

# Fundamentos de Programación

---

## Tema 1: Aspectos básicos de la programación

---

### Contenidos

1.- Introducción .....	2
2.- Objetivos de la programación .....	2
2.- Primeros aspectos léxicos .....	3
3.- Comentarios .....	5
4.- El almacenamiento de datos.....	6
5.- Operaciones con variables .....	15
6.- Conversiones entre variables .....	32
7.- Listas.....	37
8.- La estructura condicional .....	39

## Tema

# 1

## ASPECTOS BÁSICOS DE LA PROGRAMACIÓN

*En este tema realizaremos nuestros primeros programas. Como es lógico, no harán nada espectacular, pero servirán para conocer los primeros elementos básicos de los lenguajes de programación.*

### 1.- Introducción

Los primeros pasos en el mundo de la programación siempre son los más difíciles, ya que para hacer programas tenemos que adaptar de nuestra forma de pensar a la manera en la que “piensa” el ordenador. Se necesita, por tanto, un periodo de aprendizaje y adaptación en el que **la realización constante de programas es fundamental para conseguir resultados.**

**Todo el mundo está capacitado para programar ordenadores, pero hay personas que tienen más facilidad que otras. A quien le cuesta más trabajo, necesita hacer más ejercicios para interiorizar los conceptos fundamentales y saber aplicarlos.**

Por tanto, nuestro aprendizaje de la programación se realizará poco a poco a lo largo de varios temas, y en este nos centraremos en estudiar lo más simple: la estructura que tienen los programas, los elementos básicos que intervienen en ellos, y algunas acciones sencillas como la impresión de datos por pantalla y la realización de cálculos aritméticos. Como resultado, realizaremos programas de consola muy simples para resolver problemas matemáticos de 2º ESO.

### 2.- Objetivos de la programación

Lo primero que tenemos que tener en cuenta cuando vamos a hacer un programa de ordenador es que tenemos que cumplir tres objetivos:

**Un buen programa es correcto, eficiente y de fácil mantenimiento.**

Vamos a explicar qué significan estos objetivos:

- **Correcto:** El programa debe realizar, sin ningún fallo, la tarea que se le ha encomendado. Para eso, en los programas profesionales de gran tamaño suele haber fases previas a la programación donde se estudia muy bien qué es lo que se quiere hacer, para que no haya ningún problema.

- **Eficiente:** Aunque hoy día las prestaciones de los ordenadores son bastante elevadas para las pretensiones habituales de los programas, no hay que olvidar que sus recursos (CPU, Memoria...) son limitados. Un buen programa es aquel que consume únicamente los recursos que necesita. Como veremos, hacer un programa eficiente es muchísimo más difícil que uno que no lo sea y haga lo mismo. El resultado se notará durante la ejecución de ambos programas.
- **Fácil mantenimiento:** Cuando un programa se ha terminado comienza su mantenimiento, que es la fase en la que se solucionan errores no encontrados durante el desarrollo o se le realizan mejoras y ampliaciones. La fase de mantenimiento es muy importante, ya que suele ocupar el 75% de la vida del programa.

Cuando un programa tiene “fácil mantenimiento” es sencillo ampliarlo y añadirle cosas nuevas. Cuando un programa tiene “mal mantenimiento” su estructura interna es muy liosa y cualquier modificación que nos pidan puede ser muy difícil de hacer, o puede llevar a que metamos errores que antes no estaban.

Los programas con fácil mantenimiento son más difíciles de hacer, porque requiere “pensar antes de ponerse a programar”, y a la larga son mucho mejores. Los programas con mal mantenimiento son más rápidos y fáciles al principio, pero luego a la larga dan muchísimos problemas.

## 2.- Primeros aspectos léxicos

Comenzaremos abriendo el IDE NetBeans, y tras crear un nuevo proyecto, crearemos un programa llamado **Programa.java** con el siguiente código fuente:

```

1  import java.lang.*;
2
3  /* Este programa imprime Hola mundo en la pantalla
4   (c) 2020 Juan Diego
5   */
6  public class Programa {
7      public static void main(String[] args) {
8          System.out.println("Hola mundo");
9      }
10 }
```

Vamos a aprovechar este sencillo programa para estudiar las primeras características del lenguaje Java:

- Java es un lenguaje sensible a mayúsculas. Esto significa que por ejemplo “Main”, es una palabra diferente de “main”, o que “String” es una palabra distinta de “string”.

Por ejemplo, si en el código fuente cambiamos “String” por “string” veremos que el IDE nos avisa de que hemos cometido un error de escritura:

```
4 public class Programa {
5     public static void main(string[] args) {
6         System.out.println("Hola mundo");
7     }
```

- Todas las líneas de nuestro programa deben terminar en punto y coma, excepto aquellas que abren o cierran una llave.

Por ejemplo, si queremos que la frase “Hola mundo” aparezca tres veces en la pantalla escribiremos lo siguiente:

```
4 public class Programa {
5     public static void main(String[] args) {
6         System.out.println("Hola mundo");
7         System.out.println("Hola mundo");
8         System.out.println("Hola mundo");
9     }
10 }
```

- Como puede deducirse de lo anterior, el programa se ejecuta de arriba hacia abajo, comenzando su ejecución en la primera línea que hay tras abrir el main, y finalizándola al llegar a su llave de cierre.

```
4 public class Programa {
5     public static void main(String[] args) {
6         System.out.println("Hola mundo");
7         System.out.println("Hola mundo");
8         System.out.println("Hola mundo");
9     }
10 }
```

Instrucciones del programa

- El nombre del programa aparece tras las palabras “public class”, y de hecho, el archivo debe llamarse exactamente igual que él y terminar con la extensión .java.

```
Programa.java
1 /* Este programa imprime Hola mundo en la pantalla
2 (c) 2020 Juan Diego
3 */
4 public class Programa {
5     public static void main(String[] args) {
6         System.out.println("Hola mundo");
7         System.out.println("Hola mundo");
8         System.out.println("Hola mundo");
9     }
10 }
```

- Resumiendo todo lo anterior, podemos decir que la estructura mínima que tendrán todos los programas Java será la siguiente:

```
import java.lang.*;

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

Nombre del programa  
(y del archivo)

Instrucciones del programa

**Ejercicio 1:** Haz un programa Java llamado Datos\_Juan, Datos\_Ana... según tu nombre, que muestre por pantalla tu ficha personal: dni, nombre, apellidos, fecha de nacimiento, edad, dirección, localidad, teléfono y email mostrando cada dato en una línea diferente. Compila y ejecútalo para comprobar que funciona.

### 3.- Comentarios

Los **comentarios** son frases en formato libre que podemos escribir en cualquier lugar del programa, y que ayudan a explicar lo que hace el código fuente. Son muy útiles cuando tenemos que retomar un programa que hicimos hace tiempo, cuyos detalles se nos han olvidado.

Poner comentarios siempre es una buena práctica, y en las empresas se hace imprescindible, ya que alguien puede comenzar un programa y un compañero tener que modificarlo más adelante. Así, los comentarios le ayudarán para comprender lo que hizo la persona que lo programó.

Cuando compilamos el programa, lo primero que hace el compilador es retirar todos los comentarios. Por tanto, podemos poner sin miedo todos los comentarios que queramos, ya que no tendrán ningún efecto sobre el producto final.

Java incorpora los dos tipos de comentarios comunes a casi todos los lenguajes de alto nivel:

- Comentario de línea: Es un texto libre que se extiende hasta el final de su renglón. Para ponerlo escribimos dos barras `//` y todo el texto que va a continuación en la misma línea se considera comentario:

```
4 public class Programa {
5     public static void main(String[] args) {
6         // La siguiente línea imprime Hola mundo en la pantalla
7         System.out.println("Hola mundo");
8     }
9 }
```

- Comentario de bloque: Es un comentario que abarca varias líneas. Comienza con los símbolos `/*` y termina con los símbolos `*/`

