TEMA 2 : INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN JAVA: *ELEMENTOS DEL LENGUAJE.*

1. Programación Orientada a Objetos.
2. Elementos del lenguaje
   - ✓ Tipos de datos.
   - ✓ Identificadores.
   - ✓ Declaración de variables.
   - ✓ Operadores.
   - ✓ Constantes, Literales.
3. Entrada / Salida.
4. Estructuras de control.

- PROGRAMACIÓN ORIENTADA A OBJETOS

Es una evolución lógica de la programación estructurada, en la que el concepto de variable local se hace extensible a los propios subprogramas que acceden a estas variables. El elemento central de la programación orientada a objetos POO es la clase.

Una clase determina el comportamiento y las características propias de sus componentes. A los casos particulares de una clase se les denomina objetos.

Un programa se entiende como un conjunto de objetos que interactúan entre sí.

Ventajas de la POO

- Facilita la reutilización de código.
- Permite ocultar detalles no relevantes.
- Su ejecución es independiente de la plataforma. Se ejecuta igual en diferentes sistemas.

Características del objeto ⇓ ATRIBUTOS	Operaciones (procedimientos y funciones) ⇓ MÉTODOS
Ejemplo 1: Un animal es un objeto **Características:** Nombre. Nº de patas, etc..	**Operaciones:** Criar. Lavar. Hablar. Etc.
Ejemplo 2: Una casa es un Objeto **Características:** Nº de pisos. Color de fachada. Nº de ventanas etc...	**Operaciones:** Construir. Destruir. Pintar fachada. Abrir una ventana.

Propiedades

- Encapsulamiento
- Herència
- Polimorfismo

Encapsulamiento. Es la propiedad que tienen los objetos de ocultar sus atributos, e incluso sus métodos, a otras partes del programa o a otros objetos.

Ejem.

```
class Ccasa {
int nPuertas, nVentanas ;
String color;
public Ccasa (int np, int nv, String co ) {
        nPuertas =np ;
        nVentanas =nv ;
        color = co;
       }
 public void pintar (String co ) {
 color =co;
}
 public void abrirVentanas (int n) {
 nVentanas = nVentanas + n ;
  }
 public void cerrarVentanas (int n ){
    nVentanas = nVentanas - n ;
    if ( nVentanas <0)  nVentanas =0;
 }
 public void abrirPuertas (int  n)  {
     nPuertas = nPuertas + n) ;
 }
 public void cerrarPuertas (int  n) {
   nPuertas = nPuertas - n) ;
   if (nPuertas < 0) nPuertas = 0;
 }
 .....
}
Ccasa  casa1, casa2 ;
```

Herencia. Es una de las principales ventajas de la POO. Permite definir clases descendientes de otras, de forma que la nueva, hereda de la antecesora todos sus **atributos y métodos.** Esta propiedad permite la reutilización de código de clases, modificándolas mínimamente para **adaptarlas** a nuevas especificaciones.

Ejem. Tenemos construida la clase Ccasa y queremos definir una nueva clase que represente a los chalets.

class Cchalet extends Ccasa { **int mJardin;** **public Cchalet (int np, int nv, String co, int mj) {** **super (np , nv , co) ;** **mJardin = mj ;** **}** **}** **……..** ***Cchalet chalet1, chalet2 ;*** ***……..***	✓ Cchalet es descendiente de Ccasa y declaramos un nuevo atributo, ***mJardin.*** ✓ *El método constructor hay que redefinirlo para poder inicializar el nuevo atributo mJardin.* ✓ Para utilizar el método pintar heredado de la clase Ccasa ***Chalet.pintar ("Blanco")***

Polimorfismo. Permite que un mismo mensaje enviado a objetos de clases distintas, haga que estos se comporten también de forma distinta. Objetos distintos pueden tener métodos con el mismo nombre.

