U.T. 2: IDENTIFICACIÓN DE LOS ELEMENTOS DE UN PROGRAMA INFORMÁTICO

1. ELEMENTOS DE UN PROGRAMA.

Son elementos de un programa todos aquellos manipulados por las instrucciones. Mediante ellos, en un programa podremos **realizar el almacenamiento de los datos y de los resultados** de las distintas operaciones que intervienen en la solución del problema.

Todo objeto tiene tres atributos: nombre, tipo y valor.
Nombre es el identificador del mismo.
Tipo es el conjunto de valores que puede tomar el objeto. Valor es el elemento del tipo, que se le asigna.
Ejemplos.-

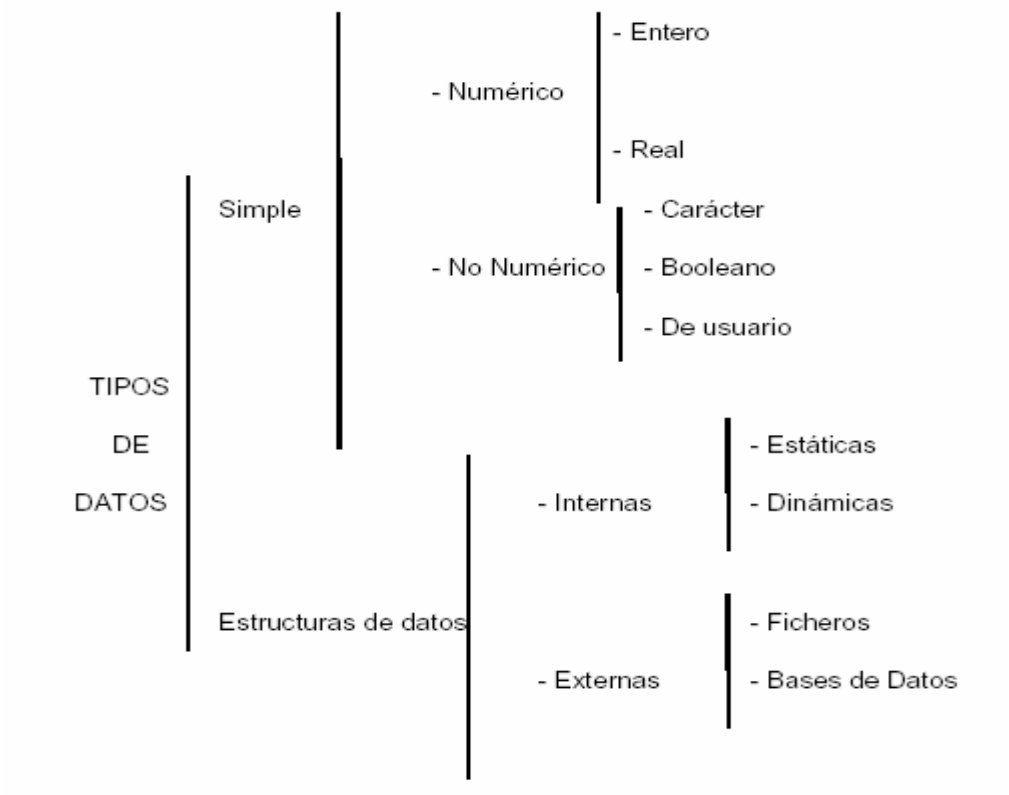
Nombre:	Edad	Peso	Nacionalidad
Tipo:	Numérico	Numérico	Carácter
Valor:	23	75	Uruguaya

1.1. Identificadores.

Son palabras creadas por el programador para dar nombre a los objetos y demás elementos que necesita declarar en un programa: **variables**, **constantes**, **tipos**, **estructuras de datos**, **archivos**, **subprogramas**, etc. En general se utiliza una cadena de letras y dígitos que empiece por una letra.

1.2. Tipos de datos.

Cada variable, constante o expresión lleva asociado un **tipo de datos** que determina el conjunto de valores que pueden tomar.



Tipo numérico Entero: Es un subconjunto de los números enteros cuyo rango o tamaño dependen del lenguaje y computadora utilizados. Se expresan mediante una cadena de dígitos que puede ir precedida de signo. Ejemplos: 4454, -12, +7

Tipo numérico Real: Es un subconjunto de los números reales limitado en tamaño y precisión:
- *notación de punto fijo*: 97.84 -12.00 +0.5
- *notación exponencial* mantisa E exponente: 0.9784E2, -12000E-4

Tipo Carácter: Es el conjunto formado por todos los caracteres o símbolos de los que dispone el ordenador. Ej. “a”, “B”, “1”, “*”. Cadenas, string: cadenas de caracteres de longitud variable.

Tipo Booleano: Es el conjunto formado por los valores true y false.

Comillas simples ' = un sólo carácter
Comillas dobles " = cadena de caracteres

1.3. Tipos de objetos.

- Valores. Ej. 18, 3.14.

- Constantes. Son objetos cuyo valor permanece invariable a lo largo de la ejecución de un programa. PI = 3.1416

- Variables. Son objetos que modificarán su valor a lo largo de la ejecución del programa.
X ← 0 X ← X + 1 X es una variable de tipo numérico.

- Expresiones. Una expresión es la representación de un cálculo necesario para la obtención de un resultado. Se define:

Un valor es una expresión
Una constante o variable es una expresión
Una función es una expresión (COS(X), SQR(25))
Una combinación de valores, constantes, variables, funciones y operadores que cumplen determinadas reglas de construcción, es una expresión.

COS(PI*X) + 1.25 2*PI*X N=”JUAN”

radio = 4 PI = 3.1416 2 * PI * radio

Las expresiones pueden ser **numéricas** (PI*SQR(X)), **alfanuméricas** ("DON" + "NADIE") o **booleanas** (A > 0 y B = 5).

1.4. Operadores.

- Aritméticos. + - * / %

- Alfanuméricos. + (concatenación. No confundir con el operador suma).

- Relacionales. > < >= <= == != ... Dan un resultado booleano al relacionar dos operandos de tipo numérico o alfanumérico.

- Lógicos. Operan dos booleanos o uno solo para dar como resultado otro booleano a través de los operadores y (AND), o (OR), no (NOT). Veamos las tablas de verdad de los operadores lógicos (cómo se comportan).

- Paréntesis. ( ) Se utilizan para anidar expresiones.

AND = &&
OR = " (poner otro debajo)
NOT = !