```

4   public class Programa {
5       public static void main(String[] args) {
6           /* La siguiente línea
7              imprime Hola mundo
8              en la pantalla */
9           System.out.println("Hola mundo");
10      }
11  }

```

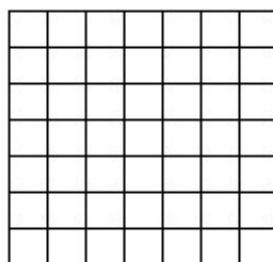
**Ejercicio 2 :** En el ejercicio 1 coloca un comentario de bloque al principio explicando lo que hace el programa, el nombre del autor, el número de versión del programa y la fecha en que se programó. Añade comentarios de línea que consideres apropiados para explicar lo que va haciendo el programa.

## 4.- El almacenamiento de datos

El objetivo de todo programa es dar solución a la tarea planteada por una persona. Como es lógico, en el problema que se pretende resolver aparecerán datos de muy diverso tipo (números, letras, frases, etc) que el ordenador deberá procesar. En los siguientes apartados vamos a ver cómo hacer que un ordenador trabaje con datos en un programa.

### 4.1.- Variables

Todos los datos que maneja el ordenador se guardan en su memoria RAM, que funciona como si estuviese formada por una serie de casillas o celdas de igual tamaño donde se guardará la información<sup>1</sup>. La memoria está lista llenarse con los datos que vayamos a usar en nuestro programa.



*Esquema de la estructura de la memoria*

En todos los lenguajes de programación existe el concepto de **variable**:

<sup>1</sup> Como ya sabemos, el ordenador solo trabaja con información binaria, y por tanto, todo lo que se guarde en la memoria estará expresado en el sistema binario. Para simplificar los apuntes, no escribiremos en binario el contenido de la memoria, y daremos por supuesto que todo en ella está en ese sistema.

**Una variable es una zona de la memoria que posee un nombre y en la que se almacena un dato**

Es decir, una variable es una zona de la memoria (que puede ocupar varias casillas consecutivas) en la que se guarda un dato que haga falta en el programa. La cantidad de casillas que ocupa la variable dependerá del tamaño del dato que se va a guardar. Por ejemplo, almacenar la frase “hola” ocupará menos casillas que almacenar la frase “hola mundo”.

Veamos ejemplos de variables que podríamos encontrar en nuestro programa:

- Si en un programa necesitamos trabajar con el sueldo de una persona, podemos hacer una variable que se llame **sueldo**, y asignarle el valor de dicho sueldo, por ejemplo, **1200**
- Si esa misma persona tiene una dirección y el programa necesita trabajar con ella, podemos hacer una variable llamada **dirección** con su valor, por ejemplo, **C/ Parque Manuel S/N**
- Si en el programa también se necesita trabajar con la altura de la persona, podemos crear una variable en la memoria llamada **altura** y guardar su dato, por ejemplo **1.72**
- Si en el programa se necesita saber si la persona es un becario, podemos hacer una variable llamada **becario** y guardar en ella el valor **si**.

Las variables no se colocan de manera ordenada en la memoria, sino que es el **sistema operativo** quien decide dónde se colocarán, según el tamaño que ocupan. Por ejemplo, la disposición en la memoria RAM de las variables creadas anteriormente en los ejemplos podría ser así:

								si	
								becario	
		C/ Parque Manuel S/N							
		dirección							
					1.72				
					altura				
	1200								
	sueldo								

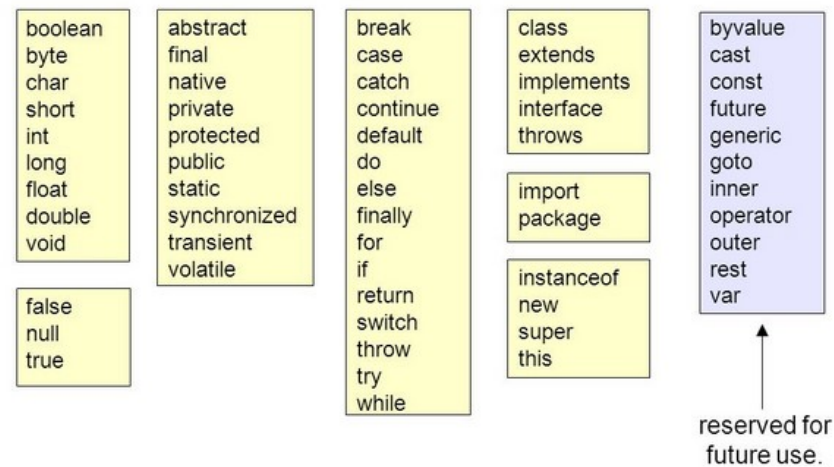
Con respecto a los nombres que ponemos a las variables, son libres y podemos utilizar cualquier nombre que queramos, pero tienen que cumplir las siguientes condiciones:

- El nombre tiene que empezar por una letra, un subrayado o el símbolo del dólar.
- No se pueden poner espacios en el nombre de una variable ni símbolos extraños<sup>2</sup>.
- El nombre no puede superar los 64 caracteres.
- Hay una serie de palabras llamadas “palabras reservadas” que tienen significado propio y, por tanto, no se pueden usar como nombres de variables:

---

<sup>2</sup> Java admite el uso de la ñ y los acentos. Pero para que el compilador los reconozca, debemos configurar nuestro editor de texto para que guarde en el sistema de codificación **ANSI** en Windows, o **UTF-8** en Linux.

## Reserved Words in Java



El nombre de una variable tiene que tener relación con el contenido que almacena. Por ejemplo, si en un programa trabajamos con el nombre de un cliente, para guardarlo podemos usar una variable llamada NombreCliente o bien, Nombre\_Cliente. También podríamos ponerle a esa variable el nombre Aj23Qm3, pero no sería apropiado porque no refleja bien lo que guarda.

**Ejercicio 3 :** Indica qué nombre de variable usarías para guardar los siguientes datos:

- El número de páginas de un libro.
- Si un alumno es repetidor o no.
- La suma de los cuatro lados de un rectángulo.
- La letra de la puerta de una casa.
- La suma de los primeros 500 números primos elevados al cuadrado.

**Ejercicio 4 :** Indica cuales de los siguientes nombres de variables son incorrectos:

- |                       |                      |               |                |
|-----------------------|----------------------|---------------|----------------|
| a) Hidrógeno          | b) mensajeTexto      | c) \$variable | d) 1erSueldo   |
| f) Tamaño_de_la_lista | f) Número de alumnos | g) P          | h) .porcentaje |

### **4.2.- Los tipos de datos**

Los datos que intervienen en un programa tienen naturaleza muy diversa: números, frases, texto... y por tanto, las operaciones que se podrán hacer con ellos serán distintas según su tipo. Por ejemplo, podemos multiplicar dos números, pero no tiene sentido querer multiplicar dos textos. O sea: lo que podemos hacer con las variables dependerá del tipo de dato que almacenan.

**El tipo de dato de una variable indica al ordenador el tipo de información que guarda**

En un programa Java, cuando creamos una variable debemos indicar cuál es su tipo de dato. Así, el ordenador sabrá trabajar correctamente con ella y en qué operaciones puede intervenir.



En Java hay dos grandes categorías de tipos de datos: los **tipos básicos** y los **tipos referencia**.

- Los tipos básicos son tipos de datos sencillos que se encuentran predefinidos:

Nombre	Descripción	Tamaño	Rango
byte	Byte con signo	1 byte	[-128, 127]
short	Entero corto	2 bytes	[-32768, 32767]
int	Entero	4 bytes	[-2147483648, 2147483647]
long	Entero largo	8 bytes	[-9223372036854775808, 9223372036854775807]
float	Real de simple precisión	4 bytes	[3.4e-038, 3.4e+038]
double	Real de doble precisión	8 bytes	[-1.7e-308, 1.7e+308]
char	Un carácter Unicode	2 bytes	[0, 65536] caracteres multilenguaje
boolean	Valor lógico si/no	1 byte	true, false

- Los tipos referencia son tipos más complicados que estudiaremos en profundidad en el próximo tema. En este tema, el único tipo referencia que vamos a utilizar es:

Nombre	Descripción	Tamaño	Rango
String	Texto genérico	Variable	Cadena de caracteres de longitud variable

La elección del mejor tipo de dato para una variable depende del programador, y normalmente habrá varias alternativas. Veamos algunos ejemplos donde necesitamos una variable y tenemos que elegir su tipo de dato:

- 1) Estamos haciendo un programa en el que se debe almacenar el número de curso de un alumno. Para ello, necesitaremos una variable que pueda guardar números enteros<sup>3</sup>. Si miramos en la tabla anterior hay tres tipos posibles: byte, short o int. Como el número de curso es un valor pequeño que nunca va a sobrepasar el valor 127, lo más eficiente sería usar el tipo byte. Sin embargo en Java, por costumbre el tipo que siempre se usa para guardar un número entero, sea cual sea su tamaño, es int. Así pues, lo más eficiente sería utilizar una variable llamada númeroCurso de tipo byte, aunque en la práctica todo el mundo la pondría de tipo int.
- 2) Un programa necesita conocer la velocidad con la que se mueve un vehículo expresada en kilómetros por hora. Como este dato puede tener decimales, para guardarlo usaremos una variable de tipo real, que podría ser de tipo float o double. En Java lo más frecuente es usar el tipo double para estos datos. Por tanto, usaríamos una variable double llamada VelocidadKm\_h
- 3) En un programa se trabaja con la dirección de una persona. Como este dato el dato que tenemos que guardar en memoria es una frase, lo haremos con una variable que almacene texto. En este caso la única posibilidad es usar una variable String. Por tanto, usaríamos una variable llamada dirección cuyo tipo de dato sería String.

---

<sup>3</sup> Recordamos que los números enteros son aquellos que no tienen cifras decimales, tanto positivos como negativos.  $\mathbb{Z} = \{ \dots, -3, -2, -1, 0, 1, 2, 3, \dots \}$

- 4) Nuestro programa necesita almacenar la letra del DNI de una persona. Al tratarse de una sola letra podemos usar una variable de tipo char, pero también podríamos utilizar una variable de texto general, es decir, un String. En la práctica son válidas las dos opciones.

Utilizaremos una variable llamada letraDNI cuyo texto puede ser char o String. El uso que tengamos que hacer de esa variable en el programa es lo que nos haría decantarnos por una u otra opción.

- 5) Nuestro programa necesita trabajar con el estado civil de una persona. En principio podríamos pensar en una variable que guarde texto, que almacene los valores “soltero” o “casado”.

Sin embargo, en Java tenemos un tipo especial mucho más adecuado para los casos en los que debemos guardar un “sí” o un “no”. Es mucho mejor hacer una variable llamada “personaCasada” que guarde el valor “sí” en caso de que la persona esté casada y el valor “no” si está soltera. El tipo de dato de esta variable será el boolean. El valor “sí” se representa con la palabra “true”, y el valor “no” con “false”.

Por tanto, usaríamos una variable de tipo boolean llamada personaCasada, que en caso de valer true indicará que la persona está casada, y si guarda false, soltera. Observa como el nombre de la variable nos sugiere que un true significa que la persona está casada.

- 6) Necesitamos guardar la fracción de alumnos aprobados en un curso, como por ejemplo 20/27. Si miramos la lista de tipos que hemos visto veremos que no hay ninguno que la represente exactamente. Incluso se nos podría ocurrir guardarla como si fuese un texto, pero esto sería una mala idea porque entonces no podríamos hacer operaciones matemáticas con ella.

La solución óptima (por ahora) consiste en utilizar dos variables, una para almacenar el numerador y otra para el denominador. Como en este ejemplo el numerador y el denominador hacen referencia a alumnos, que son valores sin decimales, el tipo adecuado para ellas será el int.

**Ejercicio 5 :** Escribe el tipo de dato que pondrías a las variables del ejercicio 3.

### **4.3.- Las variables en el lenguaje Java**

Ha llegado el momento de ver cómo se puede crear una variable en un programa Java. Para ello dentro del main se escribe una línea en la que aparece el tipo de dato, el nombre de la variable y el valor inicial que contendrá, tal y como se indica en el siguiente ejemplo:

int edad = 20;

*tipo de dato*

*nombre de la variable*

*valor inicial*

El siguiente programa, crea una variable llamada edad con valor inicial 20:

```
import java.lang.*;
public class EjemploVariable{
    public static void main(String[] args){
        int edad=20;
    }
}
```

Si analizamos la línea que crea una variable, vemos que podemos dividirla en dos partes. La parte a la izquierda del igual se denomina **declaración**, y la que hay a la derecha **inicialización**.

*declaración*  
 {  
 int edad = 20;  
 }  
*inicialización*

En un programa podemos crear todas las variables que necesitemos, procurando no abusar de variables innecesarias, ya que consumen memoria. Una vez que una variable ha sido creada, puede utilizarse en operaciones, cálculos, etc., sin necesidad de volverla a crear de nuevo.

## **4.2.- Los literales**

Cuando creamos una variable tenemos que darle un valor inicial, que escribimos “a mano”. La forma de escribir ese valor dependerá del tipo de dato de la variable, según estas normas:

- **int, short, byte:** Escribimos el número directamente, teniendo en cuenta que debe ajustarse al rango definido por cada tipo de dato.

```
int edad=20;
byte numeroHijos=0;
short sueldo=900;
long contraseña=15126212;
```

- **long:** Se escribe el número directamente como antes, pero los números muy grandes que sobrepasan el tipo int deben terminar con una L (mayúscula o minúscula).

```
long contraseña=15126212;
long likesFacebook=1529621262341225L;
```

- **double:** Escribimos un número usando el punto . para separar la parte entera de la decimal. También se puede escribir en notación científica usando la letra e para indicar  $10^x$

```
double peso=68.5;
double temperatura=0.3655e2;
```

- **float:** Es como el double, pero terminando el número en una F (mayúscula o minúscula) y teniendo la precaución de ajustarse al rango.

```
float altura=1.68f;  
float imc=0.2427e2F;
```

- **char:** Se puede escribir un carácter entre comillas simples, o un número entre 0 y 65536, que sería traducido internamente al carácter cuyo código Unicode sea dicho número.

```
char letraDNI='Z';  
char codigoMoneda=8364;
```

- **String:** Se escribe una cadena de texto entre comillas dobles.

```
String nombre="Francisco Garcia López";
```

- **boolean:** Es escriben las palabras true o false según lo que se vaya a guardar en la variable.

```
boolean soltero=true;  
boolean estudiante=false;
```

Todos estos valores que escribimos “a mano” cuando inicializamos las variables se denominan **literales**. En estos apuntes hemos visto solo las formas más sencillas de escribirlos, pero hay muchas más, que permiten por ejemplo, escribir números en un sistema distinto al decimal, números que llevan las cifras separadas en miles o caracteres especiales (salto de línea, tabulación...)

**Ejercicio 6 :** Haz un programa que cree todas las variables del ejercicio 3, según el tipo de dato que le indicaste en el ejercicio 5.

#### **4.3.- Mostrar variables por pantalla**

Si compilamos y ejecutamos el programa del ejercicio 5, veremos que funciona pero no hace nada más. Esto es bastante lógico ya que solo crea un montón de variables y el programa termina sin hacer nada más. Si queremos mostrar el contenido de las variables por pantalla, deberemos hacerlo nosotros mediante la orden que ya conocemos: **System.out.println**

Por ejemplo, el siguiente programa crea una variable llamada edad, con valor inicial 20, y la muestra por pantalla:

```
1  import java.lang.*;  
2  public class EjemploVariable{  
3      public static void main(String[] args){  
4          int edad=20;  
5          System.out.println(edad) ;  
6      }  
7  }
```

Si lo ejecutamos, veremos que la salida se limita a mostrar el valor de la variable por la pantalla. Lo normal es que queramos que aparezca un texto previo que nos informe del valor de la variable. En ese caso, podemos usar el signo + para concatenar texto escrito entre comillas dobles con una variable.

```
1 import java.lang.*;
2 public class EjemploVariable{
3     public static void main(String[] args){
4         int edad=20;
5         System.out.println("Tu edad es: "+edad);
6     }
7 }
```

Más aún, podemos seguir usando el signo + para seguir concatenando texto a la variable. Incluso podemos concatenar nuevas variables a continuación si lo deseamos.

```
1 import java.lang.*;
2 public class EjemploVariable{
3     public static void main(String[] args){
4         int edad=20;
5         System.out.println("Tu edad es: "+edad+" años");
6     }
7 }
```

**Ejercicio 7 :** Repite el ejercicio 1 usando variables para almacenar los datos. ¿Qué ventajas o inconvenientes observas con respecto a dicho ejercicio?

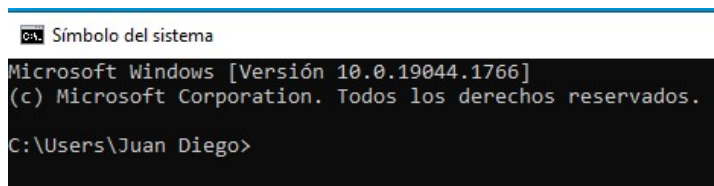
**Ejercicio 8 :** Haz un programa que muestre el siguiente texto por pantalla, teniendo en cuenta que los datos que aparecen subrayados deben encontrarse guardados en variables.

*El viernes día 26 de mayo tendrá lugar la reunión de evaluación del 1º curso de programación de aplicaciones web en el IES Hlanz de Granada. Los profesores calificarán al alumnado y se prevé que a las 20:30 horas las notas estén publicadas en el tablón de anuncios del centro.*

**Ejercicio 9 :** Realiza una copia del archivo del ejercicio anterior. Sobre la copia, haz las modificaciones oportunas para que el programa nos informe de que la reunión de evaluación de 1º de ESO será el lunes 20 de junio y que las notas se publicarán a las 13:45.

#### **4.4.- Pedir datos por teclado al usuario**

Por defecto, cuando empezamos a hacer un programa siempre es **de consola**. Esto significa que su apariencia es texto (sin nada gráfico) y el programa funcionará mediante comandos en una ventana de terminal o símbolo del sistema operativo.



Por tanto, en nuestras aplicaciones inicialmente no hay ventanas ni elementos gráficos con los que interactuar, sino que todo se hace en modo texto. Entonces, ¿cómo podemos hacer para que el usuario pueda introducir datos por teclado?

A continuación vamos a escribir la orden que sirve para que el ordenador pida un dato al usuario. Cuando se lanza una de estas órdenes, el programa se detiene hasta que el usuario escribe un dato por teclado y pulsa Intro. Según el tipo de dato que queramos leer por teclado, habrá una orden diferente.

Tipo de dato que queremos leer	Orden que usaremos
int	<code>new Scanner(System.in).nextInt()</code>
long	<code>new Scanner(System.in).nextLong()</code>
double	<code>new Scanner(System.in).nextDouble()</code>
boolean	<code>new Scanner(System.in).nextBoolean()</code>
String	<code>new Scanner(System.in).nextLine()</code>

Estas órdenes no están disponibles cuando empezamos a hacer un programa (porque podría haber programas gráficos en los que no fueran necesarias, y por tanto, no harían falta en ellos). Para poder usar estas órdenes debemos escribir al inicio del programa:

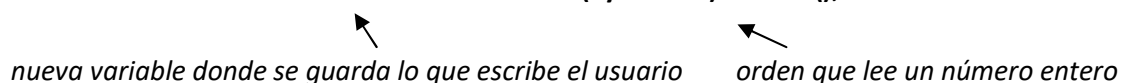
```
2  import java.util.*;
```

Esta línea se lee diciendo que **importamos el paquete java.util**. Como veremos en el tema siguiente, los paquetes contienen órdenes y otras cosas que podemos añadir a nuestros programas. Por ejemplo, las órdenes de la tabla anterior están dentro del paquete java.util, y para poder usarlas en el programa, deberemos importar dicho paquete.

Una vez importado el paquete java.util, podremos utilizar las órdenes de la tabla para leer datos por teclado. Para usarlas hay que tener en cuenta una cosa muy importante: cuando el usuario introduce un dato por teclado, será necesario crear una nueva variable para guardar ese dato.

O sea, si por ejemplo quiero que el usuario escriba un número entero con su edad, haremos esto:

```
int edad = new Scanner(System.in).nextInt();
```



*nueva variable donde se guarda lo que escribe el usuario*      *orden que lee un número entero*

Por tanto, un programa para leer el nombre del usuario por pantalla y hacer que el ordenador le salude, será:

```

1  import java.lang.*;
2  import java.util.*;
3  /* Este programa pregunta al usuario su nombre
4  por teclado y lo imprime en pantalla
5  (c) 2020 Juan Diego
6  */
7  public class Programa {
8      public static void main(String[] args) {
9          System.out.println("Escribe tu nombre:");
10         String nombre=new Scanner(System.in).nextLine();
11         System.out.println("Hola "+nombre);
12     }
13 }

```

**Ejercicio 10 :** Repite el ejercicio 8, pero haciendo que el usuario introduzca por teclado los datos que aparecen subrayados.

**Ejercicio 11 :** Haz un programa que pregunte al usuario su nombre, su primer apellido y su segundo apellido (cada uno será una pregunta independiente). El ordenador mostrará el nombre y apellidos del usuario en formato **apellido1 apellido2, nombre**

## 5.- Operaciones con variables

Una vez que ya sabemos crear variables en un programa, vamos a ver qué cosas podemos hacer con ellas. De esta forma, conseguiremos realizar acciones que procesen los datos de un programa y obtener la solución a un problema determinado.

### 5.1.- Asignación

Como ya sabemos, una variable es una zona de la memoria que guarda un dato, pero podemos preguntarnos, ¿ese dato se puede cambiar por otro? ¿o va a ser siempre el mismo?

En los lenguajes de programación se le puso el nombre de “variable” al concepto que hemos estudiado porque su valor puede cambiar en cualquier momento. Es decir, cuando nosotros creamos una variable le damos un valor inicial, pero posteriormente lo podemos cambiar por otro. La asignación es la operación que permite hacer esto.

- Se llama asignación a la operación que consiste en cambiar el valor de una variable.
- Se realiza con el signo =

Supongamos que ya tenemos hecha una variable llamada sueldo y queremos cambiar su valor. Lo haremos así:

suelto = 1220;  
                  ↖                  ↗  
nombre de la variable    nuevo valor

Como vemos, para cambiar el valor de la variable simplemente se utiliza el signo igual para indicar cuál será su nuevo valor. Ya no es necesario indicar el tipo de dato, porque se lo indicamos al ordenador cuando creamos la variable.

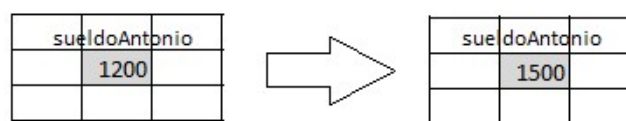
En el siguiente ejemplo creamos una variable llamada `sueltoAntonio` que guarda el valor 1200 y lo mostramos por pantalla. A continuación, le cambiamos el sueldo y lo volvemos a mostrar:

```
1  import java.lang.*;
2  public class EjemploVariable{
3      public static void main(String[] args){
4          int sueldoAntonio = 1200;
5          System.out.println("Antonio gana "+sueldoAntonio+" euros");
6          sueldoAntonio = 1500; // Asignación
7          System.out.println("Antonio gana "+sueldoAntonio+" euros");
8      }
9  }
```

Como puede verse, el programa se ejecuta de arriba abajo y la primera vez se muestra el sueldo de 1200 y la segunda vez, el sueldo de 1500.

```
run:
Antonio gana 1200 euros
Antonio gana 1500 euros
BUILD SUCCESSFUL (total time: 0 seconds)
```

En la memoria RAM lo que ha sucedido es que la casilla donde se guardaba el valor de 1200 ha sido actualizada y ahora pasa a guardar un valor de 1500.



El dato que ponemos en una asignación puede ser otra variable del mismo tipo. Por ejemplo, podemos tener los sueldos de dos empleados y hacer que uno sea igual al otro:



```

1 public class Programa {
2     public static void main(String[] args) {
3         int sueldoAntonio=1200;
4         int sueldoPedro=900;
5         System.out.println("Antonio gana "+sueldoAntonio+" euros");
6         System.out.println("Pedro gana "+sueldoPedro+" euros");
7         // la siguiente línea hace una asignación
8         sueldoPedro=sueldoAntonio;
9         System.out.println("Antonio gana "+sueldoAntonio+" euros");
10        System.out.println("Pedro gana "+sueldoPedro+" euros");
11    }
12 }

```

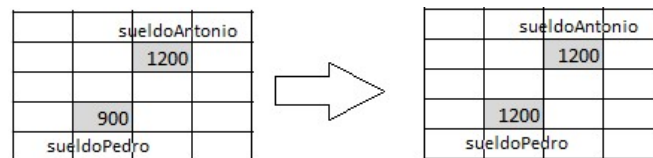
En este código creamos los sueldos de dos empleados y los mostramos por pantalla. En la línea 8 se produce una asignación que actualiza la variable sueldoPedro para guardar en ella lo que haya en la variable sueldoAntonio. Como se ve en la imagen, la variable cambia su contenido:

```

run:
Antonio gana 1200 euros
Pedro gana 900 euros
Antonio gana 1200 euros
Pedro gana 1200 euros
BUILD SUCCESSFUL (total time: 0 seconds)

```

En la memoria RAM lo que sucede es que la casilla de memoria correspondiente a sueldoPedro cambia su valor y copia en ella lo que esté dentro de la casilla sueldoAntonio.



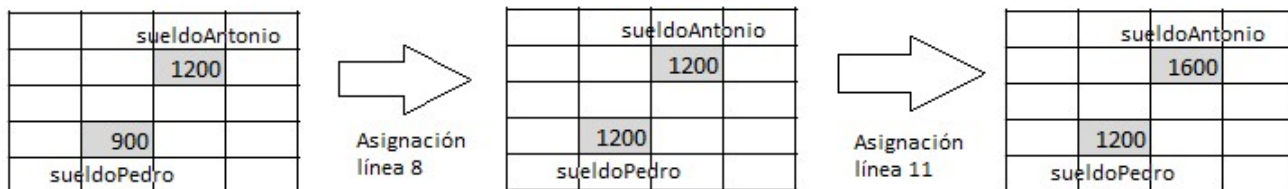
¿Y qué sucedería si a continuación del código anterior actualizásemos la variable sueldoAntonio y le asignásemos el valor 1600? ¿Cuánto valdrían los sueldos de nuestros empleados?

```

1 public class Programa {
2     public static void main(String[] args) {
3         int sueldoAntonio=1200;
4         int sueldoPedro=900;
5         System.out.println("Antonio gana "+sueldoAntonio+" euros");
6         System.out.println("Pedro gana "+sueldoPedro+" euros");
7         sueldoPedro=sueldoAntonio;
8         System.out.println("Antonio gana "+sueldoAntonio+" euros");
9         System.out.println("Pedro gana "+sueldoPedro+" euros");
10        sueldoAntonio=1600; // asignación
11        System.out.println("Antonio gana "+sueldoAntonio+" euros");
12        System.out.println("Pedro gana "+sueldoPedro+" euros");
13    }
14 }

```

Como en la memoria RAM cada variable ocupa su propio espacio de memoria y es independiente de las demás variables, sueldoPedro quedará como estaba en su lugar de la memoria y solo se actualizará la zona de memoria de la variable sueldoAntonio.



Por último, indicamos que podemos hacer variables que sean de solo lectura y no se puedan modificar. Este tipo de variables se llaman **constantes**, y su valor nunca cambia. Son útiles para almacenar datos que no pueden ser modificados, como por ejemplo, el valor del número PI.

Para crear una constante, se pone la palabra **final** en el momento de crearla, así:

final int edad = 20;

*indica que es constante*    *tipo de dato*    *nombre de la constante*    *valor*

## 5.2.- Operaciones numéricas

Lo mínimo que podemos esperar de un programa es que realice las operaciones aritméticas básicas que aprendemos en el colegio. De esta forma, si tenemos dos variables numéricas<sup>4</sup> cualesquiera, podemos hacer las operaciones elementales con los siguientes símbolos:

+	Suma
-	Resta
*	Multiplicación
/	<ul style="list-style-type: none"> <li>• Cociente (cuando es int/int)</li> <li>• División con decimales (en otro caso)</li> </ul>
%	Resto de la división

Una cosa **importantísima** es que cuando hacemos una operación, **obtenemos un resultado**. Como ya hemos dicho varias veces, la única forma de que el ordenador pueda trabajar con un dato es tenerlo guardado en una variable. Por tanto, **cuando hagamos una operación, deberemos guardar su resultado en una variable**, que podrá ser nueva, o ya existente. Si no hacemos esto, el ordenador no tendrá constancia del resultado de la operación.

<sup>4</sup> También se pueden realizar se pueden realizar las operaciones aritméticas con variables de tipo **char**, cuyo valor es transformado automáticamente en **int** por medio de su código Unicode.

### 5.2.1 – Sumas, restas y multiplicaciones

Supongamos que tenemos dos variables llamadas aula1 y aula2 que guardan la cantidad de alumnos que hay en dos aulas, y queremos obtener el número total de alumnos. Este nuevo dato es diferente al número de alumnos de cada aula, y por tanto, deberemos crear una variable diferente en la que se almacene el resultado de sumar el número de alumnos de cada aula. Se haría así:

`int total = aula1 + aula2;`

*nueva variable para guardar el resultado      alumnos del aula1      alumnos del aula2*

El programa completo sería así:

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int aula1=20;
5          int aula2=17;
6          int total=aula1+aula2; // Suma
7          System.out.println("Hay "+total+" alumnos");
8      }
9  }
```

Si lo compilamos y ejecutamos, veremos que se muestra que hay 37 alumnos, como es de esperar.

En este ejemplo hemos visto que la suma ha producido un resultado que guardamos en una nueva variable que ha sido creada para la ocasión.

Sin embargo, también podemos guardar el resultado de la suma en alguna variable que ya tengamos hecha en el programa. Usando la asignación, cambiaremos el valor de la variable para que coincida con la suma que queremos realizar.

Por ejemplo, podemos tener la variable “total” ya creada de antemano y guardar en ella el resultado de la suma, tal y como se muestra en este programa:

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int total=0; // aquí vamos a guardar el resultado
5          int aula1=20;
6          int aula2=17;
7          total=aula1+aula2; // Asignación para guardar la suma
8          System.out.println("Hay "+total+" alumnos");
9      }
10 }
```

Por último, aunque suena un poco raro al principio, es muy frecuente utilizar una de las variables que intervienen en la suma para guardar el resultado. Imaginemos que tenemos alumnos en el aula1, y que queremos añadirles los alumnos que hay en el aula2. Realizaremos una suma y guardaremos el resultado dentro de la variable aula1.

```

1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int aula1=20;
5          int aula2=17;
6          aula1=aula1+aula2; // guardamos la suma de alumnos en aula1
7          System.out.println("Hay "+aula1+" alumnos");
8      }
9  }

```

La asignación “aula1 = aula1 + aula2” significa que realizamos la suma de los datos que hay dentro de las variables aula1 y aula2, y se guarda dicho resultado en la variable aula1, que perderá por tanto, su valor inicial tras la operación.

Las operaciones de suma, resta y multiplicación funcionan de la manera habitual a la que estamos acostumbrados. Además, podemos usar paréntesis para agrupar el orden en que se realicen las operaciones<sup>5</sup>.

