Gobierno de Canarias
Servicio Canario de Empleo

Centro GASTRONÓMICO del Atlántico

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Módulo 1: Programación en Java 2023

Tema 6 Métodos y Sentencia Return

Tema 6

Contenido

1. Métodos y Sentencia Return

1.1. Objetivos

- Describir el funcionamiento de la sentencia return.
- Interpretar el resultado de una sentencia return en el código fuente de una aplicación Java.
- Codificar una tarea sencilla convenientemente especificada utilizando la sentencia return.

1.2. 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**.

La sintaxis de la sentencia **return** es la siguiente:

```
return expresion;
```

1.3. 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.

La sintaxis de declaración de un método es la siguiente:

```
[modificadores] tipoDeDato identificadorMetodo (lista de parametros) {
declaraciones de variables locales;
sentencia_1;
sentencia_2;
...
sentencia_n;
// Si el método devuelve algún valor debe indicarse la sentencia return
return valor;
// dentro de estas sentencias se incluye al menos un return
}
```

La primera línea de código corresponde a la cabecera del método. Los *modificadores* especifican como 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.

Las sentencias entre llaves componen el cuerpo del *método*. Dentro del cuerpo del *método* se localiza, al menos, una sentencia *return*.

A continuación, 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
**/
public class CalculoCubo {
        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) {
        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 statu** 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.

Ejemplo de ejecución del código anterior y salida correspondiente por pantalla:

```
$ java CalculoCubo
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 crear un objeto para poder utilizarlos se denominan **métodos estáticos** o **métodos 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.

Nota: Todo programa o aplicación Java 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.

En el siguiente ejemplo 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*.

Nota: 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
**/
public class CalculoFactorial {
	public static void main (String [ ] args) {
		System.out.println("El factorial de 10 es: " + factorial(10));
	}
	public static int factorial (int n) {
		int aux = 1;
		for (int i = 2; i <= n; i++) {
			aux *= i;
		}
		return aux;
	}
}
```

Ejemplo de ejecución y salida correspondiente por pantalla:

```
$ java CalculoFactorial
El factorial de 10 es: 3628800
```

1.4. Uso de 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
*/
public class CalculoProducto {
        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 CalculoProducto
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
```

1.5. 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 hace falta ponerla, y si se pone, no viene seguida de ninguna expresión:

```
return;
```

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

```
/**
* Demostracion del metodo tabla
*/
public class PruebaTabla {
	public static void main (String [ ] args) {
		tabla(4);
		tabla(7);
	}
	public static void tabla (int n) {
		for (int i = 0; i <= 10; i++) {
			System.out.println(n + " x " + i + " = " + producto(n,i));
			return; // No hace falta ponerlo
		}
	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 número 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 número 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 ejecuta la sentencia posterior desde la que el método fue invocado.

1.6. 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 diferente 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 solamente. El compilador se encarga de averiguar cuál de los métodos que comparten identificador debe ejecutar.

El siguiente ejemplo calcula el mayor de tres números utilizando sobrecarga:

```
/**
* Demostracion de metodos sobrecargados
*/
public class EjemploSobrecarga {
        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));
                System.out.println("El mayor de a, b y c es: " + mayor(a,b,c));
        }
        // Definicion de mayor de dos numeros enteros
        public static int mayor (int x, int y){
                if (x > y)
                        return x;
                else
                        return 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 EjemploSobrecarga
a = 34; b = 12; c = 56
El mayor de a y b es: 34 El mayor de a, b y c es: 56
```

2. Bibliografía

- Material suministrado por CGA.
- https://javadesdecero.es/poo/metodos-con-ejemplos/
- https://javabasico.osmosislatina.com/curso/progbasico2/return.htm
- https://www.arkaitzgarro.com/java/capitulo-14.html