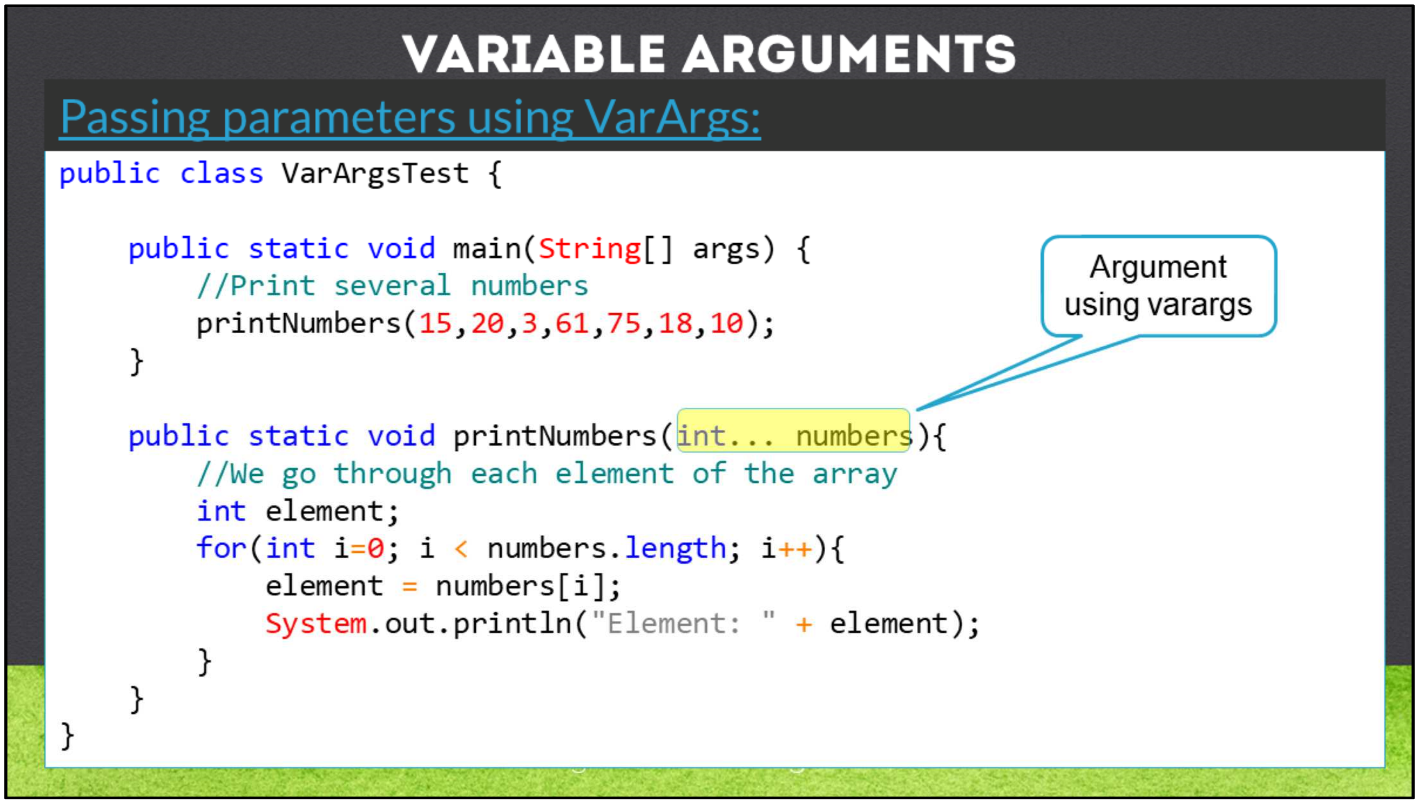
Hello, Ubaldo Acosta greets you again. I hope you're ready to start with this lesson.

We are going to study the topic variable arguments in Java.

Are you ready? Come on!



