

```
53     System.out.println(Arrays.toString(texto3.toCharArray())); // [ T, e, x, t, o]
54     System.out.println(texto3.toLowerCase()); // texto
55     System.out.println(texto3.toUpperCase()); // TEXTO
56     System.out.println(texto1.trim()); // Texto para hacer las pruebas
57     System.out.println(texto1.valueOf(valor1)); // 1234
58
59 }
60
61 }
62
63 }
```



Programa que crea aleatoriamente una tabla con los resultados de todos los encuentros entre 5 equipos de fundamentos de programación fútbol, mostrando la clasificación final.
(unidad 5) estructuras básicas de datos en Java

(5.3) arrays de Strings

Los arrays se aplican a cualquier tipo de datos y eso incluye a los Strings. Es decir, se pueden crear arrays de textos. El funcionamiento es el mismo, sólo que en su manejo hay que tener en cuenta el carácter especial de los Strings.

Ejemplo:

```
String[] texto=new String[4];
texto[0]="Hola";
texto[1]=new String();
texto[2]="Adiós";
texto[3]=texto[0];
for (int i = 0; i < texto.length; i++) {
    System.out.println(texto[i]);
}
/*se escribirá:
 Hola
 Adiós
 Hola
 */
```

(5.4) parámetros de la línea de comandos

Uno de los problemas de trabajar con entornos de desarrollo es que nos abstrae quizás en exceso de la realidad. En esa realidad un programa java se ejecuta gracias a la orden **java programa**; orden que hay que ejecutar en la línea de comandos del sistema.

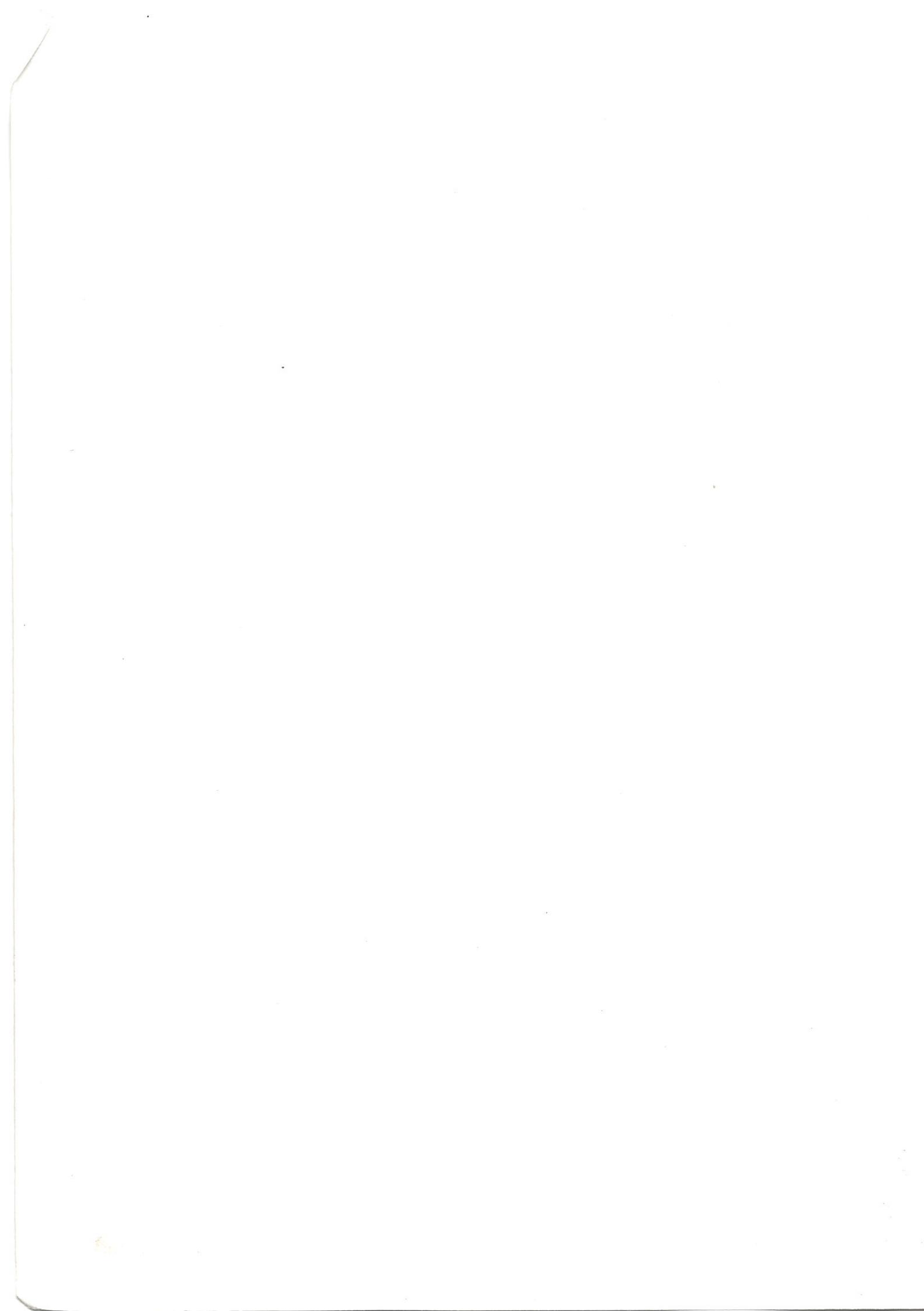
En cualquier momento podemos hacerlo. Pero es más, podemos añadir parámetros al programa. Por ejemplo supongamos que hemos creado un programa capaz de escribir un texto un número determinado de veces. Pero en lugar de leer el texto y el número por teclado, queremos que lo pasen como parámetro desde el sistema. Eso significa añadir esa información en el sistema operativo, por ejemplo con:

```
java Veces Hola 10
```

