

PROGRAMACIÓN  
Escuela Universitaria de Informática - Curso 2001-2002  
*Tema 6. ABSTRACCIÓN DE OPERACIONES.  
MÉTODOS*

Alfonso Jiménez, Francisco Marqués, Natividad Prieto

## Índice General

<b>1</b>	<b>Introducción</b>	<b>1</b>
<b>2</b>	<b>Funciones</b>	<b>5</b>
2.1	Definición y uso . . . . .	5
2.2	Parámetros y resultado . . . . .	5
2.3	Efecto de la llamada a una función . . . . .	6
2.4	Ejemplos . . . . .	6
<b>3</b>	<b>Procedimientos</b>	<b>7</b>
3.1	Definición y uso . . . . .	7
3.2	Efecto de la llamada a un procedimiento . . . . .	8
3.3	Ejemplos . . . . .	8
<b>4</b>	<b>Paso de parámetros por valor y por referencia</b>	<b>9</b>
<b>5</b>	<b>Ámbito de definición de los métodos</b>	<b>10</b>
<b>6</b>	<b>Problemas</b>	<b>11</b>

## 1 Introducción

El proceso de **abstracción** consiste en hacer caso omiso de ciertas propiedades o circunstancias poco relevantes y enfatizar lo que realmente interesa de una situación o enunciado.

En nuestro contexto, la *abstracción de operaciones* significa hacer caso omiso de '**cómo**' se realizan ciertas operaciones enfatizando el '**qué**' es lo que hacen.

En el lenguaje Java, la abstracción de operaciones se consigue a través de los llamados **métodos** (que implementan operaciones dentro de una clase), de manera que unos métodos podrán invocar a otros de los que les ineteresará únicamente qué es lo que hacen y, en ningun caso, cómo realizan internamente la operación en cuestión.

De esta manera, el repertorio de instrucciones que ofrecen los lenguajes de programación (y Java en concreto) se amplía con la posibilidad de hacer *llamadas* a *métodos* definidos por el propio programador o por otros.

En los programas que se han realizado a lo largo del curso se han utilizado ciertos métodos que se definen en librerías del propio lenguaje, como por ejemplo `Math` o en librerías específicas, como por ejemplo `input`, `output` y `format`.

En concreto, se han utilizado los métodos `sin` y `sqrt` de la librería `Math`, y los métodos `readint` o `writeln` del paquete `nsIO`. Ejemplos de su utilización son:

```
// x es una variable de tipo double
double sx=Math.sin(x);

// x es una variable de tipo double y positiva
double rx=Math.sqrt(x);

input entrada = new input();
int i=entrada.readint();

output salida = new output();
salida.writeln('Hola a todos');
```

En todos los casos se observa que la utilización del método nos oculta los detalles de cómo se realiza internamente la operación que se requiere; por ejemplo, no sabemos si la invocación o llamada al método `Math.sin(x)` realizará un bucle para el cálculo del seno mediante un determinado desarrollo en serie, realmente no nos importa, lo importante es que cuando se invoque al método, éste realice la operación que se indica. De esta forma, la posibilidad de definir métodos es una herramienta de gran importancia en la programación ya que, como se ha comentado anteriormente, permite *la abstracción de operaciones*.

Si se observan los ejemplos descritos más arriba, se pueden extraer algunas características referentes al uso de métodos:

- Todos los métodos tienen un identificador `sin`, `sqrt`, `readint`, `writeln`
- Después del identificador y entre paréntesis figuran los *parámetros* del método. Los métodos también pueden no tener parámetros, como el método `readint`, y también puede ocurrir que algunos métodos puedan tener o no tener parámetros, como es el caso del `writeln`.
- Algunos métodos devuelven un resultado, por ejemplo `sin` o `sqrt` devuelven un resultado de tipo `double`, `readint` devuelve un resultado de tipo `int`. Otros métodos no devuelven ningún resultado explícitamente, como por ejemplo `writeln`.

A los métodos que devuelven un resultado explícitamente se les denomina *funciones*, a los otros *procedimientos*.

