El lenguaje de programación Java

1. Introducción

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Java es, hoy en día, uno de los lenguajes de programación más populares, particularmente para aplicaciones de cliente-servidor de web.

Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente. (Puedes ampliar la información en el siguiente enlace: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)).

```
/* Ejemplo 1 */

public class Saludo {

        public static void main (String args [ ] )
        {
                System.out.println (“Hola mundo.”);
        }

}
```

Ejemplo de programa en Java

2. Tipos de datos primitivos

Los tipos de datos primitivos (predefinidos o simples) de un lenguaje de programación son los tipos de datos que se encuentran disponibles directamente. En Java existen los siguientes tipos de datos primitivos:

Dato	Tamaño	Rango
long	8 bytes	-9.233.372.036.854.775.808L a 9.233.372.036.854.775.807L
int	4 bytes	-2.147.483.648 a 2.147.483.647
short	2 bytes	-32.768 a 32.767
byte	1 byte	-128 a 127

Dato	Tamaño	Rango
float	4 bytes	$-3.40282347 \times 10^{-38}$ a $3.40282347 \times 10^{38}$
double	8 bytes	$-1.79769313486231570 \times 10^{-308}$ a $1.79769313486231570 \times 10^{308}$

Las constantes enteras son por defecto de tipo int y las constantes reales son por defecto de tipo double. Los tipos float y double disponen de tres valores especiales: infinito positivo (Infinity), infinito negativo (-Infinity) y NaN (Not a Number). Estos valores nos permiten representar situaciones como desbordamientos y errores.

Dato	Tamaño	Rango
char	2 bytes	Unicode

En Java los caracteres no se almacenan en un byte como en la mayoría de los lenguajes de programación. En Java se usa Unicode para representar los caracteres y por ello se emplean 16 bits para almacenar cada carácter. Al utilizar dos bytes para representar cada carácter, disponemos de un total de 65.535 posibilidades, suficiente para representar todos los caracteres de todos los lenguajes del planeta.

En Java existe un tipo de datos boolean para representar lo valores verdadero y falso.

Dato	Tamaño	Rango
boolean	1 byte	true o false

2.1. Clases envoltorio de los tipos de datos primitivos

En Java todo son clases y objetos, excepto los tipos básicos. Esto hace que en algunas circunstancias haya que convertir estos tipos básicos en objetos. Para realizar esta conversión se utilizan los envoltorios, que recubren el tipo básico con una clase y, a partir de ese momento, el tipo básico envuelto se convierte en un objeto.

Tipo	Envoltorio
long	Long
int	Integer
short	Short
byte	Byte
float	Float
double	Double
char	Character
boolean	Boolean

void	Void

Nota: Otra razón muy importante para utilizar envoltorios es poder utilizar las clases de utilidad que Java proporciona en su biblioteca de clases. Estas clases necesitan objetos para funcionar y no aceptan tipos de datos básicos.

Los envoltorios tienen métodos para convertir cadenas de caracteres en tipos básicos.

Ejemplo: Para convertir en un número entero un valor introducido por teclado (lo que se introduce por teclado son cadenas de carácteres), se puede utilizar el método parseInt() de la clase Integer.

```
int num = Integer.parseInt (“12345”);
```

3. Operadores

Los lenguajes de programación poseen diferentes tipos de operadores que son utilizados para construir expresiones por combinación de otras más sencillas En Java, se pueden distinguir los siguientes tipos de operadores:

- Operadores aritméticos.
- Operadores relacionales.
- Operadores lógicos.
- Operadores de asignación.
- Operadores de acceso.

3.1. Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Mutiplicación
/	División
%	Resto de división
-	Cambio de signo
++	Incremento
--	Decremento

3.2. Operadores relacionales

Operador	Operación
==	Igual
!=	Distinto
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual

3.3. Operadores lógicos

Operador	Operación
&	AND
|	OR
^	XOR
&&	AND en cortocircuito
||	OR en cortocircuito
!	NOT

3.4. Operadores a nivel de bit

Operador	Operación
~	NOT
&	AND
|	OR
^	XOR
>>	Desplazamiento a la derecha
>>>	Desplazamiento a la derecha sin signo
<<	Desplazamiento a la izquierda

3.5. Operadores de asignación

Operador	Operación
=	Asignación

3.6. Formas reducidas de los operadores

Operador	Operación
~	NOT
+=	Suma y asignación

-=	Resta y asignación
*=	Producto y asignación
/=	División y asignación
%=	Módulo y asignación
&=	AND y asignación
|=	OR y asignación
^=	XOR y asignación
<<=	Desplazamiento a la izquierda y asignación
>>=	Desplazamiento a la derecha y asignación
>>>=	Desplazamiento a la derecha sin signo y asignación

3.7. Precedencia y asociatividad de los operadores

A la hora de evaluar las expresiones, hay que tener en cuenta la precedencia y asociatividad de los operadores. En la tabla siguiente se muestra la precedencia y asociatividad de los diferentes operadores del lenguaje:

Prioridad	Operador	Asociatividad
1	[ ] ( ) .	Izquierda a derecha
2	++ -- + (unario), - (unario) ( ) (cast) new	Derecha a izquierda
3	*, /,%	Izquierda a derecha
4	+ -	Izquierda a derecha
5	>> >>> <<	Izquierda a derecha
6	> >= >= < instanceof	Izquierda a derecha
7	==, !=	Izquierda a derecha
8	&	Izquierda a derecha
9	^	Izquierda a derecha
10	|	Izquierda a derecha
11	&&	Izquierda a derecha
12	||	Izquierda a derecha
13	¿:	Izquierda a derecha
14	= += -= *= /= %= &= |= ^= <<= >>= >>>=	Izquierda a derecha

Para alterar la precedencia y asociatividad de los operadores se utilizan los paréntesis. En este sentido, los paréntesis son los operadores de mayor precedencia.