```

int operacion = (5 * (4 + 2) - 8 * (25 - 6) * 4) + 3;
System.out.println(operacion);

```

### 5.2.2 – La división

Las operaciones de suma, resta y multiplicación funcionan de la manera habitual a la que estamos acostumbrados. Sin embargo, un caso especial es el signo /, con el que podemos hacer dos operaciones matemáticas diferentes:

- **Cociente:** Es el resultado que obtenemos cuando hacemos una división sin sacar decimales
- **División real:** Es el resultado de una división cuando sacamos cifras decimales

$$\begin{array}{r}
 25 \overline{) 4} \\
 \underline{1} \phantom{0} \\
 6 \leftarrow \text{cociente}
 \end{array}$$

$$\begin{array}{r}
 25 \overline{) 4} \\
 \underline{0} \phantom{0} \\
 6.25 \leftarrow \text{división real}
 \end{array}$$

<sup>5</sup> Si no se ponen paréntesis, se aplicará el orden de precedencia habitual que se usa en matemáticas

En Java se usa el mismo símbolo / para realizar ambas operaciones. Por tanto, ¿cómo se sabe cuándo / actúa como cociente, y cuando actúa como división real?

Todo dependerá del tipo de dato de los números que se estén dividiendo:

- Cuando los dos números que intervienen son enteros (int, short, byte, long), entonces se hace el cociente.
- Si alguno de los números que se dividen no es entero (float, double), entonces se hace la división real.

En el siguiente ejemplo calculamos la nota media de un alumno. Como las notas tienen decimales, el operador / actuará como división real y el resultado contendrá decimales.

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          double notal = 6.5;
5          double nota2 = 4.25;
6          double media = (notal+nota2)/2; // división real
7          System.out.println("La media es "+media);
8      }
9  }
```

En este otro ejemplo, tenemos 10 caramelos para repartir entre 4 niños y queremos saber cuántos le tocan a cada uno. En este caso, como los tipos de datos de los números que intervienen es int, el operador / actuará como cociente.

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int caramelos=10;
5          int niños=4;
6          int cantidad=caramelos/niños; // cociente
7          System.out.println("Cada niño recibe "+cantidad+" caramelos);
8      }
9  }
```

¿Qué hacer cuando queramos sacar decimales y los tipos de datos de los números sea entero? Por ejemplo, supongamos que hemos realizado tres exámenes que se han calificado como números del 0 al 10 sin decimales, y queremos obtener la media, que puede tener decimales.

El truco consiste en multiplicar por 1.0 el numerador o denominador cuando hacemos la división. De esa forma, se convertirá en double y el resultado tendrá decimales.

```

import java.lang.*;
public class Ejemplo{
    public static void main(String[] args){
        int nota1=5;
        int nota2=7;
        int nota2=8;
        double media = (1.0* (nota1+nota2+nota3))/3;
        System.out.println("La media es "+media);
    }
}

```

Lo que acabamos de ver sirve siempre que queremos sacar decimales en una división entera, pero en el caso concreto del ejemplo anterior podríamos haber dividido directamente entre 3.0, porque entonces el denominador sería double y el operador / actuaría como división real.

### 5.2.3 – El resto de la división

Una operación muy poco utilizada en las matemáticas de los colegios, pero que aparece muchísimo en la informática (en redes, compresión de datos, criptografía...) es calcular el resto de la división de dos números.

$$\begin{array}{r}
 25 \overline{) 4} \\
 \underline{1} \phantom{0} \\
 3
 \end{array}$$

Resto (%)  $\Rightarrow$  1  $\leftarrow$  6  $\leftarrow$  cociente (/)

Por ejemplo, si tenemos 10 caramelos y 4 niños y queremos saber los caramelos que tocan a cada niño, usaríamos el cociente. Si queremos ver cuántos caramelos han sobrado, calcularemos el resto con el símbolo %.

```

1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int caramelos=10;
5          int niños=4;
6          int cociente=caramelos/niños;
7          int resto=caramelos%niños;
8          System.out.println("Cada niño recibe "+cociente+" caramelos");
9          System.out.println("Sobran "+resto+" caramelos");
10     }
11 }

```

La salida del programa produce el resultado esperado:

```

run:
Cada niño recibe 2 caramelos
Sobran 2 caramelos
BUILD SUCCESSFUL (total time: 0 seconds)

```



### 5.2.3 – Incrementos y decrementos

Una acción muy habitual en todos los programas consiste en incrementar o decrementar el valor de una variable. Como ya vimos en un ejemplo, podemos hacer esto realizando una suma y guardando el resultado en la misma variable, así:

```
1 import java.lang.*;
2 public class Ejemplo{
3     public static void main(String[] args){
4         int aula1=20;
5         int aula2=17;
6         aula1=aula1+aula2; // guardamos la suma de alumnos en aula1
7         System.out.println("Hay "+aula1+" alumnos");
8     }
9 }
```

Sin embargo, en los lenguajes de programación hay símbolos específicos para realizar el incremento o decremento de forma más rápida:

+=	Incremento
-=	Decremento
++	Incremento unitario
--	Decremento unitario

Por ejemplo, tenemos una variable llamada sueldoAntonio y queremos incrementarla en 250. Podemos escribir esto:

sueldoAntonio += 250;

*variable que se va a incrementar*      *operador de incremento*      *cantidad*

Un caso especial que, como veremos durante el curso, aparece innumerables veces en todos los programas es el de incrementar en una unidad. Eso se consigue con el operador de incremento unitario. Por ejemplo, si tenemos una variable llamada ventas que guarda el número de ventas registradas en una tienda y queremos añadir una más, haremos esto:

ventas++;

*variable que se va a incrementar*      *operador de incremento unitario*

Tras la ejecución de la línea anterior, la variable automáticamente quedará incrementada. Aunque aquí hemos explicado solo los operadores de incremento, también están las versiones equivalentes para realizar decremento, como vemos en este ejemplo:

```
int sueldoAntonio=1200;
int articulos=40;
sueldoAntonio-=100; // disminuye el sueldo 100
articulos--;        // retira un artículo
```

**Ejercicio 12 :** Haz un programa que pida al usuario por teclado dos números con decimales: precioNormal y porcentajeRebaja. El programa mostrará por pantalla el precio normal y el rebajado, tras haber aplicado el porcentaje de descuento indicado en la variable porcentajeRebaja. La salida del programa deberá tener el siguiente formato:

Precio normal del artículo: ..... euros  
 Porcentaje de rebaja aplicado: ..... %  
 Descuento aplicado: ..... euros  
 Precio final del artículo: ..... euros

**Ejercicio 13 :** Haz un programa que pregunte al usuario tres números enteros y guárdalos en variables llamadas horas, minutos y segundos. El programa mostrará el número total de segundos que hay juntando las cantidades de tiempo expresadas en las tres variables.

**Ejercicio 14 :** Realiza un programa que pregunte al usuario por teclado un número de “euros” y el programa mostrará por pantalla el cambio de dicha cantidad a las siguientes monedas:

Dolares	1 € = 1.42 \$
Libras	100 € = 87.13 £
Yens	1 € = 113.86 ¥
Pesetas	1 € = 166.386 pts

**Ejercicio 15 :** Crea un programa que pregunte al usuario un número entero llamado gradosCentígrados. El programa deberá convertir dicha cantidad a grados Fahrenheit y mostrarla por pantalla. La fórmula para realizar la conversión es:

$$F = \frac{9}{5}C + 32$$

**Ejercicio 16 :** Un videoclub alquila la película "El padrino" a 3.5 €/día y "Odisea 2001" por 2.95 €/día. Haz un programa que muestre por pantalla cuánto dinero se necesita para alquilar dos días estas películas. Haz que también se muestre por pantalla si es suficiente un billete de 5€ para pagar dicho alquiler.

**Ejercicio 17 :** Considera la función  $f(x) = \frac{3\left(\frac{x}{2}\right)^3}{x^2 - x + 3}$ . Haz un programa que muestre por pantalla los valores de  $f(0)$ ,  $f(1)$  y  $f(-2)$

**Ejercicio 18 :** Un estudiante de universidad cobra 7€ por cada hora de sus clases particulares. Haz un programa que muestre por pantalla:

- ¿Cuánto ganará en un mes si tiene dos alumnos, a los que les da 2 y 3 clases semanales respectivamente.
- ¿Cuántas horas de clase debe dar al mes si quiere ganar 900€?



**Ejercicio 19 :** Una academia contrata al estudiante del ejercicio anterior y le ofrece un sueldo mensual de 200€ fijos más el 15€ mensuales por cada alumno. Supongamos que hay 30 alumnos apuntados en la academia, que van una hora todas las tardes de lunes a viernes y que cada uno paga 100€ al mes. Realiza un programa que calcule:

- a) ¿Cuánto dinero gana el estudiante que da las clases?
- b) ¿Cuánto dinero gana la academia al mes?

**Ejercicio 20 :** Un camión que tiene capacidad para 3 toneladas necesita 15 viajes para transportar una cantidad de arena. Realiza un programa que muestre por pantalla cuántos viajes necesitará un camión con capacidad de 5 toneladas

**Ejercicio 21 :** Un profesor hace un examen en el que la teoría vale un 40% de la nota y los problemas un 60%. En total hay 8 preguntas de teoría (todas valen lo mismo) y 4 problemas (todos valen lo mismo). Un alumno contesta correctamente 6 preguntas de teoría y hace bien 1 problema. Realiza un programa que muestre por pantalla la nota del alumno.

**Ejercicio 22 :** Un profesor realiza un test a sus alumnos. El test tiene 30 preguntas (todas las preguntas valen lo mismo) con cuatro opciones cada una. Se supone que tres preguntas falladas te quitan una bien. Las preguntas sin contestar no quitan puntos. Realiza un programa que calcule y muestre por pantalla la nota de un alumno que tiene 22 preguntas bien, 6 mal y 2 sin contestar.

### **5.3.- Comparación**

Imaginemos que tenemos dos variables del mismo tipo y queremos que el ordenador nos diga si guardan o no el mismo valor. Nosotros respondemos fácilmente a esta pregunta usando nuestros ojos. ¿Son iguales las variables de cada uno de estos ejemplos?

	-8
-8	

M	
	K

true	
	false

	8.25		
		10.8	

Como podemos ver, mirando el contenido fácilmente podemos saber si son iguales o no cada una de estas parejas de variables.

Es importante darse cuenta que la respuesta que damos a cada apartado es **si** o **no**, por lo que cuando comparamos si dos variables guardan el mismo dato, estamos haciendo una operación cuyo resultado es un boolean.

Como ya vimos, las operaciones aritméticas producen un resultado numérico. Sin embargo, la **comparación es una operación que produce un resultado boolean**, porque devuelve verdadero (true) cuando las variables guardan el mismo dato, y falso (false) en caso contrario. Por tanto, cuando hagamos una comparación, deberemos guardar su resultado en una nueva variable, tal y como hicimos con las operaciones que producían resultados numéricos.

La comparación es la operación que comprueba si el valor de una variable coincide con el de otra.

- La comparación produce un resultado de tipo boolean
- Se realiza con el signo ==
- Solo se puede realizar para tipos básicos (no sirve para tipos referencia)

Vamos a verlo en un ejemplo. Tenemos dos variables llamadas sueldoAntonio y sueldoPedro que guardan los sueldos de dos personas y queremos que el ordenador calcule si los sueldos son iguales o no. Escribiremos la siguiente línea:

```
boolean sueldosIguales= sueldoAntonio == sueldoPedro;
```

*nueva variable para guardar el resultado*      *sueldo del empleado1*      *sueldo del empleado2*

El programa completo para hacer esto sería:

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int sueldoAntonio=1200;
5          int sueldoPedro=1500;
6          boolean sueldosIguales = sueldoAntonio == sueldoPedro; // comparación
7          System.out.println("Sueldo de Antonio: "+sueldoAntonio);
8          System.out.println("Sueldo de Pedro: "+sueldoPedro);
9          System.out.println("¿Los sueldos son iguales? "+sueldosIguales);
10     }
11 }
```

Al compilar y ejecutar, veremos que aparecen los dos sueldos y luego el resultado de la comparación, mostrando el contenido de la variable sueldosIguales, que en este caso es false.

```
run:
Sueldo de Antonio: 1200
Sueldo de Pedro: 1500
¿Los sueldos son iguales? false
BUILD SUCCESSFUL (total time: 0 seconds)
```

En este momento, no podemos cambiar el último mensaje y poner en nuestro idioma “sí” o “no”. Más adelante veremos cómo poner un mensaje personalizado en lugar de “true” o “false”.

Existe otra forma de hacer la comparación, que es la inversa de la que hemos visto. En lugar de comparar si dos variables son iguales, también podemos comparar si dos variables son diferentes. En este caso, el operador que se utiliza es !=

Por ejemplo, si lo que queremos es comprobar si los dos empleados del ejemplo anterior tienen sueldo distinto, lo haríamos así:

```

1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int sueldoAntonio=1200;
5          int sueldoPedro=1500;
6          boolean sueldosDistintos = sueldoAntonio != sueldoPedro; // comparación
7          System.out.println("Sueldo de Antonio: "+sueldoAntonio);
8          System.out.println("Sueldo de Pedro: "+sueldoPedro);
9          System.out.println("¿Los sueldos son distintos? "+sueldosDistintos);
10     }
11 }

```

Si ejecutamos el programa, podremos ver como ahora la respuesta es justo la contraria de la que obtuvimos antes:

```

run:
Sueldo de Antonio: 1200
Sueldo de Pedro: 1500
¿Los sueldos son distintos? true
BUILD SUCCESSFUL (total time: 0 seconds)

```

Para finalizar, indicamos que podemos realizar la comparación entre variables de tipos básicos, pero el operador == no sirve para realizar comparación entre datos de tipo referencia. Por tanto, **no podemos comparar dos Strings con el operador ==**

```

3  String nombre1="Pepe";
4  String nombre2="Manuel";
5  // ERROR GRAVISIMO!!!
6  // Los String no se pueden comparar con ==
   boolean iguales=nombre1==nombre2;

```

 Error muy gordo!!

