8. Sentencia return y métodos

Objetivos:
a) Describir el funcionamiento de la sentencia return
b) Interpretar el resultado de una sentencia return en el código fuente de una aplicación Java
c) Codificar una tarea sencilla convenientemente especificada utilizando la sentencia return

La sentencia `return` se emplea para salir de la secuencia de ejecución de las sentencias de un *método* y, opcionalmente, devolver un valor. Tras la salida del método se vuelve a la secuencia de ejecución del programa al lugar de llamada de dicho método.

Sintaxis:

`return` *`expresion`*`;`

8.1. Declaración y uso de métodos

Un *método* es un trozo de código que puede ser llamado o invocado por el programa principal o por otro método para realizar alguna tarea específica. El término *método* en Java es equivalente al de subprograma, rutina, subrutina, procedimiento o función en otros lenguajes de programación. El método es llamado por su nombre o identificador seguido por una secuencia de parámetros o argumentos (datos utilizados por el propio método para sus cálculos) entre paréntesis. Cuando el método finaliza sus operaciones, devuelve habitualmente un valor simple al programa que lo llama, que utiliza dicho valor de la forma que le convenga. El tipo de dato devuelto por la sentencia `return` debe coincidir con el tipo de dato declarado en la cabecera del método.

Sintaxis de declaración de un método:

```
[modificadores] tipoDeDato identificadorMetodo (parametros formales) {
    declaraciones de variables locales;
    sentencia_1;
    sentencia_2;
    ...
    sentencia_n;
    // dentro de estas sentencias se incluye al menos un return
    }
```

La primera línea de código corresponde a la cabecera del método. Los *modificadores* especifican cómo puede llamarse al método, el *tipo de dato* indica el tipo de valor que devuelve la llamada al método y los *parámetros* (entre paréntesis) introducen información para la ejecución del método. Si no existen parámetros explícitos se dejan los paréntesis vacíos. A continuación, las sentencias entre llaves componen el cuerpo del método. Dentro del cuerpo del método se localiza, al menos, una sentencia `return`.

Un ejemplo sencillo

Seguidamente se muestra un ejemplo de declaración y uso de un método que devuelve el cubo de un valor numérico real con una sentencia `return`:

```
/**
 *  Demostracion del metodo cubo
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaCubo {
    public static void main (String [] args){
        System.out.println("El cubo de 7.5 es: " + cubo(7.5));  // llamada
        }
    public static double cubo (double x) {                     // declaracion
        return x*x*x;
        }
    }
```

A diferencia de otros lenguajes de programación, como Pascal, en Java, la declaración del método puede realizarse en el código fuente después de la llamada al propio método. En el caso anterior, `public` y `static` son los modificadores especificados en la cabecera del método. El uso de estos dos modificadores permite que el tipo de método sea similar al de una función global de Pascal o C. El identificador `double` hace referencia al tipo de dato que devuelve la llamada al método, `cubo` es el identificador del método y `x` es el identificador del parámetro en la declaración de la cabecera del método (parámetro *formal*). Ejemplo de ejecución del código anterior y salida correspondiente por pantalla:

```
$>java PruebaCubo↵
El cubo de 7.5 es: 421.875
```

En Java, los métodos suelen ir asociados con los objetos o instancias en particular a los que operan (métodos *de instancia*). Los métodos que no necesitan o trabajan con objetos (y sí con números, por ejemplo) se denominan métodos *estáticos* o *de clase* y se declaran con el modificador `static`. Los métodos estáticos o de clase son equivalentes a las rutinas (funciones o procedimientos) de los lenguajes que no emplean la programación orientada a objetos. Por ejemplo, el método `sqrt` de la clase `Math` es un método estático. También lo es el método `cubo` del ejemplo anterior. Por otro lado, todo programa o aplicación Java (que no sea un *applet*) tiene un método principal `main` que será siempre un método *estático*.

¿Por qué se emplea la palabra `static` para los métodos de clase?. El significado o la acepción más común de la palabra estático (*que permanece quieto en un lugar*) parece no tener nada que ver con lo que hacen los métodos *estáticos*. Java emplea la palabra `static` porque C++ lo utiliza en el mismo contexto: para designar métodos *de clase*. Aprovechando su empleo en variables que tienen una única localización en memoria para diferentes llamadas a métodos, C++ lo empezó a utilizar en la designación de los métodos *de clase* para diferenciarlos de los métodos *de instancia* y no confundir al compilador. El problema es que nadie pensó en que el uso de la palabra `static` pudiera causar confusiones *humanas*.