4. Comentarios

Un comentario es una secuencia de caracteres encerrada entre los delimitadores "/*" y "*/". Todos los caracteres entre esos delimitadores son ignorados por el compilador. Los comentarios pueden abarcar varias líneas, pero, no se pueden anidar. Se utilizan para documentar los programas.

En Java también existen los comentarios de una línea, que empiezan con los delimitadores // y terminan con el fin de línea y los comentarios de documentación, que se encierran entre los delimitadores "/**" y "*/".

Ejemplos:

/* Este es un comentario de varias líneas. Esta es la primera línea del comentario
Esta es la segunda línea del comentario
Esta es la tercera línea del comentario */

/* Este es un comentario de una línea */

//Este es un comentario de una línea

// Este es un comentario de varias líneas. Esta es la primera línea del comentario
// Esta es la segunda línea del comentario
// Esta es la tercera línea del comentario

```
/* Ejemplo 2b */
/* Ejemplo de programa que calcula el producto de dos números */

public class Producto
{

        public static void main (String args [ ] )
        {
                /* Se declaran las variables multiplicando y multiplicador
                y se les asignan los valores 1000 y 2 respectivamente */
                int multiplicando = 1000 , multiplicador = 2 ;
                /* Se calcula el producto y se muestra por pantalla */
                System.out.println ("Resultado: " + multiplicando * multiplicador);
        }

}
```

Ejemplo de programa en Java

5. Identificadores

Un identificador consiste en una secuencia de caracteres, dígitos o caracteres de subrayado que comienzan por una letra o un carácter de subrayado. Los identificadores se usan para nombrar entidades del programa (constantes, variables, métodos, etc.).

Un identificador ha de cumplir las reglas siguientes:

- Puede estar formado por letras, números y símbolos de subrayado _.
- Ha de empezar por una letra o el símbolo de subrayado _.
- Se distingue entre mayúsculas y minúsculas.

El lenguaje es sensible a las mayúsculas (case sensitive), lo que significa que dos identificadores formados por los mismos caracteres y que difieran únicamente en el uso de mayúsculas o minúsculas se consideran diferentes. Por ejemplo nombre, Nombre y NOMBRE son identificadores diferentes. No existe limitación en cuanto a la longitud de los identificadores.

Nota: Las palabras reservadas son identificadores que tienen un significado especial para el lenguaje y por lo tanto no pueden ser utilizadas para nombrar otras entidades. Entre las palabras reservadas se pueden citar: main, void, return, int, if, else, putw, puts, while, switch, case, break, default, char, struct,etc.

6. Declaración de constantes

Una constante simbólica es la representación nombrada de un dato constante; es decir, un dato cuyo valor va a permanecer inalterado a lo largo de toda la ejecución del programa. Las constantes en Java se definen mediante el modificador **final**.

Las constantes simbólicas se suelen declarar al inicio del programa y para definirlas basta con poner el modificador **final** antes de la declaración del tipo. Al definir un dato como constante, se le puede asignar un valor por primera vez. Una vez inicializada la constante con un valor ya no será posible cambiarlo.

final TipoDeDato nombreConstante = valor;

Para cada constante declarada es necesario utilizar esta cláusula, terminando obligatoriamente en ";". Aunque no es obligatorio, por claridad del código, las constantes se suelen expresar en mayúsculas.

Ejemplos:

```
static final double PI = 3.1416;
static final int DIAS_SEMANA = 7;
private final int MAXIMO = 100;
private final int MINIMO = 0;
final int VALOR; //En este caso todavía no se ha inicializado la constante
```

```
/* Ejemplo */
/* Ejemplo de programa que utiliza constantes */

public class Constantes
{

        static final int HORAS_DIA = 24;

        public static void main (String args [ ] )
        {
                /* Se calculan los minutos que tiene un día y se muestra por pantalla */
                System.out.println ("Un día tiene " + 60 * HORAS_DIA + " minutos.");
        }

}
```

Ejemplo de programa en Java

7. Declaración de variables

Las variables hay que declararlas antes de ser utilizadas para asignarles un tipo y reservar el espacio necesario para almacenarlas en la memoria.

Para declarar variables se utiliza la siguiente sintaxis:

nombre-tipo nombre1 [=valor1], nombre2 [=valor2],…

Donde nombre-tipo es el nombre de un tipo de dato (un tipo primitivo del lenguaje o un tipo de objeto), nombre1, nombre2, ... son los identificadores de las variables declaradas y valor1, valor2,... son los valores asignados a esas variables (la asignación de valores es opcional).

Ejemplos:

```
int sumando1, sumando2, resultado;
int a, b, c;
int a=0, b=0, e=0;
float a, b;
float a = 5.0, b= 3.0;
double num1, num2;
char c;
char c[20];
```

```
/* Ejemplo 2b */
/* Ejemplo de programa que calcula el producto de dos números */

public class Producto
{

        public static void main (String args [ ] )
        {
                /* Se declaran las variables multiplicando y multiplicador
                y se les asignan los valores 1000 y 2 respectivamente */
                int multiplicando = 1000 , multiplicador = 2 ;
                /* Se calcula el producto y se muestra por pantalla */
                System.out.println ("Resultado = " + multiplicando * multiplicador);
        }

}
```

Ejemplo de programa en Java

8. Sentencias

El cuerpo de un programa o subprograma está formado por sentencias. Las sentencias pueden ser de diferentes tipos:

- Expresiones.
- Sentencias de entrada y salida.
- Sentencias de control: sentencias condicionales o de selección.
- Sentencias de control: sentencias repetitivas o bucles.

