14. Funciones (I)

14.1 Introducción

En programación es muy frecuente reutilizar código, es decir, usar código ya existente. Cuando una parte de un programa requiere una funcionalidad que ya está implementada en otro programa no tiene mucho sentido emplear tiempo y energía en implementarla otra vez.

Una función es un conjunto de código que realiza una tarea muy concreta y que se puede incluir en cualquier programa cuando hace falta resolver esa tarea.

Opcionalmente, las funciones aceptan una entrada (parámetros de entrada) y devuelven una salida.

```java
/** Calcula el fatorial de un número entero positivo
 *
 * @param numero Número del que se desea calcular el factorial
 * @return Factorial del número introducido
 */

static public int factorial(int numero) {

    int fact = 1;
    if (numero >= 0) {

        for (int i=1; i<=numero; i++) {
            fact *= i;
        }

    }
    return fact;
}
static public void main(String args[]) {

    System.out.println(factorial(0));
    System.out.println(factorial(3)+factorial(4));
}
```

```
static public int factorial(int numero) {
```

- static quiere decir que pertenece a la clase, no a los objetos (se verá más adelante).
- public quiere decir que podrá ser invocado desde esta clase o desde cualquier otra.
- int tipo que devuelve la función.
- factorial Nombre de la función
(int numero) – tipo que se le pasa a la función y nombre que va a tener como variable dentro de la función, solo dentro.

Otro ejemplo:

```
/**
 * Calcula si un número es múltiplo de otro
 *
 * @param numero Número a verificar si es divisible
 * @param divisor Número entero sobre el que se prueba la divisivilidad
 * @return Devuelve si es divisible numero entre divisor
 */
static boolean esMultiploDe(int numero, int divisor) {
    return numero % divisor == 0;
}
```

14.2 Llamando a funciones

Vamos a resolver ahora un problema usando las funciones. Queremos que, dado un número, nos diga si el factorial de ese número más el factorial de los dos siguientes, es múltiplo de 3:

```java
static public void main(String args[]) {

    // Programa que calcula dado un número, si la suma del factorial de ese número y
    // el factorial de los dos siguientes es múltiplo de 3.

    Scanner s = new Scanner(System.in);
    System.out.print("Introduzca un número: ");
    int entrada = Integer.parseInt(s.nextLine());
    int suma = factorial(entrada)+factorial(entrada+1)+factorial(entrada+2);

    System.out.print("La suma de los factoriales da " + suma + " que ");
    if (!esMultiploDe(suma,3)) {
        System.out.print("no ");
    }

    System.out.println("es múltiplo de 3");

}
```

```
Introduzca un número: 2
La suma de los factoriales da 32 que no es múltiplo de 3
```

Una función puede no devolver ningún valor, ello se indica con void:

```java
/**
 * Dibuja un número indicado de asteriscos por consola
 *
 * @param numero Número de asteriscos a dibujar.
 */
static public void dibujaAsteriscos(int numero) {

    for (int i=0; i<numero;i++) {
        System.out.print("*");
    }

    System.out.println();
}
```

14.3 Argumentos por valor y referencia

En la mayoría de lenguajes se pueden pasar los argumentos por:

Valor – Se realiza una copia de la variable que se le pasa a la función, conservando el valor la variable que se pasa como parámetro fuera de la función, que permanece inalterada.

Referencia – La variable dentro y fuera es la misma. Cuando se modifica la variable dentro de la función se está cambiando fuera también.

```
static public void main(String args[]) {

    int fuera = 3;

    System.out.println("Fuera vale:" + fuera);

    doble(fuera);

    System.out.println("Fuera vale:" + fuera);
}

public static void doble(int dentro) {

    dentro = dentro * 2;
    System.out.println("Dentro vale:" + dentro);

}
```

```
Fuera vale:3
Dentro vale:6
Fuera vale:3
```

El anterior código es incorrecto, porque la variable a la que se pretende doblar su valor, en realidad se está cambiando el valor sólo a la copia local.

```java
static public void main(String args[]) {

    int fuera = 3;

    System.out.println("Fuera vale:" + fuera);

    fuera = doble(fuera);

    System.out.println("Fuera vale:" + fuera);
}

public static int  doble(int dentro) {

    dentro = dentro * 2;
    System.out.println("Dentro vale:" + dentro);
    return dentro;

}
```

Aquí está el código corregido, teniendo en cuenta que se pasa por valor.

```
Fuera vale:3
Dentro vale:6
Fuera vale:6
```

Puedes probar a llamar igual a la variable externa y al argumento, no existe conflicto, al entenderse con ese mismo nombre la copia dentro de la función.

