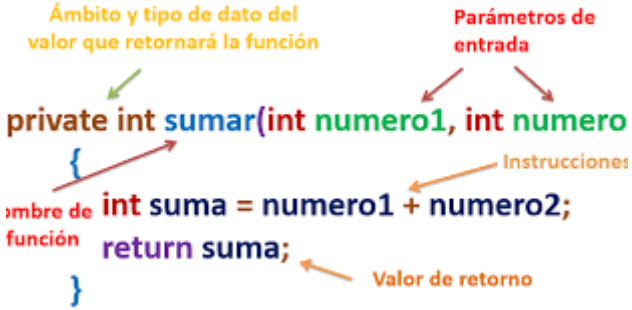
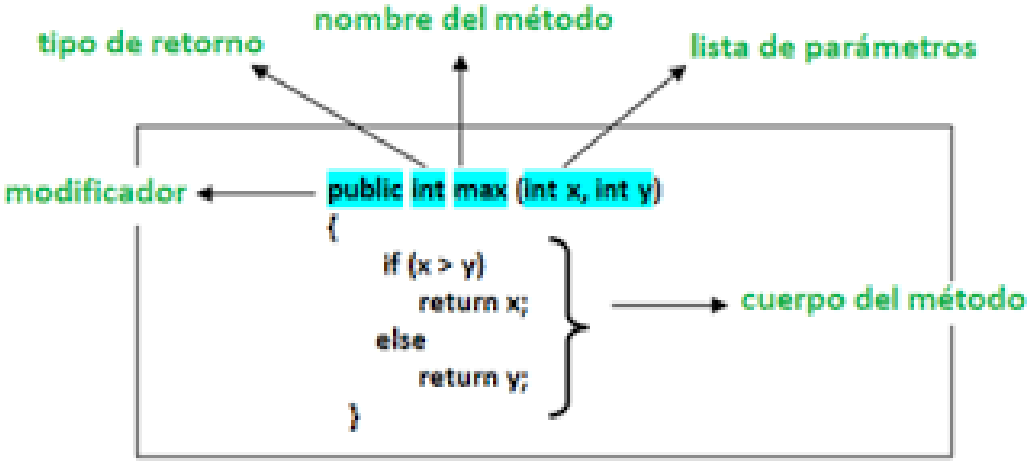


