

(4)

programación

estructurada en

Java

esquema de la unidad

(4.1) introducción. expresiones lógicas	6
(4.2) if	6
(4.2.1) sentencia condicional simple	6
(4.2.2) sentencia condicional compuesta	7
(4.2.3) anidación	8
(4.3) switch	9
(4.4) while	11
(4.5) do while	14
(4.6) for	15
(4.7) sentencias de ruptura de flujo	16
(4.7.1) sentencia break	16
(4.7.2) sentencia continue	17

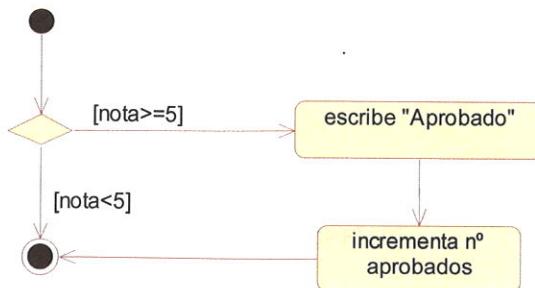


Ilustración 3-1, Diagrama de actividad de la instrucción **if** simple

(4.2.2) sentencia condicional compuesta

Es igual que la anterior, sólo que se añade un apartado **else** que contiene instrucciones que se ejecutarán si la expresión evaluada por el **if** es falsa.
Sintaxis:

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

Como en el caso anterior, las llaves son necesarias sólo si se ejecuta más de una sentencia. Ejemplo de sentencia **if-else**:

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

Ejercicio nº 5 (Pág 9)

Crear un programa en Java que pida tres números enteros por teclado, y los muestre ordenados de menor a mayor:

Entrada: u1, u2 y u3

Salida: u1, u2 y u3.

(7)

La clase estándar Scanner

La clase **Scanner** de Java provee métodos para leer valores de entrada de varios tipos y está localizada en el paquete **java.util**. Los valores de entrada pueden venir de varias fuentes, incluyendo valores que se introduzcan por el teclado o datos almacenados en un archivo.

Para utilizar esa clase tenemos que crear primero un objeto de ella para poder invocar sus métodos. La siguiente declaración crea un objeto, denominado teclado, de la clase Scanner.

```
Scanner teclado = new Scanner(System.in);
```

El propósito de pasar **System.in** como argumento es conectar o establecer una relación entre el objeto tipo **Scanner**, con nombre **teclado** en la declaración anterior, y el objeto **System.in**, que representa el sistema estándar de entrada de información en Java. Si no se indica lo contrario, el teclado es, por omisión, el sistema estándar de entrada de información en Java.

Una vez creado un objeto de la clase **Scanner** asociado al sistema estándar de entrada **System.in**, llamamos, a través del objeto creado(**teclado**) a uno de sus métodos, por ejemplo, al método **nextInt** para introducir un valor del tipo **int**. Para introducir otros valores de otros tipos de datos, se usan los métodos correspondientes.

Método	Ejemplo
nextByte()	byte b = teclado.nextByte();
nextDouble()	double d = teclado.nextDouble();
nextFloat()	float f = teclado.nextFloat();
nextInt()	int i = teclado.nextInt();
nextLong()	long l = teclado.nextLong();
nextShort()	short s = teclado.nextShort();
next()	String p = teclado.next();
nextLine()	String o = teclado.nextLine();

Cómo limpiar el buffer de entrada en Java

Cuando en un programa se leen por teclado datos numéricos y datos de tipo carácter o String debemos tener en cuenta que al introducir los datos y pulsar intro estamos también introduciendo en el buffer de entrada el intro.

Es decir, cuando en un programa introducimos un dato y pulsamos el intro como final de entrada, el carácter intro también pasa al buffer de entrada.

Buffer de entrada si se introduce un 5: 5\n

En esta situación, la instrucción:

```
n = sc.nextInt();
```

Asigna a n el valor 5 pero el intro permanece en el buffer

Buffer de entrada después de leer el entero: \n

Si ahora se pide que se introduzca por teclado una cadena de caracteres:

```
System.out.print("Introduzca su nombre: ");
nombre = sc.nextLine(); //leer un String
```

El método nextLine() extrae del buffer de entrada todos los caracteres hasta llegar a un intro y elimina el intro del buffer.

En este caso asigna una cadena vacía a la variable nombre y limpia el intro. Esto provoca que el programa no funcione correctamente, ya que no se detiene para que se introduzca el nombre.

Solución:

Se debe **limpiar el buffer** de entrada si se van a leer datos de tipo carácter a continuación de la lectura de datos numéricos.

La forma más sencilla de limpiar el buffer de entrada en Java es ejecutar la instrucción:

```
sc.nextLine();
```

```
1 import java.util.*;
2 import java.io.*;
3 import javax.swing.*;
4 public class leer{
5     public static void main(String[] args){
6         //Con clase Scanner del paquete java.util
7         Scanner texto=new Scanner(System.in);
8         System.out.print ("Introduce nombre: ");
9         String nombre1=texto.nextLine();
10        System.out.println(nombre1);
11
12        //Con clase BufferedReader del paquete java.io
13        String nombre2="";
14        System.out.print( "Introduzca su nombre: " );
15        try {
16            BufferedReader entrada = new BufferedReader(new InputStreamReader(
17                System.in));
18            nombre2 = entrada.readLine();
19        }
20        catch (IOException e) {}
21        System.out.println(nombre2);
22
23        //Con clase JOptionPane del paquete javax.swing
24        String nombre3=JOptionPane.showInputDialog("Introduce otro");
25        System.out.println(nombre3);
26    }
27}
```

Hoja de la clase Scanner (después de switch)

1^{er} curso de administración de sistemas informáticos
autor: Jorge Sánchez - www.jorgesanchez.net

Una forma más legible de escribir ese mismo código sería:

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

dando lugar a la llamada instrucción **if-else-if**.