Un mismo objeto puede tener métodos con el mismo nombre , pero con distintos parámetros.

class Ccasa {
…….
public Ccasa (int np, int nv, String co) {
nPuertas =np ;
nVentanas =nv ;
color = co;
}
public Ccasa ( ) {
nPuertas = 0 ;
nVentanas = 0 ;
color = 0 ;
}
	• Dos métodos con el mismo nombre pero parámetros distintos.
	• Métodos sobrecargados.

- ELEMENTOS DEL LENGUAJE

  - ✓ TIPOS DE DATOS

  - – **Simples**

Tipo	Byte	Rango
Byte	1	-128 - 127
Short	2	-32 .768 - 32 . 767
int	4	-2.147.483.648 - 2.147.483.647
long	8	9.223.372.036.854.775.808 - 9. 223.372. 036.854.775.807
float	4	$+/-3,4*10^{-38}$ - $+/-3,4*10^{38}$
Double	8	$+/-1,7*10^{-308}$ - $+/- 1,7*10^{308}$

Además, todos estos tipos de datos simples, pueden ser declarados como referenciales (objetos), ya que existen clases que los engloban, llamadas clases envolventes.

Tipos de datos simples	**Clase equivalente**
byte	java.lang.Byte
short	java.lang.Short
int	java.lang.Integer
long	java.lang.Long
float	java.lang.Float
double	java.lang.Double
char	java.lang.Character
boolean	java.lang.Boolean

  - – ***Referenciales.***

Estos tipos son básicamente las clases, en las que se basa la ***POO.***

Al declarar un objeto perteneciente a una determinada clase, se está reservando una zona de memoria donde se almacenarán los atributos y otros datos pertenecientes a dicho objeto. Lo que se almacena en el objeto en sí, es un puntero (referencia) a dicha zona de memoria. Dentro de estos tipos pueden considerarse los *Strings,* los vectores y las interfaces.

✓ IDENTIFICADORES

Los identificadores son los nombres que se les damos a las variables, clases, interfaces, atributos y métodos de un programa.

Reglas para la creación de identificadores:

1.- Java **hace distinción entre mayúsculas y minúsculas**, por lo tanto, nombres o identificadores como var1, Var1 y VAR1 son distintos.

2.- Pueden estar formados por cualquiera de los caracteres del código Unicode.

añoDeCreación, raïm, etc

El primer carácter no puede ser un dígito y no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores (aritméticos, lógicos de relación..).

3.- La longitud máxima de los identificadores es prácticamente ilimitada.

4.- No pueden ser palabras reservadas del lenguaje ni los valores lógicos true o false.

5.- No pueden ser iguales a otro identificador declarado en el mismo ámbito.

6.- Por convenio, los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula. Además, si el identificador está formado por varias palabras, la primera se escribe en minúsculas (excepto para las clases) y el resto de palabras se empiezan por mayúscula.

ejemplo: añoDeCreación. Es más sencillo distinguir entre clases y métodos o variables.

Estas reglas no son obligatorias, pero son convenientes ya que ayudan al proceso de codificación de un programa, así como a su legibilidad.

Ejemplos válidos	**NO válidos**
añoDeNacimiento2	3valores
_otra_variable	Dia&mes&año
BotónPulsación.	Dos más

✓ DECLARACIÓN DE VARIABLES

La declaración de una variable siempre contiene **el nombre** (identificador de la variable) y **el tipo** de dato al que pertenece.

Ejem. int x ;

Se pueden inicializar en el momento de su declaración, siempre que el valor que se les asigne coincida con el tipo de dato de la variable.

Ejem: int x = 0;

Para declarar múltiples variables del mismo tipo pueden declararse, en forma de lista, separadas por comas. Ejem. int x=0 , y, z=3 ;

Si una variable no ha sido inicializada, tiene un valor asignado por defecto.

Tipo referencial (objetos)	valor null.
Tipo numérico	cero (0).
Tipo char,	el valor '\u0000' .
Tipo boolean,	el valor false.