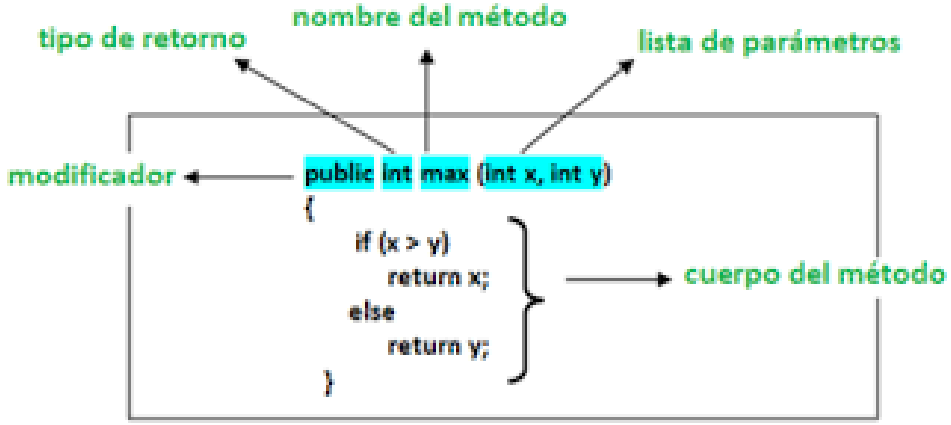
Métodos Procedimientos y funciones

Encarnación Sánchez Gallego

ÍNDICE

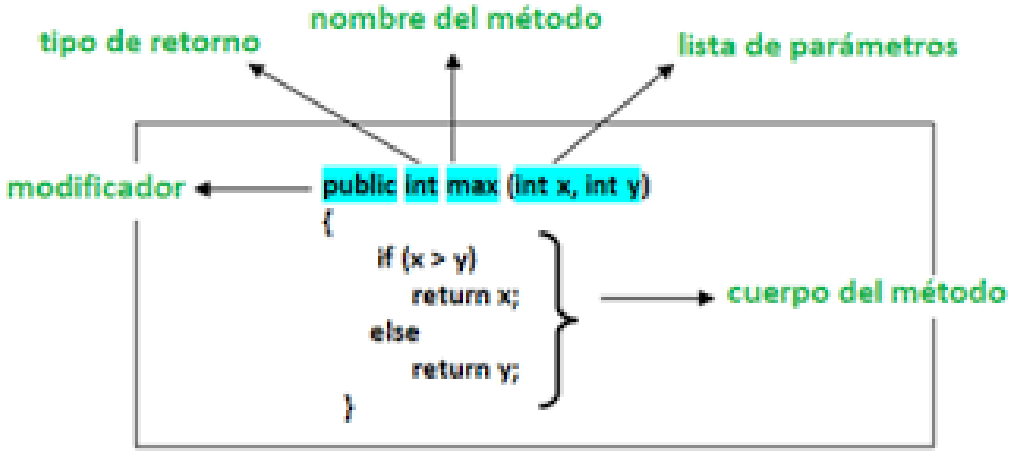


Introducción



- La experiencia ha demostrado que la mejor manera de desarrollar y mantener un programa extenso es construirlo a partir de pequeñas piezas sencillas, o módulos.
- A esta técnica se le llama divide y vencerás.
- La técnica de descomposición consiste en comenzar dividiendo un problema en un conjunto de subproblemas, a continuación se dividen éstos en otros más sencillos, y así sucesivamente.
- El proceso seguirá hasta que no se pueda o no convenga dividir más cada trozo, porque ya resulta suficientemente sencillo el desarrollo de éstos.

Introducción



- Los métodos (también conocidos como funciones o procedimientos en otros lenguajes) permiten al programador dividir un programa en módulos, por medio de la separación de sus tareas en unidades autónomas.
- Las instrucciones en los cuerpos de los métodos se escriben sólo una vez, y se reutilizan tal vez desde varias ubicaciones en un programa; además, están ocultas de otros métodos.

Algoritmos y programas

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La diferencia fundamental entre algoritmo y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado **lenguaje de programación** para que puedan ser ejecutados en el ordenador y así obtener la solución.

Procedimiento y función

- La función de nombre proviene de las matemáticas. Se utiliza para calcular un valor basado en la entrada.
- Un procedimiento es un conjunto de comandos que se pueden ejecutar en orden.
- En la mayoría de los lenguajes de programación, incluso las funciones pueden tener un conjunto de comandos. Por lo tanto, la diferencia es solo en la devolución de una parte de valor.
- Pero si desea mantener limpia una función (solo observe los lenguajes funcionales), debe asegurarse de que una función no tenga un efecto secundario.

Funciones

- **Utilidad principal de las funciones:**
  - Agrupar código que forma una entidad propia o una idea concreta.
  - Agrupar código que se necesitará varias veces en un programa, con al misión de no repetir código.
  - Dividir el código de un programa grande un subprogramas (funciones), cada uno de ellos especializados en resolver una parte del problema.
- **Características de las funciones:**
  - Se definen mediante un nombre único que representa el bloque de código.
  - Pueden ser llamadas (ejecutadas) desde cualquier parte del código.
  - Se les puede pasar valores para que los procesen de alguna forma.
  - Pueden devolver un resultado para ser usado desde donde se les llamó.

