# Rutinas

Las rutinas son uno de los recursos más valiosos cuando se trabaja en programación ya que permiten que los programas sean más simples, debido a que el programa principal se compone de diferentes rutinas donde cada una de ellas realiza una tarea determinada.

Una rutina se define como un bloque, formado por un conjunto de instrucciones que realizan una tarea específica y a la cual se la puede llamar desde cualquier parte del programa principal. Además, una rutina puede opcionalmente tener un valor de retorno y parámetros. El valor de retorno puede entenderse como el resultado de las instrucciones llevadas a cabo por la rutina, por ejemplo si para una rutina llamada sumar(a, b) podríamos esperar que su valor de retorno sea la suma de los números a y b.

En el caso anterior, a y b son los datos de entrada de la rutina necesarios para realizar los cálculos correspondientes. A estos datos de entrada los denominamos parámetros y a las rutinas que reciben parámetros las denominamos funciones, procedimientos o métodos, dependiendo del lenguaje de programación.

Ejemplos de rutinas:

*SumarPrecioProductos(precioProducto1, precioProducto2)*

Rutina que realiza la suma de los precios de los productos comprados por un cliente y devuelve el monto total conseguido.

*AplicarDescuento(montoTotal)*

Rutina que a partir de un monto total aplica un descuento de 10% y devuelve el monto total con el descuento aplicado.

Entonces nuestro programa puede hacer uso de dichas rutinas cuando lo necesite. Por ejemplo, cuando un cliente realice una compra determinada, podemos llamar a la rutina sumarPrecioProductos y cobrarle el monto devuelto por la misma. En el caso que el cliente abonara con un cupón de descuento, podemos entonces llamar a la rutina aplicarDescuento y así obtener el nuevo monto con el 10% aplicado.

¿Funciones, métodos o procedimientos?

Es muy común entre programadores que se hable indistintamente de estos tres términos sin embargo poseen deferencias fundamentales.

**Funciones**:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente retornan un valor. En otras palabras, una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y retorna un valor usando la instrucción *return*, si no

retorna algo, entonces no es una función. En java las funciones usan el modificador *static*.

**Métodos**:

Los métodos y las funciones en Java están en capacidad de realizar las mismas tareas, es decir, son funcionalmente idénticos, pero su diferencia radica en la manera en que hacemos uso de uno u otro (el contexto). Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo, en método está asociado a un objeto, SIEMPRE, básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada. Nota: Es aquí donde digo que en Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función esta debe ser *static*, para que no requiera de un objeto para ser llamada.

**Procedimientos**:

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo, en la definición no hay nada que se lo impida. En el contexto de Java un procedimiento es básicamente un método cuyo tipo de retorno es *void* que no nos obliga a utilizar una sentencia *return*.

**Crear un método en Java**

La sintaxis para declarar una función es muy simple, veamos:

```
   [acceso] [modificador] tipo nombreFuncion([tipo nombreArgumento,[tipo
nombreArgumento]...])
{
    /*
        * Bloque de instrucciones
    */

    return valor;
}
```

EL primer componente corresponde al modificador de acceso, que puede ser *public* o *private*, éste es opcional, si no ponemos nada, se asume el modificador de acceso por defecto, el segundo componente es el modificador que puede ser *final* o *static* (o ambas), también es opcional. Recordemos que un método o función siempre retorna algo, por lo tanto, es obligatorio declararle un tipo (el tercer componente de la sintaxis anterior), puede ser entero (int), booleano (boolean), o cualquiera que consideremos, inclusive tipos complejos, luego debemos darle un nombre a dicha función, para poder identificarla y llamarla (invocarla) durante la ejecución, después al interior de paréntesis, podemos poner los argumentos o parámetros. Luego de la definición de la "firma" del método, se define su funcionamiento entre llaves; todo lo que esté dentro de las llaves es parte del cuerpo del método y éste se ejecuta hasta llegar a una instrucción *return*.

