ELEMENTOS DE UN PROGRAMA INFORMÁTICO EN JAVA.

Contenidos:

1) Estructura y bloques fundamentales.
2) Variables.
3) Tipos de datos.
4) Literales.
5) Constantes.
6) Operadores y expresiones.
7) Conversiones de tipo.
8) Comentarios.
9) Estructuras: selección y control.

Comentarios.

- // Comentario de una línea

```
/* Inicio del comentario de varias líneas
Comentario1
Comentario2
Comentario3
Comentario4
Fin del comentario de varias líneas*/
```

- Los comentarios no tienen efecto como instrucciones para el ordenador, simplemente sirven para que cuando una persona lea el código pueda comprender mejor lo que lee

Estructura y bloques fundamentales.

- El código fuente de un programa en Java debe estar escrito en un fichero de texto con extensión **".java"**.

```
public class NombreClase
 {
      public static void main(String args[])
      { //Inicio del método
            System.out.println("Hola Mundo!");
      } //Fin del método
}
```

- El nombre de la clase debe corresponder exactamente con el nombre del fichero de texto que contiene el código fuente.

Estructura y bloques fundamentales.

- **public static void main(String args[])**
- La ejecución del programa Java comenzará por el código contenido en este método.

- System.out.println("Hola Mundo!");
- La llamada a System.out.println() que permite mostrar en pantalla una serie de caracteres.

Variables

- Las **variables** identifican datos mediante un nombre simbólico, haciendo referencia a una **dirección de memoria principal** en los que se sitúan los datos.
- Los nombres utilizados para identificar a las variables deben cumplir una serie de condiciones:
  - ✓ No pueden empezar por un dígito numérico.
  - ✓ No pueden utilizarse espacios, y los únicos caracteres especiales válidos son el guión bajo (_) y el símbolo del dólar ($).
  - ✓ Son sensibles a las mayúsculas y minúsculas, es decir, las variables "suma" y "Suma" se consideran variables distintas.

Variables

- No podemos utilizar como nombres las **palabras reservadas** de **JAVA**:

```
abstract  default   goto
assert    do        if
boolean   double    implements
break     else      import
byte      enum      instanceof
case      extends   int
catch     false     interface
char      final     long
class     finally   native
const     float     new
continue  for       null
package   synchronized
private   this
protected throw
public    throws
return    transient
short     true
static    try
strictfp  void
super     volatile
switch    while
```

Variables

- Nombenclatura:
  - ✓ Los nombres de las variables han de empezar por una letra minúscula.
  - ✓ Cuando el nombre de una variable está formado por más de una palabra, se suele utilizar una letra mayúscula para distinguir el comienzo de las palabras. Por ejemplo: sumaTotal.
  - ✓ Es **recomendable** utilizar nombres que hagan referencia al contenido que va a almacenar para facilitar la comprensión del código. Es mucho más claro utilizar el nombre "suma" que "s".
  - ✓ Ejemplos de nombres de variables válidos: indice, ventas, compras, saldoGeneral, importetotal, contador_lineas, $valor, num2.
  - ✓ Ejemplos de nombres de variables no válidos: 3valores, suma&total, super, edad media.

Variables

- Podemos definir variables:
  - tipoDato nombreVariable;
  - tipoDato nombreVariable1, nombreVariable2, nombreVariable3;

- Ejemplos de declaraciones de variables:
  - int num1, num2, suma;
  - char letraNIF;
  - String texto;
  - boolean mayorEdad;

Variables

- Asignaciones:
  - Inicialización: Se le asigna un valor a la variable en el momento de su inicialización
    - int num1 = 34;
    - int doble = num1 * 2;
    - String saludo = "Hola";
    - char letraA = 'A', letraB = 'B';
  - PostDefinición: Se le asignará un valor a la variable anteriormente definida, modificando cualquier valor previo:
    - nombreVariable = valor;
    - int a=5, b=0, c;
    - b = a * 3; // Se cambia el valor de b a 15
    - c = a; // Se guarda en c el valor de a que es 5
    - a = a + 6; // Se suma 6 al valor que tenía a.
    - // Ahora vale 11
    - b = a - c; // b guarda 11 - 5 que es 6

