Unidad de trabajo 2. Metodología de la programación

1.- Métodos para la representación o el diseño de algoritmos:
   A) Pseudocódigos
   B) Diagramas de flujo u ordinogramas
   C) Diagramas de Nassi-Schneiderman
   D) Tablas de decisición

2.- Metodología de resolución de problemas

3.- Técnicas de programación:
   - Programación convencional
   - Programación estructurada
   - Programación modular
   - Programación orientada a objetos

4.- Programación estructurada
   - Introducción.
   - Estructuras básicas de control de flujo
   - Estructuras avanzadas de control de flujo.
   - Controladores de bucle

5.- Programación modular
   - Introducción
   - Módulos
   - Tipos de módulos
   - Parámetros de módulos
      A) Métodos de comunicación entre módulos: variables globales y parámetros
      B) Tipos de parámetros
      C) Paso de parámetros
   - Variables globales y locales
   - Módulos recursivos

6.- Programación orientada a objetos
   - Introducción
      - ¿Qué es la POO?
      - Características de la POO
   - Clases de objetos
   - Mensajes y métodos
   - Relaciones entre clases de objetos
   - Ejemplo de un programa orientado a objetos

1. METODOS PARA LA REPRESENTACION O EL DISEÑO DE ALGORITMOS.

A) PSEUDOCÓDIGO O NOTACIÓN PSEUDOCODIFICADA: Se puede definir como el lenguaje intermedio entre el lenguaje de programación seleccionado y el lenguaje natural o humano. Esta notación se encuentra sujeta a unas determinadas reglas que nos permiten y facilitan el diseño de algoritmos.

Estructura de un algoritmo en pseudocodigo:

```
Algoritmo Ejemplo
Inicio
        Asignación (v←e)
        Escribir expr1, expr2….
        Leer var1, var2…..
        Bifurcación (saltar a instruccion si expresion)
Fin
```

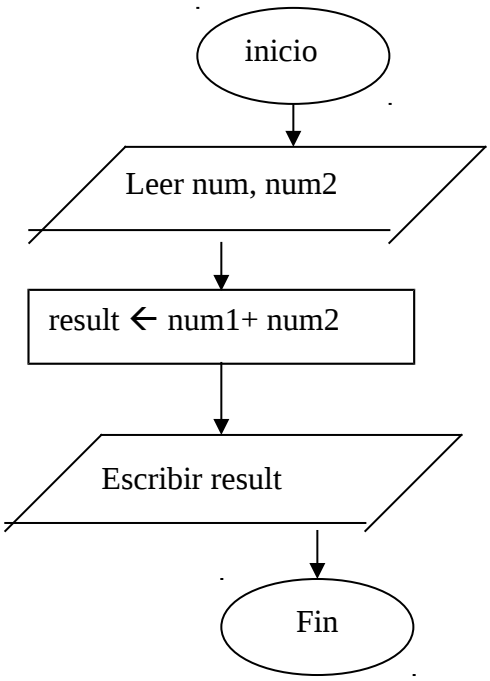
Ejemplo: algoritmo en pseudocódigo que realice la suma de dos números.

```
Algoritmo suma
Entorno: Num1, num2, result  números enteros
Inicio
        Leer num1, num2
        result←num1 + num2
        Escribir result
Fin
```

B) DIAGRAMAS DE FLUJO U ORDINOGRAMAS

Son un método de representación gráfica que mediante el uso de símbolos estándar conectados o unidos mediante líneas de flujo, muestran la secuencia lógica de las operaciones o acciones que debe realizar un ordenador, así como la corriente o flujo de datos en la resolución de un problema.

Ejemplo de algoritmo utilizando la técnica de representación de diagrama de flujo:



Símbolos para representar las acciones primitivas:

Asignación: A←Expresion

Lectura: Leer Var1, Var2

Escritura: Escribir Expr1, Expr2

Bifurcación: Expresion — **SI** / **NO**

Inicio/Fin: Inicio / Fin

C) DIAGRAMAS DE NASSI-SCHEIDERMAN

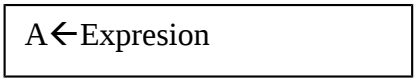
Permite representar el algoritmo en un solo bloque. Tiene el inconveniente de no permitir la corrección de errores.

Ejemplo:

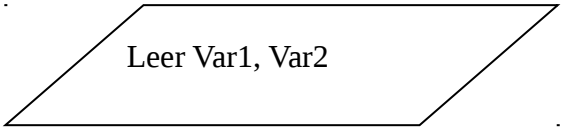
Leer A, B	**(Inicio)**
Producto←A*B	
Escribir Producto	**(Fin)**

Símbolos para representar las acciones primitivas:

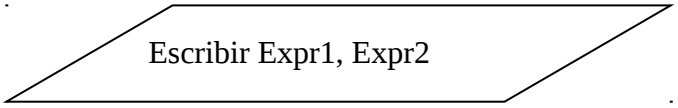
Asignación:



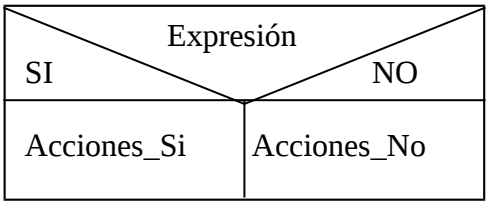
Lectura:



Escritura:



Bifurcación:



Inicio/Fin:

EJERCICIOS:

1. Escribir un algoritmo en pseudocódigo que lea dos números y diga si son iguales.

2. Escribir un algoritmo que lea el nombre de un alumno y sus tres notas. Calcule la media, la escriba y además si es superior a cinco felicite al alumno. Representar el algoritmo en los tres métodos estudiados.

3. Escribir un algoritmo que lea tres números enteros y escriba el mayor de ellos.

4. Escribir un algoritmo que lea dos números y los escriba con los valores intercambiados

5. Escribir un algoritmo que lea un número entero no negativo, que será un periodo de tiempo medido en segundos y devuelva el número de días, horas, minutos y segundos

6. Escribir un algoritmo que lea un número entero no negativo y escriba el factorial de ese número

D) TABLAS DE DECISIÓN

Son herramientas para el diseño de algoritmos que muestran las acciones que se deben ejecutar en nuestro programa cuando se cumplan ciertas condiciones.

Una tabla de decisión presenta cuatro bloques o apartados:

Matriz de condiciones	Entrada de condiciones
Matriz de acciones	Salida de acciones

- Matriz de condiciones: Contiene todas las condiciones del problema que se plantea.
- Matriz de acciones: Refleja todas las acciones a realizar.
- Entrada de condiciones: Muestra las situaciones que se pueden presentar.
- Salida de acciones: Indica el tratamiento a desarrollar para una situación concreta. La indicación se realiza mediante un símbolo x o con un número.
  Una situación es la combinación de valores de todas las condiciones.
  Una regla de decisión es una situación junto con su tratamiento.

Ejercicio: Representar a través de una tabla de decisión el siguiente planteamiento. En un colegio están preparando el viaje de fin de curso al cual solo pueden ir alumnos mayores de edad. El viaje puede realizarse a dos sitios: Santander o a palma de Mallorca. No obstante cuenta con la ayuda de la APA pero solo subvencionan si viajan a Santander. Además todos deben vender polvorones.
En la suscripción de una revista los suscriptores más antiguos 10% y estudiantes20% no se acumulan.

Alumno mayor de edad	S	S	N	N
Viajar a Santander	S	N	S	N
Asociación paga	X			
Vender polvorones	X	X		
No viajan			X	X

2. METODOLOGIA DE RESOLUCION DE PROBLEMAS.

Los pasos que se deben seguir para solucionar un problema son los siguientes:

1. Estudiar el problema

2. Diseño del algoritmo mediante **la técnica de diseño descendente (TOP-DOWN)**, que consiste en un proceso mediante el cual un problema se descompone en una serie de niveles o pasos sucesivos de refinamiento (stepwise), es decir es una forma de abordar la aplicación desde los problemas más generales hasta los más particulares, consiguiendo así una comprensión más fácil de todo el problema y llegando al final de éstos con mayor sencillez y grado de detalle.

3. Verificación del algoritmo; para ello se hace una traza para la cual hay que utilizar datos a los que podemos llamar: normales, extremos y erróneos.

EJERCICIOS:

1. Escribe el algoritmo del siguiete problema mediante una ***tabla de decision*** y en ***pseudocódigo***. En un cursillo de programadores se estudian 4 lenguajes Basic, cobol, Fortran y Pascal. La nota final del curso depende de una práctica que se realiza en los 4 lenguajes, que se clasifican con apto y no apto, según las siguientes normas. Si obtiene apto en los 4 la calificación es sobresaliente; apto en pascal y

en cobol y en otro lenguaje la calificación es notable; apto en cobol y pascal la calificación es bien; apto en pascal y en Fortran o en Basic la calificación es suficiente; el resto de casos insuficiente.

2. Escribir una **tabla de decisión** correspondiente a la situación de una persona para incorporarse o no a la mili. Dependiendo de:

a) se libra en el momento en que ha sido sorteado y siempre que este casado y con hijos
b) si es voluntario se incorpora a filas
c) si sortea, también se incorpora a filas
d) si esta incapacitado se libra
e) espera reemplazo cuando no cumpla ninguna de las anteriores

3. TÉCNICAS DE PROGRAMACIÓN

a) Programación convencional

Ante cualquier problema debemos seguir los siguientes pasos: **análisis del problema, diseño del algoritmo y resolución a través de un computador**. Estos tres pasos constituyen la **programación convencional**, imprimiendo en ella objetivos tales como la concreción, la legibilidad, la depuración de errores y la facilidad en la modificación.

La dificultad de la etapa del mantenimiento en el ciclo de vida de un proyecto surge a la hora de modificar y verificar el programa elaborado durante las anteriores etapas. Pero esto todavía se incrementa cuando la técnica utilizada es la programación convencional. La técnica utilizada era la formación de bucles y rutinas entremezcladas que se llamaban unas a otras, y conseguían el correcto funcionamiento a costa del consumo de recursos, la escasez económica, etc.

La programación **convencional** lleva implícita **poca claridad** y, como consecuencia, **falta de fiabilidad**, sobre todo cuando el algoritmo planteado es grande o complejo.

b) Programación estructurada

Como consecuencia de estos problemas aparece a principios de los setenta un nuevo concepto de programación: la programación estructurada de la mano del profesor E. W. Dijkstra de la universidad de Eindhoven. Consiste en una técnica constructiva de programas basada en un conjunto de reglas que persiguen coordinar las diferentes etapas de programación, utilizando para ello estructuras específicas y optimizando los recursos lógicos y físicos de cada lenguaje de programación.

c) Programación modular

El concepto básico de programación modular consiste en la división de un complejo programa en varios más sencillos. Estos programas sencillos reciben el nombre de módulos y son independientes. Las funciones de estos programas sencillos e independientes no interfieren en el funcionamiento de otros programas. Estos módulos pueden ejecutarse unos a continuación de otros o pueden ser llamados en momentos determinados para su ejecución.

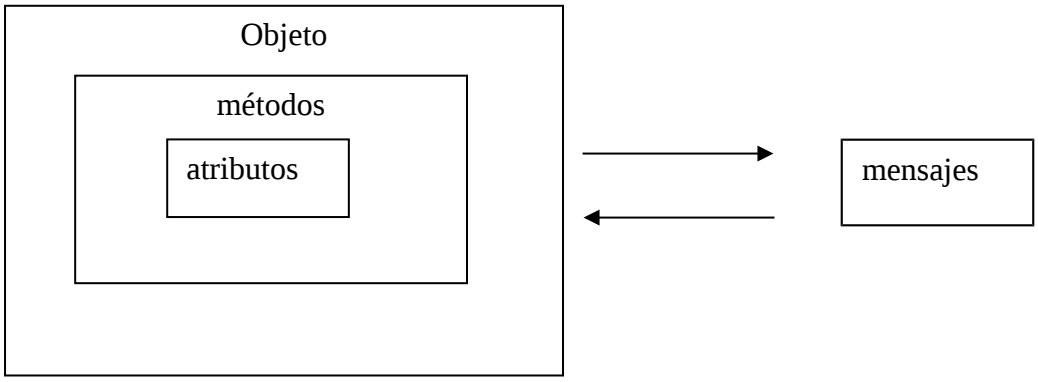
Para conseguir fiabilidad en información (software) debemos hilar cuidadosamente la estructura de un programa. Esto se consigue siguiendo las condiciones siguientes:

- El programa forma una estructura global diferenciándose en ella unos módulos.
- Estos módulos deben estar previamente definidos
- Debe existir conexión entre ellos
- Deben tener todas las estructuras de datos enlazadas, sin dejar punteros abandonados, etc.
- El modulo principal debe estar previamente definido.