**Acerca de los argumentos o parámetros**

Hay algunos detalles respecto a los argumentos de un método, veamos:

- Una función, un método o un procedimiento pueden tener una cantidad cualquier de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más parámetros. Aunque habitualmente no suelen tener más de 4 o 5.
- Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.
- Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo del método.
- Al recibir un argumento nada nos obliga a hacer uso de éste al interior del método, sin embargo para qué recibirlo si no lo vamos a usar.
- En Java los parámetros que podemos recibir pueden ser por valor por referencia, esto implica que, si modificamos los valores recibidos al interior del método, estos pueden mantener sus cambios o no después de ejecutada el método (esto lo explico con más detalla enseguida).

**Consejos acerca de return**

Debes tener en cuenta dos cosas importantes con la sentencia return:

- Cualquier instrucción que se encuentre después de la ejecución de return NO será ejecutada. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.
- El tipo del valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.
- En el caso de los procedimientos (void) podemos usar la sentencia return pero sin ningún tipo de valor, sólo la usaríamos como una manera de terminar la ejecución del procedimiento.

**Ejemplos de métodos**

Ejemplo 1:

```
int metodoEntero()//Función sin parámetros
{
    int suma = 5+5;
    return suma; //Acá termina la ejecución del método
    //return 5+5;//Este return nunca se ejecutará
    //Intenta intercambiar la línea 3 con la 5
    //int x = 10; //Esta línea nunca se ejecutará
}
```

Ejemplo 2:

```
public String metodoString(int n)//método con un parámetro
{
```

```
    if(n == 0)//Usamos el parámetro en la función
    {
        return "a"; //Si n es cero retorna a
        //Notar que de aquí para abajo no se ejecuta nada más
    }
    return "x";//Este return sólo se ejecuta cuando n NO es cero
}
```

Aquí creamos un método público, hicimos uso de múltiples sentencia return y aprovechamos la característica de que al ser ejecutadas finalizan inmediatamente la ejecución de la parte restante del método. De este modo podemos asegurar que la función retornará "a" únicamente cuando el valor del parámetro n sea cero y retornará un "x" cuando dicho valor no sea cero.

Ejemplo 3:

```
static boolean metodoBoolean(boolean n, String mensaje)//Método con dos
parámetros
{
    if(n)//Usamos el parámetro en el método
    {
        System.out.println(mensaje);//Mostramos el mensaje
    }
    return n; //Usamos el parámetro como valor a retornar
}
```

Aquí ya tenemos una función (digo función y no método porque es static) que recibe dos parámetros, uno de ellos es usado en el condicional y el otro para mostrar su valor por pantalla con System.out.println, esta vez retornamos valores booleanos true o false y utilizamos el valor propio recibido en el parámetro. Toma en cuenta que en esta ocasión únicamente usamos una sentencia return, pues usar una al interior del if habría sido innecesario y el resultado sería el mismo.

**Los procedimientos**

Los procedimientos son similares a las funciones, aunque más resumidos. Debido a que los procedimientos no retornan valores, no hacen uso de la sentencia return para devolver valores y no tienen tipo específico, sólo void. Veamos un ejemplo:

**Ejemplo de procedimientos**

```
  void procedimiento(int n, String nombre) //Notar el void
{
    if(n > 0 && !nombre.equals(""))//usamos los dos parámetros
    {
        System.out.println("hola " + nombre);
        return; //Si no ponemos este return se mostraría hola y luego
adiós
```

```
    }
    //También podríamos usar un else en vez del return
    System.out.println("adios");
}
```

De este ejemplo podemos ver que ya no se usa un tipo sino que se pone void, indicando que no retorna valores, también podemos ver que un procedimiento también puede recibir parámetros o argumentos.

Los procedimientos también pueden usar la sentencia return, pero no con un valor. En los procedimientos el return sólo se utiliza para finalizar allí la ejecución.

**Invocando funciones y procedimientos en Java**

Ya hemos visto cómo hacer funciones en Java, cómo se crean y cómo se ejecutan, ahora veamos cómo usar un método, función o procedimiento.

```
nombre([valor,[valor]...]);
```

Como puedes notar es bastante sencillo invocar o llamar funciones en Java, sólo necesitas el nombre del método, función o procedimiento y enviarle el valor de los parámetros. Hay que hacer algunas salvedades respecto a esto.