Eso podría significar que invocamos al programa llamado **Veces** y le pasamos dos parámetros: el primero es el texto a escribir () y el segundo el número de veces.

El método **main** es el encargado de recoger los parámetros a través de la variable **args**. Esta variable es un array de Strings que recoge cada uno de los parámetros. De tal modo que **args[0]** es un String que recoger el primer parámetro, **args[1]** el segundo, etc. Si no hay parámetros, **args.length** devuelve cero.





Ejemplo, se trata de comprobar si el texto que se lee mediante la clase **JOptionPane** empieza con dos guiones, le siguen tres números y luego una o más letras mayúsculas. De no ser así, el programa vuelve a pedir escribir el texto:

```
import javax.swing.JOptionPane;  
  
public class ExpresionRegular1 {  
  
    public static void main(String[] args) {  
        String respuesta;  
        do {  
            respuesta=JOptionPane.showInputDialog("Escribe un texto");  
            if(respuesta.matches("--[0-9]{3}[A-Z]+")==false) {  
                JOptionPane.showMessageDialog(null,  
                    "La expresión no encaja con el patrón");  
            }  
        }while(respuesta.matches("--[0-9]{3}[A-Z]+")==false);  
        JOptionPane.showMessageDialog(null,"Expresión correcta!"  
    }  
}
```

lista completa de métodos

método	descripción
char charAt(int index)	Proporciona el carácter que está en la posición dada por el entero index .
int compareTo(String s)	Compara las dos cadenas. Devuelve un valor menor que cero si la cadena s es mayor que la original, devuelve 0 si son iguales y devuelve un valor mayor que cero si s es menor que la original.
int compareIgnoreCase(String s)	Compara dos cadenas, pero no tiene en cuenta si el texto es mayúsculas o no.
String concat(String s)	Añade la cadena s a la cadena original.
String copyValueOf(char[] data)	Produce un objeto String que es igual al array de caracteres data .
boolean endsWith(String s)	Devuelve true si la cadena termina con el texto s
boolean equals(String s)	Compara ambas cadenas, devuelve true si son iguales
boolean equalsIgnoreCase(String s)	Compara ambas cadenas sin tener en cuenta las mayúsculas y las minúsculas.
byte[] getBytes()	Devuelve un array de bytes que contiene los códigos de cada carácter del String

startsWith

Devuelve **true** si la cadena empieza con un determinado texto.

replace

Cambia todas las apariciones de un carácter por otro en el texto que se indique y lo almacena como resultado. El texto original no se cambia, por lo que hay que asignar el resultado de **replace** en otro String para no perder el texto cambiado:

```
String s1="Mariposa";
System.out.println(s1.replace('a','e'));      //Da Meripose
System.out.println(s1);                      //Sigue valiendo Mariposa
```

replaceAll

Modifica en un texto cada entrada de una cadena por otra y devuelve el resultado. El primer parámetro es el texto que se busca (que puede ser una expresión regular), el segundo parámetro es el texto con el que se reemplaza el buscado. La cadena original no se modifica.

```
String s1="Cazar armadillos";
System.out.println(s1.replaceAll("ar","er")); //Escribe: Cazer ermedillos
System.out.println(s1);                      //Sigue valiendo Cazar armadillos
```

toUpperCase

Obtiene la versión en mayúsculas de la cadena. Es capaz de transformar todos los caracteres nacionales:

```
String s1 = "Batallón de cigüeñas";
System.out.println(s1.toUpperCase()); //Escribe: BATALLÓN DE CIGÜEÑAS
System.out.println(s1); //Escribe: Batallón de cigüeñas
```

toLowerCase

Obtiene la versión en minúsculas de la cadena.

