

M₃-UF₂ –FUNCIONES

- ¿Por qué usar funciones?
- Métodos de Java
- Parámetros y valores retornados
- Cómo implementar un método
- Paso de parámetros
- Valores de retorno
- Creación de bibliotecas de rutinas
- Uso de bibliotecas de rutinas
- Ámbito o Alcance de las variables

¿Por qué usar funciones?

- A la hora de resolver un problema complejo es recomendable dividir el problema global en subproblemas más sencillos que nos sea más fácil manejar (**diseño descendente o top down**). Ej.; mostrarTablero, calcularPosiciones, validarJugada, marcarAciertoFallo, ...
- Cada subproblema tendrá su propia implementación, su propio **bloque de código** al que nos referiremos mediante un identificador, un nombre relacionado con el subproblema que soluciona.
- Por otro lado, en programación es muy frecuente **reutilizar código**
- Una **función** es un trozo de **código que realiza una tarea concreta** y que **se puede incluir en cualquier programa** cuando hace falta resolver esa tarea.
- Las funciones pueden recibir una entrada (**parámetros de entrada**) y/o **devolver una salida**.

Métodos

- **En Java** se debe hablar de **métodos** y no de funciones.
- Un método es una **secuencia de instrucciones** con un nombre
- El método **se declara mediante un nombre** para el bloque de código.

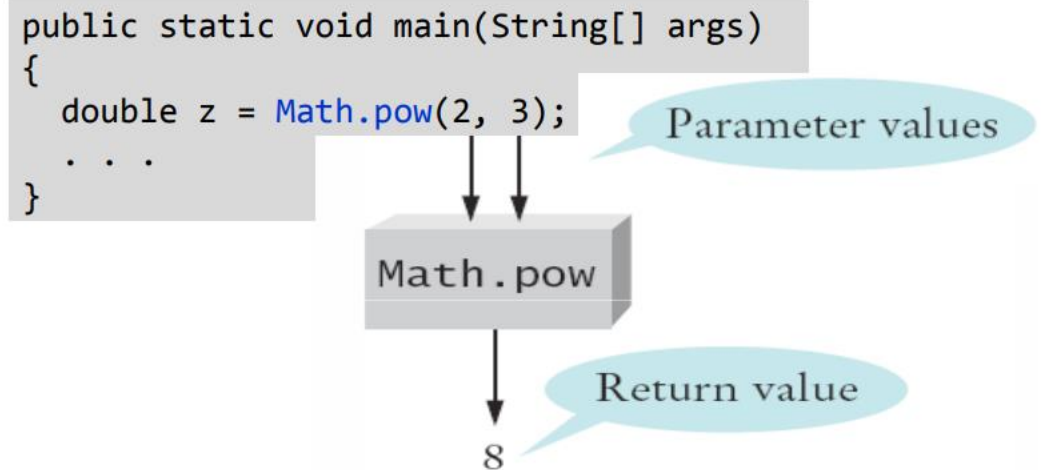
```
public static void main(String[] args)
{
    double z = Math.pow(2, 3);
    . . .
}
```

- Un método **se invoca** cuando se requiere para ejecutar sus instrucciones. Facilita la reutilización.

Métodos:

- Algunos métodos que ya se han venido usando:
 - `Math.pow()`, `Math.sqrt()`, `Math.random()`;
 - `str.length()`, `str.equals()`, `str.compareTo`
 - `Character.isDigit()`, `Character.getNumericValue()`
 - `Scanner.nextInt()`
 - `main()`
- Todos tienen:
 - Un nombre. Sigue las mismas reglas que las variables.
 - Paréntesis () para indicar los parámetros de entrada.

Parámetros y valores retornados



- main 'pasa' dos parámetros a Math.pow
- calcula y retorna un valor de 8 a main
- main almacena el valor devuelto en la variable 'z'

Cómo implementar un método

- Vamos a crear un método para calcular el volumen de un cubo.
 - ¿Qué información necesitará el método para funcionar?
 - ¿Qué responderá?
- Para desarrollar este método:
 - Crearemos la **cabecera**
 - Poner un nombre al método (`cubeVolume`)
 - Dar un tipo y un nombre para cada parámetro (`double sideLength`)
 - Especificar el tipo y el valor devuelto (`double`)
 - Agregar modificadores `public static`

```
public static double cubeVolume(double sideLength)
```

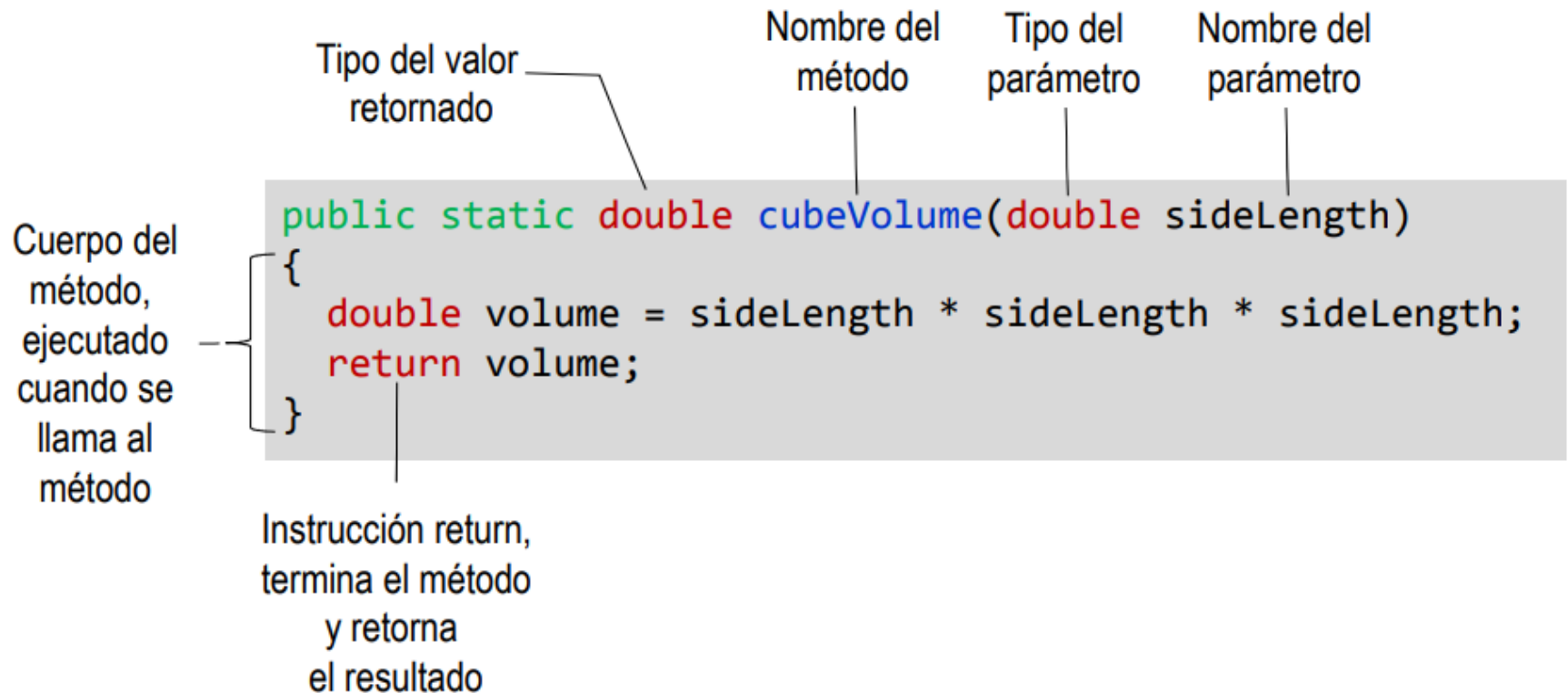
Cómo implementar un método

- Desarrollo del **cuerpo del método**
 - El cuerpo está encerrado entre llaves { }
 - El cuerpo contiene las declaraciones de variables e instrucciones que se ejecutan cuando se invoca el método
 - Retorna la respuesta calculada

```
public static double cubeVolume(double sideLength)
{
    double volume = sideLength * sideLength * sideLength;
    return volume;
}
```


Cómo implementar un método

- Desarrollo del **cuerpo del método**



Cómo implementar un método

- **Invocación** del método
 - El valor retornado por `cubeVolume` es almacenado en variables locales en main
 - Los resultados se imprimen

```
public static void main(String[] args)
{
    double result1 = cubeVolume(2);
    double result2 = cubeVolume(10);
    System.out.println("Un cubo de lado 2 tiene un volumen "
        + result1);
    System.out.println("Un cubo de lado 10 tiene un volumen "
        + result2);
}
```

Cómo implementar un método

- Programa completo

```
/** Programa que calcula el volumen de dos cubos. */
public class Cubos
{
    public static void main(String[] args)
    {
        double result1 = cubeVolume(2);
        double result2 = cubeVolume(10);
        System.out.println("Un cubo de lado 2 tiene un volumen " + result1);
        System.out.println("Un cubo de lado 10 tiene un volumen " + result2);
    }

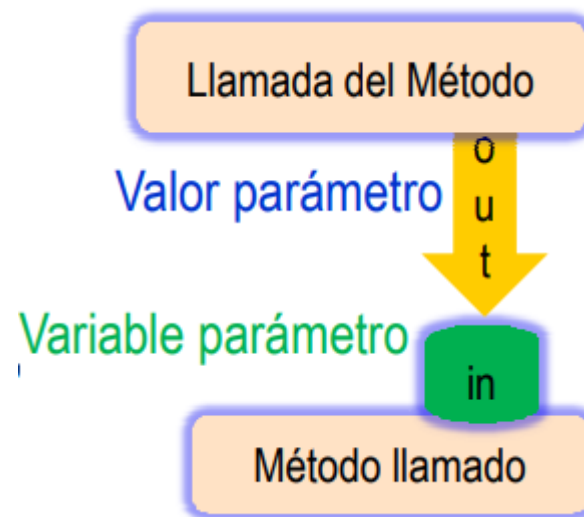
    /** Calcula el volumen de un cubo.
     * @param sideLength es la longitud del lado del cubo
     * @return volumen del cubo
     */
    public static double cubeVolume(double sideLength)
    {
        double volume = sideLength * sideLength * sideLength;
        return volume;
    }
}
```

meta-información

Comentarios de la
función

Paso de parámetros

- Las **variables parámetro** mantienen los **valores de los parámetros** suministrados en la llamada del método
 - Ambos deben ser del mismo tipo
- El **valor del parámetro** puede ser:
 - El contenido de una variable
 - Un valor literal
- La **variable parámetro** es:
 - Nombrada en la declaración del método llamado (en la cabecera)
 - Usada como una variable dentro del método llamado



Paso de parámetros

```
public static void main(String[] args)
{
    double result1 = cubeVolume(2);
    . . .
}
```

result1 = 8

```
public static double cubeVolume(double sideLength)
{
    double volume = sideLength * sideLength * sideLength;
    return volume;
}
```

sideLength = 2

volume = 8

Paso de parámetros

- **NOTA:**
 - En la mayoría de los lenguajes de programación es el programador quien decide cuándo un parámetro se pasa por valor y cuándo se pasa por referencia.
 - En Java no podemos elegir.
 - Todos los parámetros que son de **tipo int, double, float, char o String se pasan siempre por valor** mientras **que los arrays se pasan siempre por referencia**.

Paso de parámetros

- En la mayoría de lenguajes existen 2 formas de pasar parámetros a una función:
 - **Por valor:**
 - Cuando se pasa un parámetro por valor, en realidad se pasa una copia de la variable.
 - Cualquier modificación que se le haga a la variable que se pasa como parámetro dentro de la función no tendrá ningún efecto fuera de la misma.

```
public class PruebaParametros1 {  
    public static void main(String[] args) {  
  
        int n = 10;  
  
        System.out.println(n);  
        calcula(n);  
        System.out.println(n);  
    }  
  
    public static void calcula(int x) {  
        x += 24;  
        System.out.println(x);  
    }  
}
```

Salida:

10
34
10

Paso de parámetros

- **Por referencia:**
 - Cuando se pasa un parámetro por referencia, si se modifica su valor dentro de la función, los cambios se mantienen una vez que la función ha terminado de ejecutarse.

```
8 33 200 150 11
9 34 201 151 12
```

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

        int n[] = {8, 33, 200, 150, 11};
        int m[] = new int[5];

        muestraArray(n);
        incrementa(n);
        muestraArray(n);
    }


    public static void muestraArray(int x[]) {
        for (int i = 0; i < x.length; i++) {
            System.out.print(x[i] + " ");
        }
        System.out.println();
    }

    public static void incrementa(int x[]) {
        for (int i = 0; i < x.length; i++) {
            x[i]++;
        }
    }
}
```


Paso de parámetros: Error típico

- Intentar modificar los parámetros:

```
public static void main(String[] args)
{
    double total = 10;
    addTax(total, 7.5);
}
```

 total 10.0

```
public static int addTax(double price, double rate)
{
    double tax = price * rate / 100;
    price = price + tax; // No tiene efecto fuera del metodo
    return tax;
}
```

price 10.75

Métodos con valores de retorno

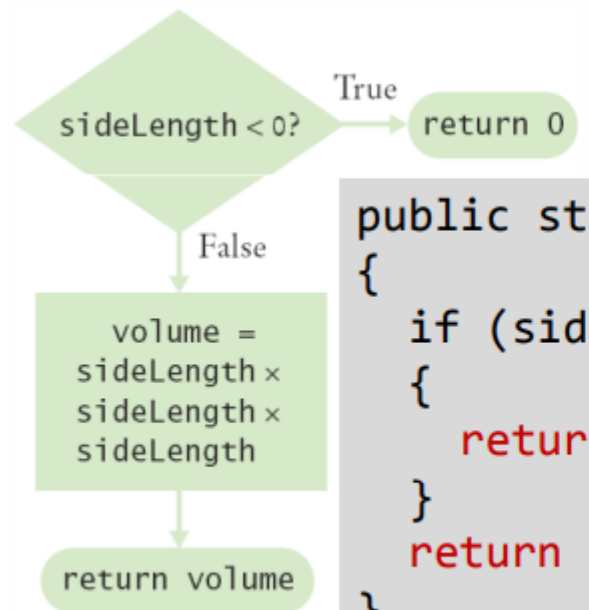
- Los métodos pueden retornar un valor
 - El tipo del **return** se especifica en la declaración del método
 - La instrucción **return** termina el método inmediatamente y retorna el valor especificado al método invocante

```
                Tipo return
                └──┬──
public static double addTax(double price, double rate)
{
    price += price * rate / 100;
    return price;
}
                └──┬──
                Valor return
```

- El valor return puede ser un valor, una variable o una expresión de cálculo
- El tipo debe coincidir con el tipo return

Métodos con valores de retorno

- Un método puede usar múltiples instrucciones return



```
public static double cubeVolume(double sideLength)
{
    if (sideLength < 0)
    {
        return 0;
    }
    return sideLength * sideLength * sideLength;
}
```

Métodos con valores de retorno

- En caso de tener métodos que no retornan valores se usa **void** como tipo del método

```
...  
boxString("Hello");  
...
```

```
-----  
!Hello!  
-----
```

```
public static void boxString(String str)  
{  
    int n = str.length();  
    for (int i = 0; i < n + 2; i++)  
        { System.out.print("-"); }  
    System.out.println();  
    System.out.println("!" + str + "!");  
    for (int i = 0; i < n + 2; i++)  
        { System.out.print("-"); }  
    System.out.println();  
}
```

Creación de bibliotecas de rutinas mediante paquetes

- Si por ejemplo tenemos una función `esPrimo()` que va a ser usada en tres programas diferentes se puede copiar y pegar su código en cada uno de los programas, pero hay una solución mucho más elegante y práctica.
- Agrupar funciones de un mismo tipo (por ejemplo matemáticas) para crear un paquete (package) que luego se importará desde el programa que necesite esas funciones.
- Cada paquete se corresponde con un directorio.
- Si hay un paquete con nombre `matematicas` debe haber un directorio llamado también `matematicas` en la misma ubicación del programa que importa ese paquete (normalmente el programa principal).
- Ej.: `matematicas`

Creación de bibliotecas de rutinas mediante paquetes

- Las funciones se pueden agrupar dentro de un paquete de dos maneras diferentes:
 - Puede haber subpaquetes dentro de un paquete
 - Cada subpaquete sería un subdirectorio dentro del directorio del paquete. Ej.: matemáticas.geometría, matemáticas.estadística
 - Crear varios ficheros dentro de un mismo directorio. Ej.: dentro de matemáticas tendríamos Geometría.java, Estadística.java

Creación de bibliotecas de rutinas mediante paquetes

```
package matematicas;

/**
 * Funciones matemáticas de propósito general
 *
 * @author Luis José Sánchez
 */
public class Varias {

    /**
     * Comprueba si un número entero positivo es primo o no.
     * Un número es primo cuando únicamente es divisible entre
     * él mismo y la unidad.
     *
     * @param x un número entero positivo
     * @return <code>true</code> si el número es primo
     * @return <code>false</code> en caso contrario
     */
    public static boolean esPrimo(int x) {

        for (int i = 2; i < x; i++) {
            if ((x % i) == 0) {
                return false;
            }
        }

        return true;
    }
}
```

```
/**
 * Devuelve el número de dígitos que contiene un número entero
 *
 * @param x un número entero
 * @return la cantidad de dígitos que contiene el número
 */
public static int digitos(int x) {

    if (x == 0) {
        return 1;
    } else {
        int n = 0;
        while (x > 0) {
            x = x / 10;
            n++;
        }
        return n;
    }
}
```

Creación de bibliotecas de rutinas mediante paquetes

```
package matematicas;

/**
 * Funciones geométricas
 *
 * @author Luis José Sánchez
 */
public class Geometria {

    /**
     * Calcula el volumen de un cilindro.
     * Tanto el radio como la altura se deben proporcionar en las
     * mismas unidades para que el resultado sea congruente.
     *
     * @param r radio del cilindro
     * @param h altura del cilindro
     * @return volumen del cilindro
     */
    public static double volumenCilindro(double r, double h) {
        return Math.PI * r * r * h;
    }

    /**
     * Calcula la longitud de una circunferencia a partir del radio.
     *
     * @param r radio de la circunferencia
     * @return longitud de la circunferencia
     */
    public static double longitudCircunferencia(double r) {
        return 2 * Math.PI * r;
    }
}
```


Uso de bibliotecas de rutinas mediante paquetes

```
import matematicas.Varias;
import matematicas.Geometria;

/**
 * Prueba varias funciones
 *
 * @author Luis José Sánchez
 */
public class PruebaFunciones {
    public static void main(String[] args) {

        int n;

        // Prueba esPrimo()

        System.out.print("Introduzca un número entero positivo: ");
        n = Integer.parseInt(System.console().readLine());

        if (matematicas.Varias.esPrimo(n)) {
            System.out.println("El " + n + " es primo.");
        } else {
            System.out.println("El " + n + " no es primo.");
        }

        // Prueba digitos()

        System.out.print("Introduzca un número entero positivo: ");
        n = Integer.parseInt(System.console().readLine());

        System.out.println(n + " tiene " + matematicas.Varias.digitos(n) + " dígitos.");

        // Prueba volumenCilindro()

        double r, h;

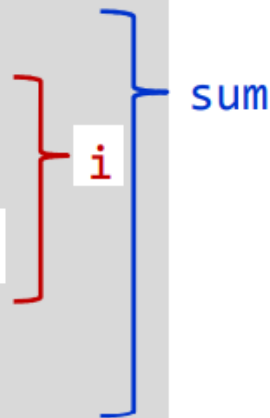
        System.out.println("Cálculo del volumen de un cilindro");
        System.out.print("Introduzca el radio en metros: ");
        r = Double.parseDouble(System.console().readLine());
        System.out.print("Introduzca la altura en metros: ");
        h = Double.parseDouble(System.console().readLine());

        System.out.println("El volumen del cilindro es " + matematicas.Geometria.volumenCilindro(r, h) + " m3");
    }
}
```

Ámbito o Alcance de las variables

- `sum` es una variable local en `main`
- `square` sólo es visible dentro del bloque del ciclo `for`
- `i` sólo es visible dentro del bloque del ciclo `for`

```
public static void main(String[] args)
{
    int sum = 0;
    for (int i = 1; i <= 10; i++)
    {
        int square = i * i;
        sum = sum + square;
    }
    System.out.println(sum);
}
```



Ámbito o Alcance de las variables

- Las variables locales dentro de un método no son visibles en otros métodos
 - `sideLength` es local a `main`
 - Causa un error de compilación

```
public static void main(String[] args)
{
    double sideLength = 10;
    int result = cubeVolume();
    System.out.println(result);
}

public static double cubeVolume()
{
    return sideLength * sideLength * sideLength; // ERROR
}
```

Ámbito o Alcance de las variables

- Las variables locales dentro de un método no son visibles en otros métodos
 - `result` es local a `square` y `result` es local a `main`
 - Son dos variables diferentes y no se solapan

```
public static int square(int n)
{
    int result = n * n;
    return result;
}

public static void main(String[] args)
{
    int result = square(3) + square(4);
    System.out.println(result);
}
```

`result`

`result`

Ámbito o Alcance de las variables

- Las variables declaradas en un bloque no son visibles en otros bloques
 - *i* está en el primer bloque for e *i* está en el segundo
 - Son dos variables diferentes y no se solapan

```
public static void main(String[] args) {  
    int sum = 0;  
    for (int i = 1; i <= 10; i++) {  
        sum = sum + i;  
    }  
    for (int i = 1; i <= 10; i++) {  
        sum = sum + i * i;  
    }  
    System.out.println(sum);  
}
```



Ámbito o Alcance de las variables

- Las variables, incluyendo los parámetros, deben tener un nombres únicos dentro de su alcance
 - `n` tiene alcance local y `n` está en un bloque dentro del alcance
 - El compilador dará error cuando se declara `n`

```
public static int sumOfSquares(int n) {  
    int sum = 0;  
    for (int i = 1; i <= n; i++) {  
        int n = i * i; // ERROR  
        sum = sum + n;  
    }  
    return sum;  
}
```

Diagram illustrating variable scope:

- A blue bracket on the right side of the code block groups the parameter `n` and the `for` loop block, labeled "Local n".
- A green bracket on the right side of the `for` loop block is labeled "alcance bloque n".

Ámbito o Alcance de las variables

- Las variables globales y locales (método) se pueden solapar
 - La variable local **same** se usará dentro de su alcance
 - No hay acceso a la v. global **same** cuando la local **same** está en su alcance

```
public class Scoper {  
    public static int same;    // global  
    public static void main(String[] args) {  
        int same = 0;        // local  
        for (int i = 1; i <= 10; i++) {  
            int square = i * i; same = same + square;  
        }  
        System.out.println(same);  
    }  
}
```

