



SINTAXIS PARA EL DESARROLLO DE ALGORITMOS EN JAVA

Guía de Usuario

JORGE HERNÁN SUAZA JIMÉNEZ

TABLA DE CONTENIDO

INTRODUCCIÓN	3
1. PROGRAMACIÓN ORIENTADA A OBJETOS	11
2. ESTRUCTURA BÁSICA	16
3. CREACIÓN DE VARIABLES	17
4. MOSTRAR UN TEXTO Y/O UNA VARIABLE POR PANTALLA	21
5. INGRESAR UN VALOR A UNA VARIABLE	22
6. ESTRUCTURAS SECUENCIALES	25
7. ESTRUCTURAS CONDICIONALES	30
8. CASOS (switch)	34
9. CICLO FOR	37
10. CICLO WHILE	39
11. CICLO DO WHILE	45
12. ARREGLOS	47
13. EXCEPCIONES (EXCEPTION Y TRY-CATCH)	58

INTRODUCCIÓN

Estos apuntes servirán de apoyo con miras de mejorar los niveles de apropiación del conocimiento y desarrollo de las competencias de pensamiento analítico y sistémico en los estudiantes inscritos a la facultad de Ingeniería del ITM, facilitándole desarrollar su capacidad analítica y creadora, de ésta forma mejorar su destreza en la elaboración de algoritmos en el lenguaje java.

Permitirá el primer acercamiento de los estudiantes con la programación, pretende ser una guía a los estudiantes a la implementación correcta de los diferentes conceptos de java y la programación orientada a objetos.

1. PROGRAMACIÓN ORIENTADA A OBJETOS

¿Qué es la programación Orientada a Objetos?

La P.O.O. (también conocida como O.O.P., por sus siglas en inglés) es lo que se conoce como un paradigma o modelo de programación. Esto significa que no es un lenguaje específico, o una tecnología, sino una forma de programar, una manera de plantearse la programación. No es la única (o necesariamente mejor o peor que otras), pero se ha constituido en una de las formas de programar más populares e incluso muchos de los lenguajes que usamos hoy día lo soportan o están diseñados bajo ese modelo (PHP, AS2, AS3, Java).

La creciente tendencia de crear programas cada vez más grandes y complejos llevó a los desarrolladores a crear una nueva forma de programar que les permita crear sistemas de niveles empresariales y con reglas de negocios muy complejas. Para estas necesidades ya no bastaba la programación estructurada ni mucho menos la programación lineal. Es así como aparece la programación orientada a objetos (POO). La POO viene de la evolución de la programación estructurada; básicamente la POO simplifica la programación con la nueva filosofía y nuevos conceptos que tiene. La POO se basa en dividir el programa en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les llama objetos. Los objetos son unidades independientes que se comunican entre ellos mediante mensajes.

¿Cuáles son las ventajas de un lenguaje orientado a objetos?

Fomenta la reutilización y extensión del código.

Permite crear sistemas más complejos.

Relacionar el sistema al mundo real.

Facilita la creación de programas visuales.

Construcción de prototipos

Agiliza el desarrollo de software

Facilita el trabajo en equipo

Facilita el mantenimiento del software

El Modelo Orientado a Objetos

Para entender este modelo vamos a revisar los siguientes conceptos básicos iniciales:

Una clase es la estructura de un objeto, es decir, es la unidad básica que reúne toda la información de un objeto como son sus propiedades (atributos) y comportamiento (métodos) en común. Las clases por sí mismas, no son más que modelos que nos servirán para crear objetos en concreto.

Una clase se compone de tres partes:

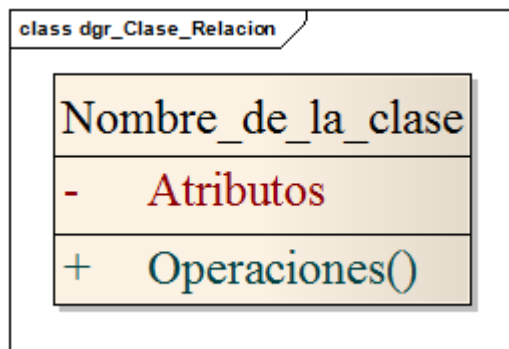
- **Nombre:** se define el nombre de la clase a la cual nos referimos ejemplo Persona (se sugiere que el nombre de la clase comience en mayúscula)
- **Atributos** (denominados, por lo general, datos miembros): esto es, los datos que se refieren al estado del objeto y se representan a modo de variable ejemplo: nombre, estatura, raza, edad
- **Métodos** (denominados, por lo general, funciones miembros): son funciones que pueden realizar los objetos ejemplo: correr, comer, saltar, dormir

Representación Gráfica de las clases (Diagrama de Clases)

La representación gráfica de una o varias clases se hará mediante los denominados *diagramas de clase*. Para los diagramas de clase se utilizará la notación que provee el Lenguaje de Modelación Unificado (UML, ver www.omg.org), a saber:

- Las clases se denotan como rectángulos divididos en tres partes. La primera contiene el nombre de la clase, la segunda contiene los atributos y la tercera las operaciones.
- Los modificadores de acceso a datos y operaciones, a saber: público, protegido y privado; se representan con los símbolos +, #, – ; se colocan al lado izquierdo del atributo o método. (+ público, # protegido, - privado).
 - **+ Público:** podemos acceder a las propiedades y métodos desde cualquier lugar, desde la clase actual, clases que heredan de la clase actual y desde otras clases.
 - **# Protegido:** se puede acceder al atributo o método desde la clase que lo define y desde cualquier otra que herede de esta clase.
 - **- Privado:** los atributos o métodos solo son accesibles desde la clase que los

define.



Estructura General de una Clase

La estructura general de una clase es la siguiente:

```
[especificadores] class Nombre_Clase
{
    Instrucciones
}
```

Especificadores(opcional): determinan el tipo de acceso a la clase (public, private o protected)

Class: palabra reservada que indica el comienzo de la clase

Nombre_Clase: Es el nombre que se le da a la clase. Para crearlo hay que seguir las mismas normas que para crear nombres de variables

Una Clase tiene un único punto de inicio, ({) la ejecución de una clase termina cuando se llega a }

public class Cls_Estudiente

```
{
    Instrucción(es)
}
```

Estructura General de un Método

La estructura general de un método es la siguiente:

```
[Especificadores] [estatico] tipoDevuelto nombreMetodo([lista parámetros])
{
    Instrucciones
    [retorne valor]
}
```

Los elementos que aparecen entre corchetes son opcionales.

Especificadores(opcional): determinan el tipo de acceso al método (public, private o protected)

Estatic : Una clase, método o campo declarado como estático puede ser accedido o invocado sin la necesidad de tener que instanciar un objeto de la clase, para efectos de este manual los métodos serán estáticos

TipoDevuelto: indica el tipo del valor que devuelve el método. Es imprescindible que, en la declaración de un método, se indique el tipo de dato que ha de devolver. El dato se devuelve mediante la instrucción **return**. Si el método no devuelve ningún valor este tipo será vacío

NombreMetodo: es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

Lista de parámetros (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas. Estos parámetros son los datos de entrada que recibe el método para operar con ellos.

Un método puede recibir cero o más argumentos. Se debe especificar para cada argumento su tipo. **Los paréntesis son obligatorios**, aunque estén vacíos.

Return: se utiliza para devolver un valor. La palabra clave **return** va seguida de una expresión que será evaluada para saber el valor de retorno. Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante.

El tipo del valor de retorno debe coincidir con el tipoDevuelto que se ha indicado en la declaración del método.

Si el método no devuelve nada (tipoDevuelto = void) la instrucción **return** es opcional.

Un método puede devolver un tipo primitivo, un arreglo, un texto o un objeto.