toCharArray

Consigue un array de caracteres a partir de una cadena. De esa forma podemos utilizar las características de los arrays para manipular el texto, lo cual puede ser interesante para manipulaciones complicadas.

```
String s="texto de prueba";
char c[]=s.toCharArray();
```

mayor devuelve **-1**. Hay que tener en cuenta que el orden no es el del alfabeto español, sino que usa la tabla ASCII, en esa tabla la letra **ñ** es mucho mayor que la **o**.

- ◆ **s1.compareToIgnoreCase(s2)**. Igual que la anterior, sólo que además ignora las mayúsculas (disponible desde Java 1.2)

(5.2.3) **String.valueOf**

Este método pertenece no sólo a la clase String, sino a otras y siempre es un método que convierte valores de una clase a otra. En el caso de los objetos String, permite convertir valores que no son de cadena a forma de cadena. Ejemplos:

```
String numero = String.valueOf(1234);
String fecha = String.valueOf(new Date());
```

No vale para cualquier tipo de datos, pero sí para casi todos los vistos hasta ahora. No valdría por ejemplo para los arrays.

(5.2.4) **métodos de los objetos String**

Cada nuevo String que creemos posee, por el simple hecho de ser un String, una serie de métodos que podemos utilizar para facilitar nuestra manipulación de los textos. Para utilizarlos basta con poner el nombre del objeto (de la variable) String, un punto y seguido el nombre del método que deseamos utilizar junto con los parámetros que necesita. método y sus parámetros después del nombre de la variable String. Es decir:

```
variableString.método(argumentos)
```

length

Permite devolver la longitud de una cadena (el número de caracteres de la cadena):

```
String texto1="Prueba";
System.out.println(texto1.length());//Escribe 6
```

concatenar cadenas

Se puede hacer de dos formas, utilizando el método **concat** o con el operador suma (**+**). Desde luego es más sencillo y cómodo utilizar el operador suma. Ejemplo:

```
String s1="Buenos ", s2=" días", s3, s4;
s3 = s1 + s2;           //s3 vale "Buenos días"
s4 = s1.concat(s2);    //s4 vale "Buenos días"
```



```
1 public class Prueba {  
2     public static void main (String args[]) {  
3         int array1[][]=new int[][]{{1,2},{3,4},{5,6}};  
4         //int array1[][]=new int[3][];  
5         //array1[0]=new int[]{1,2};  
6         System.out.println(array1[2].length);  
7         for (int valores1[]:array1)  
8             for (int valores2:valores1)  
9                 System.out.println(valores2);  
10    }  
11 }  
12 }  
13 }  
14 }
```

Ejercicio nº 22
Programa que genera una matriz de enteros de 4x4, al
macendido en cada celda el resultado de sumar fila + columna
fundamentos de programación (unidad 5) estructuras básicas de datos en Java

copyOfRange

Funciona como la anterior (también está disponible desde la versión 1.6), sólo que indica con dos números de qué elemento a qué elemento se hace la copia:

```
int a[] = {1,2,3,4,5,6,7,8,9};  
int b[] = Arrays.copyOfRange(a, 3,6); // b vale {4,5,6}
```

(5.1.6) el método System.arraycopy (paquete java.lang)

La clase System también posee un método relacionado con los arrays, dicho método permite copiar un array en otro. Recibe cinco argumentos: el array que se copia, el índice desde que se empieza a copia en el origen, el array destino de la copia, el índice desde el que se copia en el destino, y el tamaño de la copia (número de elementos de la copia). Que se toman del array

```
int uno[] = {1,1,2};  
int dos[] = {3,3,3,3,3,3,3,3};  
System.arraycopy(uno, 0, dos, 0, uno.length);  
for (int i=0; i<=8; i++){  
    System.out.print(dos[i] + " ");  
} //Sale 1 12333333
```

La ventaja sobre el método copyOf de la clase Arrays está en que este método funciona en cualquier versión de Java.

(paquete java.lang) (también clase Math) (5.2) clase String - Hoja resumen métodos

(5.2.1) introducción. declarar e iniciar textos;

Una de las carencias más grandes que teníamos hasta ahora al programar en Java, era la imposibilidad de almacenar textos en variables. El texto es uno de los tipos de datos más importantes y por ello Java le trata de manera especial.

