MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Elementos de un programa informático

2

ÍNDICE

OBJETIVOS

Conocer qué es un bloque en programación.

Distinguir los diferentes tipos de datos básicos.

Saber cuándo utilizar cada tipo de datos.

Conocer y aplicar los diferentes operadores.

Conocer y aplicar expresiones.

Diferenciar correctamente entre identificador, variable y constante.

Aplicar casting sobre diferentes tipos de datos.

Conocer y aplicar comentarios en código.

/ 1. Introducción y contextualización práctica

En este tema, hablaremos sobre tipos de datos, variables y constantes y cómo les afectan los bloques de código. Conceptos que son básicos a la hora de empezar a programar.

También, vamos a ver las posibles operaciones que se van a poder hacer con cada tipo de dato, ya que no serán las mismas para unos que para otros, pudiendo utilizar conversiones (casting) entre tipos para facilitarnos las cosas.

Por último, hablaremos sobre los comentarios de código, qué son, y qué utilidad tienen.

Todos estos conceptos los iremos repasando en todos los temas restantes, así que no te preocupes, que tenemos mucho tiempo por delante.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

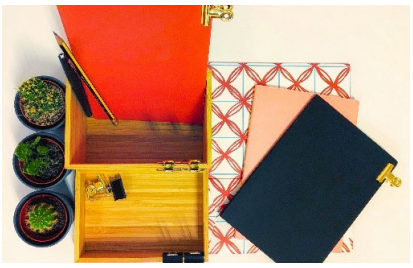

Fig. 1. Diferentes elementos.

Audio Intro. "Las variables".
https://bit.ly/30jd5ev

/ 2. Estructura y bloques fundamentales

Cuando vamos a realizar un programa, indistintamente del lenguaje de programación que vayamos a utilizar, hemos de seguir una estructura que viene ya definida por dicho lenguaje de programación. Es importante destacar que según el lenguaje de programación que utilicemos esa estructura básica del programa podrá variar, por lo que, no es la misma estructura la que utiliza un programa en PASCAL, que uno en C, en Java o en Python.

Las estructuras vienen definidas por lo que llamamos bloques. Un bloque nos va a indicar dónde podemos escribir código respetando la estructura del lenguaje de programación que utilicemos. Los bloques van a estar limitados por una serie de delimitadores, también especificados por el lenguaje de programación, por ejemplo, en Java los limitadores de bloques de código serán las llaves ({ }), mientras que en Python los limitadores de bloque son los saltos de línea y las tabulaciones.

Los lenguajes de programación que utilizan bloques se denominan lenguajes de programación estructurados.

La funcionalidad que nos ofrecen los bloques de código es la de poder tratar todas las instrucciones que lo componen como si se tratase de una única instrucción.

Normalmente, vamos a poder definir bloques dentro de otros bloques, lo que se conoce como anidamiento, permitiéndonos ejecutar instrucciones de una forma mucho más controlada y segura.

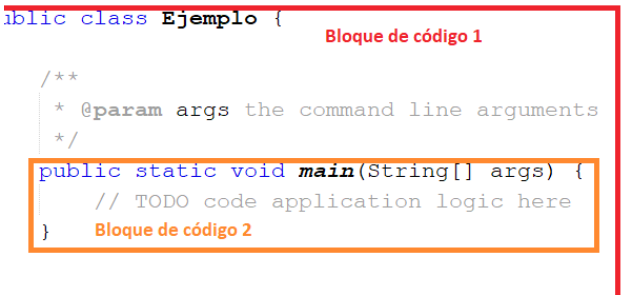

Fig. 2. Bloques de código en Java.

/ 3. Identificadores, variables y constantes

En este apartado, vamos a ver tres conceptos básicos en cualquier lenguaje de programación: los identificadores, las variables y las constantes.

Los **identificadores** son los nombres que sirven para identificar los elementos del programa, como por ejemplo el nombre de una variable, una función, una clase... Estos han de seguir las siguientes restricciones a la hora de ser creados:

1. Solo se podrán utilizar los siguientes caracteres: letras sin contar la ñ (minúsculas o mayúsculas), dígitos del 0 al 9, guion bajo (_) y símbolo del dólar ($). No se admiten espacios en blanco ni acentuaciones.