Declaración de una función

Declarar una función simplemente significa crearla para que luego pueda ser llamada (utilizada) desde otro lugar del código de nuestro programa. Una función se estructura en **cabecera** y **cuerpo**. La **cabecera** se declara en una sola línea y se compone de:

- **Modificadores de función:** Existen muchos pero los veremos en futuras unidades. Por ahora solo utilizaremos public static).
- **Tipo devuelto:** El tipo de dato que devolverá la función, como por ejemplo int, double, char, boolean, String, etc. Si la función no devuelve nada se indica mediante void.
- **Nombre de la función:** Identificador único para llamar a la función.
- **Lista de parámetros:** Indica los tipos y nombres de los datos que se le pasarán a la función cuando sea llamada. Pueden ser varios o ninguno.

El **cuerpo** es un bloque de código entre llaves { … } que se ejecutará cuando desde otra parte del código utilicemos la función.

Declaración de una función


```
[Modif_de_función] Tipo_devuelto Nombre_de_función (lista_de_parámetros)
{
  ...
}
```

Ejemplos de funciones:

```
public static void imprimeHolaMundo() {
        System.out.println("Hola mundo");
}
```

Este es un ejemplo muy sencillo de una función llamada 'imprimeHolaMundo', que no tiene parámetros de entrada (no hay nada entre los paréntesis) y no devuelve ningún valor (indicado por void). Cuando la llamemos lo único que hará será escribir por pantalla el mensaje "Hola mundo".

```
public static void imprimeHolaNombre(String nombre) {
        System.out.println("Hola " + nombre);
}
```

Esta función se llama 'imprimeHolaNombre', tiene como parámetro de entrada un dato String llamado 'nombre' y no devuelve nada. Cuando la llamemos imprimirá por pantalla el texto "Hola " seguido del String nombre que se le pase como parámetro.

Declaración de una función


```
public static int doble(int a) {
        int resultado = a * 2;
        return resultado;
}

 public static int multiplica(int a, int b) {
        int resultado = a * b;
        return resultado;
 }

public static double maximo(double valor1, double valor2) {
    double max;
    if (valor1 > valor2)
        max = valor1;
    else
        max = valor2;
    return max;
}
```

- Esta función se llama '**doble**', tiene como parámetro de entrada un dato int llamado 'a' y devuelve un dato de tipo int. Cuando la llamemos calculará el doble de 'a' y lo devolverá (con el return).

- Esta función se llama '**multiplica**', tiene dos parámetros de entrada de tipo int llamados 'a' y 'b' y devuelve un dato de tipo int. Cuando la llamemos calculará a*b y lo devolverá (con el return).

- Esta función se llama '**maximo**', tiene dos parámetros de entrada de tipo double llamados 'valor1' y 'valor2' y devuelve un dato de tipo double. Cuando la llamemos calculará el máximo entre 'valor1' y 'valor2' y lo devolverá.

```java
package unidad7;

public class programadeprueba {

    public static void imprimeHolaMundo() {
        System.out.println("Hola mundo");
    }

    public static int doble(int a) {
        int resultado = a * 2;
        return resultado;
    }

    public static int multiplica(int a, int b) {
        int resultado = a * b;
        return resultado;
    }

    public static void main(String[] args) {

        // Un programa siempre empieza ejecutándose por la función main.
        // Aquí va el código principal de nuestro programa.

    }
```

Es importante saber que las funciones se declaran dentro de class pero fuera de del main.

Las 3 funciones que hemos declarado arriba del main por sí solas no hacen nada, simplemente están ahí esperando a que sean llamadas (utilizadas), normalmente desde el propio main.

Llamadas a la función

Las funciones pueden ser invocadas o llamadas desde cualquier otra función, incluida ella misma. Sí, una función puede llamar a cualquier otra función, y una función puede llamarse a sí misma.
De todos modos **por ahora llamaremos funciones solo desde la función principal 'main'.** Así es más sencillo de aprender al principio.

Cuando se invoca una función el flujo de ejecución salta a la función (pasándole los parámetros si los hubiera), se ejecutan las instrucciones de la función y por último vuelve al punto que llamó a la función para seguir ejecutándose.

Las funciones se invocan con su nombre, pasando la lista de parámetros entre paréntesis. Si no tiene parámetros han de ponerse los paréntesis igualmente. Si la función devuelve un valor, para recogerlo hay que asignarlo a una variable o utilizarlo de algún modo (pueden combinarse funciones en expresiones e incluso pasarlo a otras funciones).