### 5.3.1 – Comparaciones numéricas

Cuando las variables guardan números, es muy frecuente preguntarnos si uno es mayor o menor que el otro. En este caso estamos haciendo una **comparación numérica**<sup>6</sup>, que producirá como resultado un boolean. Las comparaciones numéricas se realizan con los siguientes símbolos:

<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

La forma de realizarlas es similar a la comparación ordinaria. Como sucedía entonces, es muy importante recoger el resultado en una variable para que el ordenador tenga constancia de la operación.

<sup>6</sup> A las variables de tipo char también se les puede hacer esta comparación, porque internamente son convertidas en int por medio de su código Unicode.

Por ejemplo, supongamos que tenemos las variables `sueldoAntonio` y `sueldoPedro` de los ejemplos anteriores, y queremos saber si Antonio gana más que Pedro. Realizaríamos una comparación numérica de esta forma:

`boolean antonioGanaMas = sueldoAntonio > sueldoPedro;`

*nueva variable para guardar el resultado      sueldo de Antonio      sueldo de Pedro*

Aquí tenemos el programa donde realizaríamos todo el proceso:

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          int sueldoAntonio=1200;
5          int sueldoPedro=1500;
6          boolean antonioGanaMas = sueldoAntonio > sueldoPedro; // comparación
7          System.out.println("Sueldo de Antonio: "+sueldoAntonio);
8          System.out.println("Sueldo de Pedro: "+sueldoPedro);
9          System.out.println("¿Antonio gana más que Pedro? "+antonioGanaMas);
10     }
11 }
```

**Ejercicio 23 :** Realiza un programa que pregunte por teclado al usuario las tres notas de sus exámenes y nos muestre por pantalla si su media le sale aprobado.

**Ejercicio 24 :** Realiza un programa que pregunte al usuario la edad de dos personas. El programa mostrará si ambas son mayores de edad, y también si la primera persona es mayor que la segunda.

## **5.4.- Operaciones con variables booleanas**

El tipo de dato boolean guarda solamente dos tipos de valores, que son verdadero (true) y falso (false). Aunque parezca mentira, es posible realizar operaciones con ellos, que van a producir resultados también boolean. Estas operaciones, también se denominan **operaciones lógicas**, y fueron estudiadas en 1847 por el matemático inglés George Boole.



En la época de Boole, nadie comprendía qué utilidad práctica podía tener estudiar las operaciones que se pudieran realizar con el “verdadero” y el “falso”, y fue un incompromiso. Aún así, realizó su estudio sin ser consciente de lo importante que sería un siglo después.

Las variables boolean y sus operaciones son la base de todos los razonamientos que hacen los ordenadores. Gracias a ellas, los ordenadores pueden “pensar”.

Las tres operaciones booleanas fundamentales, que es necesario conocer porque aparecen constantemente en los programas, son:

#### 5.4.1.- La operación Y

Esta operación se utiliza en programación cuando queremos detectar si dos cosas pasan a la vez.

Por ejemplo, supongamos que tenemos una discoteca y la puerta de entrada solo se puede abrir a personas mayores de edad que además tengan una tarjeta de invitación. Cuando llega una persona, el software de control de la puerta crea dos variables booleanas para registrar ambas condiciones, como por ejemplo:

```
boolean mayorEdad = true;
boolean tieneInvitacion = true;
```

A partir de estos datos, el software de la puerta debe calcular si se cumplen las dos condiciones a la vez, y para ello utiliza la operación Y, cuya definición es la siguiente:

- **La operación Y (también llamada AND), es una operación entre dos variables booleanas que produce como resultado true cuando el valor de las dos variables es true.**

&&	true	false
true	true	false
false	false	False

- **Su símbolo es &&**
- **Sirve para detectar si dos condiciones se dan a la vez**

Para hacer la operación, bastará escribir el símbolo && entre las dos variables booleanas, y como siempre, no debemos olvidarnos de recoger el valor de la operación en una variable.

boolean abrirPuerta = mayorEdad && tieneInvitación;

*nueva variable para guardar el resultado*      *indica si es mayor edad*      *indica si tiene invitación*

Como vemos, el cálculo de la operación Y entre las variables mayorEdad e invitación nos devuelve como resultado si la puerta debe abrirse. El siguiente código realizaría esta situación y mostraría por pantalla el resultado:

```
1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          boolean mayorEdad = true;
5          boolean tieneInvitacion = true;
6          System.out.println("¿Es mayor de edad? "+mayorEdad);
7          System.out.println("¿Tiene invitación? "+tieneInvitacion);
8          boolean abrirPuerta = mayorEdad && tieneInvitacion;
9          System.out.println("¿Se abre la puerta? "+abrirPuerta);
10     }
11 }
```

Al ejecutar este programa, veremos cómo efectivamente el ordenador comprueba que ambas variables guardan true, e indica que la puerta se debe abrir.

```

¿Es mayor de edad? true
¿Tiene invitación? true
¿Se abre la puerta? true
BUILD SUCCESSFUL (total time: 0 seconds)

```

#### 5.4.2.- La operación O

Esta operación se utiliza cuando tenemos dos cosas y queremos detectar si alguna de ellas se cumple.

Supongamos que tenemos una tienda que ofrece un descuento a los menores de 30 años y a los parados. Imaginemos que el software que asigna los descuentos registra estas dos variables a una persona que acaba de llegar a comprar:

```

boolean menor30 = false;
boolean parado = true;

```

¿Cómo hace el ordenador para detectar si debe aplicar un descuento? Para ello, deberá utilizar la operación O, cuya definición es la siguiente:

- La operación O (también llamada OR), es una operación entre dos variables booleanas que produce como resultado true cuando alguna de ellas vale true.

	true	false
true	true	true
false	true	false

- Su símbolo es ||
- Sirve para detectar si se da (al menos) una condición entre varias

En nuestro ejemplo, para calcular la aplicación del descuento escribiríamos esto:

```

boolean aplicarDescuento = menor30 || parado;

```

nueva variable para guardar el resultado      indica si es menor de 30      indica si es parado

El código completo que mostraría toda la situación sería:

```

1  import java.lang.*;
2  public class Ejemplo{
3      public static void main(String[] args){
4          boolean menor30 = false;
5          boolean parado = true;
6          System.out.println("¿Es menor de 30 años? "+menor30);
7          System.out.println("¿Es parado? "+parado);
8          boolean aplicarDescuento = menor30 || parado;
9          System.out.println("¿Aplicar descuento? "+aplicarDescuento);
10     }
11 }

```

Si lo ejecutamos, podremos comprobar cómo el programa detecta que se da una de las dos condiciones (en este caso, está parado) y por eso, le ofrece un descuento.

```
¿Es menor de 30 años? false
¿Es parado? true
¿Aplicar descuento? true
BUILD SUCCESSFUL (total time: 0 seconds)
```

#### 5.4.3.- La operación NO

Esta operación, simplemente invierte el valor de una variable booleana para obtener su opuesto, y se utiliza cuando queremos guardar lo contrario de lo que está guardado en otra variable booleana. Su símbolo es el signo !.

Por ejemplo, si tenemos una variable que guarda verdadero si una persona está casada, podemos calcular si está soltera invirtiendo su valor, así:

boolean soltero = !casado;

*nueva variable para guardar el resultado      operador NO      indica si está casado*

**Ejercicio 25 :** Realiza “a mano” los siguientes cálculos con las variables booleanas A=true, B=false, C=false y D=true

- a) A || ( B && C )
- b) ( A && !B ) || ! ( D && A )
- c) A || ( B && ( D && C || ! ( A || B ) && C ) )
- d) ! ( A || B ) || ( !A && !B )
- e) ( !C && ( !A || !B ) ) || ( D && ( ! ( C && B ) ) )

**Ejercicio 26 :** Realiza un programa que pregunte al usuario su nota final del examen. El programa mostrará si está aprobado y también si está suspenso.

**Ejercicio 27 :** Una tienda realiza descuentos a personas menores de edad y también a personas mayores de 60 años. Realiza un programa que pregunte una edad y nos muestre si tiene descuento.

**Ejercicio 28 :** En un colegio por motivos de seguridad los niños no pueden salir solos. La puerta de entrada está programada para abrirse cuando llega un adulto, o cuando llega un niño acompañado por un familiar. Haz un programa que pregunte por teclado si la persona que llega es adulta o no, y si va a acompañada o no. Tras recoger estos datos, el programa mostrará por pantalla si la puerta deberá o no abrirse.



**Ejercicio 29 :** Un usuario de un programa pulsa el botón de imprimir. Solo se puede realizar esta operación si la impresora está encendida, tiene tinta y el número de folios que hay en la impresora es mayor que el número de folios del trabajo que se van a imprimir. Realiza un programa que pida al usuario por teclado el porcentaje de tinta, los folios que hay en la impresora, los folios que se desean imprimir y si la impresora está encendida. El programa mostrará si se puede imprimir.

**Ejercicio 30 :** Un alumno ha sacado un 6.5, un 4.2, un 5.75, un 3.5 y un 8 en sus exámenes, y además, tiene 10 faltas. Realiza un programa que nos diga si el alumno ha aprobado, teniendo en cuenta que para aprobar se siguen estos criterios:

- Si el número de faltas es superior a 10, la nota media de los exámenes debe ser mayor que 5
- Si el número de faltas está entre 15 y 20, la nota media de los exámenes debe ser más de 7
- Si el número de faltas es superior a 20, está suspenso

**Ejercicio 31 :** En los ciclos de Formación Profesional, un alumno que falte más de un 20% de las horas de clase pierde el derecho a la evaluación continua.

Lenguajes de marcas	128 horas
Redes	192 horas
Hardware	96 horas

- a) Realiza un programa que muestre por pantalla la cantidad máxima de faltas permitidas para cada una de estas asignaturas
- b) Haz que el programa pregunte al usuario por el número de faltas de un alumno en redes y nos muestre si el alumno ha superado o no la cantidad de faltas indicada.

## 6.- Conversiones entre variables

Cuando hacemos un programa muchas veces necesitamos **convertir** una variable de un tipo en una de otro tipo. Por ejemplo, podría ocurrir que tuviésemos esta variable:

**int suedo = 1200**

En próximos temas veremos que hay situaciones donde nos interesará convertirla en una variable de otro tipo, como por ejemplo, un double, un long o un String. En este apartado vamos a ver las reglas que nos permitirán convertir variables de un tipo en otro.

### 6.1.- Conversión implícita

La primera regla que vamos a estudiar es que **una variable siempre se puede guardar en una de un tipo más grande.**



Por ejemplo, supongamos que tenemos una variable de tipo int:

```
int sueldo = 1200
```

Entonces podemos guardar esa variable sin ningún problema dentro de una variable cuyo tipo sea más grande que int. O sea, podremos hacer cosas como estas sin ningún problema:

```
3 | int sueldo=1200;
4 | double sueldo2=sueldo; // guardamos un int dentro de un double -> ok
5 | long sueldo3=sueldo; // guardamos un int dentro de un long -> ok
6 | double sueldo4=sueldo2; // guardamos un long dentro de un double -> ok
```

Cuando guardamos una variable de un tipo más pequeño en una de tipo más grande estamos haciendo una **conversión implícita**. Esta es la lista ordenada de los tipos de datos, de menor a mayor:

byte	short	int char	long	float	double
------	-------	-------------	------	-------	--------

¿Qué ocurre con los tipos que no aparecen en la tabla, como el boolean y el String? Pues que esos tipos **no admiten conversión implícita**. O sea, no podemos hacer cosas como las siguientes:

```
3 | int sueldo=1200;
4 | String sueldo2=sueldo; // convertir un int en String -> ERROR
5 | boolean sueldo3=sueldo; // convertir un int en boolean -> ERROR
```

Por tanto, este tipo de conversiones habrá que hacerlas de otra forma, ya que no se produce de forma automática. Más adelante veremos cómo se realizarán estas conversiones.

Por último, mencionamos un tipo muy curioso de conversión implícita, que es cuando aparece el tipo **char**. Como hemos visto, este tipo representa un carácter. En el lenguaje Java siempre que un char aparece en una operación (suma, resta, etc) se sustituye por su **código Unicode**.

🔗 **¿Qué es el código Unicode?** Los ordenadores internamente representan las letras como si fuesen números. O sea, cada letra (o símbolo) tiene asociado un número único. Unicode es una tabla que guarda la conversión a número de los caracteres de todos los idiomas del mundo. Para las letras que usamos habitualmente, esta tabla coincide con un sistema antiguo denominado "código ASCII".

TABLA DE CARACTERES DEL CÓDIGO ASCII																														
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58		
59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87		
88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116		
117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145		
146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174		
175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203		
204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232		
233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	PRESIONA LA TECLA Alt				MÁS EL NÚMERO			
CORTESÍA DE																														

Teniendo esto en cuenta, las siguientes líneas (aunque parezcan raras), serían correctas:

```
3 | | | int a = 'Z'; // guardamos un char en un int -> OK
4 | | | char b = 126; // guardamos un int en un char -> OK
```

En este ejemplo la variable a guardará el número 90 y la variable b guardará el carácter ~

## **6.2.- Conversión explícita (casting)**

La conversión explícita sucede cuando queremos guardar una variable de un tipo “grande” en una de un tipo más pequeño. Por ejemplo, podemos tener:

**double sueldo = 1200.53;**

Supongamos que queremos guardar esa variable dentro de una de tipo **int**, que es un tipo más pequeño. O sea, estamos justo en el caso contrario al del apartado anterior.

Para guardar una variable de un tipo grande en otra de un tipo “pequeño” usaremos una operación denominada **casting**, que consiste en “forzar” a la variable grande para que se convierta en una variable del tipo pequeño.

El casting se hace escribiendo entre paréntesis el tipo de dato pequeño (al que vamos a forzar la variable del tipo más grande). El resultado del casting es un dato que ya si se puede guardar en una variable del tipo pequeño.

*esta parte se llama casting*

int sueldo2= (int) sueldo;

*variable para guardar el resultado    tipo de dato destino    variable que va a ser convertida*

Cuando se hace casting, es posible que el dato de la variable grande se vea recortado para así ajustarse correctamente al tamaño de la variable pequeña.

Por ejemplo, supongamos que una variable de tipo **int** llamada **hijosAntonio** para indicar que Antonio tiene dos hijos. Ahora queremos hacer una nueva variable de tipo **byte**, llamada **hijosAntonioConvertido**, para guardar allí también su valor. El programa haría esto:

```
3 | | | int hijosAntonio=2;
4 | | | byte hijosAntonioConvertido=(byte)hijosAntonio;
```

En este caso, como el 2 es un número que cabe bien en la variable pequeña, tendremos que **hijosAntonioConvertido** vale 2 (o sea, el dato original no se ha visto recortado).

Pero ahora vamos con este ejemplo:

```
3 double sueldo=1000.50;
4 int sueldoConvertido=(int) sueldo;
```

Ahora la cosa cambia, porque el dato original es 1000.50, que no se puede guardar en un entero. Al forzar la conversión, el ordenador le recorta la parte decimal, de manera que la variable sueldoConvertido almacenará el valor 1000;

**Ejercicio 32 :** Indica cuáles de estas conversiones son posibles y cuáles no, indicando su tipo:

```
// Variables
int a = 8;
char b = 'x';
boolean c = true;
double d = 2.25;

// Conversiones
int conversion1 = d;
float conversion2 = c;
boolean conversion3 = (boolean)a;
float conversion4 = d;
double conversion5 = a;
short conversion6 = (short)a;
long conversion7 = b;
int conversion8 = (int)d;
boolean conversion9 = c;
char conversion10 = (char)d;
```

**Ejercicio 33 :** En una clase de 26 alumnos ha aprobado el 66% de alumnos, ha suspendido el 19.5% y el resto no se han presentado. Realiza un programa que muestre por pantalla el número de alumnos de cada tipo.

### 6.3.- Conversiones entre String y tipos básicos

Un String puede ser un texto cualquiera. Sin embargo, como no hay ninguna restricción al contenido de un String, puede ocurrir (y sucede con mucha frecuencia, como veremos en el punto siguiente) que en nuestros programas tengamos números escritos como texto dentro de un String.

En estos casos, surge la necesidad de poder “rescatar” el número que hay dentro del String y guardarlo en una variable numérica, para así poder hacer operaciones matemáticas.

Para convertir un String (cuyo contenido textual es un número o un valor boolean) en una variable de tipo básico, utilizaremos las siguientes órdenes, según el tipo al que queramos realizar la conversión:

Integer.parseInt	Convierte un String en un int
Long.parseLong	Convierte un String en un long
Double.parseDouble	Convierte un String en un double
Boolean.parseBoolean	Convierte un String en un boolean

La forma de utilizar estas expresiones es la misma. Entre paréntesis se encierra el dato String que queremos convertir, y en una nueva variable recogeremos el valor de la conversión.

Por ejemplo, supongamos que tenemos una variable String llamada alturaString que guarda “1.72”. Como vemos, se trata de un número guardado como si fuese un texto. Esto limita las cosas que puede hacer nuestro programa, porque con un texto no podemos hacer sumas, ni otras cuentas.

Si nuestro programa necesita hacer cálculos con ese dato, tendremos que obtener una variable numérica en la que se guarde su valor. En este ejemplo, como “1.72” guarda un número con decimales, haremos la conversión de String a double mediante Double.parseDouble escribiendo esto:

```
double altura= Double.parseDouble(alturaString);
```

*variable para guardar el resultado      orden para convertir String en double      variable String*

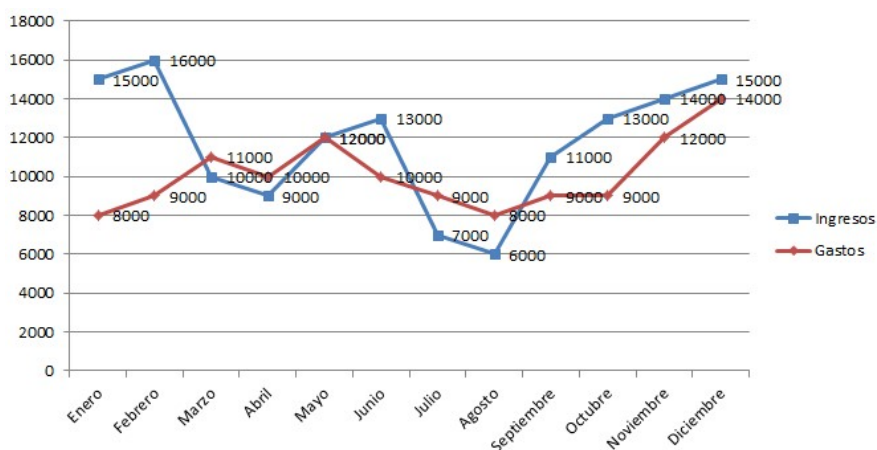
De esta forma, la variable altura contendrá el valor 1.72 como número, y nuestro programa ya si podrá hacer operaciones con ella.

Es importante destacar que este tipo de conversiones de String a tipo básico solo funcionan cuando el String contiene un texto que realmente se corresponde con un dato del tipo básico al que queremos hacer la conversión. En caso de que no sea así, el programa compilará correctamente, pero al ponerlo en marcha se producirá un error<sup>7</sup> indicando que no es posible realizar la conversión.

**Ejercicio 34 :** Realiza un programa con las variables que aparecen a continuación, y a continuación muestre por pantalla su suma y su producto.

```
String n1 = "125";
String n2 = "456";
```

**Ejercicio 35 :** La siguiente gráfica recoge los ingresos y gastos de una empresa en un año:



Haz un programa que muestre por pantalla:

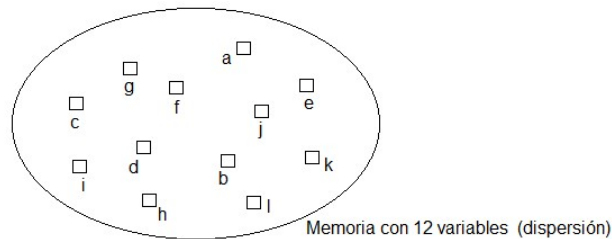
- Si cada mes ha tenido un balance positivo o negativo.
- La media de ingresos y gastos anual
- El balance final anual y si ha sido positivo o no.

<sup>7</sup> Los errores que aparecen cuando el programa está funcionando se denominan **errores en tiempo de ejecución**, o en inglés, **Runtime Errors**

## 7.- Listas

Los programas profesionales trabajan con muchísimos datos, y entonces puede pasarnos lo que hemos visto al hacer el ejercicio 36: que necesitamos una cantidad grandísima de variables.

Si usamos una variable por cada dato del programa, nos encontraremos con una cantidad inmanejable de variables que nos complicará trabajar con el conjunto.



Lo ideal es crear una **lista**, que nos va a permitir agrupar en una sola variable (la lista) muchos datos que sean similares, evitándonos tener que hacer cientos de variables<sup>8</sup> para cada uno de ellos.

Si hacemos una lista y guardamos en ella todos los datos que pertenecen a un mismo concepto (por ejemplo, todas las notas de la clase, o los nombres de todos los alumnos de la clase), en lugar de muchas variables, tendremos **una sola variable, que será la lista.**



### 7.1.- Los arrays

Para crear una lista nos basaremos en el siguiente ejemplo, que crea una lista de 5 números enteros:

```
int[] notas = { 4, 6, 9, 2, 5 };
      ↑           ↑           ↑
      |           |           |
tipo de dato lista nombre de la variable datos de la lista
```

Aquí vemos como el tipo de dato de una lista de números enteros es **int[]**. De igual forma, podemos hacer listas de los demás tipos de datos, como por ejemplo:

---

<sup>8</sup> Veremos más utilizades de las listas en el tema 3

```

3 | | | char[] vocales = {'a', 'e', 'i', 'o', 'u'};
4 | | | double[] temperaturas = {36.5, 40, 37.5, 36.75, 38.26};

```

Las listas que hemos aprendido a hacer de esta forma se llaman **arrays** y son tipos de dato **referencia**. Es frecuente ver que algunos libros traducen la palabra “array” como **vector** o **arreglo**.

## 7.2.- Cómo usar un array

Una vez que tenemos un array en nuestro programa, vamos a ver cómo podemos utilizarlo y qué cosas podemos (y no podemos) hacer con él.

### 1. No se puede ampliar ni disminuir el tamaño de un array

Esto significa que una vez que hemos creado un array, ya no le podremos añadir ni retirar más elementos. En la página siguiente veremos cómo podemos sustituir los elementos que guarda por otros diferentes, pero ya está.

### 2. Podemos consultar el total de datos guardados en el array

Esto lo haremos añadiendo **.length** a la variable del array, como se muestra en la línea 4 de este ejemplo:

```

3 | | | char[] vocales = {'a', 'e', 'i', 'o', 'u'};
4 | | | int totalVocales = vocales.length;

```

Si ejecutamos este programa podremos comprobar que la variable totalVocales guarda 5

### 3. Podemos acceder al elemento que ocupa una posición de la lista

Lo primero que tenemos que tener claro es la diferencia entre **dato** y **posición**. Un array es una lista, y por tanto, es un conjunto ordenado de datos (por ejemplo, la lista de alumnos de una clase ordenados por orden alfabético).

- Los **datos** son los elementos que hay guardados dentro de la lista
- Una **posición** es un número que se refiere a una casilla de la lista, según su número de orden. **En programación, la primera posición es el número 0 y no el 1, como estamos acostumbrados en la vida real.**



Por ejemplo, si tenemos una lista con los nombres de los jugadores y queremos mostrar por pantalla el jugador que ocupa la primera posición de la lista, escribiremos:

```
3 String[] jugadores={"Casillas","Albiol","Pique"};
4 System.out.println("El primer jugador es: "+jugadores[0]);
```

#### 4. Podemos reemplazar el elemento que ocupa una posición de la lista

Bastará con usar la operación de asignación e indicar el nuevo dato, de esta forma:

```
3 String[] jugadores={"Casillas","Albiol","Pique"};
4 jugadores[1]="Juan Diego";
5 System.out.println("El segundo jugador ahora es: "+jugadores[1]);
```

**Ejercicio 36 :** Repite el ejercicio 35 usando listas.

**Ejercicio 37 :** Realiza un programa que tenga una lista con los dos números enteros que tú quieras y los sume, mostrando por pantalla cada uno de los números y el resultado de la suma.

**Ejercicio 38 :** Realiza un programa que cree una lista de 5 palabras. El programa deberá mostrar por pantalla el número de palabras de la lista, la primera y la última de ellas. El programa deberá estar hecho con fácil mantenimiento (esto es que si ampliamos o disminuimos el número de palabras de la lista no debemos tocar nada más en las restantes líneas del programa).

## 8.- La estructura condicional

Hasta ahora todos los programas y ejercicios que hemos hecho son **secuenciales**, esto es, se ejecutan siempre desde el principio y avanzan línea a línea hasta que se llega al final del programa.