A	no A
V	F
F	V

A	B	A y B	A o B
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

1.5. Orden de evaluación de los operadores.

1.- Paréntesis (comenzando por los más internos) ( )
2.- Signo
3.- Potencias ^
4.- Productos y divisiones y porcentajes %
5.- Sumas y restas
6.- Concatenación (alfanumérico)
7.- Relacionales < <= > >= == !=
8.- Negación NOT
9.- Conjunción AND
10.- Disyunción OR
La evaluación de operadores de igual orden se realiza normalmente de izquierda a derecha.

Ejemplos.-

((2+4)^1 – 3)	(6^1-3)	(6-3)	3	
(30 – 10)/2*2	20/2*2	10*2	20	
((3+2)^2-15)/2*5	(5 ^ 2 – 15) / 2 * 5	(25 – 15) / 2 * 5	10 / 2 * 5	5 * 5	25
5-2>4 y no 0.5 = ½	5 – 2 > 4 y no 0.5 = 0.5		3 > 4 y no 0.5 = 0.5	FALSO
y no 0.5 = 0.5	FALSO y no CIERTO		FALSO y FALSO	FALSO

2. HERRAMIENTAS Y NOTACIONES PARA EL DISEÑO DE ALGORITMOS.

2.1. Diagramas de Flujo (flowchart).

Los diagramas de flujo nos permiten representar el recorrido de la información desde su entrada como **dato** hasta su salida como **resultado**. Son el lenguaje común para los programadores.

Dependiendo de la fase en la que los utilicemos, hablaremos de:

- ORGANIGRAMAS: Diagramas de flujos del sistema. Representación gráfica de la circulación de datos e informaciones dentro de un programa. Se utiliza en la fase de ANÁLISIS.

- ORDINOGRAMAS: Diagramas de flujos del programa. Representación gráfica de la secuencia de operaciones que se han de realizar en un programa. Se utiliza en la fase de PROGRAMACIÓN o DISEÑO.

2.2. Pseudocódigo.

Además de la utilización de representaciones gráficas, un programa se puede describir mediante un **lenguaje intermedio** entre el lenguaje natural y el lenguaje de programación.
LA UTILIZACIÓN DE UNA NOTACIÓN INTERMEDIA PERMITE EL DISEÑO DEL PROGRAMA SIN DEPENDER DE NINGÚN LENGUAJE DE PROGRAMACIÓN.

Pseudocódigo: Es una notación mediante la cual podemos describir la solución de un problema en forma de algoritmo, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas.
EL PSEUDOCÓDIGO HA DE CONSIDERARSE COMO UNA HERRAMIENTA PARA EL DISEÑO DE PROGRAMAS MÁS QUE UNA NOTACIÓN PARA LA DESCRIPCIÓN DE LOS MISMOS Debido a su flexibilidad, PERMITE OBTENER LA SOLUCIÓN A UN PROBLEMA MEDIANTE APROXIMACIONES SUCESIVAS **DISEÑO DESCENDENTE** (Programación modular).

La notación en pseudocódigo se caracteriza por:

a) No puede ser ejecutado directamente por un ordenador, por lo que tampoco es considerado como un lenguaje de programación propiamente dicho
b) Permite el diseño y desarrollo de algoritmos totalmente independientes del lenguaje de programación posteriormente utilizado en la fase de codificación del algoritmo, pues no está sujeto a las reglas sintácticas de ningún lenguaje excepto las del suyo propio
c) Es sencillo de aprender y utilizar
d) Facilita al programador el paso del algoritmo al correspondiente lenguaje de programación e) Permite una gran flexibilidad en el diseño del algoritmo a la hora de expresar acciones concretas
f) Permite con cierta facilidad la realización de futuras correcciones o actualizaciones gracias a que no es un sistema de representación rígido
g) La escritura o diseño de un algoritmo mediante el uso de esta herramienta, exige la "indentación" o "sangría" del texto en el margen izquierdo de las diferentes líneas
h) Permite obtener la solución de un problema mediante aproximaciones sucesivas, es decir, lo que se conoce comúnmente como diseño descendente o **Top down** (Programación Modular) y que consiste en la descomposición sucesiva del problema en niveles o subproblemas más pequeños, lo que nos permite la simplificación del problema general

Todo pseudocódigo debe permitir la descripción de los siguientes elementos:

Instrucciones de entrada/salida y de asignación(PRIMITIVAS o SIMPLES)
Instrucciones de declaración
Sentencias de control del flujo de ejecución
Acciones compuestas (subprogramas) que hay que refinar

posteriormente. La estructura general de un pseudocódigo es la siguiente:

Programa NOMBRE DEL PROGRAMA

Entorno:

Descripción del conjunto de objetos de un programa: declaración de sus nombres y tipos

Algoritmo:

Secuencia de instrucciones que forman el programa

Finalgoritmo

Subprograma NOMBRE DEL SUBPROGRAMA

Entorno:

Algoritmo:

Finalgoritmo

Ejemplo.- Pseudocódigo de un programa que calcule el área de un rectángulo. Se debe introducir la base y la altura para realizar el cálculo:

```
Programa CALCULA AREA
Entorno:
BASE, AREA, ALTURA son numéricas enteras (suponemos que son enteros)
Algoritmo:
       escribir "Introduzca la base y la altura"
       leer BASE, ALTURA
       AREA ← BASE*ALTURA
       escribir "El área del rectángulo es la siguiente ", AREA
Finalgoritmo
```

3. ESTRUCTURA GENERAL DE UN PROGRAMA.

3.1. Introducción.

Un programa puede considerarse como una secuencia lógica de acciones (instrucciones) que manipulan un conjunto de objetos (datos) para obtener unos resultados que serán la solución al problema que resuelve dicho programa-algoritmo.
Todo programa tiene dos partes o bloques. El primer bloque es el bloque de declaraciones. En éste se especifican todos los objetos que utiliza el programa (constantes, variables, etc.) indicando sus características. El segundo bloque es el de instrucciones formado por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados.
La ejecución de un programa consiste en la realización secuencial del conjunto de instrucciones de que se compone. Las instrucciones de un programa consisten en general en modificaciones sobre los objetos del programa, desde un estado inicial hasta otro final. Al conjunto de objetos de un programa se le llama también **entorno** del programa.

Las partes principales de un programa serían tres:

1) Entrada de datos (desde los dispositivos externos hasta memoria central).
2) Proceso (paso de un estado inicial a un estado final).
3) Obtención de resultados (desde memoria central hacia los dispositivos externos).

3.2. Instrucciones o sentencias.

Una instrucción se caracteriza por un **estado inicial**, que es el estado de los objetos antes de

la ejecución de la instrucción, y otro **estado final** que es en el que quedan los objetos después de la ejecución de la misma.