Para Java las cadenas de texto son objetos especiales. Los textos deben manejarse creando objetos de tipo **String** (String se suele traducir como cadena; cadena de texto).

Declarar e iniciar un texto se suele hacer de esta forma:

```
String texto1 = "¡Prueba de texto!";
```

Las cadenas pueden ocupar varias líneas utilizando el operador de concatenación "+".

```
String texto2 = "Este es un texto que ocupa " +  
    "varias líneas, no obstante se puede " +  
    "perfectamente encadenar";
```

(5.1.4) longitud de un array (atributo de los arrays)

Los arrays poseen un método que permite determinar cuánto mide un array; es decir, de cuántos elementos consta. Se trata de **length**. Ejemplo:

```
int a[] = new int [17];  
int b[][] = new int [30][7];
```

```
System.out.println(a.length); //Escribe: 17  
System.out.println(b.length); //Escribe: 30  
System.out.println(b[7].length); //Escribe: 7
```

Gracias a este método el bucle de recorrido de un array (para cualquier tipo de array) es (ejemplo para escribir cada elemento del array **x** por pantalla):

```
for (int i = 0; i < x.length; i++) {  
    System.out.print(x[i] + " ");  
}  
System.out.println();
```

(5.1.5) la clase Arrays

- Hoja Resumen métodos.

En el paquete **java.util** se encuentra una clase estática llamada **Arrays**. Una clase estática permite ser utilizada como si fuera un objeto (como ocurre con **Math**), es decir que para utilizar sus métodos hay que utilizar simplemente esta sintaxis:

```
Arrays.método(argumentos);
```

fill

Permite llenar todo un array **unidimensional** con un determinado valor. Sus argumentos son el array a llenar y el valor deseado:

```
int a[] = new int[23];  
Arrays.fill(valores,-1); //Todo el array vale -1
```

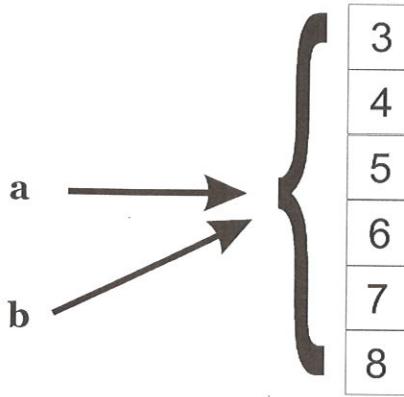
También permite decidir desde qué índice hasta qué índice rellenamos:

```
Arrays.fill(valores,5,8,-1); //Del elemento 5 al 7 valdrán -1
```

Un array se puede asignar a otro array (si son del mismo tipo):

```
int a[];  
int b[] = new int[]{3,4,5,6,7,8};  
a=b;
```

A diferencia de lo que podríamos intuir, lo que ocurre en el código anterior es que tanto *a* como *b* hacen referencia al mismo array. Es decir el resultado sería:



Esta asignación provoca que cualquier cambio en *a* también cambie el array *b* (ya que, de hecho, es el mismo array). Ejemplo:

```
int a[]={3,3,3};  
int b[];  
b= a;  
b[0]=8;  
System.out.println(a[0]); //Escribirá el número 8
```

Finalmente, como detalle final a este tema, el operador de igualdad (`==`) se puede utilizar con arrays, pero nuevamente no compara el contenido sino si las referencias señalan al mismo array. Es decir:

```
int a[]={3,3,3};  
int b[]={3,3,3};  
System.out.println(a==b); //escribe false, aunque ambos arrays  
//tienen el mismo contenido  
int c[]=b;  
System.out.println(b==c); //escribe true
```

for para lectura de arrays

```
int a[]={1,2,3};  
for (int lista: a)  
    System.out.println(lista);
```

(10)

Ejercicio n° 23

Crea un array con un tamaño indicado por teclado.
El array se rellenará con valores entre los enteros 1 y 300
y mostrará los valores del array separados por espacios y el valor
de aquellos valores iguales a la cifra solicitada por teclado.



Ejemplo 2: Programa que lea 5 números por teclado y los muestre en orden inverso a como se han introducido.

fundamentos de programación

(unidad 5) estructuras básicas de datos en Java

En Java (como en otros lenguajes) el primer elemento de un array es el cero. El primer elemento del array notas, es `notas[0]`. Se pueden declarar arrays a cualquier tipo de datos (enteros, booleanos, doubles, ... e incluso objetos como se verá más adelante).

