

PROGRAMACIÓN

UP4. PROGRAMACIÓN MODULAR



1º CFGS DAW

Curso 2024-25

Estructura de un programa Java

Estructura de un programa Java

En muchas ocasiones, se tiene que usar un mismo bloque de código para resolver un mismo problema con datos diferentes.

Por ejemplo, viniendo del tema anterior, si miramos los ejercicios realizados, nos encontramos con que tenemos mucho código repetido cada vez que necesitábamos imprimir una matriz o vector. Escribir el mismo código varias veces es un trabajo repetitivo, improductivo, y difícil de mantener y modificar. Por todo ello, este tipo de código duplicado tiene una gran tendencia a provocar errores.

La mejor forma de crear y mantener un programa grande es construirlo a partir de piezas más pequeñas (subprogramas), donde cada una de estas piezas es más manejable que el "monstruo" del programa. En Java, la forma de trocear los programas es usando métodos.

Los métodos son utilizados para evitar la repetición de un bloque de código en un mismo programa. Se pueden ejecutar desde varios puntos de un proyecto, con el fin de reutilizar el código y que no haya duplicidades.

Un método es un bloque de código que:

- Se usa (se invoca su ejecución) desde algún punto del código del programa principal para resolver un problema dado.
- Este método puede emplear datos externos (parámetros).

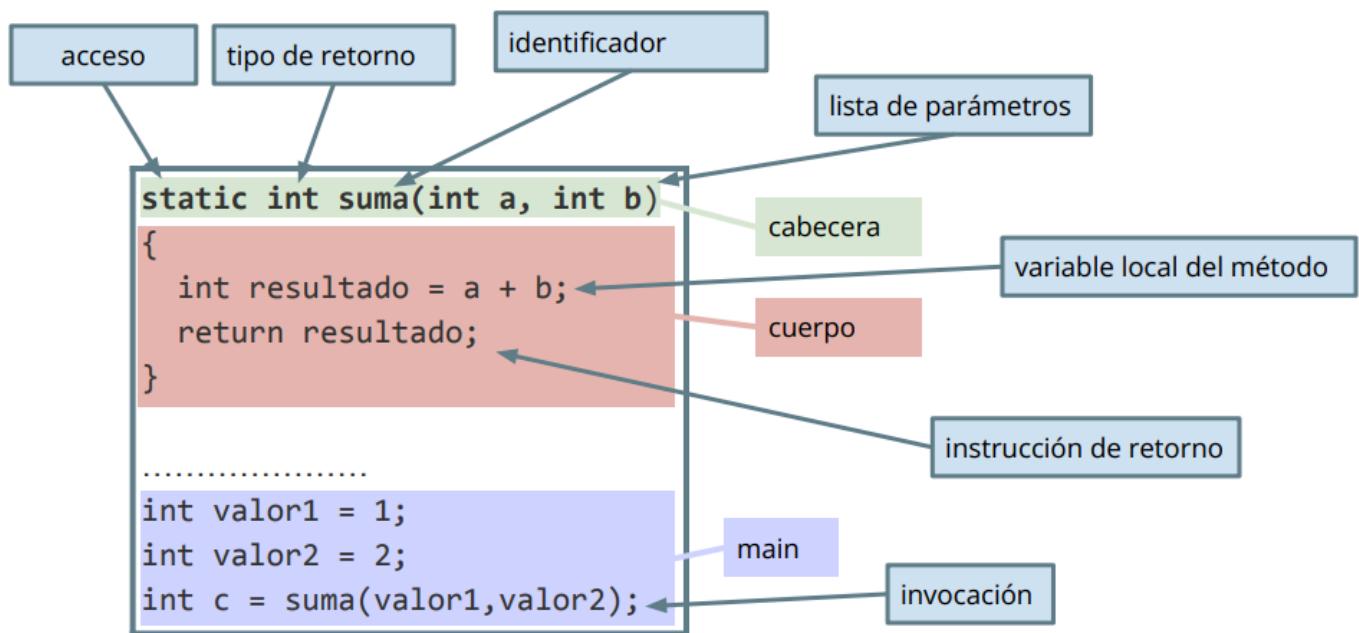
Dichos datos deben ser proporcionados cuando llamamos (invocamos) al método. Si proporcionamos variables como parámetros, siempre se proporcionará una copia de la variable original (paso por valor).

- Devolverá (*return*) un resultado calculado en base a los parámetros que le hayamos proporcionado. Hay métodos que no devuelven nada, simplemente ejecutan el código incluido dentro del método.

4.1. Funciones y procedimientos

Como ya hemos comentado, el método es el mecanismo de subprogramación que ofrece el lenguaje *Java*.

Un método se declara definiendo su cabecera y su cuerpo, dentro de una clase:

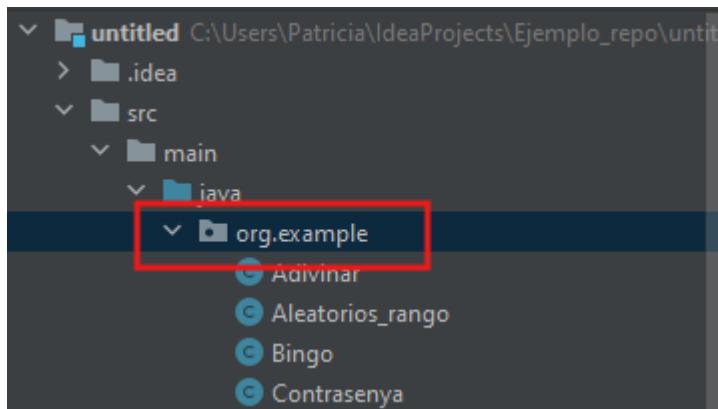


La **cabecera** de un método se define en este orden:

- **Acceso:** modificadores de visibilidad (*public*) y directiva *static*.

(*) Por ahora, aunque es opcional, se usará siempre el modificador *static* para describir el acceso de un método. Los métodos estáticos están asociados a la clase en lugar de una instancia de la clase (objeto). Es decir, que estos métodos definidos como *static* se pueden invocar directamente sin la necesidad de crear un objeto. En temas posteriores se darán más detalles sobre esto.

En cuanto a la **visibilidad** (*public/private*), usaremos siempre *public*. Cuando no se especifica ningún modificador de acceso, *Java* utiliza un nivel de acceso predeterminado, a menudo denominado *package-private*. Esto significa que el miembro sólo es accesible dentro de las clases del mismo paquete.



Es menos restrictivo que *private*, pero más restrictivo que *public*.

- **Tipo de dato** de lo que devuelve. Si no devuelve nada, indicamos *void* como hasta ahora.
- **Nombre** (identificador). El nombre de un método y el de sus parámetros siguen las reglas habituales de los identificadores de las variables, constantes, etc.
- **Lista de parámetros:** tipos y nombres de los datos.

```
static int suma(int a, int b) { ... } /*declaramos un método con
identificador suma y que retorna un valor entero(int) */
```

Se pueden declarar métodos sin parámetros. En este caso, se utilizarán paréntesis vacíos después del identificador:

```
static void mostrarMenu( ) { ... }
```

El **cuerpo** es la secuencia de instrucciones que se ejecutan al invocarlo. Puede incluir cualquier tipo de instrucción o bloque de instrucciones: asignación, condicional, bucle, e incluso, llamadas o invocaciones a otros métodos.

Aquí se muestra un ejemplo de método en *Java* con su correspondiente invocación:

```
1 | public static int suma(int a, int b){
2 |
3 |     int resultado = a + b;
4 |     return resultado;
5 |
6 | }
```

```
1 | class Main {  
2 |
```

```

3  public static void main(String[] args) {
4
5      int valor1 = 1;
6      int valor2 = 2;
7      int c = suma(valor1,valor2);
8
9  }
10
11 }
```

Otro ejemplo, para estas características:

acceso	tipo de retorno	identificador	lista de parámetros
static	double	areaRectangulo	(double base, double altura)

```

1  public static double areaRectangulo(double base, double altura){
2
3      double area = base*altura/2;
4
5      return area;
6
7 }
```

```

1  public class Main {
2      public static void main(String[] args) {
3
4          double base = 4;
5          double altura = 2;
6          double area = areaRectangulo(4,2);
7          System.out.println("El área para un triángulo de base " + base +
8
9      }
10 }
```

¿Función o procedimiento? Uso de *Return*

Como ya hemos ido comentando, como resultado de su invocación, un método suele devolver unos resultados, denominados valores de retorno. Pero no tiene por qué ser

siempre así. Podría ser que un método solamente se llame para realizar alguna acción sin devolvernos nada (por ejemplo, imprimir una matriz).

El tipo de retorno de un método puede ser cualquier tipo de dato en Java: *int*, *double*, *char*, *String*, *boolean*, etc. (También pueden devolver objetos).

En concreto, en caso de haberlo, el **valor de retorno debe aparecer después de la palabra reservada *return*.** A este tipo de método se les llama **FUNCIONES**.

Además, se deben tener en cuenta las siguientes consideraciones:

- Como ya sabemos del tema anterior, la ejecución de un método finaliza tras la ejecución de cualquiera de los posibles *return* que aparecen en su cuerpo y vuelve al programa principal que ha realizado la invocación.
- Si un método no devuelve ningún valor, entonces el tipo de su resultado es ***void***. En este caso, se ejecuta “*return*” implícitamente como última instrucción del método, aunque nosotros no lo vemos. A este tipo de métodos se les llama **PROCEDIMIENTOS**.

```
1 //FUNCIÓN
2 static double areaCuadrado(double lado){
3     return lado*lado;
4 }
5
6 //FUNCIÓN
7 static double perimetroCuadrado(double lado){
8     return lado*4;
9 }
10
11 //PROCEDIMIENTO
12 static void mostrarMenu(){
13     System.out.println("Esto es un método que muestra un menú");
14     System.out.println("1 - Obtener área");
15     System.out.println("2 - Obtener perímetro");
16 }
```

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

    Scanner teclado = new Scanner(System.in);
    System.out.println("Introduce el valor del lado del cuadrado: ");
    double lado = teclado.nextDouble();

    mostrarMenu();
```

```
8     int opcion = teclado.nextInt();
9
10    if(opcion==1){
11        double area = areaCuadrado(lado);
12        System.out.println("El área es: " + area);
13    } else if (opcion==2) {
14        double perimetro = perimetroCuadrado(lado);
15        System.out.println("El perímetro es: " + perimetro);
16    }else{
17        System.out.println("No se reconoce la acción.");
18    }
19
20 }
```

4.2. Uso de parámetros

Los parámetros definidos en la cabecera de un método, al declararlo, se denominan parámetros formales (no tienen asociado un valor real).

Sintaxis:

tipo1 nomPar1, tipo2 nomPar2 ... tipoN nomParN

Además, a través de sus identificadores, pueden usarse como variables en el cuerpo del método para realizar las operaciones necesarias. Será en el momento de al invocar a un método desde el programa principal cuando se le proporcionen los valores de los datos requeridos para ejecutar su código.

```
int base = 10, altura = 5, lado = 4; String nombre = "nom";
areaRectangulo(base, altura); //invocamos utilizando variables base y altura
areaCuadrado(lado); //invocamos utilizando la variable lado
tablonDeAnuncios(2*1); //invocamos utilizando un valor numérico como parámetro
saludo("nombre"); //invocamos utilizando la palabra nombre como parámetro
saludo(nombre); //invocamos utilizando la variable nombre como parámetro
```

Los parámetros **NO deben volver a declararse durante la ejecución del cuerpo del método.**

Paso por valor en Java

Los parámetros de los métodos siempre se pasan por valor, es decir, pasamos una copia al método, **la variable original nunca se modifica**.

Esto hace a Java más sencillo y menos propenso a errores. Aunque por otro lado, la manipulación de punteros en lenguajes como C y C++ es útil en la programación de sistemas operativos, debido a que es un modelo que está más cerca de la máquina y permite la manipulación de la memoria de una forma mucho más avanzada.