3.2.1. Instrucciones de declaración.

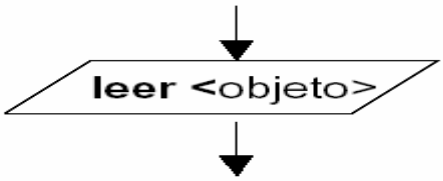
Anuncian la utilización de objetos en un programa indicando qué identificador, tipo y otras características corresponde a cada uno de ellos.

NombreObjeto ES TipoObjeto

3.2.2. Instrucciones de entrada.

Su misión consiste en tomar uno o varios datos desde un dispositivo de entrada y almacenarlos en la Memoria Central, en los objetos cuyos identificadores aparecen en la propia instrucción. Su sintaxis metodológica es:

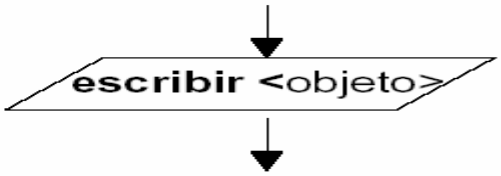
leer <lista de objetos>



3.2.3. Instrucciones de salida.

Es el conjunto de instrucciones que muestran el valor de algunos objetos en los dispositivos de salida (monitor...). Sintaxis:

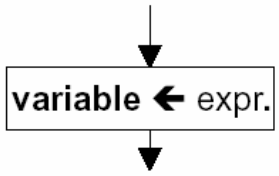
escribir <lista de objetos>



3.2.4. Instrucciones de asignación.

Permiten realizar cálculos evaluando una expresión y depositando su valor final en un objeto o realizar movimientos de datos de un objeto a otro. Su sintaxis es:

variable ← expresión



Ejemplo.- EDAD ← EDAD_MAX

Una expresión puede ser un valor, una variable, una constante, una función o una combinación de todo lo anterior empleando operadores.

Ejemplo.- EDAD ← 17 * EDAD

Esta instrucción **se realiza en dos tiempos**; primero se evalúa la expresión convirtiéndose en su valor final; segundo, el valor final se asigna al objeto borrándose el valor previo que éste pudiese tener. **El objeto y la expresión deben coincidir en tipo** y se admite que el propio objeto que recibe el valor final de la expresión pueda intervenir en la misma, pero entendiéndose que lo hace con su valor anterior.

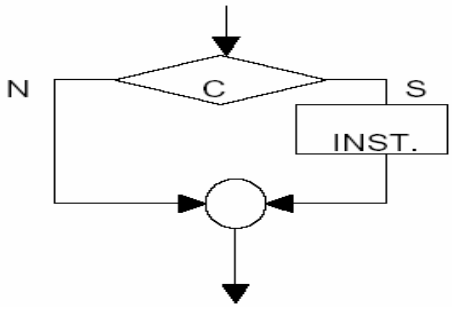
3.2.5. Instrucciones de control.

Son instrucciones que tienen como objetivo el controlar la ejecución de otras instrucciones o alterar el orden de ejecución normal de las mismas.

a) Instrucciones alternativas: controlan la ejecución de uno o varios bloques de instrucciones dependiendo del cumplimiento o no de alguna condición o del valor final de una expresión.

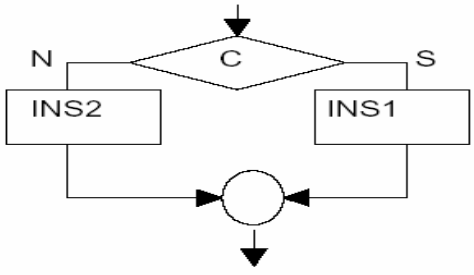
a1) *Alternativa simple*. Controla la ejecución de un conjunto de instrucciones por el cumplimiento o no de una condición. Sintaxis:

si CONDICION **entonces**
    Ins1
    ...
    Insn
finsi



a2) *Alternativa doble*. Controla la ejecución de dos conjuntos de instrucciones por el cumplimiento o no de una condición. Si se cumple, se ejecutan las instrucciones del primer bloque y si no se cumple, las del segundo. Es una ampliación del anterior. Sintaxis:

si CONDICION **entonces**
    ...
si no
    ...
finsi



a3) *Alternativa múltiple*. Controla la ejecución de varios conjuntos de instrucciones por el valor final de una expresión, de tal forma que cada conjunto de instrucciones está

ligado a un posible valor de la expresión. Se ejecutará el conjunto que se encuentre relacionado con el valor que resulte de la evaluación de la expresión. Sintaxis:

opción EXPRESION **de**

V1 **hacer** Ins1, Ins2....

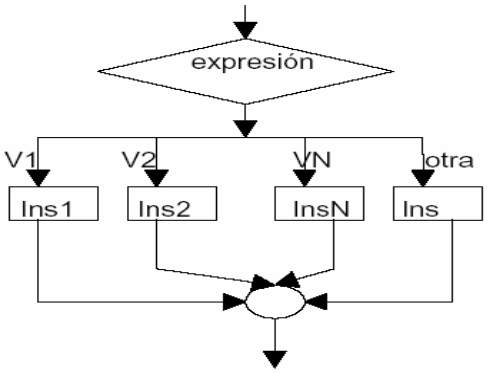
V2 **hacer** Ins1, Ins2...

....

VN **hacer** Ins1, Ins2...

otra hacer ....

finopción



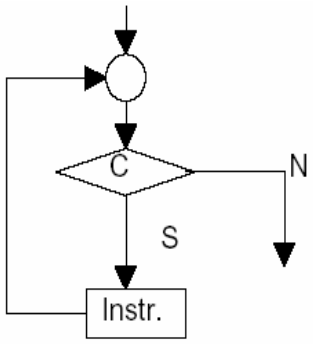
b) Instrucciones repetitivas: Son aquellas que controlan la repetición de un conjunto de instrucciones denominado ***rango*** mediante la evaluación de una condición que se realiza cada nueva repetición o por medio de un *contador* asociado.

b1) *Instrucción MIENTRAS*. El conjunto de instrucciones que configuran su rango se ejecutan mientras se cumpla la condición que será evaluada siempre antes de cada repetición, es decir, mientras la condición sea CIERTA.

mientras CONDICION **hacer**

....

