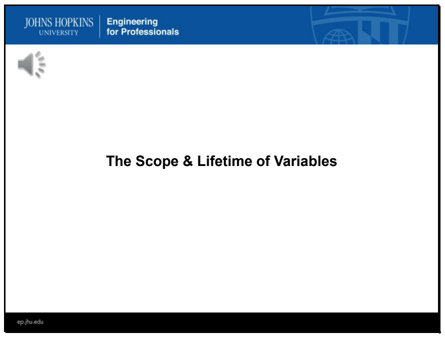
1

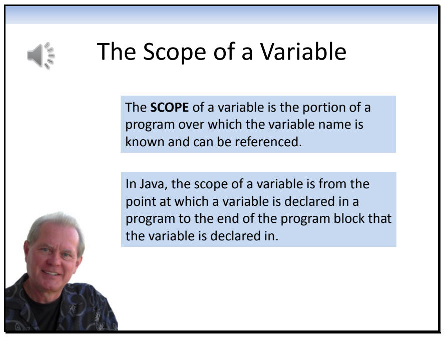

JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

The Scope & Lifetime of Variables

ep.jhu.edu

In this lecture you will learn two very important programming concepts…the scope and the lifetime of Java variables.

2


The Scope of a Variable

The **SCOPE** of a variable is the portion of a program over which the variable name is known and can be referenced.

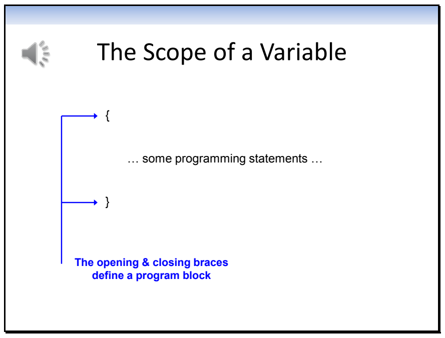
In Java, the scope of a variable is from the point at which a variable is declared in a program to the end of the program block that the variable is declared in.

The SCOPE of a variable is the portion of a program over which the variable name is known and can be referenced. When I say ‘can be referenced’ I mean using the variable name in an expression or programming statement.

In Java, the scope of a local variable is from the point at which the variable is declared in a program to the end of the program block that the variable is declared in.

Let’s look at some examples to help understand the concept of scope.

3


The Scope of a Variable

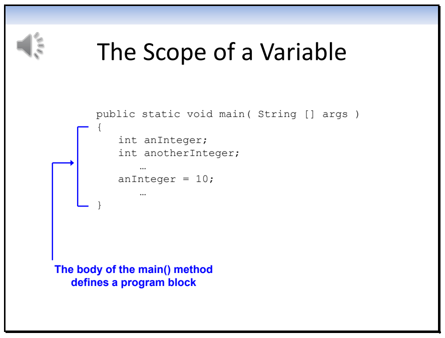
```
{

    … some programming statements …

}
```

First, let’s understand what a Java program block is. A block is a portion of a program that is contained within a set of opening and closing braces. Any programming statements appearing within the set of braces are in that program block.

4



The Scope of a Variable

```
public static void main( String [] args )
{
   int anInteger;
   int anotherInteger;
      …
   anInteger = 10;
      …
}
```

The body of the main() method defines a program block

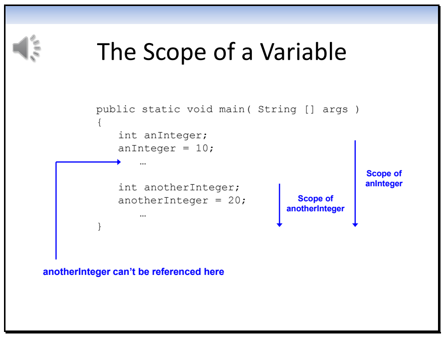
Take a look at the main() method in this example.

The braces define what is called the body of the method. All the code that is inside the braces belongs to the main() method. The braces also define a block.

Earlier, I said that the scope of a variable is from the point at which the variable is declared to the end of the block the variable is declared in. In this example, the variables ‘anInteger’ and ‘anotherInteger’ are defined at the beginning of the main() method, so their scope is from the point at which they are declared to the end of the main() method.

I can reference them…meaning use them…anywhere in main() after their declarations. In this case, I can also say, for example, that the scope of ‘anInteger’ and ‘anotherInteger’ is the main() method. The main() method is the only part of the program over which these variables are known.

5



The Scope of a Variable

```
public static void main( String [] args )
{
   int anInteger;
   anInteger = 10;
      …

   int anotherInteger;
   anotherInteger = 20;
      …
}
```

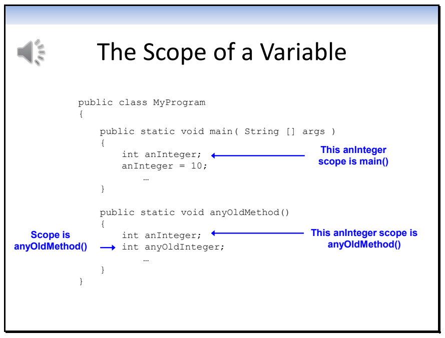
Scope of anInteger
Scope of anotherInteger
anotherInteger can't be referenced here

Now let’s look at a slightly different example. In this example, the scope of ‘anInteger’ is the entire main() method [*]. I can reference it anywhere in main().

