1.7 Introducción a funciones

1. Introducción

Una subrutina o subprograma, como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica.

Podemos distinguir tres términos que poseen diferencias:

1. **Función**: conjunto de instrucciones que devuelven un resultado.
2. **Procedimiento**: conjunto de instrucciones que se ejecutan sin devolver ningún resultado.
3. **Método**: función o procedimiento que pertenece a un objeto.

2. Construcción

Una función se construye de la siguiente manera:

```
modificador_acceso tipo_resultado nombre_función (tipo_parámetro
nombre_parámetro, ... ) {
   instrucciones
   return expresión;
}
```

- **modificador_acceso**: es la visibilidad que posee la función (Lo veremos más adelante. De momento, lo utilizaremos como public).
- **tipo_resultado**: es el tipo del resultado que devuelve la función.
- **nombre_función**: es el nombre que identifica a la función. Utiliza la notación lowerCamelCase. Ejemplo: imprimirResultadoDecimal.
- **tipo_parámetro nombre_parámetro, ...**: puede ocurrir que la función necesite ciertos valores para efectuar la misión para la que ha sido creada. Por ejemplo, la función suma necesitaría los valores que tiene que sumar. En este caso, se deben indicar cada uno de dichos valores con sus tipos correspondientes. A estos valores se les conoce como parámetros de la función. Si la función no necesita parámetros, entonces solamente se ponen los paréntesis: `nombre_funcion( )`

A todo esto se le conoce como la **firma** (signature) de la función:

```
modificador_acceso tipo_resultado nombre_función (tipo_parámetro
nombre_parámetro, ... )
```

- **instrucciones**: instrucciones que conforman el algoritmo de la función, para que realice la misión para la que ha sido creada.
- **return expresión;** : el return es el que nos devuelve el resultado, por lo tanto la expresión que acompaña al return tiene que devolver un valor correspondiente al *tipo_resultado* indicado en la firma de la función.

Ejemplo de función con dos parámetros:

```
public static int add(int sum1,int sum2) {
    return sum1+sum2;
}
```

Puede ser también que haya más de un return. En ese caso, el flujo de ejecución abandona la función en cuanto ejecute el primer return. Ejemplo:

```
public static boolean isPar(int n){
    if(n%2==0){
        return true;
    }
    else{
        return false;
    }
}
```

En el caso de que estemos definiendo un **procedimiento**, no tendremos *return* ya que no devuelve ningún resultado y el tipo_resultado es *void*. Como por ejemplo `System.out.println`, que escribe en pantalla lo que recibe por parámetro pero no devuelve nada.

3. Llamada a la función

Una función permite que reutilicemos un algoritmo ya que se puede utilizar cuando nos haga falta. Para ello, solamente tendremos que llamar a la función por su nombre y pasarle los parámetros en el mismo orden que se han definido y pertenecientes al mismo tipo de dato. En la llamada, dichos parámetros se llaman **argumentos**.

```
package tema1_7_IntroduccionAFunciones;

public class Functions {

    public static void main(String[] args) {

        boolean par;
        int result;

        par = isPar(5);//Se llama a la función isPar con un valor de 5 en el
argumento
        System.out.println(par);//Mostrará false
        par = isPar(4);//Se llama a la función isPar con un valor de 4 en el
argumento
        System.out.println(par);//Mostrará true

        result = add(5, 2);//Se llama a la función add con los valores 5 y 2
en los argumentos
        System.out.println(result);//Mostrará 7

    }

    public static boolean isPar(int n) {

        if (n % 2 == 0) {
            return true;
        } else {
            return false;
        }

    }
```

```
    public static int add(int sum1, int sum2) {

        return sum1 + sum2;
    }

}
```