En Java sólo se pasa por valor, pero hay que tener en cuenta matices, cuando creamos un array lo que nos guarda en la variable es una referencia al objeto. Cuando se realiza la copia se copia la misma referencia, por lo que se apunta al mismo objeto. El comportamiento es por tanto como si se pasase por referencia.

```java
/**
 * Muestra el contenido de un array de enteros
 * @param a Array a mostrar
 */
    public static void muestraArray(int a[]) {
        for(int i=0;i<a.length;i++) {
            System.out.print(a[i]);
        }
        System.out.println();

    }

    /**
     * Multiplica un array de enteros por dos
     * @param dentro array a convertir
     */
    public static void  doble(int []dentro) {

        for (int i=0;i<dentro.length;i++) {
            dentro[i] *=2;
        }
        System.out.println("Dentro vale:");
        muestraArray(dentro);


    }
```

```java
static public void main(String args[]) {

    int fuera[]= {1,2,3,4,5};


    System.out.println("Fuera vale:" );
    muestraArray(fuera);

    doble(fuera);

    System.out.println("Fuera vale:" );
    muestraArray(fuera);
}
```

```
Fuera vale:
12345
Dentro vale:
246810
Fuera vale:
246810
```

A pesar de este hecho, por regla general, no cambiaremos el array que se pasa como argumento, y devolveremos el array que se quiere obtener:

```java
/**
 * Devuelve un array de enteros por dos
 * @param dentro array a convertir
 */
public static int []  doble(int []array) {
// Se crea un array del mismo tamaño que el de origen.
    int [] arrayDoble = new int[array.length];

    for (int i=0;i<array.length;i++) {
        arrayDoble[i] =array[i] * 2;
    }
    return arrayDoble;


}
```

```java
int fuera[]= {1,2,3,4,5};
muestraArray(fuera);
fuera = doble(fuera);
muestraArray(fuera);
```

14.4 Creando paquetes y organizando el código.

Una función puede ser utilizada en varios programas diferentes, pero no es mantenible el copiar y pegar las funciones, ya que si quisiésemos hacer un cambio habría que realizarlo en todos los programas a la vez.

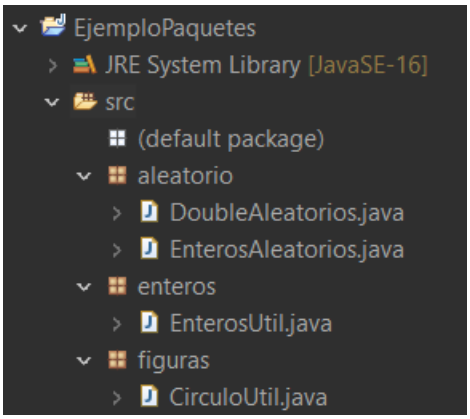
Las funciones se pueden agrupar en paquetes (package), organizados a nivel semántico. Los paquetes luego se importarán (import) desde el programa que necesite esas funciones.

Cada paquete se corresponde con un directorio (puedes comprobarlo en el espacio de trabajo y en el sistema de ficheros). Por tanto, si hay un paquete con nombre aleatorios debe haber un directorio llamado también aleatorios en la misma ubicación.

Definimos un paquete figuras, con utilidades de figuras geométricas.

```java
package figuras;

public class CirculoUtil {

    static public double area(double radio) {
        return Math.PI * radio * radio;

    }

    static public double perimetro(double radio) {
        return Math.PI * 2 * radio;
    }


}
```

Agrupamos las clases en una estructura de paquetes coherente a nivel semántico.



Para utilizarla simplemente importamos con import:

```java
import java.util.Scanner;

import figuras.CirculoUtil;

public class Ejemplo {

    static public void main(String args[]) {

        double radio;
        double area;
        double perimetro;
        Scanner s = new Scanner(System.in);

        System.out.println("Introduzca el radio: ");

        radio = Double.parseDouble(s.nextLine());
        area = CirculoUtil.area(radio);
        perimetro = CirculoUtil.perimetro(radio);

        System.out.println("El area es " + area+ " y el perímetro es "+ perimetro);

    }

}
```

Ejercicio 14.1

Convierte en función el ejercicio 8.5, que devuelva el signo zodiacal en función de una fecha.

Ejercicio 14.2

Convierte en función el ejercicio 8.4, haciendo que la función devuelva el resultado de una ecuación de segundo grado.

Ejercicio 14.3

Convierte en función el ejercicio 9.12, haciendo que la función nos devuelva si un entero es primo o no como valor booleano.