The scope of ‘anotherInteger’ is not the entire main() method. In this case, it begins at the point of declaration and extends to the remainder of the main() method.

‘anotherInteger’ can not be referenced from any statement above its point of declaration. If I try to reference it before it is declared the program will not compile.

6



The Scope of a Variable

```
public class MyProgram
{
    public static void main( String [] args )
    {
        int anInteger;
        anInteger = 10;
            …
    }

    public static void anyOldMethod()
    {
        int anInteger;
        int anyOldInteger;
            …
    }
}
```

This anInteger scope is main()

Scope is anyOldMethod()

This anInteger scope is anyOldMethod()

Let’s take yet another example. In this example, my program consists of two methods…a main() method and a method called ‘anyOldMethod()’.

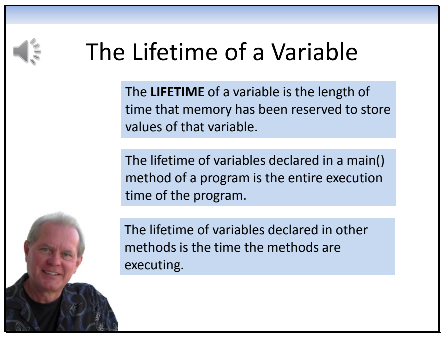
Notice that a variable called ‘anInteger’ is declared in main() and in ‘anyOldMethod()’. These are two different variables even though they have the same name. Values for these variables will be stored in two different memory locations.

The scope of the first ‘anInteger’ is the main() method [*]…land the scope of the second ‘anInteger’ is ‘anyOldMethod()’ [*].

When thinking about, or talking about, these variables, professional programmers would say that the ‘anInteger’ variable defined in main() is LOCAL to main, and the ‘anInteger’ variable defined in ‘anyOldMethod() is LOCAL to that method.

Another variable, named ‘anyOldInteger’ is defined in the ‘anyOldMethod()’ , so its scope is that method.

7



The Lifetime of a Variable

The **LIFETIME** of a variable is the length of time that memory has been reserved to store values of that variable.

The lifetime of variables declared in a main() method of a program is the entire execution time of the program.

The lifetime of variables declared in other methods is the time the methods are executing.

Another important Java programming concept is the LIFETIME of a variable.

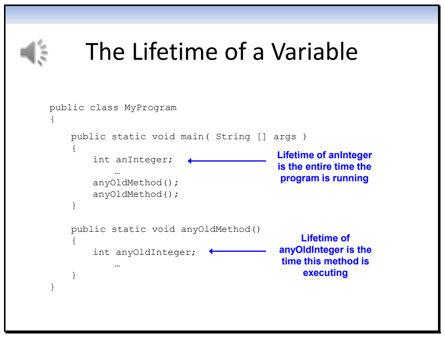
The lifetime of a variable is the length of time that memory has been reserved to store values of that variable when the program executes.

The lifetime for variables declared in the main() method is the entire time the program is executing. Memory for storing these variables is allocated at the start of program execution, and this storage allocation endures for the entire time the program is running.

For variables defined in other methods, memory for storing those variables will be allocated each time the method is called, and will only endure for the amount of time that method is executing. When a method completes execution, the memory used to store its variables is freed up…or de-allocated. Each time a method is invoked, the allocation and de-allocation of memory for its variables will be repeated, but the actual storage locations may be different.

LIFETIME is also called DURATION.

8



The Lifetime of a Variable

```
public class MyProgram
{
    public static void main( String [] args )
    {
        int anInteger;
            …
        anyOldMethod();
        anyOldMethod();
    }

    public static void anyOldMethod()
    {
        int anyOldInteger;
            …
    }
}
```

Lifetime of anInteger is the entire time the program is running

Lifetime of anyOldInteger is the time this method is executing

In this program, memory for storing ‘anInteger’ will be allocated at the beginning of program execution and will endure for the entire time the program is running.

Note that the main() method calls ‘anyOldMethod()’ twice. Also note that ‘anyOldMethod()’ declares a variable called ‘anyOldInteger’.

The lifetime of ‘anyOldInteger’ is the time that ‘anyOldMethod()’ is executing. The first time ‘anyOldMethod()’ is called, memory will be allocated for ‘anyOldInteger’. When the method finishes, the memory location that was allocated for ‘anyOldInteger’ is released, and may be used to store something else. The second time ‘anyOldMethod()’ is called, memory is once again allocated for ‘anyOldInteger’, and is released again when the method completes execution. The specific memory locations for storing ‘anyOldInteger’ may be different each time the method is called.