El programador también puede definir *funciones* o *procedimientos* propios. De hecho, la utilización de procedimientos y funciones en el desarrollo de programas es la base del diseño por refinamientos sucesivos y presenta las siguientes ventajas:

- Ahorra esfuerzo y tiempo cuando en la resolución de un problema se repite frecuentemente una misma secuencia de acciones.
- Facilita la resolución de problemas complejos describiendo su solución en términos de subproblemas más sencillos.
- Incrementa la legibilidad de los programas.

A esta forma de diseñar algoritmos o programas se le llama '**Diseño descendente**', '**Top Down**' o por '**refinamientos sucesivos**'. Los pasos a seguir para emplear esta metodología son:

1. Se considera el problema como un 'todo'.
2. Se divide el problema en subproblemas tal que, siendo de menor complejidad, forman parte del problema completo.
3. Se vuelve a aplicar el paso 2 a los subproblemas obtenidos hasta llegar a problemas de fácil solución o triviales.
4. Se solucionan los subproblemas 'triviales' y se agregan estas soluciones para obtener la solución al problema inicial.

De esta forma, el programador se concentra en solucionar problemas de menor complejidad, tal que, la solución combinada de todos ellos, es la solución al problema completo. En Java, la solución a cada uno de estos problemas, se corresponde con un **método**. Un programa en Java es una combinación de, al menos, 1 método. El método que siempre existirá, en todo programa Java, es: '**main**'.

En concreto, en la práctica número 4 se pedía el cálculo de la media aritmética de los valores almacenados en un cierto vector, de todos y también de una serie de valores consecutivos, y hacíamos:

```
double suma24h=0;

for (int i=0;i<72;i++) {
    suma24h+=p_sisto[i];
}
m_sisto_24h=suma24h/72;

double sumad=0;

for (int i=24;i<69;i++) {
```

```

    sumad+=p_sisto[i];
}
m_sisto_d=sumad/45;

```

Se observa que el cálculo que se realiza es exactamente el mismo, las únicas diferencias son: la posición inicial del vector y el número de valores que participan en el cálculo, o la posición del último. Estos serán los *parámetros de la operación* que serán de tipo del del índice del vector, esto es de tipo `int`. El *resultado* será la media aritmética que será de tipo real `double`. Para efectuar el cálculo es necesaria la variable sobre la que se acumula la suma de los valores que será una *variable local* de la función, variable que será del mismo tipo que los valores que se suman y por lo tanto entera, `int`. En Java esta función se escribe de la manera siguiente:

```

//Calcula la media aritmética de n valores de p_sisto a partir del
índice inicio
double media (int inicio, int n) {
    double suma=0;
    for (int i=inicio;i<(ini+n);i++) {
        suma+=p_sisto[i];
    }
    return (suma/n);
}

```

Si el segmento de programa anterior se reescribe haciendo uso de esta definición de la función `media` quedaría:

```

m_sisto_24h=media(0,72);
m_sisto_d=media(24,45);

```

En el programa propuesto en la práctica 4 también se calculaban los valores de las medias para el vector de medidas de presión diastólica. Para favorecer la reutilización de esta función `media` se puede añadir un parámetro que sea el vector donde están almacenadas las medidas que se van a analizar. Así:

```

//Calcula la media aritmética de n valores de v a partir del
índice inicio
double media (int v[], int inicio, int n) {
    double suma=0;
    for (int i=inicio;i<(ini+n);i++) {
        suma+=v[i];
    }
    return (suma/n);
}

```

Y el segmento de código quedaría:

```
m_sisto_24h = media(p_sisto,0,72);
m_sisto_d   = media(p_sisto,24,45);

m_diasto_24h= media(p_diasto,0,72);
m_diasto_d  = media(p_diasto,24,45);
```

## 2 Funciones

### 2.1 Definición y uso

Una función es un método que define el cálculo de un determinado valor. Distinguiremos entre la *definición de la función* y la *llamada a la función*. La definición de una función consta de una *cabecera*, una parte de *declaraciones de variables locales* y el *bloque de instrucciones* que indican el cálculo a realizar. El resultado de la función será devuelto mediante la instrucción `return`.

### 2.2 Parámetros y resultado

La cabecera de la función contiene la siguiente información:

- El tipo del valor que devuelve la función.
- El nombre de la función.
- Entre paréntesis los nombres de los parámetros y los tipos de los mismos.

