Variable visibility

1. Local variables

Let's have a more serious talk about variables. But this time we won't discuss their internal structure. Instead, we'll focus on how variables interact with the code where they are located.

All variables that are declared inside methods are called *local variables*. A local variable exists only in the block of code in which it is declared. Or, to be more precise, it exists from the moment it is declared until the end of the block of code in which it is declared.

For simplicity, let's consider an example:

Code	Variable visibility
`public static void main(String[] args)`	
`{`	
`   int a = 5;`	`a`
`   if (a < 10)`	`a`
`   {`	`a`
`      int b = 10;`	`a, b`
`      while (true)`	`a, b`
`      {`	`a, b`
`         int x = a + b;`	`a, b, x`
`         System.out.println(x);`	`a, b, x`
`      }`	`a, b`
`      System.out.println(b);`	`a, b`
`   }`	`a`
	`a`
`}`	

Let's talk about accessing local variables one more time. Here is a block of code consisting of curly braces: this could be a method body, the body of a loop, or just a block of code for a conditional statement. A variable declared in a block of code exists until the end of that block of code.

If a variable is declared in the body of the loop, then it will exist only in the body of the loop. It is created and destroyed at every iteration of the loop.

You cannot declare two local variables with the same name in one method — the program will not compile. But you can do this if the blocks of code where the variables are declared do not overlap.

Example:

Code	Variable visibility
`public static void main(String[] args)`	
`{`	
`   int a = 5;`	a
`   if (a < 10)`	a
`   {`	a
`      int b = 10;`	a, b
`      System.out.println(b);`	a, b
`   }`	a
	a
`   if (a < 20)`	a
`   {`	a
`      int b = 20;`	a, b
`      System.out.println(b);`	a, b
`   }`	a
`}`	

We were able to declare a second local variable named b only because the first b variable is not visible in the code block where the second b variable is declared.

---

2. Parameters

As we said before, each method can have variables that we call parameters. What about their visibility and lifetime?

It's all straightforward. Parameters are created when execution steps into the method (i.e. when the code of the method starts executing). They are eliminated when the method ends. They are visible throughout the body of the method.

Example:

Code	Variable visibility
`public static void main(String[] args)`	
`{`	`args`
`   int a = 5;`	`args, a`
`   if (a < 10)`	`args, a`
`   {`	`args, a`
`     int b = 10;`	`args, a, b`
`     while (true)`	`args, a, b`
`     {`	`args, a, b`
`       int x = a + b;`	`args, a, b, x`
`       System.out.println(x);`	`args, a, b, x`
`     }`	`args, a, b`
`     System.out.println(b);`	`args, a, b`
`   }`	`args, a`
	`args, a`
`}`	

As we said earlier, `args` is just a variable whose type is an array of strings. And like all parameters, it is available everywhere within the body of the method. That said, we usually ignore it in our examples.

3. Variables in a class

You will recall from the lessons in Level 1 that a class can have methods and variables. Methods are sometimes called instance methods, and variables — instance variables or fields. These are actually synonyms in Java.

What are the variables (or fields) of a class?

They are variables that are declared not in a method, but in a class.

They can be accessed from any (non-static) method of a class. Roughly speaking, *instance variables* are variables that are **shared** by all the methods of a class.

Example:

Code	Variable visibility
`public class Solution`	
`{`	
`   public int count = 0;`	`count`
`   public int sum = 0;`	`count, sum`
	`count, sum`
`   public void add(int data)`	`count, sum`
`   {`	`count, sum, data`
`     sum = sum + data;`	`count, sum, data`
`     count++;`	`count, sum, data`
`   }`	`count, sum`
	`count, sum`
`   public void remove(int data)`	`count, sum`
`   {`	`count, sum, data`
`     sum = sum - data;`	`count, sum, data`
`     count--;`	`count, sum, data`
`   }`	`count, sum`
`}`	`count, sum`

.In this example, we have two methods — `add()` and `remove()`. The `add()` method increments the `sum` and `count` instance variables, and the `remove()` method decreases the `sum` and `count` variables. Both methods work on shared instance variables.

Local variables exist while a method is executing. The **instance variables** of a **class** exist within an object of a class as long as that object exists. You'll learn details about objects of a class in the next level.

---

4. Static variables

Like methods, the variables in a class can be static or non-static. **Static methods can only access static variables.**

In Level 11, we'll analyze the structure of static variables and methods and you'll understand the reasons for these restrictions.

To make a static variable (class variable), you must write the `static` keyword in its declaration.

Static variables are not bound to an object or instance of the class in which they are declared. Instead, they belong to the class itself. That's why they exist **even if not a single object of the class has been created**. You can refer to them from other classes using a construct like:

```
ClassName.variableName
```

.Example:

Code	Variable visibility
`public class Solution`	
`{`	`Storage.count, Storage.sum`
`   public void add(int data)`	`Storage.count, Storage.sum`
`   {`	`Storage.count, Storage.sum, data`
`     Storage.sum = Storage.sum + data;`	`Storage.count, Storage.sum, data`
`     Storage.count++;`	`Storage.count, Storage.sum, data`
`   }`	`Storage.count, Storage.sum`
	`Storage.count, Storage.sum`
`   public void remove(int data)`	`Storage.count, Storage.sum`
`   {`	`Storage.count, Storage.sum, data`
`     Storage.sum = Storage.sum - data;`	`Storage.count, Storage.sum, data`
`     Storage.count--;`	`Storage.count, Storage.sum, data`
`   }`	`Storage.count, Storage.sum`
`}`	
`public class Storage`	
`{`	`Storage.count, Storage.sum`
`   public static int count = 0;`	`Storage.count, Storage.sum`
`   public static int sum = 0;`	`Storage.count, Storage.sum`
`}`	

In the above example, we created a separate `Storage` class, moved the `count` and `sum` variables into it, and declared them *static*. Public static variables can be accessed from any method in a program (and not only from a method).