```
public class Ejemplo {
    public static void main (String args[])
    {
        int num = 5;
        int numDoble = doble(num);
    } //numDoble vale 10, pero num vale 5
    static int doble(int a)
    {
        a = a*2;
        return a;
    }
}
```

```
public class Ejemplo {
    public static void main (String args[])
    {
        int num = 5;
        int doble = doble(num);
    } //doble vale 10, pero num vale 5
    static int doble(int num)
    {
        num = num*2;
        return num;
    }
}
```

Si al método le pasamos una variable como parámetro, al método le llega una copia del valor, y al modificar ese valor sobre el parámetro, no modificaremos el valor de la variable usada para invocar al método.

4.2.1. El famoso String[] args en el main de Java

- Cabecera: **public static void main (String[] args)**
 - Su tipo de retorno es **void**, es decir, no retorna ningún valor.
 - Su único parámetro es un vector o **array** de **Strings**. Esta lista se recibe al ejecutar la aplicación con el comando **>java** por consola.
-

Algunos ejemplos:

```
class HolaMundo {  
    public static void main (String args[]) {  
        System.out.print("Hola");  
        for(int i = 0; i < args.length; i++) System.out.print(" " + args[i]);  
        System.out.println(", qué tal estás? ");  
    }  
}  
  
java HolaMundo           // Hola, qué tal estás?  
java HolaMundo Pepe      // Hola Pepe, qué tal estás?  
java HolaMundo Pepe García // Hola Pepe García, qué tal estás?
```

```
1  class HolaMundo {  
2  
3      public static void main (String args[]) {  
4  
5          System.out.print("Hola");  
6  
7          for(int i = 0; i < args.length; i++){  
8              System.out.print(" " + args[i]);  
9          }  
10  
11         System.out.println(", qué tal estás? ");  
12  
13     }  
14  
15 }
```

```
>java HolaMundo.java  
>java HolaMundo.java Pepe  
>java HolaMundo.java Pepe García
```

```
PS C:\Users\Patricia\Downloads> java HolaMundo.java
Hola, qué tal estás?
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe
Hola Pepe, qué tal estás?
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe García
Hola Pepe García, qué tal estás?
PS C:\Users\Patricia\Downloads>
```

```
class HolaMundo {
    public static void main(String[] args) {
        if (args.length > 1) {
            for (int i = 0; i < Integer.parseInt(args[1]); i++)
                System.out.println("Hola " + args[0]);
        }
        else System.out.println("Hola mundo simple");
    }
}
```

```
java HolaMundo                                // Hola mundo simple
java HolaMundo Pepe                            // Hola mundo simple
java HolaMundo Pepe 3                          // Hola Pepe repetido en 3 líneas
java HolaMundo Pepe García                    // ERROR en ejecución
```

```
1  class HolaMundo {
2
3      public static void main(String[] args) {
4
5          if (args.length > 1) {
6
7              for (int i = 0; i < Integer.parseInt(args[1]); i++){
8                  System.out.println("Hola " + args[0]);
9              }
10
11         }else{
12             System.out.println("Hola mundo simple");
13         }
14
15     }
16
17 }
```

```
PS C:\Users\Patricia\Downloads> java HolaMundo.java
Hola mundo simple
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe
Hola mundo simple
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe 3
Hola Pepe
Hola Pepe
Hola Pepe
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe García
Exception in thread "main" java.lang.NumberFormatException: For input string: "García"
        at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
        at java.base/java.lang.Integer.parseInt(Integer.java:588)
        at java.base/java.lang.Integer.parseInt(Integer.java:685)
        at HolaMundo.main(HolaMundo.java:7)
PS C:\Users\Patricia\Downloads>
```

4.2.2. Varargs en Java

Los *varargs* son parámetros de longitud variable en Java. Para poder utilizarlos, se añaden 3 puntos suspensivos después de indicar el tipo de datos del parámetro.

- Internamente se transforma en un *array*.
- Podemos acceder a través de índices a las posiciones.
- Podemos obtener la longitud con el método *.length()*.

```
static void mostrarEnteros(int... nums) {
    System.out.print("Hay " + nums.length + " enteros: ");
    for (int i = 0; i < nums.length; i++) {
        System.out.print(nums[i] + " ");
    }
}

int a = 2, b = 3;
mostrarEnteros();                                // Hay 0 enteros:
mostrarEnteros(1,2,3);                          // Hay 3 enteros: 1 2 3
mostrarEnteros(b,a++,a*b,++b);                  // Hay 4 enteros: 3 2 9 4
```

MAIN

OJO: El método únicamente puede tener un *varargs*, y debe ser siempre el último parámetro.

De hecho, se puede utilizar incluso en el *main*.

Otro ejemplo:

```
class HolaMundo {
    public static void main(String... args) {
        if (args.length > 0) {
            System.out.println("Hola mundo con " + args.length + " args");
            for (int i = 0; i < args.length; i++)
                System.out.println("arg[" + i + "] = " + args[i]);
        } else System.out.println("Hola mundo sin args");
    }
}
```

```
java HolaMundo          // Hola mundo sin args
```

```
java HolaMundo Pepe García // Hola mundo con 2 args
                           // args[0] = Pepe
                           // args[1] = García
```

```
1 class HolaMundo {  
2  
3     public static void main(String... args) {  
4         if (args.length > 0) {  
5  
6             System.out.println("Hola mundo con " + args.length + " args")  
7  
8             for (int i = 0; i < args.length; i++) {  
9                 System.out.println("arg[" + i + "] = " + args[i]);  
10            }  
11        } else {  
12            System.out.println("Hola mundo sin args");  
13        }  
14    }  
15}  
16}  
17}  
18}
```

```
PS C:\Users\Patricia\Downloads> java HolaMundo.java  
Hola mundo sin args  
PS C:\Users\Patricia\Downloads> java HolaMundo.java Pepe García  
Hola mundo con 2 args  
arg[0] = Pepe  
arg[1] = García  
PS C:\Users\Patricia\Downloads>
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<http://creativecommons.org/licenses/by-sa/4.0/>

Ambitos y sobrecarga

4.2. Ámbitos de una variable

Como ya hemos ido viendo, en el cuerpo de un método se pueden declarar variables que solamente existen dentro de dicho método y el programa principal no conoce. A estas se les denomina **variables locales**. Por lo tanto:

- Las variables locales sólo se pueden usar en el método en el que se declaran, por lo que su ámbito es dicho método.
- Los parámetros de un método se pueden considerar variables locales del mismo.

Por otro lado, en una clase, se pueden declarar **variables globales** en cualquier punto fuera del cuerpo de un método (dentro del ‘class’). De momento, para crear estas variables globales utilizaremos la directiva ‘**static**’. Más adelante, en POO, ya veremos en detalle el significado real del ‘**static**’ y todas sus implicaciones.

- Los métodos de una clase pueden usar las variables globales de esta.

Ejemplos:

```
public class Ejemplo {  
    static int x; //es una variable global  
    static int cubo (int z)  
    {  
        int x; //variable local del método cubo  
        x = z*z*z; //x es la variable local, no la global  
        return x;  
    }  
    public static void main (String args[])  
    {  
        int p; //variable local del método main  
        p = 10; //correcto: p es local en este ámbito  
        z = 100; //error: z no está en este ámbito  
        x = cubo(p); //correcto: x es global, x vale 1000  
    }  
}
```

```

public class Ejemplo {
    static int num = 5; //es una variable global
    static int cubo (int num)
    {
        int x = num*num*num; //num es la variable local del método cubo
        return x;
    }
    public static void main (String args[])
    {
        num = cubo(num-1); // se utiliza la variable global → en num se guarda 64
        System.out.println(num); // se imprime 64(variable local)
    }
}

```

El uso de variables globales no es aconsejable a pesar de que aparentemente nos parezca muy útil, esto se debe a varias razones que atentan contra los principios de la programación modular:

- Legibilidad menor.
- El uso indiscriminado de variables globales produce efectos colaterales. Esto sucede cuando existe una modificación del valor de una variable global dentro de un método por olvidar definir una variable local o un parámetro con ese nombre. La corrección de dichos errores puede ser muy difícil de ver.
- Supone no compartir espacios de memoria con otras funciones y no potenciar el paso de información (llamadas) para que un método trate la información localmente.

Para protegernos frente a los problemas que podrían causar las variables globales y hacer un buen uso de ellas, lo lógico es **crearlas como una constante** con la finalidad de que no puedan ser modificadas por ningún método.

```

1 | public class PruebaVariables {
2 |
3 |     static final double PI = 3.1415926535897932384626433832795;
4 |
5 | }

```

Sobrecarga de métodos

En Java podemos tener **métodos con el mismo nombre (sobrecarga)**, pero se debe tener en cuenta que:

- Si el método tiene un parámetro, deben ser de distintos tipos en cada método.
- Si tiene más de un parámetro, al menos uno debe ser distinto.
- Si no tiene parámetros, no podemos tener métodos con el mismo nombre.

```
static String saludar()
{
    return "Hola, cómo estás??";
}
static String saludar(String nombre)
{
    return "Hola " + nombre + ", cómo estás??";
}
static String saludar(String nombre, String ciudad)
{
    return "Hola" + nombre + ", qué tal por "+ciudad+"??";
}
```

```
1 static String saludar()
2 {
3     return "Hola, cómo estás??";
4 }
5 static String saludar(String nombre)
6 {
7     return "Hola " + nombre + ", cómo estás??";
8 }
9 static String saludar(String nombre, String ciudad)
10 {
11     return "Hola" + nombre + ", qué tal por "+ciudad+"??";
12 }
```

Aunque el tipo de retorno sea distinto, si los parámetros son iguales, se producirá un error de compilación. Esto se debe a que Java realmente no sabe el método que debe utilizar:

```

static String saludar()
{
    return "Hola, cómo estás??";
}
static void saludar()
{
    System.out.println("Hola, cómo estás??");
}
public static void main (String args[])
{
    saludar(); //ERROR: JAVA no sabe el método que debe utilizar
}

```

```

1 | static String saludar()
2 | {
3 |     return "Hola, cómo estás??";
4 | }
5 | static void saludar()
6 | {
7 |     System.out.println("Hola, cómo estás??");
8 | }
9 | public static void main (String args[])
10| {
11|     saludar(); //ERROR: JAVA no sabe el método que debe utilizar
12| }

```

```

static int suma(int a, int b)
{
    return a + b;
}
static void suma(int a, int b)
{
    System.out.println(a + b);
}
public static void main (String args[])
{
    suma(2,4); //ERROR: JAVA no sabe el método que debe utilizar
}

```

```

1 | static int suma(int a, int b)
2 | {
3 |     return a + b;
4 | }
5 | static void suma(int a, int b)
6 | {
7 |

```

```
8  System.out.println(a + b);
9 }
10 public static void main (String args[])
11 {
12     suma(2,4); //ERROR: JAVA no sabe el método que debe utilizar
}
```

Batería de ejercicios sobre métodos



Ejercicio 1

Implementa un método que, dado un número entero, calcule el cubo (N^3). Realiza un programa principal que pregunte un número N al usuario, llame al método e imprima el resultado que este devuelve.