VARIABLE ARGUMENTS

Passing parameters using VarArgs:

```java
public class VarArgsTest {

    public static void main(String[] args) {
        //Print several numbers
        printNumbers(15,20,3,61,75,18,10);
    }

    public static void printNumbers(int... numbers){
        //We go through each element of the array
        int element;
        for(int i=0; i < numbers.length; i++){
            element = numbers[i];
            System.out.println("Element: " + element);
        }
    }
}
```

We have already seen the topic of parameters, but what happens when we want to pass several parameters of the same type to a function? One way would be to pass them as an array, however, as we know, we need to know the number of elements that the array will have.

One way to achieve send parameters in a simple and indirect way to create this array of parameters, is to use the concept of varargs, which are basically variable arguments.

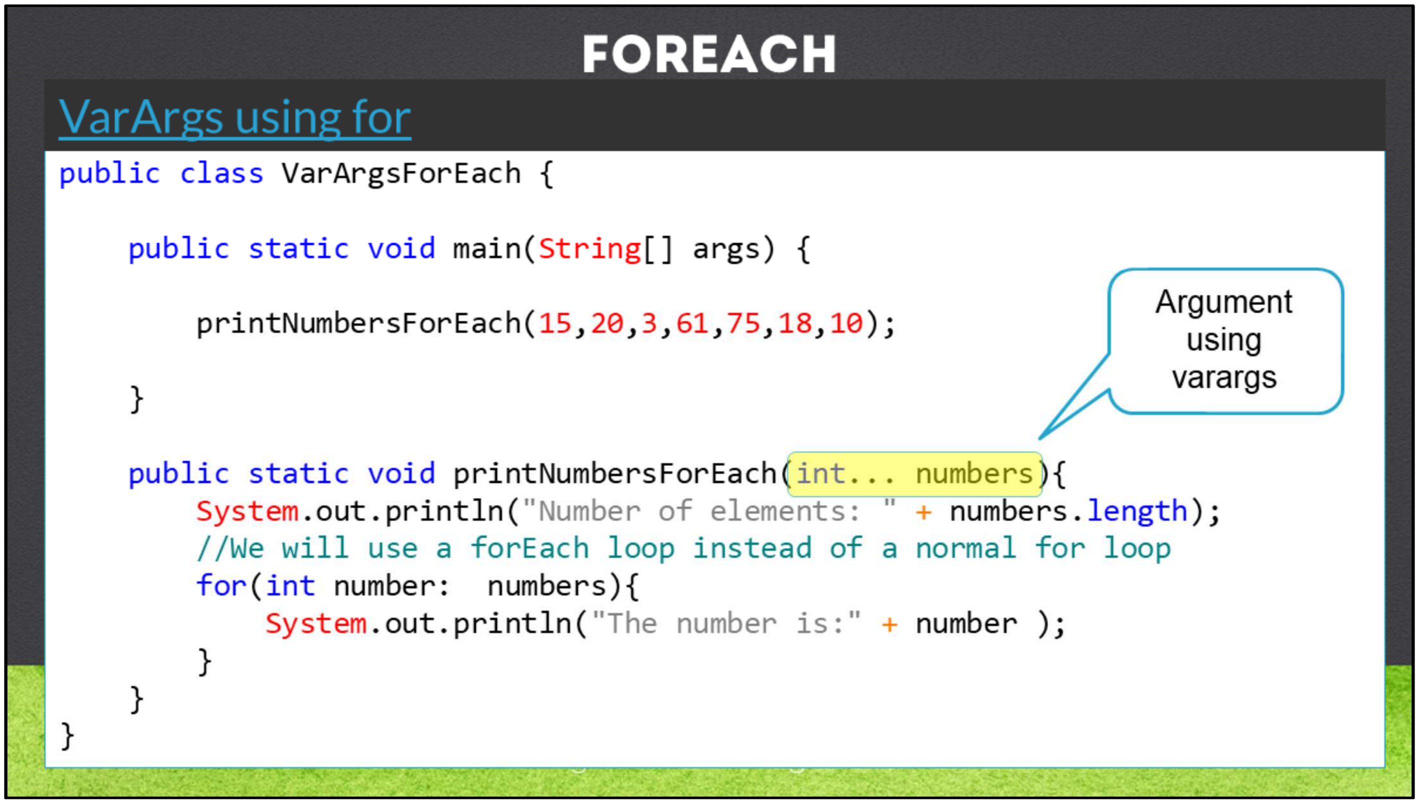
There are several rules for using varargs, one of which is that the elements that are sent, when they become an array at the end, must be of the same type.