La ventaja de usar arrays (volviendo al caso de las notas) es que gracias a un simple bucle `for` se puede recorrer fácilmente todos los elementos de un array:

```
//Calcular la media 18 notas  
suma=0;  
for (int i=0;i<=17;i++){  
    suma+=nota[i];  
}  
media=suma/18;
```

A un array se le puede inicializar las veces que haga falta:

```
int  
int notas[]=new notas[16];  
...  
notas=new notas[25];
```

Pero hay que tener en cuenta que el segundo `new` hace que se pierda el contenido anterior. De hecho elimina ese contenido.

En la perspectiva de Java, un array es una referencia a una serie de valores que se almacenan en la memoria. El operador `new` en realidad lo que hace es devolver una **referencia** a los valores a los que ha asignado el hueco en memoria. Es decir el array es la manera de poder leer y escribir en esos valores.

Veamos paso a paso estas instrucciones y su efecto en la memoria:

Ejercicio uº 18

Programa que lea 5 números por teclado y los muestre ordenados de menor a mayor.
Salida: Los 5 números, ordenados: 1, 2, 4, 5 y 7

Ejercicio uº 19

Programa que genere 6 números aleatorios comprendidos entre 1 y 49 y los muestre ordenados de menor a mayor.

una misma estructura que comúnmente se conoce como array¹. Esa estructura permite referirnos a todos los elementos, pero también nos permite acceder individualmente a cada elemento.

Los arrays son una colección de datos del mismo tipo al que se le pone un nombre (por ejemplo **nota**). Para acceder a un dato individual de la colección hay que utilizar su posición. La posición es un número entero, normalmente se le llama **índice** (por ejemplo **nota[4]** es el nombre que recibe el cuarto elemento de la sucesión de notas).

Hay que tener en cuenta que en los arrays el primer elemento tiene como índice el número cero. El segundo el uno y así sucesivamente; es decir **nota[4]** en realidad es el quinto elemento del array.

Esto (con algunos matices funciona igual en casi cualquier lenguaje). Sin embargo en Java los arrays (como casi todo) son **objetos**, y aunque la programación orientada a objetos de Java será abordada más adelante, ya ahora conviene tenerlo en cuenta.

Por otro lado mientras en lenguajes como C los arrays son **estructuras estáticas** (significa que su tamaño queda definido en cuanto se declara el array), en Java son **estructuras dinámicas** (sí que es posible modificar su tamaño en tiempo de ejecución).

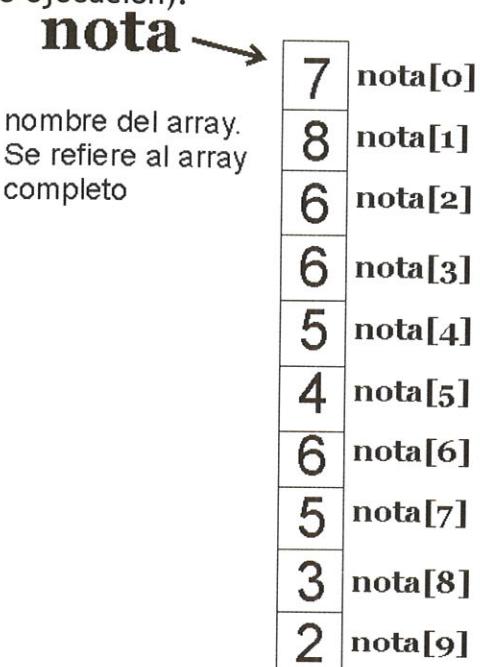


Ilustración 4-1, Ejemplo de array de notas

(5.1.2) unidimensionales

Los arrays habituales son los llamados unidimensionales. Coincidén con la idea comentada en el apartado anterior. Agrupan datos del mismo tipo y un

¹ Otros nombres habituales además de **arrays** son: listas, matrices, arreglos,... Estimo que array es el más aceptado en la actualidad, los otros términos son más ambiguos.



REPASO

Ejercicio nº 13

Programa que pida un número entero entre 0 y 99, y lo muestre escrito en letra.

Ejercicio nº 14

Programa que pida 10 números enteros, mostrando la media de los números positivos, la media de los números negativos y el número de 0 introducido.

Ejercicio nº 15

Programa que muestre un contador de 3 dígitos (Entre 000 y 999).

Ejercicio nº 16

Programa que nos pida un entero n por teclado, y muestre cuantos números primos hay entre 1 y n .

Ejercicio nº 17

Programa que nos muestre el factorial de un número entero introducido por teclado.
El factorial de n , es _____

menos entre 100 y 1.