**Detalles para invocar métodos funciones y procedimientos**

- No importa si se trata de un método en Java o de una función o de un método, sólo debes ocuparte de enviar los parámetros de la forma correcta para invocarlos.
- El nombre debe coincidir exactamente al momento de invocar, pues es la única forma de identificarlo.
- El orden de los parámetros y el tipo debe coincidir. Hay que ser cuidadosos al momento de enviar los parámetros, debemos hacerlo en el mismo orden en el que fueron declarados y deben ser del mismo tipo (número, texto u otros).
- Cada parámetro enviado también va separado por comas.
- Si una función no recibe parámetros, simplemente no ponemos nada al interior de los paréntesis, pero SIEMPRE debemos poner los paréntesis.
- Invocar una función sigue siendo una sentencia común y corriente en Java, así que ésta debe finalizar con ';' como siempre.
- El valor retornado por un método o función puede ser asignado a una variable del mismo tipo, pero no podemos hacer esto con un procedimiento, pues no retornan valor alguno.
- Una función puede llamar a otra dentro de sí misma o incluso puede ser enviada como parámetro a otra (mira el siguiente ejemplo).

**Ejemplos de uso de funciones**

En el siguiente código vamos a hacer un llamado a algunas de las funciones y al procedimiento, que declaramos anteriormente.

```
public class Ejemplos
{
public static void main(String args[])//Siempre necesitamos un main
{
Ejemplos ejemplo = new Ejemplos(); //Cuando no es estático, debe usarse un
método

//Llamando a un método sin argumentos, usando el objeto
ejemplo.metodoEntero();

//Asignando el valor retornado a una variable boolean respuesta =
metodoBoolean(true, "hola");

// El procedimiento no es static, así que debe llamarse desde el
objeto.
ejemplo.procedimiento(0, "Juan");//Invocando el procedimiento

//Usando una función como parámetro ejemplo.procedimiento(metodoBoolean(1,
"hola"), "Juan");
//Lo que retorne metodoBoolean (en este caso 1) se envía al procedimiento
}
}
```

**Ejercicios resueltos de métodos en Java**

Vamos a realizar un par de ejercicios sobre métodos y los vamos a resolver y explicar.

**Funciones anidadas**

Para este ejercicio vamos a crear una función que llama a otra al interior de ella (por eso las quise llamar anidadas). Es un sistema de validación de un usuario que recibe un usuario y una contraseña y según sean válidos o no, muestra un mensaje al usuario.

```
public class Ejercicios
{
public static String saludar(String nombre)
{
//Se crea el mensaje de saludo
String saludo = "Hola. Bienvenido " + nombre;

return saludo;//Se retorna el saludo
}

public static String error(String nombre)
{
```

```
        //Se crea el mensaje de error
        String error = "Ups. No pudimos validar tus datos. " + nombre + "
es tu usuario?";

        return error; //Se retorna el error
    }

    public static void verificar(String usuario, String contrasenia)
    {
        String usuarioValido = "JuanDMeGon";

        String contraseniaValida = "MiPass";

        //Se validan los datos
        if(usuarioValido.equals(usuario) &&
contraseniaValida.equals(contrasenia))
        {
            //Si son validos se llama ala función saludar y se muestra el
mensaje retornado por pantalla
            System.out.println(saludar(usuario));
            return; //Terminamos la ejecución
        }
        //Si no son válidos entonces mostramos el mensaje de error de la
funcion error.
        System.out.println(error(usuario));
    }

    public static void main(String[] args)
    {
        String usuario = "Juan";
        String contrasenia = "pass";

        //Se hace la verificación
        verificar(usuario, contrasenia);

        //Mostrará el mensaje error.
    }
```

En este ejercicio tenemos dos funciones públicas (funciones porque usamos el modificador static) y un procedimiento (porque es void) que hace la validación (muy muy básica) de unos datos de usuario y según estos, llama a una función de saludo o de error. Debes notar que el procedimiento es llamado desde la función main.

Fuente: Apunte Técnicas de Programación v 1.3.Corregido.pdf

https://www.programarya.com/Cursos/Java