Ejemplo

```
public static void main(String[] args) {
        // No tiene parámetros ni devuelve valor. Simplemente imprime "Hola Mundo"
        imprimeHolaMundo();

        // Es habitual llamar a una función y guardar el valor devuelto en una variable
        int a = doble(10); // a = 20   (10*2)
        int b = multiplica(3, 5); // b = 15   (3*5)

        // Pueden pasarse variables como parámetros
        int c = doble(a); // c = 40   (20*2)
        int d = multiplica(a, b); // d = 300   (20*15)

        // Pueden combinarse funciones y expresiones
        int e = doble(4) + multiplica(2,10); // e = 8 + 20
        System.out.println("El doble de 35 es " + doble(35) ); // "El doble de 35 es 70"
        System.out.println("12 por 12 es " + multiplica(12,12) ); // "12 por 12 es 144"
}
```

Ejemplo

```java
public class Suma {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int num1, num2, suma;

        System.out.print("Introduce un número: ");
        num1 = sc.nextInt();

        System.out.print("Introduce otro número: ");
        num2 = sc.nextInt();

        suma = suma(num1, num2);

        System.out.println("La suma es: " + suma);

    }

    public static int suma(int n1, int n2) {

        int suma;

        suma = n1 + n2;

        return suma;
    }

}
```

Ejemplo

```java
public class ParImpar {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int num;

        System.out.print("Introduce un número: ");
        num = in.nextInt();

        if(par(num) == true) // Llamada a la función desde la expresión
            System.out.println(num + " es par.");
        else
            System.out.println(num + " es impar.");
    }

    public static boolean par(int numero)
    {
        boolean par = false;

        if(numero % 2 == 0) // Si el resto es 0 par será 'true' sino 'false'
            par = true;

        return par;
    }
}
```

Ámbito de las Variables

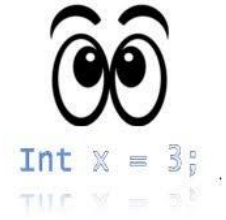
- **Una función solo puede utilizar las variables de ámbito local**, es decir, sus propias variables (los parámetros de la cabecera y las variables creadas dentro de la función). Cuando una función se ejecuta se crean sus variables, se utilizan y cuando la función termina se destruyen las variables.
- Por todo ello **una función no puede utilizar variables que estén fuera de ella**, y fuera de una función no es posible utilizar variables de la propia función. A esta característica se le llama **encapsulación** y permite que las funciones sean independientes entre sí, facilitando el diseño de programas grandes y complejos.

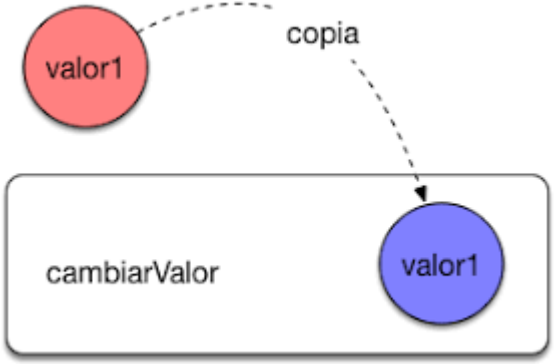
Técnicamente sí es posible que una función utilice variables que están fuera de ella, pero eso lo veremos en futuras unidades cuando aprendamos Programación Orientada a Objetos.

