1.- Fundamentos de programación 1

Contenido

1.- Estructura elemental de un programa

El programa más sencillo en Java está formado por una ***clase.*** La clase contiene un único ***método.***

El nombre de la clase debe coincidir con el nombre del programa.

El método ha de llamarse ***main***. Contiene, entre llaves, la secuencia de instrucciones a ejecutar.

```
class NombrePrograma {
    public static void main (String args[]) {
        ...
        // Instrucciones del programa
        ...
    }
}
```

Ejemplo:

Un programa cuyo efecto sea mostrar una frase de saludo en pantalla, por ejemplo *"Hola mundo"*, sería el siguiente:

```
class Hola {
    public static void main (String args[]){
        System.out.println("Hola mundo");
    }
}
```

El bloque principal de este programa consta de una única instrucción que escribe en pantalla la secuencia de caracteres comprendidos entre las dobles comillas.

La clase Hola tiene que guardarse en el archivo Hola.java. Ya hemos comentado que el nombre de la clase ha de coincidir con el nombre del programa.

2.- Entrada y salida (E/S) elemental.

Generalmente, los programas procesan datos de entrada y generan, a partir de éstos, una serie de resultados. Para ello necesitan recibir información del usuario y comunicarle, de alguna forma, los resultados.

Entrada: Leer los datos que proporciona el usuario del programa o que están presentes en algún archivo o fuente de datos.

Salida: Mostrar o proporcionar información al usuario del programa a través de algún dispositivo: pantalla, impresora, archivos, …

Inicialmente realizaremos la entrada desde el ***teclado*** del ordenador y la salida, hacia la ***pantalla***, aunque más adelante estudiaremos cómo obtener y enviar información a ficheros.

Salida por pantalla

Para mostrar información por pantalla utilizamos una de estas dos instrucciones:

- System.out.println (…. Información a mostrar ….);
- System.out.print (…. Información a mostrar ….);

La clase System.out representa a la salida estándar en Java. La salida estándar es por defecto la pantalla, aunque esta asignación por defecto se puede cambiar.

Se usan estos dos métodos para mostrar cualquier tipo de información: números, caracteres sueltos textos, …

El primero de ellos (println) realiza un salto de línea (ln) tras mostrar la información. El segundo no salta de línea.

Entrada desde teclado.

Para leer la información que proporciona el usuario usamos la ***clase Scanner*** del ***paquete java.util***.

Dependiendo del tipo de información que tengamos que leer tendremos que utilizar un método u otro. Así, por ejemplo, si tenemos que leer un número entero, usaremos el método **nextInt( ),** pero si tenemos que leer un número real, usaremos **nextDouble( )**

En el siguiente código podemos ver cuales son algunos de los métodos que tiene la clase Scanner para leer información:

```
import java.util.*;
class NombrePrograma {
   public static void main (String args[]){
        Scanner tec = new Scanner(System.in);
```

```
        ...
        // Para leer números enteros
        byte b = tec.nextByte();
        int  i = tec.nextInt();
        short s = tec.nextShort();
        long l = tec.nextLong();

        // Para leer números reales
        float f = tec.nextFloat();
        double d = tec.nextDouble();

        // Para leer cadenas de caracteres
        String s1 = tec.next();
        String s2 = tec.nextLine();

        //Para leer valores lógicos
        boolean lg = nextBoolean();
    }
}
```

3.- Variables y asignación.

El ordenador está formado por una serie de elementos físicos. Dos de ellos son la ***memoria RAM*** y el ***procesador (CPU).*** La CPU es la encargada de ejecutar las instrucciones de los programas. Las instrucciones realizan operaciones con datos que se encuentran en la memoria RAM.

Los datos que manejan los programas se almacenan en una o varias posiciones consecutivas de la memoria. En los lenguajes de programación, estas posiciones de memoria se manipulan mediante ***variables***.

Podríamos decir que una variable es una porción de la memoria (la porción que ocupa un dato) al que damos un nombre (identificador) para referirnos a él.

Declaración, inicialización y asignación.

En Java, para poder utilizar una variable y almacenar algún dato en ella, es necesario ***declarar*** la variable.

Declarar una variable consiste en indicar su nombre (identificador) y su tipo antes de utilizarla. El identificador lo usaremos para referirnos a la variable,

mientras que el tipo determinará la información que la variable podrá almacenar y las operaciones que se podrán hacer con ellas.