2. Deberán empezar forzosamente por una letra, mayúscula o minúscula.

3. No se podrán repetir los nombres de los identificadores.

4. No se podrán utilizar palabras reservadas del lenguaje.

5. Los nombres deberán ser significativos, es decir, que den a entender qué representará ese identificador.

Una **variable** es una posición en memoria al que vamos a dar un nombre. Ésta tendrá un valor que podrá cambiar a lo largo del programa. Se escriben según las normas de los identificadores.

Éstas tienen tres partes:

1. El tipo.

2. El nombre.

3. El valor.

Una **constante** es un tipo de variable, pero cuyo valor, una vez asignado, no va a poder cambiar durante la ejecución del programa. Utilizaremos la palabra reservada final para declarar constante cualquier tipo de dato.

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

Fig. 3. Palabras reservadas en Java.

/ 4. Caso práctico 1: "¿Qué tipo de dato elegir para una variable?"

Planteamiento. Pilar y José están realizando su primer programa en Java y necesitan crear una variable para representar la nota media de los exámenes de un alumno. Ambos están convencidos de que tienen que utilizar un dato de tipo numérico, pero dentro de estos hay dos tipos, los enteros y los reales, lo cual hace que no se decidan en cuál de los dos elegir.

Nudo. ¿Qué piensas sobre ello? ¿Crees que es apropiado representar la variable con un tipo de dato entero? ¿Crees que es apropiado representar la variable con un tipo de dato real?

Desenlace. Lo primero que tenemos que hacer siempre que vayamos a realizar un programa es analizar qué variables creemos que vamos a necesitar para codificarlo. Puede ser que más adelante nos demos cuenta de que necesitamos más variables de las que habíamos pensado, es normal, ya que, al principio, no vamos a adivinar todas las variables necesarias.

En este caso, hay que representar una variable que almacenará una nota media. Si lo analizamos detenidamente, nos daremos cuenta que las notas medias van a poder tener decimales, por lo cual, lo más apropiado para representar esta variable sería un tipo de dato numérico real.

Fig. 4. Debemos elegir el contenedor adecuado para la información que vayamos a almacenar.

/ 5. Tipos de datos enteros

Los tipos de datos enteros nos permitirán almacenar números enteros tanto positivos como negativos, es decir, números que no tienen una parte decimal, solo una parte entera. Por ejemplo: -85, -9, 0, 5, 6...

En Java, tenemos los siguientes tipos de datos para representar números enteros.

Tipo	Bytes ocupados en memoria	Rango de valores
byte	1	[-128, 127]
short	2	[-32768, 32767]
int	4	$[-2^{31}, 2^{31}-1]$
long	8	$[-2^{63}, 2^{63}-1]$

Tabla 1. Tipos de datos enteros en Java.

Como vemos, tenemos varios tipos de datos para poder representar a un entero. Por comodidad, siempre vamos a utilizar el dato int, cuyo rango de posibles valores va desde -231 hasta 231-1, pero debemos tener en cuenta que puede quedarse corto si trabajamos con números muy grandes, o desaprovechar la memoria si los valores que vamos a usar son pequeños.

Para poder declarar variables o constantes del tipo entero lo haremos de la siguiente forma:

int nombre_variable;

Podremos asignarles un valor directamente en la declaración.

int nombre_variable = 8;

Si es necesario, podremos declarar más de una variable en la misma línea, separándolas mediante comas.

int nombre_variable1, nombre_variable2 = 9;

```
public class Ejemplo {

    public static void main(String[] args) {
        final int numero_constante = 10;
        int numero_con_valor = 7;
        int numero1 = -3, numero2 = 14, numero3;
    }

}
```

Fig. 5. Declaración de varias variables del tipo entero.

/ 6. Tipos de datos reales

Los tipos de datos reales nos permitirán almacenar números reales tanto positivos como negativos, es decir, números que tienen una parte decimal además de una parte entera. Por ejemplo: -2.3, 1.0, 23.547...