Un método tiene un único punto de inicio ({) , La ejecución de un método termina cuando se llega a } o cuando se ejecuta la instrucción **return**

La instrucción ***return*** puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final.

Cuando se llama a un método, la ejecución del programa pasa al método y cuando éste acaba, la ejecución continúa a partir del punto donde se produjo la llamada.

Todo proyecto puede tener muchos o pocos métodos pero siempre debe existir el método `main()` Como punto de inicio

2 ESTRUCTURA BÁSICA

Al ser java un lenguaje de programación orientado a objetos, los códigos o instrucciones que den solución al problema informático debe estar enmarcado dentro de un método y éste a su vez dentro de una clase. Al crear un proyecto, java nos brinda una clase y dentro de esta un método inicial al que llamaremos método principal o método main() , es dentro de este método donde colocaremos todas nuestras instrucciones.

Tanto la clase como el método principal deben indicar dónde empieza y donde termina, para esto utilizaremos las llaves { } como se muestra a continuación

```
public class suma_numeros
{
    public static void main (String [] args)
    {
        Instrucción 1;
        Instruccion2;
    }
}
```

Todo programa en Java necesita una función, o método, denominada main() para establecer el inicio del programa. El contenido de dicha función, String[] args, define un vector de cadenas de caracteres donde recogerá los argumentos pasados desde la línea de comandos del sistema operativo
public significa que el método es visible y puede ser llamado desde otros objetos. Otras alternativas son privadas, protegidas
static significa que el método está asociado con la clase, no a una instancia (objeto) de la clase. Esto significa que usted puede llamar a un método estático y sin crear un objeto de la clase.
void significa que el método no tiene valor de retorno. Si el método devuelve un int escribe int en lugar de void.

NOTA: El nombre de la clase y de los métodos debe seguir las siguientes reglas

- No espacios en blanco
- No debe contener caracteres especiales tales como (@#\$%^&*? Entre otros) excepto el guion bajo (_), el cual se usa cuando el nombre es compuesto como, por ejemplo:

Public class suma_numeros

- Debe comenzar con una letra

Tanto el método como la clase NO terminan en punto y coma (;) pero si las instrucciones que están contenidas dentro del método

3. CREACIÓN DE VARIABLES

Uno de los factores más importantes dentro de la programación son las variables ya que allí es donde se almacenarán los diferentes datos que se necesitarán dentro del programa ya sean ingresados por el usuario o datos calculados

Para la creación de las variables se debe tener en cuenta que el nombre debe tener las mismas reglas de los nombres de las clases, pero también se debe tener en cuenta el tipo de valores que va a almacenar porque éste determinará el tamaño en memoria que se reservará.

Los tipos de datos en java para este manual serán:

Tipo	Tamaño	Rango
byte	Entero de 1 byte (8 bits)	de -128 a 127
short	Entero de 2 byte (16 bits)	de -32768 a 32767
Int	Entero de 4 byte (32 bits)	de -2.147.483.648 a 2.147.483.647
long	Entero de 8 byte (64 bits)	de -2^{63} a $2^{63} - 1$
float	Real de 4 byte (32 bits)	6 dígitos significativos (10^{-46} , 10^{38})
double	Real de 8 byte (64 bits)	15 dígitos significativos (10^{-324} , 10^{308})
String	Cadena de caracteres	Comillas dobles
char	Un solo carácter	Comillas sencillas
boolean	Dato Lógico de 1 byte (8 bits)	false y true

En java se antepone el tipo de dato seguido de las variables a crear, cabe anotar que en la línea de la construcción de las variables también se pueden inicializar con un valor.

EJEMPLO

```
public class suma_numeros
{
    public static void main(String[] args)
    {
        int numero1, numero2=5;
        double suma;
        String nombre;
        String nombre, opcion= "si"
        char Respuesta = 'A';

    }
}
```

Nota: los valores de variables tipo texto o cadena deben ir entre comillas dobles, las de carácter con comillas simples

Nota: el tipo de dato String comienza con S en mayúscula, el resto de tipos de datos son en minúscula

Nota: los tipos de datos numéricos (int, double, long, etc) comienzan con minúscula pero el tipo de dato Texto (String) comienza con mayúscula y todas las instrucciones termina en punto y coma.

Las variables se pueden clasificar según su ámbito en:

- Variables miembro de una clase o atributos de una clase
- Variables locales
- Variables de bloque

VARIABLE MIEMBRO O ATRIBUTOS DE UNA CLASE

Son las declaradas dentro de una clase y fuera de cualquier método.

Aunque suelen declararse al principio de la clase, se pueden declarar en cualquier lugar siempre que sea fuera de un método. Para esto se debe anteponer la palabra **static**

Son accesibles en cualquier método de la clase.

Pueden ser inicializadas.

VARIABLES LOCALES

Son las declaradas dentro de un método.

Su ámbito comienza en el punto donde se declara la variable.

Están disponibles desde su declaración hasta el final del método donde se declaran.

No son visibles desde otros métodos.

Distintos métodos de la clase pueden contener variables con el mismo nombre. Se trata de variables distintas.

El nombre de una variable local debe ser único dentro de su ámbito.

Si se declara una variable local con el mismo nombre que una variable miembro de la clase, la variable local tiene preferencia. La variable miembro queda inaccesible en el ámbito de la variable local con el mismo nombre.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del método.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos.

VARIABLES DE BLOQUE

Son las declaradas dentro de un bloque de instrucciones

Su ámbito comienza en el punto donde se declara la variable.

Están disponibles desde su declaración hasta el final del bloque donde se declaran.

No son visibles desde otros bloques.

Distintos bloques pueden contener variables con el mismo nombre. Se trata de variables distintas.

Si un bloque de instrucciones contiene dentro otro bloque de instrucciones, en el bloque interior no se puede declarar una variable con el mismo nombre que otra del bloque exterior.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del bloque.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos

Ejemplo

```
public class Calcular
```

```
{
```

```
    static int suma;
```

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

```
    {
```

```
        int numero1, numero2=5 ;
```

```
    }
```

```
}
```

Variables miembros
de una clase o
atributos de una clase

Variables locales

Nota: las variables de bloque se explicarán en el capítulo 9

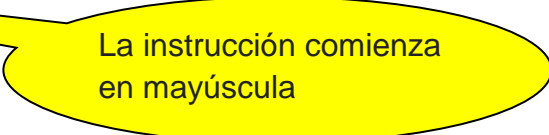
4. MOSTRAR UN TEXTO Y/O UNA VARIABLE POR PANTALLA

Una buena práctica es antes de que el usuario ingrese un valor es mostrar un mensaje indicando que debe de ingresar, esto ocurre de igual forma cuando queremos mostrar resultados, es conveniente mostrar un mensaje indicando que el valor a mostrar a que corresponde.

En java se puede realizar mediante la instrucción **System.out.println ("texto");**

EJEMPLO

```
public class suma_numeros
{
    public static void main (String [] args)
    {
        int numero1, numero2=5;
        double suma;
        String nombre;
        System.out.println ("algoritmo que muestra una variable
inicializada");
        System.out.print ("el segundo número es " + numero2);
    }
}
```



Nota: si se desea mostrar un texto y luego una variable deben estar separados por un signo de más (+), este símbolo indica concatenación o unión.

El texto debe ir entre comillas, lo que NO ocurre con las variables.

System.out.println: El println es para que una vez muestre el mensaje realice un salto de línea

System.out.print: El print es para que una vez muestre el mensaje NO realice salto de línea lo que indica que el cursor se mostrara al frente del mensaje.