- ANIDACION DE BUCLES

Ejercicio n° 10) Mostrar un rectángulo construido con asteriscos, sus anchos y altos se introducen por teclado.

fundamentos de programación

(unidad 4) programación estructurada en Java

- (2) Se comprueba la condición
- (3) Si la condición es cierta, entonces se ejecutan las sentencias. Si la condición es falsa, abandonamos el bloque **for**
- (4) Tras ejecutar las sentencias, se ejecuta la instrucción de incremento y se vuelve al paso 2

Ejemplo (contar números del 1 al 1000):

```
for(int i=1;i<=1000;i++){  
    System.out.println(i);  
}
```

La ventaja que tiene es que el código se reduce. La desventaja es que el código es menos comprensible. El bucle anterior es equivalente al siguiente bucle **while**:

```
int i=1; /*sentencia de inicialización*/  
while(i<=1000) { /*condición*/  
    System.out.println(i);  
    i++; /*incremento*/  
}
```

Como se ha podido observar, es posible declarar la variable contadora dentro del propio bucle **for**. De hecho es la forma habitual de declarar un contador. De esta manera se crea una variable que muere en cuanto el bucle acaba.

(4.7) sentencias de ruptura de flujo

No es aconsejable su uso ya que son instrucciones que rompen el paradigma de la programación estructurada. Cualquier programa que las use ya no es estructurado. Se comentan aquí porque puede ser útil conocerlas, especialmente para interpretar código de terceros.

(4.7.1) sentencia break

Se trata de una sentencia que hace que el flujo del programa abandone el bloque en el que se encuentra.

```
for(int i=1;i<=1000;i++){  
    System.out.println(i);  
    if(i==300) break;  
}
```

En el listado anterior el contador no llega a 1000, en cuanto llega a 300 sale del **for**

Sería:

```
boolean salir = false; //centinela
int n;
int i=1; //contador
while (salir == false && i<=5) {
    n = (int) Math.floor(Math.random() * 500 + 1);
    System.out.println(n);
    i++;
    salir = (i % 7 == 0);
}
```

(4.5) do while

La única diferencia respecto a la anterior está en que la expresión lógica se evalúa después de haber ejecutado las sentencias. Es decir el bucle al menos se ejecuta una vez. Es decir los pasos son:

- (1) Ejecutar sentencias
- (2) Evaluar expresión lógica
- (3) Si la expresión es verdadera volver al paso 1, sino continuar fuera del while

Sintaxis:

```
do {
    instrucciones
} while (expresión lógica)
```

Ejemplo (contar de uno a 1000):

```
int i=0;
do {
    i++;
    System.out.println(i);
} while (i<1000);
```

Ejemplo : Pedir un número por teclado entre el 1 y el 9 y mostrar su tabla de multiplicar

(Ejercicio nº 9)

Crear un programa que acepte como entrada un número entero positivo, y lo muestre convertido a binario : Salida :

El número , en binario es .

bucles con contador

Se llaman así a los bucles que se repiten una serie determinada de veces. Dichos bucles están controlados por un contador (o incluso más de uno). El contador es una variable que va variando su valor (de uno en uno, de dos en dos,... o como queramos) en cada vuelta del bucle. Cuando el contador alcanza un límite determinado, entonces el bucle termina.

En todos los bucles de contador necesitamos saber:

- (1) Lo que vale la variable contadora al principio. Antes de entrar en el bucle
- (2) Lo que varía (lo que se incrementa o decrementa) el contador en cada vuelta del bucle.
- (3) Las acciones a realizar en cada vuelta del bucle
- (4) El valor final del contador. En cuanto se rebasa el bucle termina. Dicho valor se pone como condición del bucle, pero a la inversa; es decir, la condición mide el valor que tiene que tener el contador para que el bucle se repita y no para que termine.

Ejemplo:

```
i=10; /*Valor inicial del contador, empieza valiendo 10  
       (por supuesto i debería estar declarada como entera, int) */  
  
while (i<=200){ /*condición del bucle, mientras i sea menor de  
                  200, el bucle se repetirá, cuando i rebasa este  
                  valor, el bucle termina */  
  
    printf("%d\n",i); /*acciones que ocurren en cada vuelta del bucle  
                      en este caso simplemente escribe el valor  
                      del contador */  
  
    i+=10; /*Variación del contador, en este caso cuenta de 10 en 10*/  
}  
/*Al final el bucle escribe:  
10  
20  
30  
...  
y así hasta 200  
*/
```

Ejemplo 1: Mostrar los números del 1 al 10.
Ejemplo 2: Mostrar los números pares entre 1 y 20.