In addition, to indicate that it is a variable argument, three points must be added (...) after the type of data that the argument will receive.

As seen in the code we have created a function called printNumbers, which receives an argument called numbers, and to define what is a variable type of arguments we have added the three points after the data type.

Later from the main method, what we have done is to call this method indicating a number of parameters of integer type, but without worrying about the number of elements that we are sending to the printNumbers method. This is precisely the characteristic of varargs, which create an array with the length of elements when receiving all the elements sent to the method.

Once the printNumbers method receives all the elements, create an array of the indicated type, and we can now go through each of the elements as we know that we must go through an array of one dimension.



FOREACH

VarArgs using for

```java
public class VarArgsForEach {

    public static void main(String[] args) {

        printNumbersForEach(15,20,3,61,75,18,10);

    }

    public static void printNumbersForEach(int... numbers){
        System.out.println("Number of elements: " + numbers.length);
        //We will use a forEach loop instead of a normal for loop
        for(int number:  numbers){
            System.out.println("The number is:" + number );
        }
    }
}
```

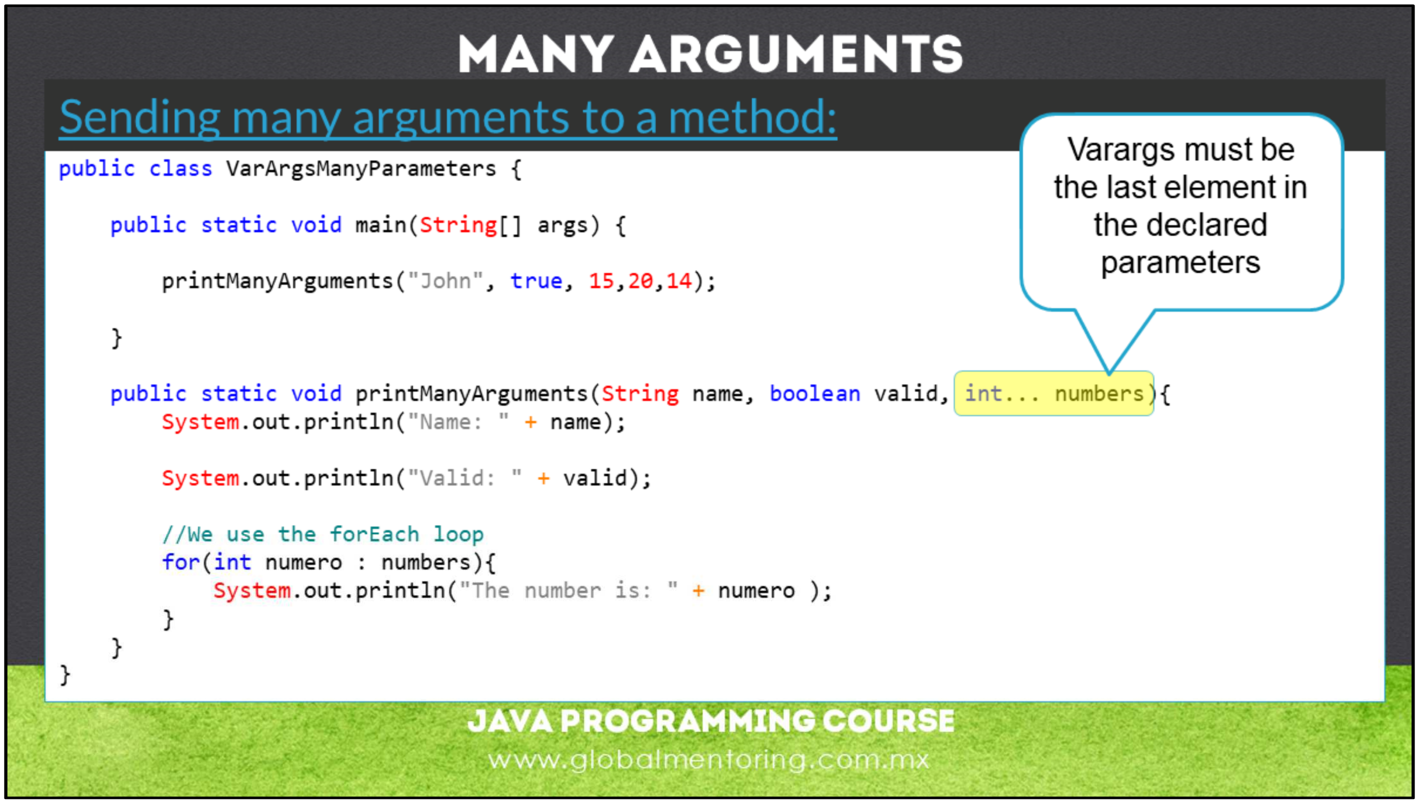
In the previous slide we used a for loop to iterate each of the elements of the vararg array, however there is a simplified notation of the for loop called forEach, which we can use to iterate arrays or collections, which we will study later.