8.1. Expresiones

Una expresión es una construcción del lenguaje que devuelve un valor de retorno al contexto sintáctico del programa donde se evaluó la expresión. Las expresiones no deben

aparecer de forma aislada en el código. Es decir, han de estar incluidas como parte de una sentencia allá donde se espere una expresión. Las expresiones se pueden clasificar en:

- Expresiones aritméticas.
- Expresiones lógicas.
- Expresiones de asignación.
- Llamadas a funciones.

8.1.1. Expresiones aritméticas

Las expresiones aritméticas son aquellas cuya evaluación devuelve un valor de tipo numérico al contexto del programa donde se evalúan.

Ejemplo:

```
resultado = sumando_1 + sumando_2;
a = b + c * d;
```

8.1.2. Expresiones lógicas

Las expresiones lógicas son aquellas cuya evaluación devuelve un valor lógico (verdadero o falso) al contexto del programa donde se evalúan.

Ejemplos:

```
b = a + 5;
a >= 5;
a && b;
```

8.1.3. Expresiones de asignación

Las expresiones de asignación sirven para asignar un valor a una variable, elemento de una matriz o campo o atributo de un objeto. Para ello se escribe primero una referencia a alguno de estos elementos seguido del operador de asignación "=" y a su derecha una expresión. El compilador deberá comprobar en primer lugar la compatibilidad entre el tipo de la expresión a la derecha del operador de asignación con el de la referencia a la izquierda. La sintaxis de la expresión de asignación es:

```
referencia = expresión
```

Donde referencia es una referencia a una posición de memoria (variable, parámetro, elemento de un vector o atributo de un objeto) y expresión una expresión que le da valor.

Operador	Operación
=	Asignación

Ejemplos:

a=5;
resultado=3 + 4;
resultado=3 * a + 5;
b = 6 >= 5
c = b
e = a % d

```
/* Ejemplo asignaciones */
/* Ejemplo de programa que calcula el producto de dos números */

public class Producto
{

        public static void main (String args [ ] )
        {
                /* Se declaran las variables a, b y c
                A las variables a y b se les asignan los valores 5 y 3 respectivamente */
                int a=5 , b = 3 ;
                /* Se calcula la suma de a + b y se asigna a la variable c*/
                c = a + b ;
                /* se muestra el resultado por pantalla */
                System.out.println ("La suma de " + a + " y " + b + " es igual a " + c);
        }

}
```

Ejemplo de programa en Java

8.2. Sentencias de entrada y salida

Las sentencias de entrada y salida permiten mostrar mensajes por la salida estándar (pantalla) y leer información introducida a través del teclado.

Código java	Ejemplo
System.out.println (mensaje)	System.out.println (“Hola”);
Variable_entera=System.in.read()	numero=System.in.read ();

Para mostrar información por la salida estándar se utilizan los métodos print() y println() del objeto System.out. Para leer información desde la entrada estándar se utiliza el método read() del objeto System.in.

Ejemplos:

```
System.out.println  ("Hola mundo!");
System.out.println  ("lntroduzca el valor deseado: ");
System.out.println  ("EI resultado de sumar " + a + " y " + b +  " es " + resultado);

// Lectura de un byte y asignación a una variable de tipo carácter
char c = System.in.read();

// Lectura de un byte y asignación a una variable de tipo entero
int a = System.in.read();

// Lectura de hasta 10 bytes
byte [] buffer = new byte[10];
System.in.read(buffer);
```

También se puede leer información desde el teclado utilizando la clase Scanner del paquete java.util.

```
// Lectura de una cadena utilizando la clase Scanner
Scanner sc = new Scanner(System.in);
String cadena = sc.nextLine();

// Lectura de tres enteros utilizando la clase Scanner
Scanner sc = new Scanner(System.in);
int a = sc.nextInt();      // Lectura del primer entero
int b = sc.nextInt();     // Lectura del segundo entero
int c = sc.nextInt();     // Lectura del tercer entero
```

Nota: la clase Scanner tiene métodos para leer diferentes tipos de datos y asignarlos a la variable correspondiente (nextInt(), nextFloat(), nextDouble(), etc. También puede leer una línea completa utilizando el método nextLine().

```
/* Ejemplo 5 */

impor java.util.Scanner;

public class Saludar
```

```
{

        public static void main (String args [ ] )
        {
                Scanner teclado;
                String nombre=""; /* se define nombre como una cadena de caracteres */
                teclado=new Scanner (System.in);
                System.out.println ("Como te llamas? ");
                nombre=teclado.nextLine(); /* se asigna a nombre la cadena de caracteres
        introducida por el teclado */
                System.out.println ("Hola " + nombre); /* se muestra el saludo */

        }

}
```

Ejemplo de programa en Java

8.2.1. Secuencias de escape

Como se ha visto, las constantes de cadena, que se encierran entre comillas dobles (por ejemplo "hola"). Las constantes de cadena pueden contener cualquier carácter UNICODE. El carácter de backslash (barra invertida) introduce códigos que representan caracteres de escape, por ejemplo '\n' representa carácter un salto de línea.

Secuencia de escape	Carácter	Significado
\n	NL	Nueva línea
\t	HT	Tabulador horizontal
\v	VT	Tab vertical
\b	BS	Retroceso
\r	CR	Retorno de de carro
\f	FF	Avance de forma
\a	BEL	Señal audible
\\	\	Barra invertida
\?	¿	Interrogación
\’	‘	Comilla simple o Apóstrofo
\”	“	Comilla doble o Comillas
\ooo	Ooo	Número octal
\xhh	hh	Número hexadecimal

8.3. Sentencias de control: sentencias condicionales o de selección.

8.3.1. Sentencia if

Esta sentencia permite alterar el flujo normal de ejecución de un programa en virtud del resultado de la evaluación de una determinada expresión lógica. Sintácticamente esta sentencia puede presentarse de esta forma:

```
if (expresión lógica)
        sentencia o bloque de sentencias 1
[else
        sentencia o bloque de sentencias 2]
```

Los corchetes en este caso indican que la parte else es opcional; es decir, puede aparecer o no. La expresión lógica debe ir siempre entre paréntesis.

Código Java	Ejemplo

if (condición) {
  sentencias
}
	if (edad>=18) {
	  System.out.println (“Soy mayor de edad.”);
	}

if (condición) {
  sentencias
}
else {
  sentencias
}
	if (edad>=18) {
	  System.out.println (“Soy mayor de edad.”);
	}
	else {
	  System.out.println (“Soy menor de edad.”);
	}

Ejemplos:

```
if (a>b)
        a=b;

if (a>b) {
        a=b;
}

if (a>b)
        a=b;
else
        b=a;

if (a>b)
        a=b;
else {
        b=a;
        a=2;
        c=b;
```

```
}
```

Este tipo de construcciones pueden anidarse con otras construcciones de tipo if-else o con otros tipos de sentencias de control de flujo que estudiaremos a continuación.

8.3.2. Otras sentencias condicionales o de selección

Código Java	Ejemplo

if (condición 1) {
  sentencias
}
else if (condición 2) {
  sentencias
}
…
}
else if (condición i) {
  sentencias
}
…
else {
  sentencias
}
	if (a>b) {
	  System.out.println (“a es mayor que b.”);
	}
	else if (a=b) {
	  System.out.println (“a es igual a b.”);
	}
	else {
	  System.out.println (“a es menor que b.”);
	}

Código Java	Ejemplo

switch (expresión) {
  case exp_1:
  sentencias
  case exp_2:
  sentencias
  …
  case exp_i:
  sentencias
  …
  case default:
  sentencias
}
	switch (opcion) {
	  case '1':
	  System.out.println (“Opción 1.”);
	  break;
	  case '2':
	  System.out.println (“Opción 1.”);
	  break;
	  case '3':
	  System.out.println (“Opción 1.”);
	  break;
	  case default:
	  System.out.println (“Otra opción.”);
	}

```
/* Ejemplo 9 */

public class Calculadora
{

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

```
                char c;

                System.out.println ("Introduzca una opcion (S/R/M/D/F): ");
                System.out.print (">> ");
                c = System.in.read(); /* se asigna valor a la variable c */

                while (c != 'F') {

                        switch (c) {
                                case 'S':
                                        suma();
                                        break;
                                case 'R':
                                        resta();
                                        break;
                                case 'M':
                                        mutiplicacion();
                                        break;
                                case 'D':
                                        division();
                                        break;
                                default:
                                        System.out.println ("Opción incorrecta");
                                        break;
                        }

                        System.out.println ("Introduzca una opcion (S/R/M/D/F): ");
                        System.out.print (">> ");
                        c = System.in.read(); /* se asigna valor a la variable c */

                }
        }

}
```

Ejemplo de programa en Java

8.4. Sentencias de control: sentencias repetitivas o bucles

8.4.1. Sentencias while y do-while

Esta sentencia se utiliza para realizar iteraciones sobre un bloque de sentencias alterando así el flujo normal de ejecución del programa. Antes de ejecutar en cada iteración el bloque de sentencias el compilador evalúa una determinada expresión lógica para determinar si debe seguir iterando el bloque o continuar con la siguiente sentencia a la estructura while

mientras se cumpla una determinada condición, determinada por una expresión. Su estructura es:

```
while (expresionLogica)
        sentencia o bloque de sentencias
```

Código Java	Ejemplo

while (condición) {
  sentencias
}
	int i=0;
	while (i<10) {
	  System.out.println (“Me portaré bien.”);
	  i++;
	}

do {
  sentencias
} while (condición);
	int i=0;
	do {
	  System.out.println (“Me portaré bien.”);
	  i++;
	} while (i<10);

8.4.2. Sentencia for

Su estructura es:

```
for (inicialización; condición; incremento)
        sentencia o bloque de sentencias
```

Código Java	Ejemplo

for (exp_1; exp_2; exp_3) {
  sentencias
}
	for (int i=0; i<10; i++) {
	  System.out.println(“Me portaré bien.”);
	}

```
/* Ejemplo 4a */

public class Repetir
{

        public static void main (String args [ ] )
        {
                /* Repetir 10 veces */
                for (int i=0; i<10; i++) {
                        println ("Estaré muy atento en clase.");
                }
        }
```

```
}
```

Ejemplo de programa en Java

9. Funciones (Métodos)

Las funciones (métodos en POO) se utilizan (se declaran) para organizar modularmente el código. Una función es una secuencia de instrucciones encapsuladas bajo un nombre con un conjunto ordenado de parámetros tipificados (opcionalmente ninguno) y un valor de retorno también tipificado. Para llamar a una función, ésta debe haber sido declarada con anterioridad en el programa. La definición se hace indicando el tipo de retorno, el nombre de la función y después, entre paréntesis, los argumentos. Por último se añaden las sentencias. Debe existir una sentencia return que devuelva el valor de retorno de la función.