```
7 public class Programa {
8     public static void main(String[] args) {
9         System.out.println("Escribe tu nombre:");
10        String nombre=new Scanner(System.in).nextLine();
11        System.out.println("Hola "+nombre);
12        System.out.println("Adios "+nombre);
13    }
14 }
```



La ejecución del programa siempre va de arriba hacia abajo

Sin embargo, habrá muchas ocasiones donde nos va a interesar que el programa **tome una decisión**, y haga unas acciones si sucede algo, y otras alternativas en caso contrario.

Por ejemplo, supongamos que queremos hacer un programa que pregunte al usuario dos números enteros y nos muestre el resultado de su división. Podríamos hacerlo así:

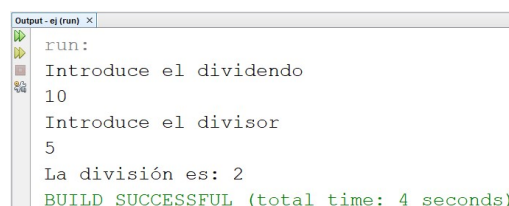


```

7   public class Programa {
8       public static void main(String[] args) {
9           System.out.println("Introduce el dividendo");
10          int dividendo=new Scanner(System.in).nextInt();
11          System.out.println("Introduce el divisor");
12          int divisor=new Scanner(System.in).nextInt();
13          // calculo la división
14          int division=dividendo/divisor;
15          System.out.println("La división es: "+division);
16      }
17  }

```

Si ejecutamos el programa y ponemos dividendo=10 y divisor=5, todo irá bien:

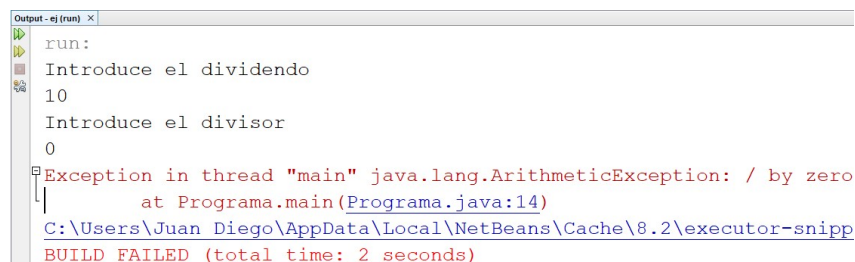


```

Output - ej (run) x
run:
Introduce el dividendo
10
Introduce el divisor
5
La división es: 2
BUILD SUCCESSFUL (total time: 4 seconds)

```

Pero cuando íbamos al instituto nos enseñaron que nunca se puede dividir un número entre cero. ¿Qué pasa si por ejemplo ponemos dividendo=10 y divisor=0? Aquí tenemos el resultado:



```

Output - ej (run) x
run:
Introduce el dividendo
10
Introduce el divisor
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Programa.main(Programa.java:14)
    C:\Users\Juan Diego\AppData\Local\NetBeans\Cache\8.2\executor-snipp
BUILD FAILED (total time: 2 seconds)

```

El programa ha fallado porque cuando se divide entre 0 queremos hacer una operación matemática no permitida y de ahí, el error. Como vemos, este programa no funciona correctamente en todos los casos, y da muy mala imagen a los usuarios. Este programa necesita que podamos examinar el valor de la variable “divisor” y tomar una decisión, según sea ese valor:

- Si divisor es distinto de 0, calculamos la división y mostramos el resultado en pantalla
- Si divisor es 0, mostramos un mensaje amigable de que no es posible calcular esa división

Las palabras de Java que nos van a permitir tomar una decisión en nuestro programa son **if** y **else**

## 8.1.- El bloque if-else

El bloque **if-else** sirve para alterar el ritmo de ejecución secuencial de un programa. Con él, el ordenador puede **tomar una decisión** y ejecutar unas líneas u otras alternativas, según el valor que tome **algo de tipo booleano**.



Básicamente consiste en que al llegar a un punto del programa, si un booleano vale "true", el ordenador ejecutará un bloque de líneas de código, y si vale "false", ejecutará otras alternativas, creándose así una especie de bifurcación dentro de la ejecución del programa.

La utilización de la palabra **if** es muy sencilla. Bastará con escribir a su lado entre paréntesis cualquier cosa cuyo tipo de dato sea boolean<sup>9</sup> y a continuación, se encierran entre llaves las líneas que se ejecutarán si dicho boolean vale true. Opcionalmente se puede escribir la palabra **else** y un bloque de llaves con las líneas que se ejecutarán si el boolean vale false.

```
if( /*cualquier boolean*/ ){  
    // líneas que se ejecutan si el boolean es true  
}else{  
    // líneas que se ejecutan si el boolean es false  
}
```

Por tanto, los componentes de la estructura condicional son:

- Un **boolean**, cuyo valor determinará si se ejecutan las líneas del if o las del else.
- Un bloque de líneas que acompaña al if. Este bloque es obligatorio y se ejecutará si el boolean es true.
- Un bloque de líneas opcional que acompaña a la palabra else. Este bloque se ejecutará si el boolean es false.

Gracias al bloque if-else es posible arreglar el programa que hicimos al principio del apartado, así

```
7 public class Programa {  
8     public static void main(String[] args) {  
9         System.out.println("Introduce el dividendo");  
10        int dividendo=new Scanner(System.in).nextInt();  
11        System.out.println("Introduce el divisor");  
12        int divisor=new Scanner(System.in).nextInt();  
13        // examino si es posible dividir o no  
14        if(divisor==0){  
15            System.out.println("No se puede dividir entre 0");  
16        }else {  
17            int division=dividendo/divisor;  
18            System.out.println("La división es: "+division);  
19        }  
20    }  
21 }
```

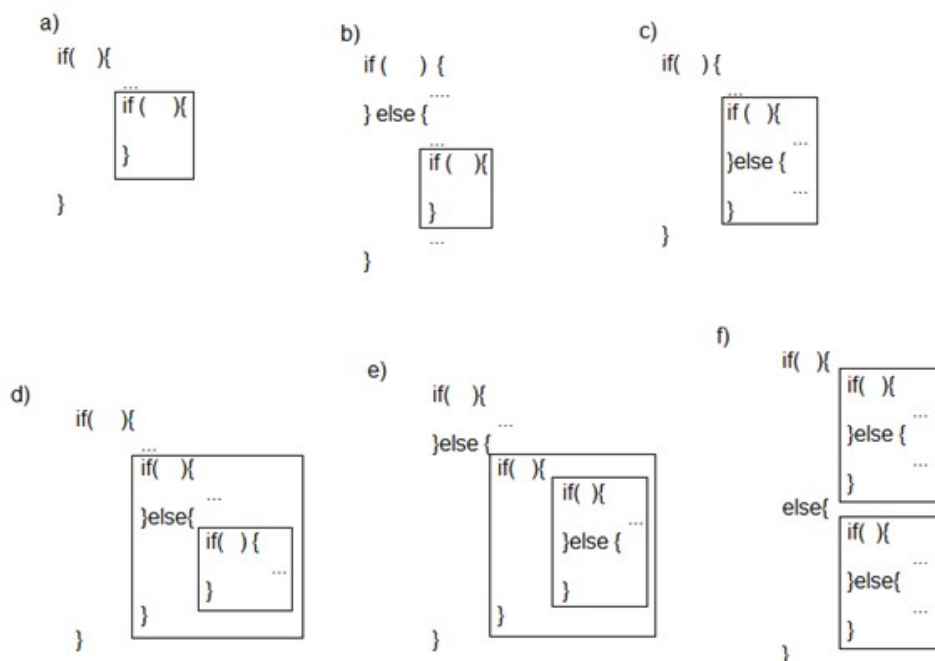
Si ahora el usuario introduce los valores que antes fallaban, dividendo=10 y divisor=0, el programa detectará esa situación y reaccionará para evitar el error:

---

<sup>9</sup> O sea, sirve cualquier cosa que podamos guardar dentro de una variable boolean. Por ejemplo, una comparación, una operación Y, etc...

```
Output - ej (run) x
run:
Introduce el dividendo
10
Introduce el divisor
0
No se puede dividir entre 0
BUILD SUCCESSFUL (total time: 4 seconds)
```

En los programas reales, los bloques if-else pueden contener todas las líneas de código que sean necesarias, y además se pueden **anidar**. Esto significa que se puede poner un bloque if-else dentro de otro. Si la situación lo requiere, se pueden formar estructuras tan complejas como estas:



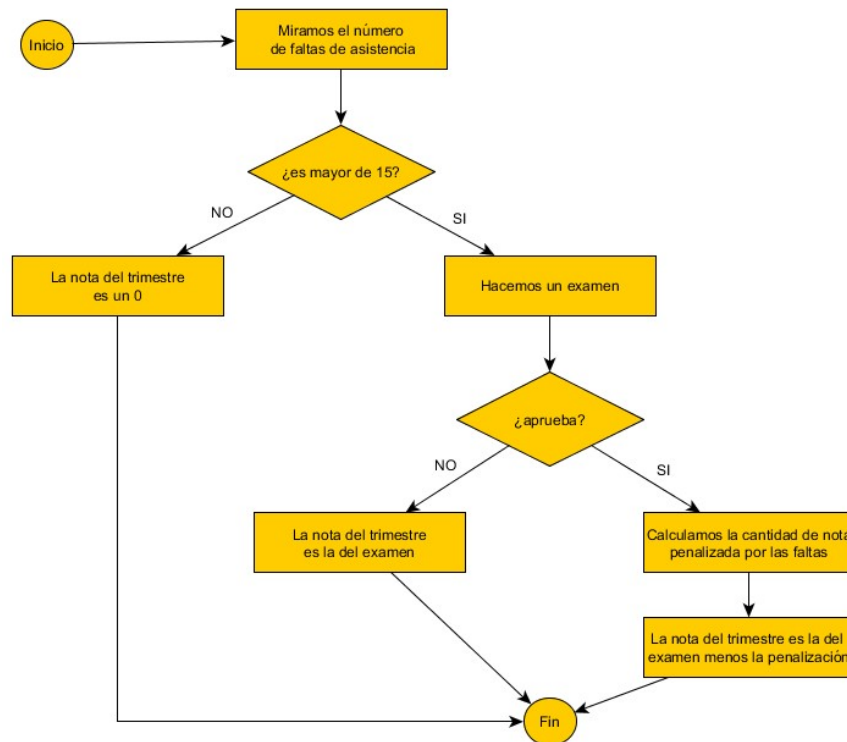
Vamos a ver un ejemplo más complicado donde encontraremos if-else anidados.

Ejemplo: Un profesor evalúa a los alumnos de la siguiente forma:

- Si un alumno tiene 15 o más faltas de asistencia no le deja hacer el examen y su nota del trimestre es un 0.
- Si un alumno tiene menos de 15 faltas, se le hace un examen y entonces:
  - Si el alumno suspende, su nota es la del examen
  - Si el alumno aprueba, su nota es la del examen menos un punto por cada 3 faltas.

Necesitamos un programa que ayude al profesor a obtener la nota del alumno. El programa comenzará preguntando el número de faltas y según lo que vaya pasando, irá preguntando otras cosas hasta llegar a calcular la nota final.

Solución: Un programa como este tiene ya cierto nivel de complejidad. Podemos hacer un esquema llamado **ordinograma**<sup>10</sup> que refleje la situación:



Cuando tenemos que hacer este tipo de programas es muy importante **ir poco a poco**. Si lo programamos todo del tirón y luego comprobamos si lo hemos hecho bien, lo más normal es que haya un montón de fallos y nos costaría mucho trabajo encontrarlos porque el código que hemos escrito sería muy largo y los fallos podrían estar en cualquier sitio.

Por tanto, lo ideal en estos programas más complicados es dar pequeños pasos y programar solo una pequeña parte del ejercicio. Después se prueba que lo que hemos hecho funciona. Así, si nos equivocamos, tenemos un código más pequeño que mirar y corregir. Cuando todo va bien, cogemos y repetimos el proceso programando un pequeño paso más.

Siguiendo esta norma, vamos a comenzar programando la primera parte del ejercicio: *“Si un alumno tiene 15 o más faltas de asistencia no le deja hacer el examen y su nota del trimestre es un 0”*

Para hacer esta parte escribimos código que pregunte al usuario el número de faltas de asistencia. Si el número de faltas es menor de 15, hacemos que termine el programa. En caso de que sea mayor de 15, ponemos un mensaje provisional que nos diga que eso lo haremos más adelante.

---

<sup>10</sup> Los ordinogramas son gráficos que describen cómo es un programa de ordenador. Cada rectángulo es una parte del programa que hace algo. Los rombos representan las decisiones que hay que tomar. Las empresas usan los ordinogramas cuando tienen que describir procesos difíciles de comprender de cabeza.

El código fuente sería este:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         System.out.println("Escribe el número de faltas:");
4.         int faltas=new Scanner(System.in).nextInt();
5.         if(faltas>15){
6.             System.out.println("Tu nota es: suspenso");
7.         }else {
8.             System.out.println("TODO - Hay menos de 15 faltas. Se hará después");
9.         }
10.    }
11. }
```

Como vemos, solo hemos hecho la primera parte del ejercicio. Si lo probamos, veremos que el programa funciona correctamente cuando metemos menos de 15 faltas, y si ponemos más de 15 faltas, entonces nos sale el mensaje “TODO”, que significa en inglés “to do = por hacer”.

Hecho esto ya estamos en condiciones de hacer la siguiente parte: *“Si un alumno tiene menos de 15 faltas, se le hace un examen y entonces, si el alumno suspende, su nota es la del examen”*.

Esta parte ocurre cuando el alumno tiene menos de 15 faltas. Eso se corresponde con el else donde hemos puesto el “TODO”. Así que eliminamos la línea “TODO” y nos ponemos a programar allí esa parte. El enunciado nos dice que se hace un examen y si suspende, la nota es la del examen. Por tanto, nuestro programa preguntará la nota del examen y en caso de que sea menor de 5, mostrará esa nota por pantalla. En cambio, si la nota es mayor de 5, estamos en un caso que se hará más adelante y pondremos un mensaje provisional indicando que lo haremos después.