Operador ? (if de una linea) pag 39. TEMA 03
~~variable = (expresión lógica) ? valor si verdadero : valor si falso;~~

(4.3) switch

EJERCICIO n°5

(pag 7)

Se la llama **estructura condicional compleja** porque permite evaluar varios valores a la vez. En realidad sirve como sustituta de algunas expresiones de tipo **if-else-if**. Sintaxis:

```
switch (expresiónEntera) {  
    case valor1:  
        instrucciones del valor 1  
        [break]  
    case valor2:  
        instrucciones del valor 1  
        [break]  
    [  
    .  
    .  
    .  
    ]  
    default:  
        instrucciones que se ejecutan si la expresión no toma  
        ninguno de los valores anteriores  
        ..]  
}
```

Estructuras de Java

Estructuras de uso de

switch

Esta instrucción evalúa una expresión (que debe ser **short, int, byte o char**), según el valor de la misma ejecuta instrucciones. Cada **case** contiene un valor de la expresión; si efectivamente la expresión equivale a ese valor, se ejecutan las instrucciones de ese case y de los siguientes.

String

Ejercicio n° 7

Crear un programa que muestre el número de días que
se tarda en leer un libro de un determinado autor,
según se lea a un ritmo constante de páginas por hora.
Introducir los datos y mostrar el resultado.

1º curso de administración de sistemas informáticos
autor: Jorge Sánchez - www.jorgesanchez.net

La clase Scanner

(4.4) while

La instrucción **while** permite crear bucles. Un bucle es un conjunto de sentencias que se repiten mientras se cumpla una determinada condición. Los bucles **while** agrupan instrucciones las cuales se ejecutan continuamente hasta que una condición que se evalúa sea falsa.

La condición se mira antes de entrar dentro del while y cada vez que se termina de ejecutar las instrucciones del while

Sintaxis:

```
while (expresión lógica) {  
    sentencias que se ejecutan si la condición es true  
}
```

El programa se ejecuta siguiendo estos pasos:

- (1) Se evalúa la expresión lógica
- (2) Si la expresión es verdadera ejecuta las sentencias, sino el programa abandona la sentencia **while**
- (3) Tras ejecutar las sentencias, volvemos al paso 1

Ejemplo (escribir números del 1 al 100):

```
int i=1;  
while (i<=100){  
    System.out.println(i);  
    i++;  
}
```

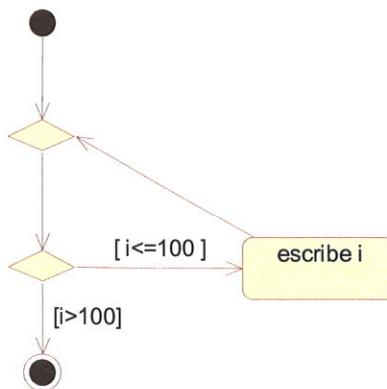


Ilustración 3-3, Diagrama UML de actividad del bucle while

Ejemplo Centinela: Programa que manda numeros al azar entre 1 y 50, hasta que uno de ellos sea múltiplo de 5.

1º curso de administración de sistemas informáticos
autor: Jorge Sánchez - www.jorgesanchez.net

Bucles de centinela

Es el segundo tipo de bucle básico. Una condición lógica llamada **centinela**, que puede ser desde una simple variable booleana hasta una expresión lógica más compleja, sirve para decidir si el bucle se repite o no. De modo que cuando la condición lógica se incumpla, el bucle termina.

Esa expresión lógica a cumplir es lo que se conoce como centinela y normalmente la suele realizar una variable booleana.

Ejemplo:

```
boolean salir=false; /* En este caso el centinela es una variable booleana que inicialmente vale falso */
int n;
while(salir==false){ /* Condición de repetición: que salir siga siendo falso. Ese es el centinela.
También se podía haber escrito simplemente:
while(!salir)
*/
    n=(int)Math.floor(Math.random()*500+1); // Lo que se repite en el
    System.out.println(i); // bucle: calcular un número
    aleatorio de 1 a 500 y escribirlo */
    salir=(i%7==0); /* El centinela vale verdadero si el número es
múltiplo de 7
*/
}
```

Comparando los bucles de centinela con los de contador, podemos señalar estos puntos:

- ◆ Los bucles de contador se repiten un número concreto de veces, los bucles de centinela no
- ◆ Un bucle de contador podemos considerar que es seguro que finalice, el de centinela puede no finalizar si el centinela jamás varía su valor (aunque, si está bien programado, alguna vez lo alcanzará)
- ◆ Un bucle de contador está relacionado con la programación de algoritmos basados en series.

Un bucle podría ser incluso mixto: de centinela y de contador. Por ejemplo imaginar un programa que escriba números de uno a 500 y se repita hasta que llegue un múltiplo de 7, pero que como mucho se repite cinco veces.

Ejercicio nº 8

Crear un programa que adivine un numero entre 1 y 20 generado al azar. Tras cada intento, fallido, dese. Solicitar un numero con las pistas Mayor/Menor. Cuando se acierte el numero, mostrará el mensaje:

Correcto en _____ intentos. (13)

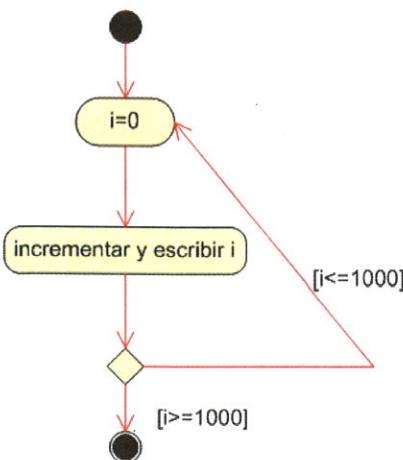


Ilustración 3-4, Diagrama de actividad del bucle do..while

Se utiliza cuando sabemos al menos que las sentencias del bucle se van a repetir una vez (en un bucle **while** puede que incluso no se ejecuten las sentencias que hay dentro del bucle si la condición fuera falsa, ya desde un inicio).