```
/* Ejemplo de función que devuelve la suma de dos números que se pasan como parámetros*/

int suma (int sumando_1, sumando_2)
{
        int resultado;
        resultado = sumando_1 +  sumando_2;
        return resultado;
}
```

Ejemplo de programa en Java

Ejemplo sencillo del uso de funciones

```
/* Ejemplo 06a */

public class Funciones01
{

        public static void main (String args [ ] ) {

                System.out.println ("Buenos dias.");

                System.out.println ();
                System.out.println ("Un programa puede descomponerse en funciones.");
                System.out.println ("Cada función se encarga de resolver una parte del problema.");
                System.out.println ("Esto permite construir una aplicación de forma modular.");
                System.out.println ();
```

```
                System.out.println ("Adios.");
        }

}
```

Ejemplo de programa en Java

```
/* Ejemplo 06b */

public class Funciones02
{

        prívate void saludo() {
                System.out.println ("Buenos dias.");
        }

        prívate void despedida() {
                System.out.println ("Adios.");
        }

        prívate void mensaje() {
                System.out.println ();
                System.out.println ("Un programa puede descomponerse en funciones.");
                System.out.println ("Cada función se encarga de resolver una parte del
                problema.");
                System.out.println ("Esto permite construir una aplicación de forma
                modular.");
                System.out.println ();
        }

        public static void main (String args [ ] ) {
                saludo(); /* Llamada a la función saludo */
                mensaje(): /* Llamada a la función mensaje */
                despedida(); /* Llamada a la función despedida */
        }

}
```

Ejemplo de programa en Java

9.1. El método main

Todo programa en Java debe obligatoriamente declarar un método principal llamado main. Este método constituye el punto de arranque del programa.

```
/* Ejemplo 7: Programa que calcula la suma de dos números que se introducen por teclado
*/

import java.util.Scanner;

public class Sumar
{

        public static void main (String args [ ]) {

                float a, b, c; /* se declaran tres variables a, b y c como decimales */
                Scanner teclado = new scanner(System.in);

                System.out.println ("Primer sumando: ");
                a=teclado.nextFloat; /* se asigna valor a la variable a */

                System.out.println ("Segundo sumando: ");
                b=teclado.nextFloat; /* se asigna valor a la variable b */

                teclado.close();

                c = a + b;

                /* se muestra el resultado */
                System.out.println ("La suma de " + a + " y " + b + " es " + c);

        }

}
```

Ejemplo de programa en Java

10. Ejemplo: Programa completo para el cálculo de las raíces de una ecuación de segundo grado

```
/* Ejemplo 8: Programa para el cálculo de las raíces de una ecuación de segundo grado */

import java.math.*;

public class EcuacionSegundoGrado
{

        public static void main (String args [ ] ) {

                double a, b, c, x1 , x2, discriminante;
                Scanner teclado = new scanner(System.in);
```

```
            System.out.println ("Valores de los coeficientes a, b y c: ");
            a=teclado.nextDouble; /* se asigna valor al coeficiente a */
            b=teclado.nextDouble; /* se asigna valor al coeficiente b */
            c=teclado.nextDouble; /* se asigna valor al coeficiente c */
            teclado.close();

            if (a==0)
                  System.out.println ("Error: Es una ecuación de primer grado");
            else {
                  discriminante=b*b - 4*a*c;
                  if (discriminante>0) {
                        x1=(-b + sqrt(discriminante)) / (2.0 * a);
                        x2=(-b - sqrt(discriminante)) / (2.0 * a);
                        System.out.println ("Primera raiz: " + x1);
                        System.out.println ("Segunda raiz: " + x2);
                  }
                  else if (discriminante==0) {
                        x1 =(-b) / (2.0 * a);
                        System.out.println ("Raiz doble: " + x1);
                  }
                  else
                        System.out.println ("La ecuacion no tiene soluciones reales.");

            }
      }

}
```

Ejemplo de programa en Java

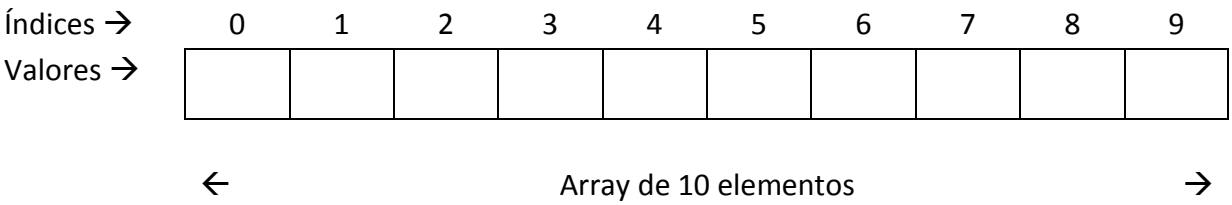
11. Tipos de datos estructurados

Se llama tipo primitivo a los tipos de datos originales de un lenguaje de programación, esto es, aquellos que nos proporciona el lenguaje y con los que se pueden construir tipos de datos abstractos y estructuras de datos. Los tipos de datos primitivos definen como se almacena la información en la memoria del ordenador y los diferentes procesos que se pueden realizar sobre ella. Son abstracciones útiles que facilitan el manejo de los datos, pero en la mayoría de los casos la información que se debe procesar está estructurada de alguna manera. Por eso la mayoría de los lenguajes de programación proporcionan uno o varios mecanismos para combinar los tipos primitivos en estructuras más complejas llamadas tipos de datos estructurados. Los tipos de datos estructurados o tipos compuestos son agrupaciones de otros tipos de datos. Los tipos de datos estructurados más habituales son

los arreglos, las cadenas de caracteres (String) y los archivos o ficheros. En POO, donde aparece el concepto de clase, con sus campos y métodos, los registros no tienen sentido.

11.1. Arrays

Un array (o arreglo) es una estructura de datos que almacena una cantidad fija de elementos del mismo tipo, a los cuales se puede acceder por medio de uno (unidimensional) o varios índices (multidimensional) que indican su posición dentro de la estructura.



Cuando se declara un array se está creando una referencia a un array, pero no se tiene todavía ningún array sino algo que puede apuntar a un array. Por eso, a la hora de declarar un array no es necesario indicar el número de elementos del array.

```
int [ ] miArrayDeEnteros;
int miArrayDeEnteros [ ];
char [ ] miArrayDeCaracteres;
char miArrayDeCaracteres [ ];
Persona [ ] personas;
Persona personas [ ];
```

Cuando se crea un array se le asigna un espacio de almacenamiento en memoria. Esto se hace con el operador new. En este momento si que hay que indicar el número de elementos del array, ya que el espacio que habrá que reservar en memoria será igual al número máximo de elementos del array por el tamaño del tipo de dato que almacena.

```
int miArrayDeEnteros [ ];
miArrayDeEnteros = new int [10];

char [ ] miArrayDeCaracteres;
miArrayDeCaracteres = new char [30];

Persona personas;
```

```
Personas = new Persona [5];
```

Una vez creado el array (se ha reservado en memoria el espacio necesario para el array), lo siguiente sería inicializar el array.

Para inicializar (por ejemplo a 0) los elementos de un array:

```
int [ ] miArrayDeEnteros = new int [10];
for (int i = 0; i < miArrayDeEnteros.length; i++) {
        miArrayDeEnteros [i] = 0;
}
```

Nota: el atributo length almacena el número de elementos del array. A la hora de trabajar con arrays hay que tener en cuenta que el subíndice no puede ser inferior a 0 ni superior a al valor (length – 1).

La declaración, creación e inicialización de un array son operaciones distintas que, como hemos visto pueden realizarse de manera independiente. No obstante, la creación de un array puede hacerse en el momento de la declaración:

```
int [ ] miArrayDeEnteros = new int [10];
char [ ] miArrayDeCaracteres = new char [30];
Persona personas = new Persona [5];
```

También se puede crear e inicializar el array en el momento de la declaración. Para ello se añade un signo igual y una lista de valores encerrados entre llaves y separados por comas.

```
int [ ] miArrayDeEnteros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

El acceso a los diferentes elementos del array se hace mediante su nombre (o identificador) seguido de la posición (o índice) del elemento entre corchetes.

```
miArrayDeEnteros [0]=15;
miArrayDeEnteros [1]=2;
miArrayDeEnteros [2]=5;

int a;
a = miArrayDeEnteros [0];
int b = miArrayDeEnteros [1];
```

Los elementos de un array se comportan como cualquier variable de su tipo y pueden usarse en expresiones.

```
int suma = miArrayDeEnteros [3] + miArrayDeEnteros [5];
```

```
/* Ejemplo de uso de arrays */

public class Arrays
{

        public static void main (String args [ ] )
        {
                int [] miArray = new int [10];
                int i;
                /* Inicializar el array */
                for (i=0; i<10; i++) {
                        miArray [i] = i;
                }

                /* Imprimir el array */
                println ("Los valores de mi array son:");
                for (i=10; i>0; i++) {
                        println (miArray [i-1]);
                }
        }

}
```

Ejemplo de programa en Java

11.2. Arrays multidimensionales

Los arrays multidimensionales son arrays que necesitan más de un subíndice para identificar los elementos.

```
int [ ] [ ] miArrayDeEnteros;
int miArrayDeEnteros [ ] [ ];

int [ ] [ ]miArrayDeEnteros;
miArrayDeEnteros = new int [5][10];

int [ ] [ ]miArrayDeEnteros = new int [5][10];
```

El acceso a los diferentes elementos del array se hace mediante su nombre (o identificador) seguido de los índices que indican la posición del elemento entre corchetes.

```
miArrayDeEnteros [0][0]=15;
miArrayDeEnteros [0][1]=2;
miArrayDeEnteros [2][5]=5;

int a;
a = miArrayDeEnteros [0][0];
int b = miArrayDeEnteros [0][1];

int suma = miArrayDeEnteros [3][4] + miArrayDeEnteros [3][5];
```

Para inicializar (por ejemplo a 0) los elementos de un array multidimensional:

```
int [ ] miArrayDeEnteros = new int [5][10];
for (int i = 0; i < miArrayDeEnteros.length; i++) {
        for (int j = 0; j < miArrayDeEnteros[i].length; j++) {
                miArrayDeEnteros [i][j] = 0;
        }
}
```

11.3. Cadenas de caracteres (Strings)

En programación, una cadena de caracteres (String) es una secuencia ordenada (de longitud arbitraria, aunque finita) de elementos que pertenecen a un cierto lenguaje formal o alfabeto análogas a una fórmula o a una oración.

En general, una cadena de caracteres es una sucesión de caracteres (letras, números u otros signos o símbolos).

Algunas de las operaciones comunes con cadenas son las siguientes:

- **Asignación**: Consiste en asignar una cadena a otra.
- **Concatenación**: Consiste en unir dos cadenas o más (o una cadena con un carácter) para formar una cadena de mayor tamaño.
- **Búsqueda**: Consiste en localizar dentro de una cadena una subcadena más pequeña o un carácter.
- **Extracción**: Se trata de sacar fuera de una cadena una porción de la misma según su posición dentro de ella.

- **Comparación**: Se utiliza para comparar dos cadenas.
- **Longitud**: Número de caracteres de una cadena

Nota: Las cadenas de caracteres se escriben entre comillas dobles ("esto es una cadena de caracteres"), mientras que un carácter de esa cadena (char) se escribe entre comillas simples ('p'). Generalmente para acceder a un carácter en una posición determinada se suele usar la forma variable [posición] como cuando se accede a un vector.

```
/* Ejemplo cadenas de caracteres */
/* Esté programa pide al usuario que introduzca por teclado dos cadenas de caracteres y muestra por pantalla la cadena resultante de unir las dos cadenas introducidas */

import java.util.Scanner;

public class Cadenas
{

	public static void main (String args [ ] )
	{
		Scanner teclado;
		String cadena_1=""; /* se define cadena_1 como una cadena de caracteres */
		String cadena_2=""; /* se define cadena_2 como una cadena de caracteres */
		String resultado=""; /* se define resultado como una cadena de caracteres */
		teclado=new Scanner (System.in);
		System.out.println ("Introduce una palabra? ");
		cadena_1=teclado.nextLine(); /* se asigna a cadena_1 la cadena de caracteres introducida por el teclado */
		System.out.println ("Introduce otra palabra: ");
		cadena_2=teclado.nextLine(); /* se asigna a cadena_2 la cadena de caracteres introducida por el teclado */
		resultado = cadena_1 + cadena_2;
		System.out.println (resultado); /* se muestra la cadena formada por la unión de las dos cadenas */

	}

}
```

Ejemplo de programa en Java

12. Ficheros

Un fichero es un conjunto de información relacionada, grabada en el sistema de almacenamiento secundario y a la que se hace referencia mediante un nombre. Los ficheros permiten almacenar datos en memoria secundaria para utilizarlos en el futuro.

En Java existen dos tipos de ficheros: ficheros binarios (formados por secuencias de bytes) y ficheros de texto (formados por una secuencia de caracteres subdivida en registros de longitud variable llamados líneas).

Las operaciones típicas sobre ficheros son crear, borrar, abrir, cerrar, leer y escribir. Generalmente, al trabajar con ficheros, será necesario realizar un subconjunto de las operaciones anteriores. En particular, siempre que se trabaje con un fichero se deberían realizar las siguientes operaciones:

1. **Crear**, o asignar, un nombre lógico al fichero físico.
2. **Abrir** el fichero.
3. **Operar** sobre el fichero (lectura/escritura, inserción/borrado, etc.).
4. **Cerrar** el fichero.

Nota: Un fichero puede abrirse de diferentes maneras. La apertura del fichero en modo lectura, escritura, lectura/escritura dependerá del tipo de operaciones que vayan a realizarse con los datos almacenados en el fichero.

El API de Java incluye el paquete java.io, que contiene numerosas clases para implementar operaciones de E/S independientes de la plataforma en que se realicen. Para utilizar las clases relacionadas con la entrada y salida de datos es necesario incluir la librería Input-Output de Java mediante la instrucción:

```
import java.io.*;
```

Las clases del paquete java.io se ubican dentro de dos categorías principales: aquellas que operan con archivos de texto (lectores y escritores) y las que operan con archivos binarios (manejadores de flujo).

Nota: En Java, para leer ficheros de texto, también puede utilizarse la clase Scanner del paquete java.util.

13. POO. Clases, objetos, campos y métodos

En la POO, un programa es un conjunto de objetos de distintas clases enviándose mensajes y comunicándose entre sí.

Una clase es una abstracción de un concepto. Una clase es una plantilla donde se definen las propiedades y comportamientos que tienen en común un conjunto de elementos.

Un objeto es un ejemplar concreto de una clase. Un objeto tiene una estructura (campos) con unos valores (estado) y un comportamiento (método).

Clases

Una clase es una abstracción de un concepto. Una clase es una plantilla donde se definen las propiedades y comportamientos que tienen en común un conjunto de elementos.

Componentes de una clase

Las clases están formadas por los siguientes elementos:

- **Campos**: atributos de la clase.
- **Constructores**: modo de construir los objetos de las clases.
- **Métodos**: comportamiento de las clases.
- **Otras clases (clases internas) dentro de la clase**: (No se contemplan en este manual)

Todas las clases pertenecen a un paquete. Un paquete es un conjunto de clases bajo un mismo nombre.

Estructura de una clase

La estructura general de una clase en Java es la siguiente:

```
Public class Nombre_de_la_clase
{
        //Campos o atributos de la clase
        …

        // Constructores
        …

        // Métodos
        …
}
```

Public class Persona

```
{

        //Campos o atributos de la clase
        private String nombre;
        private String primerApellido;
        private String segundoApellido;
        private long dni;
        private int edad;

        // Constructor por defecto
        public Persona () {

        }

        // Otros constructores
        public Persona (String n, String a1, String a2) {
                nombre=n;
                primerApellido=a1;
                segundoApellido=a2;
                dni=0;
                edad=0;
        }

        public Persona (String n, String a1, String a2, long d, int e) {
                this (n, a1, a2);
                dni=d;
                edad=e;
        }

}
```

Ejemplo de clase en Java
(Sólo se muestran los campos y los constructores)

Objetos

Un objeto es un ejemplar concreto de una clase. Un objeto tiene una estructura (campos) con unos valores (estado) y un comportamiento (método).

Para crear un objeto hay que llamar a alguno de los constructores de la clase.

Ejemplo: para crear un objeto de la clase Persona, se llama al constructor de la clase:

Persona persona = new Persona (“Pepe”, “Mellamo” “Nosecomomellamo”, 12345678, 18);

Cuando se crea y manipula un objeto, el valor de los campos determina su estado y los métodos su comportamiento.

Campos

Las variables de una clase de denominan campos. Los campos son los atributos de un objeto. Todos los objetos de una clase tienen los mismos campos pero con diferentes valores. Cada objeto tendrá un valor concreto de sus atributos que lo diferenciarán de los demás.