Realice el ejemplo anterior invirtiendo las dos líneas de impresión, Observar los resultados

Estas instrucciones también terminan en punto y coma (;)

5. INGRESAR UN VALOR A UNA VARIABLE

Se puede ingresar un valor a una variable por medio de dos formas, una por medio de asignación directa, esto es cuando una variable se inicializa o cuando se almacenan los diferentes cálculos u operaciones que se realiza dentro del algoritmo, la otra forma es cuando las operaciones de entrada permiten al usuario ingresar un valor a una variable para esto se necesita la utilización de la clase Scanner.

ASIGNACION DIRECTA

```
NUM1 = 5;
AREA = LADO1*LADO2;
```

Dentro de los cálculos se deben usar los operadores Aritméticos

Operadores Aritméticos

+	Suma
-	Resta
*	Multiplicación
/	División
Math.pow	Exponente,
se realizará de la siguiente forma:	
Math.pow (base, exponente) ejemplo	
Math.pow (5,2)	
%	Residuo de una división

Funciones Especiales

String.format: permite darle formato a un numero
System.out.println(String.format("%.2f",dolares));

donde 2f significa dos decimales

NOTA: En la asignación directa se usa el símbolo de igual indicando que se va a almacenar un valor en una variable para esto la variable debe estar al lado izquierdo del igual y el valor o el resultado del cálculo a almacenar debe estar al lado derecho del igual y se termina con punto y coma

INGRESO POR MEDIO DE UN DISPOSITIVO DE ENTRADA (TECLADO)

Para utilizar Scanner en el programa tendremos que hacer lo siguiente:

1. Escribir el import

La clase Scanner se encuentra en el paquete java.util por lo tanto se debe incluir al inicio del programa la instrucción:

```
import java.util.Scanner;    O   import java.util.*;
```

2. Crear un objeto Scanner

Tenemos que crear un objeto de la clase Scanner asociado al dispositivo de entrada.

Si el dispositivo de entrada es el teclado escribiremos:

```
Scanner sc = new Scanner (System.in);
```

Se ha creado el objeto sc asociado al teclado representado por System.in

Una vez hecho esto podemos leer datos por teclado.

Ejemplos de lectura:

Para leer podemos usar el método nextXxx () donde Xxx indica en tipo, por ejemplo, nextInt () para leer un entero, nextDouble () para leer un double, next() o nextLine() para leer un texto.

Ejemplo de lectura por teclado de un número entero:

```
int n;
System.out.print ("Introduzca un número Entero: ");
n = sc.nextInt ();
```

Ejemplo de lectura de un número de tipo double:

```
double x;
System.out.print ("Introduzca número de tipo double: ");
x = sc.nextDouble ();
```

Ejemplo de lectura de una cadena de caracteres:

```
String s;
System.out.print ("Introduzca texto: ");
s = sc.nextLine ();
```

EJEMPLO sumar dos números

ALGORITMO	JAVA
publico class Calcular_Suma publico estatico vacio principal() real num1,num2,Suma imprima("digite numero1") lea num1 imprima("digite numero2") lea num2 Suma = num1+num2 imprima("la suma es"+ suma) finmetodo finclase	<pre>import java.util.Scanner; public class Calcular_Suma { public static void main(String[] args) { double num1,num2,Suma; Scanner sc = new Scanner(System.in); System.out.print("digite un número"); num1=sc.nextDouble(); System.out.print("digite otro número"); num2=sc.nextDouble(); Suma = num1 + num2; System.out.print("la suma es "+ Suma); } }</pre>

OBSERVACION: Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o String debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro.

Esto ocasiona problema si el orden de ingreso es datos numéricos y luego datos tipo texto porque estos últimos el método `nextLine()` extrae del buffer de entrada todos los caracteres hasta llegar a un intro. Esto provoca que el programa no funcione correctamente, ya que no se detiene para que se introduzca el valor de tipo texto.

Solución: una solución es leer los datos de tipo texto primero y luego los datos de tipo numérico al final, la otra solución es leer los datos de tipo numérico luego limpiar el buffer (`sc.nextLine()`) y a continuación leer los datos de tipo texto, como se describe a continuación;

INCORRECTA	CORRECTO
<pre>import java.util.Scanner; public class JavaApplication335 { public static void main(String[] args) { Scanner sc = new Scanner(System.in); String nombre; double radio; int n; System.out.print("Introduzca el radio: "); radio = sc.nextDouble(); System.out.println("Longitud de la circunferencia: " + 2*Math.PI*radio); System.out.print("Introduzca un número entero: "); n = sc.nextInt(); System.out.println("El cuadrado es: " + Math.pow(n,2)); System.out.print("Introduzca su nombre: "); nombre = sc.nextLine(); //leemos el String después del double System.out.println("Hola " + nombre + "!!!"); } }</pre>	<pre>import java.util.Scanner; public class JavaApplication335 { public static void main(String[] args) { Scanner sc = new Scanner(System.in); String nombre; double radio; int n; System.out.print("Introduzca el radio: "); radio = sc.nextDouble(); System.out.println("Longitud de la circunferencia: " + 2*Math.PI*radio); System.out.print("Introduzca un número entero: "); n = sc.nextInt(); System.out.println("El cuadrado es: " + Math.pow(n,2)); sc.nextLine(); System.out.print("Introduzca su nombre: "); nombre = sc.nextLine(); System.out.println("Hola " + nombre + "!!!"); } }</pre>

6. ESTRUCTURAS SECUENCIALES

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia, es decir, que una instrucción no se ejecuta hasta que finaliza la anterior. Al tomar el ejemplo propuesto en java quedaría de la siguiente forma.

EJEMPLO

ALGORITMO	JAVA
publico clase Rectangulo publico estatco vacio principal() real lado1,lado2,area imprima("digite un lado") lea lado1 imprima("digite otro lado") lea lado2 area = lado1*lado2 imprima("el área es"+ área) finmetodo finclase	<pre>import java.util.Scanner; public class Rectangulo { public static void main(String[] args) { Scanner sc = new Scanner(System.in); double lado1,lado2,area; System.out.print("digite un lado"); lado1=sc.nextDouble(); System.out.print("digite otro lado"); lado2=sc.nextDouble(); area = lado1 * lado2; System.out.print("el area es "+ area); } }</pre>

TRABAJANDO CON DOS O MÁS METODOS

Si bien es cierto que al colocar todo el código en el método principal (main) mostrará los resultados esperados, también hay que tener en cuenta que la creciente tendencia de crear programas cada vez más grandes y complejos conlleva a la nueva filosofía y nuevos conceptos que tiene la POO (programación orientada a objetos) de dividir el programa en pequeñas unidades lógicas de código (Métodos).

En general, un problema complejo puede ser resuelto de manera más fácil y eficiente si se divide en problemas más pequeños y concentrándonos en cada etapa en la solución de ese "subproblema".

Esto implica que el gran problema original será resuelto por medio de varios módulos, cada uno de los cuales se encarga de resolver un subproblema determinado. En la POO esos módulos, se conocen con el nombre de métodos.

Los métodos se escriben sólo una vez, luego es posible hacer referencia a ellos ("llamarlos") desde diferentes puntos del programa.

La ventaja es que nos permite reutilizarlos y evita la duplicación de códigos.

Los métodos son independientes entre sí, en el sentido de que se puede escribir y verificar cada módulo en forma separada sin preocuparse por los demás módulos. Por ello, es más fácil identificar un error y también se puede modificar el código sin tener que tocar o rehacer varias partes del mismo.

Como se mencionó en el capítulo 1 los métodos pueden utilizar parámetros

Parámetros Actuales: son los argumentos que aparecen en la llamada a un método. Contienen los valores que se le pasan al método. Un parámetro actual puede ser una variable, un objeto, un valor literal válido, etc.

Parámetros Formales: son los argumentos que aparecen en la cabecera del método. Reciben los valores que se envían en la llamada al método. Se utilizan como variables normales dentro del método.