Las ventajas del **diseño descendente con descomposición modular** son: legibilidad presente y futura, las modificaciones, el mantenimiento y comprobación de errores son más fáciles.

d) Programación orientada a objetos (POO)

Según lo estudiado hasta ahora, un programa está formado por un conjunto de instrucciones que le indican a la máquina qué hacer, en cambio desde la POO un programa es un conjunto de **objetos** que dialogan entre si, a través de **mensajes**, para realizar las distintas tareas programadas.
Pero, ¿qué son los objetos? Son entidades o elementos del mundo real que poseen unas **propiedades o atributos** y un conjunto de **métodos u operaciones** mediante los cuales muestran su comportamiento.



Ejemplo1: Objeto Persona, formado por los atributos (nombre, edad, grupo sanguineo) y las operaciones (pensar, andar, hablar, identificarse)

En el caso de la operación hablar: persona1.hablar (persona2);

Hablar es el mensaje que envía el objeto persona2 a la persona 1, solicitando hablar con ella, siendo la respuesta a tal mensaje la ejecución del método u operación hablar.

Ejemplo2: Objeto cuenta bancaria tiene unos atributos: nombre, número de cuenta y saldo, y un conjunto de métodos como son: IngresarDinero, RetirarDinero, AbonarIntereses, SaldoActual, Transferencia, etc. En el caso de una transferencia:

cuenta1.Transferencia(cuenta2);

Transferencia sería el mensaje que el objeto cuenta2 envía al objeto cuenta1, solicitando le sea hecha una transferencia, siendo la respuesta a tal mensaje la ejecución del método Transferencia.

Ventajas de la POO:
- Permite modificar los programas durante todas las fases de su desarrollo a bajo coste.
- Reutilización de los objetos, que benefician el aumento de la productividad, disminuye el esfuerzo de mantenimiento y aumenta la fiabilidad y eficiencia de los programas. Por lo tanto, es una técnica que permite mejorar la calidad del software.

4. PROGRAMACION ESTRUCTURADA.

4.1. INTRODUCCIÓN

Razones para la estructuración

La programación estructurada aparece con el propósito principal de minimizar la probabilidad de error en el proceso de programación, es decir **intentar minimizar el error humano**. Para conseguirlo, la programación estructurada parte de varias técnicas o estructuras de programación encaminadas a hacer los programas **más claros**, y en consecuencia **más fiables**.

Cualquier programa largo y complejo siempre se puede desarrollar mediante el anidamiento apropiado de estas estructuras de control.

La programación estructurada no es un criterio contrapuesto a la programación modular sino que es complementario. La programación modular tiende a dividir un programa en partes más pequeñas, llamadas módulos, y la programación estructurada se encarga de desarrollar estructuradamente cada una de estas partes.

La **programación estructurada** es un criterio de programación basado en el **Teorema de la Estrutura de Bohn y Jacopini** que dice lo siguiente: "todo programa propio, es decir aquel que tiene un solo punto de entrada y uno solo de salida, puede ser escrito utilizando únicamente tres tipos de estructuras de control básicas: la estructura **secuencial**, la **condicional** y la **repetitiva**".

Podemos resumir las ventajas de la programación estructurada:
- Permite modificar fácilmente el programa
- Mejorar el programa fácilmente.
- Permite corregir errores
- Permita la comunicación
- La legibilidad

Por tanto la programación estructurada busca una manera de conseguir unos programas más claros y más fiables.

Principios básicos de la programación estructurada:

a) Diseño descendente
b) Estructuras de control de flujos:
   i. Básicas:
      1. Secuencia
      2. Bifurcación, alternativa ó decisión
      3. Iteración, bucle ó ciclo
   ii. Avanzadas:
      1. Alternativa GENERALIZADA
      2. Iterativa REPETIR
      3. Iterativa PARA
      4. Iterativa GENERAL

4.2. ESTRUCTURAS BÁSICAS DE CONTROL DE FLUJO

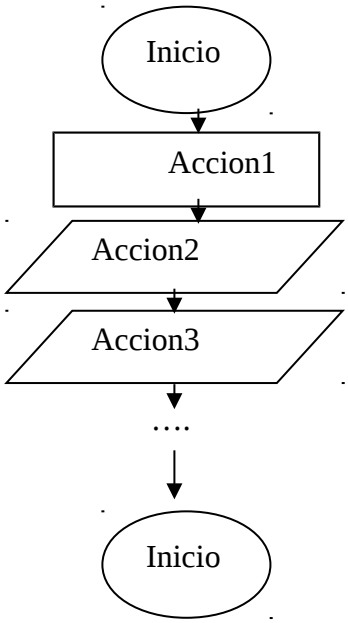
Cada una de estas estructuras vamos a representarlas en tres de los distintos métodos de representación de algoritmos que hemos estudiado: pseudocodigos, diagramas de flujo u organigramas y diagramas de Nassi-Sheiderman

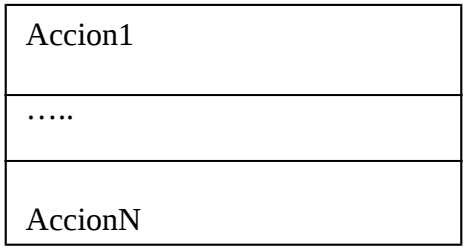
a) Secuencia

Son acciones que se ejecutan una detrás de la otra según el orden en el que aparecen.

(pseudocodigo) (organigrama) (Nassi-Scheiderman)

```
inicio
        accion1
        accion2
        ….
        accionN
fin
```



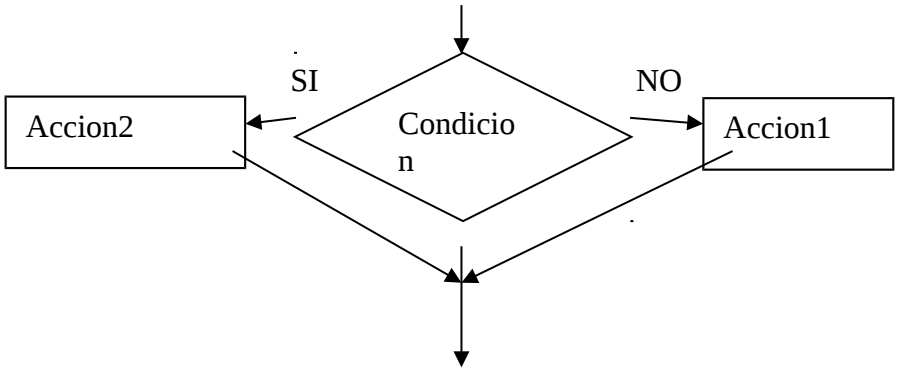


b) Bifurcación, alternativa o decisión

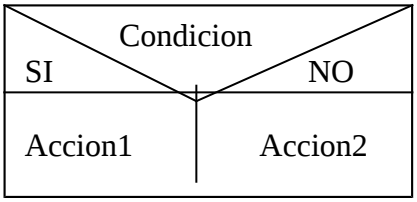
Permite decidir entre la realización de dos acciones a partir de una condición.

(Pseudocodigo) (Organigrama)

```
Si (Condicion) entonces
        Accion1
Sino
        Accion2
finSi
```



(Nassi-Scheiderman)



Ejemplo1: Un algoritmo que lee un número que diga si el número es par o impar.
Inici
Leer num
Si num mod 2= 0
Escribir “par”
Sino escribir “impar”

Ejemplo2: Un algoritmo que lea un número real y muestre su valor absoluto.
Inicio
Leer num
Si num <0
Num←Num*(-1)
Escribir num
Fin

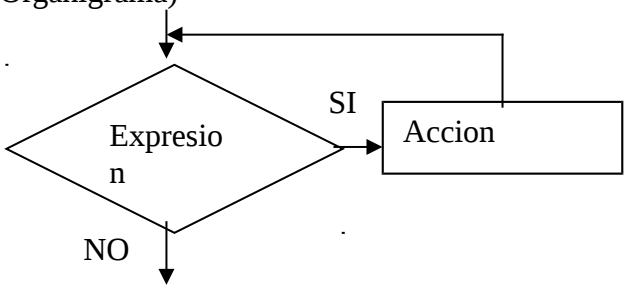
c) Iteración, bucle o ciclo (mientras)

Expresa el número de veces que se ejecutan las acciones determinado por una expresión lógica.

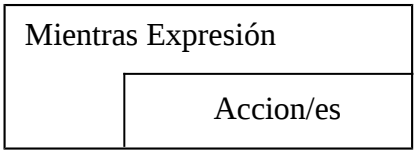
(Pseudocodigo)

Mientras (expresión) hacer
Inicio
Accion
FinMientras

(Organigrama)



(Nassi-Scheiderman)



EJERCICIOS

1. Algoritmo que lea un número positivo y que escriba los números menores que él.
Inicio

2. Algoritmo que lea un número entero positivo y escriba una lista con las potencias de 2 que sean menores o iguales que él.

3. Algoritmo que lea 3 números y devuelva el máximo.

4. Algoritmo que calcule el factorial de un número entero no negativo.

4.3. ESTRUCTURAS AVANZADAS DE CONTROL DE FLUJO

No todos los lenguajes de programación tienen estas estructuras avanzadas.

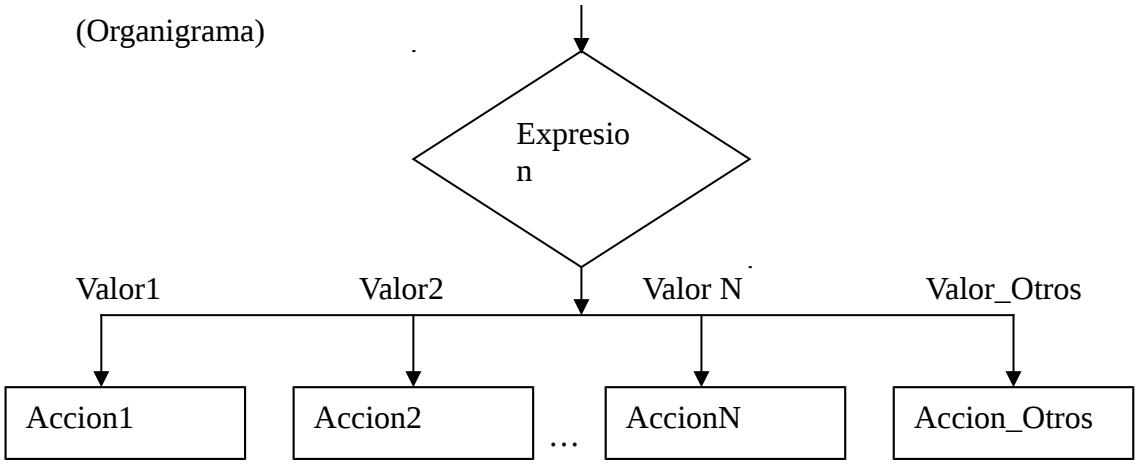
a) Alternativa GENERALIZADA

Nos permite elegir entre una serie de opciones.

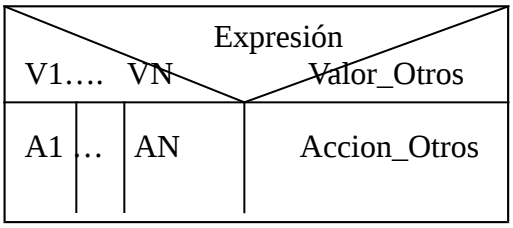
(Pseudocodigo)

Opcion expresión **de**
Valor 1: accion1
Valor 2: accion2
….
Valor N: accionN
Otros: accionOtros
FinOpcion

(Organigrama)



(Nassi-Scheiderman)



Ejemplo: Escribe un algoritmo que lea un número que representa el día de la semana y que devuelva a partir de él el nombre de dicho día

Algoritmo Dia_semana
Inicio
Leer dia
Opcion dia de
1: escribir “Es lunes”
2: escribir “Es martes”
3: escribir “Es miércoles”
….
7: escribir “Es domingo”

```
        Otros: escribir "Error: numero introducido no es valido"
    FinOpcion
Fin
```

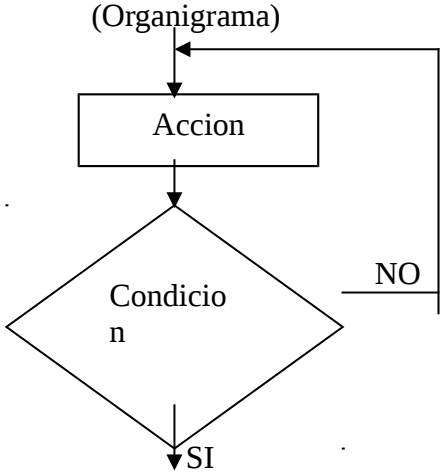
b) Iterativa REPETIR

Permite ejecutar una o varias instrucciones un determinado número de veces. La diferencia de este tipo de instrucción con la instrucción mientras es que en este caso la acción que esta dentro del bucle se realiza como mínimo una vez.