Tipo	Bytes ocupados en memoria	Rango de valores	
		En los negativos	*En los positivos*
float	4	$[-3.4E^{38}, -1.4E^{45}]$	$[1.4E^{-45}, 3.4E^{48}]$
double	8	$[-1.8E^{308}, -4.9E^{324}]$	$[4.9E^{-324}, 1.48E^{308}]$

Tabla 2. Tipos de datos reales en Java.

En Java, tenemos los siguientes tipos de datos para representar números reales.

Como vemos, tenemos varios tipos de datos para poder representar a un real. Por comodidad, siempre vamos a utilizar el dato double, cuyo rango de posibles valores va desde $4.9E^{-324}$ hasta $1.48E^{308}$ en los positivos y de $-1.8E^{308}$ hasta $-4.9E^{324}$ en los negativos.

Para poder declarar variables o constantes del tipo real, lo haremos de la siguiente forma:

double nombre_variable;

Podremos asignarles un valor directamente en la declaración.

double nombre_variable = 3.141592;

Si es necesario, podremos declarar más de una variable en la misma línea, separándolas mediante comas.

double nombre_variable1, nombre_variable2 = 9;

Vídeo 1. "Declaración de variables de tipo real".
https://bit.ly/3cJdTvN

/ 7. Tipos de datos alfanuméricos

Los tipos de datos alfanuméricos **nos permitirán almacenar caracteres, es decir, letras, símbolos especiales y también números**. Por ejemplo: 'a', "hola", "En un lugar de la mancha", "clave123#" ...

En Java, tenemos los siguientes tipos de datos para representar datos alfanuméricos:

- **char**: Este tipo de dato representará un único carácter. Se representarán entre comillas simples.
- **String**: Este tipo de dato representará varios caracteres juntos, es decir, representará lo que se conoce como una cadena de caracteres. Se representarán entre comillas dobles: "Hola, mundo", "Introduzca su nombre:" ...

Como vemos, tenemos dos posibilidades a la hora de representar un dato alfanumérico, la cuestión es, que si únicamente **necesitamos representar un único carácter elegiremos un dato de tipo Char**, pero si necesitamos representar una o varias palabras **elegiremos un dato de tipo String**. Aunque también **podremos representar un carácter como un String, siendo una cadena de un único elemento.**

Cabe destacar que también podemos tener cadenas vacías: "".

Para poder declarar variables o constantes de los tipos *char* o *String* lo haremos de la siguiente forma:

char nombre_variable;

String nombre_variable;

Podremos asignarles un valor directamente en la declaración.

Char nombre_variable = 'H';

String nombre_variable = "Hola que tal".

Audio 1. "Secuencias de escape en los String".
https://bit.ly/3f32rwJ

/ 8. Tipos de datos booleanos

Los tipos de datos booleanos son datos que nos permitirán almacenar únicamente dos posibles valores: verdadero y falso (*TRUE y FALSE*).

En Java, tenemos el siguiente tipo de datos para representar booleanos:

Tipo	Bytes ocupados en memoria	Rango de valores
boolean	8	true / false

Tabla 3. Tipos de datos booleanos en Java

Para poder declarar variables o constantes del tipo booleano lo haremos de la siguiente forma:

boolean nombre_variable;

Podremos asignarles un valor directamente en la declaración.

boolean nombre_variable = true;

boolean nombre_variable = false;

Si es necesario, podremos declarar más de una variable en la misma línea, separándolas mediante comas.

boolean nombre_variable1, nombre_variable2 = false;

```
public class Ejemplo {

    public static void main(String[] args) {
        final boolean booleano_constante = true;
        boolean booleano = false;
        boolean booleano1 = true, booleano2 = false;
    }
```

Fig. 6. Declaración de varias variables del tipo booleano.

/ 9. Expresiones y operadores aritméticos

En programación, las expresiones **son un conjunto de variables, constantes y operadores que nos permitirán manipular los datos**, obteniendo así nueva información como el resultado de aplicar los operadores utilizados en las variables o constantes iniciales.

Los operadores que vamos a poder utilizar **van a depender del tipo de dato con el que queramos operar**, por ejemplo, no podremos hacer las mismas operaciones con un número entero que con una cadena de caracteres.

Los operadores existentes son los siguientes:

- Operadores aritméticos
- Operadores lógicos
- Operadores relacionales
- Operadores de asignación

Los operadores aritméticos son los que **nos van a permitir realizar operaciones aritméticas**. Estos se podrán utilizar con datos de tipo numérico, tanto reales como enteros.

Las operaciones aritméticas que podremos realizar son:

Operación	Símbolo	Descripción
Suma	+	Suma dos números
Resta	-	Resta dos números
Producto	*	Multiplica dos números
División	/	Divide con decimales dos números
Resto de la división	%	Obtiene el resto de una división

Tabla 4. Operaciones aritméticas disponibles en Java.

```java
public static void main(String[] args) {
    int numero1 = 6, numero2 = 3;

    int suma = numero1 + numero2;
    int resta = numero1 - numero2;
    int producto = numero1 * numero2;
    double division = numero1 / numero2;
    int resto = numero1 % numero2;
}
```

Fig. 7. Realización de una suma, resta, producto, división y resto en Java.

9.1. Operadores lógicos

Los operadores lógicos nos **van a permitir realizar operaciones con los datos del tipo boolean**. Estos se pueden aplicar únicamente a estos datos. No tiene sentido aplicarlos a un *int, double o String*.

Estos datos están basados en la lógica matemática y son los siguientes:

- **Operador AND**, representado como &&.
- **Operador OR**, representado como ||.
- **Operador NOT**, representado como !.

Los operadores **AND y OR son operadores binarios**, esto quiere decir, que van a operar sobre dos variables, **mientras que el operador NOT es monario**, esto quiere decir que va a operar sobre una única variable.

Su funcionamiento es el siguiente:

- El operador AND resultará verdadero cuando las dos variables sobre las que se aplica sean verdaderas, en caso contrario devolverá falso.
- El operador OR será verdadero cuando una de las dos variables sobre las que se aplica sea verdadero, en caso contrario devolverá falso.
- El operador NOT devolverá el valor contrario a la variable a la que se aplica.

Para entender mejor el comportamiento de los operadores lógicos, **podemos utilizar una tabla de verdad por cada operación lógica,** la cual nos mostrará los posibles resultados de la misma.

AND	**TRUE**	**FALSE**
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR	**TRUE**	**FALSE**
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

NOT	
TRUE	FALSE
FALSE	TRUE

Tabla 5. Tablas de verdad de los operadores AND, OR y NOT

```
public static void main(String[] args) {
    boolean booleano1 = true, booleano2 = false;

    boolean resultado1 = booleano1 && booleano2;
    boolean resultado2 = booleano1 || booleano2;
    boolean resultado3 = !booleano1;
    boolean resultado4 = !booleano2;
```

Fig. 8. Ejecución de los operadores lógicos en Java.

9.2. Operadores relacionales y de asignación

Los operadores relacionales nos **van a permitir realizar operaciones de comparación entre variables del mismo tipo.** Estos se pueden aplicar a cualquier tipo de variable salvo a los String.

Es importante recordar que solo se podrán comparar variables con otras del mismo tipo.

Los operadores relacionales disponibles en Java son:

- **Operador igual que**, se representa como **==**.
- **Operador distinto,** se representa como **!=**.
- **Operador mayor que**, se representa como **>**.
- **Operador menor que**, se representa como **<**.
- **Operador mayor o igual que**, se representa como **>=**.
- **Operador menor o igual que**, se representa como **<=**.

Estos operadores devolverán verdadero o falso, según si se cumple la condición del operador o no.

Destacar también, que mediante los operadores lógicos podremos concatenar más de un operador de comparación.

Los operadores de asignación, como su propio nombre indica, asignarán un valor a una variable.

Operador	Ejemplo	Equivalencia
=	int a = b;	
+=	int a += b;	int a = a + b;
-=	int a -= b;	int a = a - b;
*=	int a *= b;	int a = a * b;
/=	int a /= b;	int a = a / b;
%=	int a %= b;	int a = a % b;

Tabla 6. Operadores de asignación.

Por último, es conveniente saber que Java proporciona una librería para las operaciones matemáticas que facilita enormemente la implementación de operaciones complejas, como pueden ser raíces cuadradas, potencias, operaciones trigonométricas, etc.

Esta librería se llama Math y es conveniente familiarizarse con ella: https://www.javatpoint.com/java-math

/ 10. Caso práctico 2: "¿Por dónde empezamos?"

Planteamiento: Pilar y José acaban de realizar su primer programa completo en Java, lo han probado y han visto que funciona correctamente, ya que proporciona los resultados que esperaban. Se disponen a empezar con su siguiente programa, pero se dan cuenta de que han nombrado las variables de forma aleatoria tal y como se les ha ido ocurriendo, sin coherencia con lo que representan, aunque no les preocupa demasiado ya que ahora mismo entienden a la perfección su programa.

Nudo: ¿Qué piensas al respecto? ¿Crees que hacen bien en dejar las variables nombradas de esta forma? ¿Crees que se seguirán acordando dentro de una semana del funcionamiento del programa que acaban de hacer?

Desenlace: Cuando en un programa creemos una variable, hemos de intentar nombrarla lo más significativamente posible, es decir, que cuando abramos de nuevo el programa habiendo pasado un tiempo, con el mero hecho de leer el nombre de las variables podamos llegar a saber qué es lo que representan. Por ejemplo, si queremos el cálculo de la edad de una persona, necesitaremos saber en qué año, mes y día nació. Estos tres datos los deberemos almacenar en tres variables y una buena técnica de nombrado sería llamar a la variable que almacena el día de nacimiento *dia_nacimiento*, a la que almacena el mes de nacimiento *mes_nacimiento* y a la que almacena el año de nacimiento *anio_nacimiento*, ya que no podremos utilizar la letra ñ en el nombrado de variables.

Fig.9. Es conveniente usar nombres adecuados para saber la información que contienen las variables.

/ 11. Conversiones de tipo o castings

Como hemos visto anteriormente, solo podremos realizar operaciones entre variables del mismo tipo, pero, si nos paramos a pensar un momento, por ejemplo, vemos que se pueden sumar dos números entre sí: los enteros y los reales, al fin y al cabo, son números, aunque son de tipos distintos. Este es un claro ejemplo de castings entre tipos.

Siempre que los tipos de datos sean compatibles entre si podremos realizar un casting entre ellos, por ejemplo, los *int* y los *double* son compatibles, por lo cual, podremos realizar un casting entre ellos, pero los *double* y los *string* no son compatibles, por lo que, no podremos realizar un casting entre ellos.

Existen dos tipos de castings:

- **Castings implícitos:** Es el caso en el que los datos guardan el mismo tipo de dato, pero no son el mismo tipo de dato. Por ejemplo, convertir un int a long. Este casting se realiza automáticamente.

- **Castings explícitos:** Es el caso en el que los datos no guardan el mismo tipo de dato, pero los tipos son compatibles entre sí. Por ejemplo, convertir un double a int. Este casting hay que ponerlo obligatoriamente.

Para realizar un casting, tendremos que indicar entre paréntesis el tipo al que queremos convertir el dato.

Puede ocurrir que al realizar un casting perdamos información, por ejemplo, si convertimos un *double* a un *int* perdemos la parte decimal del dato original.

```
public static void main(String[] args) {
    double numero_real = 8.3;

    int numero_entero = (int) numero_real;
    // Aquí perderemos información, ya que convertimos
    // un double a int.
    // numero_entero valdrá 8, no 8.3
}
```

Fig. 10. Casting de double a int en Java.

/ 12. Comentarios de código

Los comentarios nos van a servir para explicar qué es lo que hace el código que estamos escribiendo, ya sea para otros programadores o para nosotros mismos pasado un tiempo. Una muy buena práctica a la hora de aprender a programar es la de acostumbrarse a comentar nuestro código, así, cuando volvamos a abrir ese programa que hicimos hace tiempo recordaremos fácilmente qué hacía gracias a los comentarios de código.

El 100% de los lenguajes de programación que existen tienen la posibilidad de escribir comentarios en el código.