Variables

- Ámbito de una variable:
  - Las variables pueden ser utilizadas en el bloque de código en el que han sido definidas,es decir:

```
public class AmbitoVariables
{
    static int variableGlobal;
    public static void main(String[] args)
    {
        int variableDelMain = 10;
        /*Aquí se pueden usar variableGlobal y  variableDelMain. No se puede usar  variableDeOtroMetodo */
        System.out.println("variableGlobal " + variableGlobal);
        System.out.println("variableDelMain " + variableDelMain);
        otroMetodo();
    }
    static void otroMetodo()
     {
        int variableDeOtroMetodo=90;
        /* Aquí se pueden usar variableGlobal y  variableDeOtroMetodo. No se puede usar  variableDelMain */
        System.out.println("variableGlobal " + variableGlobal);
        //System.out.println("variableDelMain " + variableDelMain);
        System.out.println("variableDeOtroMetodo " + variableDeOtroMetodo);
    }
}
```

Tipos de datos en JAVA

- **Números enteros:** Representan a los números enteros (sin parte decimal) con signo (pueden ser positivos o negativos). Se dispone de varios tipos de datos, ocupando cada uno de ellos un espacio distinto en memoria. Cuanta más capacidad de almacenamiento, más grande es el rango de valores permitidos, aunque ocupará más espacio de memoria principal. Se dispone de los siguientes tipos:
  - ✓ **byte**: Ocupan 8 bits (1 byte), permitiendo almacenar valores entre -128 y 127.
  - ✓ **short**: Ocupan 16 bits (2 bytes), permitiendo almacenar valores entre -32.768 y 32.767.
  - ✓ **int**: Ocupan 32 bits (4 bytes), permitiendo almacenar valores entre -2.147.483.648 y 2.147.483.647. Es el tipo de datos por defecto para los valores numéricos enteros. Este tipo de datos es lo suficientemente grande para almacenar los valores numéricos que vayan a usar tus programas. solo se suelen usar los tipos anteriores si se pueden producir problemas con el espacio de memoria.
  - ✓ **long**: Ocupan 64 bits (8 bytes), permitiendo almacenar valores entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807.

- **Números reales**: Representan a los números reales con parte decimal y signo positivo o negativo. Hay dos tipos de datos numéricos reales que permiten obtener mayor o menor precisión. Utilizan un método para almacenar los datos que puede ocasionar que el valor original varíe levemente del valor almacenado realmente. Cuanta más precisión se utilice, habrá menor variación.
  - ✓ **float**: Ocupan 32 bits (4 bytes). Se le denomina de simple precisión. Almacenan valores desde -3.40282347E+38 a + 3.40282347E+38
  - ✓ **double**: Ocupan 64 bits (8 bytes). Se le denomina de doble precisión. Es el tipo de datos por defecto para los valores numéricos reales. Almacenan valores desde: - 1.79769313486231570E+308 a +1.79769313486231570E+308

Valores literales

- Los valores correspondientes a los tipos de datos numéricos enteros (byte, short, int y long) se pueden expresar usando el sistema numérico decimal, octal o hexadecimal.
- Para representar un valor numérico entero en sistema octal, debe ir precedido del carácter 0 (cero), por ejemplo, el valor 284 se representa en octal como 0434. Asimismo, para representar un valor en el sistema hexadecimal, se debe emplear el prefijo 0x, por lo que el valor 284 se representa en hexadecimal como 0x11C.
- Para representar valores literales de los tipos de datos numéricos reales (float y double) se puede emplear el sistema decimal o la notación científica. Por ejemplo, el valor 21.843,83 se debe expresar como 21843.83 en el código fuente.
- Los valores literales de los tipos char (carácter) y String (cadena de caracteres), pueden contener cualquier carácter Unicode. Los valores de tipo char se debe expresar encerrados entre comillas simples, por ejemplo, la letra A se debe expresar como 'A'. Por otro lado, las cadenas de caracteres (tipo String) se expresan entre comillas dobles, por ejemplo, el texto Saludos a todos, se debe escribir como "Saludos a todos".