Los parámetros actuales y los formales **deben coincidir en número, orden y tipo**. Si el tipo de un parámetro actual no coincide con su correspondiente parámetro formal, el sistema lo convertirá al tipo de este último, siempre que se trate de tipos compatibles. Si no es posible la conversión, el compilador dará los mensajes de error correspondientes.

Si el método devuelve un valor, la llamada al método puede estar incluida en una expresión que recoja el valor devuelto.

Nota: solo se retorna un valor o vacio

Existen diversas formas de usar o invocar un método, bien sea desde el programa principal(main) o desde otros métodos, en este ejemplo vamos a trabajar con 2 métodos, pero tiene la misma funcionalidad para 3, 4 o mas métodos

1. **No Envía Parámetros, No Recibe Parámetros:** en este caso solo se invoca un método, pero no se envía valores, este método realiza los cálculos necesarios e imprime los resultados (no los retorna)

```
import java.util.Scanner;

public class Operaciones
{
    public static void main(String[] args)
    {
        calcular();
    }

    public static void calcular()
    {
        Scanner sc = new Scanner(System.in);
        double nota1,nota2,nota3,nd;
        nota1= sc.nextDouble();
        nota2= sc.nextDouble();
        nota3= sc.nextDouble();
        nd=(nota1 + nota2 + nota3)/3;
        System.out.print(nd);
    }
}
```

De esta forma se invoca un método sin enviar parámetros

En este ejemplo el método calcular no recibe parámetros y tampoco retorna parámetro, al no retornar parámetro se coloca la palabra void

2. **Envía parámetros, pero no recibe parámetros:** en este caso se invoca un método y se le envían valores para que pueda realizar ciertos cálculos e imprime los resultados (no los retorna)

```
import java.util.Scanner;

public class Operaciones
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        double nota1,nota2,nota3,nd;
        nota1= sc.nextDouble();
        nota2= sc.nextDouble();
        nota3= sc.nextDouble();
        calcular(nota1,nota2,nota3);
    }
}
```

Se invoca un método y se envían parámetros
Estos valores van separados por comas (,) si el valor enviado es un texto va entre comillas
Ejemplo
Calcular(nota1," Luis")

```

    }

    public static void calcular(double n1, double n2, double n3)
    {
        double nd;
        nd=(n1 + n2 + n3)/3;
        System.out.print(nd);
    }
}

```

Se reciben los parámetros indicando por cada variable el tipo de dato, se reciben en su estricto orden

3. **No Envía Parámetros y Recibe parámetro:** en este caso se invoca un método y no se envían parámetros, sin embargo, el método destino realiza los cálculos necesarios y retorna el valor al método que lo invoco

```

import java.util.Scanner;

public class Operaciones
{
    public static void main(String[] args)
    {
        double notadef;
        notadef=calcular();
        System.out.print(notadef);
    }

    public static double calcular()
    {
        Scanner sc = new Scanner(System.in);
        double nota1, nota2, nota3, nd;
        nota1= sc.nextDouble();
        nota2= sc.nextDouble();
        nota3= sc.nextDouble();
        nd=(nota1 + nota2 + nota3)/3;
        return nd;
    }
}

```

Se invoca un método, no se envían parámetros y se recibe el resultado, en este ejemplo el valor devuelto lo recibe la variable notadef

Se debe cambiar la palabra void por el tipo de dato que está retornando, en este caso es double porque la variable nd que es la que retorna es double

La palabra return indica devolver un valor al método que lo invoco

4. **Envía Parámetros y recibe parámetros:** en este caso se invoca un método, se envían parámetros y el método destino retorna un valor al método que lo invoco

```
import java.util.Scanner;

public class Operaciones
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        double nota1,nota2,nota3,nd;
        nota1= sc.nextDouble();
        nota2= sc.nextDouble();
        nota3= sc.nextDouble();
        notadef=calcular(nota1,nota2,nota3);
        System.out.print(notadef);
    }

    public static double calcular(double n1, double n2, double n3)
    {
        double nd;
        nd=(n1 + n2 + n3)/3;
        return nd;
    }
}
```

Se invoca el método, se envían parámetros y se recibe el resultado

Se debe cambiar la palabra void por el tipo de dato a retornar

La palabra *return* indica devolver un valor al método que lo invoco

7. ESTRUCTURAS CONDICIONALES

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que, con base al resultado de esta comparación, se siga un curso de acción dentro del programa. El resultado de esta comparación siempre deberá ser un valor lógico (verdadero o falso). Existen dos tipos básicos, las estructuras simples y las compuestas

NOTA: La estructura condicional anidada es cuando una condición está dentro de otra condición

<u>SIMPLES</u>	<u>MÚLTIPLES</u>
<pre> if(condición(es)) { Instruccion1 Instruccion2 } </pre>	<pre> if(condición(es)) { Instrucción(es) } else { if(condición(es)) { Instrucción(es) } else { if(condición(es)) { Instrucción(es) } } else { Instrucción(es) } } } </pre>
<u>COMPUESTAS</u>	
<pre> if(condición(es)) { Instruccion1 Instruccion2 } else { Instruccion3 Instruccion4 } </pre>	

Dentro de las condiciones se debe usar tanto los operadores Relaciones como Lógicos

Operadores Relacionales

<	Menor que
<=	menor o igual
>	Mayor que
>=	Mayor o igual
==	Igual
!=	Diferente

Operadores Lógicos

&&	(Y, Conjunción)
	(O, Disyunción)
Not	(Negación)

ERRORES PARA NO COMETER

<u>ERROR</u>	<u>CORRECTA</u>
if(A > B < C)	if(A>B && B<C)
if(10 < A < 20)	if(10<A&&A<20)
if(A < B ^ < C)	if(A<B&&A<C)

TABLAS DE VERDAD DE LOS OPERADORES LÓGICOS			
X	Y	X && Y	X Y
Verdadero	Verdadero	Verdadero	Verdadero
Verdadero	Falso	Falso	Verdadero
Falso	Verdadero	Falso	Verdadero
Falso	Falso	Falso	Falso

Para determinar los operadores relacionales y lógicos que se deben de utilizar en cada caso, depende de una buena interpretación del problema y de identificar las variables, datos o expresiones aritméticas que se deben comparar en una expresión.

NOTA: Dentro de la condición, si existe más de una comparación estas deben estar separadas por un conector lógico (&&, ||)

RECOMENDACIÓN: TABULAR LAS INSTRUCCIONES DENTRO DE LAS ESTRUCTURAS

Ejemplo

DE FORMA TABULADA	NO TABULADA
<pre> if(condición(es)) { Instrucción(es) } else { if(condición(es)) { Instrucción(es) } else { if(condición(es)) { </pre>	<pre> if(condición(es)) { Instrucción(es) } else { if(condición(es)) { Instrucción(es) } else { if(condición(es)) { </pre>

Instrucción(es)	Instrucción(es)
<pre> } sino { Instrucción(es) } } </pre>	<pre> } sino { Instrucción(es) } } </pre>

Al tabular la instrucción se ve muy claro que el condicional azul está en la zona falsa del condicional morado y éste a su vez está en la parte falsa del condicional rojo

Al colocar las instrucciones sin tabulación, no quiere decir que este mal el condicional, solo que es más difícil de identificar cada condicional, encontrar errores o hacer mantenimiento al código

Ejercicio

Lea dos números e imprima ambos solo si son positivos

DIAGRAMA DE LA CLASE

Cls_Numeros
- real num1 - real num2
+ principal ():vacío +Mostrar_Numero_Positivo (): vacío