En los comentarios podemos escribir lo que queramos, frases, aclaraciones, fórmulas, y utilizar cualquier tipo de carácter, ya que lo primero que hará el compilador al iniciar el proceso de compilación de código es obviarlos, así que no tienen influencia alguna en el código, salvo la de aclararnos qué estamos haciendo.

En Java existes dos tipos de comentarios:

- **Comentarios de una sola línea:** Estos comentarios ocupan una línea desde el lugar en el que los coloquemos, es decir, si escribimos una línea de código y después de ella un comentario, la línea de código no se verá afectada. Se representan con dos barras (//).

- **Comentarios multilínea:** Estos comentarios pueden ocupar varias líneas, y todo lo que esté entre las marcas de inicio y fin será considerado como parte del comentario. La marca de inicio de los comentarios multilínea es una barra y un asterisco (/*) y la marca de fin de línea es un asterisco y una barra (*/).

Podremos colocar tantos comentarios como queramos dentro de un programa

```
public static void main(String[] args) {
    // Aquí declaramos una variable (Comentario de una línea)
    int numero = 7;

    /*
        Ahora vamos a multiplicar el número por 5
        y mostrarlo por pantalla
        (Comentario multilínea)
    */
    numero *= 5;
    System.out.println(numero);
}
```

Fig. 11. Casting de double a int en Java.

Vídeo 2. "Lectura de datos por teclado".
https://bit.ly/30ikVVS

/ 13. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos visto los diferentes tipos de datos que existen en el lenguaje de programación Java y que vamos a utilizar en toda nuestra andanza por el mundo de la programación.

Recuerda que es muy importante tener en cuenta que las operaciones que podamos hacer con los diferentes tipos de datos que existen solo se podrán hacer con otros datos del mismo tipo, a no ser que sean datos compatibles, como ocurría con los datos de tipo entero y los de tipo real.

Las operaciones que podremos realizar con los datos que estemos utilizando serán únicamente las compatibles con su tipo.

Por último, hemos visto la importancia de los comentarios, que sirven para aclarar qué es lo que hace nuestro código en cualquier punto del programa y que, después de un tiempo, puede ser lo que nos recuerde cómo funciona ese programa que hace tanto ya que creamos.

Si quieres aprender más, puedes echar un vistazo a las recomendaciones de código limpio en el siguiente enlace:

https://www.hostgator.mx/blog/clean-code-codigo-limpio/

Resolución del Caso Práctico de la unidad

Cuando necesitamos realizar un algoritmo que resuelva un problema vamos a tener que representar información, refiriéndonos a información como datos y siempre que haya datos vamos a tener que utilizar variables.

Cuando utilicemos variables tenemos que tener muy en cuenta el tipo que les tenemos que asignar. Según el tipo que se use, va a incluso facilitar el proceso. Así, si necesitamos representar un número tenemos que preguntarnos si tiene decimales o no, y si necesitamos representar caracteres necesitaremos elegir entre un carácter o una cadena.

Una cosa muy a tener en cuenta es que es necesario comentar el código, para así poder aclarar lo que estamos haciendo para nosotros mismos y para posibles personas que puedan llegar a ver ese código en un futuro.

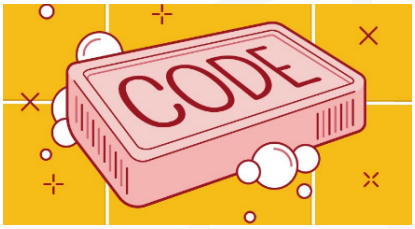

Fig.12. Es importante escribir código claro, limpio y ordenado.

/ 14. Bibliografía

Bloque de código. (2020, abril 13). Recuperado 15 de mayo de 2020, de https://es.wikipedia.org/wiki/Bloque_de_c%C3%B3digo

Conversión entre tipos primitivos (casting) - Programación 2012-2013. (s. f.-a). Recuperado 15 de mayo de 2020, de https://sites.google.com/site/pro012iessanandres/java/conversion-entre-tipos-primitivos-casting

Ramírez, J. M. (2019, marzo 23). Código Fuente. Identificadores, variables y constantes. Recuperado 15 de mayo de 2020, de https://www.masqueteclas.com/masqueteclas/013-identificadores-variables-y-constantes/