Sintáxis para declarar una variable:

Declarar una variable:

tipo identificador ;

Declarar varias variables del mismo tipo:

tipo iden1, iden2, ident3

Ejemplos

```
// declara una variable de tipo
// int llamada edad
int edad;

// declara dos variables de
// tipo double llamadas estatura y peso
double estatura, peso;
```

Inicializar variables

A las variables se les puede dar un valor inicial cuando se declaran.

tipo identificador = valor;

Ejemplos

```
int edad = 35;
double estatura, peso = 80.5;
```

La asignación

La asignación se utiliza para dar valores a las variables o para reemplazar los valores que ya tienen por otros nuevos.

identificador = expresión ;

A la izquierda de la asignación deberá aparecer un identificador de variable, mientras que a la derecha pondremos cualquier expresión cuyo resultado sea del mismo tipo que la variable (o de algún tipo compatible):

Ejemplos

```
int edad = 35; //Declaración e inicialización
    edad = 40; //Asignación (cambia el valor de edad)
```

No se puede usar una variable de tipo primitivo (ya veremos cuáles son estos tipos) si antes no se le ha dado valor.

Ejemplos

```
int edad;
System.out.println("Años: " + edad); //Incorrecto

int edad = 30;
System.out.println("Años: " + edad); //Correcto
```

4.- Tipos de datos: int y double.

En Java existen ***8 tipos de datos primitivos*** (ver anexo sobre tipos de datos). Dos de estos tipos son int y doublé

- Las variables de tipo ***int*** almacenan valores enteros en determinado rango
- Las variables de tipo ***double*** almacenan valores reales en determinado rango y con determinada precisión.

Cada variable de tipo int ocupa 4 bytes de memoria. Cada variable de tipo double ocupa 8 bytes de memoria. Para que los programas no requieran más memoria de la necesaria, hay que declarar las variables con el tipo adecuado. Tendremos que tener en cuenta para ello si el dato que queremos almacenar puede tener o no decimales.

Operadores aritméticos

Son operadores que operan con datos numéricos, ya sean enteros o reales.

Binarios (dos operandos)	+ - * / %	Suma Resta Producto División Módulo (resto de la div.)
Unarios (un operando)	-	Cambio de signo

División entera vs división real

El ***operador /*** se usa, en realidad para hacer dos operaciones distintas: la división entera y la división real. Se hará una u otra dependiendo de cuál sea el tipo de sus operandos

Operación	Resultado	¿Cuando se hace?	Ejemplos

División entera	entero	Cuando dividendo y divisor son números enteros	int a = 9, b = 2, c;
			c = a / b; // Resultado 4
			c = a / 4 ; // Resultado 2
			double d = a / b; // Resultado 4

División real	real	Cuando dividendo o divisor (o ambos) son números reales	int a = 9;
			double b = 2 .0, c;
			c = a / b; // Resultado 4.5
			c = a / 4.0; // Resultado 2.25

Resto de la división entera (%).

Esta operación obtiene el resto que queda si realizamos una división entera (sin decimales en el cociente)

5 % 2 es igual a 1

8 % 4 es igual a 0

185 % 3 es igual a 2

Es una operación a la que no estamos habituados, pero que resulta muy útil.

Por ejemplo:

- ¿Cuantos minutos y segundos hay en 528 segundos?

  528 / 60 = 8 minutos y

  528 % 60 = 48 segundos

- ¿Como podemos averiguar si un número es par o impar?

  Comprobando si numero % 2 es cero o distinto de cero

Los identificadores son los nombres que se les da a las variables (y a otros elementos:, clases, interfaces, atributos y métodos)

5.- Estructuras de control: La sentencia if

En programación estructurada los programas están construidos a partir de tres tipos de estructuras:

- La ***secuencia***, que permite ejecutar sentencias una tras otra.
- Las estructuras ***alternativa***, que posibilita que los programas realicen bifurcaciones, es decir, que ejecuten un conjunto de instrucciones u otro dependiendo de si se da o no alguna condición.
- Las estructuras ***repetitivas***, que hace que una instrucción o un conjunto de instrucciones se repintan un número de veces o mientras se da alguna condición.

Es habitual que los lenguajes de programación dispongan de varias sentencias alternativas y varias sentencias repetitivas.

En java tenemos las siguientes:

- Sentencias alternativas: ***if*** y ***switch***
- Sentencias repetitivas: ***while***, ***do-while*** y ***for***
- Hasta ahora hemos trabajado únicamente con secuencias de instrucciones. En este apartado del tema vamos a introducir el uso de ***la sentencia if***

Su sintaxis es la siguiente:

```
if ( condición ) {
        ....
        Instrucciones a ejecutar si se cumple la condición
        ...
} else {
        ....
        Instrucciones a ejecutar si no se  cumple la condición
        ...
}
```

Ejemplo

```
…
int edad;
System.out.println("Introduce tu edad");
edad = tec.nextInt();
if(edad >= 18) {
    System.out.println("Tienes " + edad + " años");
    System.out.println("Eres mayor de edad");
} else {
    System.out.println("Tienes " + edad + " años");
    System.out.println("Eres menor de edad");
}
```

Se pueden omitir las llaves correspondientes cuando solo hay una instrucción asociada al if o al else.

Ejemplo

```
…
int edad;
System.out.println("Introduce tu edad");
edad = tec.nextInt();
System.out.println("Tienes " + edad + " años");
if(edad >= 18)
    System.out.println("Eres mayor de edad");
else
    System.out.println("Eres menor de edad");
```

Se pueden anidar unas sentencias dentro de otras.