De hecho cualquier sentencia **do..while** se puede convertir en while. El ejemplo anterior se puede escribir usando la instrucción while, así:

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

(4.6) for

Es un bucle más complejo especialmente pensado para llenar arrays o para ejecutar instrucciones controladas por un contador. Una vez más se ejecutan una serie de instrucciones en el caso de que se cumpla una determinada condición. Sintaxis:

```
for(inicialización;condición;incremento){  
    sentencias  
}
```

Las sentencias se ejecutan mientras la condición sea verdadera. Además antes de entrar en el bucle se ejecuta la instrucción de inicialización y en cada vuelta se ejecuta el incremento. Es decir el funcionamiento es:

- (1) Se ejecuta la instrucción de inicialización

(15)

Ejemplo 3: Muestra, separados por comas, los números entre 1 y 100.
Muestra, separados por coma, los nú-

(4.7.2) sentencia continue

Es parecida a la anterior, sólo que en este caso en lugar de abandonar el bucle, lo que ocurre es que no se ejecutan el resto de sentencias del bucle y se vuelve a la condición del mismo:

```
for(int i=1;i<=1000;i++){  
    if(i%3==0) continue;  
    System.out.println(i);  
}
```

En ese listado aparecen los números del 1 al 1000, menos los múltiplos de 3 (en los múltiplos de 3 se ejecuta la instrucción **continue** que salta el resto de instrucciones del bucle y vuelve a la siguiente iteración).

El uso de esta sentencia genera malos hábitos, siempre es mejor resolver los problemas sin usar **continue**. El ejemplo anterior sin usar esta instrucción quedaría:

```
for(int i=1;i<=1000;i++){  
    if(i%3!=0) System.out.println(i);  
}
```

La programación estructurada prohíbe utilizar las sentencias **break** y **continue**

Ejemplo: Mostrar los numeros pares comprendidos entre 1 y 30.

Ejercicio nº 11

Programa que muestre una pirámide con hielo con asteriscos. El numero de filas se introducirá por teclado.

Ejercicio nº 12

Programa que muestra un rombo, cuya diagonal se introduzca por teclado.

printf (Hoja)

Ejemplo 1: Introducir por teclado un numero float, y mostrar el numero y su cuadrado, ambos con dos decimales: El cuadrado de n es +

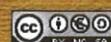
(17)

Ejemplo 2: Introducir por teclado tu nombre y tu edad en cm., mostrando como salida:

Me llamo Nombre en mayusculas, y mi edad es edad

Unidad 5: Estructuras básicas de datos en el lenguaje Java

**Fundamentos de Programación.
1º de ASI**



Esta obra está bajo una licencia de Creative Commons.
Autor: Jorge Sánchez Asenjo (año 2009) <http://www.jorgesanchez.net>
e-mail:info@jorgesanchez.net

Esta obra está bajo una licencia de Reconocimiento-NoComercial-CompartirIgual de Creative Commons
Para ver una copia de esta licencia, visite:
<http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es>
o envíe una carta a:
Creative Commons, 559 Nathan Abbot



Reconocimiento-NoComercial-CompartirIgual 2.5 España

Usted es libre de:



copiar, distribuir y comunicar públicamente la obra

hacer obras derivadas

Bajo las condiciones siguientes:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

No comercial. No puede utilizar esta obra para fines comerciales.

Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Apart from the remix rights granted under this license, nothing in this license impairs or restricts the author's moral rights.

Advertencia

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
Esto es un resumen legible por humanos del texto legal (la licencia completa) disponible en los idiomas siguientes:
Catalán Castellano Euskera Gallego

Para ver una copia completa de la licencia, acudir a la dirección <http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es>

(5) estructuras básicas de datos en Java

esquema de la unidad

(5.1) arrays	5
(5.1.1) introducción.....	5
(5.1.2) unidimensionales	6
(5.1.3) arrays multidimensionales	11
(5.1.4) longitud de un array	12
(5.1.5) la clase Arrays	12
(5.1.6) el método System.arraycopy	14
(5.2) clase String	14
(5.2.1) introducción. declarar e iniciar textos.....	14
(5.2.2) comparación entre objetos String	15
(5.2.3) String.valueOf.....	16
(5.2.4) métodos de los objetos String	16
(5.3) arrays de Strings	22
(5.4) parámetros de la línea de comandos	22

(5.1) arrays

(5.1.1) introducción

Los tipos de datos que conocemos hasta ahora no permiten solucionar problemas que requieren gestionar muchos datos a la vez. Por ejemplo, imaginemos que deseamos leer las notas de una clase de 25 alumnos. Desearemos por tanto almacenarlas y para ello, con lo que conocemos hasta ahora, no habrá más remedio que declarar 25 variables.

Eso es tremadamente pesado de programar. Manejar esos datos significaría estar continuamente manejando 25 variables. Por ello en casi todos los lenguajes se pueden agrupar una serie de variables del mismo tipo en

Declaración de Arrays.pdf

1º curso de administración de sistemas informáticos
autor: Jorge Sánchez - www.jorgesanchez.net

índice encerrado entre corchetes nos permite acceder a cada elemento individual.

La **declaración de un array unidimensional** se hace con esta sintaxis.

tipo nombre[];

tipo[] nombre;

Ejemplo:

```
double cuentas[]; //Declara un array que almacenará valores  
//doubles
```

Declara un array de tipo double. Esta declaración indica para qué servirá el array, pero no reserva espacio en la RAM al no saberse todavía el tamaño del mismo. En este sentido hay una gran diferencia entre Java y la mayoría de lenguajes clásicos. No es necesario conocer el tamaño del array en el momento de la declaración. Simplemente se avisa de que aparecerá un futuro array.

Tras la declaración del array, se tiene que **iniciar**. Eso lo realiza el operador **new**, que es el que realmente crea el array indicando un tamaño. Cuando se usa new es cuando se reserva el espacio necesario en memoria. **Un array no inicializado no se puede utilizar en el código.**

Ejemplo de iniciación de un array:

```
int notas[]; //sería válido también int[] notas;  
notas = new int[3]; //indica que el array constará de tres  
//valores de tipo int
```

También se puede hacer ambas operaciones en la misma instrucción:

```
int notas[] = new int[3];
```

En el ejemplo anterior se crea un array de tres enteros (con los tipos básicos se crea en memoria el array y se inicializan los valores, los números se inician a 0).