La instrucción **break** se utiliza para salir del **switch**. De tal modo que si queremos que para un determinado valor se ejecuten las instrucciones de un apartado **case** y sólo las de ese apartado, entonces habrá que finalizar ese **case** con un **break**.

El bloque **default** sirve para ejecutar instrucciones para los casos en los que la expresión no se ajuste a ningún **case**.

Ejemplo 1:

```
switch (diasemana) {  
    case 1:  
        dia="Lunes";  
        break;  
    case 2:  
        dia="Martes";  
        break;  
    case 3:  
        dia="Miércoles";  
        break;  
    case 4:  
        dia="Jueves";  
        break;  
    case 5:  
        dia="Viernes";  
        break;  
    case 6:  
        dia="Sábado";  
        break;  
    case 7:  
        dia="Domingo";  
        break;  
    default:  
        dia="?";  
}
```

Ejemplo 2:

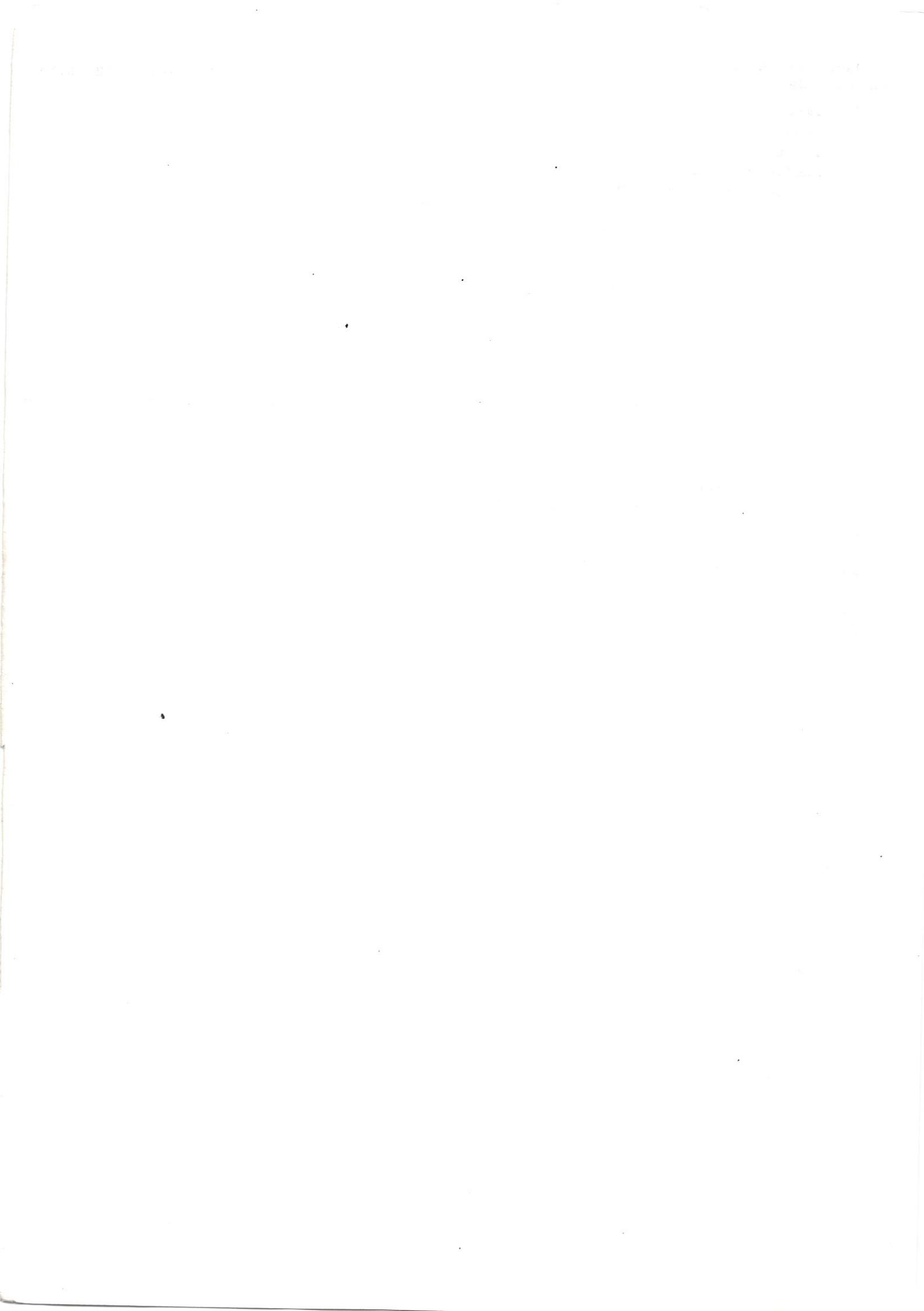
```
switch (diasemana) {  
    case 1:  
    case 2:  
    case 3:  
    case 4:  
    case 5:  
        laborable=true; break;  
    case 6:  
    case 7:  
        laborable=false;  
}
```

Ejercicio nº 6

(10)

Crear un menú:

- 1.- Área del rectángulo.
 - 2.- Área del triángulo.
 - 3.- Área del círculo.
 - 4.- Salir
- El área del _____ es _____ cm².