JAVA

```
import java.util.Scanner;

public class Cls_Numeros
{
    private static double num1, num2;

    public static void Mostrar_Numero_Positivo()
    {
        Scanner sc = new Scanner(System.in);
        num1= sc.nextDouble();
        num2= sc.nextDouble();
        if(num1> 0 && num2> 0)
        {
            System.out.print(num1+" " + num2);
        }
    }

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

8. CASOS (switch)

Permite seleccionar una, dentro de un conjunto de alternativas, con base en el valor almacenado en **un campo variable** denominado campo controlador de la estructura. De acuerdo con el valor que tenga el controlador, se realiza una determinada tarea una sola vez

Objetivo: Evitar el nido exagerado de preguntas (SI anidados).

FORMATO GENERAL

```
switch(variable)
{
    case <valor variable>:
        Instrucción(es);
        break;
    case <valor variable>:
        Instrucción(es);
        break;
    default:
        Instrucción(es);
        break;
}
```

Break se utiliza para finalizar el switch

Las siguientes reglas se aplican a una sentencia switch

La variable que se utiliza en una sentencia switch sólo puede ser un byte, short, int, char o String

Puedes tener cualquier número de sentencias case dentro de un switch. Siempre y cuando no haya dos con el mismo valor.

El valor de un caso debe ser el mismo tipo de datos que la variable en el switch.

Cuando la variable del switch es igual a un caso, las instrucciones que siguen a ese caso se ejecutarán hasta que se alcanza una sentencia break.

La sentencia break es opcional y se utiliza para finalizar el switch tras la ejecución de un case. Si la sentencia break no estuviera, al salir de un bloque case entraría en el siguiente, aunque el valor de ese case no coincidiera con el evaluado en la expresión. No todos los casos tienen que contener un break.

Una sentencia switch puede tener un caso por defecto (opcional), que debe aparecer al final del switch. El case default se ejecuta si el resultado de la expresión no coincide con ningún case. Su uso también es opcional.

NOTA: En <valor variable> se coloca el valor que la variable posiblemente puede tomar

EJEMPLO digite el estado civil de una persona y de acuerdo a lo digitado muestre un texto

<pre> publico clase Mostrar_estado_civil Privado texto estado_civil Publico estatico vacio metodo mostrar() Imprima("digite su estado civil") lea estado_civil segun(estado_civil) caso "soltero": Imprima("sigue asi") salto caso "casado": Imprima("lastima") salto caso "divorciado" Imprima("otra oportunidad") salto en otro caso: Imprima ("otro estado") salto finsegun finmetodo publico estatico vacio principal() mostrar() finmetodo finclase </pre>	<pre> import java.util.Scanner; public class Mostrar_estado_civil { private static String estado_civil; public static void mostrar() { Scanner sc = new Scanner(System.in); System.out.print("digite su estado civil"); estado_civil=sc.nextl(); switch(estado_civil) { case "soltero": System.out.print("Sigue asi"); break; case "casado": System.out.print("Lastima"); break; case "divorciado": System.out.print("otra oportunidad"); break; default: System.out.print("otro estado"); break; } } public static void main(String[] args) { mostrar(); } } </pre>
--	---

ESTRUCTURAS CICLICAS

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces.

Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).

Las estructuras cíclicas se clasifican en:

CICLO FOR

CICLO WHILE

CICLO DO WHILE

9. CICLO FOR

En muchas ocasiones, se conoce de antemano el número de veces que se desean ejecutar las acciones de un ciclo, en estos casos en el que el número de iteraciones es fijo, se debe usar el ciclo FOR.

Su representación es la siguiente:

for(variable_control=valor_inicial ; condicion ; incremento o decremento)

{

 Instrucción(es)

}

EJEMPLO

for(i=1; i <= 10; i= i+1)

{

 Instrucción(es);

}

FORMAS DE INCREMENTO Y DECREMENTO

Sintaxis	Equivalencia
i++	i = i+1
i--	i = i-1
A+=B	A= A+B
A-=B	A=A - B

NOTA: Se coloca la instrucción FOR y entre paréntesis las 3 partes que forman el ciclo, se debe inicializar una variable en un valor específico luego se coloca la condición (esta cumple con los mismos requisitos de un condicional), y luego va el incremento o decremento. Las 3 partes del ciclo van separadas por punto y coma (;).

EJEMPLO

Mostrar una tabla de multiplicar

Algoritmo	Java
<pre>publico clase Tabla_Multiplicar privado entero num privado entero prod privado entero i publico estatico vacio metodo mostrar_tabla() imprima("digite un número") lea num para (i=1;i<=10; i= i+1) prod = num * i imprima(num + "*" + i+ "=" + prod) finpara finmetodo publico estatico vacio principal() mostrar_tabla() finmetodo finclase</pre>	<pre>import java.util.Scanner; public class Tabla_Multiplicar { private static int num,prod,i; public static void mostrar_tabla() { Scanner sc = new Scanner(System.in); System.out.print("digite un número"); num=sc.nextInt(); for(i=1;i<=10;i++) { prod= num*i; System.out.println(num+"*" +i+"="+prod); } } public static void main(String[] args) { mostrar_tabla(); } }</pre>

10. CICLO WHILE

Esta estructura, repetirá un proceso cierta cantidad de veces por medio de una condición. La condición que controla esta estructura está situada al principio del bucle o ciclo y las instrucciones del interior se repetirán mientras sea verdadera la condición. Es decir, para que el bloque de instrucciones se repita, debe cumplirse la condición, cuando ésta no se cumpla, entonces deja de ejecutarse el proceso.

La estructura se representa de la siguiente forma:

```
<Iniciar una variable de control>  
while(<Condición(es)>)  
{
```

Bloque de instrucciones;