Los valores del array se asignan utilizando el índice del mismo entre corchetes:

```
notas[2]=8;
```

También se pueden asignar valores al array en la propia declaración:

```
int notas[] = {8, 7, 9};  
int notas2[] = new int[] {8, 7, 9}; //Equivalente a la anterior
```

Esto declara e inicializa un array de tres elementos. En el ejemplo lo que significa es que **notas[0]** vale **8**, **notas[1]** vale **7** y **notas[2]** vale **9**.

```
1 public class DeclaracionDeArrays {
2     public static void main (String args[]) {
3         // Opcion 1, todos los elementos inicializados a 0
4         int notas[];
5         notas=new int[3];
6         // o tambien
7         int notas[] = new int[3];
8
9         // Opcion 2, elementos inicializados con valores concretos
10        int notas[];
11        notas=new int[]{1,2,3};
12        // o tambien
13        int notas[] = new int[]{1,2,3};
14        // o tambien
15        int notas[] = {1,2,3};
16
17        //En cualquiera de los casos, mostrará el valor del tercer elemento del Array
18        System.out.println(notas[2]);
19    }
20
21
22
```

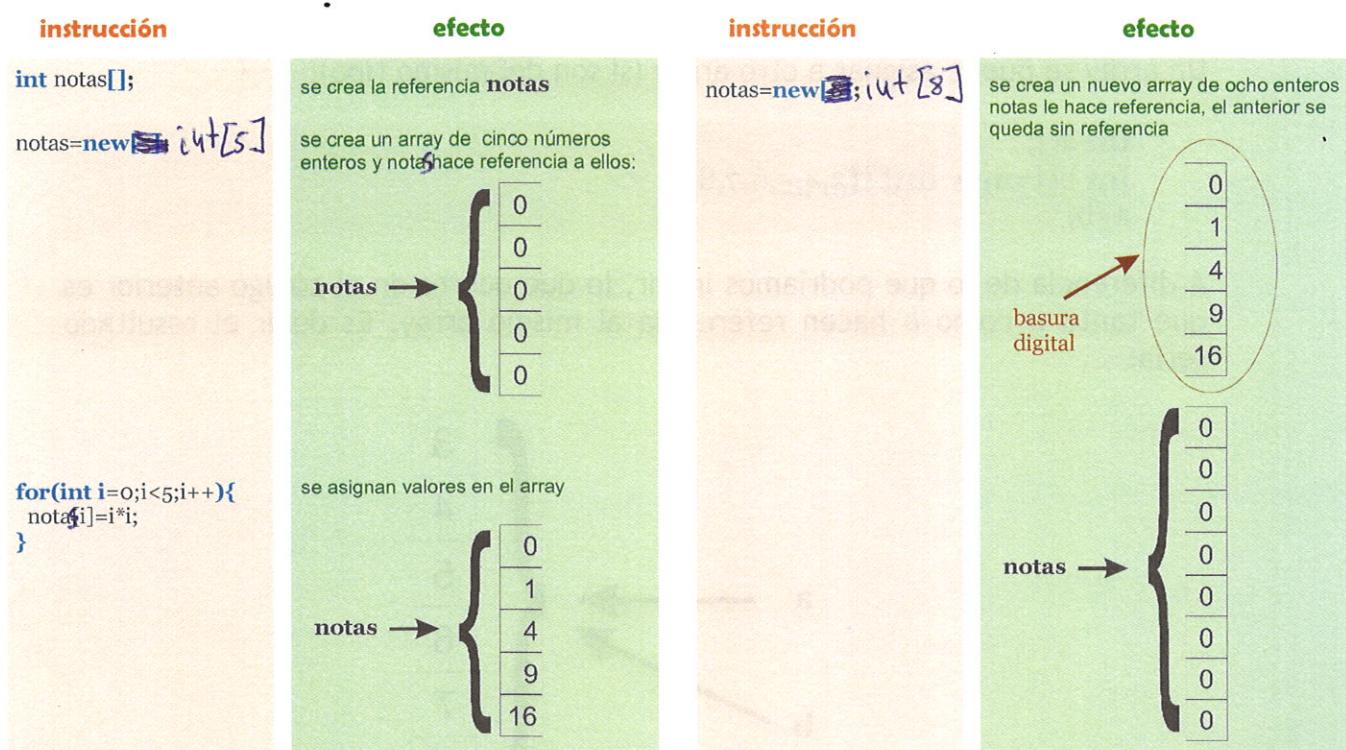


Ilustración 4-2, efecto del uso del operador **new** en los arrays

La imagen anterior trata de explicar el funcionamiento del operador **new** (no sólo para los arrays). Utilizar dos veces el operador **new** con la misma referencia de array, provoca que haya valores que se quedan **sin referencia**. A eso se le suele llamar **basura digital**, es espacio que se consume en la memoria y que no se puede utilizar.

En Java un recolector de basura se encarga cada cierto tiempo de eliminar la basura (a diferencia de lo que ocurre en lenguajes menos seguros como C). Desde el código podemos forzar la recolección de basura mediante:

System.gc();

Aunque dicha instrucción no garantiza la recolección de basuras. Sólo se suele utilizar cuando nuestro código genera basura más rápidamente de lo que el recolector puede eliminar.

Nota:
 - Se pueden recoger datos desde el teclado con:
 String dato = System.console().readLine();
 Clase console que se encuentra dentro del paquete
 java.io.*

Ejercicio nº 21

Crea dos arrays del mismo tamaño, el tabulado por un número de nodos que se indican. Puedes usar el mismo array con letras minúsculas separadas, alternativamente, y el segundo con letras mayúsculas separadas entre el rango correspondiente al único de las tres minúsculas. Muestra ambos arrays y las posiciones donde haya coincidencia entre ambos.