1990-0000000000000000

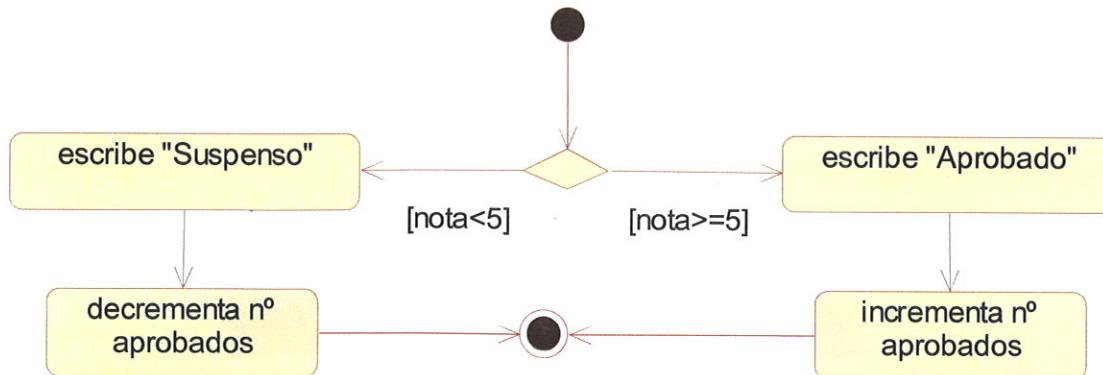


Ilustración 3-2, Diagrama de actividad de una instrucción *if* doble

(4.2.3) anidación

Dentro de una sentencia **if** se puede colocar otra sentencia **if**. A esto se le llama **anidación** y permite crear programas donde se valoren expresiones complejas. La nueva sentencia puede ir tanto en la parte **if** como en la parte **else**.

Las anidaciones se utilizan muchísimo al programar. Sólo hay que tener en cuenta que siempre se debe cerrar primero el último **if** que se abrió. Es muy importante también tabular el código correctamente para que las anidaciones sean legibles.

El código podría ser:

```
if (x==1) {  
    instrucciones  
    ...  
}  
else {  
    if(x==2) {  
        instrucciones  
        ...  
    }  
    else {  
        if(x==3) {  
            instrucciones  
            ...  
        }  
    }  
}
```

Ejemplo 2: Programa que lee un número entre 1 y 1000 y determina si acaba en 1;

Ejemplo 2: Programa que lee una letra y determine si es o no letra mayúscula (8)

(4.1) introducción. expresiones lógicas

Hasta ahora las instrucciones que hemos visto, son instrucciones que se ejecutan secuencialmente; es decir, podemos saber lo que hace el programa leyendo las líneas de izquierda a derecha y de arriba abajo.

Las instrucciones de control de flujo permiten alterar esta forma de ejecución. A partir de ahora habrá líneas en el código que se ejecutarán o no dependiendo de una condición.

Esa condición se construye utilizando lo que se conoce como expresión lógica. Una expresión lógica es cualquier tipo de expresión Java que dé un resultado booleano (verdadero o falso).

Las expresiones lógicas se construyen por medio de variables booleanas o bien a través de los operadores relacionales (`==`, `>`, `<`,...) y lógicos (`&&`, `||`, `!`).

(4.2) if

(4.2.1) sentencia condicional simple

Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de instrucciones en caso de que la expresión lógica sea verdadera. Si la expresión tiene un resultado falso, no se ejecutará ninguna expresión. Su sintaxis es:

```
if(expresión lógica) {  
    instrucciones  
    ...  
}
```

Las llaves se requieren sólo si va a haber varias instrucciones. En otro caso se puede crear el **if** sin llaves:

```
if(expresión lógica) sentencia;
```

Ejemplo:

```
if(nota>=5){  
    System.out.println("Aprobado");  
    aprobados++;  
}
```

La idea gráfica del funcionamiento del código anterior sería: