Universidad
Carlos III de Madrid
www.uc3m.es

Lesson 5
Methods

Programming

Grade in Computer Engineering

Outline

1. Modular programming
2. Java methods
3. Method definition
4. Method calling
5. Variables and parameters in methods

Outline

1. **Modular programming**
2. Java methods
3. Method definition
4. Method calling
5. Variables and parameters in methods

Until now, if we wanted to perform an action in two different places, we had to write the code twice, one on each place (*code duplication*)

Example: Adding two matrices *MatrixWithoutMethods.java*

Is there a way to group and label a section of code which is frequently used?

or,

Programming languages must support code **reutilization** and **encapsulation**

Reutilization avoids implementing twice (or more) a code that does not change

E.g.: The code to add two matrices is the same, independently of their values

Encapsulation packages frequently used code in order to make possible to other programmers the use of common functionalities without knowing the actual implementation

E.g.: If we want to reuse the code for adding two matrices, we do not need to know that two `for` instructions are required

Modular programming

Programming paradigm that structures the code in sections (*modules*), each one with a well-defined purpose

In its simplest form, modular programming promotes the creation of **methods** encapsulating common operations

A method is a piece of code that uses a set of parameters as input and returns a result value as output

A method resembles the mathematical notion of function

Function definition *f: R --> R*
f(x) is a function applied on real numbers that has a real number as result

Function use *f(10)*
f(10) is a real number resulting from the application of f on 10

Outline

1. Modular programming
2. **Java methods**
3. Method definition
4. Method calling
5. Variables and parameters in methods

We have been using methods for a while…

`double Math.random()`
- no parameters
- returns a `double` value

```
double a = Math.random();
```

`double Math.sqrt(double x)`
- one parameter: value to calculate square root
- returns a `double` value

```
double a = Math.sqrt(26);
```

Methods that do not require the creation of an object to be called (the name of a class is used)

`int nextInt()`
- applied on `Scanner` objects
- no parameters
- returns an `int` value

```
Scanner sc = new Scanner(System.in);
int a = sc.nextInt();
```

`String substring(String s)`
- applied on `String` objects
- one parameter: `int` index to begin substring
- returns a `String`

```
String s = "Hello world!";
String sub = s.substring(6);
```

Methods that require the creation of an object to be called (the name of the object reference is used)

Methods are part of expressions. The type of a method call is the type of the returned value.
Methods are evaluated in expressions by applying the operations on the actual parameters.

Methods for dealing with 2D matrices

(A matrix is a two dimensional array of double values)

Operations

Add two matrices

Parameters: Matrices to be added (A, B)
Result: Result matrix (C)
Action: Calculates C = A + B

Matrix transpose

Parameters: Matrix to be transposed (A)
Result: Result matrix (B)
Action: B = Transposed of A

Matrix printing

Parameters: Matrix to be printed (A)
Result: Nothing
Action: Prints on the screen the values of A

…

A method is a delimited section of code that implements a complete operation

Creation and use of programmer-defined methods in Java requires:

1. Definition

Specification of the features of the method, mainly:

Name of the method	*addMatrices*
Parameters	2D `double` array A, 2D `double` array B
Type of the result value	2D `double` array
Name of the method	*factorial*
Parameters	`int` value
Type of the result value	`long` value

2. Call / use / invoke

Use of the method

Name of the class where the method is defined (optional)
`.` (point) operator
Name of the method
Parameter values (can be complete expressions, as long as they have a valid type)

```
double [][] matrix1, matrix2;
…
double [][] result = TestMatrix.addMatrices(matrix1, matrix2);


long f = factorial(120 + a);
```

Outline

1. Modular programming
2. Java methods
3. **Method definition**
4. Method calling
5. Variables and parameters in methods

`static` keyword

No objects have to created to run the method

Returning type

Type of the returning value

Name of the method

Name which will be used to call the method

Parameters

Type and name of the values that are *passed* to the method in the method call

Implementation (body)

Instructions performed by the method to obtain the result value

```
static long factorial(int x) {
        for(int i=0; …
}
```

Methods can return a value as a result of its execution

The `Math.sqrt` method returns a `double` value

`void` means that the method do not return any value

The `System.out.println` method does not return any value

Basic datatypes (`int`, `float`, `String`, etc.), arrays, and classes are valid returning types

```
static long factorial(int x) { … }
// returns a long

static double [][] add(double [][] A, double [][] B) { … }
// returns an array of double

static Student getRepresentative() { … }
// returns a Student object
```

To return a value, we use the keyword **`return`** inside the method body

If the method header indicates that a value is returned, it is compulsory to include a **`return`** with an expression with the expected type

Only **one value** can be returned

Returning a value **does not mean printing the value**

- The code who invoked the method can use the returned value in an expression (e.g. in the right side of an assignment)
- If the value is not stored in a variable in the method call, the value is lost

Method **parameters** store values that change in each call to the method

Actual parameters

Specific values used in the call to the method

Can be any expression of a suitable type

```
long a = factorial(25);
// the method is called with value 25

double x = Math.sqrt(a/2);
// the method is called with the value resulting from the
// evaluation of the expression
```

Parameters / arguments

Both names can be used

Arguments to the main method is just a particular case of method parameters

Formal parameters

Parameters declared in the method definition and used in the method body

```
static long factorial(int x) {
     …
     for(int i=0; i<=x; …
}
// the parameter is an integer value, which will be called x inside the method

static double sqrt(double x) {
    …
}
// the parameter is a dobule value, which will be called x inside the method
```

3. Method definition

Syntax

```
[modifiers] <type of returning value> <name of the method> (
          [  <type of parameter 1> <name of parameter 1>
             [, <type of parameter 2> <name of parameter 2> …] ] ) {

          <method body -- statements>
}
```

[modifiers]

public	**The method can be used by any class**
private	The method can be used only by the class which contains it
static	**The method belong to the class, it does not requires the creation of an object**
none	By default, the method is not static and can be used by the classes of the same package

<type of returning value>

type of the result value

<name of the method>

valid Java name

<type of parameter>

type of the input parameter

<name of parameter>

name which will be used inside the method to access the parameter value

Method definitions are place inside the class, but outside the `main` method

3. Method definition

Example - factorial

Factorial.java

```
import java.util.*;

public class Factorial {

    /** Factorial
     *  @param  n Integer value
     *  @return n!
     */
    public static double factorial(int n) {
        int f = 1;
        for(int i=2; i<=n; i++)
            f *= i;

        return f;
    }

    /** Main method */
    public static void main(String [] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter an integer value: ");
        int number = sc.nextInt();

        System.out.println("Factorial: " + factorial(number));
    }
}
```

Method definition

Method use

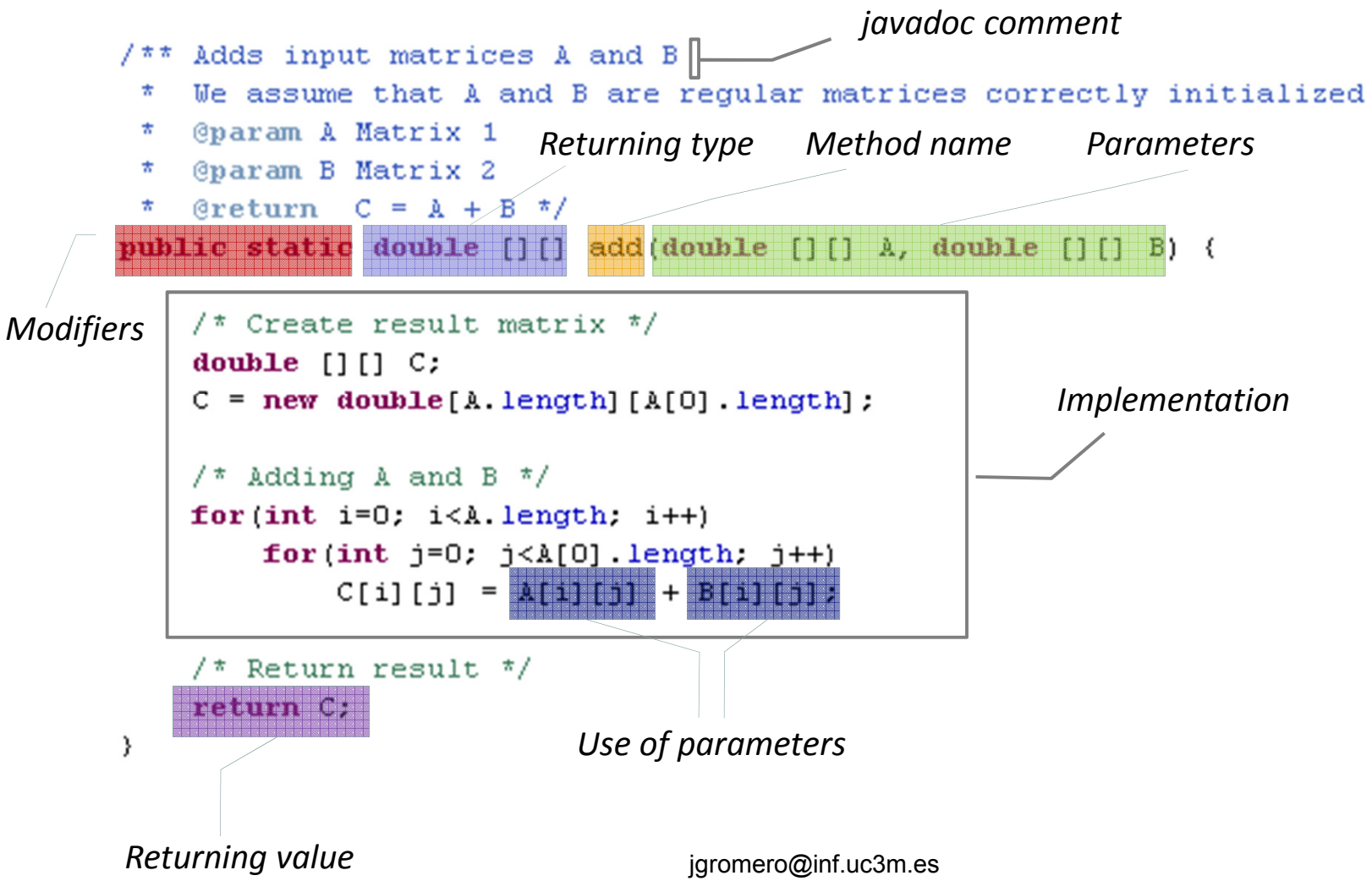
Method with returning value and parameters

```
/** Adds input matrices A and B
 *  We assume that A and B are regular matrices correctly initialized
 *  @param A Matrix 1
 *  @param B Matrix 2
 *  @return  C = A + B */
public static double [][] add(double [][] A, double [][] B) {

    /* Create result matrix */
    double [][] C;
    C = new double[A.length][A[0].length];

    /* Adding A and B */
    for(int i=0; i<A.length; i++)
        for(int j=0; j<A[0].length; j++)
            C[i][j] = A[i][j] + B[i][j];

    /* Return result */
    return C;
}
```

MatrixBasicMethods.java

3. Method definition

Examples - matrix




```
/** Adds input matrices A and B
 *  We assume that A and B are regular matrices correctly initialized
 *  @param A Matrix 1
 *  @param B Matrix 2
 *  @return  C = A + B */
public static double [][] add(double [][] A, double [][] B) {
    /* Create result matrix */
    double [][] C;
    C = new double[A.length][A[0].length];

    /* Adding A and B */
    for(int i=0; i<A.length; i++)
        for(int j=0; j<A[0].length; j++)
            C[i][j] = A[i][j] + B[i][j];

    /* Return result */
    return C;
}
```

3. Method definition

Examples - matrix

MatrixBasicMethods.java

```
public class MatrixBasicMethods {

    /** Adds input matrices A and B */
    public static double [][] add(double [][] A, double [][] B) {

        /* Create result matrix */
        double [][] C;
        C = new double[A.length][A[0].length];

        /* Adding A and B */
        for(int i=0; i<A.length; i++)
            for(int j=0; j<A[0].length; j++)
                C[i][j] = A[i][j] + B[i][j];

        /* Return result */
        return C;
    }

    /**
     * @param args No arguments are required in this program
     */
    public static void main(String[] args) {

        /* Create matrices */
        double [][] matrix1 = new double[][] {
                {1.0, 1.1},
                {1.0, -1.1}
        }; // 2x2 matix

        double [][] matrix2 = new double[][] {
                {-1.0, -1.1},
                {-1.0, 1.1}
        }; // 2x2 matix

        /* Add matrices */
        double [][] result; // result does not need to be initialized
        result = add(matrix1, matrix2);

        /* Print result */
        System.out.println("Result matrix (" + result.length + "x" + result[0].length + ")");
        for(int i=0; i<result.length; i++) {
            System.out.println();
            for(int j=0; j<result[i].length; j++)
                System.out.print(result[i][j] + " ");
        }
    }
}
```

Methods are defined **out** of the main, but **inside** the class

MatrixWithMethods.java

Method without return value

```
/** Print input matrix A
 *   @param A Matrix to be printed */
public static void print(double [][] A) {

    /* Print each element of A */
    for(int i=0; i<A.length; i++) {
        System.out.println();
        for(int j=0; j<A[i].length; j++)
            System.out.print(A[i][j]);
    }

}
```

Method without return value

Returning type (none)

Parameters

```
/** Print input matrix A
 *   @param A Matrix to be printed */
public static void print(double [][] A) {

    /* Print each element of A */
    for(int i=0; i<A.length; i++) {
        System.out.println();
        for(int j=0; j<A[i].length; j++)
            System.out.print(A[i][j]);
    }

}
```

No return instruction

Method without return value or arguments

```
/** Print author's name */
public static void printAuthorSignature() {

    /* Pretty print of author's name */
    System.out.println(
            "\n" +
            "    o" + "\n" +
            "    ,  ,  , __,  _ _    _,  __ _ _ _  _  __," + "\n" +
            "  _/|_(_/_(_/(_/ / /_  (_)_(_)/ / / /_(/_/_/_" + "\n" +
            "   /|                   /|                (/" +  "\n" +
            "  (/                   (/");
}
```

Method without return value or arguments

Returning type (none)

Parameters (none)

```
/** Print author's name */
public static void printAuthorSignature() {

    /* Pretty print of author's name */
    System.out.println(
            "\n" +
            "    o" + "\n" +
            "    ,  ,  , __,  _ _    _,  __ _ _ _  _  __," + "\n" +
            "  _/|_(_/_(_/(_/ / /_  (_)_(_)/ / / /_(/_/_/_" + "\n" +
            "   /|                   /|                (/" +  "\n" +
            "  (/                   (/");
}
```

Outline

1. Modular programming
2. Java methods
3. Method definition
4. **Method calling**
5. Variables and parameters in methods

4. Method calling

Example – matrix

```
 * @param args No arguments are required in this program
public static void main(String[] args) {

    /* Create matrices */
    double [][] matrix1 = new double[][] {
            {1.0, 1.1},
            {1.0, -1.1}
    }; // 2x2 matix

    double [][] matrix2 = new double[][] {
            {-1.0, -1.1},
            {-1.0, 1.1}
    }; // 2x2 matix

    /* Add matrices */
    double [][] result; // result does not need to be initialized
    result = add(matrix1, matrix2);

    /* Print result */
    print(result);

    /* Print result without storing the value */
    print(add(matrix1, matrix2));

    /* Print author's name */
    printAuthorSignature();
}
```

Methods can be called from the main, or from **other** methods

Result of a method can participate in an **expression**

4. Method calling

Example – matrix

MatricesMethods.java

```
/**
 * MatricesMethods
 *
 * @param args No arguments are required in this program
 */
public class MatricesMethods {

    /** Adds input matrices A and B */
    public static double [][] add(double [][] A, double [][] B) {

        /* Create result matrix */
        double [][] C;
        C = new double[A.length][A[0].length];

        /* Adding A and B */
        for(int i=0; i<A.length; i++)
            for(int j=0; j<A[0].length; j++)
                C[i][j] = A[i][j] + B[i][j];

        /* Return result */
        return C;
    }

    /** Print input matrix A */
    public static void print(double [][] A) {

        /* Print each element of A */
        for(int i=0; i<A.length; i++) {
            System.out.println();
            for(int j=0; j<A[i].length; j++)
                System.out.print(A[i][j]);
        }

    }

    /** Print author's name */
    public static void printAuthorSignature() {

        /* Pretty print of author's name */
        System.out.println(
                "\n" +
                "    o" + "\n" +
                "    ,  , , __,  _ _    _,  __ _ _ _  _ __," + "\n" +
                "  _/|_(_/_(_/(_/ / /_  (_)_(_)/ / / /_(/_/_/_" + "\n" +
                "   /|                  /|                (/" +  "\n" +
                "  (/                  (/");
    }

    /**
     * @param args No arguments are required in this program
     */
    public static void main(String[] args) {

        /* Create matrices */
        double [][] matrix1 = new double[][] {
                {1.0, 1.1},
                {1.0, -1.1}
        }; // 2x2 matix

        double [][] matrix2 = new double[][] {
                {-1.0, -1.1},
                {-1.0, 1.1}
        }; // 2x2 matix

        /* Add matrices */
        double [][] result; // result does not need to be initialized
        result = add(matrix1, matrix2);

        /* Print result */
        print(result);

        /* Print result without storing the value */
        print(add(matrix1, matrix2));

        /* Print author's name */
        printAuthorSignature();
    }
```

MatrixWithMethods.java

Outline

1. Modular programming
2. Java methods
3. Method definition
4. Method calling
5. **Variables and parameters in methods**

A copy of the value in the call to the method (*actual parameter*)is stored in the parameter (*formal parameter*)

The parameter is used as a variable, but it is not a variable: it does not need to be initialized

The scope of the parameter name (and the local variables defined inside the method) is the method

- The variables declared inside the method are not known outside it
  `main` does not know variables in methods, and methods do not know variables in `main`
- With basic types, the changes in the value of the variable passed as parameter does not affect the instructions outside the method
- With arrays, the changes in the variable does not affect the instructions outside the method, but modifications of the elements of the array are reflected

5. Variables and parameters in methods

Using a literal in a method call

```
public static void main(String[] args) {

    /* Variables in the method are not known by the main */
    // x = 10;                  // ERROR!

    /* If a variable is used in the call,
     * name matching is not required */
    int i = 5;
    method2(i);              // OK!

    /* Methods can be called with variables or values */
    method2(25);             // OK!

    /* Changes in the value of a variable used in a method
     * call are not reflected */
    int j = 12;
    method4(j);
    System.out.println("Printing j after assignment in method: " + j);

    /* Changes in an array variable passed as parameter are not reflected */
    int [] array = new int[] {1, 2, 3, 4, 5};
    System.out.println("\nArray length before the method: " + array.length);
    method6(array);
    System.out.println("Array length after the method: " + array.length);

    /* But changes in an array passed as argument are reflected */
    System.out.println("First element of array before the method: " + array[0]);
    method7(array);
    System.out.println("First element of array after the method: " + array[0]);

    /* Returned values can be used in expressions */
    int k;
    k = method5();
    System.out.println("\nk: " + k);

    /* If not assigned, the values are lost */
    int l;
    l = 10 + method5();
    System.out.println("\nl: " + l);


}
}
```

```
public static void method2(int a) {
    System.out.println("\nEntering method 2");
    System.out.println("Printing parameter a: " + a);

    /* The method does not known the variables of the main */
    // System.out.println(y);    // ERROR!

    System.out.println("Exit of method 2");
}
```

Console

<terminated> TestVariableScope [Java Application]

```
Entering method 2
Printing parameter a: 5
Exit of method 2

Entering method 2
Printing parameter a: 25
Exit of method 2
```

5. Variables and parameters in methods

Using a variable in a method call

```
public static void main(String[] args) {

    /* Variables in the method are not known by the main */
    // x = 10;                  // ERROR!

    /* If a variable is used in the call,
     * name matching is not required */
    int i = 5;
    method2(i);              // OK!

    /* Methods can be called with variables or values */
    method2(25);             // OK!

    /* Changes in the value of a variable used in a method
     * call are not reflected */
    int j = 12;
    method4(j);
    System.out.println("Printing j after assignment in method: " + j);

    /* Changes in an array variable passed as parameter are not reflected */
    int [] array = new int[] {1, 2, 3, 4, 5};
    System.out.println("\nArray length before the method: " + array.length);
    method6(array);
    System.out.println("Array length after the method: " + array.length);

    /* But changes in an array passed as argument are reflected */
    System.out.println("First element of array before the method: " + array[0]);
    method7(array);
    System.out.println("First element of array after the method: " + array[0]);

    /* Returned values can be used in expressions */
    int k;
    k = method5();
    System.out.println("\nk: " + k);

    /* If not assigned, the values are lost */
    int l;
    l = 10 + method5();
    System.out.println("\nl: " + l);


}
}
```

```
public static void method4(int a) {
    System.out.println("\nEntering method 4");
    System.out.println("Printing parameter a: " + a);

    /* Changes in the value of a are not seen out of the method */
    a = 50;
    System.out.println("Printing parameter a after assignment: " + a);
    System.out.println("Exit of method 4");
}
```

```
Console
<terminated> TestVariableScope [Java Application] C:\Archivos de programa\
Entering method 4
Printing parameter a: 12
Printing parameter a after assignment: 50
Exit of method 4
Printing j after assignment in method: 12
```

5. Variables and parameters in methods

Array reference – Formal parameter is changed

```
public static void main(String[] args) {

    /* Variables in the method are not known by the main */
    // x = 10;                  // ERROR!

    /* If a variable is used in the call,
     * name matching is not required */
    int i = 5;
    method2(i);              // OK!

    /* Methods can be called with variables or values */
    method2(25);             // OK!

    /* Changes in the value of a variable used in a method
     * call are not reflected */
    int j = 12;
    method4(j);
    System.out.println("Printing j after assignment in method: " + j);

    /* Changes in an array variable passed as parameter are not reflected */
    int [] array = new int[] {1, 2, 3, 4, 5};
    System.out.println("\nArray length before the method: " + array.length);
    method6(array);
    System.out.println("Array length after the method: " + array.length);

    /* But changes in an array passed as argument are reflected */
    System.out.println("First element of array before the method: " + array[0]);
    method7(array);
    System.out.println("First element of array after the method: " + array[0]);

    /* Returned values can be used in expressions */
    int k;
    k = method5();
    System.out.println("\nk: " + k);

    /* If not assigned, the values are lost */
    int l;
    l = 10 + method5();
    System.out.println("\nl: " + l);


}
}
```

```
public static void method6(int [] theArray) {
    /* Changes in the array are not seen out of the method */
    theArray = new int[15];
}
```

```
Console
<terminated> TestVariableScope [Java Application] C:\Archivos de programa\Jav

Array length before the method: 5
Array length after the method: 5
```

5. Variables and parameters in methods

Array reference – Array element is changed

```
public static void main(String[] args) {

    /* Variables in the method are not known by the main */
    // x = 10;                  // ERROR!

    /* If a variable is used in the call,
     * name matching is not required */
    int i = 5;
    method2(i);              // OK!

    /* Methods can be called with variables or values */
    method2(25);             // OK!

    /* Changes in the value of a variable used in a method
     * call are not reflected */
    int j = 12;
    method4(j);
    System.out.println("Printing j after assignment in method: " + j);

    /* Changes in an array variable passed as parameter are not reflected */
    int [] array = new int[] {1, 2, 3, 4, 5};
    System.out.println("\nArray length before the method: " + array.length);
    method6(array);
    System.out.println("Array length after the method: " + array.length);

    /* But changes in an array passed as argument are reflected */
    System.out.println("First element of array before the method: " + array[0]);
    method7(array);
    System.out.println("First element of array after the method: " + array[0]);

    /* Returned values can be used in expressions */
    int k;
    k = method5();
    System.out.println("\nk: " + k);

    /* If not assigned, the values are lost */
    int l;
    l = 10 + method5();
    System.out.println("\nl: " + l);


}
}
```

```
public static void method7(int [] theArray) {
    /* Changes in the elements of the array are seen out of the method */
    theArray[0] = -1;
}
```

```
Console
<terminated> TestVariableScope [Java Application] C:\Archivos de programa\J

First element of array before the method: 1
First element of array after the method: -1
```