finmientras

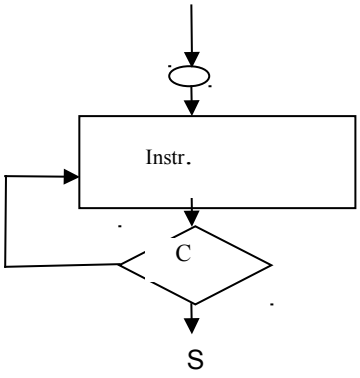


b2) *Instrucción REPETIR*. Controla la ejecución del conjunto de instrucciones que configuran su rango de tal forma que éstas se ejecutan hasta que se cumpla la condición, que será evaluada siempre después de cada repetición, es decir, hasta que la condición sea CIERTA.

repetir

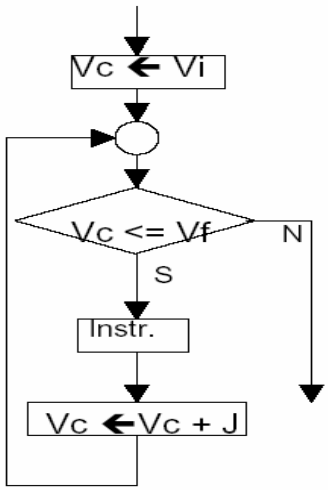
…….

hasta CONDICIÓN



b3) *Instrucción PARA*. Controla la ejecución del conjunto de instrucciones de su rango de tal forma que éstas se ejecutan un número determinado de veces que queda definido en lo que se denomina cabecera del bucle. En la cabecera se define una variable de control del bucle, su valor inicial, su valor final y su incremento o decremento. Con esta clase de bucle nos ahorramos instrucción de incremento al estar especificada en la cabecera del bucle. Es ideal para repeticiones con incrementos fijos. Su sintaxis, así como un ejemplo se muestra en la siguiente imagen:

```
para Vc de Vi a Vf con incremento/decremento J hacer
        ....
        ....
finpara
resultado ← 1
para H de 1 a potencia con incremento 1 hacer
        resultado ← resultado * numero
finpara
escribir resultado
```



3.3. Variables auxiliares.

3.3.1. Contadores.

Son variables que se utilizan para contar cualquier evento que pueda ocurrir dentro de un programa. Se utilizan realizando sobre ellos dos operaciones básicas: **inicialización**, e **incremento**.

Ejemplo.- Programa que lee una lista de 50 notas de alumnos e indica cuantos están aprobados:

```
Programa: APROBADOS
Entorno:
        NUMERO (contador) es numérico entero
        APROB (contador de aprobados) es numérico entero
        NOTA es numérico entero
Algoritmo:
        APROB ← 0
        NUMERO ← 0
        Mientras NUMERO < 50 hacer
                NUMERO ← NUMERO + 1
                Leer NOTA
                Si NOTA >= 5 entonces
                        APROB ← APROB + 1
                Finsi
        Finmientras
        Escribir APROB
Finalgoritmo
```

3.3.2. Acumuladores.

Se utilizan para realizar sumatorios o productos de distintas cantidades. Para el sumatorio se inicializan a 0 y para el producto a 1. Para utilizarlos se realizan sobre ellos dos operaciones básicas: **inicialización**, y **acumulación**.

Ejemplo.- Algoritmo que calcula y escribe la suma y el producto de los 10 primeros números naturales:

```
Programa: SUMANAT
Entorno:
       SUMA (acumulador) es numérico entero
       PRODUCTO (acumulador) es numérico entero
       CONTA es numérico entero
Algoritmo:
       SUMA ← 0
       PRODUCTO ← 1
       CONTA ← 1
       Mientras CONTA <= 10  hacer
              SUMA ← SUMA + CONTA
              PRODUCTO ← PRODUCTO * CONTA
              CONTA ← CONTA + 1
       Finmientras
       Escribir SUMA, PRODUCTO
Finalgoritmo
```

3.3.3. Interruptores, conmutadores o switches.

Son variables que pueden tomar dos valores exclusivamente. Normalmente 0 y 1, -1 y +1, cierto y falso, etc. Se utilizan para transmitir información de un punto a otro de un programa y para conmutar alternativamente entre dos caminos posibles. Se utilizan inicializándolos con un valor y en los puntos en que corresponda se cambian al valor contrario, de tal forma que examinando su valor posteriormente podemos realizar la transmisión de información que deseábamos

Ejemplo.- Algoritmo que lee una secuencia de notas (con valores que van de 0 a 10) que termina con el valor –1 y nos dice si hubo o no alguna nota con valor 10:

```
Programa: NOTAS
Entorno:
       NOTA  es numérico entero
       NOTA10 booleano (switch)
Algoritmo:
       NOTA10 ← FALSO
       Leer NOTA
       Mientras  NOTA <> -1 hacer
              Si NOTA = 10 entonces
                                    NOTA10 ← CIERTO
              Finsi
              Leer NOTA
       Finmientras
       Si NOTA10 = CIERTO entonces
              Escribir "Hubo 10"
       Sino
              Escribir "No hubo 10"
       Finsi
Finalgoritmo
```

4. EL LENGUAJE JAVA.

4.1. El lenguaje Java. Características generales.

Java está relacionado con C++, que es descendiente directo de C. La mayor parte del carácter de Java está heredado de estos dos lenguajes. De C, Java deriva su sintaxis. La mayoría de las características orientadas a objetos están basados en C++. Fue concebido por James Gosling, Patrick Naughton, Chris Warth, Ed Frank y Mike Sheridan en Sun Microsystems Inc. en 1991. La idea de Java evolucionó por dos razones: para adaptarse a los cambios del entorno y para introducir avances en el arte de programación.

4.1.1. Características generales

Internet ha ayudado a Java a situarse como líder de los lenguajes de programación y, por su lado, Java ha tenido un profundo efecto sobre Internet.
Con Java se pueden crear dos tipos de programas: aplicaciones y applets.
Una aplicación es un programa que se ejecuta en el ordenador, utilizando un sistema operativio de ese ordenador.
Una applet es un pequeño programa Java que se transfiere dinámicamente a través de la red, comi si fuese una imagen, un archivo de sonido o vídeo, es un programa inteligente, es decir, es un programa que puede reaccionar ante las acciones del usuario y cambiar dinámicamente, no sólo ejecutar repetidamente la misma animación.

Java ofrece:
a) Seguridad, cuando se utiliza un navegador compatible con Java, es posible transferir de forma segura las applets de Java sin temor a ser infectados por un virus o a recibir intentos de acceso malicioso.
b) Portabilidad, permite generar código ejecutable que sea portable a todos los tipos de plataformas que están conectadas a Internet (distintos tipos de ordenadores y de sistemas operativos en todo el mundo).

La llave que permite a Java resolver los problemas de seguridad y portabilidad es que la salida de un compilador Java no es código ejecutable, sino que es código binario (bytecode). Este código binario es un conjunto de instrucciones diseñado para ser ejecutado en una máquina virtual que emula el intérprete de Java.
Es decir, el intérprete de Java es un intérprete de código binario.
Java fue diseñado para ser un lenguaje interpretado, al no ser compilados, es mucho más fácil ejecutarlos en una gran variedad de entornos. Cada entorno tendrá su propio intérpete de Java.
Si Java fuese un lenguaje compilado tendrían que existir diferentes versiones del mismo programa para cada uno de los tipos de CPU que están conectados a Internet.

Es un lenguaje orientado a objetos, que incorpora las características de control apuntadas como deseables por la teoría y práctica de la informática.

- Simple.
- Robusto.
- Multihilo. (permite escribir programas que hacen varias cosas a la vez)
- Distribuido. (Trabaja con el protocolo TCP/IP para ejecutar procedimientos remotos)
- Interpretado.

La generación de lenguajes de programación anteriores a Java (C, COBOL, FORTRAN, PASCAL, etc.), se basan en el modelo orientado a proceso, que consiste en construir un programa como una serie de pasos lineales, es decir código, es decir “el código actúa sobre los datos”. Pero cuando los programas crecen y se hacen más complejos, se siente la necesidad de crear otro estilo de programación: “La programación orientada a objetos”. (la iremos viendo los temas siguientes)

4.2. Programa HOLA MUNDO

Un programa en Java consiste en una clase que contiene uno o más métodos, de los cuales uno de ellos, debe llamarse ***main()*** y es el principal de todos.

El programa comienza con la clase que contiene el método :
public static void *main(String args[])*
Cada método y clase, consta de un cuerpo delimitado por llaves de comienzo y fin ***{ }***
En el cuerpo de la clase irán incluidas: sentencias, variables, objetos, métodos, etc. terminadas cada una de ellas por un punto y coma ***;***

```
public static void main(String args[]) {
        variables;
        sentencias;
}
```

Programa ejemplo:

```
/* Este programa imprime un mensaje */
import java.io.*;
public class Principal
{
 public static void main(String args[])
 {
        System.out.println("HOLA MUNDO\n");
 }
}
```

4.3. Identificadores

Un identificador: un nombre con el que se hace referencia a un método, clase y variable. . Reglas:

- Un identificador se forma con una secuencia de letras (minúsculas de la a a la z; mayúsculas de la A a la Z; y dígitos del 0 al 9). **$ no se puede usar**
- Los caracteres subrayado o underscore (_) y dólar ($) se consideran como una letra más. Un identificador no puede contener espacios en blanco, ni otros caracteres distintos de los citados, como por ejemplo (*,;.:-+, etc.).
- El primer carácter de un identificador debe ser siempre una letra o un (_), es decir, no puede ser un dígito.
- Se hace distinción entre letras mayúsculas y minúsculas. Así, Masa es considerado como un identificador distinto de masa y de MASA.
- ANSI C permite definir identificadores de hasta 31 caracteres de longitud.

Ejemplos de identificadores válidos son los siguientes:
tiempo, distancia1, caso_A, PI, velocidad_de_la_luz
Por el contrario, los siguientes nombres no son válidos (¿Por qué?)
1_valor, tiempo-total, dolares$, %final

En general, es muy aconsejable elegir los nombres de los métodos, clases y las variables de forma que permitan conocer a simple vista qué tipo de variable, clase o método representan, utilizando para ello tantos caracteres como sean necesarios. Esto simplifica enormemente la tarea de programación y –sobre todo– de corrección y mantenimiento de los programas. Es cierto que los nombres largos son más laboriosos de teclear, pero en general resulta rentable tomarse esa pequeña molestia.

Palabras reservadas

abstract	const	finally	int	public	throw
boolean	continue	float	interface	return	throws
break	default	for	long	short	transient
byte	do	goto	native	static	try

case	double	if	new	super	void
catch	else	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	final	instanceof	protected	this	

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- Tipos de datos: **boolean, float, double, int, char**
- Sentencias condicionales: **if, else, switch**
- Sentencias iterativas: **for, do, while, continue**
- Tratamiento de las excepciones: **try, catch, finally, throw**
- Estructura de datos: **class, interface, implements, extends**
- Modificadores y control de acceso: **public, private, protected, transient**
- Otras: **super**, **null, this**.

4.4. Comentarios

El lenguaje Java permite que el programador introduzca comentarios en los ficheros fuente que contienen el código de su programa. La misión de los comentarios es servir de explicación o aclaración sobre cómo está hecho el programa, de forma que pueda ser entendido por una persona diferente (o por el propio programador algún tiempo después) El compilador ignora por completo los comentarios. A continuación se muestran los dos tipos de comentarios que admite Java.

```
// Comentario para una sola línea
/*Comentario para una
        o varias líneas
*/
```

Separadores

En Java, hay algunos caracteres que se utilizan como separadores.

Símbolo	*Nombre*	*Para qué se utiliza*
()	Paréntesis	Para contener listas de parámetros en la definición y llamada a método. También se utiliza para definir precedencia en expresiones, contener expresiones en sentencias de control de flujo y en conversiones.
{}	Llaves	Para contener los valores de matrices inicializadas automáticamente. También se utiliza para definir un bloque de código, para clases, métodos y ámbitos locales.
[]	corchetes	Para declarar tipos matriz. También se utiliza cuando se referencian valores de matriz.
;	Puntos y coma	Separa sentencias
,	coma	Separa identificadores consecutivos en una declaración de variables. También se utiliza para encadenar sentencias dentro de una sentencia for.
.	Para separar	Para separar nombres de paquete, subpaquetes y clases. También se utiliza para separar una variable o método de una variable de referencia.

4.5. Constantes y valores literales.

Las variables pueden cambiar de valor a lo largo de la ejecución de un programa, o bien en ejecuciones distintas de un mismo programa. Además de variables, un programa utiliza también constantes, es decir, valores que siempre son los mismos. Un ejemplo típico es el número π que vale 3.141592654. Este valor, con más o menos cifras significativas, puede aparecer muchas veces en las sentencias de un programa. En Java existen distintos tipos de constantes:

Constantes numéricas. Son valores numéricos, enteros o de punto flotante. Se permiten también constantes octales (números enteros en base 8) y hexadecimales (base 16).
Ej, 100

Constantes carácter. Cualquier carácter individual encerrado entre apóstrofos (tal como 'a', 'Y', ')', '+', etc.) es considerado por Java como una constante carácter, o en realidad como un número entero pequeño (entre 0 y 255, o entre -128 y 127, según los sistemas). Existe un código, llamado código ASCII, que establece una equivalencia entre cada carácter y un valor numérico correspondiente.
Ej, 'X'

Tabla ASCII
128 caracteres

D / I	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	0	1	2	3	4	5	6	7	8	9	*LF* 10	11	12	*CR* 13	14	15
0001	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0010	*SP* 32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	' 44	- 45	. 46	/ 47
0011	**0** 48	**1** 49	**2** 50	**3** 51	**4** 52	**5** 53	**6** 54	**7** 55	**8** 56	**9** 57	: 58	; 59	< 60	= 61	> 62	? 63
0100	@ 64	**A** 65	**B** 66	**C** 67	**D** 68	**E** 69	**F** 70	**G** 71	**H** 72	**I** 73	**J** 74	**K** 75	**L** 76	**M** 77	**N** 78	**O** 79
0101	**P** 80	**Q** 81	**R** 82	**S** 83	**T** 84	**U** 85	**V** 86	**W** 87	**X** 88	**Y** 89	**Z** 90	[91	\ 92	] 93	^ 94	_ 95
0110	` 96	**a** 97	**b** 98	**c** 99	**d** 100	**e** 101	**f** 102	**g** 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
0111	**p** 112	**q** 113	**r** 114	**s** 115	**t** 116	**u** 117	**v** 118	**w** 11	**x** 120	**y** 121	**z** 122	{ 123	| 124	} 125	~ 126	127

Caracteres 0-31: no imprimibles (p.e. 10: LF, fin de línea; 13:CR, "retorno de carro")
Caracter 32: "Espacio en blanco" (SP)
Caracter 127: no imprimible (DEL)

Cadenas de caracteres. Un conjunto de caracteres alfanuméricos encerrados entre comillas es también un tipo de constante del lenguaje Java, como por ejemplo: "espacio","Esto es una cadena de caracteres", etc.

Un tipo especial de carácter es la secuencia de escape, similares a las del lenguaje C/C++, que se utilizan para representar caracteres de control o caracteres que no se imprimen. Una secuencia de escape está formada por la barra invertida (\) y un carácter. En la siguiente tabla se dan las secuencias de escape más utilizadas.

Carácter	Secuencia de escape
retorno de carro	**\r**
tabulador horizontal	**\t**
nueva línea	**\n**
barra invertida	****

Constantes simbólicas. Las constantes simbólicas tienen un nombre (identificador) y en esto se parecen a las variables. Sin embargo, no pueden cambiar de valor a lo largo de la ejecución del programa.

Sintaxis: final tipo_dato identificador = valor;
final double PI=3.14159;

4.5. Programa MUESTRA EL DOBLE DE 100

```
Ej,/* Este programa imprime el doble de 100 */
        import java.io.*;
        public class Principal
        {
         public static void main(String args[])
         {
                int num; //Declara una variable llamada num
                num=100;
                System.out.print("Esto es num:" + num);
                num=num *2;
                System.out.print("El valor de num * 2 es ");
                System.out.println(num);
         }
 }
```

4.6. Variables

Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable. Hay varios tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas, la declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable. Una vez declarada se le podrá asignar valores.

Java tiene tres tipos de variables:

- de instancia
- de clase
- locales

Las variables de instancia o miembros dato como veremos más adelante, se usan para guardar los atributos de un objeto particular.

Las variables de clase o miembros dato estáticos son similares a las variables de instancia, con la excepción de que los valores que guardan son los mismos para todos los objetos de una determinada clase. En el siguiente ejemplo, *PI* es una variable de clase y *radio* es una variable de instancia. *PI* guarda el mismo valor para todos los objetos de la clase *Circulo*, pero el radio de cada círculo puede ser diferente

```
class Circulo{
      static final double PI=3.1416;
      double radio;
//...
}
```

Las variables locales se utilizan dentro de las funciones miembro o métodos. En el siguiente ejemplo *area* es una variable local a la función *calcularArea* en la que se guarda el valor del área de un objeto de la clase *Circulo*. Una variable local existe desde el momento de su definición hasta el final del bloque en el que se encuentra.

```
class Circulo{
//...
      double calcularArea(){
            double area=PI*radio*radio;
            return area;
      }
}
```

En el lenguaje Java, las variables locales se declaran en el momento en el que son necesarias. Es una buena costumbre inicializar las variables en el momento en el que son declaradas. Veamos algunos ejemplos de declaración de algunas variables

```
int x=0;
String nombre="Angel";
double a=3.5, b=0.0, c=-2.4;
boolean bNuevo=true;
int[] datos;
```

Delante del nombre de cada variable se ha de especificar el tipo de variable que hemos destacado en letra negrita. Las variables pueden ser

- Un tipo de dato primitivo
- El nombre de una clase
- Un array

El lenguaje Java utiliza el conjunto de caracteres Unicode, que incluye no solamente el conjunto ASCII sino también carateres específicos de la mayoría de los alfabetos. Así, podemos declarar una variable que contenga la letra ñ

```
int año=1999;
```

Se ha de poner nombres significativos a las variables, generalmente formados por varias palabras combinadas, la primera empieza por minúscula, pero las que le siguen llevan la letra inicial en mayúsculas. Se debe evitar en todos los casos nombres de variables cortos como *xx*, *i*, etc.

```
double radioCirculo=3.2;
```

Las variables son uno de los elementos básicos de un programa, y se deben

- Declarar
- Inicializar
- Usar

Sintaxis para declarar una variable:
tipo_dato nombre_variable;

También se le puede dar un valor inicial a la vez que se declara:
tipo_dato nombre_variable=valor;

4.6.1. Utilización de variables y alcance

a) ***Bloques de código***
Un bloque de código es un grupo de sentencias que se comportan como una unidad. Un bloque de código está limitado por las llaves de apertura **{** y cierre **}.**

Como ejemplos de bloques de código tenemos la definición de una clase, la definición de una función miembro, una sentencia iterativa **for**, los bloques **try ... catch**, para el tratamiento de las excepciones, etc.

b) ***Alcance o ámbito de las variables***
La existencia de una variable se conoce en el bloque donde se creó.

4.7. Tipos de Datos Simples en Java

Cada vez que queramos crear una variable, hemos de tener muy claro que clase de valores va a tener, así como el rango posible de los mismos (desde el valor mínimo al máximo). En base a ello, elegiremos un tipo de datos de entre los siguientes:

Para crear una variable (de un tipo simple) en memoria debe declararse indicando su tipo de variable y su identificador que la identificará de forma única. La sintaxis de declaración de variables es la siguiente:

TipoSimple Identificador; //declaración de la variable

TipoSimple Identifiador3=valor; //se le puede dar un valor inicial a la vez que se declara.