Valores literales

- Los caracteres Unicode que no se encuentre en el teclado, se puede hacer indicando el código hexadecimal correspondiente a dicho carácter precedido del modificador \u. Se debe encerrar entre comillas simples o dobles según se vaya a tratar como carácter o dentro de una cadena de caracteres. Por ejemplo, para escribir la letra griega beta (ß) se puede emplear '\u00DF', o para escribir la palabra España es posible utilizar "Espa\u00F1a".

- El lenguaje de programación Java también soporta un pequeño conjunto de caracteres especiales que se pueden utilizar en los valores literales char y String:

  - ✓ \b (retroceso).
  - ✓ \t (tabulador).
  - ✓ \n (nueva línea).
  - ✓ \f (salto de página).
  - ✓ \r (retorno de carro).
  - ✓ \" (comilla doble).
  - ✓ \' (comilla simple).
  - ✓ \\ (barra invertida).

Constantes

- Similares a las variables: son datos a los que se hace referencia mediante un nombre y a los que se les asigna un valor.
- A diferencia de las variables, a las constantes no se les puede modificar el valor asignado.
  - final tipoDato nombreConstante = valor;
- Ejemplo de declaración:
  - final double PI = 3.1415926536;

Operadores

- **Operadores Aritméticos**:
  - ✓ + (suma)
  - ✓ - (resta)
  - ✓ * (multiplicación)
  - ✓ / (división entera o con decimales según operandos)
  - ✓ % (resto de la división)

- Ejemplos:
  - ✓ 4 + 3
  - ✓ 8 - 5 + 2
  - ✓ 6 * 2 / 3
  - ✓ 8.5 - 3 + 4.3

Operadores

- **Operadores Relacionales**
  - Los operadores relacionales permiten comparar dos valores numéricos.
  - > (mayor que)
  - >= (mayor o igual que)
  - < (menor que)
  - <= (menor o igual que)
  - == (igual que)
  - != (distinto de)
- Ejemplos:
  - 4 > 3 resulta true.
  - 7 <= 2 resulta false.
  - 5 + 2 == 4 + 3 resulta true.
  - 4 * 3 != 12 resulta false.

Operadores

- **Operadores Lógicos**
  - Los operadores lógicos permiten unir valores o expresiones lógicas, obteniendo como resultado si es verdadera o falsa la expresión combinada.
  - && (Y lógico - conjunción) => Resulta true solo si ambos operandos son true.
  - || (O lógico - disyunción) => Resulta true si al menos uno de los operandos son true.
  - ! (NO lógico) => Resulta true si el operando es false.

- Ejemplos:
  - 4 > 3 && 5 <= 5 resulta true, porque las dos expresiones son true.
  - 4 > 3 && 5 != 5 resulta false, porque al menos una expresión es false.
  - 4 > 3 || 5 != 5 resulta true, porque al menos una expresión es true.
  - 4 > 3 && !(5 != 5) resulta true, porque las dos expresiones son true.

Operadores

- **Operadores Incrementales**
  - ✓Nos permiten incrementar las variables en una unidad
  - ✓++ (incrementa en 1 el valor de una variable).
  - ✓-- (decrementa en 1 el valor de una variable).

- Ejemplo:
  - ✓int x=5, y=5, z;
  - ✓z=x++; /* z vale 5, x vale 6 porque primero se asigna
  - ✓el valor de x a z después se incrementa x */
  - ✓z=++y; /* z vale 6, y vale 6 porque primero incrementa
  - ✓la variable y, después se asigna el valor a z */

Operadores

- **Operador Condicional.**
  - Permite asignar a una variable un valor u otro dependiendo de una expresión condicional. El formato es el siguiente:
  - variable = condición ? valor1 : valor2;
  - La condición que se indica debe ser una variable booleana o una expresión condicional que resulte un valor de tipo booleano (true o false).
- Ejemplo:
  - mensaje = (num1 >= 0) ? "Positivo" : "Negativo";

Operadores: Prioridad

- Los operadores del mismo nivel, fila, tienen la misma prioridad.
- Los operadores de distinto nivel, inferior, tienen menor prioridad.