Ejercicio 2

Implementa un método para mostrar un menú de N opciones con su número correspondiente. La última opción será para salir. Realiza un programa principal que imprima el menú y pida la opción al usuario, comprobando que esta sea válida.



Ejercicio 3

- a) Implementa un método (1) para pasar a mayúsculas una cadena.
 - b) Implementa otro método (2) para contar las vocales de una cadena.
 - c) Realiza un programa principal que pida una cadena al usuario, se la pase al método (1) e imprima lo que devuelve (una palabra en mayúsculas).
 - d) Haz que el programa llame también al método (2) con la palabra resultado del método (1) en mayúsculas para que cuente las vocales e imprima la cantidad que este devuelve.
-



Ejercicio 4 (ProgramaMe)

F Ventas

Debido a la crisis, el bar de Javier ha notado un descenso de las consumiciones. Además, según dicen en los telediarios, la ley antitabaco le está perjudicando aún más. Cómo no termina de creerse todo lo que dicen en la televisión, ha decidido hacer un estudio de mercado semanal de su establecimiento. Para ello, ha estado apuntando la caja diaria que se ha realizado en las últimas semanas. Le gustaría saber qué día de la semana se producen el mayor y el menor número de ventas, y si las ventas del domingo superan a la media semanal. De esta manera podrá establecer estrategias de marketing que le permitan recuperar algo de las ganancias perdidas.

Javier abre su bar todos los días menos los Lunes, que utiliza para descansar.

Realiza un programa que ayude a Javier en su cometido. Dada una lista de valores correspondiente a una semana nuestro programa deberá decirle a Javier el día de la semana que más y menos ha vendido, y si las ventas del domingo superan la media.

Entrada

El programa recibirá una lista de semanas a evaluar. Cada semana constará de un valor para cada día. El número de semanas es indeterminado. El programa terminará de ejecutarse cuando para el primer día de la semana se indique una venta de -1.

Salida

Para cada caso de prueba, el programa escribirá una línea conteniendo dos días de la semana (**MARTES, MIERCOLES, JUEVES, VIERNES, SABADO o DOMINGO**). El primero indicará el día de más ventas y el segundo el de menos. Después se indicará un **SI** si el domingo se realizaron más ventas que la media semanal, y **NO** en caso contrario. Las tres palabras se separarán entre ellas por un tabulador.

Si hay empate en alguno de los valores de ventas mínimo y máximo, se especificará **EMPATE**.

Entrada de ejemplo

```
185.50
250.36
163.45
535.20
950.22
450.38
-1
```

Salida de ejemplo

```
SABADO JUEVES SI
```

Resuelve el problema implementando los siguientes métodos:

- Método que permita al usuario introducir la recaudación de una semana y devuelva los datos en un vector.
- Método que a partir de la lista creada en el apartado anterior con los importes diarios, devuelva el día de más ventas.
- Método que a partir de la lista creada en el apartado a) con los importes diarios, devuelva el día con menos ventas.
- Método que calcule la media semanal.
- Método que devuelva la recaudación del domingo.



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<<http://creativecommons.org/licenses/by-sa/4.0/>>



Número hyperpar

Se dice que un número es *hyperpar* cuando todos sus dígitos son pares. ¿Sabes identificarlos?

Entrada

La entrada consta de una serie de casos de prueba. Cada uno está compuesto de una única línea con un número no negativo ni mayor que 10^9 .

Los casos de prueba terminan con un número negativo que no habrá que procesar.

Salida

Para cada caso de prueba se escribirá, en una línea independiente, **SI** si el número es hyperpar y **NO** si no lo es.

Entrada de ejemplo

```
2460
1234
2
-1
```

Salida de ejemplo

```
SI
NO
SI
```



Los saltos de Mario

Mario se encuentra ante el castillo final. Puede verlo desde lo alto del muro en el que se encuentra. En breve podrá entrar en la Cámara de Koopa, enfrentarse (y vencer) al monstruo final y salvar a la princesa.

Sin embargo, tiene ante sí una serie de muros que tendrá que sobrepasar. Para eso, saltará desde el primero de ellos, donde se encuentra, al siguiente, y desde él al siguiente, y así sucesivamente hasta llegar al último.

La pregunta que nos hacemos es, ¿cuántos de estos saltos serán hacia arriba y cuántos hacia abajo? Mario realiza un salto hacia arriba cuando tiene que alcanzar un muro que está por encima de él, y hacia abajo cuando tiene que alcanzar un muro que está por debajo.

Entrada

Cada caso de prueba comienza con un entero mayor que cero y no mayor que 10 que indica el número de muros del escenario (recuerda que Mario se encuentra situado en la parte de arriba del primero). A continuación se proporciona la serie de enteros que indican la altura cada uno de ellos.

Salida

Para cada caso de prueba se mostrará una línea en la que aparecerán dos enteros, uno con los saltos hacia arriba y otro con los altos hacia abajo, separados por un espacio.

Entrada de ejemplo

```
8
1 4 2 2 3 5 3 4
```

Salida de ejemplo

```
4 2
```

F Ventas

Debido a la crisis, el bar de Javier ha notado un descenso de las consumiciones. Además, según dicen en los telediarios, la ley antitabaco le está perjudicando aún más. Cómo no termina de creerse todo lo que dicen en la televisión, ha decidido hacer un estudio de mercado semanal de su establecimiento. Para ello, ha estado apuntando la caja diaria que se ha realizado en las últimas semanas. Le gustaría saber qué día de la semana se producen el mayor y el menor número de ventas, y si las ventas del domingo superan a la media semanal. De esta manera podrá establecer estrategias de marketing que le permitan recuperar algo de las ganancias perdidas.

Javier abre su bar todos los días menos los Lunes, que utiliza para descansar.

Realiza un programa que ayude a Javier en su cometido. Dada una lista de valores correspondiente a una semana nuestro programa deberá decirle a Javier el día de la semana que más y menos ha vendido, y si las ventas del domingo superan la media.

Entrada

El programa recibirá una lista de semanas a evaluar. Cada semana constará de un valor para cada día. El número de semanas es indeterminado. El programa terminará de ejecutarse cuando para el primer día de la semana se indique una venta de -1.

Salida

Para cada caso de prueba, el programa escribirá una línea conteniendo dos días de la semana (MARTES, MIERCOLES, JUEVES, VIERNES, SABADO o DOMINGO). El primero indicará el día de más ventas y el segundo el de menos. Después se indicará un SI si el domingo se realizaron más ventas que la media semanal, y NO en caso contrario. Las tres palabras se separarán entre ellas por un tabulador.

Si hay empate en alguno de los valores de ventas mínimo y máximo, se especificará EMPATE.

Entrada de ejemplo

```
185.50
250.36
163.45
535.20
950.22
450.38
-1
```

Salida de ejemplo

```
SABADO JUEVES SI
```

El pan en las bodas

A Jack Dauson le cuesta mucho aprenderse las normas de etiqueta cuando se sienta en una mesa. ¿Cuál es el cuchillo de la carne? ¿Y el tenedor del pescado? Pero lo que más le incomoda es no saber cuál es el pan que le pertenece. Cuando se sienta a la mesa se encuentra que tanto a la izquierda como a la derecha hay un platito con una barrita de pan. Uno de los dos es el suyo, y el otro es el de la persona que se sienta a su lado pero... ¿a qué lado? Si el suyo es el de la izquierda, el de la derecha será para el comensal que se siente a la derecha. Si la barrita que hay a la derecha es la suya, será justo al contrario.

Normalmente lo que hace es esperar a que algún otro comensal coma algo de pan para deducir cuál es el suyo y así no confundirse (o al menos poder echar la culpa a otro sobre su confusión). Y es que es importante no confundirse de lado porque es posible que eso provoque que haya alguien que se quede sin comer pan.

Hoy Jack está de invitado en una boda donde hay mesas redondas. Cuando se acerca a la suya se da cuenta de que ya hay varias personas que han empezado a comer barritas. ¿Podrán comer pan todos, o hay ya alguien que no podrá porque sus dos vecinos se comerán las dos barritas que él tenía a sus dos lados?



Entrada

La entrada contiene distintos casos de prueba. Cada uno de ellos contiene, en una línea, la descripción de la configuración (válida) de una mesa. La línea comienza con un número positivo que indica el número de asientos (como mucho 1000 sillas). Tras un espacio, aparece un carácter por cada una de las sillas que hay en la mesa. El carácter I indica que en esa silla el comensal ha empezado la barrita de su izquierda; una D indica que ha comido de la barrita de la derecha. Por último un . indica que en esa silla aún no se ha sentado nadie (o que, quién lo ha hecho, aún no ha probado el pan). Los invitados a la boda son personas decentes y no se roban el pan los unos a los otros, por lo que si un comensal ya ha empezado una barrita, la persona de al lado no la probará.

Ten en cuenta que la mesa es circular, por lo tanto se considera que el último carácter tiene a su derecha el primero.

El último caso de prueba, que no deberá ser procesado, contiene una mesa sin sillas.

Salida

Para cada caso de prueba se mostrará una única línea indicando si, una vez llena la mesa, todos los comensales podrán comer pan (**TODOS COMEN**) o hay al menos uno que no podrá (**ALGUNO NO COME**).

Entrada de ejemplo

```
6 .I....  
6 .D.I..  
4 ....  
0
```

Salida de ejemplo

```
TODOS COMEN  
ALGUNO NO COME  
TODOS COMEN
```

G

Radares de tramo

La Dirección Particular de Tráfico (DPT) está empeñada en hacer que los conductores respeten los límites de velocidad. Sin entrar en si es por razones de seguridad, por ahorrar combustible, o con un mero afán recaudatorio, ahora sabemos que además de los radares fijos tradicionales, están poniendo en funcionamiento los radares de tramo.

Desde un punto de vista formal, estos radares se basan en el teorema de Lagrange (también llamado de *valor medio* o de Bonnet-Lagrange), y viene a decir algo así como que, en algún punto de un intervalo cerrado, una función continua y derivable en ese intervalo tendrá derivada instantánea igual a la derivada media en el intervalo.

Aunque asuste a primera vista, la repercusión es sencilla: si hacemos un viaje desde Madrid a Zaragoza y nuestra velocidad media es de 111Km/h, *forzosamente* en algún punto del camino, nuestra velocidad ha sido de 111Km/h.

Los radares de tramo consisten en colocar dos cámaras en dos puntos alejados de una carretera para poder comprobar cuánto tiempo ha tardado el coche en recorrer ese tramo. Si la velocidad media supera la velocidad máxima permitida, gracias al teorema anterior podremos saber (aunque no le hayamos visto) que en algún punto del trayecto ha superado esa velocidad. Por ejemplo, si colocamos las cámaras separadas 10Km en un tramo cuya velocidad está limitada a 110Km/h, y un coche tarda 5 minutos en ser visto por la segunda cámara, sabremos que su velocidad media ha sido de 120Km/h, y por tanto en algún sitio ha superado el límite de velocidad aunque al pasar por debajo de las dos cámaras el coche fuera a 80Km/h.

Entrada

La entrada estará formada por un número indeterminado de casos de prueba. Cada caso de prueba consistirá en tres números: el primero será la distancia (en metros) que separan las dos cámaras, el segundo indicará la velocidad máxima permitida en todo ese tramo (en Km/h) y el tercer y último número indicará el número de segundos que ha tardado un coche en recorrer el tramo. Todos esos números serán enteros.