TipoSimple Identificador1, Identificador2;

Esta sentencia indica al compilador que reserve memoria para dos variables del tipo simple *TipoSimple* con nombres *Identificador1* e *Identificador2.*

Cada tipo de datos simple soporta un conjunto de literales que le pueden ser asignados, para darles valor.

a) ***Tipos de datos enteros.***
Se usan para representar números enteros con signo. Hay cuatro tipos: *byte, short, int y long.*

Tipo	**Tamaño**	
byte	1Byte (8 bits) Rango: -128 a 127	
short	2 Bytes (16 bits) Rango -32768 a 32767	
int	4 Bytes (32 bits) Rango: -214743648 a 214743647	
long	8 Bytes (64 bits)	

b) ***Tipos de datos en coma flotante***
Se usan para representar números con partes fraccionarias. Hay dos tipos de coma flotante: *float* y *double*. El primero reserva almacenamiento para un número de precisión simple de 4 bytes y el segundo lo hace para un numero de precisión doble de 8 bytes.

Tipo	**Tamaño**
float	4 Byte (32 bits) Aproximadamente -3.4*10-38 a 3.4*1038 con precisión de 7 dígitos
double	8 Bytes (64 bits) con precisión de 15 dígitos

La declaración de variables de coma flotante es muy similar a la de las variables enteras. Por ejemplo:

float f = 12.3f;

c.) Tipo de datos boolean

Se usa para almacenar variables que presenten dos estados, que serán representados por los valores *true* y *false*.

Para declarar un dato del tipo booleano se utiliza la palabra reservada *boolean:*

boolean reciboPagado = false; // ¡¿Aun no nos han pagado?!

d.) Tipo de datos carácter

Se usa para almacenar caracteres *Unicode* simples. Debido a que el conjunto de caracteres *Unicode* se compone de valores de 16 bits, el tipo de datos *char* se almacena en un entero sin signo de 16 bits (2 Bytes).

e.) Conversión de tipos de datos

En Java es posible transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado. Este proceso se denomina "conversión", "moldeado" o "tipado". La conversión se lleva a cabo colocando el tipo destino entre paréntesis, a la izquierda del valor que queremos convertir de la forma siguiente:

float x=14.7f;

double y =(double) x;

Por ello se establece la norma de que "en las conversiones el tipo destino siempre debe ser igual o mayor que el tipo fuente":

Tipo Origen	Tipo Destino
byte	*double, float, long, int, char, short*
short	*double, float, long, int*
char	*double, float, long, int*
int	*double, float, long*
long	*double, float*
float	*double*

Tabla Conversiones sin pérdidas de información

4.8. Operadores

Los operadores son un tipo de símbolos que indican una evaluación o computación para ser realizada en objetos o datos, y en definitiva sobre identificadores o constantes.

Además de realizar la operación, un operador devuelve un valor, ya que son parte fundamental de las expresiones.

El valor y tipo que devuelve depende del operador y del tipo de sus operandos. Por ejemplo, los operadores aritméticos devuelven un número como resultado de su operación.

Los operadores realizan alguna función sobre uno, dos o tres operandos.

Los operadores que requieren un operando son llamados *operadores unarios.* Por ejemplo, el operador "++" es un operador unario que incrementa el valor de su operando en una unidad.

Los operadores unarios en Java pueden utilizar tanto la notación prefija como la posfija.

La notación prefija indica que el operador aparece antes que su operando.

++contador // Notación prefija, se evalúa a: contador+1

La notación posfija indica que el operador aparece después de su operando:

contador++ // Notación posfija, se evalúa a: contador

Los operadores que requieren dos operandos se llaman *operadores binarios*. Por ejemplo el

operador "=" es un operador binario que asigna el valor del operando del lado derecho al operando del lado izquierdo.

Todas los operadores binarios en Java utilizan notación infija, lo cual indica que el operador aparece entre sus operandos.

operando1 operador operando2

Por último, los operadores ternarios son aquellos que requieren tres operandos. El lenguaje Java tiene el operador ternario, "*?*":, que es una sentencia similar a la if-else.

Este operador ternario usa notación infija; y cada parte del operador aparece entre operandos:

expresión ? operación1 : operación2

Los operadores de Java se pueden dividir en las siguientes cuatro categorías:

- Aritméticos.
- De comparación y condicionales.
- A nivel de bits y lógicos.
- De asignación.

A. Operadores aritméticos

El lenguaje Java soporta varios operadores aritméticos para los números enteros y en coma flotante. Se incluye + (suma), - (resta), * (multiplicación), / (división), y % (módulo, es decir, resto de una división entera). Por ejemplo:

sumaEste + aEste; //Suma los dos enteros

divideEste % entreEste; //Calcula el resto de dividir 2 enteros

Operador	Uso	Descripción
+	*op1 + op2*	Suma op1 y op2
-	*op1 - op2*	Resta op2 de op1
*	*op1 * op2*	Multiplica op1 por op2
/	*op1 / op2*	Divide op1 por op2
%	*op1 % op2*	Calcula el resto de dividir op1 entre op2

Tabla 9: Operadores aritméticos binarios de Java

El tipo de los datos devueltos por una operación aritmética depende del tipo de sus operandos; si se suman dos enteros, se obtiene un entero como tipo devuelto con el valor de la suma de los dos enteros.

Estos operadores se deben utilizar con operandos del mismo tipo, o si no realizar una conversión de tipos de uno de los dos operandos al tipo del otro.

El lenguaje Java sobrecarga la definición del operador + para incluir la concatenación de cadenas. El siguiente ejemplo utiliza + para concatenar la cadena "*Contados "*, con el valor de la variable *contador* y la cadena " *caracteres."*:

System.out.print("Contados" + contador + "caracteres.");

Esta operación automáticamente convierte el valor de *contador* a una cadena de caracteres.

Los operadores + y - tienen versiones unarias que realizan las siguientes operaciones:

Operador	Uso	Descripción

+	+op	Convierte op a entero si es un byte, short o char
-	-op	Niega aritméticamente op

Tabla Versiones unarias de los operadores "+" y "-"

Existen dos operadores aritméticos que funcionan como atajo de la combinación de otros: ++ que incrementa su operando en 1, y -- que decrementa su operando en 1.