El siguiente listado muestra el código fuente de nuestro programa provisional. Observa que la situación de nuestro programa nos lleva a programar dentro del else un nuevo bloque if-else.

```
1. public class Programa{
2.     public static void main(String[] args){
3.         System.out.println("Escribe el número de faltas:");
4.         int faltas=new Scanner(System.in).nextInt();
5.         if(faltas>15){
6.             System.out.println("Tu nota es: suspenso");
7.         }else {
8.             double nota=new Scanner(System.in).nextDouble();
9.             if(nota<5){
10.                 System.out.println("Tu nota es: "+nota);
11.             }else {
12.                 System.out.println("TODO - Examen aprobado. Se hace después");
13.             }
14.         }
15.    }
16. }
```

Si lo probamos veremos que el código funciona correctamente, y que si un alumno tiene menos de 15 faltas y suspende el examen, su nota es la del examen, tal y como nos pide el enunciado. Además, el caso que habíamos hecho antes (suspender si hay más de 15 faltas) sigue funcionando. Siempre que ampliamos un programa con nuevas funciones conviene probar que las funciones anteriores siguen funcionando correctamente. Muchas veces no nos damos cuenta e introducimos errores que hacen que las funciones anteriores dejen de funcionar. Aquí no nos ha pasado.

Por último, vamos a programar la parte final: *“Si el alumno aprueba, su nota es la del examen menos un punto por cada 3 faltas”*.

Esta parte se corresponde con el else que hay en la línea 11 del programa, que es el caso en que el alumno tiene menos de 15 faltas y ha aprobado el examen. Por tanto, borramos el mensaje provisional de la línea 12 y programamos allí.

La frase que tenemos que programar nos dice que hay una penalización que calcularemos restando un punto por cada 3 faltas. Esto nos dice que la penalización es el número de faltas (que está en la variable “faltas”) dividido entre 3.0 (ponemos 3.0 para que dividamos entre double y la división nos de decimales). La nota final será decrementar la variable nota la penalización.

El código final es este:

```
1. public class Programa{
2.     public static void main(String[] args){
3.         System.out.println("Escribe el número de faltas:");
4.         int faltas=new Scanner(System.in).nextInt();
5.         if(faltas>15){
6.             System.out.println("Tu nota es: suspenso");
7.         }else {
8.             double nota=new Scanner(System.in).nextDouble();
9.             if(nota<5){
10.                 System.out.println("Tu nota es: "+nota);
11.             }else {
12.                 double penalizacion=faltas/3.0;
13.                 nota -= penalizacion;
14.                 System.out.println("Tu nota es: "+nota);
15.             }
16.         }
17.     }
18. }
```

**Ejercicio 39 :** Realiza un programa que pregunte al usuario dos números enteros y nos diga si el primero es mayor o igual que el segundo

**Ejercicio 40 :** Realiza un programa que pregunte al usuario tres números enteros y el programa nos muestre el mayor de ellos.

**Ejercicio 41 :** Realiza un programa que pregunte un número comprendido entre 10 y 56 por teclado al usuario. Si el número está en dicho rango, el programa mostrará un mensaje de que es correcto, y en caso contrario, se indicará que no lo es.

**Ejercicio 42 :** Realiza un programa que pregunte al usuario su nota numérica, y el ordenador muestre por pantalla la calificación obtenida según esta tabla:

Nota	Menor de 5	Entre 5 y 6	Entre 6 y 7	Entre 7 y 9	Entre 9 y 10	10
Calificación	Suspenso	Suficiente	Bien	Notable	Sobresaliente	Matrícula

**Ejercicio 43 :** Haz un programa en el que haya una variable entera llamada número con un valor cualquiera. El programa deberá mostrar por pantalla si dicho número es par.

**Ejercicio 44 :** Realiza un programa que indique la cantidad de dinero que hay que cobrar o devolver a un cliente de una tienda, de esta forma:

- El programa comienza preguntando el precio del artículo.
- A continuación, se pregunta la cantidad de dinero que da el cliente. (*esta cantidad puede ser mayor o menor que el precio del artículo*)
- El programa indicará si al cliente le falta dinero o si es necesario darle el cambio. En ambos casos se indicará la cantidad de dinero necesaria.

**Ejercicio 45 :** Haz un programa que nos diga el tipo de triángulo que forman tres longitudes A, B y C:

- Rectángulo:  $A^2 = B^2 + C^2$
- Acutángulo:  $A^2 < B^2 + C^2$
- Obtusángulo:  $A^2 > B^2 + C^2$



**Ejercicio 46 :** Realiza un programa que pregunte al usuario su año de nacimiento, si es hombre o mujer, su altura (en metros), peso (en kg) y muestre por pantalla su frecuencia cardiaca máxima, su índice de masa corporal y su estado según la Organización Mundial de la Salud.

- Frecuencia cardiaca máxima: Si es hombre se calcula como  $220 - \text{edad}$ , y si es mujer se calcula como  $226 - \text{edad}$ .
- Índice de masa corporal (IMC): Se calcula dividiendo el peso entre la altura elevada al cuadrado y se muestra con dos cifras decimales.
- El estado según la OMS se mira en esta tabla:

IMC	Estado
Menor de 16.00	Infrapeso: delgadez severa
Entre 16.00 y 16.99	Infrapeso: delgadez moderada
Entre 17.00 y 18.49	Infrapeso: delgadez aceptable
Entre 18.50 y 24.99	Peso normal
Entre 25.00 y 29.99	Sobrepeso
Entre 30.00 y 34.99	Sobrepeso: tipo I
Entre 35.00 y 40.00	Sobrepeso: tipo II
Mayor de 40.00	Sobrepeso: tipo III

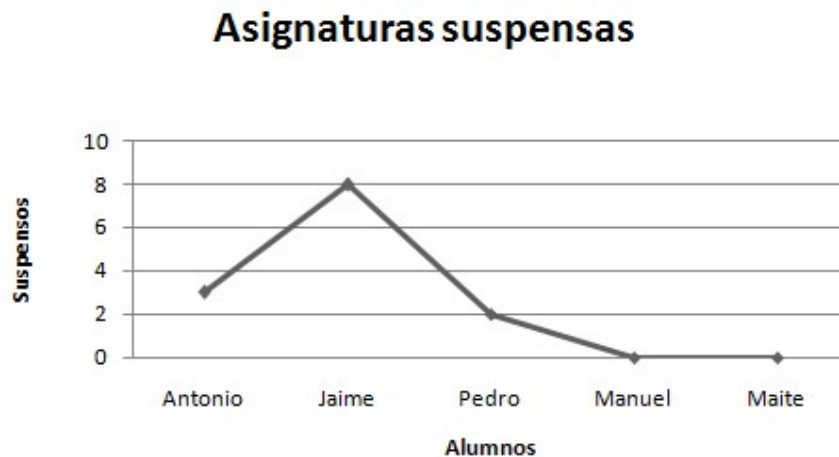
**Ejercicio 47 :** Realiza un programa en el que haya una variable que almacena un carácter. El programa deberá mostrar por pantalla si ese carácter corresponde a una letra.

*Sugerencia: Convierte el carácter a número y usa la tabla ASCII*

**Ejercicio 48 :** Realiza un programa en el que haya una variable entera llamada año inicializada con cualquier valor positivo. El programa mostrará por pantalla si el año guardado en dicha variable es bisiesto. Ten en cuenta que un año es bisiesto cuando se cumple alguna de estas dos condiciones:

- a) Es múltiplo de 400
- b) No es múltiplo de 400, pero es múltiplo de 4 y no es múltiplo de 100

**Ejercicio 49 :** La siguiente gráfica nos muestra el número de asignaturas suspensas de unos alumnos. Se supone que un alumno pasa de curso si no tiene asignaturas suspensas. Realiza un programa que pregunte al usuario por un número de lista entre 0 y 4 y nos muestre por pantalla el nombre del alumno seleccionado, su número de suspensos y si pasa de curso o no.



**Ejercicio 50 :** Hacer un programa en el que haya una variable entera llamada dinero e inicializarla a cualquier valor. El programa nos expresará esa cantidad en billetes de 500, 100, 50, 20 y 10 € y monedas de 2 y 1 €. Se ignoran los céntimos.

*Sugerencia: Antes de programar, hacer un ejemplo con lápiz y papel. Una vez que tengas claro el procedimiento que hay que seguir para resolver el problema planteado, traslada a un programa las mismas operaciones que realizas sobre el papel.*

**Ejercicio 51 :** Haz un programa que pregunte al usuario por su número de DNI. El programa mostrará la letra que corresponde a dicho DNI, teniendo en cuenta que se obtiene así:

- Calculamos el resto de dividir el número de DNI entre 23
- Buscamos en la siguiente lista la letra que corresponde a dicho resto y esa será la letra del DNI

0 → T	1 → R	2 → W	3 → A	4 → G	5 → M
6 → Y	7 → F	8 → P	9 → D	10 → X	11 → B
12 → N	13 → J	14 → Z	15 → S	16 → Q	17 → V
18 → H	19 → L	20 → C	21 → K	22 → E	23 → T

**Ejercicio 52 :** Estamos haciendo un programa de loterías y tenemos dos variables llamadas número y NúmeroPremiado, que se encuentran inicializadas con números de 5 cifras. El programa deberá mostrar por pantalla:

- a) Si el número coincide con el número premiado.
- b) Si el número tiene reintegro, es decir, se dan a la vez las dos siguientes condiciones:
  - a. La primera cifra del número coincide con la primera cifra del número premiado
  - b. La última cifra del número coincide con la última cifra del número premiado

*Sugerencia:* Obtén el cociente y resto de dividir el número premiado entre 10, y mira si tienen algo que ver con su última cifra. Inventa algo parecido para obtener la primera cifra.

**Ejercicio 53 :** Un equipo de 7 amigos futbolistas llamados Pedro, Luis, Miguel, Jorge, Juan, Manolo y Pepe echa a suerte quien se pone de portero. Para eso, todos sacan a la vez la mano y se cuenta el número de dedos que ha sacado cada uno. Entonces, se va contando de uno en uno y al que le toque, se pone de portero. Si al llegar al último jugador no se ha terminado, se sigue contando comenzando por el primero.

Realiza un programa que muestre por pantalla el nombre del jugador que se pone de portero si el número de dedos que ha sacado cada amigo es:

4	5	1	0	3	2	3
---	---	---	---	---	---	---

## **8.2.- La asignación condicional**

La **asignación condicional** es una estructura que hace unos años estaba “maldita” y no se recomendaba su uso porque “hacía más difíciles de leer los programas”. Sin embargo, actualmente se ha visto su utilidad para hacer código menos propenso a fallos<sup>11</sup>, por lo que se ha puesto de moda otra vez. De hecho, los lenguajes de programación más recientes, como Kotlin, incorporan estructuras de este tipo debido a que se ha visto que reducen la aparición de errores.

**La asignación condicional sirve para inicializar una variable basándonos en una condición.**

Vamos a ver como funciona con un ejemplo. Supongamos que un cine hace este descuento según la edad:

Menores de edad	3€
Mayores de edad	7€

Queremos hacer un programa que pida la edad de una persona por teclado y nos calcule cuánto descuento le corresponde. Este programa lo podemos hacer, de la forma habitual, con un bloque if-else de esta forma:

---

<sup>11</sup> En el tema 2 veremos situaciones donde si no vamos con cuidado, utilizar if nos podrá dar error, pero si usamos la asignación condicional, no.



```

7 public class Programa {
8     public static void main(String[] args) {
9         System.out.println("Introduce tu edad");
10        int edad=new Scanner(System.in).nextInt();
11        int descuento = 0;
12        if (edad>=18) {
13            descuento=7;
14        }else {
15            descuento=3;
16        }
17        System.out.println("Tu descuento es: "+descuento+" euros");
18    }
19 }

```

No hay ningún problema con esto, pero observa que hemos necesitado escribir 6 líneas para asignar correctamente el valor de la variable “descuento”.

Con la asignación condicional, podemos hacer lo mismo en una sola línea, así:

`int descuento = edad>=18 ? 7 : 3`

O sea, toda la parte encargada de dar valor a la variable “descuento” se puede hacer así:

```

7 public class Programa {
8     public static void main(String[] args) {
9         System.out.println("Introduce tu edad");
10        int edad=new Scanner(System.in).nextInt();
11        int descuento = edad<18 ? 3:7;
12        System.out.println("Tu descuento es: "+descuento+" euros");
13    }
14 }

```

Lo que hacemos es preguntarnos **edad<18?** y si la respuesta es verdadera, asignamos el valor **3** a la variable descuento, y en caso contrario, el valor **7**. El signo **:** se usa para separar el valor que se guardará en la variable si la condición es verdadera del valor que se guardará si la condición es falsa.

En resumen:

- Se escribe una condición que determinará el valor que se guardará en la variable. El signo **?** que se pone al final de la condición, como si nos estuviésemos haciendo una pregunta.
- El valor que hay a la izquierda del signo **:** es el valor que se guardará en la variable en caso de que la condición sea true
- El valor que hay a la derecha del signo **:** es el valor que se guardará en la variable en caso de que la condición sea false

Como podemos ver, la asignación condicional nos “acorta” la forma de dar el valor inicial a la variable “descuento”, aunque hay gente que le cuesta trabajo entender cómo funciona.

**Ejercicio 54 :** Busca algún ejercicio donde puedas hacer uso de la asignación condicional y prográmalo haciendo uso de ella.