La entrada terminará cuando todos los números sean cero.

Salida

Para cada caso de prueba, el programa generará una línea, indicando si el coche debe ser multado o no. En concreto, indicará “OK” si el coche no superó la velocidad máxima, indicará “MULTA” si se superó esa velocidad en menos de un 20 % de la velocidad máxima permitida, y “PUNTOS” si la velocidad fue superada en un 20 % o más de esa velocidad; en ese caso además de la multa se le quitarán puntos del carnet.

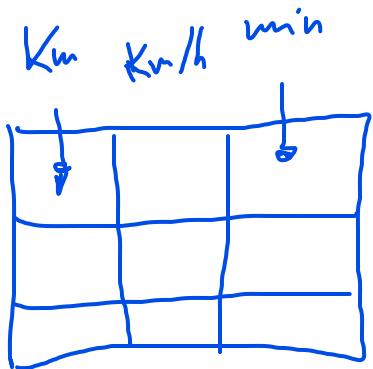
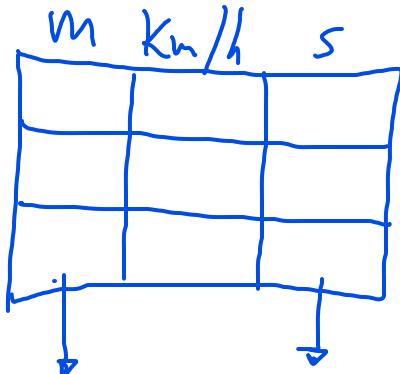
El sistema de radar puede fallar y registrar entradas incorrectas (por ejemplo, indicando que el tiempo que ha tardado el coche es negativo). En esos casos, el sistema mostrará la cadena “ERROR”.

Entrada de ejemplo

```
9165 110 300
9165 110 299
12000 100 433
12000 100 431
12000 100 359
-1000 -50 -100
0 0 0
```

Salida de ejemplo

```
OK
MULTA
OK
MULTA
PUNTOS
ERROR
```



$$\frac{m}{1m} \cdot \frac{1Km}{1000m} \quad \frac{s}{1s} \cdot \frac{1min}{60s}$$

P.E.

$$\frac{10Km}{5min} \cdot \frac{60min}{1h} = 120Km/h$$

for(:)
vel = m[i][0] / m[i][2] * 60

? vel <= m[i][1]?

OK
? vel <= (m[i][1] + m[i][2] * 0.2)?

MULTA
? vel > (m[i][1] + m[i][2] * 0.2)?

PUNTOS

4.4. Recursividad

4.4. Recursividad

La recursividad es una forma de resolver un problema mediante una función que se llama a sí misma hasta que se cumpla una determinada condición.

```
1 public static void funcion(){
2
3     //código
4
5     if /*lo que sea*/{
6         //código
7         funcion();
8
9     }else{
10        //código
11        return;
12    }
13 }
```

Para que un problema pueda ser resuelto mediante recursividad han de cumplirse 3 condiciones:

1. El problema debe tener una salida recursiva.
2. El problema debe tener una salida no recursiva.
3. El problema debe reducirse sí o sí cada vez que realizamos una nueva llamada a la función.

```
1 public static void funcion(){
2
3     //código
4
5     if /*lo que sea*/{
6         //código
7         return funcion(); //SALIDA RECURSIVA
8     }
9 }
```

```

8
9     }else{
10        //código
11        return; //SALIDA NO RECURSIVA
12    }
13 }
```

El ejemplo más típico es el **cálculo del factorial de un número N**, que como ya sabemos, consiste en multiplicar un número por todos los anteriores hasta llegar al 1.

$$\begin{aligned}3! &= 3 \times 2 \times 1 = 6 \\4! &= 4 \times 3 \times 2 \times 1 = 4 \times 3! = 24\end{aligned}$$

En los ejemplos anteriores vemos cómo se calcula el factorial para el número 3 y 4, pero fíjate que el cálculo para el 4 tiene una peculiaridad, y es que es lo mismo que multiplicar el mismo número (**4**) por el factorial del número anterior (**3!**):

$$4! = 4 \times 3! = 24$$

Por lo tanto, tendremos:

```

1 public static int factorial(int n) {
2     return n*factorial(n-1);
3 }
```

Pero, ¿¡cuándo paramos!?

Como ya hemos comentado, en nuestras soluciones recursivas a programas necesitaremos dos tipos de salidas, recursivas y no recursivas. Por lo tanto, **nuestra salida no recursiva** en este caso será cuando se cumpla lo obvio: que el factorial de 0 y 1 es igual a 1.

```

1 public static int factorial(int n) {
2     if (n==0 || n==1) return 1;
3     else return n*factorial(n-1);
4 }
```

De esta forma, nuestra función ya cumple las 3 condiciones vistas anteriormente. Sólo quedará llamarla desde un programa principal, como por ejemplo este:

```
1 public static void main(String args[]) {  
2     int n = factorial(4);  
3     System.out.println(n); //24  
4 }
```

Antes de seguir...

Antes de ver más ejemplos, necesitamos explicar que la recursión hace un uso intensivo de la **pila de llamadas**. Esta pila de llamadas es utilizada por la *JVM* para implementar la ejecución de la llamada a cualquier método.

4.4.1. La pila de llamadas (básica)

La pila de llamadas se aloja en la memoria RAM, y está compuesta por los denominados registros de activación. Cada uno de estos registros...

- Tiene asociada una llamada dada a un método, la del *main* incluida.
- Contiene las variables locales del método y sus parámetros, junto con el valor que se les ha asignado a estos últimos al realizar la llamada (parámetros actuales).

```
public class Ejemplo {  
    static int proceso (int n)  
    {  
        int i, s = 0;  
        for(int i = 0; i < n; i++) s = s+i;  
        return s;  
    }  
    public static void main (String args[])  
    {  
        int num, sum;  
        num = 3;  
        sum = proceso(num);  
    }  
}
```

PILA DE LLAMADAS	
num = 3 sum = ?	main

```
public class Ejemplo {  
    static int proceso (int n)  
    {  
        int i, s = 0;  
        for(int i = 0; i < n; i++) s = s+i;  
        return s;  
    }  
    public static void main (String args[])  
    {  
        int num, sum;  
        num = 3;  
        sum = proceso(num);  
    }  
}
```

PILA DE LLAMADAS	
n = 3 i = 0 s = 0	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 1 s = 0	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 0 s = 0 + 0	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 1 s = 0 + 1	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 2 s = 1	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 2 s = 1 + 2	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
n = 3 i = 3 s = 3	proceso
num = 3 sum = ?	main

PILA DE LLAMADAS	
num = 3 sum = 3	main

Un método sólo puede usar:

- La información contenida en su correspondiente registro de activación de la pila de llamadas.
- Los datos contenidos en las variables globales.

Más ejemplos...

Pila de llamadas con 3 métodos

```
public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
      int p2 = 1;
      for(int i = 0; i < n2; i++) p2 += n2;
      return p2;
    }
    public static void main (String args[])
    {
      int num = 3;
      int res = proceso1(num);
    }
}
```

PILA DE LLAMADAS	
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 1 i = 0	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 0 p2 = 1 i = 0	proceso2
n1 = 3 p1 = 1 + ? i = 0	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 1 + 1 i = 0	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 2 i = 0	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 2 i = 1	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 1 p2 = 1 i = 0	proceso2
n1 = 3 p1 = 2 + ? i = 1	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 1 p2 = 1 + 1 i = 0	proceso2
n1 = 3 p1 = 2 + ? i = 1	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 1 p2 = 2 i = 1	proceso2
n1 = 3 p1 = 2 + ? i = 1	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 2 + 2 i = 1	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 4 i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 2 p2 = 1 i = 0	proceso2
n1 = 3 p1 = 4 + ? i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 2 p2 = 1 + 2 i = 0	proceso2
n1 = 3 p1 = 4 + ? i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 2 p2 = 3 i = 1	proceso2
n1 = 3 p1 = 4 + ? i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 2 p2 = 3 + 2 i = 1	proceso2
n1 = 3 p1 = 4 + ? i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n2 = 2 p2 = 5 i = 2	proceso2
n1 = 3 p1 = 4 + ? i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 4 + 5 i = 2	proceso1
num = 3 res = ?	main

```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

n1 = 3 p1 = 9 i = 3	proceso1
num = 3 res = ?	main

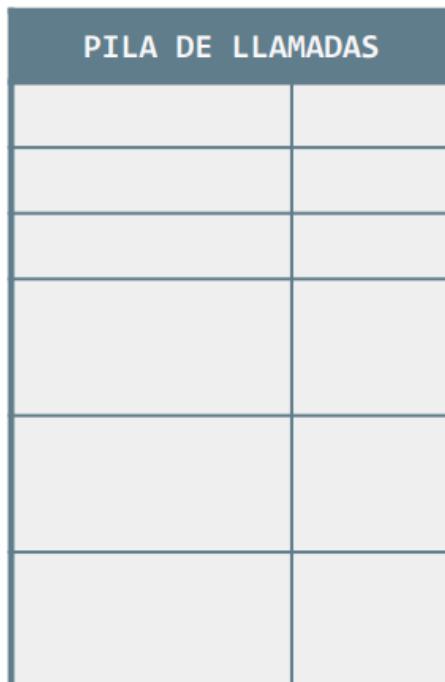
```

public class Ejemplo {
    static int proceso1(int n1)
    { int p1 = 1;
      for(int i = 0; i < n1; i++) p1 += proceso2(i);
      return p1;
    }
    static int proceso2(int n2)
    {
        int p2 = 1;
        for(int i = 0; i < n2; i++) p2 += n2;
        return p2;
    }
    public static void main (String args[])
    {
        int num = 3;
        int res = proceso1(num);
    }
}

```

PILA DE LLAMADAS

num = 3 res = 9	main



EJERCICIO

Imita la pila de llamadas que ejecutará la JVM con estos 3 métodos:

```
public class Ejemplo {
    static int proceso1(int a)
    { int x = 0;
        for(int i = 0; i <= a; i+=2) x += proceso2(i,i+1);
        return x;
    }
    static int proceso2(int a, int b)
    {
        int y = 1;
        for(int j = 0; j < a+b; j++) y++;
        return y;
    }
    public static void main (String args[])
    {
        int x = proceso1(4);
    }
}
```

4.4.2. La pila de llamadas (recursividad)

Si volvemos al ejemplo del cálculo del factorial recursivo, podemos generar la pila de llamadas que haría la *JVM* de la siguiente manera:

```
static int factorial(int n) {  
  
    int r;  
    if(n == 0) r = 1;  
    else r = n*factorial(n-1);  
    return r;  
}  
  
public static void main(String args[]) {  
  
    int f = factorial(4);  
    System.out.println(f);  
}
```

PILA DE LLAMADAS	
f = ?	main

PILA DE LLAMADAS	
$n = 4$ $r = 4 * ?$	fact(4)
$f = ?$	main

PILA DE LLAMADAS	
$n = 3$ $r = 3 * ?$	fact(3)
$n = 4$ $r = 4 * ?$	fact(4)
$f = ?$	main

PILA DE LLAMADAS	
$n = 2$ $r = 2 * ?$	fact(2)
$n = 3$ $r = 3 * ?$	fact(3)
$n = 4$ $r = 4 * ?$	fact(4)
$f = ?$	main

PILA DE LLAMADAS

n = 1 r = 1 * ?	fact(1)
n = 2 r = 2 * ?	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

n = 0 r = 1	fact(0)
n = 1 r = 1 * ?	fact(1)
n = 2 r = 2 * ?	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

n = 0 r = 1	fact(0)
n = 1 r = 1*1	fact(1)
n = 2 r = 2 * ?	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

n = 1 r = 1*1	fact(1)
n = 2 r = 2 * ?	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

n = 1 r = 1*1	fact(1)
n = 2 r = 2*1	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

n = 2 r = 2*1	fact(2)
n = 3 r = 3 * ?	fact(3)
n = 4 r = 4 * ?	fact(4)
f = ?	main

PILA DE LLAMADAS

$n = 2$ $r = 2 * 1$	fact(2)
$n = 3$ $r = 3 * 2$	fact(3)
$n = 4$ $r = 4 * ?$	fact(4)
$f = ?$	main

PILA DE LLAMADAS

$n = 3$ $r = 3 * 2$	fact(3)
$n = 4$ $r = 4 * ?$	fact(4)
$f = ?$	main

PILA DE LLAMADAS

$n = 3$ $r = 3 * 2$	fact(3)
$n = 4$ $r = 4 * 6$	fact(4)
$f = ?$	main

PILA DE LLAMADAS	
$n = 4$ $r = 4*6$	fact(4)
$f = ?$	main

PILA DE LLAMADAS	
$n = 4$ $r = 4*6$	fact(4)
$f = 24$	main

PILA DE LLAMADAS	
$f = 24$	main

PILA DE LLAMADAS

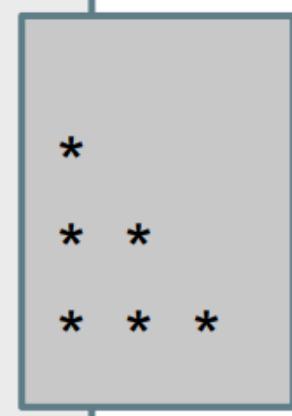
4.4.3. Ampliación sobre recursividad

A estas alturas estará pensando... ¿pero esto no es como usar bucles?

La realidad es que prácticamente todos los programas que se resuelven mediante bucles pueden resolverse también con recursividad de forma más eficiente, pero la complejidad de la lógica a implementar de forma recursiva no siempre lo permite.

Por ejemplo, crear un triángulo:

```
static void tri(int contB, int contA, int n)
{
    if (contB < contA) {
        System.out.print("* ");
        tri(contB + 1, contA, n);
    } else {
        System.out.println();
        if (contA < n)
            tri(0, contA + 1, n);
    }
}
public static void main(String[] args)
{
    int altura = 3;
    tri(0, 0, altura);
}
```



```
1 public static void tri(int contB, int contA, int n){

2

3     if (contB < contA) {
4         System.out.print("* ");
5         tri(contB + 1, contA, n);
6     } else {
7         System.out.println();
8         if (contA < n)
9             tri(0, contA + 1, n);
10    }
11 }

12

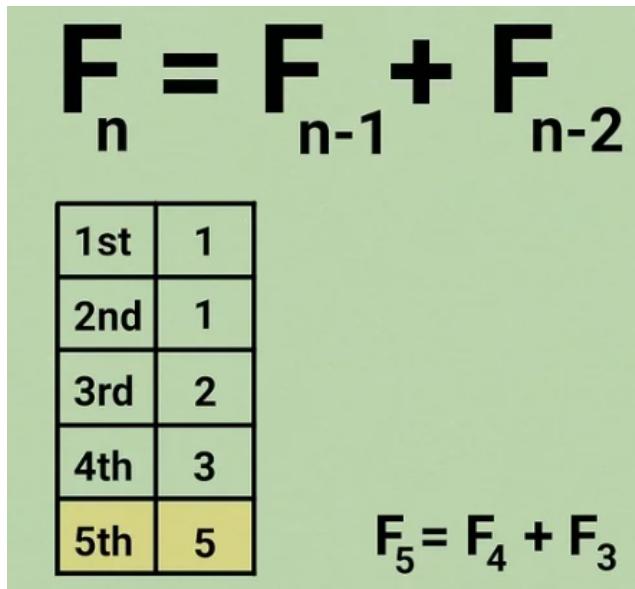
13 public static void main(String[] args){
14     int altura = 3;
15
16 }
```

```

        tri(0, 0, altura);
    }

```

Otro ejemplo típico cuando se cuenta recursividad es la llamada **serie de Fibonacci**.



Aplicando los mismos pasos del factorial, necesitaremos una condición para que la función tenga una salida recursiva y otra no recursiva. En nuestro caso, sabemos que cuando N sea < 2 (para 0 y 1) devolveremos siempre N.

```

1 public static int fibonacci(int n){
2
3     if (n<2) {
4         return n;
5     } else {
6         return fibonacci(n-1) + fibonacci(n-2);
7     }
8 }
9

```

Si quisiéramos imprimir los 10 primeros números de la serie de Fibonacci, implementaríamos el siguiente programa principal que llama a la función anterior:

```

1 public static void main(String[] args){
2     for (int i = 0; i <= 10; i++) {
3         System.out.println("fib(" + i + ") = " + fibonacci(i));
4     }
5 }

```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:  
fib(0) = 0  
fib(1) = 1  
fib(2) = 1  
fib(3) = 2  
fib(4) = 3  
fib(5) = 5  
fib(6) = 8  
fib(7) = 13  
fib(8) = 21  
fib(9) = 34  
fib(10) = 55
```

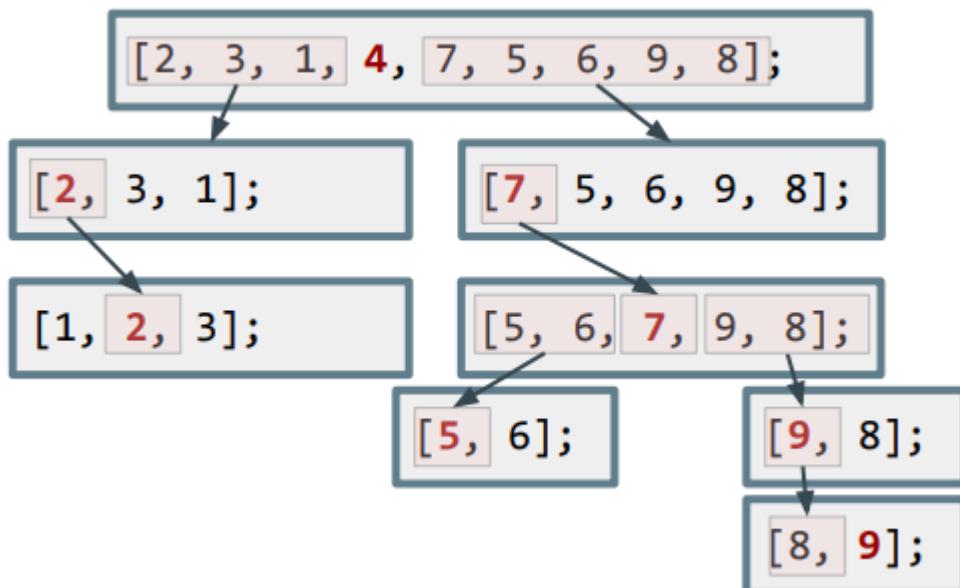
El algoritmo QuickSort...

Tal y como vimos en el tema 3, Quicksort es uno de los algoritmos de ordenación más rápidos que existe haciendo uso de la estrategia **divide y vencerás**.

Por recordar cómo se comportaba...

```
1 | int nums[] = {4, 7, 5, 6, 2, 3, 1, 9, 8};
```

Teniendo en cuenta el vector anterior, debíamos elegir como pivote el elemento de más a la izquierda (4). A partir de ahí, colocábamos a su izquierda los menores, y a su derecha los mayores. Repetimos el mismo proceso con las 2 sublistas de la izquierda y de la derecha; y así sucesivamente sobre cada una de las sublistas, mientras las sublistas tengan más de un elemento:



Como ves, el problema se va haciendo cada vez más "pequeño", y ahora conociendo la recursividad, es posible sospechar que la forma de implementarlo sea usando una función recursiva... Y de hecho, así es.



Ejercicio 1

Busca en Internet alguna implementación del algoritmo *QuickSort* y comprueba que usa la recursividad para solucionarlo.



Ejercicio 2

Imprime los números de 1 a N sin usar bucles.



Ejercicio 3

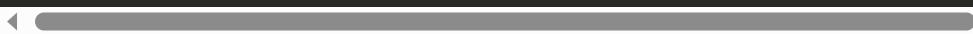
Implementa un programa que sume los números del 1 al 10 sin usar bucles.

Soluciones



Ejercicio 2

```
1 public static void imprimir(int cont, int fin){  
2  
3     if (cont<=fin) {  
4         System.out.println(cont);  
5         cont++;  
6         imprimir(cont,fin);  
7     }  
8  
9 }  
10  
11 public static void main(String[] args){  
12     Scanner teclado = new Scanner(System.in);  
13     System.out.println("Introduce el número N hasta el que quieras");  
14     int n = teclado.nextInt();  
15     imprimir(1,n);  
16 }
```



Ejercicio 3

```
1 public static void main(String[] args){  
2     int n = 10;  
3     System.out.println("Suma = " + contar(1,n));  
4 }  
5  
6 public static int contar(int cont, int fin){  
7  
8     if (cont<fin){  
9         return cont + contar(cont+1,fin);  
10    }  
11 }
```

```
12 |         return cont;  
13 |     }  
|
```

Actividades sobre recursividad [entregable]

Crea un programa principal que muestre un menú de opciones y permita ejecutar cualquiera de los ejercicios que realices a continuación. Por ejemplo:

```
C:\Program Files\Java\JDK-25\bin\java.exe" "-javaagent:c  
*** Batería de ejercicios sobre recursividad ***  
Selecciona a continuación el modo que quieras ejecutar...  
1 - Dígitos  
2 - Potencias  
3 - Del Revés  
4 - Binario  
5 - A binario  
6 - Orden alfabético  
7 - Mostrar suma
```

Una vez ejecutado el ejercicio deseado, debemos preguntar siempre al usuario si quiere volver al menú principal o salir del programa. Por ejemplo:

```
El resultado es 24  
  
Elige una opción:  
[M] - Volver al menú principal  
[X] - Salir
```

Usa el método:

```
public static void borrar() throws IOException, InterruptedException {  
    new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();  
}
```

para limpiar la pantalla antes de imprimir de nuevo el menú principal.



Ejercicio 1

Crea un método que obtenga la cantidad de dígitos de un número N mayor que cero. Se debe pasar como parámetro el número N.

⚠️ **OJO:** no vale pasarlo a *String* y calcular el tamaño con la función *length()!!!*

- Si ese número N es 0, devuelvo 0.
 - Si N no es 0, recorremos del número dividiéndolo entre 10 hasta llegar a 0.
 - Cada vez que dividamos entre 10, aumentamos en 1 nuestro contador.
-



Ejercicio 2

Crea un método que obtenga el resultado de elevar un número a otro. Ambos números se deben pasar como parámetros. Los números deben ser positivos.



Ejercicio 3

Crea un método que dado un número positivo, lo imprima invertido por pantalla.

- Haz lo mismo pero con una cadena de texto como entrada. Usa el método *.toCharArray()*.
 - En el programa principal de este ejercicio se deberá preguntar al usuario si quiere invertir una cadena o un número, y llamar al método que corresponda.
-



Ejercicio 4

Crea un método que compruebe si un número es binario. Un número binario está formado únicamente por ceros y unos.