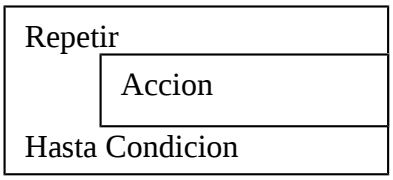
(Pseudocodigo)

```
Repetir
        Accion/es
Hasta Condicion
```

(Organigrama)



(Nassi-Scheiderman)



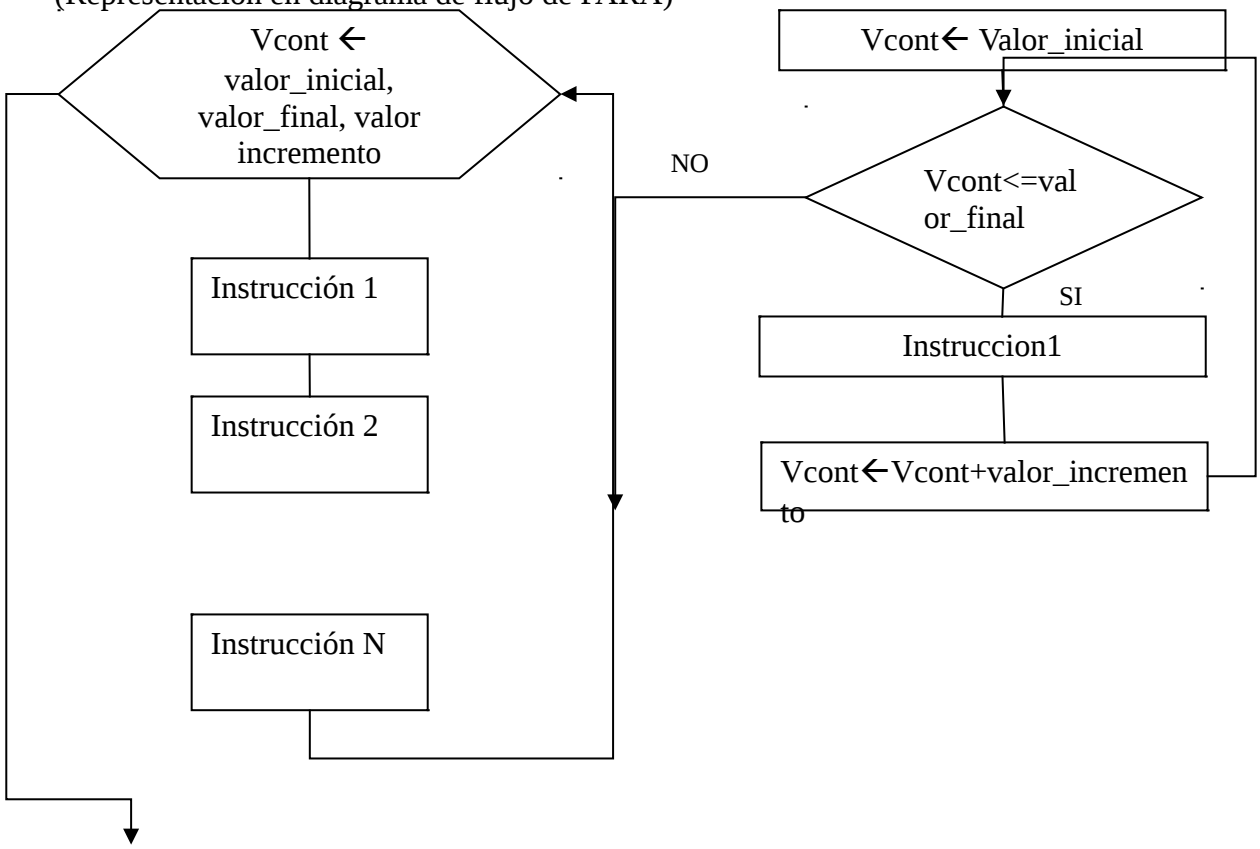
Ejemplo: Escribir un algoritmo que te escriba los primeros cinco números positivos

```
Algoritmo CincoNumeros
Inicio
    Num←1
    Repetir
        Escribir num
        Num←num+1
    Hasta ( num>5  )
```

c) Iterativa PARA

Permite ejecutar un número determinado de veces una determinada acción.

(Representación en diagrama de flujo de PARA)



(Pseudocodigo)

Para Vcont **de** valor_inicial **a** valor_final [incremento valor_incremento] **hacer**
Accion/es
FinPara

Ejemplo: Algoritmo que lea un número y escriba su tabla de multiplicar

```
Algoritmo Tabla_multiplicar
Inicio
    Leer num
    Para  cont=0 hasta cont=10
          resultado← num*cont
          escribir resultado
    finPara
fin
```

d) Iterativa GENERAL

Es parecida a la instrucción de repetir y a la de mientras

(Pseudocodigo)

Iterar
Accion1

Salir si condicion
Accion2
FinIterar

Ejercicios:

1. Realizar el pseudocódigo que permita visualizar un mensaje dependiendo de la tecla que pulsemos:
MENU PRINCIPAL
1.- Procesador de textos
2.- Hoja de cálculo
3.- Base de datos
Introduzca una opción…

2. Realizar el pseudocódigo que permita saber si un número es mayor, menor o igual a 0.

4.4. CONTROLADORES DE BUCLE

Existen tres variables principales controladoras de bucles:
A) CONTADOR
B) ACUMULADOR
C) BANDERA, INTERRUPTOR O FLAG

a) CONTADOR
Se encarga de contar el número de veces que se realiza una acción.
Ejemplo:

```
Contador←Valor_inicial(0)
Mientras (Condicion) hacer
        Accion
        Contador←Contador+Incremento(1)
finMientras
```

Ejercicio: algoritmo que lea un número entero positivo (n) y escriba las n primeras potencias de 2.

b) ACUMULADOR

Variable en la que se almacenan valores intermedios del bucle que se pueden utilizar tanto dentro del bucle como al final.

Ejemplo:

```
Acumulador←Valor_inicial (0 o 1)
Mientras (Expresion) hacer
        Accion
        Acumulador ← Acumulador +Incremento
finMientras
```

Ejercicio: Escribir un algoritmo que lea una serie de números acabada en 0 y calcule la media.

Ejemplo: 3, 2, -2, 1, 0 → 4/4=1

c) BANDERA, INTERRUPTOR O FLAG

Variable de tipo lógico utilizada para saber si se cumple una determinada condición dentro del bucle

Ejemplo:

```
Bandera←valor_inicial (V o F)
Mientras (expresión) hacer
        Accion1
        Si (condicion) entonces
                Bandera←valor_opuesto (F o V)
        finSi
        Accion2
finMientras
```

Ejercicio: Hacer un algoritmo que diga si un número positivo es primo o no.

5. PROGRAMACION MODULAR.

5.1. INTRODUCCIÓN

Ejemplo: Escribir un programa para hacer el combinacional de $\binom{n}{m}$

$$\binom{n}{m} = \frac{n!}{m!(n-m)!}$$

```
Algoritmo Combinacional
 Inicio
        Leer datos
        Calcular_factorial (n)
```

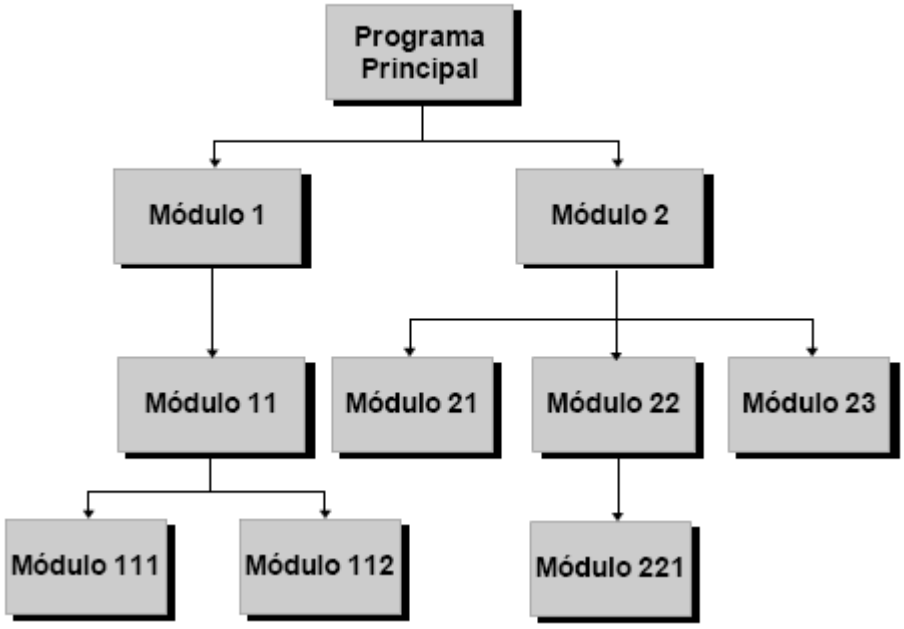
Calcular_factorial(n-m)
Calcular_factorial(m)
Calcular_combinacional
Escribir resultado
Fin

El tener que repetir códigos idénticos implica que el programa sea más complejo (de leer, de verificar). Para evitar esto se utiliza la *programación modular.*

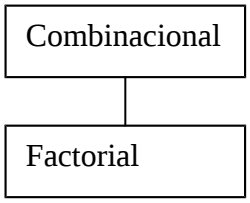
Con esta programación se intenta llevar un mecanismo que evite tener que repetir códigos; los programas constarán de partes (módulos) independientes del resto hasta cierto punto. Otra ventaja es que se consigue una verificación mejor del programa.

OBJETIVO DE LA PROGRAMACIÓN MODULAR: Descomposición de un problema grande en una serie de subproblemas más pequeños y sencillos de resolver que se tratan por separado, independientemente del resto del programa.

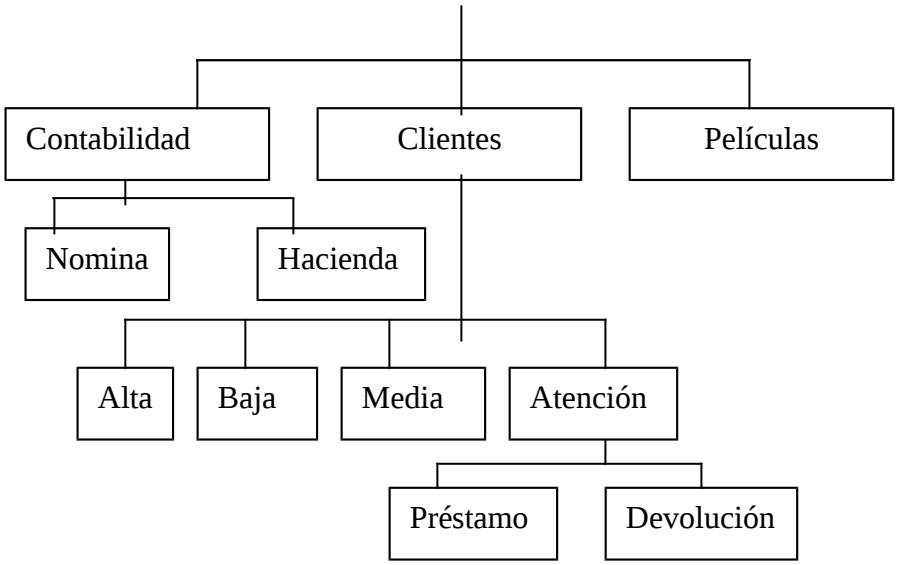
- Para representar los módulos se utiliza el diagrama modular.
- Modulo: Programa sencillo que realiza una determinada función.