Un ejemplo un poco más complicado

Las sentencias incluidas en el cuerpo del método no tienen porqué reducirse a una única sentencia `return`. Por ejemplo, en el siguiente código se introduce la declaración del método *estático* `factorial` que devuelve el factorial de un valor entero `n` dado como parámetro o argumento. Dentro del método `factorial` se declara localmente la variable `aux` de tipo `int` y se incluye una sentencia `for`. El factorial, n!, se define como el producto de 1·2·3·...·(n-1)·n cuando `n` es mayor que 1, siendo 1! = 1.

```
/**
```

```
 *  Demostracion de la funcion factorial
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaFactorial {
    public static void main (String [] args){
        System.out.println("El factorial de 10 es: " + factorial(10));
        }
    public static int factorial (int n) {    // declaracion del metodo
        int aux = 1;                         // declaracion local
        for (int i = 2; i<=n; i++)
            aux *= i;                        //  similar a aux = aux * i
        return aux;
        }
    }
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
$>java PruebaFactorial↵
El factorial de 10 es: 3628800
```

Una declaración, innumerables llamadas

La declaración del método se realiza una única vez en el código fuente, mientras que el número de llamadas al método puede ser cualquiera. Por ejemplo, el siguiente programa *intenta* realizar una tabla con el valor del factorial de los veinte primeros números naturales:

```
/**
 *  Demostracion de la funcion factorial
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaTablaFactorial {
    public static void main (String [] args){
        for (int k=1; k<=20; k++)
          System.out.println(k + "! = " + factorial(k));
        }
    public static int factorial (int n) {
        int aux = 1;
        for (int i = 2; i<=n; i++)
            aux *= i;                        //  aux = aux * i
        return aux;
        }
    }
```

El resultado de la ejecución del programa anterior por pantalla es el siguiente:

```
$>java PruebaTablaFactorial↵
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 1932053504
14! = 1278945280
```

```
15! = 2004310016
16! = 2004189184
17! = -288522240
18! = -898433024
19! = 109641728
20! = -2102132736
$>
```

Como se puede observar, el valor del factorial calculado para enteros superiores a 12 es incorrecto. El motivo es que el valor del factorial de un número crece muy deprisa con el operando. Así 12! es 479.001.600, cerca ya del límite superior del intervalo de representación del tipo primitivo `int` (2.147.483.647). El factorial de 13 (6.227.020.800) supera este límite y genera un error de cálculo por desbordamiento (*overflow*) del intervalo de representación. El problema se puede solventar (de nuevo limitadamente) utilizando un tipo primitivo con un intervalo de representación numérica mayor, por ejemplo, el tipo `long`:

```
/**
 *  Demostracion de la funcion factorial
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaTablaFactorial2 {
    public static void main (String [] args) {
        for (int k=1; k<=24; k++)
          System.out.println(k + "! = " + factorial(k));
        }
    public static long factorial (int n) {
        long aux = 1;
        for (int i = 2; i<=n; i++)
            aux *= i;                        //  aux = aux * i
        return aux;
        }
    }
```

El resultado de la ejecución del programa anterior por pantalla es el siguiente:

```
$>java PruebaTablaFactorial2↵
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
13! = 6227020800
14! = 87178291200
15! = 1307674368000
16! = 20922789888000
17! = 355687428096000
18! = 6402373705728000
19! = 121645100408832000
20! = 2432902008176640000
21! = -4249290049419214848
22! = -1250660718674968576
23! = 8128291617894825984
24! = -7835185981329244160
```

En este caso, el error se produce al tratar de calcular el factorial de un valor superior a 20.

Más parámetros

Por otro lado, el número de parámetros o argumentos de los métodos puede ser 0, 1, 2... En el siguiente ejemplo, el método `producto` devuelve el producto de dos valores enteros, `a` y `b`, dados como parámetros o argumentos:

```
/**
 *  Demostracion de la funcion producto
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaProducto {
    public static void main (String [] args) {
        System.out.println("Tabla de multiplicar del 5");
        for (int i=0; i<=10; i++)
            System.out.println("5 x " + i + " = " + producto(5,i));
        }
    public static int producto (int a, int b) {
        return a*b;
        }
    }
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
$>java PruebaProducto↵
Tabla de multiplicar del 5
5 x 0 = 0
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

8.2. Return y void

En algunas ocasiones, no es necesario que el método estático tenga que devolver un valor al finalizar su ejecución. En este caso, el tipo de dato que debe indicar en la cabecera de declaración del método es el tipo `void` y la sentencia `return` no viene seguida de ninguna expresión.

Sintaxis:

```
return;
```

En el siguiente código se incluye un ejemplo de método que no devuelve un valor (de tipo `void`):

```
/**
 *  Demostracion del metodo tabla
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaTabla {
```

```
    public static void main (String [] args){
        tabla(4);                                   // ejemplo de llamada
        tabla(7);
        }
    public static void tabla (int n) {              // de tipo void
        System.out.println("Tabla de multiplicar del numero " + n);
        for (int i=0; i<=10; i++)
            System.out.println(n + " x " + i + " = " + producto(n,i));
        return;                                     // No devuelve ningun valor
        }
    public static int producto (int a, int b) {
        return a*b;
        }
    }
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
$>java PruebaTabla↵
Tabla de multiplicar del numero 4
4 x 0 = 0
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
4 x 10 = 40
Tabla de multiplicar del numero 7
7 x 0 = 0
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Si no hay sentencia `return` dentro de un método, su ejecución continúa hasta que se alcanza el final del método y entonces se devuelve la secuencia de ejecución al lugar dónde se invocó al método.

