EFA
MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

PROGRAMACIÓN

DANIEL GONZÁLEZ-CALERO JIMÉNEZ

UT2 – ESTRUCTURAS DE CONTROL

EFA MORATALAZ

1º CFGS Desarrollo de Aplicaciones Web

PROGRAMACIÓN

UT2 – ESTRUCTURAS DE CONTROL

INDICE

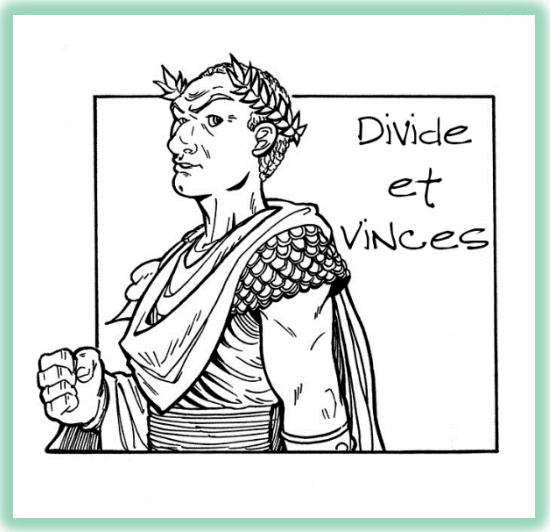
PROGRAMACIÓN ESTRUCTURADA

Los programas son de muy diverso tipo y debido a ello, pueden tomar los datos con los que trabajan desde varios orígenes. Además, también son diseñados para mostrar los resultados de distintas maneras. Pero independientemente de cómo se obtengan los datos, la ejecución de las instrucciones contenidas en ellos es secuencial y sigue un orden que va desde las primeras líneas del programa hasta las últimas.

La programación estructurada es un paradigma de programación que usa tan sólo a tres tipos de estructuras: **secuencia**, **selección** e **iteración**.

- Surge a finales de 1970
- Su objetivo es mejorar la claridad, calidad y tiempo de desarrollo de un programa.
- Facilita el mantenimiento de los programas una vez finalizados.

```java
package demo01;
public class Demo01 {
    public static void main(String[] args) {
        //Mensaje
        System.out.println("Uso de la estructura if");
        int xnum;
        int xedad = 20;

        if (xedad>=18){
            System.out.println("Es mayor de edad");
        }else{
            System.out.println("Es menor de edad");
        }
    }
}
```

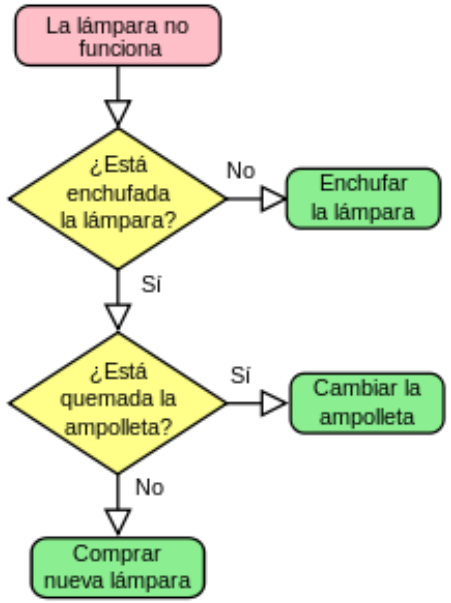


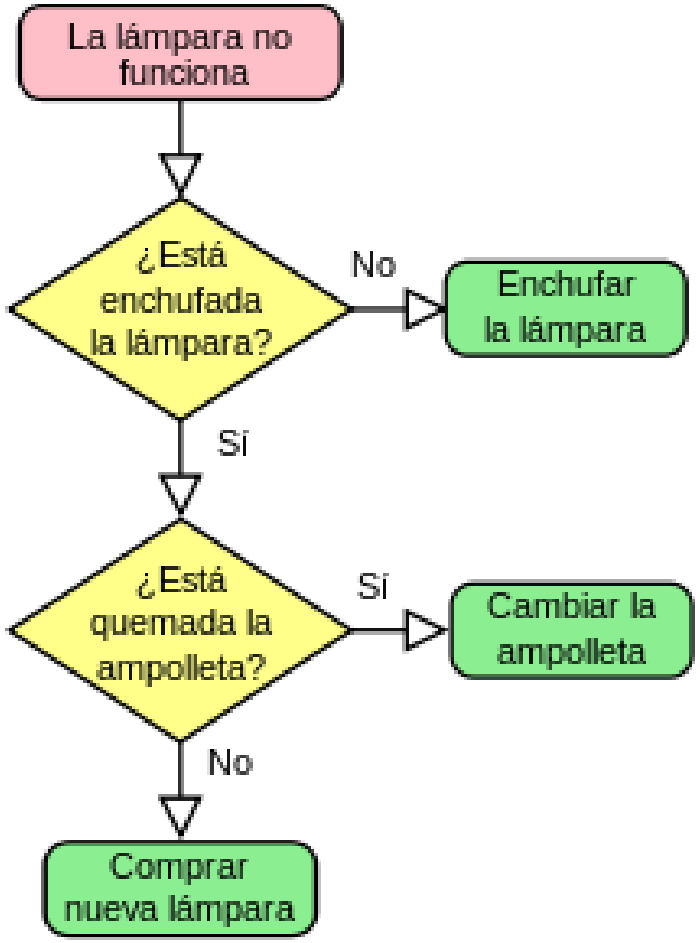
La programación estructurada consiste en dividir los programas en módulos y se basa en el desarrollo de programas que van de lo general a lo particular

¡Divide y vencerás!