La sintaxis de la cabecera de una función en Java es:

```
static tipoResultado nombreFuncion (tipo1 param1, tipo2 param2,..)
```

Los parámetros que se usan en la definición de la función (`param1` y `param2`) se llaman *parámetros formales*. Son nombres genéricos que serán sustituidos por los valores que se indiquen en la llamada a la función; a éstos se les denomina *parámetros reales*. En el momento de la llamada, ambos tipos de parámetros han de coincidir en número y tipo (según orden de aparición).

El *resultado* que devuelve la función es el que se indica en la instrucción `return` y ha de ser del tipo especificado en la cabecera `tipoResultado`. En una función sólo se ejecutará una instrucción 'return', aunque en el cuerpo de la misma aparezcan varias. Por ejemplo:

```
static int mayor (int x, int y) {
    if (x>=Y) return x;
        else return y;
}
```

El modificador de Java `static` es necesario para indicar que la función no está asociada a un objeto, sino que es accesible como parte de la clase donde se define. En el tema siguiente se explicará detalladamente lo que esto significa.

### 2.3 Efecto de la llamada a una función

Una llamada a una función se puede realizar siempre que el contexto sea válido para el resultado que devuelve la función. Por ejemplo, si una función devuelve un resultado de tipo `int`, puede ser llamada formando parte de una expresión aritmética, y en general en la parte derecha de una instrucción de asignación sobre una variable de tipo `int`.

Cuando se produce una llamada a una función en un contexto adecuado y con unos parámetros reales correctos (en número y tipos), se realizan los siguientes pasos:

1. Se crean los objetos locales, parámetros formales y variables locales del bloque principal que define la función. Los nuevos objetos correspondientes a los parámetros formales toman como valores los de los parámetros reales. Esta sustitución se llama *paso de parámetros por valor*.
2. Se ejecutan las instrucciones del cuerpo de la función y la función devuelve su valor.
3. Los objetos locales dejan de existir.
4. La ejecución del programa continúa en la instrucción inmediatamente siguiente a la llamada a la función.

Es necesario señalar que una función no se ejecuta hasta que no se produce una llamada a la misma. De la misma manera, los objetos locales a la función se crean cuando se produce la llamada y se destruyen cuando termina la ejecución de la función.

### 2.4 Ejemplos

La siguiente función comprueba si un determinado caracter es una letra minúscula o no:

```
static boolean minuscula? (char x) {  
    return ('a'<=x) & (x<='z');  
}
```

La siguiente función convierte una letra minúscula en mayúscula:

```
static char minusAMayus(char x) {  
    if (minuscula?(x))return (char)((int)x-(int)'a'+(int)'A');  
}
```

Estas funciones se podrían utilizar como parte de un programa para hacer: “si la letra leída es minúscula pasarla a mayúscula”. El código quedaría:

```
//Supongamos que se ha leído un carácter sobre la variable ch  
  
if minuscula?(ch) salida.write(minusAMayus(ch));
```

Se desea calcular el valor del número combinatorio 'N sobre M' que vale:  $N!/(M!(N-M)!)$ . Se puede diseñar un método (función) que calcule el factorial de un número positivo e invocarlo varias veces para solucionar el problema completo. La función quedaría:

```
static int fact (int x) {  
    // x es un número positivo o cero.  
    int f=1;  
    for (int i=2; i<=x; i++) f=f*i;  
    return (f);  
}
```

Ahora la solución al problema completo:

```
int N, M, combi; ...  
// Supongamos N y M valores enteros positivos, tal que N >= M  
  
combi=fact(N)/(fact(M)*fact(N-M));
```

## 3 Procedimientos

Un procedimiento es un método que define un conjunto de instrucciones que no devuelven explícitamente un valor. Por ejemplo, el conjunto de instrucciones que permite presentar por pantalla las componentes de un vector, o el el cálculo de la suma o el producto matricial son procedimientos típicos.

### 3.1 Definición y uso

En la definición de un procedimiento también se distingue entre la cabecera y el cuerpo del procedimiento. La cabecera de un procedimiento contiene la siguiente información:

- El nombre del procedimiento.
- Entre paréntesis los nombres de los parámetros y los tipos de los mismos.