Un método cuyo tipo de retorno no es `void` necesita siempre devolver algo. Si el código de un método contiene varias sentencias `if` debe asegurarse de que cada una de las posibles opciones devuelve un valor. En caso contrario, se generaría un error de compilación. Por ejemplo:

```
/**
 *  Demostracion de la funcion esPositivo
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaPositivo {
    public static void main (String [] args) {
        for (int i=5; i>=-5; i--)
            System.out.println(i + " es positivo: " + esPositivo(i));
        }
```

```
    public static boolean esPositivo(int x) {
        if (x<0) return false;
        if (x>0) return true;
        // Error: retorno perdido si x es igual a cero.
        }
    }
```

Ejemplo de intento de compilación del código anterior:

```
$>javac PruebaPositivo.java↵
pruebaPositivo.java:14: missing return statement
        }
        ^
```

Un ejemplo de código correcto sería:

```
/**
 *  Demostracion de la funcion esPositivo
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaPositivo2 {
    public static void main (String [] args){
        for (int i=5; i>=-5; i--)
            System.out.println(i + " es positivo: " + esPositivo(i));
        }
    public static boolean esPositivo(int x) {
        if (x<0) return false;
        else return true;
        }
    }
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
$>java PruebaPositivo2↵
5 es positivo: true
4 es positivo: true
3 es positivo: true
2 es positivo: true
1 es positivo: true
0 es positivo: true
-1 es positivo: false
-2 es positivo: false
-3 es positivo: false
-4 es positivo: false
-5 es positivo: false
```

8.3. Recursión o recurrencia

Java permite la *recursión* o *recurrencia* en la programación de métodos. La recursión consiste en que un método se llame a sí mismo. Un ejemplo muy típico de empleo de la recursión puede verse en la construcción de un método que devuelva el factorial de un entero. Se basa en el hecho de que n! es igual a n·(n-1)! si n es mayor que 1. Por ejemplo:

```
/**
 *  Demostracion de la funcion recursiva factorial
 *  A. Garcia-Beltran - marzo, 2004
 */
public class PruebaFactorialR {
    public static void main (String [] args){
```

```
        for (int i=1; i<=20; i++)
            System.out.println("Factorial de " + i + " = " + factorialR(i));
        }
    public static long factorialR (int n) {
        if (n==0) return 1;
           else return n * factorialR(n-1);
        }
    }
```

Ejemplo de salida por pantalla:

```
$>java PruebaFactorialR↵
Factorial de 1 = 1
Factorial de 2 = 2
Factorial de 3 = 6
Factorial de 4 = 24
Factorial de 5 = 120
Factorial de 6 = 720
Factorial de 7 = 5040
Factorial de 8 = 40320
Factorial de 9 = 362880
Factorial de 10 = 3628800
Factorial de 11 = 39916800
Factorial de 12 = 479001600
Factorial de 13 = 6227020800
Factorial de 14 = 87178291200
Factorial de 15 = 1307674368000
Factorial de 16 = 20922789888000
Factorial de 17 = 355687428096000
Factorial de 18 = 6402373705728000
Factorial de 19 = 121645100408832000
Factorial de 20 = 2432902008176640000
$>
```

Como ya se ha visto en un ejemplo anterior, el factorial de un número crece muy deprisa con el operando. Para evitar problemas a la hora de calcular el factorial de enteros superiores a 12 se ha vuelto a indicar el tipo `long` como tipo de dato de retorno del método estático `factorialR`.

En la construcción de métodos recursivos es importante evitar el problema de la recursión infinita. Es decir, que en algún caso, la ejecución del método definido de forma recursiva no implique una nueva llamada al propio método.

8.4. Sobrecarga de métodos

Java permite asignar el mismo identificador a distintos métodos, cuya diferencia reside en el tipo o número de parámetros que utilicen. Esto resulta especialmente conveniente cuando se desea llevar a cabo la misma tarea en difererente número o tipos de variables. La *sobrecarga* (*overloading*) de los métodos puede resultar muy útil al efectuar llamadas a un método, ya que en lugar de tener que recordar identificadores de métodos distintos, basta con recordar uno sólo. El compilador se encarga de averiguar cuál de los métodos que comparten identificador debe ejecutar. Por ejemplo:

```
/**
 *  Demostracion de metodos sobrecargados
 *  A. Garcia-Beltran - marzo, 2002
 */

public class PruebaSobrecarga {
   public static void main (String[] args) {
     int a=34;
```

```
     int b=12;
     int c=56;
     System.out.println("a = " + a + "; b = " + b + "; c = " + c);
     System.out.println("El mayor de a y b es: " + mayor(a,b);  // El primero
     System.out.println("El mayor de a, b y c es: " + mayor(a,b,c));  // El 2º
     }
   // Definicion de mayor de dos numeros enteros
   public static int mayor (int x, int y) {
     return x>y ? x : y;
     }
   // Definicion de mayor de tres numeros enteros
   public static int mayor (int x, int y, int z) {
     return mayor(mayor(x,y),z);
     }
}
```

Ejemplo de salida por pantalla:

```
$>java PruebaSobrecarga↵
a = 34; b = 12; c = 56
El mayor de a y b es: 34
El mayor de a, b y c es: 56
```