Ambos operadores tienen una versión prefija, y otra posfija. La utilización la correcta es crítica en situaciones donde el valor de la sentencia es utilizado en mitad de un cálculo más complejo, por ejemplo para control de flujos:

Operador	Uso	Descripción
++	*op*++	Incrementa op en 1; se evalúa al valor anterior al incremento
++	++*op*	Incrementa op en 1; se evalúa al valor posterior al incremento
--	*op*--	Decrementa op en 1; se evalúa al valor anterior al incremento
--	--*op*	Decrementa op en 1; se evalúa al valor posterior al incremento

Tabla 11: Operaciones con "++" y "--"

B. Operadores de comparación y condicionales

Un operador de comparación compara dos valores y determina la relación existente entre ambos. Por ejemplo, el operador *!=* devuelve verdadero (*true*) si los dos operandos son distintos. La siguiente tabla resume los operadores de comparación de Java:

Operador	Uso	Devuelve verdadero si
>	op1 > op2	op1 es mayor que op2
>=	op1 >= op2	op1 es mayor o igual que op2
<	op1 < op2	op1 es menor que op2
<=	op1 <= op2	op1 es menor o igual que op2
==	op1 == op2	op1 y op2 son iguales
!=	op1 != op2	op1 y op2 son distintos

Tabla 12: Operadores de comparación

Los operadores de comparación suelen ser usados con los operadores condicionales para construir expresiones complejas que sirvan para la toma de decisiones. Un operador de este tipo es *&&*, el cual realiza la operación booleana *and*. Por ejemplo, se pueden utilizar dos operaciones diferentes de comparación con *&&* para determinar si ambas relaciones son ciertas. La siguiente línea de código utiliza esta técnica para determinar si la variable *index* de una matriz se encuentra entre dos límites (mayor que cero y menor que la constante *NUMERO_ENTRADAS)*:

(0 < index) && (index < NUMERO_ENTRADAS)

Se debe tener en cuenta que en algunos casos, el segundo operando de un operador condicional puede no ser evaluado. En caso de que el primer operando del operador *&&* valga falso, Java no evaluará el operando de la derecha:

(contador < NUMERO_ENTRADAS) && (in.read() != -1)

Si *contador* es menor que *NUMERO_ENTRADAS*, el valor de retorno de *&&* puede ser determinado sin evaluar el operando de la parte derecha. En este caso *in.read* no será llamado y un carácter de la entrada estándar no será leído.

Si el programador quiere que se evalúe la parte derecha, deberá utilizar el operador *&* en lugar de *&&*.

De la misma manera se relacionan los operadores || y | para la exclusión lógica (OR).

Java soporta cinco operadores condicionales, mostrados en la siguiente tabla:

Operador	Uso	Devuelve verdadero si...
&&	*op1 && op2*	op1 y op2 son ambos verdaderos, condicionalmente evalúa op2
&	*op1 & op2*	op1 y op2 son ambos verdaderos, siempre evalúa op1 y op2
||	*op1* || *op2*	op1 o op2 son verdaderos, condicionalmente evalúa op2
|	*op1* | *op2*	op1 o op2 son verdaderos, siempre evalúa op1 y op2
!	*! op*	op es falso

Tabla 13: Operadores condicionales

Además Java soporta un operador ternario, el *?:*, que se comporta como una versión reducida de la sentencia *if-else*:

expresion ? operacion1 : operacion2

El operador *?*: evalúa la *expresion* y devuelve *operación1* si es cierta, o devuelve *operación2* si *expresion* es falsa.

C. Operadores de asignación

El operador de asignación básico es el =, que se utiliza para asignar un valor a otro. Por ejemplo:

int contador = 0;

Inicia la variable *contador* con un valor *0.*

Java además proporciona varios operadores de asignación que permiten realizar un atajo en la escritura de código. Permiten realizar operaciones aritméticas, lógicas, de bit y de asignación con un único operador.

Supongamos que necesitamos sumar un número a una variable y almacenar el resultado en la misma variable, como a continuación:

i = i + 2;

Se puede abreviar esta sentencia con el operador de atajo +=, de la siguiente manera*:*

i += 2;

La siguiente tabla muestra los operadores de atajo de asignación y sus equivalentes largos:

Operador	Uso	Equivalente a

+=	*op1 += op2*	op1 = op1 + op2
-=	*op1 -= op2*	op1 = op1 - op2
*=	*op1 *= op2*	op1 = op1 * op2
/=	*op1 /= op2*	op1 = op1 / op2
%=	*op1 %= op2*	op1 = op1 % op2
&=	*op1 &= op2*	op1 = op1 & op2

Tabla : Operadores de atajo de asignación

4.9. Expesiones aritméticas y lógicas. Precedencia y asociatividad

Cuando en una sentencia aparecen varios operadores el compilador deberá de elegir en qué orden aplica los operadores. A esto se le llama *precedencia*.

Los operadores con mayor precedencia son evaluados antes que los operadores con una precedencia relativa menor.

Cuando en una sentencia aparecen operadores con la misma precedencia:

- Los operadores de asignación son evaluados de derecha a izquierda.
- Los operadores binarios, (menos los de asignación) son evaluados de izquierda a derecha.

Se puede indicar explícitamente al compilador de Java cómo se desea que se evalúe la expresión con paréntesis balanceados ( ). Para hacer que el código sea más fácil de leer y mantener, es preferible ser explícito e indicar con paréntesis que operadores deben ser evaluados primero.

La siguiente tabla muestra la precedencia asignada a los operadores de Java. Los operadores de la tabla están listados en orden de precedencia: cuanto más arriba aparezca un operador, mayor es su precedencia. Los operadores en la misma línea tienen la misma precedencia:

Tipo de operadores	**Operadores de este tipo**
Operadores posfijos	[ ] . (parametros) expr++ expr--
Operadores unarios	++expr --expr +expr -expr ~ !
Creación o conversión	new (tipo) expr
Multiplicación	* / %
Suma	+ -
Desplazamiento	<<
Comparación	< <= = instanceof
Igualdad	== !=
AND a nivel de bit	&

OR a nivel de bit	^
XOR a nivel de bit	|
AND lógico	&&
OR lógico	||
Condicional	? :
Asignación	= += -= *= /= %= &= ^= |= <<= = =

Tabla : Precedencia de operadores

Por ejemplo, la siguiente expresión produce un resultado diferente dependiendo de si se realiza la suma o división en primer lugar:

x + y / 100

Si no se indica explícitamente al compilador el orden en que se quiere que se realicen las operaciones, entonces el compilador decide basándose en la precedencia asignada a los operadores. Como el operador de división tiene mayor precedencia que el operador de suma el compilador evaluará *y/100* primero.

Así:

x + y / 100

Es equivalente a:

x + (y / 100)