Ejercicio 14.4

Convierte en función el ejercicio 9.20, devolviendo un valor booleano diciendo si el número es capicúa o no.

Ejercicio 14.5

Convierte en función el ejercicio 9.21, devolviendo el número con el dígito insertado.

Ejercicio 14.6

Crea una función:

- int generaAleatorio(int min, int max), que genere un número entero aleatorio entre min y max.

Ejercicio 14.7

Realiza una función que pinte la letra L por pantalla hecha con asteriscos. A la función se le pasará la altura como parámetro. El palo horizontal de la L tendrá una longitud de la mitad (división entera entre 2) de la altura más uno. (Ejercicio 9.17)

Ejercicio 14.8

Crea una biblioteca de funciones matemáticas que contenga las siguientes funciones. Recuerda que puedes usar unas dentro de otras si es necesario.

Observa bien lo que hace cada función ya que, si las implementas en el orden adecuado, te puedes ahorrar mucho trabajo. Por ejemplo, la función esCapicua resulta trivial teniendo voltea y la función siguientePrimo también es muy fácil de implementar teniendo esPrimo. Se permite hacer “trampa” con los métodos de String.

1. esCapicua: Devuelve verdadero si el número que se pasa como paráme#tro es capicúa y falso en caso contrario.

2. esPrimo: Devuelve verdadero si el número que se pasa como parámetro es primo y falso en caso contrario.

3. siguientePrimo: Devuelve el menor primo que es mayor al número que se pasa como parámetro.

4. potencia: Dada una base y un exponente devuelve la potencia.

5. digitos: Cuenta el número de dígitos de un número entero.

6. voltea: Le da la vuelta a un número.

7. digitoN: Devuelve el dígito que está en la posición n de un número entero. Se empieza contando por el 0 y de izquierda a derecha.

8. posicionDeDigito: Da la posición de la primera ocurrencia de un dígito dentro de un número entero. Si no se encuentra, devuelve -1.

9. quitaPorDetras: Le quita a un número n dígitos por detrás (por la derecha).

10. quitaPorDelante: Le quita a un número n dígitos por delante (por la izquierda).

11. pegaPorDetras: Añade un dígito a un número por detrás.

12. pegaPorDelante: Añade un dígito a un número por delante.

13. trozoDeNumero: Toma como parámetros las posiciones inicial y final

dentro de un número y devuelve el trozo correspondiente.

14. juntaNumeros: Pega dos números para formar uno

Ejercicio 14.9

Muestra los números primos que hay entre 1 y 1000.

Ejercicio 14.10

Muestra los números capicúa que hay entre 1 y 99999.

Ejercicio 14.11

Crea una biblioteca de funciones para arrays (de una dimensión) de números enteros que contenga las siguientes funciones:

1. generaArrayInt: Genera un array de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.

2. minimoArrayInt: Devuelve el mínimo del array que se pasa como parámetro.

3. maximoArrayInt: Devuelve el máximo del array que se pasa como parámetro.

4. mediaArrayInt: Devuelve la media del array que se pasa como parámetro.

5. estaEnArrayInt: Dice si un número está o no dentro de un array.

6. posicionEnArray: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra.

7. volteaArrayInt: Le da la vuelta a un array.

8. rotaDerechaArrayInt: Rota n posiciones a la derecha los números de un array.

9. rotaIzquierdaArrayInt: Rota n posiciones a la izquierda los números de un array.

Ejercicio 14.12

Crea una biblioteca de funciones para arrays bidimensionales (de dos dimensiones) de números enteros que contenga las siguientes funciones:

1. generaArrayBiInt: Genera un array de tamaño n x m con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.

2. filaDeArrayBiInt: Devuelve la fila i-ésima del array que se pasa como parámetro.

3. columnaDeArrayBiInt: Devuelve la columna j-ésima del array que se pasa como parámetro.

4. coordenadasEnArrayBiInt: Devuelve la fila y la columna (en un array con dos elementos) de la primera ocurrencia de un número dentro de un array bidimensional. Si el número no se encuentra en el array, la función devuelve el array {-1, -1}.

5. diagonal: Devuelve un array que contiene una de las diagonales del array bidimensional que se pasa como parámetro. Se pasan como parámetros fila, columna y dirección. La fila y la columna determinan el número que marcará las dos posibles diagonales dentro del array. La dirección es un booleano. Si es true indica que se elige la diagonal que va del noroeste hacia el sureste, mientras que si es false indica que se elige la diagonal que va del noreste hacia el suroeste.