Importante: A veces, sin embargo, se desea que un campo sea compartido por todos los objetos de una clase. Esto se consigue declarando los campos como **static**, por lo que se denominan campos estáticos o de clase.

En Java, los campos y los métodos pueden ser públicos (se puede acceder a ellos desde cualquier clase), protegidos (se puede acceder a ellos desde las clases del paquete al que pertenece la clase y desde sus clases derivadas) y privados (sólo son accesibles desde los métodos de la propia clase). Lo recomendable es declarar los campos de una clase como privados, de tal manera que sólo sean accesibles a través de los métodos de la clase, los cuáles se declararán como públicos.

Métodos

Un método es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que se puede invocar mediante un nombre. Los métodos de una clase contienen habitualmente código que permite manipular el estado de un objeto; es decir, sus campos. Los métodos determinan el comportamiento de una clase.

Los métodos se invocan como operaciones sobre los objetos a través de sus referencias usando el operador “.”:

referencia.método (parámetros);

Los métodos pueden tener cualquier número de parámetros. Cada parámetro tiene un tipo de dato, y los métodos devuelven un valor de un tipo dado.

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.

La utilización de los métodos permite construir programas modulares y la reutilización de código.

Nota: Todo programa java tiene un método llamado **main**. Este método es el punto de entrada al programa y también el punto de salida.

Ejemplos: Algunos ejemplos de métodos para la clase Persona pueden ser los siguientes:

```
public String getNombre () {
        return nombre;
}

public void setNombre (String n) {
        nombre = n;
}

public String getNombreCompleto () {
        return nombre + " " + primerApellido + " " + segundoApellido;
}
```

La invocación de los métodos de una clase se hace a través de los objetos de la misma utilizando el operador punto (“.”):

```
Persona persona = new Persona (“Pepe”, “Mellamo” “Nosecomomellamo”, 12345678, 18);
persona.getNombreCompleto();
```

Al igual que los constructores, los métodos se pueden sobrecargar, esto es, pueden aparecer varios métodos con igual nombre pero distinta lista de parámetros.

```
public Persona (String n,)
public Persona (String n, String a1, String a2)
public Persona (String n, String a1, String a2, long d, int e)
```

Ejemplo: Código completo de la clase Persona

```
/* Ejemplo 1 */

public class Persona
{

	//Campos o atributos de la clase
	private String nombre;
	private String primerApellido;
	private String segundoApellido;
	private long dni;
	private int edad;

	// Constructor por defecto
	public Persona () {

	}

	// Otros constructores
	public Persona (String n, String a1, String a2) {
		nombre=n;
		primerApellido=a1;
		segundoApellido=a2;
		dni=0;
		edad=0;
	}

	public Persona (String n, String a1, String a2, long d, int e) {
		this (n, a1, a2);
		dni=d;
		edad=e;
	}

	// Métodos de la clase
	public String getNombre () {
		return nombre;
	}

	public void setNombre (String n) {
		nombre = n;
	}
```

```
public String getPrimerApellido () {
        return primerApellido;
}

public void setPrimerApellido (String a) {
        primerApellido = a;
}

public String getSegundoApellido () {
        return segundoApellido;
}

public void setSegundoApellido (String a) {
        segundoApellido = a;
}

public long getDni () {
        return dni;
}

public void setDni (long d) {
        dni = d;
}

public int getEdad () {
        return edad;
}

public void setEdad (int e) {
        edad = e;
}

public String getNombreCompleto () {
        return nombre + " " + primerApellido + " " + segundoApellido;
}

// Programa principal. Podría estar implementado en otra clase.
public static void main (String args [ ] )
{
        Persona persona = new Persona ("Pepe", "Mellamo" "Nosecomomellamo",
        12345678, 18);
```

```
                System.out.println (“Hola, me llamo “ +  persona.getNombreCompleto());
        }

}
```

Ejemplo de clase en Java

14. Actividades

Actividad 1

El alumno creará una aplicación que muestre por pantalla los datos personales del alumno (nombre y apellidos, lugar donde vive, centro donde estudia, etc.) y un saludo.

Actividad 2

El alumno creará una aplicación que calcule el máximo de dos números que el programa pedirá al usuario. El funcionamiento de la aplicación será el siguiente:

1. Aparecerá un mensaje de bienvenida.
2. Pedirá el primer número.
3. Pedirá el segundo número.
4. Calculará el máximo de los dos números.
5. Devolverá el resultado del cálculo con un mensaje de despedida.

Importante: Se valorará el aspecto estético de la aplicación y la interacción con el usuario.

Actividad 3

El alumno creará una aplicación que simule una calculadora. El programa calculará la suma, la diferencia, el producto y la división de dos números que el programa pedirá al usuario. El funcionamiento de la aplicación será el siguiente:

1. Aparecerá un mensaje de bienvenida.
2. Aparecerá un menú de opciones (suma, diferencia, producto, división y salir).
3. Si se selecciona una operación:
   3.1. Pedirá el primer número.
   3.2. Pedirá el segundo número.
   3.3. Calculará el resultado de la operación.
   3.4. Devolverá el resultado de la operación y pedirá al usuario si volver al menú o salir de la aplicación.

3.4.1. Si se selecciona volver al menú, se limpiará la pantalla y volverá a aparecer el menú de la aplicación.

3.4.2. Si se selecciona la opción Salir, el programa se cerrará con un mensaje de despedida.

4. Si se selecciona la opción Salir:
   4.1. El programa se cerrará con un mensaje de despedida.

Importante: Se valorará el aspecto estético de la aplicación y la interacción con el usuario.