Ejemplo1:



Ejemplo2: Programa que lleve la gestión de un video-club



VENTAJAS DE LA PROGRAMACIÓN MODULAR

- Facilita el diseño descendente: Descomposición funcional de los problemas
- Disminuye la complejidad del algoritmo
- Disminuye el tamaño total del programa, al no tener que repetir los mismos códigos.
- Reusabilidad: ahorro de tiempo de programación
- División de la programación entre un equipo de programadores, por tanto reducción del tiempo de desarrollo
- Facilidad en la depuración: comprobación individual de los módulos
- Programas más fáciles de mantener o modificar, puesto que los problemas (módulos) son independientes.
- Estructuración en librerías específicas (biblioteca de módulos): los módulos que tenemos escritos para un problema los podemos utilizar en otros.

5.2 MODULOS

Un modulo es un conjunto de instrucciones que resuelven un problema determinado y que se identifica con un nombre.

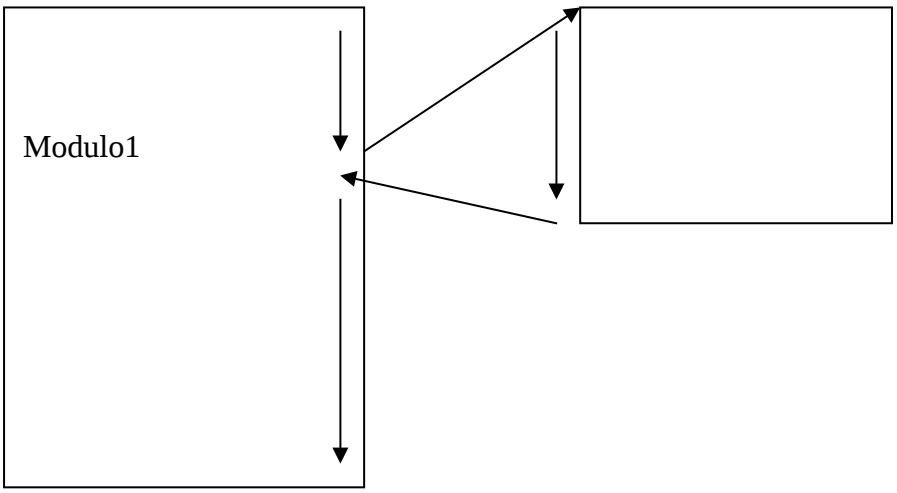
Hay varias formas de denominarlo: modulo, subprograma, rutina, subrutina.

La estructura de un módulo es muy similar a la de un programa (entrada-proceso-salida).

Para ejecutar el módulo hay que realizar una *llamada* al módulo.

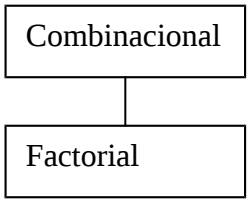
Por ejemplo:

(Combinacional) PROGRAMA	(Factoriall) Modulo1



Explicación del ejemplo: Cuando se produce la llamada al modulo se ejecutan las instrucciones del módulo y luego se sigue o se vuelve al programa.

Cada módulo realiza una función determinada. Por ejemplo: el modulo1: factorial, realiza el factorial de un determinado numero.



El tamaño del modulo no debe ser ni muy grande ni muy pequeño.

5.3. TIPOS DE MODULOS

a) Modulo principal o programa principal: es el modulo que resuelve el problema y se encarga de llamar a otros módulos (secundarios), que son los que realizan alguna función determinada, los cuales pueden llamar a otros módulos.

Por ejemplo:

Combinacional → programa principal
Factorial → modulo secundario

Representación en pseudocódigo:

```
Algoritmo MODULO_PRINCIPAL
 Inicio
  instruccion1
  ……
  instruccionN
 Fin

Modulos_secundarios
```

b) Módulos secundarios: funciones y procedimientos

FUNCION: Una función es un módulo que realiza una serie de funciones y devuelve un valor al módulo que realiza la llamada.

PROCEDIMIENTO: Es un modulo que realiza una serie de funciones pero que directamente no devuelve ningún valor al módulo que realiza la llamada.

Sintaxis del módulo procedimiento:

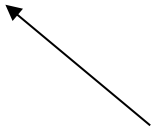
```
PROCEDIMIENTO NombreProcedimiento [(parámetros)]
Inicio
::::::::::
Fin
```

Para ejecutar el procedimiento se escribe su nombre y entre paréntesis los parámetros:

NombreProcedimiento(parámetro)

Sintaxis del módulo función:

```
FUNCION NombreFuncion (parámetros):TipoResultado
Inicio
  …..
  Retornar (Resultado)
Fin
```



- Para que la función devuelve un valor, en alguna parte de su código hay poner esta instrucción
- Para almacenar el valor que devuelve la función utilizaremos una variable: *variable ←NombreFuncion(parametros)*

5.4. PARAMETROS DE MODULOS

Los parámetros constituyen un mecanismo para comunicar los módulos con los datos. Además de los parámetros, otro método de comunicación entre módulos son las variables globales o compartidas.

A) Métodos de comunicación entre módulos:

PRIMER METODO: Variables globales o compartidas

Variable global o compartida: es aquella que se puede utilizar en todos los módulos que constituyen el algoritmo.

Ejemplo: Realizar un algoritmo que lea una serie de números hasta leer un 0 y calcule y muestre el cuadrado de cada uno los números.

```
Variables globales: numero es un entero
Algoritmo Calcular_Cuadrados
Inicio
  Repetir
    Leer numero
    Escribir_Cuadrado
```

```
  Hasta numero=0
Fin

Procedimiento Escribir_Cuadrado
Entorno: cuadrado
Inicio
  Cuadrado←numero*numero
  Escribir “El cuadrado de “, numero, “es”, Cuadrado
Fin
```

SEGUNDO METODO: Parámetros

Parámetros: Datos que se comunican entre el módulo que realiza la llamada y el modulo llamado.

Hay que definir los parámetros a la vez que se define el módulo, puesto que se sitúan detrás del módulo.

Procedimiento NombreProcedimiento (Param1 tipoParam1, Param2 tipoParam2, …ParamN tipoParamN)
Funcion NombreFuncion (Param1 tipoParam1, Param2 tipoParam2, …ParamN tipoParamN): Resultado

Param1, Param2…ParamN: Nombres de variables, que representan los parámetros del procedimiento o función. Contienen los datos utilizados dentro del procedimiento o de la función.

tipoParam1, tipoParam2…tipoParamN: son los tipos de datos de cada uno de los parámetros del procedimiento o función

A los parámetros definidos en la cabecera del módulo se les denomina PARAMETROS FORMALES.

A los parámetros que se utilizan en la llamada del modulo/s se les denomina PARAMETROS ACTUALES.

Ejemplo:

```
Algoritmo Calcular_Cuadrados
Variables: numero es entero
Inicio
  Repetir
    Leer numero
    Escribir_Cuadrado (numero)        ← parámetro ACTUAL
  Hasta numero=0
Fin

Procedimiento Escribir_Cuadrado (N es entero)        ← parámetro FORMAL
Inicio
 Cuadrado←N*N
 Escribir “Es cuadrado de, “ N, “es”, Cuadrado
Fin
```

El número, orden y tipo de los parámetros debe coincidir en la llamada y en la definición del modulo.

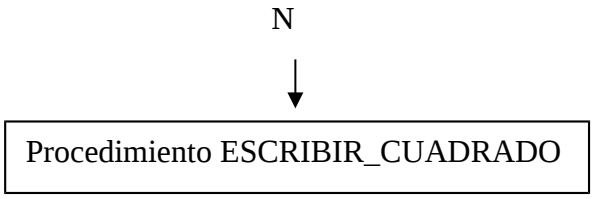
B) Tipos de parámetros:

a. De entrada
b. De salida
c. De entrada y salida

a. DE ENTRADA o parámetros POR VALOR o POR EXPRESION: Permiten enviar información (constante, expresión o variable) del módulo que realiza la llamada al modulo llamado.

La entrada actual puede ser una constante, una expresión o una variable.

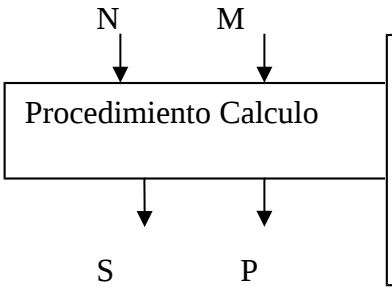
Ejemplo: (el anterior)




```
Procedimiento Escribir_cuadrado (N↓ es un entero)
{
 Escribir N*N
}
```

b. DE SALIDA: Permiten enviar información desde el módulo llamado hasta el módulo que realiza la llamada.

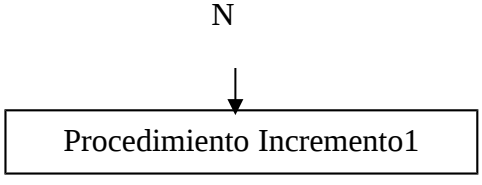
Ejemplo: Modulo que devuelva dos valores (suma(S) y producto(P)) a partir de dos numeros (N, M). El modulo deberá ser un procedimiento porque la función devuelve un solo valor.




```
Procedimiento Calculo (N↓, M↓ , S↑, P↑ : son enteros)
{
  S←N+M
  P←N*M
}
```

c. DE ENTRADA/SALIDA o **PARAMETROS POR REFERENCIA:** Permiten enviar un valor al módulo llamado y devuelven otro valor al modulo que llama.

La entrada/salida actual es un variable.




```
Procedimiento Incremento1 (N↓↑: es entero)
{
  N←N+1
}
```

N

El parámetro de salida es un parámetro de entrada y salida en el que no nos interesa el valor que tenga al principio el parámetro.

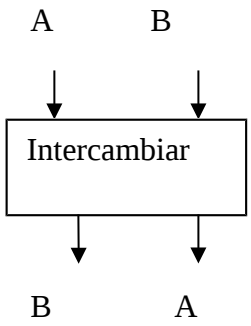
En la cabecera de la definición del módulo se pone delante de los parámetros la palabra VAR, para indicar que son parámetros de entrada y salida.

Por ejemplo:

Procedimiento NombreProcedimiento(VAR Param1 tipoParam1, VAR Param2 tipoParam1, …)

No es conveniente utilizar parámetros de entrada y de salida en una función. La función solo se utiliza para sacar un valor y para más de uno se utiliza los procedimientos.

Por ejemplo: Crear un procedimiento que intercambie dos números A y B.



A y B son dos parámetros de E/S

```
Programa PRINCIPAL
Variables: X, Y son enteros
Inicio
  X←7
  Y ←5
  Intercambiar(X,Y)
  Escribir (X)
  Escribir (Y)
Fin

Procedimiento Intercambiar (VAR A, VAR B: son enteros)
Variables : Aux es un entero
Inicio
  Aux←A
  A←B
  B←Aux
Fin
```

C) Paso de parámetros: Por valor y por referencia

- **POR VALOR**:
  - Se pasa una copia del valor de una expresión del modulo principal al modulo secundario, el cual trabajará con dicha copia del valor pasado.

    Podemos definir una **expresión** como aquello que tiene o devuelve un valor.

    Las expresiones básicas están integradas por constantes, variables y operadores.

    Ejemplos de expresiones:

    z ← 9 La sentencia de asignación es una expresión

    x← not ((z>1) or (z > 30))

  - Se usa cuando únicamente nos interesa el valor, no las modificaciones que pueda tener dentro del modulo secundario.

  - Son parámetros unidireccionales o de entrada, que pasan información desde el modulo principal al modulo secundario. Puede ser cualquier expresión evaluable en ese momento.

  Ejemplo de paso de parámetros por valor

```
Modulo principal
Inicio
  Num←5
  Escribir_Cuadrado(Num)
  Escribir_Cuadrado (5)
  Escribir_Cuadrado (1+3)
Fin

Procedimiento Escribir_cuadrado (N ↓es un entero)
{

    N←N*N
    Escribir N
}
```

- **POR REFERENCIA**: se pasa una referencia a la posición de memoria donde se encuentra dicho valor. Se utilizan tanto para recibir como para transmitir información entre el modulo principal y el secundario. Debe ser obligatoriamente una variable.

Tipo de parámetro	*Paso por*
Entrada	Por valor
Salida Entrada/Salida	Por referencia

Ejemplo de paso de parámetros por referencia:

```
Modulo principal
Variables: Num, suma, producto son enteros
Inicio
  Num←6
  Calculo (5, 3, suma, producto)
  Escribir suma, producto
  Incremento1(Num)
  Escribir Num
Fin

Procedimiento Calculo (N↓, M↓ , S↑, P↑ : son enteros)
{
  S←N+M
  P←N*M
}

Procedimiento Incremento1 (Var N: es entero)
{
  N←N+1
}
```

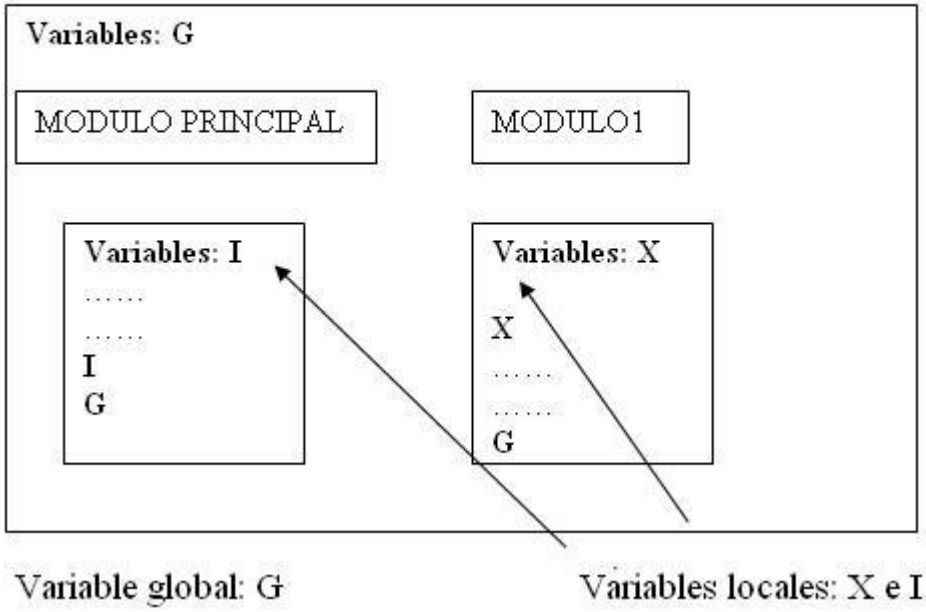
5.5. VARIABLES GLOBALES Y LOCALES

Variable global: es una variable definida fuera de todo módulo (normalmente por encima del módulo principal) y que se utiliza en todos los módulos que constituyen el algoritmo o programa.

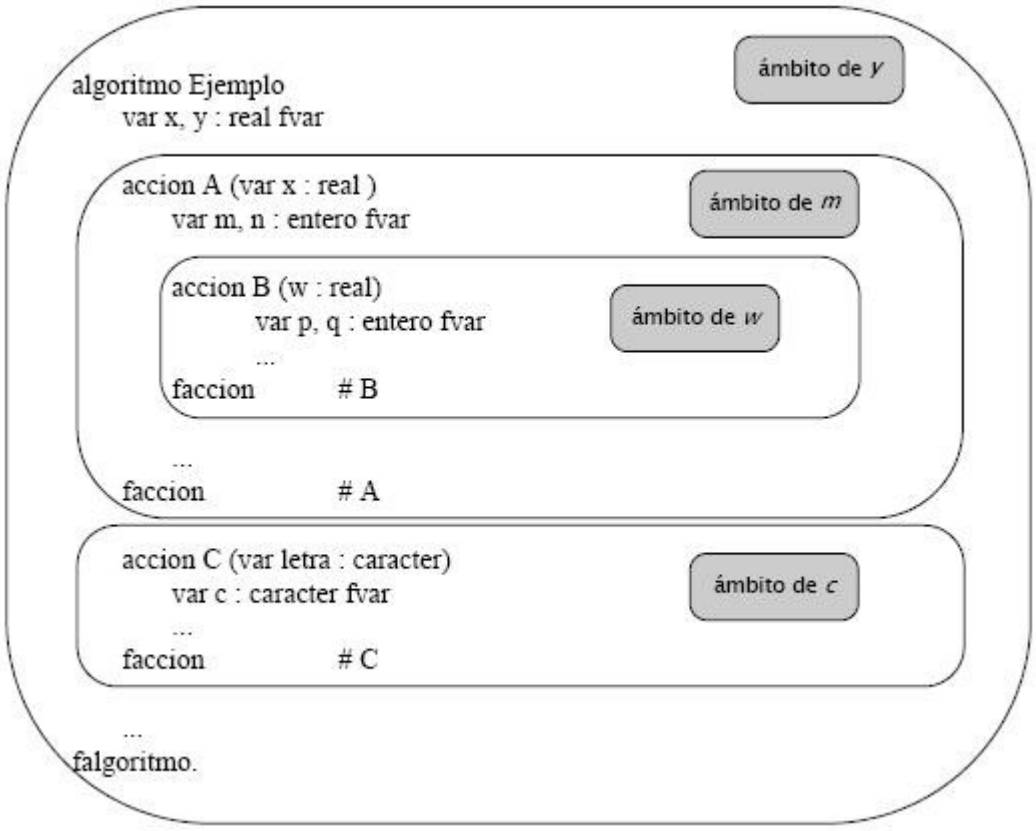
Variable local: es una variable definida en un módulo y que sólo se utiliza en ese módulo.

EJEMPLO1:

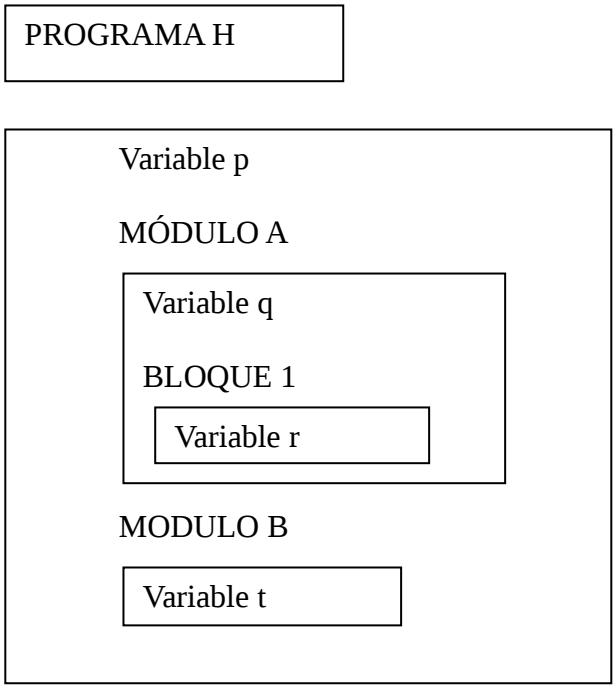
EJEMPLO DE PROGRAMA MODULAR FORMADO POR DOS MODULOS,
y donde se muestra el ámbito de declaración y uso de las variables



EJEMPLO2:



EJEMPLO 3:



VARIABLES		AMBITO
Global	p	A,B
Local	q	A
Local	r	Bloque 1
Local	t	B

Para <u>pasar información del módulo principal a otro modulo secundario</u>, se han <u>de utilizar parámetros y no variables globales</u>, ya que éstas pueden causar el problema de "efecto lateral".

Algoritmo o programa: Efecto_lateral

Variables: Numero1,Numero2, Mayor, Resultado:entero

```
Modulo Principal
Inicio
  Resultado←0
  Leer(Numero1)
  Leer(Numero2)
  Resultado←Numero1+Numero2
  Mayor ← NumMayor(Numero1,Numero2)
  escribir('La suma es' ,Resultado)
  escribir('El mayor es ', Mayor)

Fin

Funcion NumMayor(n1, n2 :enteros): entero
Variables: Resultado es un entero
Inicio
                Si (n1>n2) entonces
                  Resultado←n1
                Sino
   Resultado←n2
  Finsi
```

retornar Resultado
Fin

- **REGLA DE AMBITO DE LAS DECLARACIONES DE VARIABLES:**
  Una variable se puede utilizar en el programa en el que está declarada y en todos sus módulos o subprogramas.

Ejemplo: Algoritmo que lea una serie de números hasta que encuentre un 0 y calcule el factorial.

```
Algoritmo SerieFactoriales
Variables: N es entero
Inicio
  Repetir
    Leer N
    Escribir Factorial (N)
  Hasta N=0
Fin

Funcion Factorial(N es entero): Entero
Inicio
  Fact ←1
  Mientras N>0 hacer
    Fact ←Fact *N
    N ←N-1
  finMientras
 Factorial←Fact
Fin
```

5.6. MODULOS RECURSIVOS

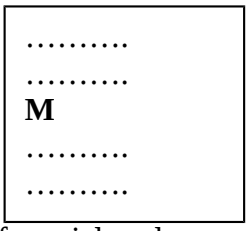
Modulo recursivo: es aquel que en alguna de las instrucciones del cuerpo del módulo incluye una llamada asimismo.

. **¡¡muy importante¡¡** Un módulo recursivo tiene como mínimo una llamada asimismo y además tiene que tener al menos una condición de salida para asegurar que termine.

Características:

- Herramienta muy potente
- Idónea para la resolución de aquellos problemas que pueden definirse de modo natural en términos recursivos
- Tiene su equivalente iterativo
- Necesitan mayor cantidad de memoria para su ejecución
- Son más lentos en su ejecución

Modulo **M**



Ejemplo: La función factorial podemos escribirla de forma recursiva ya que para calcular un valor del factorial utilizamos el factorial.

N! = N * (N-1)! Si N>0
0! = 1

```
Funcion Factorial (N es entero): Entero
Inicio
  Si (N=0) entonces
    Retornar 1
  Sino
    Retornar N*Factorial (N-1)
  FinSi

Fin
```

Ejemplo: Calculo del combinacional

```
Programa Calcular_combinacional
VAR N, M enteras          ←——————Variables locales: N y M
Inicio
Leer N, M
Combinacional←factorial (N) / factorial (N-M) * factorial (M)
Escribir Combinacional
Fin

Funcion Factorial (N es entero): Entero
VAR Fact, Contador        ←————— Variables LOCALES:  Fact y Contador
Inicio
  Fact←1
  Contador←N
  Mientras (Contador<>0) hacer
    Fact←Fact *Contador
    Contador←Contador-1
  FinMientras
  Retornar Fact
Fin
```

6.- PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

- Introducción
  - ¿Qué es la POO?
  - Características de la POO

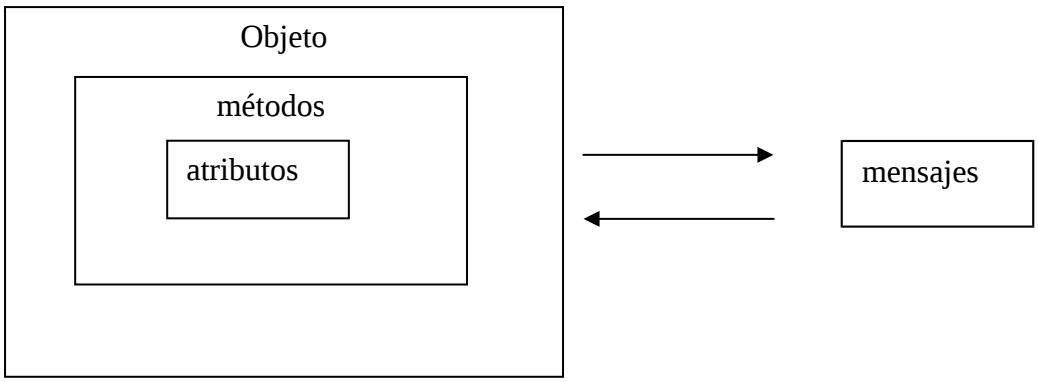
- La interfaz de los objetos
- Clases de objetos
- Mensajes y métodos
- Relaciones entre clases de objetos
- Clases versus objetos
- Ejemplo de un programa orientado a objetos

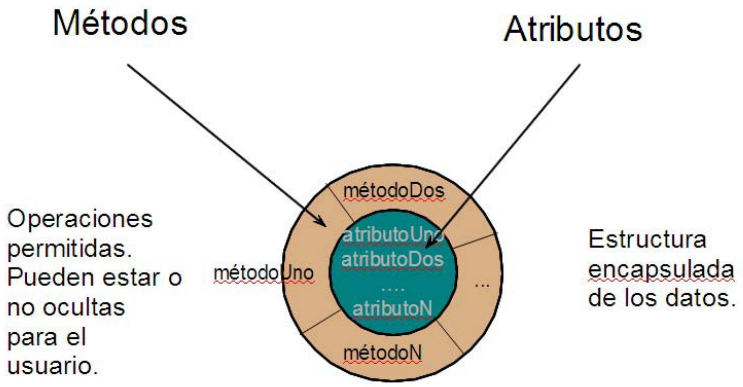
6.1.- INTRODUCCIÓN

6.1.1 ¿QUÉ ES LA POO?

La POO es una técnica de programación que surge para mejorar la calidad de software, basada en organizar cada programa como un conjunto de objetos que dialogan entre si, mediante mensajes, para realizar las distintas tareas programadas. Cada objeto representa una entidad o elemento del mundo real que posee una estructura (propiedades o atributos o datos) y un comportamiento (métodos u operaciones)

Método: es un conjunto de instrucciones que realiza alguna tarea sobre los datos o propiedades del objeto. Los métodos son los procedimientos y las funciones.



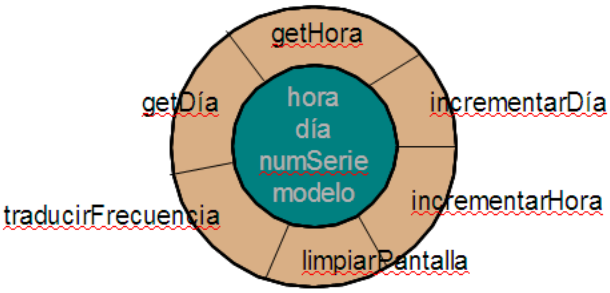


Ejemplo1: objeto **uncoche**

Tiene las **características:**
•Color
•Velocidad
•Tamaño
•Carburante

Funciones que puede realizar:
•Ir
•Parar
•Girar a la derecha
•Girar a la izquierda
•Arrancar

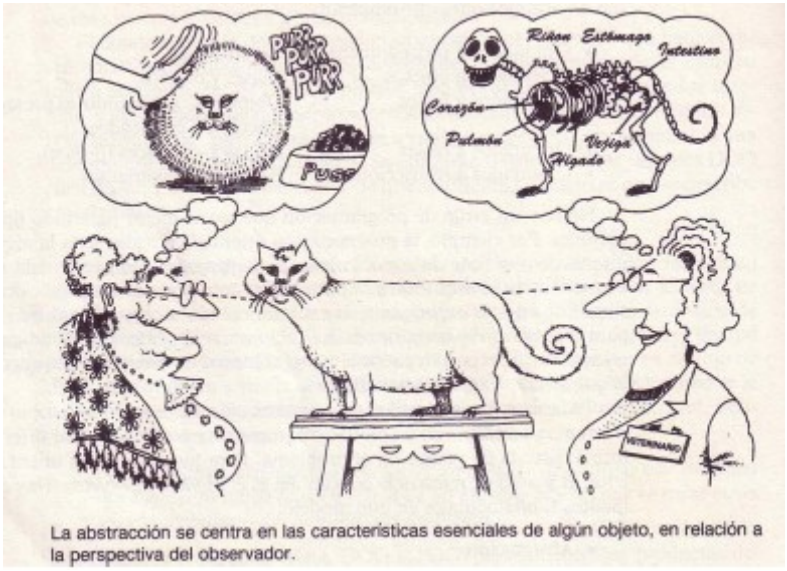
Ejemplo2: Para el objeto **unReloj**, la estructura sería la siguiente:



6.1.2. CARACTERÍSTICAS DE LA POO

Cabe destacar las siguientes:

a) **Abstracción**: Capacidad de definir los datos y operaciones que se necesitan, de las entidades u objetos de nuestro mundo real. Dicho de otra forma, consiste en la generalización conceptual de los atributos y comportamiento de un determinado conjunto de objetos.

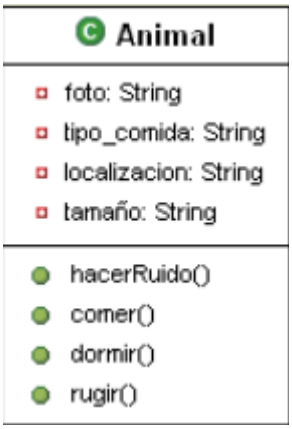


La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

La clave de la programación Orientada a Objetos está en abstraer los métodos y los datos comunes a un conjunto de objetos y almacenarlos en una clase. Así todos los objetos de una clase, se diferenciaran solamente en el estado (valor de los atributos de un objeto), teniendo todos ellos el mismo comportamiento.

Primeramente hay que centrarse en lo que es y lo que hace un objeto (atributos y comportamiento), antes de decidir cómo debería ser implementado. Nos centramos por tanto en la definición, en lugar de la implementación.

Ejemplo de abstracción: En nuestro mundo real, tenemos los siguientes objetos, miGato, miPerro, miLeon, miTigre y miLobo. Si abstraemos los atributos comunes que queremos tener contemplados en el ámbito de nuestra solución, encontramos que en todos ellos, queremos tener una foto, que tipo de alimentación, donde habitan y su tamaño, y como comportamiento, queremos saber como hacen ruido, como comen, como duermen y como rugen. Así de una realidad, hemos abstraído estado y comportamiento y hemos definido la clase Animal.



Más información sobre abstracción

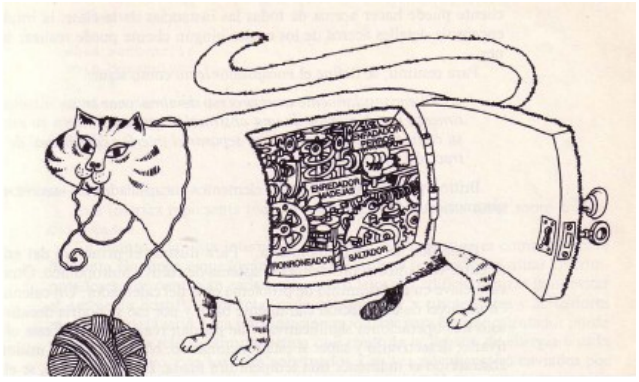
Es una nueva forma de pensar sobre el software, basándose en **abstracciones** que existen en el mundo real. Se basa en la observación de que, en el mundo real, los objetos se construyen a partir de otros objetos.

¿Qué significa **abstracción**?: “Supresión intencionada, u ocultamiento, de algunos detalles de un proceso o artefacto, con el objeto de destacar de manera más clara otros aspectos, detalles o estructuras”

- La abstracción se consigue mediante la descomposición modular atendiendo a funciones o a datos. Tipos de abstracción:
  - Funciones: Descomposición funcional descendente. Enfoque tradicional.
  - Datos: Enfoque orientado a objetos.
- Métodos de abstracción:
  - Parametrización
  - Especificación

b) La **encapsulación y ocultamiento:**

Se tratan los dos paradigmas o caracteristicas de forma conjunta, puesto que se utilizan normalmente de forma simultánea.



Encapsular, significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Dicho de otra forma, consiste en agrupar los datos y métodos (procedimientos y funciones) en una unidad llamada objeto.

Ocultamiento, consiste en separar el aspecto externo del objeto o interfaz, al cual pueden acceder otros objetos, del aspecto interno e implementación del mismo, que es inaccesible para los demás. Permite tratar a un objeto como una caja negra, la cual solo es tratada por el resto de objetos por su interfaz.

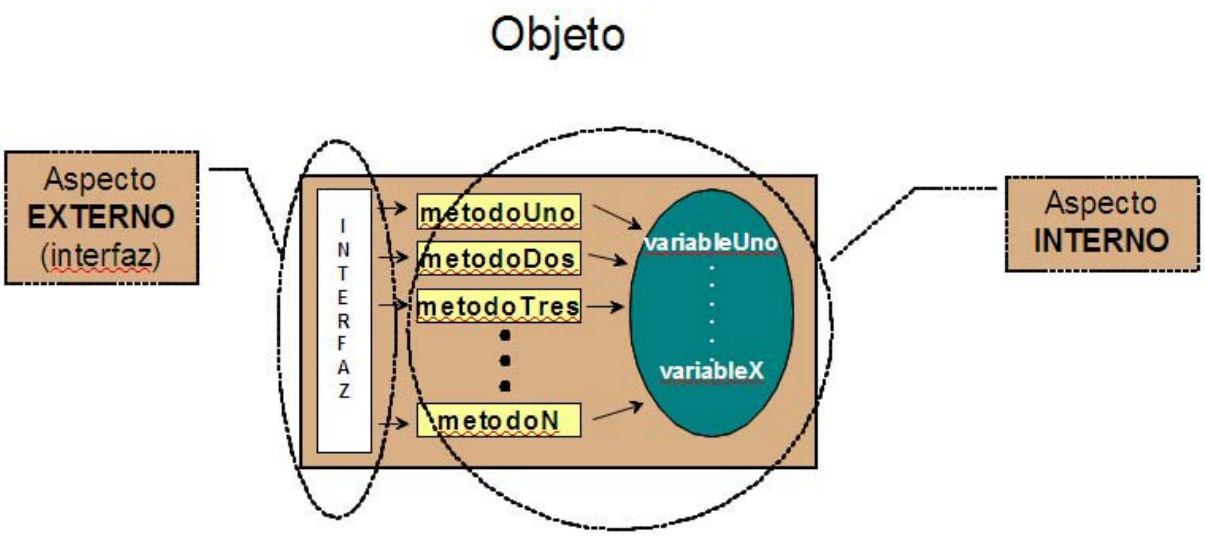
Permite, por tanto que se modifique la implementación interna de un objeto sin afectar a los clientes que lo utilizan. De esta manera, mientras el interfaz no varíe, se puede modificar la implementación o el aspecto interno, sin que los objetos con los que interrelaciona se vean afectados.

c) La **interfaz** de los objetos
d) La **clase**
e) La **herencia**
f) Polimorfismo

6.2. LA INTERFAZ DE LOS OBJETOS

Interfaz: Es el aspecto exterior que es visible al resto de objetos. Puede estar formado por uno o varios métodos.

Cuando se habla del aspecto de los objetos, no nos estamos refiriendo a los conceptos de buen o mal aspecto visual. Nos referimos a como el objeto se ve internamente o aspecto interno y como ven al objeto desde otros objetos también llamado aspecto exterior.



Este aspecto exterior, es llamado también **interfaz**, siendo la parte visible y accesible para el resto de objetos. Puede estar formado por uno o varios métodos. También se le define como el protocolo de comunicación de un objeto.

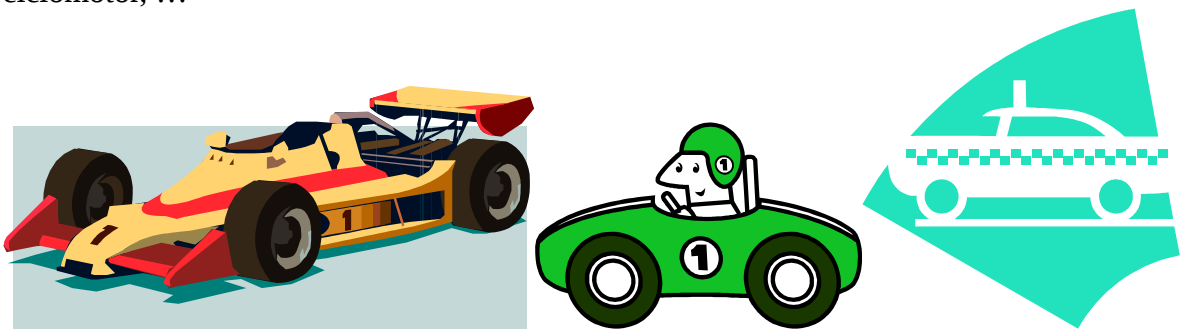
Es posible que exista algún método que solo pertenezca al aspecto interno pero no pertenezca al interfaz. En este caso, estos métodos no pueden ser llamados desde otros objetos, sino que solamente pueden ser llamados desde métodos del propio objeto.

 Ejemplo: Para el objeto unReloj, el interfaz estaría formado por los métodos getHora, getDia, incrementarHora e incrementarDia. Los métodos limpiarPantalla y traducirFrecuencia solamente pertenecen (conjuntamente con los que forman el interfaz) al aspecto interno. Así el método limpiarPantalla, es llamado por getHora y getDia antes de mostrar la información pedida en el método.

6.3. CLASES DE OBJETOS.

Clase de objetos es una estructura que engloba a distintos objetos con propiedades similares y el mismo comportamiento.

Por ejemplo: objetos de la clase vehiculo pueden ser un turismo, camión, coche de carrera, autobús, ciclomotor, …



Ejemplo1: Clase Coche

Clase Coche

Atributos: Color Velocidad Nºasientos Carburante
Metodos: Ir Parar Girar a la derecha Girar a la izquierda Arrancar

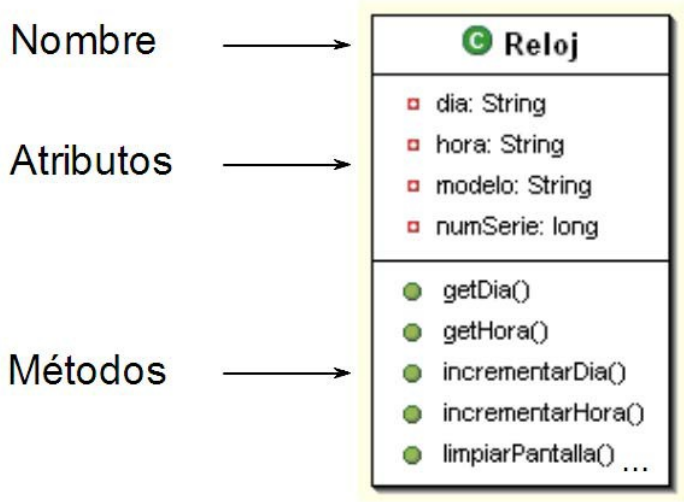
Un objeto (instancia o ejemplar) de una determinada clase se crea en el momento en que se define una variable de dicha clase. Por ejemplo, la siguiente línea declara el **objeto turismo** de la **clase o tipo Coche**

Turismo es de tipo Coche

Una **clase equivale a la generalización de un tipo específico de objetos**, pero **cada objeto que construyamos de esa clase tendrá sus propios datos**.

Turismo.Color=”rojo”
Turismo.Velocidad=”200 Km/h”
Turismo.Nºasientos=5
Turismo.Carburante=”diesel”

Ejemplo2: El objeto unReloj, pertenece a la clase Reloj, cuyo nombre es Reloj, cuyos atributos son dia, hora, modelo y numSerie y cuyos métodos son getHora, getDia, incrementarHora, incrementarDia, limpiarPantalla y traducir.



6.4. MENSAJES Y MÉTODOS

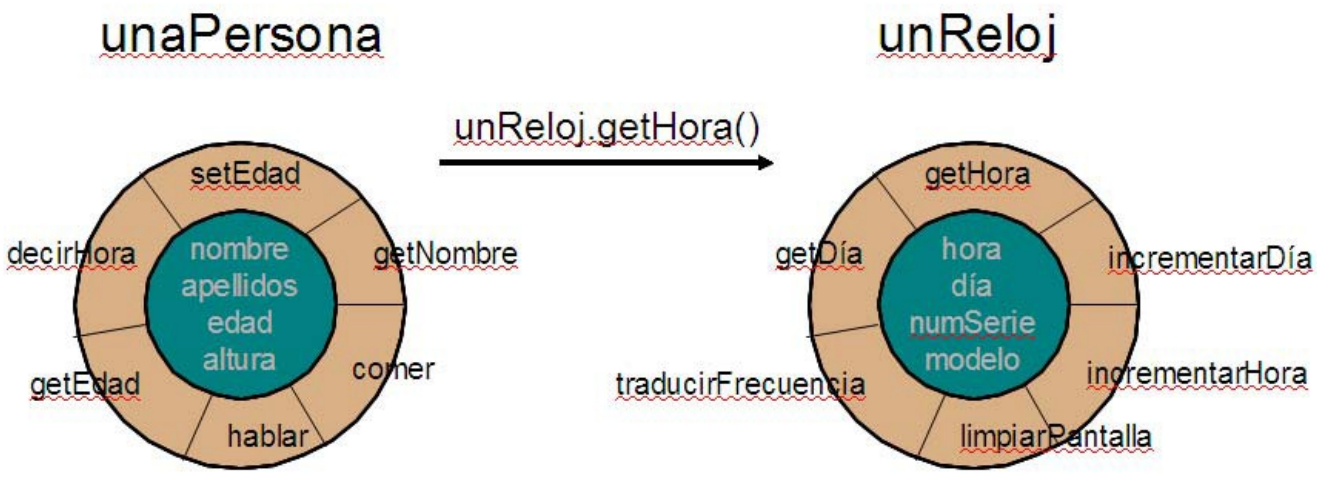
Cuando se ejecuta un programa orientado a objetos, los objetos están recibiendo, interpretando y respondiendo a mensajes de otros objetos. En la POO un mensaje está asociado con un método, de tal

forma que cuando un objeto recibe un mensaje la respuesta a ese mensaje es ejecutar el método asociado.

Un **método** se escribe en una **clase de objetos** y determina cómo tiene que actuar el objeto cuando recibe el mensaje vinculado con ese método. A su vez, un método puede también enviar mensajes a otros objetos solicitando una acción o información. En adición, los atributos definidos en la clase permitirán almacenar información para dicho objeto.

Por **ejemplo**, cuando un usuario quiere maximizar una ventana de una aplicación Windows, lo que hace simplemente es pulsar el botón de la misma que realiza esa acción. Eso provoca que Windows envíe un mensaje a la ventana para indicar que tiene que maximizarse. Como respuesta a este mensaje se ejecutará el método programado para ese fin.

Otro ejemplo de mensaje: El mensaje en este ejemplo es la llamada desde unaPersona al objeto unReloj, para que le de la hora mediante el método getHora. Para ello, el objeto unaPersona tiene que conocer el interfaz de unReloj, para saber que método es el que tiene que llamar, si tiene que pasarle parámetros, de que tipos y si le va a devolver alguna información y una vez más, de que tipo.



Según lo expuesto, podemos decir que la ejecución de un programa orientado a objetos realiza fundamentalmente tres cosas:

1. Crea las clases y los objetos necesarios
2. Los mensajes enviados a unos y a otros objetos dan lugar a que se procese internamente la información
3. Finalmente, cuando los objetos no son necesarios, son borrados

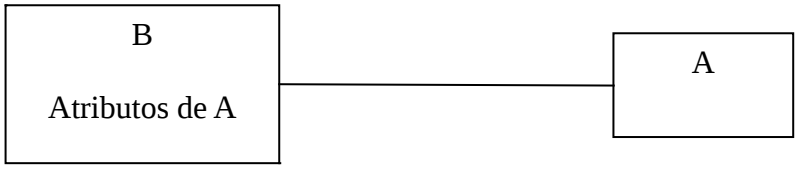
6.5. RELACIONES ENTRE CLASES DE OBJETOS

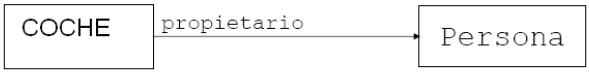
Pueden ser dos:

g) Relación de clientela
h) Relación de herencia

Relación de clientela:

Se dice que una clase B es cliente de otra A, si la clase B puede contener información de la clase A

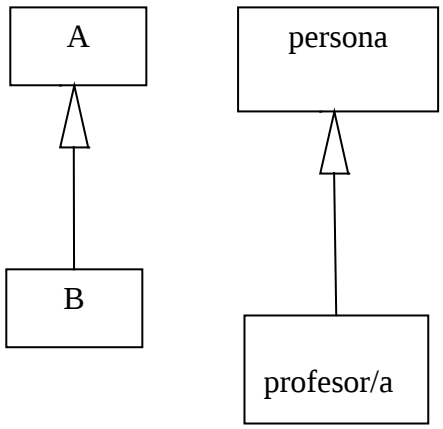




Relación de herencia:

Se dice que una clase B hereda de otra A, si B es una versión especializada o especialización de A.

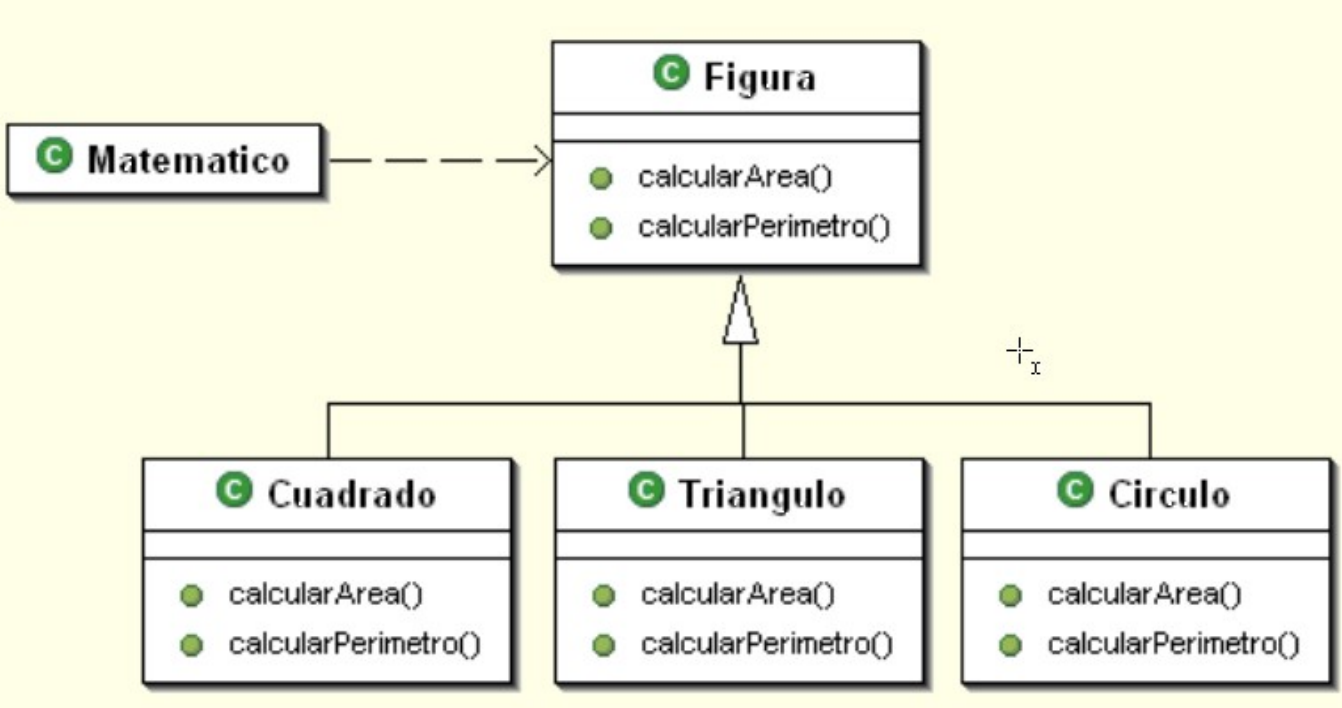
La herencia permite que una clase herede los atributos y métodos de otra clase llamada superclase (los métodos constructores no se heredan). Esta característica garantiza la reutilización del código.