1º curso de administración de sistemas informáticos

autor: Jorge Sánchez - www.jorgesanchez.net

(5.1.3) arrays multidimensionales:

Los arrays además pueden tener varias dimensiones. Entonces se habla de arrays de arrays (arrays que contienen arrays). Un uso podría ser representar datos identificables por más de un índice. Por ejemplo:

```
int nota[][];
```

nota es un array que contiene arrays de enteros. La primera dimensión podría significar el número de un aula y el segundo el número del alumno. De modo que, por ejemplo, **nota[2][7]** significaría la nota del alumno número ocho del tercer aula.

Como ocurre con los arrays unidimensionales, hasta que no se define el array con el operador **new**, no se puede utilizar el array en el código.

¡¡¡**HAY QUE DIMENSIONARLOS Y INICIALIZARLOS !!!**

```
notas = new int[3][12]; //notas está compuesto por 3 arrays  
//de 12 enteros cada uno  
notas[0][0]=9; //el primer valor es un 9
```

Los arrays multidimensionales se pueden inicializar de forma más creativa incluso. Ejemplo:

```
int notas[][]=new int[5][]; //Hay 5 arrays de enteros (aún por definir)  
notas[0]=new int[100]; //El primer array es de 100 enteros  
notas[1]=new int[230]; //El segundo de 230  
notas[2]=new int[400]; //...  
notas[3]=new int[100];  
notas[4]=new int[200];
```

Hay que tener en cuenta que en el ejemplo anterior, **notas[0]** es un array de **100** enteros. Mientras que **notas**, es un array de **5** arrays de enteros. Esta forma irregular de arrays da muchas posibilidades y permite optimizar el gasto de espacio de almacenamiento; la idea es recordar que los arrays multidimensionales en Java son manejados como arrays de arrays.

Se pueden utilizar más de dos dimensiones si es necesario. Por ejemplo:

```
int nota[][][] = new int[5][7][90];
```

Esta instrucción crea cinco arrays de siete arrays de noventa números enteros. Es un array de arrays de arrays (un tanto complicado de describir) o también un array tridimensional. Es raro necesitar más de tres dimensiones, pero sería posible hacerlo.

for para lectura de matrices :
~~int array[J][J] = new int[J][J];~~ {0,1,2,3,4}
~~for (int array0[J]: array)~~

(11)
for (int ~~valores~~: array)
System.out.println (valores);

(Muestra 0 1 0 1 2 3)

equals

Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores. A diferencia del operador de igualdad (`==`), este operador sí compara el contenido. Ejemplo (comparación entre el operador de igualdad y el método `equals`):

```
int a[] = {2,3,4,5,6};  
int b[] = {2,3,4,5,6};  
int c[] = a;  
System.out.println(a == b); //false  
System.out.println(Arrays.equals(a,b)); //true  
System.out.println(a == c); //true  
System.out.println(Arrays.equals(a,c)); //true
```

sort

Permite ordenar un array en orden ascendente. Se pueden ordenar todo el array o bien desde un elemento a otro:

```
int x[] = {4,5,2,3,7,8,2,3,9,5};  
Arrays.sort(x,2,5); //El array queda {4 5 2 3 7 8 2 3 9 5}  
Arrays.sort(x); //Estará completamente ordenado
```

binarySearch

Permite buscar un elemento de forma ultrarrápida en un array ordenado (en un array desordenado sus resultados son impredecibles). Devuelve el índice en el que está colocado el elemento. Ejemplo:

```
int x[] = {1,2,3,4,5,6,7,8,9,10,11,12};  
Arrays.sort(x);  
System.out.println(Arrays.binarySearch(x,8)); //Escribe: 7
```

copyOf

Disponible desde la versión 1.6 del SDK, obtiene una copia de un array. Recibe dos parámetros: el primero es el array a copiar y el segundo el tamaño que tendrá el array resultante. De modo que si el tamaño es menor que el del array original, sólo obtiene copia de los primeros elementos (tantos como indique el tamaño); si el tamaño es mayor que el original, devuelve un array en el que los elementos que superan al original se rellenan con ceros o con datos de tipo `null` (dependiendo del tipo de datos del array).

```
int a[] = {1,2,3,4,5,6,7,8,9};  
int b[] = Arrays.copyOf(a, a.length); //b es {1,2,3,4,5,6,7,8,9}  
int c[] = Arrays.copyOf(a, 12); //c es {1,2,3,4,5,6,7,8,9,0,0,0}  
  
int d[] = Arrays.copyOf(a, 3); //d es {1,2,3}
```

```
1 import java.util.*;
2 public class Prueba {
3     public static void main (String args[]) {
4         int array1[][]=new int[2][];
5         array1[0]=new int[2];
6         array1[1]=new int[3];
7         int i=0;
8         for (int valores1[]:array1)
9             System.out.println(Arrays.toString(valores1));
10    }
11 }
12
13
```

Métodos de la clase Arrays. (Resumen)

Nombre	Descripción	Parámetros	Dato devuelto
binarySearch	Busca un valor que le pasamos por parámetro, devuelve su posición. Debe estar ordenado.	Un array y un valor. Los dos del mismo tipo. Estos pueden ser un byte, char, double, float, int, long, short u objeto.	int Si existe el valor que se busca.
copyOf	Copia un array y lo devuelve en un nuevo array.	Un array y la longitud. Si se pasa del tamaño del array original, rellena con ceros las posiciones sobrantes. Estos pueden ser un byte, char, double, float, int, long, short u objeto.	array del mismo tipo que se introduce.
copyOfRange	Copia un array y lo devuelve en un nuevo array. Le indicamos la posición de origen y la longitud.	Un array, posición origen y destino. Estos pueden ser un byte, char, double, float, int, long, short u objeto.	array del mismo tipo que se introduce.
equals	Indica si dos arrays son iguales.		true o false
fill	Rellena un array con un valor que le indiquemos como parámetro.	Un array y el valor a llenar. Estos pueden ser un byte, char, double, float, int, long, short u objeto.	No devuelve nada
sort	Ordena el array.	Estos pueden ser un byte, char, double, float, int, long, short u objeto.	No devuelve nada
toString	Muestra el contenido del array pasado como parámetros	Un array. Estos pueden ser un byte, char, double, float, int, long, short u objeto.	Devuelve una cadena con el contenido del array. [valor, valor, ...]

Ejercicio n°23

Una empresa que se dedica a la venta de cosméticos necesita un
programa para gestionar facturas. En cada factura debe figurar: código
de artículo, cantidad en litros vendidos y precio por
litro. Se vende, tras introducir las facturas: cantidad de litros vendidos y
número de facturas de más de 600 €.

1º curso de administración de sistemas informáticos

autor: Jorge Sánchez - www.jorgesanchez.net

Otras formas de También se pueden crear objetos String sin utilizar constantes
entrecomilladas, usando otros constructores.

Por ejemplo podemos utilizar un array de caracteres, como por ejemplo:

```
char[] palabra = {'P','a','l','a','b','r','a'}; //palabra es un array de  
caracteres  
  
String //no es lo mismo que un
```

a partir de ese array de caracteres podemos crear un String:

```
String cadena = new String(palabra); //mediante el operador new  
//podemos convertir el array de  
caracteres en  
//un String
```

De forma similar hay posibilidad de convertir un array de bytes en un String. En este caso se tomará el array de bytes como si contuviera códigos de caracteres, que serán traducidos por su carácter correspondiente al ser convertido a String. Hay que indicar la tabla de códigos que se utilizará (si no se indica se entiende que utilizamos el código ASCII):

```
byte[] datos = {97,98,99};  
String codificada = new String(datos, "8859_1");
```

En el último ejemplo la cadena **codificada** se crea desde un array de tipo byte que contiene números que serán interpretados como códigos Unicode. Al asignar el valor 8859_1 indica la tabla de códigos a utilizar.

METODOS de los objetos String (String.metodo(argumentos)) (5.2.2) comparación entre objetos String

Los objetos String (como ya ocurría con los arrays) no pueden compararse directamente con los operadores de comparación. En su lugar se deben utilizar estas expresiones:

- ◆ **cadena1.equals(cadena2)**. El resultado es **true** si la **cadena1** es igual a la **cadena2**. Ambas cadenas son variables de tipo **String**.
- ◆ **cadena1.equalsIgnoreCase(cadena2)**. Como la anterior, pero en este caso no se tienen en cuenta mayúsculas y minúsculas. Por cierto en cuestiones de mayúsculas y minúsculas, los hispanohablantes (y el resto de personas del planeta que utilice otro idioma distinto al inglés) podemos utilizar esta función sin temer que no sea capaz de manipular correctamente caracteres como la eñe, las tildes,... Funciona correctamente con cualquier símbolo alfabético de cualquier lengua presente en el código **Unicode**.
- ◆ **s1.compareTo(s2)**. Compara ambas cadenas, considerando el orden alfabético. Si la primera cadena es mayor en orden alfabético que la segunda devuelve **1**, si son iguales devuelve **0** y si es la segunda la

charAt

Devuelve un carácter de la cadena. El carácter a devolver se indica por su posición (el primer carácter es la posición 0) Si la posición es negativa o sobrepasa el tamaño de la cadena, ocurre un error de ejecución (se pararía el programa), una excepción tipo **IndexOutOfBoundsException**. Ejemplo:

```
String s1="Prueba";
char c1=s1.charAt(2); //c1 valdrá 'u'
```

substring

Da como resultado una porción del texto de la cadena. La porción se toma desde una posición inicial hasta una posición final (sin incluir esa posición final). Si las posiciones indicadas no son válidas ocurre una excepción de tipo **IndexOutOfBoundsException**. Se empieza a contar desde la posición 0. Ejemplo:

```
String s1="Buenos días";
String s2=s1.substring(7,10); //s2 = "día"
```

indexOf

Devuelve la primera posición en la que aparece un determinado texto en la cadena. En el caso de que la cadena buscada no se encuentre, devuelve -1. El texto a buscar puede ser **char** o **String**. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";
System.out.println(s1.indexOf("que")); //Escribe: 15
```

Se puede buscar desde una determinada posición. En el ejemplo anterior:

```
System.out.println(s1.indexOf("que",16)); //Ahora escribe: 26
```

lastIndexOf

Devuelve la última posición en la que aparece un determinado texto en la cadena. Es casi idéntica a la anterior, sólo que busca desde el final. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";
System.out.println(s1.lastIndexOf("que")); //Escribe: 26
```

También permite comenzar a buscar desde una determinada posición.

endsWith

Devuelve **true** si la cadena termina con un determinado texto. Ejemplo:

```
String s1="Quería decirte que quiero que te vayas";
System.out.println(s1.endsWith("vayas")); //Escribe true
```

matches

Es una función muy interesante disponible desde la versión 1.4. Examina la expresión regular que recibe como parámetro (en forma de String) y devuelve verdadero si el texto que examina cumple la expresión regular.

Una expresión regular es una expresión textual que utiliza símbolos especiales para hacer búsquedas avanzadas.

Las expresiones regulares pueden contener:

- ◆ **Caracteres.** Como `a`, `s`, `\n`,... y les interpreta tal cual. Si una expresión regular contuviera sólo un carácter, **matches** devolvería verdadero si el texto contiene sólo ese carácter. Si se ponen varios, obliga a que el texto tenga exactamente esos caracteres.
- ◆ **Caracteres de control** (`\n`,`\r`,...)
- ◆ **Opciones de caracteres.** Se ponen entre corchetes. Por ejemplo `[abc]` significa `a`, `b` ó `c`.
- ◆ **Negación de caracteres.** Funciona al revés impide que aparezcan los caracteres indicados. Se pone con corchetes dentro de los cuales se pone el carácter circunflejo (^). `[^abc]` significa ni `a` ni `b` ni `c`.
- ◆ **Rangos.** Se ponen con guiones. Por ejemplo `[a-z]` significa: cualquier carácter de la `a` la `z`.
- ◆ **Intersección.** Usa `&&`. Por ejemplo `[a-x&&r-z]` significa de la `r` a la `x` (intersección de ambas expresiones).
- ◆ **Substracción.** Ejemplo `[a-x&&[^cde]]` significa de la `a` la `x` excepto la `c`, `d` ó `e`.
- ◆ **Cualquier carácter. Se hace con el símbolo punto (.)**
- ◆ **Opcional.** El símbolo `?` sirve para indicar que la expresión que le antecede puede aparecer una o ninguna veces. Por ejemplo `a?` indica que puede aparecer la letra `a` o no.
- ◆ **Repetición.** Se usa con el asterisco (*). Indica que la expresión puede repetirse varias veces o incluso no aparecer.
- ◆ **Repetición obligada.** Lo hace el signo `+`. La expresión se repite una o más veces (pero al menos una).
- ◆ **Repetición un número exacto de veces.** Un número entre llaves indica las veces que se repite la expresión. Por ejemplo `\d{7}` significa que el texto tiene que llevar siete números (siete cifras del 0 al 9). Con una coma significa **al menos**, es decir `\d{7,}` significa **al menos** siete veces (podría repetirse más veces). Si aparece un segundo número indica un máximo número de veces `\d{7,10}` significa de siete a diez veces.

Método String matches de Java

Hola a todos, hoy os explicare como usar el método matches de String.

El método matches de String nos permite comprobar si un String cumple una expresión regular pasado como parámetro. Si es cierta devuelve true, sino false.

Una expresión regular es una expresión textual que utiliza símbolos especiales para hacer búsquedas avanzadas.

Las expresiones pueden contener:

- **Caracteres.**
- **Caracteres de control**, por ejemplo, \s, \d, etc. Recuerda que añadir un \ mas al introducirlo en una cadena. Los mas usados son:
 - \d , dígito, es igual que [0-9]
 - \D , no dígito, es igual que [^0-9]
 - \s , carácter en blanco, es igual que [\t\n\x0B\f\r]
 - \S , no carácter en blanco, es igual que [^\s]
 - \w , carácter alfanumérico, es igual que [a-zA-Z_0-9]
 - \W , no carácter alfanumérico, es igual que [^\w]
- **Opciones de caracteres**, se usa el corchete. Por ejemplo, [afgd] significa que puede contener a, f, g o d.
- **Negación de caracteres**, funciona al revés que el anterior, se usa ^. Por ejemplo, [^afgd]
- **Rangos**, se usa para que incluya un rango de caracteres. Por ejemplo, para que incluya los caracteres entre a y z [a-z]
- **Intersección**: permite unir dos condiciones, es como el operador &&.
- **Cualquier carácter**: se usa un punto.
- **Opcional**: se usa el símbolo ?, indica que un carácter puede o no aparecer.
- **Repetición**: se usa el símbolo *, indica que un conjunto de caracteres se pueden repetir o no.
- **Repetición obligada**: se usa el símbolo +, es como el anterior pero debe aparecer mínimo una vez.
- **Repetición un número exacto de veces**: después de una expresión abrimos llaves {} con un número dentro, indica el numero de veces que debe repetirse un carácter o expresión. Si después del numero escribimos una coma, indica que debe repetirse como mínimo el número que indiquemos y como máximo los que queramos. Si después de la coma escribimos un número, indica que debe repetirse entre los números que le indiquemos como si fuera un rango.

```
1 import java.util.*;
2 public class DeclararStrings {
3     public static void main (String args[]) {
4         //Modo 1
5         String texto1="Texto";
6
7         //Modo 2
8         char letras[]={ 'T', 'e', 'x', 't', 'o' };
9         String texto2=new String(letras);
10
11        //Modo 3
12        byte numeros[]={84,101,120,116,111};
13        String texto3=new String(numeros);
14
15        //Salida de cada uno
16        System.out.println(texto1);
17        System.out.println(texto2);
18        System.out.println(texto3);
19    }
20}
21
22
```

método	descripción
<code>void getBytes(int srcBegin, int srcEnd, char[] dest, int dstBegin);</code>	Almacena el contenido de la cadena en el array de caracteres <code>dest</code> . Toma los caracteres desde la posición <code>srcBegin</code> hasta la posición <code>srcEnd</code> y les copia en el array desde la posición <code>dstBegin</code>
<code>int indexOf(String s)</code>	Devuelve la posición en la cadena del texto <code>s</code>
<code>int indexOf(String s, int primeraPos)</code>	Devuelve la posición en la cadena del texto <code>s</code> , empezando a buscar desde la posición <code>PrimeraPos</code>
<code>int lastIndexOf(String s)</code>	Devuelve la última posición en la cadena del texto <code>s</code>
<code>int lastIndexOf(String s, int primeraPos)</code>	Devuelve la última posición en la cadena del texto <code>s</code> , empezando a buscar desde la posición <code>PrimeraPos</code>
<code>int length()</code>	Devuelve la longitud de la cadena
<code>boolean matches(String expReg)</code>	Devuelve verdadero si el String cumple la expresión regular
<code>String replace(char carAnterior, char ncarNuevo)</code>	Devuelve una cadena idéntica al original pero que ha cambiando los caracteres iguales a <code>carAnterior</code> por <code>carNuevo</code>
<code>String replaceFirst(String str1, String str2)</code>	Cambia la primera aparición de la cadena <code>str1</code> por la cadena <code>str2</code>
<code>String replaceFirst(String str1, String str2)</code>	Cambia la primera aparición de la cadena uno por la cadena dos
<code>String replaceAll(String str1, String str2)</code>	Cambia la todas las apariciones de la cadena uno por la cadena dos
<code>String startsWith(String s)</code>	Devuelve true si la cadena comienza con el texto <code>s</code> .
<code>String substring(int primeraPos, int segundaPos)</code>	Devuelve el texto que va desde <code>primeraPos</code> a <code>segundaPos</code> .
<code>char[] toCharArray()</code>	Devuelve un array de caracteres a partir de la cadena dada
<code>String toLowerCase()</code>	Convierte la cadena a minúsculas
<code>String toLowerCase(Locale local)</code>	Lo mismo pero siguiendo las instrucciones del argumento <code>local</code>
<code>String toUpperCase()</code>	Convierte la cadena a mayúsculas
<code>String toUpperCase(Locale local)</code>	Lo mismo pero siguiendo las instrucciones del argumento <code>local</code>
<code>String trim()</code>	Elimina los blancos que tenga la cadena tanto por delante como por detrás
<code>static String valueOf(tipo elemento)</code>	Devuelve la cadena que representa el valor <code>elemento</code> . Si elemento es booleano, por ejemplo devolvería una cadena con el valor <code>true</code> o <code>false</code>

