To declare and initialize variables in the same place if possible.

And to declare variables in the narrowest scope possible.

Local Variables and the For Statement

In this **for** statement, as part of the declaration, there is an initialization part, as we've described.

In this case, we declared a variable, i, that isn't accessible outside of the loop.

This is because any variables declared in the init section, are local to the loop, meaning they exist and are accessible in memory, only while the loop is executing, and only to the loop code block.

```
{   // Starts on outer block - for example a method block

    for (int i = 1; i <= 5; i++) {  // i declared in for loop declaration
        System.out.println(i);
    }

    System.out.println(i); // ERROR! i is out of scope
}
```

Declaring variables in a switch statement block

Local Variables declared in an **if** statement block, are not accessible outside of that block.

This also includes other parts of the **if** statement, like the **else if** block, or the **else** block.

```
public static void aMethod(int counter) {

    if (counter > 0 ) {
        int i = 10;
    }
    else {
        System.out.println(i);  // ERROR:  i is out of scope
    }

    System.out.println(i);   // ERROR:  i is out of scope
}
```

Declaring variables in a switch statement block

However, the **switch** statement is different from the if then **else** statement blocks.

Consider the code below:

```
public static void aMethod(int value) {

    switch (value) {

        case 1:
            int i = 10;
            break;

        default:
            i = value;                // this is ok
            System.out.println(i);    // this is ok
            break;
    }

    System.out.println(i);  //  ERROR:  i is out of scope outside of the switch
}
```

Declaring variables in a switch statement block

Let's look at another example, of code blocks and local variables, in the switch statement.

```
public static void aMethod(int value) {

    switch (value) {

        case 1:
            System.out.println(i);  // this is NOT ok, i declared below
            break;

        case 2:
            int i = 10;
            System.out.println(i);  // this is ok
            break;

        default:
            i = value;    // this is ok
            System.out.println(i);  // this is ok
            break;

    }

    System.out.println(i);  //  ERROR:  i is out of scope outside of the switch
}
```