La sintaxis de la cabecera de un procedimiento en Java es la de una función que devuelve un valor de tipo vacío, `void`:

```
static void nombreProcedimiento (tipo1 param1, tipo2 param2,...)
```

Los parámetros que se usan en la definición de la función (`param1` y `param2`) son los *parámetros formales*, que serán sustituidos por los valores de los *parámetros reales* en el momento de la llamada. Ambos parámetros (formales y reales) han de coincidir en número y tipo, según orden de aparición.

### 3.2 Efecto de la llamada a un procedimiento

Una llamada a un procedimiento se puede realizar como una instrucción más del lenguaje.

Cuando se produce una llamada a un procedimiento se realizan los siguientes pasos:

1. Se crean los objetos locales del procedimiento (parámetros formales y variables locales). Los nuevos objetos correspondientes a los parámetros formales toman como valores los de los parámetros reales (*paso de parámetros por valor*).
2. Se ejecutan las instrucciones del cuerpo del procedimiento.
3. Los objetos locales dejan de existir.
4. La ejecución del programa continúa en la instrucción inmediatamente siguiente a la llamada a la función.

### 3.3 Ejemplos

Un procedimiento para escribir los valores distintos de 0 de la matriz de pluviosidad del ejemplo del tema anterior sería:

```
static void EscribeMatriz ()
{
for (int mes=1; mes<=12; mes++)
  for (int dia=1; dia<=diasM[mes]; dia++)
    if (lluvia[mes][dia]>0)
      fs.writeln(dia+' '\t'+mes+' '\t'+lluvia[mes][dia]);
}
```

Si se desea que este procedimiento sea válido para escribir todos los valores de la matriz que superen un cierto umbral, sólo se tiene que definir dicho umbral como parámetro de este procedimiento. Así:

```
static void EscribeMatriz (int umbral)
{
for (int mes=1; mes<12; mes++)
  for (int dia=1; dia<=diasM[mes]; dia++)
    if (lluvia[mes][dia]>umbral)
      fs.writeln(dia+' '\t'+mes+' '\t'+lluvia[mes][dia]);
}
```

## 4 Paso de parámetros por valor y por referencia

Hasta el momento sólo se ha hablado del *paso de parámetros por valor*; estos parámetros se consideran *variables locales* al método, que se inicializan con los *valores* que tienen los parámetros reales cuando se produce la llamada al método.

Por el hecho de ser variables locales al método, cualquier cambio que éste realice sobre los parámetros no modifica el valor que tiene el parámetro real cuando termina la llamada. En Java *todos los parámetros de tipo simple (int, double, char, boolean) se pasan por valor*. Cuando tiene lugar un paso de parámetros por valor, ambos parámetros (formales y reales) son diferentes variables en memoria, la única relación entre ambos es que, en el momento de la llamada al método, el **valor** del parámetro real se copia en el parámetro formal y, a partir de ese momento, ya son totalmente independientes.

Por el contrario, *los parámetros de tipo vector, y en general cualquier objeto, se pasa siempre por referencia*.

En el *paso de parámetros por referencia* lo que se pasa en realidad es la dirección de la variable u objeto, es por esto que el papel del parámetro formal es el de ser una *referencia* al parámetro real; la llamada al método no provoca la creación de una nueva variable. De esta forma, las modificaciones que el método pueda realizar sobre estos parámetros se realizan efectivamente sobre los parámetros reales. En este caso, ambos parámetros (formal y real) se pueden considerar como la misma variable con dos nombres, uno en el método llamante y otro en el llamado o invocado, pero hacen **referencia** a la misma posición de memoria.

A continuación se muestran algunos ejemplos:

Dada la siguiente función, que calcula el factorial de un número entero positivo:

```
// Cálculo del factorial de x. x debe ser un entero positivo o 0
static int fact (int x) {
// x >= 0
  int f=1;
  for (int i=2; i<=x; i++) f=f*i;
  return (f);
}
```

Y el siguiente segmento de código:

```
{
  ....
  int y=4;
  System.out.println("El factorial de "+y+" es "+fact(y));
  ....
}
```

¿Qué salida produce la ejecución de este segmento de código?:

El factorial de 4 es 24

Supóngase que se modifica el código de la función `fact` añadiéndole una instrucción que modifique el valor del parámetro, por ejemplo `x++`. La salida que produce la ejecución del código anterior sería exactamente la misma, el valor del parámetro real: y no se ha incrementado.