()	**[]**	**.**			
-(unario)	**--**	**++**	**!**		
new (tipo)					
*****	**/**	**%**			
+	**-**				
>	**>=**	**<**	**<=**		
==	**!=**				
&&					
||					
? :					
=	**+=**	**-=**	***=**	**/=**	**%=**

Conversiones de tipo

- **Conversiones implícitas**:
  - Se realizan de manera automática, es decir, el valor o expresión que se va a asignar a una variable es convertido automáticamente por el compilador.

- Ejemplos:

```
// Declaraciones.
int k = 5, p;
short s = 10;
char c = 'ñ';
float h;

// Conversiones implícitas.
p = c; // Conversión implícita de char a int.
h = k; // Conversión implícita de int a float.
k = s; // Conversión implícita de short a int.
```

Conversiones de tipo

- **Conversiones explícita**:
  - Cuando no se cumplan las condiciones para una conversión implícita, ésta podrá realizarse de manera explícita utilizando la expresión:
  - variable_destino = (tipo_destino) dato_origen;

- Ejemplos:
  - // Declaraciones.
  - char c;
  - byte k;
  - int p = 400;
  - double d = 34.6;
  - // Conversiones explícitas.
  - c = (char)d; // Se elimina la parte decimal (trunca), no se redondea.
  - k = (byte)p; // Se provoca una pérdida de datos, pero la conversión es posible.

Estructura de selección

Pseudocódigo

```
Proceso Ejemplo
    Definir A Como Entero;
    Leer A;
    Si A > 0 Entonces
        Escribir "Bonito número";
    SiNo
        Escribir "Menor o igual que 0 :(";
    FinSi
FinProceso
```

JAVA

```
public static void main(String[] args)
{
    int a;
    Scanner input = new Scanner(System.in);
    a = input.nextInt();
    if(a > 0)
        System.out.println("Bonito número");
    else
        System.out.println("Menor o igual que 0 :(");
}
```

Operaciones: estructuras de Control

Pseudocódigo

```
1  Proceso Ejemplo
2      Definir A Como Entero;
3      A <- 0;
4      Mientras A < 10 Hacer
5          Escribir A;
6          A <- A + 1;
7      FinMientras
8  FinProceso
```

JAVA

```
public static void main(String[] args)
{
    int a = 0;
    while (a < 10)
    {
        System.out.println(a);
        a = a + 1;//++a;
    }
}
```

Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
    Definir A Como Entero;
    A <- 0;
    Repetir
        Escribir A;
        A <- A +1;
    Hasta Que A >= 10
FinProceso
```

JAVA

```
public static void main(String[] args)
{
    int a = 0;
    do
    {
        System.out.println(a);
        a = a + 1;//++a;
    }
    while (a >= 10);
}
```

Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
    Definir A Como Entero;
        Para A<-0 Hasta 9 Hacer
        Escribir A;
    FinPara
FinProceso
```

JAVA

```
public static void main(String[] args)
{
    for(int i =0;i<10;i++)
    {
        System.out.println(i);
    }
}
```

Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
    Definir A Como Entero;
    Leer A;
    Segun A Hacer
        1:
            Escribir 'Has introducido 1';
        2:
            Escribir 'Has introducido 2';
        3:
            Escribir 'Has introducido 3';
        De Otro Modo:
            Escribir 'Has introducido otro valor';
    FinSegun
FinProceso
```

JAVA

```
public static void main(String[] args)
{
    int a;
    Scanner input = new Scanner(System.in);
    a = input.nextInt();
    switch(a)
    {
    case 1:
        System.out.println("Valor 1");
    break;

    case 2:
        System.out.println("Valor 2");
    break;

    case 3:
        System.out.println("Valor 3");
    break;
    default:
        System.out.println("Has introducido otro valor");
    break;
    }
}
```