Ejercicio 5

Crea un método que obtenga el número binario de un número N pasado como parámetro.



Ejercicio 6

Crea un método que compruebe si una palabra está ordenada alfabéticamente.



Ejercicio 7

Crea un método que obtenga la suma de los números naturales desde 1 hasta N. Se debe pasar como parámetro el número N, siempre mayor que cero. Se debe imprimir toda la cadena por consola. Por ejemplo, para N=4:

$$1+2+3+4 = 10$$



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<http://creativecommons.org/licenses/by-sa/4.0/>

Bonus. Javadoc

BONUS. Javadoc

Documentar proyectos Java con Javadoc

Documentar un proyecto es algo fundamental de cara a su futuro mantenimiento. Cuando programamos una clase, debemos generar documentación lo suficientemente detallada sobre ella como para que otros programadores sean capaces de usarla.

Javadoc es una utilidad de *Oracle* para la generación de documentación en formato *HTML* a partir de código fuente *Java*. Javadoc es el estándar para documentar clases de *Java*. La mayoría de los IDEs utilizan Javadoc para generar de forma automática documentación de clases.

En IntelliJ IDEA, bastará con escribir `/**` justo antes de la cabecera de algún método y presionar enter:

```
 /**
 * |
 * @param semana
 * @return
 */
public static float buscarMinimo(float[] semana){ 1 usage

    Arrays.sort(semana);
    float min = semana[0];
    return min;

}
```

Automáticamente se nos generará un fragmento de comentarios que contiene la etiqueta `@param` y `@return`. La idea es que cada programador escriba para cada etiqueta qué parámetros acepta el método correspondiente y qué devuelve. Por ejemplo:

```

/**
 * 
 * @param semana - es un vector de tipo float que contiene los importes hechos en caja de toda la semana
 * @return min - devuelve el valor mínimo encontrado en el vector pasado como parámetro.
 */
public static float buscarMinimo(float[] semana){ 1 usage

    Arrays.sort(semana);
    float min = semana[0];
    return min;

}

```

Existen muchas más etiquetas que podríamos usar para documentar y dejar registrado qué hace nuestro código, pero nos ceñiremos a las siguientes:

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre autor o autores
@deprecated	Indica que el método o clase es obsoleto (propio de versiones anteriores) y que no se recomienda su uso.	Descripción
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	Nombre de parámetro y descripción
@return	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor de retorno
@see	Asocia con otro método o clase.	Referencia cruzada referencia (#método()); clase#método(); paquete.clase; paquete.clase#método()).
@version	Versión del método o clase.	Versión

Las etiquetas **@author** y **@version** se usan para documentar clases. Por lo tanto, no son recomendables en la cabecera de constructores ni métodos.

```

/**
 * Esta clase contiene los métodos necesarios para resolver el problema 314 del Concurso Programame
 * @author: Patricia Benavente
 * @version: 1.0 (10/12/2024)
 * @see <a href = "https://aceptaelreto.com/problem/statement.php?id=314" /> Temperaturas extremas </a>
 */
public class Olimpiada {

    static Scanner teclado = new Scanner(System.in); 2 usages

    public static void main(String[] args) {
        while (casoDePrueba()) {
        }
    } // main

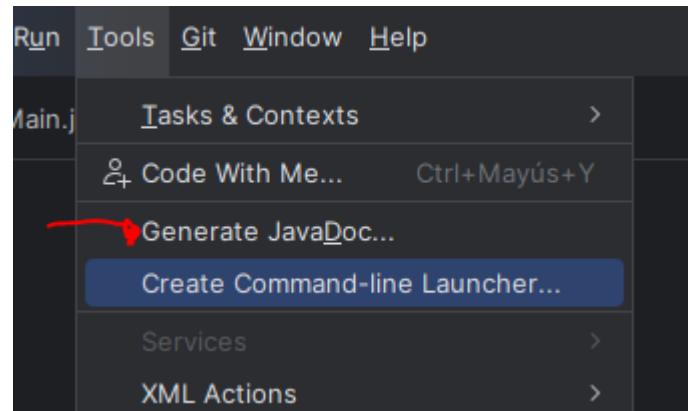
```

Dentro de los comentarios se admiten etiquetas *HTML*. Por ejemplo, con **@see** se puede referenciar una página web como link para recomendar su visita de cara a ampliar

información.

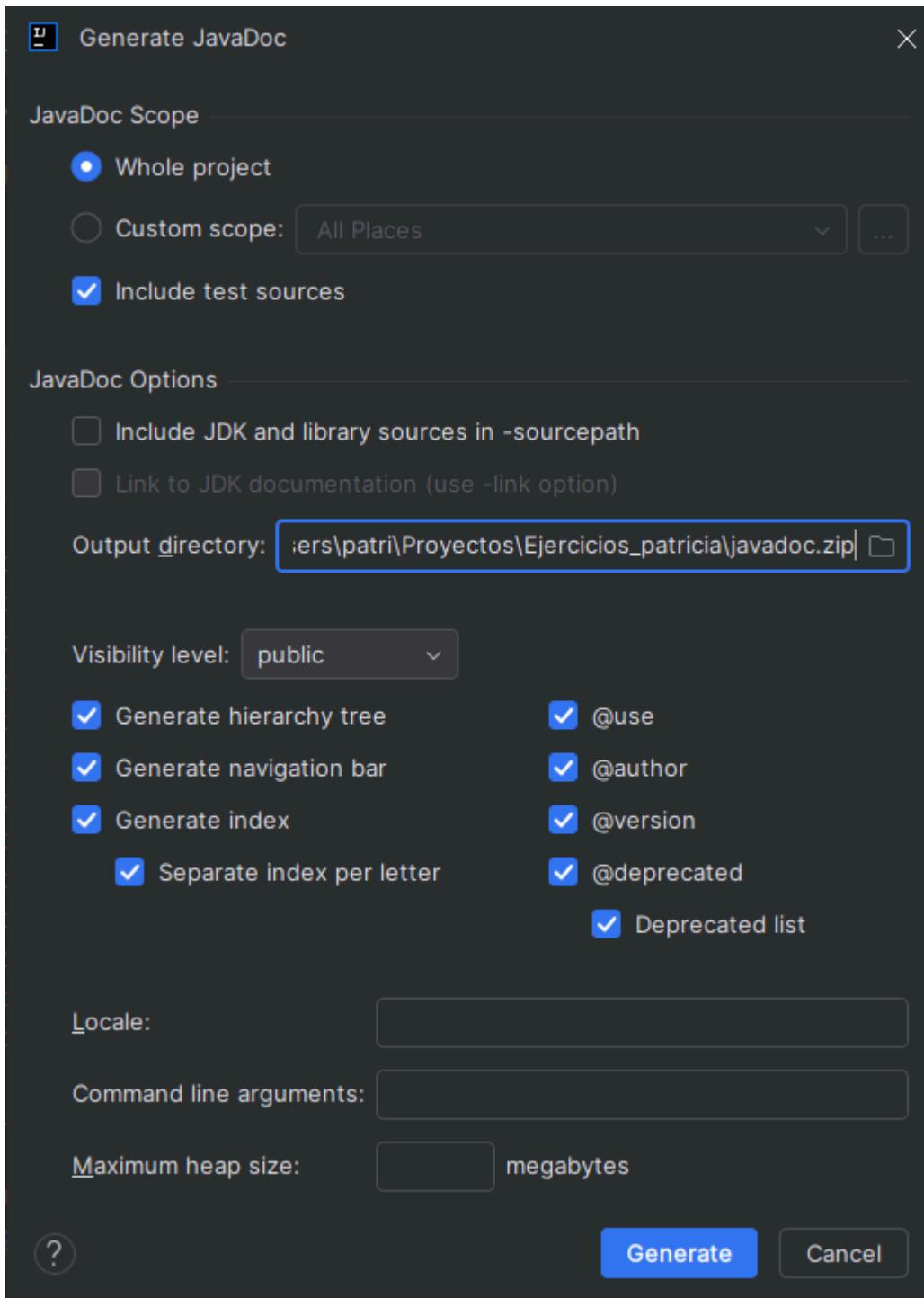
Generación de HTML

Para generar el archivo que creará el formato de documentación formal, desde *IntelliJ IDEA* accederemos al menú **Tools --> Generate JavaDoc...**



Configuramos el documento a nuestro gusto y lo guardamos en la carpeta que queramos.

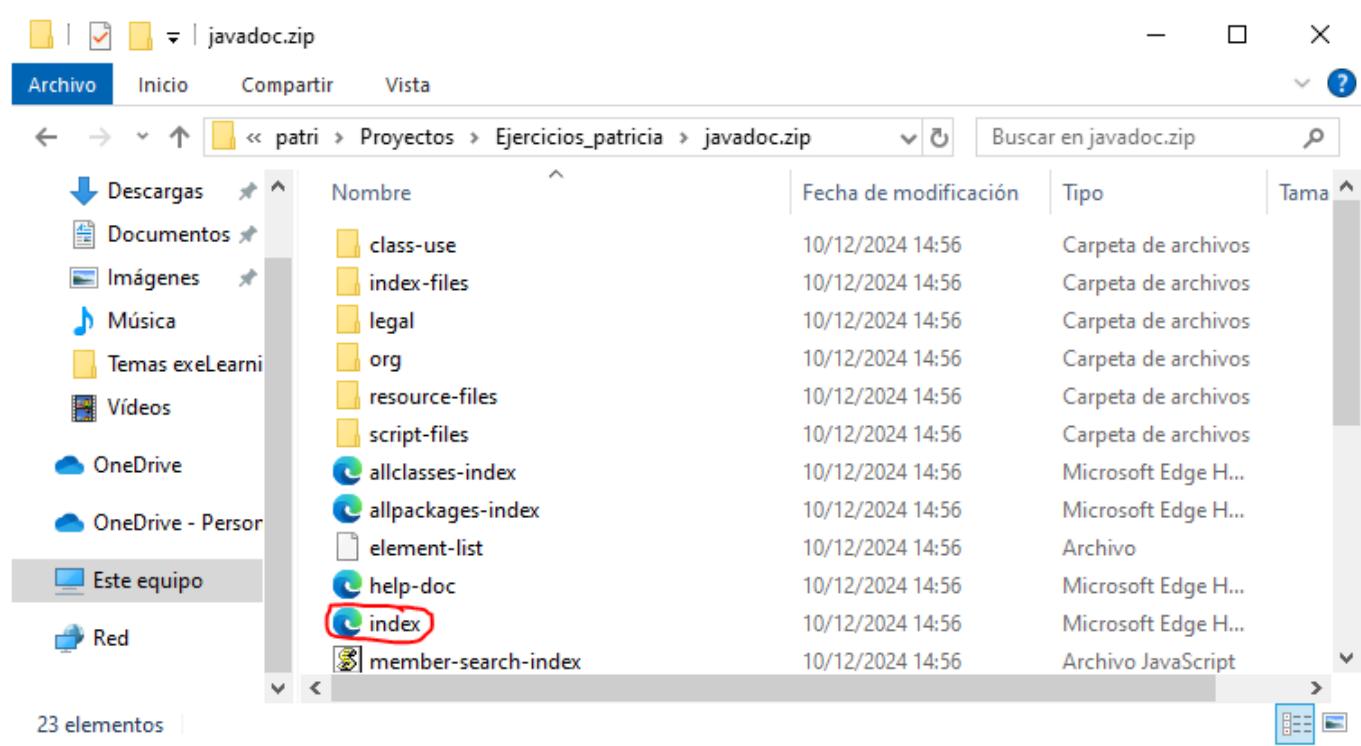
RECOMENDACIÓN: crear un archivo .zip



Se empezará a ejecutar...

```
*C:\Program Files\Java\jdk-23\bin\javadoc.exe" -public -splitindex -use -author -version -d C:\Users\patri\Proyectos\Ejercicios_patricia @C:\Users\patri\A
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Matrices.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Examen_UD2_DAW.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Primitiva.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\NumerosSuerte.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example>Main.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Metodos.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Ejercicios.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\CarreraDeBuses.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Examen_UD1.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\test\java\Pruebas.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Olimpiada.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Examen_UD2_DAW.java...
Loading source file C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Pruebas.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 23.37-2369
Building tree for all the packages and classes...
C:\Users\patri\Proyectos\Ejercicios_patricia\Ejercicios_patricia\src\main\java\org\example\Olimpiada.java:8: error: unknown tag. Mistyped @author or an un
```