Sea el siguiente procedimiento:

```
static void Incrementa (int v[], int n)
{
    for (int i=0; i<n;i++) v[i]++;
}
```

¿Cuál sería la salida del siguiente segmento de programa?

```
{
    int contador[]=new int[10];
    for (int i=0; i<10; i++) contador[i]=0;
    Incrementa(contador,10);
    for (int i=0; i<10; i++) System.out.println(contador[i]);
}
```

El efecto de este fragmento de programa es escribir en cada línea (10) el contenido de cada componente del vector que en este caso es 1. Hay que destacar que en este ejemplo sí se modifica el valor del parámetro de tipo vector ya que se trata de un paso de parámetros por referencia.

Veamos un método que suma dos matrices de números enteros:

```
static void SumaMatriz (int m1[][] ,int m2[][] ,int Suma[][] ,int N)
{
    for (int fila=1;fila<=N; fila++)
        for (int col=1;col<=N; col++)
            Suma[fila][col]=m1[fila][col]+m2[fila][col];
}
```

## 5 Ámbito de definición de los métodos

Las reglas que determinan el ámbito de definición de los métodos son muy similares a las que determinan el ámbito de definición de las variables. Algunas de estas reglas son:

- Todos los métodos que se definen en una clase deben de tener, en general, nombres (o identificadores) diferentes.
- Un método definido en una clase se puede utilizar desde cualquier punto de la clase.

- Un parámetro definido en un método se puede usar únicamente en ese método.
- No hay restricciones en el orden en el que se escriben en el fichero los métodos que pertenecen a una clase.

Por último, un método siempre ha de aparecer en una clase.

## 6 Problemas

1. Diseñese una función que devuelva el mayor de cuatro enteros que se le pasan como argumentos.
2. Escribese un programa en Java que construya un vector de 100 enteros, generados aleatoriamente, lea un valor entero y compruebe si dicho valor está o no en el vector. El proceso de búsqueda se debe implementar como una función.
3. Modifíquese la función de búsqueda escrita en el apartado anterior para que devuelva como resultado el índice dónde se encuentra el valor en caso de que la búsqueda tenga éxito y -1 si el valor no estaba en el vector.
4. Un grupo de amigos van de camping y llevan tres tiendas de campaña de 4, 3 y 2 plazas respectivamente. Escribid un programa que tome como dato el número de personas y devuelva como resultado el número de formas diferentes en las que pueden distribuirse.
5. Modifíquese la función `factorial` para que realice lo siguiente: tome como entrada cualquier entero, si el entero es positivo o 0 haga el cálculo habitual, si es negativo y par calcule el factorial del valor absoluto de la entrada y si es negativo pero impar, calcule este mismo valor pero con el signo cambiado.
6. Escribese una función que dada una secuencia de dígitos, compruebe si dicha secuencia es capicua..
7. Escribid un programa para que realice las operaciones matriciales de suma, resta y multiplicación sobre matrices cuadradas.
8. Escribid un programa que lea un texto y calcule el número de palabras de longitud 1, 2 y hasta 20.
9. Escribid un programa que lea un texto y muestre como resultado la frecuencia de aparición de cada uno de las letras del alfabeto.
10. Una frase es un *pangram* si contiene todas las letras del alfabeto. Escribese un programa para comprobar si una determinada frase es o no un *pangram*.
11. Diseñad un programa en Java que presente un menú de opciones para realizar las siguientes operaciones sobre vectores (todas las operaciones y el menú serán métodos, siendo *main* el método que vaya invocando al resto):

- Lectura, desde teclado, de los elementos de un vector de  $N$  componentes de tipo entero.
- Presentación en pantalla de los elementos de dicho vector.
- Búsqueda del máximo y el mínimo del vector.
- Sumatorio de los elementos del vector.
- Media de los elementos del vector.
- Desviación típica de los elementos del vector.
- Invertir el orden de los componentes del vector.
- Indicar la posición del primer número primo del vector (si existe) o devolver -1 si no existe ningún número primo.
- Sumatorio de los números primos que existan en el vector.