However we want to show you how to simplify the iteration of the elements of an array using the concept of forEach. Basically what we do is declare a for loop, but unlike a normal for loop, which has 3 elements to declare, a forEach loop, it only has two elements to declare, which is the variable that each of the elements will receive, and the array to be iterated.

Java internally will be responsible for doing the respective iteration and within the forEach block what we will use is the variable that will contain each of the elements of the array for each iteration.

The only disadvantage is that if we need to know the index of the iterated element of this for loop, this syntax will not work, since at no time has a counter been declared to help us know which element is being iterated , since the forEach loop only returns the iterated element, but not the index that is being iterated.

That would be the only drawback to take into account, otherwise we see clearly how the syntax is cleaner and less prone to errors, since we do not need to declare several elements to iterate an array, so it is simpler and safer to use this notation to traverse the elements of an array.



MANY ARGUMENTS

Sending many arguments to a method:

```
public class VarArgsManyParameters {

    public static void main(String[] args) {

        printManyArguments("John", true, 15,20,14);

    }

    public static void printManyArguments(String name, boolean valid, int... numbers){
        System.out.println("Name: " + name);

        System.out.println("Valid: " + valid);

        //We use the forEach loop
        for(int numero : numbers){
            System.out.println("The number is: " + numero );
        }
    }
}
```

Varargs must be the last element in the declared parameters
JAVA PROGRAMMING COURSE
www.globalmentoring.com.mx

In this code we will basically explain one more rule of the use of varargs, which indicates that if we are going to use the varargs concept, it is necessary that it be the last one of the declared arguments, in this way to call the function, there will be no confusion for the compiler, because as soon as it detects that the data type of the defined vararg type begins, it will begin to create the array of elements sent to the function.

In short, the Java compiler detects the number of items sent and once it has finished assigning the non-vararg elements, it will check if the last argument declared is vararg type, and this is how all the remaining values that are of the same type as the vararg argument defined, will assign them to this vargarg element and as we know it finally converts them into an array.

This rule is very important for the use of vararg, since sometimes we want to combine it with several arguments, and that is where we must respect this rule.

Next we will create an exercise to practice each of these examples.

ONLINE COURSE

JAVA PROGRAMMING

Author: Ubaldo Acosta

Global Mentoring

JAVA UNIVERSITY

JAVA FUNDAMENTALS COURSE

www.globalmentoring.com.mx