Cuando finalice, buscamos los archivos generados y abrimos el principal en cualquier navegador: **index.html**



The screenshot shows a JavaDoc-style interface for the 'Olimpiada' class. The main content area includes:

- Class Olimpiada**
java.lang.Object ↗
org.example.Olimpiada
- public class Olimpiada**
extends Object ↗
- Description: Esta clase contiene los métodos necesarios para resolver el problema 314 del Concurso Programame
- See Also: Temperaturas extremas ↗
- Constructor Summary**

Constructors	Description
Olimpiada()	
- Method Summary**

All Methods	Static Methods	Concrete Methods
main(String[])		
casoDePrueba()		
rellenarSemana(int)		
buscarMínimo(float[])		
buscarMáximo(float[])		
mediaSemanal(float[])		
imprimirResultados(float, float, float, float, float[])		
diaSemana(int)		

Contents

Description
Constructor Summary
Method Summary
Constructor Details
Olimpiada()
Method Details

main(String[])

casoDePrueba()
rellenarSemana(int)
buscarMinimo(float[])
buscarMaximo(float[])
mediaSemanal(float[])
imprimirResultados(float, float, float, float, float[])
diaSemana(int)

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

casoDePrueba

```
public static boolean casoDePrueba()
```

rellenarSemana

```
public static float[] llenarSemana(int pos1)
```

buscarMinimo

```
public static float buscarMinimo(float[] semana)
```

Parameters:

semana -- es un vector de tipo float que contiene los importes hechos en caja de toda la semana

Returns:

min - devuelve el valor mínimo encontrado en el vector pasado como parámetro.

buscarMaximo

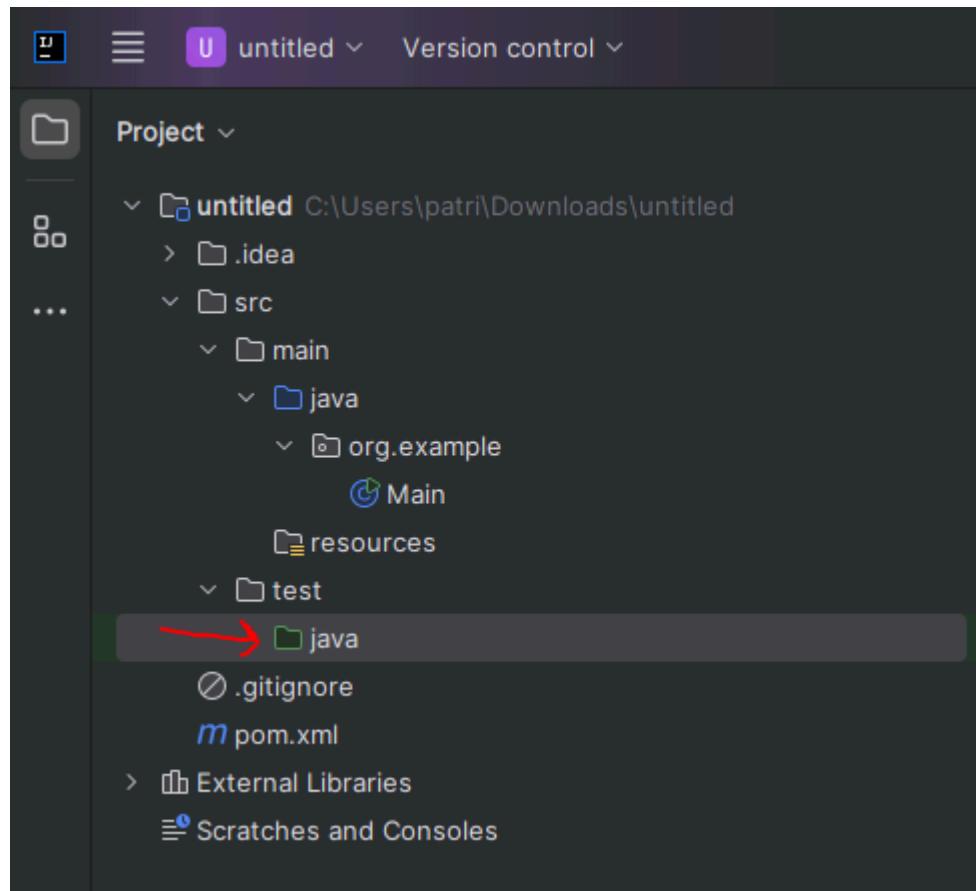
Obra publicada con **Licencia Creative Commons Reconocimiento Compartir igual 4.0**
<http://creativecommons.org/licenses/by-sa/4.0/>

4.3. Introducción a las pruebas unitarias con JUnit

Como ya te habrás dado cuenta, en el desarrollo de software las pruebas son una parte fundamental para garantizar la calidad y fiabilidad de nuestras aplicaciones. Una de las herramientas más populares y potentes para realizar pruebas unitarias en Java es *JUnit*, un marco de trabajo que nos permite escribir y ejecutar pruebas de manera sencilla y eficiente.

Tutorial de iniciación a *JUnit*

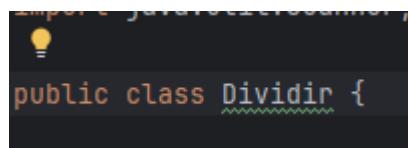
1. Detecta en tu proyecto la carpeta llamada *test* y comprueba que contiene una carpeta verde, llamada *java*:



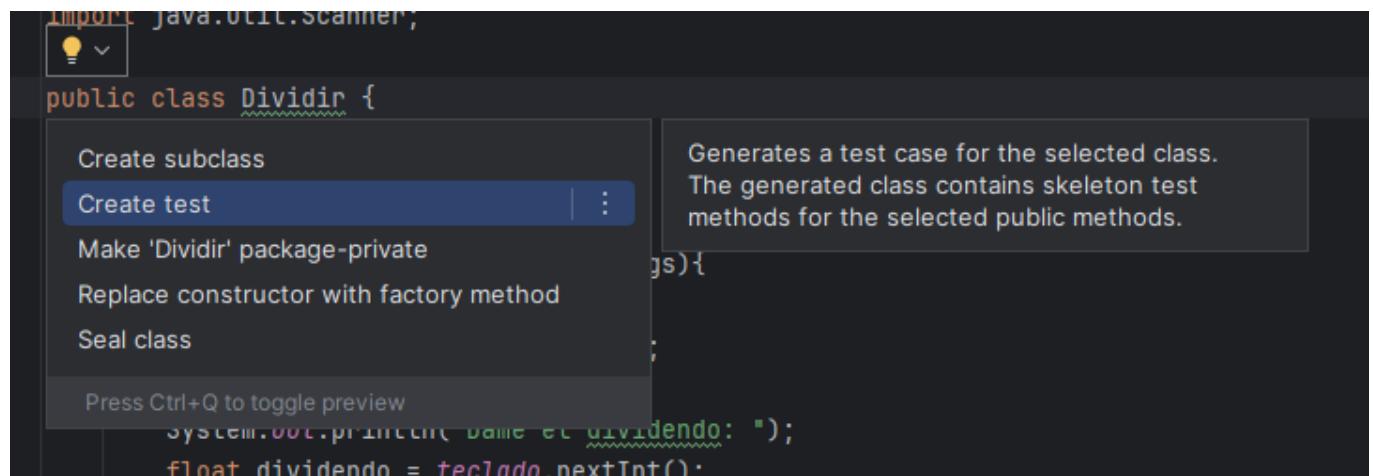
2. Crea o ubica alguna clase que contenga algún ejercicio sencillo que tengas en tu proyecto. Por ejemplo, el siguiente programa calcula el resultado de una división:

```
1 import java.util.Scanner;
2
3 public class Dividir {
4
5     static Scanner teclado;
6
7     public static void main (String[] args){
8
9         teclado = new Scanner(System.in);
10
11        System.out.println("Dame el dividendo: ");
12        float dividendo = teclado.nextInt();
13        System.out.println("Dame el divisor: ");
14        float divisor = teclado.nextInt();
15
16        boolean ok = comprobarDivisor(divisor);
17
18        if (ok){
19            float resultado = dividir(dividendo,divisor);
20            System.out.println("El resultado es " + resultado);
21        }
22
23    }
24
25    public static float dividir(float dividendo, float divisor){
26
27        return dividendo / divisor;
28    }
29
30    public static boolean comprobarDivisor(float divisor){
31
32        if (divisor==0){
33            System.out.println("No se puede dividir entre 0.");
34            return false;
35        }
36
37        return true;
38    }
39}
40}
```

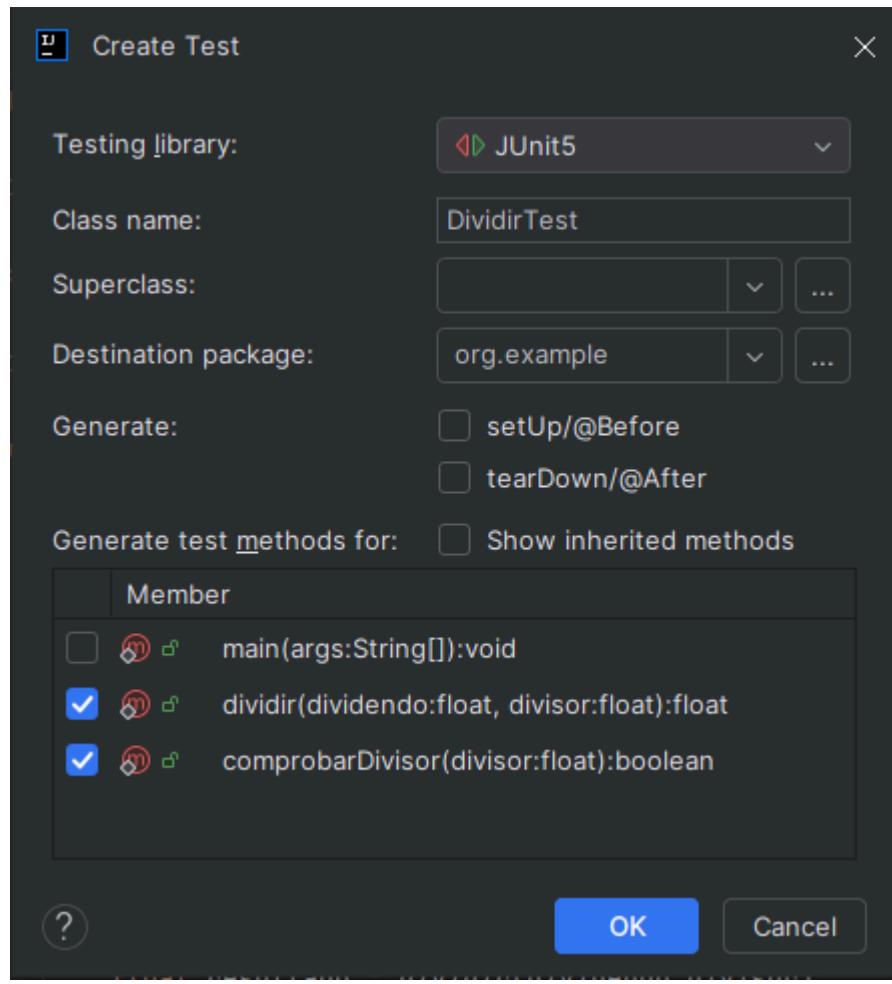
3. Sitúa tu ratón en la cabecera de la definición de la clase, y ubica la bombilla típica que activa *IntelliJ IDEA*:



4. Haz *click* sobre ella y elige la opción *Create test*.



5. En la siguiente pantalla elegiremos los métodos que queremos testear y aceptamos:



6. Se creará una clase nueva que se abrirá automáticamente. Observa que crea un método para cada método que le hemos dicho que vamos a testear, con el mismo nombre:

```
1 package org.example;
2
3 import org.junit.jupiter.api.Test;
4
5 import static org.junit.jupiter.api.Assertions.*;
6
7 class DividirTest {
8
9     @Test
10    void dividir() {
11
12    }
13
14
15     @Test
16    void comprobarDivisor() {
17
18    }
19}
```

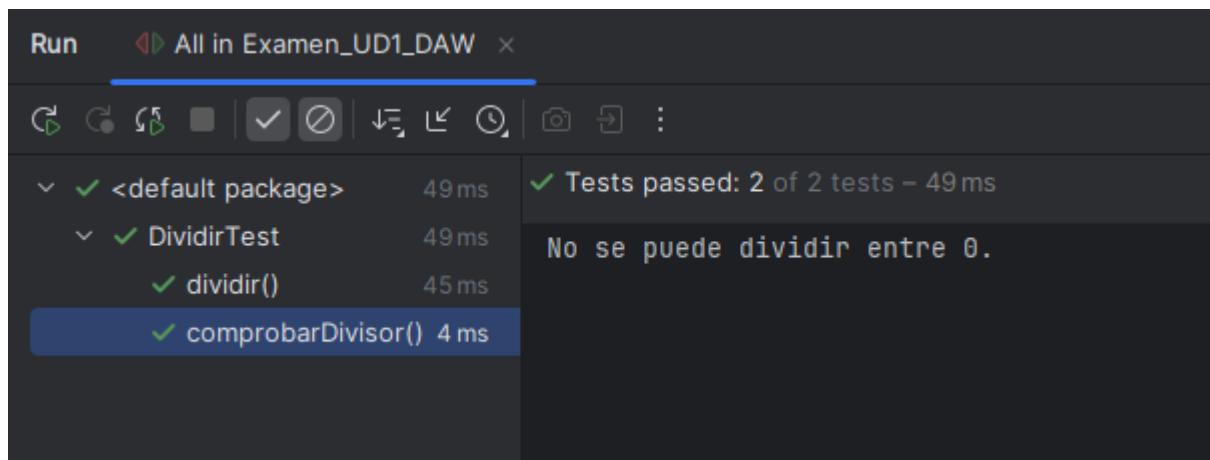
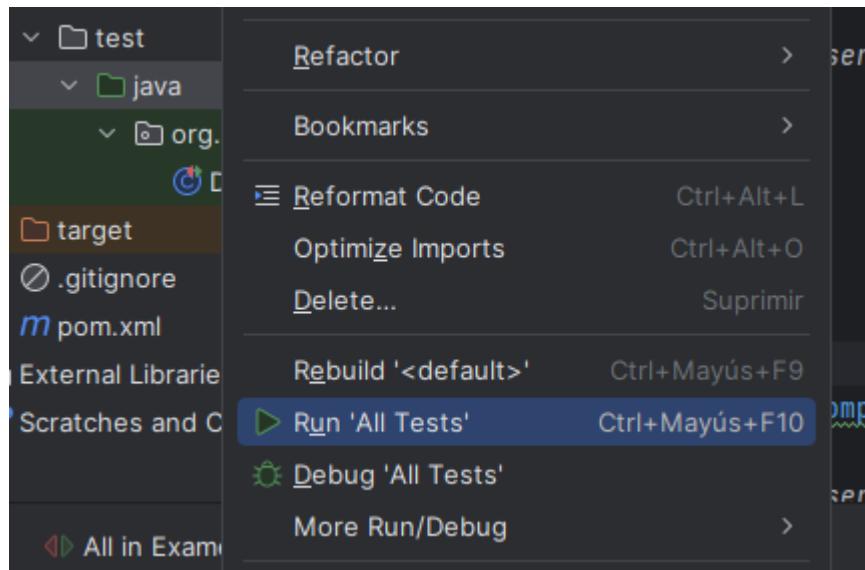
8. Es hora de empezar a rellenar nuestros tests...

```
1 import org.junit.jupiter.api.Test;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class DividirTest {
6
7     @Test
8     void dividir() {
9
10         assertAll(() -> assertEquals(1, Dividir.dividir(2, 2)),
11                  () -> assertEquals(-1, Dividir.dividir(2, -2)),
12                  () -> assertEquals(-5, Dividir.dividir(-10, 2)),
13                  () -> assertEquals(12.5, Dividir.dividir(25, 2)));
14
15    }
16
17    @Test
18    void comprobarDivisor() {
19
```

```

21         assertAll(() -> assertTrue(Dividir.comprobarDivisor(2)),
22                     () -> assertFalse(Dividir.comprobarDivisor(0)),
23                     () -> assertTrue(Dividir.comprobarDivisor(-7)));
24     }
25 }
```

9. Para ejecutarlos, simplemente deberemos hacer *click* sobre la carpeta verde vista anteriormente, y ejecutar la opción ***Run All Tests***:



Validación de la entrada tipo int (try-catch)

Una de las validaciones más típicas es la de formato de entrada, y como error, el que lanza la excepción `java.util.InputMismatchException` cuando introducimos caracteres cuando esperamos un número entero.

Una forma de hacer uso de `JUnit` para validar esta casuística es la siguiente:

- Clase que contiene el método a validar:

```
1 import java.util.InputMismatchException;
2 import java.util.Scanner;
3
4 public class Prueba {
5
6     public int leerEntero() {
7         Scanner teclado = new Scanner(System.in);
8         System.out.print("Introduce un número: ");
9
10        try {
11            return teclado.nextInt();
12        } catch (InputMismatchException e) {
13            System.out.println("Formato de entrada no válido (número en");
14            return -1; //definimos un valor predeterminado en caso de e
15        }
16    }
17 }
```

- Caso de prueba `@Test`:

```
import org.junit.jupiter.api.Test;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;
import static org.junit.jupiter.api.Assertions.*;
class PruebaTest {
```

```
11
12     @Test
13     void leerEntero() {
14
15         //simulamos que el usuario ingresa caracteres
16         String entradaSimulada = "aaa"; // 'a' es un carácter no válido
17         System.setIn(new ByteArrayInputStream(entradaSimulada.getBytes(
18
19             //Creamos un ByteArrayOutputStream para guardar la salida por consola
20             ByteArrayOutputStream salida = new ByteArrayOutputStream();
21             System.setOut(new PrintStream(salida)));
22
23         Prueba prueba = new Prueba();
24
25         //llamamos al método que procesa la entrada
26         int resultado = prueba.leerEntero();
27
28         //verificamos que el valor que devuelve es el esperado (-1)
29         assertEquals(-1, resultado);
30
31         //verificamos que se haya impreso el mensaje de error esperado
32         String salidaEsperada = "Formato de entrada no válido (número entero)";
33         // assertEquals(salidaEsperada, salida.toString().trim());
34         assertTrue(salida.toString().trim().contains(salidaEsperada));
35
36     }
37 }
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0
<http://creativecommons.org/licenses/by-sa/4.0/>

PROYECTO INDIVIDUAL



Elección de proyecto

Entra en el sitio web de **ProgramaMe: Concurso de Programación para Ciclos Formativos** y elige alguno de los ejercicios que aparecen en los cuadernillos del histórico de problemas: <https://programame.com/archive.php>



Entrega del proyecto



Exposición + taller de evaluación entre iguales

Programación

EXAMEN TEMA 4 – PROGRAMACIÓN MODULAR

(15/01/2025)

1. (2,5p) Teniendo en cuenta el siguiente menú de opciones a ejecutar,

```
***** BIENVENIDO A LA CALCULADORA RÁPIDA *****
[C] - Iniciar
[X] - Salir
```

```
CALCULADORA INICIADA...
Elige la operación a realizar:
[+] Sumar
[-] Restar
[x] Multiplicar
[/] Dividir
[X] - Volver a pantalla principal
```

- a) (1p) Indica cuántos métodos crearías dentro de un programa *Java* para darle funcionalidad y explica por qué.
- b) (1,5p) Escribe la cabecera que tendrían cada uno de los métodos (incluidos los parámetros) y lo que devolverían (en caso de que devuelvan algo).

2. (1p) Identifica las variables globales y locales sobre los métodos de la siguiente clase:

```
public class Ejemplo {
    static int x;
    static int cuadrado(int z){
        return z*z;
    }
    public static void main (String args[]){
        int p = 5;
        z = 3;
        x = cuadrado(p);
        System.out.print(x);
    }
}
```

- ¿Qué pasará en el programa?

3. (1p) Enumera las etiquetas típicas de *Javadoc* e indica con tus palabras la utilidad de documentar una aplicación usando esta herramienta.
4. (1p) ¿En qué consiste la sobrecarga de métodos en *Java*? Pon ejemplos.
5. (2p) Dada la siguiente función recursiva,

```

public static int funcion_recursiva(int n) {
    if (n <= 0) {
        return 0;
    } else {
        return -n + funcion_recursiva(--n);
    }
}

```

- a) (0,5p) Marca sobre el código dado el caso base (salida no recursiva) y el caso general (salida recursiva).
- b) (1,5p) ¿Qué mostraría el siguiente programa que llama a la función recursiva anterior?

```

public static void main(String[] args) {
    System.out.println("El resultado es: " + funcion_recursiva(4));
}

```

6. (1p) Crea el código para el método ***comprobarDivisor()*** teniendo en cuenta que se van a lanzar las siguientes pruebas contra él para verificar su funcionamiento en *JUnit*:

```

@Test
void comprobarDivisor() {

    assertAll(() -> assertTrue(Dividir.comprobarDivisor(2)),
              () -> assertFalse(Dividir.comprobarDivisor(0)),
              () -> assertTrue(Dividir.comprobarDivisor(-7)));
}

```

7. (1,5p) Intenta replicar el comportamiento de la pila de llamadas del siguiente fragmento de código y contesta a las preguntas:

- a) ¿Qué mostrará el método principal ***main()***?
- b) ¿Qué devolverá el método ***proceso2()*** en cada una de sus llamadas?

```

public class Examen {

    static int procesol(int n1) {
        int p1 = 0;
        for (int i = 0; i < n1; i += 2) {
            p1 += proceso2(i);
        }
        return p1;
    }

    static int proceso2(int n2) {
        int p2 = 0;
        for (int i = 0; i < n2; i++) {
            p2 += n2;
        }
        return p2;
    }

    public static void main(String args[]) {
        System.out.println(procesol(5));
    }
}

```