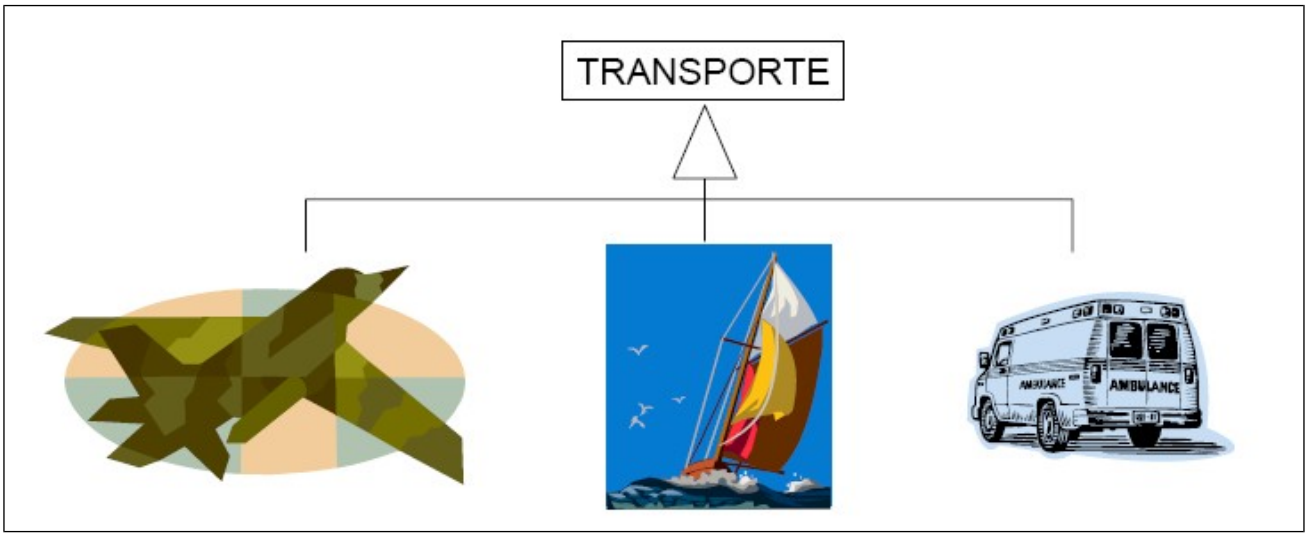


Llamamos **polimorfismo**, al mecanismo que permite redefinir un método heredado de una superclase, dicho de otra forma consiste en utilizar un mismo nombre de método (procedimiento o función) para realizar distintas tareas.

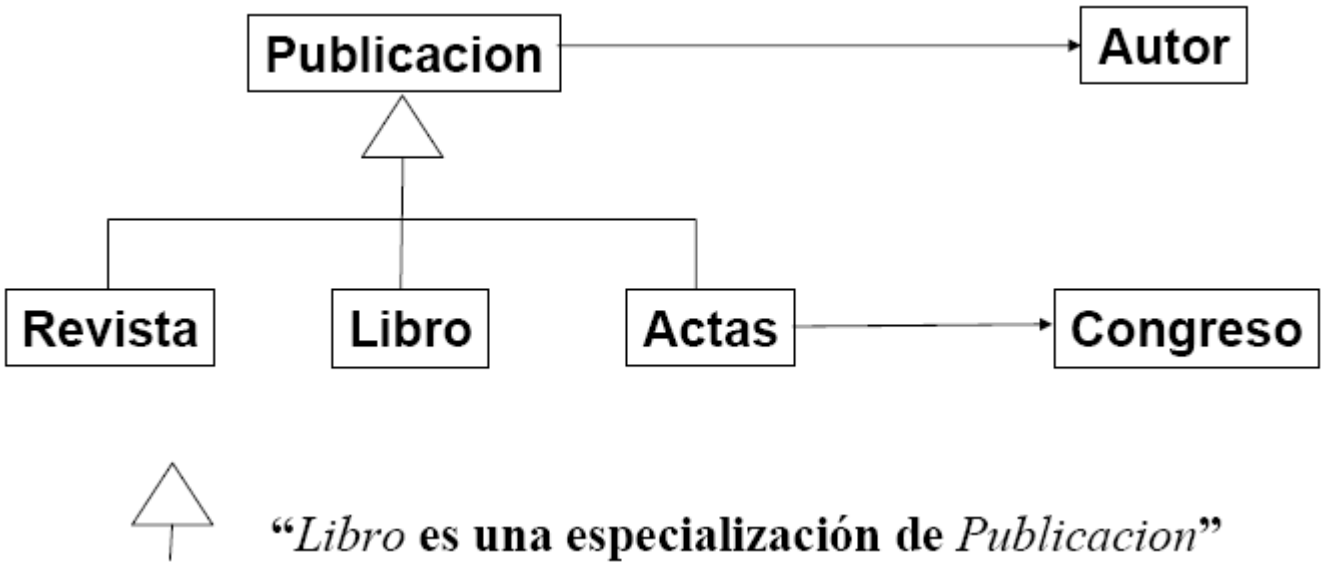
Ejemplo de polimormismo: En el ejemplo del diagrama de clases UM (siglas de Unified modeling language), el Matemático, solo va a tener relación con Figura para calcularArea y calcularPerimetro.

Pero realmente es cada una de las figuras, la que sabe cómo tiene que calcularArea o calcularPerimetro. Por eso, cada una de ellas, Cuadrado, Triangulo y Circulo van a implementar cada uno de los métodos de una manera más especializada que el padre.





Por tanto, se deduce que la estructura de un programa orientado a objetos está formada por objetos de clases con relaciones de clientela y herencia.



6.6. CLASES VERSUS OBJETOS

- Es importante saber diferenciar que es una clase y que es un objeto y en que consiste cada una de ellas. Por ello, recapitulamos toda la información mostrada hasta el momento.

- Una clase es un patrón para la definición del estado y el comportamiento de un tipo particular de objetos.
- Todos los objetos de una clase dada son idénticos en estructura y comportamiento, pero tienen identidad única.
- Un objeto pertenece a una clase en particular.
- Los objetos son creados y destruidos en tiempo de ejecución. Residen en el espacio de memoria.
- Así para la clase Reloj descrita anteriormente, tendremos los objetos unReloj (dia="01-03-2010", hora="13:01:03", modelo="Rolex", numSerie="123456") y otroReloj (dia="01-03-2010", hora="13:01:03", modelo="Swatch", numSerie="Sab748").

6.7. EJEMPLO DE UN PROGRAMA ORIENTADO A OBJETOS

//en este ejemplo explicar polimorfismo, herencia, clientela, método constructor, …

**Declaración de las clases:

**Declaración de la clase Publicacion
Clase Publicacion(
Datos:
Nombre_publicacion es una cadena de caracteres
Nombre_autores una cadena de caracteres

Métodos (procedimientos y funciones):

**Método constructor: para crear un objeto de la clase Publicacion e inicializar sus datos o atributos
Procedimiento constructor Publicacion ()
Inicio

```
                Nombre_publicacion← " "
                Nombre_autor← " "
        Fin
        Procedimiento insertar_datos_publicacion (npublicacion, nautor son cadenas de caracteres)
        Inicio
                Nombre_publicacion←npublicacion
                Nombre_autor←nautor
        Fin
        Procedimiento mostrar_datos_publicacion ()
        Inicio
                Escribir "Los datos de la publicación son: "
                Escribir Nombre_publicacion, Nombre_autor
        Fin
        **Método destructor: Limpia el valor de los datos de un objeto de la clase Publicacion y lo elimina

        Procedimiento destructor D-Publicacion ()
        Inicio
                Escribir "Fin del objeto de esta clase Publicacion"
        Fin
) **Fin de la clase publicacion

**Declaración de la clase Revista, que es subclase de Publicacion
Clase Revista : Publicacion(
        Datos:
                Tipo_revista  es una cadena de caracteres
                Numero_revista es una cadena de caracteres
                **Se heredan los datos de la superclase Publicacion: Nombre_publicacion  y Nombre_autor

        Métodos (procedimientos y funciones):
                ** Se heredan todos los métodos de la superclase Publicacion
                Procedimiento constructor Revista ()
                Inicio
                        Publicacion()          **esto es la llamada al procedimiento constructor Publicacion
                        Tipo_revista← " "
                        Numero_revista← " "
                Fin
                Procedimiento insertar_datos_revista (npublicacion, nautor, t_revista, n_revista son cadenas de caracteres)
                Inicio
                        insertar_datos_publicacion (npublicacion, nautor)
                        Tipo_revista←t_revista
                        Numero_revista←n_revista
                Fin
                Procedimiento mostrar_datos_revista ()
                Inicio
                        mostrar_datos_publicacion()
                        Escribir Tipo_revista, Numero_revista
```

```
            Fin
            Procedimiento destructor D-Revista ()
            Inicio
                  D-Publicacion()
                  Escribir “Fin del objeto de esta subclase Revista”
            Fin
) **Fin de la subclase Revista


**Declaración de la clase Libro, que es subclase de Publicacion
Clase Libro : Publicacion(
      Datos:
            ISBN  es una cadena de caracteres

            **Se heredan los datos de la superclase Publicacion: Nombre_publicacion  y
            Nombre_autor

      Métodos (procedimientos y funciones):
            ** Se heredan todos los métodos de la superclase Publicacion
            Procedimiento constructor Libro()
            Inicio
                  Publicacion()        **esto es la llamada al procedimiento constructor
Publicacion
                  ISBN← “ ”
            Fin
            Procedimiento insertar_datos_libro (npublicacion, nautor,  codigo_libro son cadenas de
caracteres)
            Inicio
                  insertar_datos_publicacion (npublicacion, nautor)
                  ISBN←codigo_libro
            Fin
            Procedimiento mostrar_datos_libro ()
            Inicio
                  mostrar_datos_publicacion()
                  Escribir ISBN
            Fin
            Procedimiento destructor D-libro ()
            Inicio
                  D-Publicacion()
                  Escribir “Fin del objeto de esta subclase libro ”
            Fin
) **Fin de la subclase Libro

**Declaración de la clase Acta, que es subclase de Publicacion
Clase Acta : Publicacion(
      Datos:
            Fecha_acta es una cadena de caracteres
            Nombre_congreso son cadenas de caracteres
            **Se heredan los datos de la superclase Publicacion: Nombre_publicacion  y
            Nombre_autor
```

```
        Métodos (procedimientos y funciones):
                ** Se heredan todos los métodos de la superclase Publicacion
                Procedimiento constructor Acta()
                Inicio
                        Publicacion()          **esto es la llamada al procedimiento constructor Publicacion
                        Fecha_acta← " "
                        Nombre_congreso←" "
                Fin
                Procedimiento insertar_datos_acta (npublicacion, nautor,  f_acta, n_congreso son cadenas de caracteres)
                Inicio
                        insertar_datos_publicacion (npublicacion, nautor)
                        Fecha_acta← f_acta
                        Nombre_congreso←n_congreso
                Fin

                Procedimiento mostrar_datos_acta ()
                Inicio
                        mostrar_datos_publicacion()
                        Escribir Fecha_acta, Nombre_congreso, Fecha_congreso
                Fin

                **Procedimiento polimórfico (o con poliformismo)
                Procedimiento mostrar_datos_publicacion ()
                Inicio
                        Escribir "Sólo se muestra los datos propios del objeto de la subclase Acta"
                        Escribir Fecha_acta, Nombre_congreso, Fecha_congreso

                Fin
                Procedimiento destructor D-acta ()
                Inicio
                        D-Publicacion()
                        Escribir "Fin del objeto de esta subclase Acta "
                Fin
) **Fin de la subclase Acta

**Uso de la clase Publicacion y sus subclases en el programa
PROGRAMA PRINCIPAL

Inicio
        **Creación de objetos
        Revista R ("PC Computer", "U.K.", "Informatica", "Enero 2010")
        Libro L ("Los intereses creados", "Jacinto Benavente", "84-376-0027-8")
        Acta A ()

        **Mensajes o llamadas a métodos
        R.mostrar_datos_revista()
        R.mostrar_datos_publicacion()
        L.mostrar_datos_libro()
        A.mostrar_datos_acta()
```

Fin

EJERCICIO1 PROPUESTO:

Clase Operaciones_Binarias
Atributos: Operando1 Operando2
Metodos: Sumar Restar Multiplicar Dividir

a. Declara la siguiente clase
b. Utilízala en un programa, donde se cree al menos un objeto de este tipo de clase y llamadas a sus métodos.
c. ¿Si quisieras realizar operaciones con tres operandos qué harías?

EJERCICIO2 PROPUESTO: Decidir cuál es el concepto o conceptos (Clase, Objeto, Método, Atributo) que cuadran con las siguientes definiciones.

1. El valor de mis atributos puede ser distinto al de los de mi semejante:
2. Yo me comporto como una plantilla:
3. A mi me gusta hacer cosas:
4. Yo puedo tener muchos métodos:
5. Yo represento el estado:
6. Yo represento el comportamiento:
7. Yo estoy en los objetos:
8. Yo vivo en memoria:
9. Yo soy usado para crear instancias:
10. Mi estado puede cambiar:
11. Yo declaro métodos:
12. Yo puedo cambiar en ejecución:

PARA RECORDAR: En esta Unidad, en el apartado sobre POO hemos visto los conceptos y características básicas en los que se apoya la Orientación a objetos:

- Objeto.
- Atributo.
- Método.
- Abstracción
- Encapsulación y ocultamiento
- Interfaz.
- Clase
- Polimorfismo