- **Ámbito de las variables.** El ámbito de una variable es la porción de programa donde dicha variable es visible para el código del programa y, por tanto, referenciable. El ámbito depende del lugar del programa donde se declara, pudiendo pertenecer a cuatro categorías distintas:
  - Variable local.
  - Atributo.
  - Parámetro de un método.
  - Parámetro de un tratador de excepciones

- **Variables locales.** Una variable local se declara dentro *del cuerpo de un método de una clase* y es visible únicamente dentro de dicho *método.* Se puede declarar en cualquier lugar del cuerpo, incluso después de instrucciones ejecutables, aunque es una buena costumbre declararlas justo al principio. También pueden declararse variables dentro de un bloque parentizado por llaves { … }. En ese caso, sólo serán "visibles" dentro de dicho bloque.

```
class Caracter  {
  char ch;
  public Caracter (char c) {
  ch = c;
   }
  public void repetir(int num) {
   int  i  ;
   for (i=0; i<num; i++)
        System.out.println(ch);
   }
}
```

```
class   Ej1 {
   public static void main  (String args [ ] ) {
   Caracter   car ;
     car = new Caracter ( 'H' ) ;
     Caracter.repetir (20);
      }

}
```

En este ejemplo existe una variable local: **int** i; definida en el método repetir de la clase Caracter, por lo tanto, únicamente es visible dentro del método repetir ; una variable local en el método main de la clase Ej1,, en este caso es un objeto, **Caracter** car, que sólo será visible dentro del método en el que está declarada (main). Esta declaración indica al compilador el **tipo** de la variable **car** pero no crea un objeto. El operador que crea el objeto es **new**, que necesita como único parámetro el nombre del *constructor* (que será el procedimiento que asigna valor a ese objeto recién instanciado).

- ✓ OPERADORES**.**

Operadores aritméticos

+	Suma
-	Resta
*	Multiplicación.
/	División
%	Módulo
++	Incremento en 1
--	Decremento en 1

Ejemplo

Operadores aritméticos

```
class opera1 {
   public static void main (String args [ ] ) {
      int  x=33;
      double  y = 33.33;

     System.out.println (" x mod 10 = " + x%10);
      }
  } //cierra la clase
```

Visualiza: x mod 10 = 3

Operadores relacionales

>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual
==	Igual
!=	Distinto**.**

Actúan sobre valores enteros, reales y caracteres (char) y devuelven un valor del tipo *boolean (true o false)*

Operadores lógicos

&&	Y lógico
||	O lógico
!	Negación

Estos operadores actúan sobre operandos o expresiones lógicas

Operadores de asignación

+=	Suma y asignación	**/=**	División y asignación
-=	Resta y asignación	%=	Módulo y asignación
***=**	Multiplica y asigna		

Precedencia de operadores en Java:

Operadores postfijos	[ ] . (paréntesis)
Operadores unarios	++expr --expr -expr !
Creación o conversión de tipo	new (tipo) expr
Multiplicación y división	* / %
Suma y resta	+ -
Relacionales	< > <= >=
Igualdad y Desigualdad	= = !=
AND lógico	&&
OR lógico	||
Condicional al estilo C	? :
Asignación	= += - = *= /= %= =

Ejem: Operador condicional ? :

```
class opera2 {
public static void main (String args [ ] ) {
  int a=28 , b=4,  c=45, d=0;
  int e = (b==0) ? 0 : (a / b);
  int f = (d==0) ? 0 : (c / d);

   System.out.println(" a = " + a) ;
   System.out.println(" b = " + b) ;
   System.out.println(" c = " + c) ;
   System.out.println(" d = " + d) ;
    System.out.println(" e = " + e) ;
   System.out.println(" f  = " + f) ;
 }
}
```

Utilizamos el operador ternario para asignar valor a las variables **e** y **f** al mismo tiempo que las definimos

Visualiza:

✓ CONSTANTES Y LITERALES

Constantes. Las constantes en Java son declaradas como atributos de tipo **final.**

Valores literales. A la hora de tratar con valores de los tipos de datos simples (y Strings) se utiliza lo que se denominan "literales".

Los literales son elementos que sirven para representar un valor en el código fuente del programa. En Java existen literales para los siguientes tipos de datos:

Lógicos (boolean).	– Son únicamente dos: las palabras reservadas **true** y **false**.
Carácter (char).	– Se representan siempre entre comillas simples.
Enteros (byte, short, int y long).	– Pueden expresarse en decimal (base 10), octal (base 8) o hexadecimal (base 16). Además, puede añadirse al final del mismo la letra L para indicar que el entero es considerado como **long** (64 bits).
Reales (double y float).	– Existen dos formatos de representación: decimal y científica o exponencial. Igual que en los enteros, se puede poner una letra como sufijo. F Trata el literal como de tipo **float.** D Trata el literal como de tipo **double.**
Cadenas de caracteres (String).	– No forman parte de los tipos de datos elementales en Java, sino que son instanciados a partir de la clase java.lang.String pero aceptan su inicialización a partir de literales de este tipo. – Un literal de tipo *String* va encerrado entre comillas dobles (") y debe estar incluido completamente en una sola línea del programa fuente. Entre las comillas dobles puede incluirse cualquier carácter del código Unicode, además de las secuencias de escape de los literales de tipo carácter.

Entre las comillas simples puede aparecer:

- Un símbolo (letra) siempre que el carácter esté asociado a un código Unicode. ‘a’ , ‘B’ , ‘{‘ , ‘ñ’ , ‘á’ .

- Una “secuencia de escape”. Combinaciones del símbolo contrabarra \ seguido de una letra, y sirven para representar caracteres que no tienen una equivalencia en forma de símbolo. Las posibles secuencias de escape son:

Secuencia de escape	**Significado**	**La \ seguida de 3 dígitos octales.**	**Equivalente a**	**La \ seguida de u y 4 dígitos hexadecimales (el código Unicode).**	**Equivalente a**
’\’’	Comilla simple.			´	
’\”’	Comillas dobles	’\141’	’a’	\u0061’	’a’
’\\’	Contrabarra.	’\101’	’A’	’\u0041’ ’\u0022’	’A’
’\b’	Backspace	’\042’	’\”’		’\”’
’\n’	Cambio de línea.				
’\f’	Form feed.				
’\r’	Retorno de carro.				
’\t’	Tabulador.				

Siempre que sea posible, es aconsejable utilizar las secuencias de escape en lugar del código Unicode (en octal o hexadecimal),

Literales de tipo String. Ejemplo:

```
System.out.println(”Primera línea\nSegunda línea del string\n”);

System.out.println(”Hol\u0061”);.
```

ENTRADA / SALIDA

- Entrada como argumento de main

System.in
 	
System.out
	System.out.print("El resultado es: " + num);
	
	System.out.println("El resultado es: " + num);
	
	int c = System.in.read();
		class argmain **{**
		
		public static void main (String args [ ]) {
		
		for (int i = 0;i< args.length; i++)
		System.out.println(args[i]);
		
		}
		
		}

- Entrada y Salida de Datos : Con **System.in :**

Para leer una fuente de datos definimos tres objetos: canal, flujo y filtro.

canal identifica la fuente de datos , en el caso de teclado existe el objeto System.in que identifica la consola como periférico de entrada.

flujo es un objeto encargado de transferir los datos mediante un flujo de bytes indiferenciado .

filtro es el objeto que proporciona los métodos para interpretar el flujo de bytes y los convierte en datos primitivos o cadenas.

Declaración de las variables flujo y teclado de las clases InputStreamReader y BufferedReader respectivamente.

InputStreamReader flujo;

BufferedReader teclado ;

Construimos el objeto flujo:

flujo = new InputStreamReader (System.in) // el constructor recibe como argumento el objeto que identifica la fuente de datos.

Con la referencia del objeto **flujo** como argumento, construimos el objeto **filtro** que llamamos teclado:

teclado = new BufferedReader (flujo) ;

Con el objeto teclado leemos la cadena de caracteres escrita por el usuario a través de la consola:

Nombre = teclado.readLine() ;

Hacemos la declaración de los objetos y la construcción en una misma sentencia:

InputStreamReader flujo **= new InputStreamReader (System.in)**

BufferedReader teclado **= new BufferedReader (**flujo**) ;**

Para trabajar con las clases InputStreamReader y BufferedReader tendremos que importar las clases del paquete **java.io.***

Para leer un entero: **Para leer un real**

texto= teclado.readLine(); año= Integer.parseInt(texto) ;	texto = teclado.readLine(); altura = Double.parseDouble(texto) ;

Con System.out :

System.out.print("El resultado es: " + num);

System.out.println("El resultado es: " + num);

– Con la clase Scanner de java

La entrada de datos desde consola es muy sencilla y tiene un código simple y fácil de entender, sólo tendremos que crear un objeto de la clase Scanner que nos permita utilizar los métodos de dicha clase para leer una cadena, nextLine(), un entero nextInt() , pero aún da problemas el método que lee un Double, por lo que los reales los leeremos como cadenas y los convertiremos a reales.

Para poder utilizar la clase Scanner tenemos que importarla con la orden

import java.util.*;

Esta clase pertenece al paquete java.util y utilizando el asterisco importamos todas las clases de ese paquete.

```
import java.util.*;
 class ejer_1 {
  public static void main( String arg [] ) {
      Scanner teclado = new Scanner(System.in);
      String nombre;
      System.out.println(" ¿ como te llamas ?);
      nombre=teclado.nextLine();
      System.out.println(" Hola " + nombre);
      }
}
```

Ejem. Programa que lee tu nombre y visualiza un saludo

- ESTRUCTURAS DE CONTROL

- **Alternativas / Condicionales:** **if - else**

Ejemplo if anidado

if (expresión)
Bloque de instrucciones_1
else
Bloque de instrucciones_2
 	
(*) Esta función convierte un String a entero
	class Notas {
	public static void main (String args []) {
	int nota ;
	if (args.lenht < 1) { System.out.println ("Uso : Nota num");
	System.out.println ("Donde num = nº entre 0 y 10");
	}
	else **{** nota = Integer.valueOf (args[0]).intValue()) ; // (*)
	if (nota< 5) System.out.println("Insuficiente") ;
	else if (nota < 6) System.out.println("Suficiente") ;
	else if (nota < 7) System.out.println ("Bien") ;
	else if (nota <9) System.out.println("Notable");
	else System.out.println("Sobresaliente");
	}
	}// cierra main
	} // cierra la clase

Switch	import java.util.*;
	class DiaSemana {
	public static void main (String args [] {
	int dia ;
	Scanner teclado = new Scanner(System.in);
	dia= teclado.nextInt();
	if (dia < 1) System.out.println(”Dia erroneo ");
	else { swicht (dia) {
	case 1: System.out.println ("Lunes") ; break ;
	case 2: System.out.println("Martes") ; break ;
	case 3: System.out.println ("Mi,rcoles") ; break;
	case 4: System.out.println ("Jueves") ; break ;
	case 5: System.out.println ("Viernes") ; break ;
	case 6: System.out.println ("S bado"); break ;
	case 7: System.out.println ("Domingo");
	default : System.out.println("Este día no existe ") ;
	}
	}
	}

switch (expresión) {
case valor_1 : instrucciones1;
break ;
case valor_2 : instrucciones2;
break ;
:
case valor_N : instruccionesN ;
 	
[default : InstruccionesPorDefecto]
}	

- **Repetitivas : for do _ while while**

for (inicialización ; condición ; incremento)
{
bloque instrucciones ;
}
	inicialización, es una instrucción que se ejecuta una sola vez al inicio del bucle, normalmente para inicializar un contador.

condición , es una expresión lógica, que se evalúa al inicio de cada nueva iteración del bucle. Cuando dicha expresión se evalúe a false, se dejará de ejecutar el bucle .	**incremento**, es una instrucción que se ejecuta en cada iteración del bucle como si fuera la última dentro del bloque de instrucciones. Generalmente se trata de una instrucción de incremento o decremento de alguna variable.

Cualquiera de estas tres cláusulas pueden estar vacías, aunque SIEMPRE hay que poner punto y coma (;)

do {
 	
bloque instrucciones
 	
}
while (expresión);
	bloque instrucciones, se ejecuta siempre **al menos una vez**, y lo hará **mientras expresión se evalúe a true** Entre las instrucciones que se repiten deberá existir alguna que, en algún momento, haga que la expresión se evalúe a false, de lo contrario el bucle sería infinito.

while (expresión) { bloque instrucciones }	**bloque de instrucciones** se ejecuta mientras se cumple una condición. La expresión se evalúa ANTES de empezar a ejecutar por primera vez el bucle, si expresión es false en la primera iteración, el bloque de instrucciones **no se ejecutará ninguna vez**.

Las tres construcciones de bucle (for, do-while y while) pueden utilizarse indistintamente realizando unas pequeñas variaciones en el programa.

La serie de Fibonacci es una serie de números enteros que comienza con 0, 1.

A partir del tercer término, el siguiente término de la serie se calcula como la suma de los dos anteriores. De esta forma, la serie estaría formada por los siguientes enteros:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ….

El siguiente programa muestra en pantalla la serie de Fibonacci hasta el término que se le indique al programa:

Ejemplo for

```
class Fibonacci {
  public static void main(String args[ ] ) {
  int numTerm, v1=0, v2=1 ,aux, cont;
  Scanner teclado= new Scanner(System.in);
  numTer= teclado.nextInt();  //  validar  numTerm
  System.out.println ("0, 1") ;
     for (cont=2; cont< numTerm  ; cont++) {
        aux=v2 ;
        v2+=v1 ;
        v1=aux ;
        System.out.println(","+v2) ;
     }   // cierra el for
  teclado.close();
  }// cierra main
} // cierra la clase
```

Ejemplo while

```
class Fibonacci3 {
public static void main(String arg[ ] ) {
    int numTerm ,v1=1 ,v2=1, aux, cont=2;
  Scanner teclado= new Scanner(System.in);

  numTer= teclado.nextInt(); // validar
                                    numTerm
    System.out.print("0, 1");
     while (cont++ < numTerm) {
          aux=v2;
          v2=v2+v1;
          v1=aux
          System.out.println  ("","+v2);
     }
    System.out.println();
    teclado.close();

  } // cierra main()

} // cierra la clase
```

Ejemplo do – while

```
  class Fibonacci2 {
 public static void main(String args[ ] ) {
 int numTerm ,v1=0 ,v2=1, aux, cont=2;
 Scanner teclado= new Scanner(System.in);

   numTer= teclado.nextInt(); // validar
                                    numTerm


  pantalla.write("0 , 1");
   do {
        aux=v2;
        v2+=v1;
        v1=aux;
        System.out.print(", "+v2); }
    while (++cont<numTerm);
     teclado.close();
  }// cierra main()
}// cierra la clase
```

- **Saltos**

En Java existen dos formas de realizar un salto incondicional en el flujo "normal" de un programa., las instrucciones break y continue.

break sirve para abandonar una estructura de control, tanto de las alternativas (if-else y switch) como de las repetitivas (for, do-while y while).

En el momento que se ejecuta la instrucción break, el control del programa sale de la estructura en la que se encuentra.

continue sirve para transferir el control del programa desde la instrucción directamente a la cabecera del bucle (for, do-while o while) donde se encuentra.

- **Práctica**: Codifica en JAVA todos los algoritmos del tema anterior.