```
<Modificar la variable de  
control>  
}
```

Recordar: Entre las condiciones se debe evaluar la variable de control y dentro del ciclo debe existir una instrucción que altere dicha variable, de tal manera que en algún momento la condición es falsa

EJEMPLO

Calcular la nota definitiva de 5 estudiantes

<p>PSEUDOCODIGO</p> <p>publico clase Nota_Definitiva</p> <p> privado real n1 privado real n2 privado real nd privado real cantidad_estudiante</p> <p>publico estatico vacio metodo Calcular_Nota()</p> <p> cantidad_estudiante = 1</p> <p> mientras(cantidad_estudiante <= 5)</p> <p> lea n1 lea n2 nd= (n1+n2)/2 imprima("su nota definitiva es"+ nd) cantidad_estudiante = cantidad_estudiante +1</p> <p> finmientras</p> <p>finmetodo</p> <p>publico estatico vacio principal()</p> <p> Calcular_Nota()</p> <p>finmetodo</p> <p>finclase</p>	<pre>import java.util.Scanner; public class Tabla_Multiplicar { private static int n1,n2,nd,cantidad_estudiante public static void Calcular_Nota() { Scanner sc = new Scanner(System.in); cantidad_estudiante= 1 while(cantidad_estudiante <= 5) { n1=sc.nextInt(); n2=sc.nextInt(); nd=(n1+n2)/2; System.out.println("su nota definitiva es" + nd); Cantidad_estudiante=cantidad_estudiante+1; } } public static void main(String[] args) { Calcular_Nota(); } }</pre>
--	--

Ejemplo 2

Realice un algoritmo donde se entre por pantalla un numero entero. Calcular y mostrar la factorial del número

DIAGRAMA DE LA CLASE	JAVA			
<table><tr><th>Factorial</th></tr><tr><td>- entero numero - entero fac</td></tr><tr><td>+ principal():vacio +Capturar_Numero():vacio +Calcular_Factorial(entero num): entero</td></tr></table>	Factorial	- entero numero - entero fac	+ principal():vacio +Capturar_Numero():vacio +Calcular_Factorial(entero num): entero	<pre>import java.util.Scanner; public class Factorial { private static int numero, fac; public static void Capturar_Numero() { Scanner sc = new Scanner(System.in); Numero = sc.nextInt(); fact = Calcular_Factorial (numero); System.out.println("fact"); } public static int Calcular_Factorial (int num) { int cont =1; int F =1; while(cont <= num) { F= F * cont; cont = cont +1; } return F; } public static void main(String[] args) { Capturar_Numero(); } }</pre> <div>Retorna el valor del factorial</div>
Factorial				
- entero numero - entero fac				
+ principal():vacio +Capturar_Numero():vacio +Calcular_Factorial(entero num): entero				

CENTINELAS Y BANDERAS.

Cuando no se conoce a priori el número de iteraciones que se van a realizar, el ciclo puede ser controlado por centinelas.

CENTINELAS.

En un ciclo controlado por centinela el usuario puede suspender la introducción de datos cuando lo desee, introduciendo una señal adecuada llamada *centinela*. Un ciclo controlado por centinela es cuando el usuario digita una letra para salir como por ejemplo S o N para indicar si desea continuar o no. El ciclo debe repetirse hasta que la respuesta del usuario sea "n" o "N".

Cuando una decisión toma los valores de -1 o algún posible valor que no esté dentro del rango válido en un momento determinado, se le denomina centinela y su función primordial es detener el proceso de entrada de datos en una corrida de programa.

Por ejemplo, si se tienen las calificaciones de un test (comprendida entre 0 y 5); un valor centinela en esta lista puede ser -1, ya que nunca será una calificación válida y cuando aparezca este valor se terminará de ejecutar el ciclo.

Si la lista de datos son números positivos, un valor centinela puede ser un número negativo. Los centinelas solamente pueden usarse con las estructuras **while** y **do while**, no con estructuras **for**.

Ejemplo:

Suponga que debemos obtener la nota definitiva de cada uno de los estudiantes de una universidad, pero no sabemos exactamente cuántos son, la solución sería

DIAGRAMA DE LA CLASE	JAVA			
<table><tr><th>Nota_Definitiva</th></tr><tr><td>- real n1 - real n2 - real nd - texto respuesta</td></tr><tr><td>+ principal():vacio +Calcular_Nota():vacio</td></tr></table>	Nota_Definitiva	- real n1 - real n2 - real nd - texto respuesta	+ principal():vacio +Calcular_Nota():vacio	<pre>import java.util.Scanner; public class Nota_Definitiva { private static double n1; private static double n2; private static double nd; private String respuesta; public static void Calcular_Nota() { Scanner sc = new Scanner(System.in); respuesta = "si"; while(respuesta.equals("si"))</pre> <div>Variable centinela</div>
Nota_Definitiva				
- real n1 - real n2 - real nd - texto respuesta				
+ principal():vacio +Calcular_Nota():vacio				

	<pre> { n1=sc.nextDouble(); n2= sc.nextDouble(); nd= (n1+n2)/2; System.out.println("su nota definitiva es"+ nd); System.out.println("desea continuar? si/no"); respuesta=sc.next(); } } public static void main(String[] args) { Calcular_Nota(); } } </pre>
--	---

BANDERAS.

Conocidas también como interruptores, switch, flags o conmutadores, son variables que pueden tomar solamente dos valores durante la ejecución del programa, los cuales pueden ser 0 ó 1, o bien los valores booleanos True o False. Se les suele llamar interruptores porque cuando toman los valores 0 ó 1 están simulando un interruptor abierto/cerrado o encendido/apagado.

Ejemplo 1:

Leer un número entero N y calcular el resultado de la siguiente serie: $1 - 1/2 + 1/3 - 1/4 + \dots \pm 1/N$.

Algoritmo:

DIAGRAMA DE LA CLASE	JAVA			
<table><tr><td>Calcular_Serie</td></tr><tr><td><ul style="list-style-type: none">- real serie- real n- entero cont = 1- entero sw = 0</td></tr><tr><td><ul style="list-style-type: none">+ principal():vacio+Mostrar_Serie():vacio</td></tr></table>	Calcular_Serie	<ul style="list-style-type: none">- real serie- real n- entero cont = 1- entero sw = 0	<ul style="list-style-type: none">+ principal():vacio+Mostrar_Serie():vacio	<pre>import java.util.Scanner; public class Calcular_Serie { private static double serie =0; private static double n; private static int cont=1; private static int sw=0; public static void Mostrar_Serie() { Scanner sc = new Scanner(System.in);</pre>
Calcular_Serie				
<ul style="list-style-type: none">- real serie- real n- entero cont = 1- entero sw = 0				
<ul style="list-style-type: none">+ principal():vacio+Mostrar_Serie():vacio				

	<pre>n= sc.nextDouble(); while(cont <= n) { if(sw == 0) { serie = serie + 1/cont; sw = 1; } else { serie = serie - 1/cont; sw = 0; } Cont++; } System.out.println(serie); public static void main(String[] args) { Mostrar_Serie(); } }</pre>
--	--

11. CICLO DO WHILE

En esta estructura las instrucciones interiores del bucle o ciclo se repetirán mientras que la condición se cumpla. Permite realizar el proceso cuando menos una vez, ya que la condición se evalúa al final del ciclo, a diferencia del ciclo **While**, en el cual el proceso puede ser que nunca llegue a entrar si la condición a evaluar no se cumple desde el principio. 569042

Do

```
{
    Instrucción(es);
    <Modificar la variable de control>
    Instrucción(es);
}
while(<condición(es)>);
```

En este ciclo la condición esta por fuera de las llaves () y termina en ;

NOTA: A diferencia de las demás estructuras cíclicas, ésta tiene la particularidad de que en la línea donde va la condición termina en punto y coma (;)

EJEMPLO ingresar dos números luego mostrar un menú para realizar las operaciones básicas

<pre>publico clase CIs_menu privado entero opcion,num1,num2,res publico estatico vacio metodo mostrar_menu() Imprima("digite número") lea num1 Imprima("digite otro número") lea num2 Hacer Imprima("1.suma") Imprima("2.resta") Imprima("3.multiplicacion") Imprima("4.salir") Imprima ("digite su opción") lea opcion segun(opcion)</pre>	<pre>import java.util.Scanner; public class menu { private static int opcion,num1,num2,res; public static void mostrar_menu() { Scanner sc =new Scanner(System.in); System.out.print("digite un número "); num1=sc.nextInt(); System.out.print("digite otro número "); num2=sc.nextInt(); do { System.out.println("1. suma"); System.out.println("2. resta"); System.out.println("3. multiplicacion"); System.out.println("4. salir"); System.out.print("digite su opcion "); opcion=sc.nextInt();</pre>
--	--

<pre> caso 1: res=num1+num2 imprima ("la suma es"+ res) salto caso 2: res=num1-num2 imprima("la resta es"+ res) salto caso 3: res=num1*num2 imprima("la multiplicación es"+ res) salto finsegun mientras(opción <> 4) finmetodo publico estatico vacio principal() mostrar_menu() finmetodo finclase </pre>	<pre> switch(opcion) { case 1: res=num1+num2; System.out.println("la suma es "+ res); break; case 2: res=num1-num2; System.out.println("la resta es "+ res); break; case 3: res=num1*num2; System.out.println("la multiplicación es "+ res); break; } // fin del case } // fin del ciclo while(opcion != 4); } // fin del metodo public static void main(String[] args) { mostrar_menu(); } } </pre>
--	---

12. ARREGLOS

Un Arreglo es una estructura de datos que almacena bajo el mismo nombre (variable) a una colección de datos del mismo tipo.

Los arreglos se caracterizan por:

- Almacenan los elementos en posiciones contiguas de memoria
- Tienen un mismo nombre de variable que representa a todos los elementos. Para hacer referencia a esos elementos es necesario utilizar un índice que especifica el lugar que ocupa cada elemento dentro del archivo.

Clasificación de los arreglos

- Arreglos Unidimensionales (vectores)
- Arreglos Bidimensionales (matrices)

Arreglos Unidimensional (Vector): tiene una sola dimensión, cada elemento se diferencia del otro mediante un subíndice.

Para efectos de este manual, la primera posición será la posición 0 (cero).

Edades

0 1 2 3 4 5

18	15	20	10	12	30
----	----	----	----	----	----

Para este caso, el vector se llama Edades, tiene 6 elementos enteros, los números 0, 1, 2, 3, 4 y 5 representan las posiciones del vector. 18, 15, 20, 10, 12 y 30 son los elementos que están dentro del vector.

¿Cómo se declara un Vector?

Como los vectores en la POO son objetos se deben de instanciar (se llama instanciar a la acción de crear objetos)

- <Tipo de dato> <nombre del vector> <[]> = <new> <tipo de dato><[tamaño]>

Ejemplo