Parámetros: Paso por valor y por referencia

- Existen dos tipos de parámetros y es importante comprender la diferencia.
- **Parámetros de tipo simple (paso por valor):** Como int, double, boolean, char, etc. En este caso **se pasan por valor**. Es decir, **el valor se copia al parámetro** y por lo tanto si se modifica dentro de la función esto no afectará al valor fuera de ella porque **son variables distintas**.

```
public static void main(String[] args) {
        int a = 10;
        System.out.println("Valor inicial de a: " + a); // a vale 10
        imprime_doble(a); // Se le pasa el 10 a la función
        System.out.println("Valor final de a: " + a); // a sigue valiendo 10
}
```






```
// El parámetro 'a' es independiente de la 'a' del main. ¡Son variables distintas!
public static void imprime_doble(int a) { // Se copia el valor 10 a esta nueva 'a'
        a = 2 * a; // Se duplica el valor de la 'a' de esta función, no afecta fuera
        System.out.println("Valor de a en la función: " + a); // 'a' vale 20
}
```

SALIDA:

```
run:
Valor inicial de a: 10
Valor de a en la función: 20
Valor final de a: 10
```

Parámetros: Paso por valor y por referencia

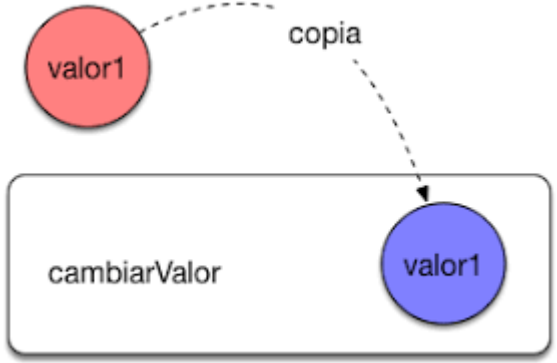
- **<u>Parámetros de tipo objeto (paso por referencias)</u>**: Como objetos de tipo String, los Arrays, etc. En este caso no se copia el objeto sino que **se le pasa a la función una referencia al objeto original (un puntero).** Por ello **desde la función se accede directamente al objeto** que se encuentra fuera. Los cambios que hagamos dentro de la función afectarán al objeto.

```
// Suma x a todos los elementos del vector v
public static void suma_x_al_vector(int v[], int x) {
        for (int i = 0; i< v.length; i++)
                v[i] = v[i] + x;
}

public static void main(String[] args) {
        int v[] = {0, 1, 2, 3};
        System.out.println("Vector antes: " + Arrays.toString(v));
        suma_x_al_vector(v, 10);
        System.out.println("Vector después: " + Arrays.toString(v));
}
```

SALIDA:

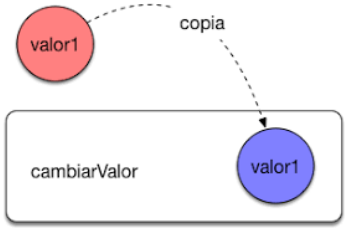
```
run:
Vector antes: [0, 1, 2, 3]
Vector después: [10, 11, 12, 13]
```



Parámetros: Paso por valor y por referencia

IMPORTANTE: Como un parámetro de tipo objeto es una referencia al objeto String o Array que está fuera de ella, si se le asigna otro objeto se pierde la referencia y ya no se puede acceder al objeto fuera de la función. Aunque Java permite hacerlo, no se aconseja hacerlo.


```
// Asignamos a x un nuevo vector, por lo que x dejará de apuntar al vector original.
// ¡El vector original no cambia! Simplemente ya no podemos acceder a él desde x
// porque se ha perdido la referencia a dicho objeto.
```




```
public static void funcion1(int x[]) {
        x = new int[10]; // x apuntará a un nuevo vector, el original queda intacto
        // lo que hagamos aquí con x no afectará al vector original
}

// Lo mismo sucede en este ejemplo, perdemos la referencia al String original
public static void funcion2(String x) {
        x = "Hola"; // x apuntará a un nuevo String, el original queda intacto
        // lo que hagamos aquí con x no afectará al String original
}
```

NO SE ACONSEJA HACER ESTE TIPO DE COSAS.

Devolución de un valor

Los métodos pueden devolver valores de tipo básico o primitivo (int, double, boolean, etc.) y también de tipo objeto (Strings, arrays, etc.).

- En todos los casos es el comando ***return*** el que realiza esta labor. En el caso de arrays y objetos, devuelve una referencia a ese array u objeto.