Ejemplo

```
if(estatura > 2) {
    System.out.println("Muy alto");
} else {
    if(estatura > 1.70) {
        System.out.println("Alto");
    } else {
        if( estatura > 1.60) {
            System.out.println("Estatura media");
        } else {
            System.out.println("Bajo");
        }
    }
}
```

Podríamos omitir las llaves porque una sentencia if-else se considera una sola instrucción.

Ejemplo

```
if(estatura > 2)
    System.out.println("Muy alto");
else
    if(estatura > 1.70)
        System.out.println("Alto");
    else
        if( estatura > 1.60)
            System.out.println("Estatura media");
        else
            System.out.println("Bajo");
```

O incluso hacer que varios if-else compartan línea

Ejemplo

```
if(estatura > 2)
    System.out.println("Muy alto");
else if(estatura > 1.70)
    System.out.println("Alto");
else if( estatura > 1.60)
    System.out.println("Estatura media");
else
    System.out.println("Bajo");
```

Operadores relacionales

Para escribir las condiciones se utilizan ***operadores relacionales***. Los operaciones relacionales permiten comparar dos expresiones del mismo tipo y comprobar si son iguales, distintas, mayor una que la otra, etc

Operador	Nombre de la operación	Ejemplo
< > <= >= == !=	menor mayor menor o igual mayor o igual igual distinto	temperatura < 100 minuto > 6 temperatura <= 35.6f (a+b) >= 7 inicialNombre == 'J' valor1 != (valor2 – valor3)

6.- Anexos

Comentarios

Parte de la documentación de un programa se realiza dentro del propio programa utilizando comentarios. El compilador ignora los comentarios, de manera que estos no pasan a formar parte del programa traducido al lenguaje de la máquina.

En java hay dos tipos de comentarios:

- Comentarios de una sola línea: Van desde la marca **//** hasta el final de la línea.
- Comentarios que pueden ocupar varias líneas: Van precedidos de una marca de apertura **/*** y una de cierre ***/**

```
Ejemplo:
/* Este es un sencillo programa
 que muestra por pantalla un
 saludo al usuario */
class Hola {
    public static void main (String args[]){
        // Se muestra el saludo
        System.out.println("Hola a todos"); //Saludo
    }
}
```

Reglas para construir identificadores

Se utilizan ***identificadores*** para dar nombre a distintos elementos que aparecen en los programas: variables, métodos, clases, constantes, etc.

¿Qué reglas hay que tener en cuenta a la hora de poner nombre (identificador) a estos elementos?

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. Sin embargo, no es recomendable utilizar caracteres especiales como acentos, diéresis, circunflejos, etc

añoDeCreación, raïm, ...

3.- El primer carácter no puede ser numérico y no pueden utilizarse espacios en blanco ni símbolos coincidentes con operadores de Java (*,+,-,&,...)

4.- No puede ser una palabra reservada del lenguaje

5.- La longitud máxima de los identificadores es prácticamen-te ilimitada.

6.- Por convenio (no es obligado, pero facilitan la legibilidad):

- los nombres de las variables y los métodos deberían empezar por una letra minúscula y los de las clases por mayúscula.
- 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

Ejemplos válidos	**Ejemplos no válidos**.
añoDeNacimiento2	3valores
_otra_variable	Dia+mes
Dividendo	fecha nacimiento
edad	int

Palabras reservadas

Las palabras reservadas tienen un significado fijo en el lenguaje de programación y no se pueden usar como identificadores. En la tabla siguiente se muestran las de Java. A lo largo del curso se utilizaran muchas de ellas.

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

Símbolos y caracteres especiales

En Java se usan también algunos caracteres especiales que hacen la función de separadores

Paréntesis	**( )**	Contiene listas de parámetros en la definición y llamada a un método.
Llaves	**{ }**	Sirven para englobar bloques de código y para valores iniciales de arrays
Corchetes	**[ ]**	Sirven para la declaración de arrays, y para hacer referencia a sus elementos
Coma	**,**	Separador de identificadores del mismo tipo en una declaración de variables, separación de argumentos en definición y llamada a métodos.
Punto y coma	**;**	Símbolo terminador de algunas instrucciones.
Punto	**.**	Separador de nombres de paquetes, de subpaquetes y de clases y para referenciar elementos de un objeto.

Tipos de datos elementales

En Java existen ***8 tipos de datos elementales*** (o básicos, o primitivos):

El tipo de una variable determina qué información puede contener y que operaciones se pueden realizar con ella.

Numéricos	Enteros	**byte**
		short
		int
		long
	Reales	**float**
		double
Carácter		**char**
Lógicos		**boolean**

Tipos numéricos y rango de valores

Tipo	Nº bytes	Rango de valores	
		Desde	Hasta
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	$+/- 1,7*10^{-308}$	$+/- 1,7*10^{308}$