```
int Edades [] = new int[6] ;
```

- Valores constantes (el compilador deduce automáticamente la dimensión del arreglo)

Ejemplo

```
int Edades[] = {1,2,3,4};
```

¿Cómo se ingresa un dato al vector?

lea <nombre del vector> < [posición]>

EJEMPLO:

```
Edades [3] = sc.nextInt();
```

¿Cómo se muestra un dato del vector?

```
System.out.print( <nombre del vector> < [posición]>);
```

EJEMPLO:

```
System.out.print(Edades [2]);
```

Nota: La posición del vector debe ser un número entero, una variable entera o el resultado entero de un cálculo matemático.

Observación: podemos usar el atributo length quien nos determina la longitud del vector

```
int n= Edades.length();
```

EJEMPLO:

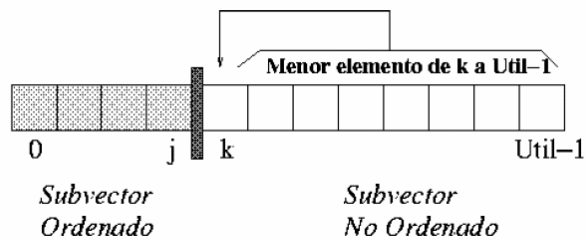
Realice un algoritmo que cree un vector con N edades y luego las muestre desde la última edad ingresada hasta la primera.

Algoritmo	Java
<pre>publico Clase Vector_Edades privado entero i,n privado entero edades[]; publico estatico vacio metodo crear_vector () imprima("digite longitud del vector") lea n edades=nuevo entero[n] finmetodo publico estatico vacio metodo llenar_vector()</pre>	<pre>import java.util.Scanner; public class cls_Vector_Edades { private static int i,n; private static int[] edades; private static Scanner sc; public static void crear_vector() { sc =new Scanner(System.in); System.out.print("digite longitud del vector"); n=sc.nextInt(); edades=new int[n]; } }</pre>

<pre> para (i=0; i<longitud(edades); i=i+1) imprima("Digite una edad") lea edades[i] finpara finmetodo publico estatico vacio metodo mostrar_vector () para(i=n-1; i>0; i= i-1) imprima("la edad es"+ edades[i]) finpara finmetodo publico estatico vacio principal() crear_vector() llenar_vector() mostrar_vector() finmetodo finclase </pre>	<pre> public static void llenar_vector() { for(l = 0 ; l < edades.length ; i++) { System.out.print("digite una edad"); edades[i]=sc.nextInt(); } } public static void mostrar_vector() { for(i=n-1;i>=0;i--) { System.out.print("la edad es " + edades[i]); } } public static void main(String[] args) { crear_vector(); llenar_vector(); mostrar_vector(); } } </pre>
---	--

Algoritmos de ordenación

Ordenación por selección



```
static void ordenarSeleccion (int v[])
{
    int tmp;
    int i, j, pos_min;
    int N = v.length;

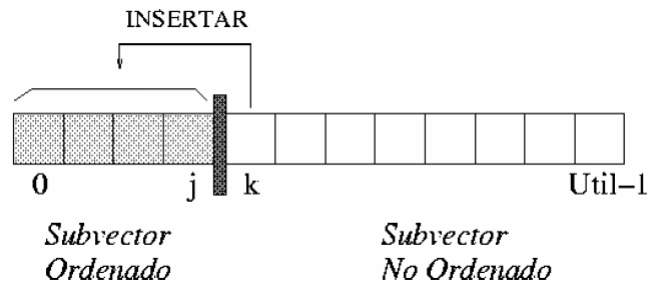
    for (i=0; i<N-1; i++)
    {
        // Menor elemento del vector v[i..N-1]
        pos_min = i;

        for (j=i+1; j<N; j++)
            if (v[j]<v[pos_min])
                pos_min = j;

        // Coloca el mínimo en v[i]
        tmp = v[i];
        v[i] = v[pos_min];
        v[pos_min] = tmp;
    }
}
```

En cada iteración, se selecciona el menor elemento del subvector no ordenado y se intercambia con el primer elemento de este subvector.

Ordenación por inserción

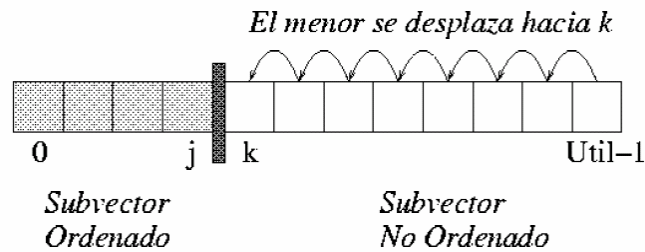


```
static void ordenarInsercion (int v[])
{
    int tmp;
    int i, j;
    int N = v.length;

    for (i=1; i<N; i++)
    {
        tmp = v[i];
        for (j=i; (j>0) &&(tmp<v[j-1]);j--)
            v[j] = v[j-1];
        v[j] = tmp;
    }
}
```

En cada iteración, se inserta un elemento del subvector no ordenado en la posición correcta dentro del subvector ordenado.

Ordenación por intercambio directo (método de la burbuja)



```
static void ordenarBurbuja (int v[])
{
    int tmp;
    int i, j;
    int N = v.length;

    for (i=1; i<N; i++)
        for (j=N-1; j>=i; j--)
            if (v[j] < v[j-1])
            {
                tmp = v[j];
                v[j] = v[j-1];
                v[j-1] = tmp;
            }
}
```

Este método consiste en ir comparando cada par de elementos del arreglo e ir moviendo el mayor elemento hasta la última posición, comenzando desde la posición cero. Una vez acomodado el mayor elemento, prosigue a encontrar y acomodar el segundo más grande comparando de nuevo los elementos desde el inicio de la lista, y así sigue hasta ordenar todos los elementos del arreglo

Ordenación Shellsort