- Para la solución de un problema en particular, se inicia considerando las funciones que tiene que cumplir el programa en general y después se va desmembrando estas funciones en subfunciones más pequeñas hasta llegar al caso último o más particular

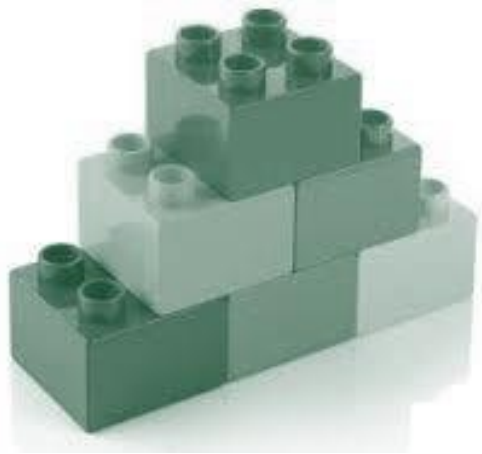
- La representación gráfica de la programación estructurada se realiza a través de **diagramas de flujo**, el cual representa el programa con sus entradas, procesos y salidas.






```
if (enchufada) {
    if (quemada la bombilla){
        //Cambiar la bombilla
    } else {
        //Comprar nueva lámpara
    }

} else {
    // Enchufar la lámpara
}
```

Todo programa con un único punto de entrada y un único punto de salida, cuyas sentencias alcancen todas en algún momento y que no posea bucles infinitos **se puede construir con** tres constructores elementales: **secuencia**, **selección** e **iteración**.



➢ Secuencia

- Conjunto de secuencias que se ejecutan en orden.
- Son las sentencias comunes, las que acaban en ;
  - Sentencias de asignación, operaciones…

➢ Selección

- Elige que sentencias se ejecutan en función de una condición
  - Estructuras de control como *if-else* o *switch*

➢ Iteración

- Las estructuras de control repetitivas. Repiten conjuntos de instrucciones
  - Estructuras de repetición como *while*, *for* o *do-while*

ESTRUCTURA SECUENCIAL

Serie de instrucciones que suceden unas tras otras y guardan relación entre sí.

¡Efecto Dominó!

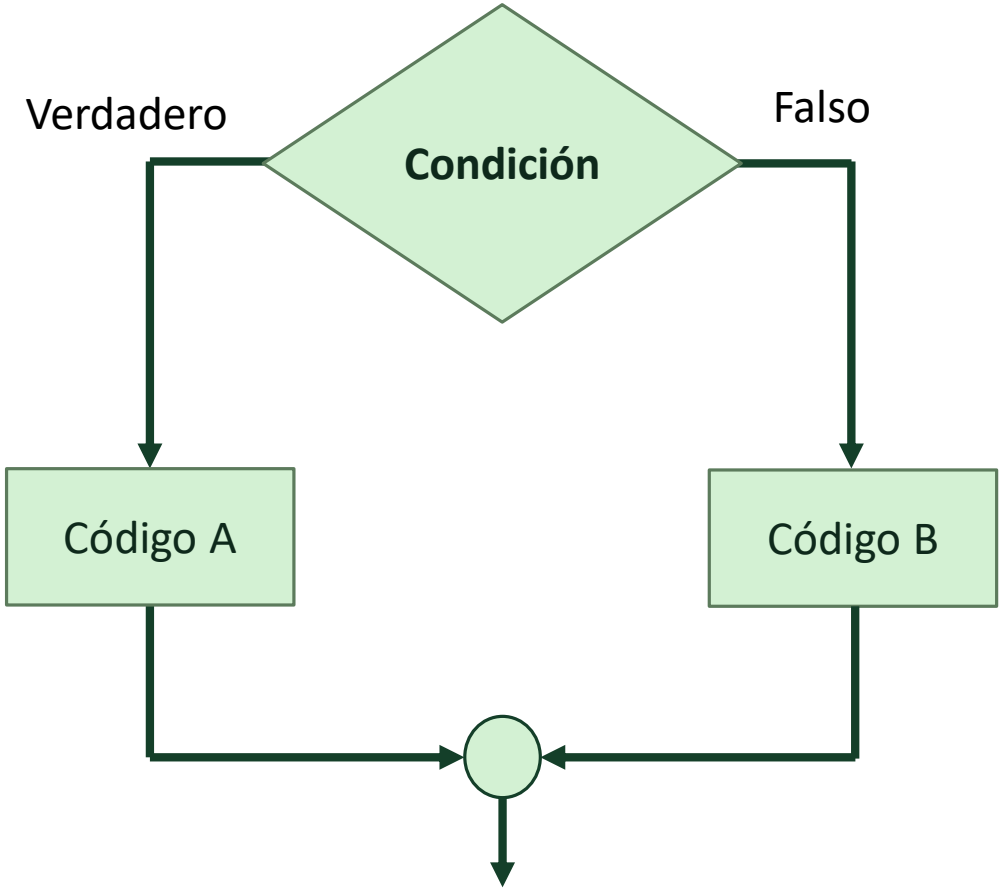
```
1  public class Circunferencia {
2      public static void main(String[] args) {
3          int radio = 3;
4          final double PI = 3,141592;
5          System.out.println(2*PI*radio);
6      }
7  }
```

En Java, el principio y el final de una secuencia se marca con **{** y **}**

La separación de instrucciones se realiza con **;**

ESTRUCTURAS DE SELECCIÓN

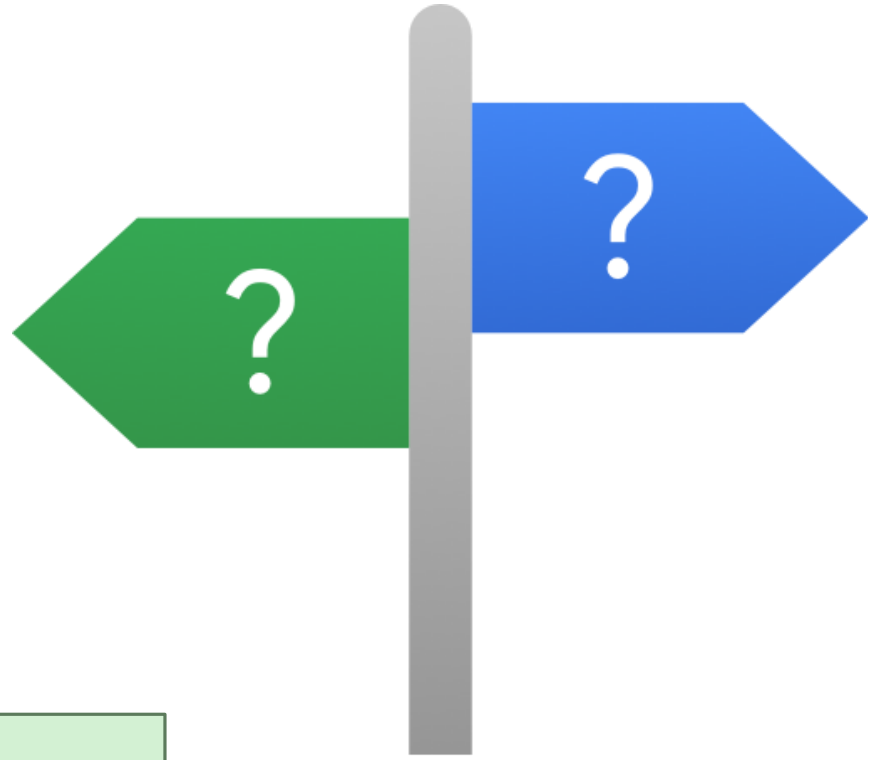
Plantea la selección entre dos alternativas en base al resultado de evaluar una condición



- ***If***
- ***If-Else***
- ***If-ElseIf-Else***
- ***Switch***

Estructura **IF**

```
Si (condición)
    Sentencias_Condicion
Si_no
    Sentencias_NoCondicion
Fin_Si
```



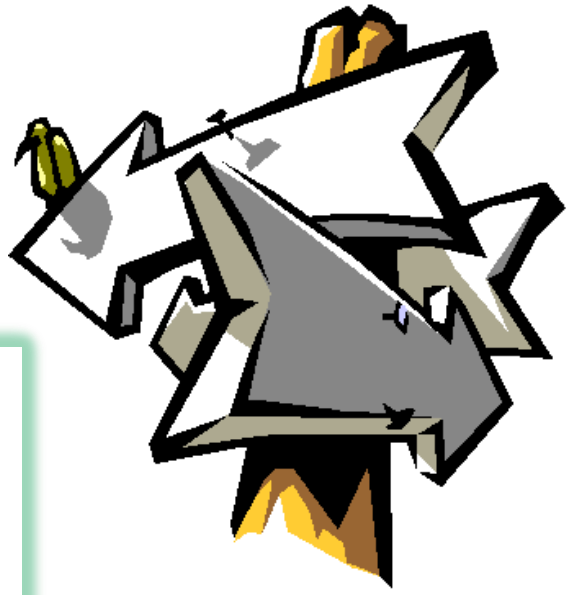
En Java:

```
if (condición) {
   sentencias CONDICION = TRUE
}
else {
   sentencias CONDICION = FALSE
}
```

Estructura IF

```
1  public class Notas{
2      public static void main(String[] args) {
3          int notaAprobado = 5;
4          int notaProgramacion = 7;
5          if( notaProgramacion >= notaAprobado ){
6              System.out.println("Aprobado");
7          }
8      }
9  }
```

Estructura **IF-ELSE**



```java
public class Notas{
    public static void main(String[] args) {
        int notaAprobado = 5;
        int notaProgramacion = 7;
        if( notaProgramacion >= notaAprobado ){
            System.out.println("Aprobado");
        }else{
            System.out.println("Suspenso");
        }
    }
}
```

Estructura **IF-ELSEIF-ELSE**

```java
public class Notas{
    public static void main(String[] args) {
        int notaProg = 7;
        if( notaProg >=0 && notaProg<5 ){
            System.out.println("Suspenso");
        }else if(notaProg >=5 && notaProg<7 ){
            System.out.println("Bien");
        }else if(notaProg >=7 && notaProg<9 ){
            System.out.println("Notable");
        }else{
            System.out.println("Sobresaliente");
        }
    }
}
```

Estructura **IF** : Ejemplo

Diseñe un algoritmo que decida si un número leído desde teclado es par o impar:

Algoritmo Par-Impar

Datos:

numero: Entero

Instrucciones:

numero ← leer un valor entero desde el teclado
Si el resto de dividir el número entre 2 es 0
Mostrar "El número es par"
Sino
Mostrar "El número es impar"
Fin_si

Fin_AlgoritmoParImpar

Estructura IF : Ejemplo

```
import java.util.Scanner;
public class ParImpar {

  public static void main(String[] args){
  //DATOS
    final Scanner TECLADO = new Scanner(System.in);
    int numero;
    String textoSalida, resultado;

  //INSTRUCCIONES
    System.out.println("Este programa decide si el entero introducido es par o impar\n");

    //obtención y/o inicialización de los datos
    System.out.println("Introduce un número entero"); numero =
    TECLADO.nextInt();

    //procesamiento de los datos y generación de resultados
    textoSalida = "El número " + numero + " es ";
    if (numero % 2 == 0){
              resultado = "par";
    }else{
             resultado = "impar";
    }
    textoSalida = textoSalida + resultado + ".\n\nFin del programa...";

    //muestra los resultados
    System.out.println(textoSalida);
  }
}
```

Estructura **IF-ELSE** : Ejemplo

Diseñe un algoritmo que decida si un número leído desde teclado es mayor, menor o igual a cero:

```
Algoritmo Decidir
Datos:
    numero: Entero
Instrucciones:
    numero ← leer un valor entero desde el teclado
    Si numero = 0
        Mostrar "El número es cero"
    Sino
        Si numero > 0
            Mostrar "El número es positivo" Sino
        Mostrar "El número es negativo" Fin_Si
    Fin_si
Fin_AlgoritmoDecidir
```

Estructura IF-ELSE : Ejemplo

```
import java.util.Scanner;
public class DecisionSimple {
  public static void main(String[] args){
    final Scanner TECLADO = new Scanner(System.in);
    int numero;
    String textoSalida, textoResultado;
    System.out.println("Este programa decide si el entero  introducido es mayor,
                        menor o igual a 0\n");

    System.out.println("Introduce un número entero");
    numero = TECLADO.nextInt();

    textoSalida = "El número ";
    if (numero == 0)
       textoResultado=" es cero";
    else if (numero > 0)
       textoResultado = numero + " es positivo";
    else textoResultado = numero + " es negativo";
    textoSalida = textoSalida + textoResultado + ".\nFin  del programa...";

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

Estructura **Switch**

```
Según (expresión) Hacer:

     valorExpr 1: bloque de sentencias 1
     valorExpr 2: bloque de sentencias 2
        ...
     valorExpr n: bloque de sentencias n

     resto: bloque se sentencias resto

Fin_según
```

En Java:

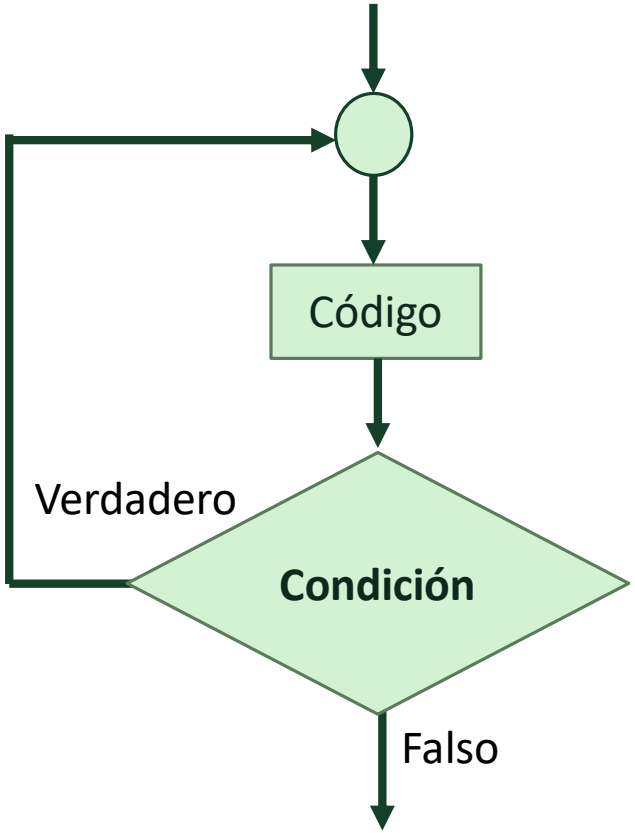
```
switch (expresión) {
   case literal1:
     Bloque sentencias para = literal1
     break;
   case literal2:
     Bloque sentencias para = literal2
     break;
   ...
   default:
```

Estructura **Switch**

```java
public class Carrera{
    public static void main(String[] args) {
        int posicionCarrera = 2;
        switch(posicionCarrera){
            case 1: System.out.println("Oro");
                    break;
            case 2: System.out.println("Plata");
                    break;
            case 3: System.out.println("Bronce");
                    break;
            default: System.out.println("Sin premio");
                break;
        }
    }
}
```

ESTRUCTURAS DE REPETICIÓN

Ejecuta una serie de instrucciones mientras que se cumple una determinada condición

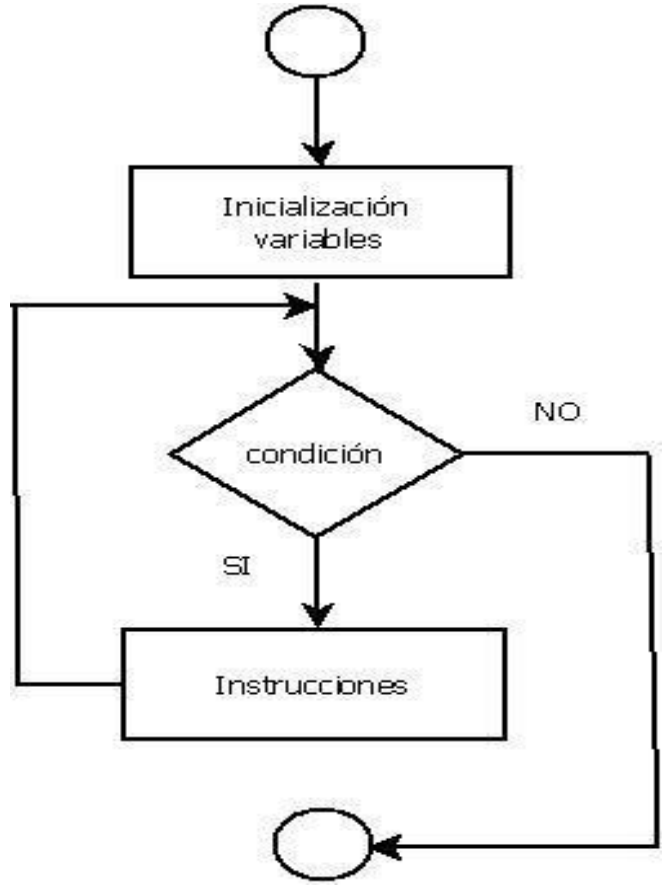


TIPOS:

- Bucle con condición inicial → ***While***
- Bucle con condición final → ***Do-While***
- Bucle con nº de iteraciones finitas → ***For***

Estructura **While** (bucle con condición inicial)

```
Mientras (condición)
      sentencias a repetir si condición = VERDADERO
Fin_Mientras
```



En Java:

Expresión de tipo **boolean**

```
while (condición) {
      sentencias a repetir
      mientras condición = VERDADERO
}
```

Estructura **While** (bucle con condición inicial)

```java
public class Numero{
    public static void main(String[] args) {
        int numero = 1;
        while( numero <= 3){
            System.out.println(numero);
            numero++;
        }
    }
}
```

Estructura **While** : Ejemplo

Algoritmo que controla que el número que se introduce es positivo.

N: Entero

Mostrar ("Introduzca un número positivo")

N ← leer el valor de N del teclado

Mientras (N <= 0) Hacer

Mostrar ("Valor introducido incorrecto. Vuelva a intentarlo")

N ← leer el valor de N del teclado

Fin_Mientras

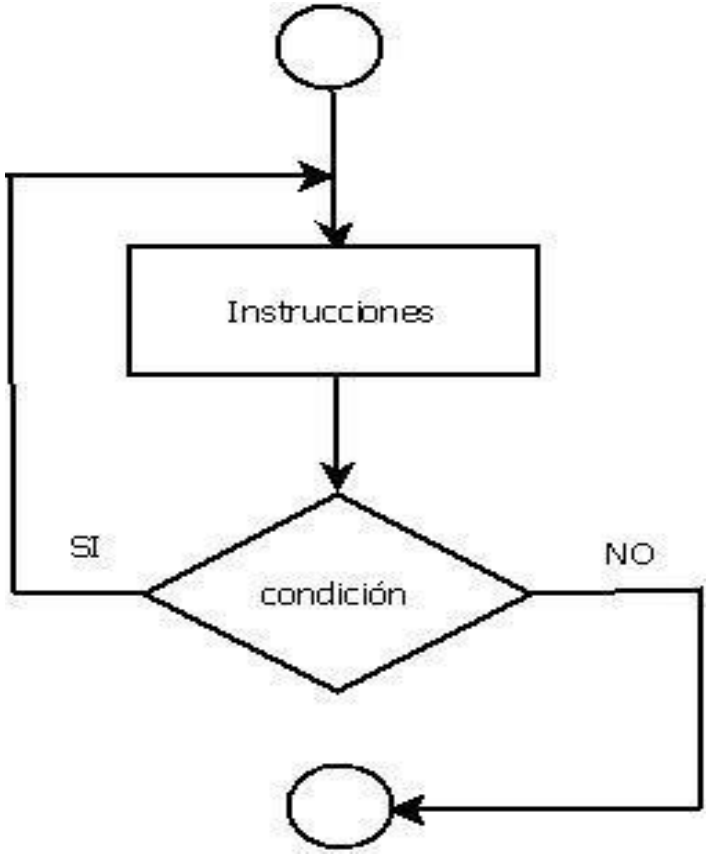
Mostrar ("Ahora sí lo ha hecho bien. Gracias!!")

```
int n;

System.out.println("Introduzca un número
                    positivo");

n = TECLADO.nextInt();

while (n <= 0){

    System.out.println("Valor incorrecto.
                        Vuelva a intentarlo");

    n = TECLADO.nextInt();

}

System.out.println("Ahora sí lo ha hecho bien.
                    Gracias!!");
```

Estructura **Do-while** (bucle con condición final)

(Se ejecuta al menos una vez)

```
Repetir
      sentencias a repetir mientras condición = VERDADERO
Mientras (condición)
```



En Java:

```
do {
     sentencias a repetir

     mientras condición = VERDADERO
} while (condicion);
```

Expresión de tipo **boolean**

Estructura **Do-while** (bucle con condición final)

```
1  public class Numero{
2      public static void main(String[] args) {
3          int numero = 1;
4          do {
5              System.out.println(numero);
6              numero++;
7          }while( numero <= 3);
8      }
9  }
```

Estructura **Do-while** : Ejemplo

Algoritmo que calcula si es par el número introducido por teclado hasta que se mete un cero que termina el programa.

N: Entero

Repetir

Mostrar (“Introduzca un nº entero. (0 para acabar)”)

N ← leer el valor de N del teclado

Si (N es par)

Mostrar (N, “Es par”)

Sino Mostrar (N, “Es impar”)

Mientras (N≠0)

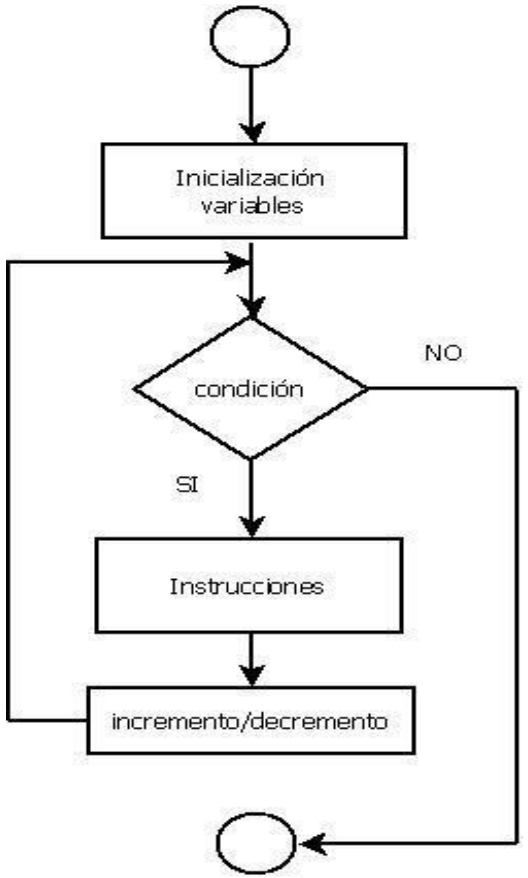
Mostrar (“Fin del programa…”)

```
int n;

do{

   System.out.println("Introduzca un n°
                  entero.(0 para acabar");

   n = TECLADO.nextInt();

   if (n % 2 == 0)

            System.out.println(n + "es par");

   else System.out.println(n + "es impar");

}while (n != 0); System.out.println("Fin del

programa");
```

Estructura **For** (bucle con nº de iteraciones finitas)

(Se ejecuta un número fijo de veces)

```
Para variable desde valor_inicial hasta valor_final incremento n Hacer
        sentencias a repetir [(valor_final-valor_inicial)+1]/n veces
Fin_Para
```



En Java:

```
for (inicialización; condición; incremento){
     sentencias a repetir
}
```

- **Inicialización** de una o varias variables
- **Condición**. Expresión de tipo *boolean* en función de las variables iniciales
- **Incremento** (positivo o negativo) de las variables iniciales

Estructura **For** (bucle con condición final)

```
1  public class Numero{
2      public static void main(String[] args) {
3          int numero = 1;
4          for(numero=1;numero<=10;numero++){
5              System.out.println(numero);
6          }
7      }
8  }
```

ESTRUCTURAS DE SALTO

Se desaconseja el uso de este tipo de sentencias, ya que rompe con los principios de la programación estructurada. Son sentencias que OBLIGAN a hacer determinada acción.

➢ Break

- Sirve para estructuras de repetición y selección
- El programa saldrá del bloque donde se esté ejecutando al encontrar un break
- Solo es adecuado usarlo dentro de un switch

➢ Continue

- Se utiliza en estructuras de repetición
- Termina la iteración en la que se está (i) y continua con la siguiente (i+1)

CADENAS
CLASE STRING

La clase String tiene implementados métodos muy útiles que podemos utilizar en el desarrollo de nuestros programas, por ejemplo:

- **charAt**
  - Retorna el carácter especificado de un String

```
1  String cadena = "hola";
2  char caracter = cadena.charAt(3);
3  System.out.println(caracter); // imprime a
```

- **equals**
  - Compara dos Strings
  - Los String no se comparan con ==

```
1  String nombre = "Pedro", apellido = "Perez";
2  if(nombre.equals(apellido)){
3      System.out.println("Son iguales");
4  }
```

- **length**
  - Devuelve la longitud de un String

```
1  String cadena = "hola";
2  System.out.println(cadena.length());
3  // imprime 4
```

PROGRAMACIÓN MODULAR

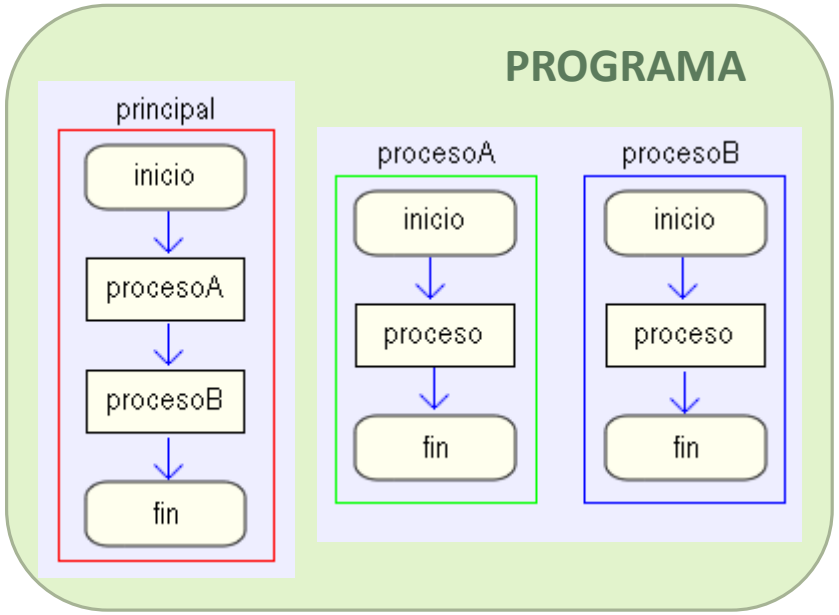
La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Idea principal: Agrupar un conjunto de instrucciones mediante un nombre, para posteriormente utilizar esa "funcionalidad" tantas veces como se quiera sin necesidad de repetir código.

Por ejemplo:

```
System.out.println ("Introduce un valor entero positivo");
n=lector.nextInt();
while(n<1){
        System.out.println ("Error. El valor tiene que ser positivo");
        n=lector.nextInt();
}
```

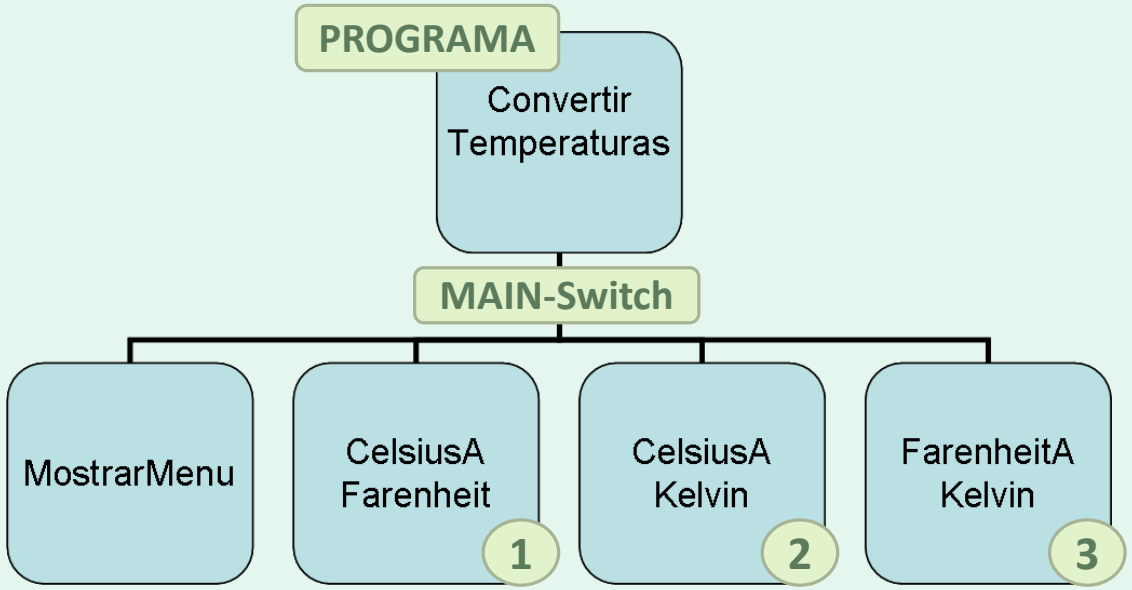
IntroducirEnteroPositivo()



EJEMPLO

Ejercicio:

Diseña un algoritmo MODULAR para convertir los grados de Celsius a Farenheit, de Celsius a Kelvin y de Farenheit a Kelvin a partir de una cantidad dada introducida por el usuario.



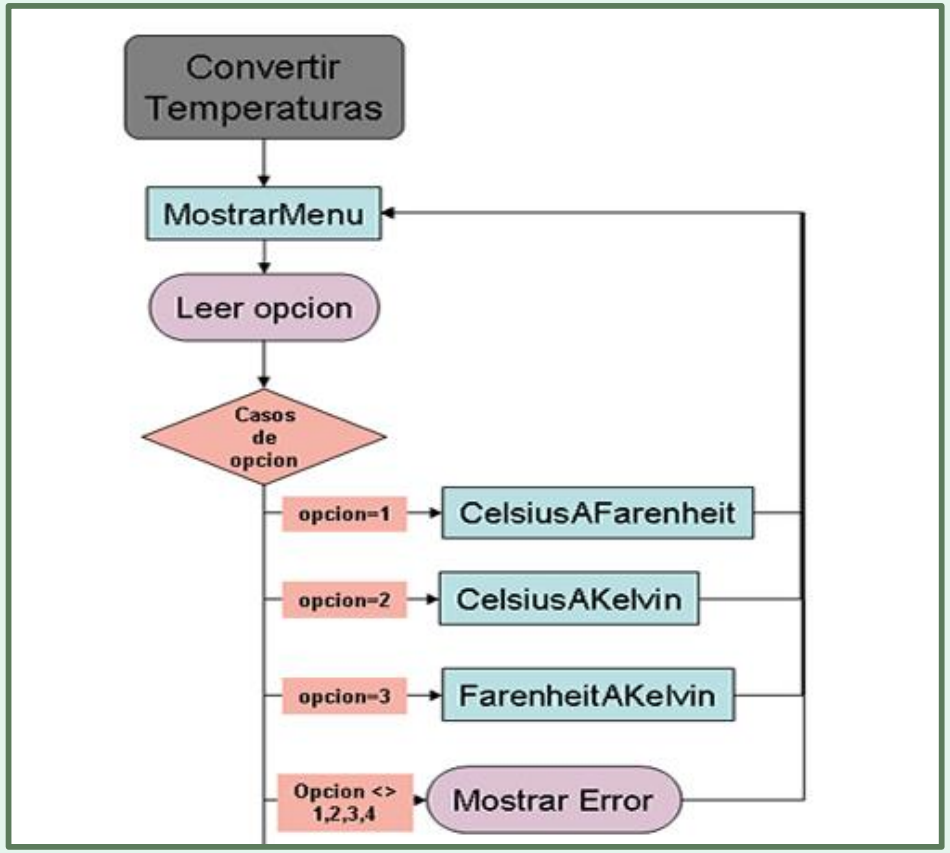

```java
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
       double grados;
       int opcion;
       Scanner lector = new Scanner(System.in);
       System.out.println("Introduce los grados a convertir:");
       grados=lector.nextDouble();

       //Imprimir Menú
       opcion=lector.nextInt();
       switch(opcion) {
              case 1:
                 //CelsiusAFarenheit
                 break;
              case 2:
                 //CelsiusAKelvin
                 break;
              case 3:
                 //FarenheitAKelvin
                 break;
              default:
                 System.out.println("Error. Opción incorrecta");
                 break;
       }
   }
}
```

EJEMPLO

En Java:

Tendremos que implementar pequeños trozos de código que implementen la funcionalidad concreta o la acción que se pide:

```java
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
       double grados;
       int opcion;
       Scanner lector = new Scanner(System.in);
       System.out.println("Introduce los grados a convertir:");
       grados=lector.nextDouble();

       //Imprimir Menú
       opcion=lector.nextInt();
       switch(opcion) {
              case 1:
                 //CelsiusAFarenheit
                 break;
              case 2:
                 //CelsiusAKelvin
                 break;
              case 3:
                 //FarenheitAKelvin
                 break;
              default:
                 System.out.println("Error. Opción incorrecta");
                 break;
       }
   }
}
```



EJEMPLO

Diseñamos pequeñas funcionalidades dentro de trozos de código a los que le damos un nombre para poder reutilizarlos cuando queramos. En ellos tenemos que indicarle los valores que necesitaremos para que funcione y diseñar el algoritmo necesario para crear esa función:

En Java:

```java
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
      double grados;
      int opcion;
      Scanner lector = new Scanner(System.in);
      System.out.println("Introduce los grados a convertir:");
      grados=lector.nextDouble();

      //Imprimir Menú
      opcion=lector.nextInt();
      switch(opcion) {
            case 1:
               //CelsiusAFarenheit
               break;
            case 2:
               //CelsiusAKelvin
               break;
            case 3:
               //FarenheitAKelvin
               break;
            default:
               System.out.println("Error. Opción incorrecta");
               break;
      }
   }
}
```

```java
public static void ImprimirMenu() {
   System.out.println("¿Que conversión desea hacer?");
   System.out.println("1. De Celsius a Farenheit");
   System.out.println("2. De Celsius a Kelvin");
   System.out.println("3. De Farenheit a Kelvin");
}
```

```java
public static void CelsiusAFarenheit(double numero) {
   double grados;
   grados=numero*1.8+32;
   System.out.println(numero + "ºC son: " + grados + "ºF");
}
```

```java
public static void CelsiusAKelvin(double numero) {
   double grados;
   grados=numero+273;
   System.out.println(numero + "ºC son: " + grados + "ºK");
}
```

```java
public static void FarenheitAKelvin(double numero) {
   double grados;
   grados=(numero-32)/1.8 + 273;
   System.out.println(numero + "ºF son: " + grados + "ºK");
}
```

EJEMPLO

```
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
       double grados;
       int opcion;
       Scanner lector = new Scanner(System.in);
       System.out.println("Introduce los grados a convertir:");
       grados=lector.nextDouble();

       ImprimirMenu();
       opcion=lector.nextInt();
       switch(opcion) {
          case 1:
              CelsiusAFarenheit(grados);
              break;
          case 2:
              CelsiusAKelvin(grados);
              break;
          case 3:
              FarenheitAKelvin(grados);
              break;
          default:
              System.out.println("Error. Opción incorrecta");
              break;
       }
   }//fin main

   public static void ImprimirMenu() {
       System.out.println("¿Que conversión desea hacer?");
       System.out.println("1. De Celsius a Farenheit");
       System.out.println("2. De Celsius a Kelvin");
       System.out.println("3. De Farenheit a Kelvin");
   }
```

Tenemos 4 métodos en el programa:

- *Main: Método principal (método de arranque)*
- *ImprimirMenu*
- *CelsiusAFarenheit*
- *CelsiusAKelvin*
- *FarenheitAKelvin*

En el main se "llama" a los métodos poniendo su nombre y entre paréntesis los valores necesarios que se han indicado en su definición para que funcionen:

CelsiusAFarenheit(Double);

```
   public static void CelsiusAFarenheit(double numero) {
      double grados;
      grados=numero*1.8+32;
      System.out.println(numero + "ºC son: " + grados + "ºF");
   }

   public static void CelsiusAKelvin(double numero) {
      double grados;
      grados=numero+273;
      System.out.println(numero + "ºC son: " + grados + "ºK");
   }

   public static void FarenheitAKelvin(double numero) {
      double grados;
      grados=(numero-32)/1.8 + 273;
      System.out.println(numero + "ºF son: " + grados + "ºK");
   }

}//fin class
```

EJEMPLO (sin módulos)

```
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
       double grados;
       int opcion;
       Scanner lector = new Scanner(System.in);
       System.out.println("Introduce los grados a convertir:");
       grados=lector.nextDouble();

       System.out.println("¿Que conversión desea hacer?");
       System.out.println("1. De Celsius a Farenheit");
       System.out.println("2. De Celsius a Kelvin");
       System.out.println("3. De Farenheit a Kelvin");
       opcion=lector.nextInt();

       switch(opcion) {
          case 1:
              grados=numero*1.8+32;
              System.out.println(numero + "ºC son: " + grados + "ºF");
              break;
          case 2:
              grados=numero+273;
              System.out.println(numero + "ºC son: " + grados + "ºK");
              break;
          case 3:
              grados=(numero-32)/1.8 + 273;
              System.out.println(numero + "ºF son: " + grados + "ºK");
              break;
          default:
              System.out.println("Error. Opción incorrecta");
              break;
       }
   }//fin main
```

En este caso, la optimización no es muy evidente porque realmente estas funcionalidades tan sólo se utilizan una vez, pero en un programa repetitivo que se pueda aislar una funcionalidad dentro de un trozo de código si se vería claro el ejemplo y reduciría muchas líneas de código.

Aunque la legibilidad del código si que es mucho más clara en el programa modulado que en el programa estructurado, ya que al darle un nombre característico a la acción queda definido y si se quiere ver lo que se hace para obtener el resultado correcto, podríamos ir a esa parte del código concreta.

En la programación modular, el main queda limpio y sin código adicional, ese código se quedaría en los propios módulos auxiliares.

BENEFICIOS

- Mantenimiento más sencillo
  - Mejora la legibilidad
  - Facilita la depuración, corrección y prueba de los programas
- Permite la reutilización de código
- Favorece al trabajo en equipo
- Simplifica el diseño
- Disminuye la complejidad de los algoritmos

```java
import java.util.Scanner;
public class ConvertidorApp {

   public static void main (String [] args) {
       double grados;
       int opcion;
       Scanner lector = new Scanner(System.in);
       System.out.println("Introduce los grados a convertir:");
       grados=lector.nextDouble();

       ImprimirMenu();
       opcion=lector.nextInt();
       switch(opcion) {
          case 1:
              CelsiusAFarenheit(grados);
              break;
          case 2:
              CelsiusAKelvin(grados);
              break;
          case 3:
              FarenheitAKelvin(grados);
              break;
          default:
              System.out.println("Error. Opción incorrecta");
              break;
       }
   }//fin main

   public static void ImprimirMenu() {
      System.out.println("¿Que conversión desea hacer?");
      System.out.println("1. De Celsius a Farenheit");
      System.out.println("2. De Celsius a Kelvin");
      System.out.println("3. De Farenheit a Kelvin");
   }
```

Módulo: Unidad **independiente** y generalmente pequeña que realiza completamente una **tarea** y se comunica con otros elementos del sistema a través de parámetros y/o a través de la **devolución de un valor** después de su ejecución.

- Se identifica mediante un nombre
- Pueden usar otros módulos. Es decir, puede ser invocado desde el programa principal (main) o incluso desde otros módulos.
- Cuando se invoca el módulo, se ejecutan sus acciones. Al acabar éstas, se devuelve el control del programa al punto en el que se invocó.

Ámbito de las variables

Parte del programa en las que es accesible una variable

Locales

- Las definidas dentro de un módulo
- Su ámbito es **sólo el módulo** en el que están definidas: se crean al comenzar su ejecución y dejan de existir cuando ésta finaliza
- Facilitan la independencia entre módulos

Globales

- Las definidas dentro del programa principal
- Su ámbito abarca **todo el programa**

Variables **locales al main.** No se puede acceder a ellas desde los demás métodos

Variable **global**. Se puede acceder a ella desde cualquier parte del programa

```
import java.util.Scanner;
public class VocalesYConsonantes {
    static final Scanner TECLADO = new Scanner(System.in);

  public static void main(String[] args) {
    // DATOS
      int N,M;
      char[][] mo, mv, mc;

    //INSTRUCCIONES
      //obtener datos de entrada
      N = leerEnteroPositivo();
      M = leerEnteroPositivo();
      mo = cargarDatosMatriz(N,M);

      //calcular los resultados
      mv = obtenerVocales(mo);
      mc = obtenerConsonantes(mo);

      //mostrar los resultados
      mostrar(mo);
      mostrar(mv);
      mostrar(mc);
    }
}
```

```
int leerEnteroPositivo(){
      int res = TECLADO.nextInt();
      while (res <= 0){
        System.out.println("Error. Inténtelo otra vez");
        res = TECLADO.nextInt();
      }
      return res;
}
```

Variable **local al método.** No se puede acceder desde fuera del método

```
void mostrar(char[][] m){
      for (int f = 0; f < m.length; f++){
        for (int c = 0; c < m[f].length; c++)
            System.out.print(m[f][c]+" ");
        System.out.println();
      }
}
```

Tipos de módulos

FUNCIONES

→ **Devuelve un resultado**

esPrimo() : boolean
sumar() : double

MÉTODOS

→ **No devuelve nada**

mostrarMenu()
imprimirResultados()

```
import java.util.Scanner;
public class VocalesYConsonantes {
    static final Scanner TECLADO = new Scanner(System.in);

  public static void main(String[] args) {
    // DATOS
       int N,M;
       char[][] mo, mv, mc;

    //INSTRUCCIONES
       //obtener datos de entrada
       N = leerEnteroPositivo();
       M = leerEnteroPositivo();
       mo = cargarDatosMatriz(N,M);

       //calcular los resultados
       mv = obtenerVocales(mo);
       mc = obtenerConsonantes(mo);

       //mostrar los resultados
       mostrar(mo);
       mostrar(mv);
       mostrar(mc);
    }
}
```

Devuelve un resultado

```
int leerEnteroPositivo(){
       int res = TECLADO.nextInt();
       while (res <= 0){
        System.out.println("Error. Inténtelo otra vez");
        res = TECLADO.nextInt();
       }
       return res;
}
```

No devuelve nada

```
void mostrar(char[][] m){
       for (int f = 0; f < m.length; f++){
         for (int c = 0; c < m[f].length; c++)
             System.out.print(mv[f][c]+" ");
         System.out.println();
       }
}
```

Ideas a tener en cuenta

Cada módulo realiza una tarea

Cada módulo es independiente de los demás. Su código se debe poder implementar sin tener en cuenta el resto del programa

Hay que pensar si necesita datos con los que empezar a trabajar (datos de entrada) y qué datos proporciona como resultado (datos de salida)

Los datos de salida se proporcionan a quien invoca al módulo

Si necesita realizar una tarea que realiza otro módulo, lo invoca directamente, sin pensar si el módulo está bien implementado o no, eso hay que evaluarlo en el otro módulo correspondiente

En JAVA:

- Los módulos se denominan métodos.

- Sintaxis:

```
modificador tipo_resultado nombre_método (tipo parámetro, …) {
        //BLOQUE SENTENCIAS CÓDIGO
}
```

  - **Modificador**: (de momento, utilizaremos **static**)
  - **tipo_resultado**: tipo de datos del resultado que proporciona el método. Si no devuelve nada, se indica **void**.
- El método **main** es el que se ejecuta cuando se invoca a un programa Java
- Los métodos que devuelven un resultado deben hacerlo mediante **return**