The values of basic-type variables and the references to arrays (and objects) used in method calls **are not changed** inside methods

Array elements and object attributes **can be changed** inside methods

The only way to change one of these variable values is to assign the returning value of the method to it

(In the terminology of C++ and other languages, that means that all the parameters in Java are passed ***by value***)

5. Variables and parameters in methods

Summary

```
// Main method
public static void main(String [] args) {
    double x = 5.3;
    double x_squared = square(x);
}
```

x	5.3
x_squared	28.9

Idea:
value = x
value has the same value as x

5.3

28.9

```
// Square value method
public static double square(double value) {
    double result;
    result = value * value;

    return result;
}
```

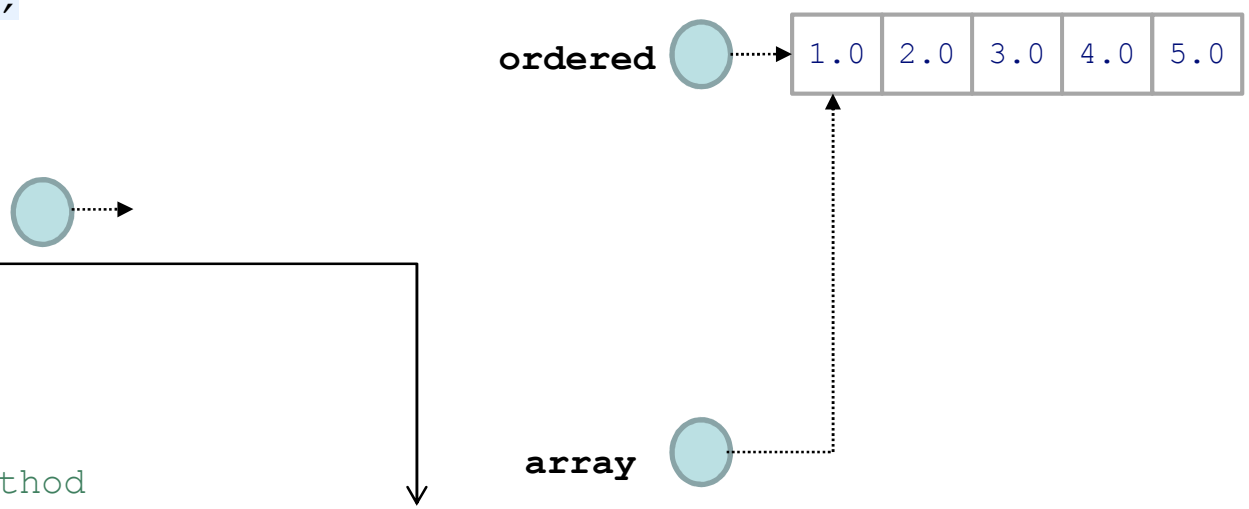
value	5.3
result	28.09

5. Variables and parameters in methods

Summary

```
// Main method
public static void main(String [] args) {
    double [] ordered = new double[] {1, 2, 3, 4, 5};
    reverse(ordered);
}
```

Idea:
array = ordered
array points to the same position as *ordered*




```
// Reverse array method
public static void reverse(double [] array) {
  for(int i=0, j=array.length-1; i < j; i++, j--) {
    double aux = array[i];
    array[i] = array[j];
    array[j] = aux;
  }
}
```

> Call to a method

a. Static methods

Do not require the creation of an object: *Math.sqrt*

b. Non static methods

Require the creation of an object of a class: *sc.nextInt*

> Static method definition

access modifier *public*

returning type type of the returning value

returning value is used in an expression

name of the method

parameters values that are specified when the method is called

A copy of the *actual parameter* is stored in the *formal parameter*

Recommended lectures

See Lesson 6 - Introduction to Object-Oriented Programming

Programming – Grado en Ingeniería Informática	
Authors	
Of this version: Juan Gómez Romero *Based on the work by:* Ángel García Olaya Manuel Pereira González Silvia de Castro García Gustavo Fernández-Baillo Cañas	 Universidad Carlos III de Madrid