```
public static void Shellsort (int[] a)
{
    int salto=a.length/2;
    while(salto>=1)
    {
        for(int rec=salto;rec<a.length;rec++)
        {
            int temp=a[rec];
            int j=rec-salto;
            while(j>=0&& a[j]>temp)
            {
                a[j+salto]=a[j];
                j-=salto;
            }
            a[j+salto]=temp;
        }
        salto/=2;
    }
}
```

Este método es una mejora del algoritmo de ordenamiento por Inserción. El algoritmo Shell sort mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del Shell sort es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

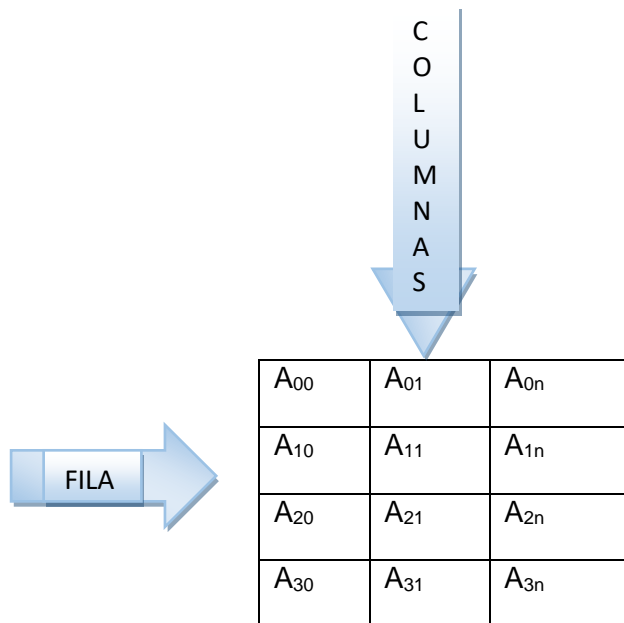
Busqueda Binaria

```
public static int busquedaBinaria( int [] arreglo, int dato)
{
    int inicio = 0;
    int fin = arreglo.length - 1;
    int pos;
    while (inicio <= fin)
    {
        pos = (inicio+fin) / 2;
        if ( arreglo[pos] == dato )
            return pos;
        else
            if ( arreglo[pos] < dato )
            {
                inicio = pos+1;
            }
            else
            {
                fin = pos-1;
            }
    }
    //cierra el ciclo mientras
    return -1;
} //cierra metodo
```

Se utiliza cuando el vector está previamente ordenado. Este algoritmo reduce el tiempo de búsqueda considerablemente, ya que disminuye exponencialmente el número de iteraciones necesarias.

Para implementar este algoritmo se compara el elemento a buscar con un elemento cualquiera del arreglo (normalmente el elemento central): si el valor de éste es mayor que el del elemento buscado se repite el procedimiento en la parte del arreglo que va desde el inicio de éste hasta el elemento tomado, en caso contrario se toma la parte del arreglo que va desde el elemento tomado hasta el final. De esta manera obtenemos intervalos cada vez más pequeños, hasta que se obtenga un intervalo indivisible. Si el elemento no se encuentra dentro de este último entonces se deduce que el elemento buscado no se encuentra en todo el arreglo.

Arreglos Bidimensionales (Matrices): las matrices son un arreglo bidimensional, rectangular de datos dispuestos en filas y columnas, es decir se manejan dos subíndices Para efectos de este manual, la primera posición será la posición 0 (cero).



Sentido horizontal (Filas)

Sentido vertical (columnas)

Cada término de la matriz lo denominamos $A_{i,j}$ en donde los subíndices i, j determinan la fila y la columna del elemento de la matriz

B =

20	10
15	45

La matriz B es de orden 2x2 porque tiene 2 filas y 2 columnas

$B_{00} = 20$ $B_{01} = 10$ $B_{10} = 15$ $B_{11} = 45$

Nota: las matrices cuyo número de filas es igual al número de columnas recibe el nombre de matrices cuadradas entonces decimos que son de orden N

- ¿Cómo se declara una Matriz?

Como las matrices en la POO son objetos se deben de instanciar

- `<Tipo de dato><nombre de la matriz> <[][]> = <new> <tipo de dato><[filas][columnas]>`

Ejemplo

```
int Edades [][] = new int [6] [5];
```

Valores constantes (el compilador deduce automáticamente la dimensión del arreglo)

Ejemplo

```
int Edades[][] = {1,2,3}, {4,5,6} };
```

- ¿Cómo se ingresa un dato a la matriz?

`<Nombre de la matriz> <[fila] [columna]> = <dato del scanner>`

EJEMPLO:

```
Edades [3] [5] = sc.nextInt();
```

- ¿Cómo se muestra un dato de la matriz?

Muestre `<nombre de la matriz> <[fila] [columna]>`

EJEMPLO:

```
System.out.print(Edades [2] [3]);
```

Para recorrer una matriz ya sea para llenarlo o mostrar sus elementos se usan dentro de dos ciclos dos variables que controlan los índices de la matriz

Ejemplo

```
for (f=0;f<=5;f++)
{
    for (c=0;c<=5;c++)
    {
        System.out.print("digite su edad");
        edades [f]
        [c]=dato.nextInt();
    }
}
```

Nota: Las posiciones de la matriz deben ser dos números enteros, variables enteras o el resultado entero de un cálculo matemático.

Observación: al igual que los vectores podemos utilizar el atributo length para recorrer una matriz.

El atributo length almacena la cantidad de filas de la matriz

Ejemplo

```
System.out.print ("Cantidad de filas de la matriz:" + matriz.length)
```

también podemos preguntarle a cada fila de la matriz la cantidad de elementos que almacena:

Ejemplo

```
System.out.print ("Cantidad de elementos de la primer fila:" + matriz[0].length)
```

EJEMPLO:

Hacer un algoritmo que almacene números en una matriz de 5 * 6. Imprimir la suma de los números almacenados en la matriz.

ALGORITMO	JAVA
<pre>publico clase Matriz_Numeros privado entero numeros[5][6] privado entero f,c publico estatico vacio metodo llenar_matriz () para(f=0; f<= 4 ; f=f+1) para (c=0; c<=5;c=c+1) mprima("Digite un numero") lea numeros[f][c] finpara finpara calcular_suma(); finmetodo publico estatico vacio metodo calcular_suma () entero suma= 0 para(f=0; f<=4;f=f+1) para (c=0; c<=5;c=c+1) suma= suma+números[f][c] finpara finpara imprima("la suma es"+ suma) finmetodo publico estatico vacio principal() llenar_matriz() finmetodo finclase</pre>	<pre>import java.util.Scanner; public class Matriz_Numeros { private static int f,c; private static int numeros[][] = new int [5][6]; private static Scanner sc = new Scanner(System.in); public static void llenar_matriz() { for(f=0;f<=4;f++) { for(c=0;c<=5;c++) { System.out.print("digite un numero"); numeros[f] [c]=sc.nextInt(); } } calcular_suma(); } public static void calcular_suma() { Int suma=0; for(f=0;f<=4;f++) { for(c=0;c<=5;c++) { Suma= suma + numeros[f] [c]; } } System.out.print("la suma es"+suma); } public static void main(String[] args) { llenar_matriz(); } }</pre>

13. EXCEPCIONES (EXCEPTION Y TRY-CATCH)

- Cuando en java puede fallar algo, por ejemplo, la conversión de la cadena en int, suele avisarnos. Esto lo hace "lanzando excepciones". Una excepción es algo que lanza Java para avisarnos que ha habido un problema.
- En nuestro código podemos "capturar" esas excepciones y hacer algo para tratar de arreglarlo. Por ejemplo, si le pedimos al usuario que ingrese un d3etynúmero, el usuario escribe "abc", lo leemos e intentamos convertirlo a int, nos salta una excepción. Si capturamos esa excepción, en el código que hagamos para tratarla, podemos avisar al usuario que se ha equivocado un poco al teclear, que lo intente de nuevo y volver a pedirle el número.
- Para capturar una excepción, tenemos que hacer el código así:

```
Try
{
    // Aquí el código que puede fallar
}

catch (Exception e)
{
    System.out.println(e);
    e.printStackTrace();
}
```

Para mostrar el error que causo la excepción se pude hacer de dos formas:

Por un lado, con `System.out.println(e);` Esto nos muestra una línea de texto con el error correspondiente.

Por otro lado, llamando al método `printStackTrace()` de la excepción que se ha provocado. Esta llamada escribe la misma línea de texto de error, pero además nos dice exactamente en qué línea de código se produce el error. Esta información es muy útil si nuestro programa todavía está en construcción y no acaba de hacer lo que queremos.