Ejercicio u'24)

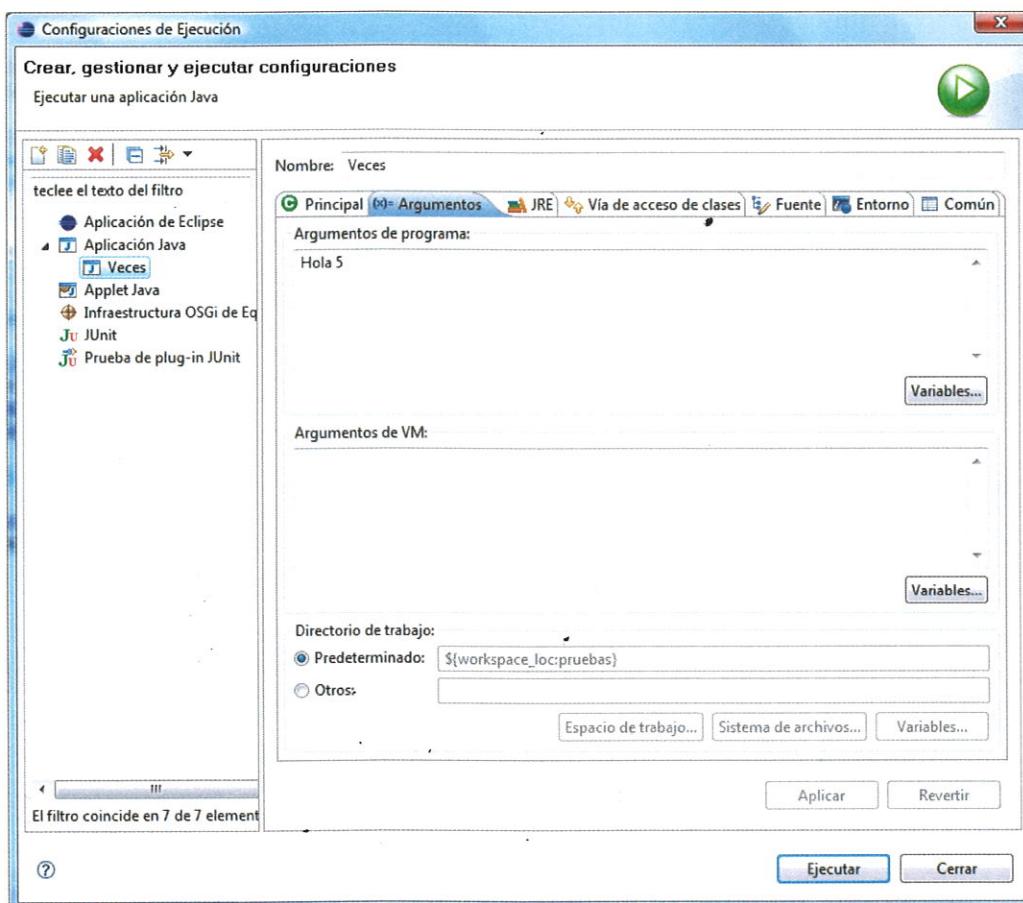
Programa que indique los caracteres y cuantos de veces que aparece cada uno, en un texto introducido por teclado.

(21)

Así el programa **Veces** comentado antes quedaría:

```
public class Veces {  
    public static void main(String[] args) {  
        if(args.length>=2) {  
            int tam=Integer.parseInt(args[1]);  
            for (int i = 1; i <=tam; i++) {  
                System.out.println(args[0]);  
            }  
        } else {  
            System.out.println("Faltan parámetros");  
        }  
    }  
}
```

En casi todos los IDE se pueden configurar los parámetros sin necesidad de tener que ejecutar el programa desde la línea de comandos. Por ejemplo en Eclipse se realiza desde el menú **Ejecutar-Configuración de ejecución**:



```
1 import java.util.*;
2 public class MetodosStrings {
3     public static void main (String args[]) {
4         String texto1="Texto para hacer las pruebas";
5         String texto2="texto para hacer las pruebas";
6         char tabla1[]={'P','r','u','e','b','a'};
7         String texto3="Texto";
8         char tabla2 []=new char[10];
9         String texto4="Txt";
10        String texto5="En"+'\n'+"dos lineas";
11        System.out.println(texto5);
12        String texto6="1a2b3c4d5e66f";
13        int valor1=1234;
14
15     //Salida de cada método
16     System.out.println(texto1.charAt(2)); // x
17     System.out.println(texto1.compareTo(texto2));// -1 ó negativo
18     System.out.println(texto1.compareToIgnoreCase(texto2));// 0
19     System.out.println(texto1.concat(texto2));// Texto para hacer las
pruebas
20     System.out.println(texto1.copyValueOf(tabla1));// Prueba
21     System.out.println(texto1.endsWith("as")); // true
22     System.out.println(texto1.equals(texto2)); // false
23     System.out.println(texto1.equalsIgnoreCase(texto2)); // true
24     System.out.println(Arrays.toString(texto3.getBytes())); // [84, 101,
120, 116, 111]
25
26     texto1.getChars(1,3,tabla2,1);//
27     System.out.println(Arrays.toString(tabla2)); // [ , e, x, , , ,
, , ]
28
29     System.out.println(texto1.indexOf("a")); // 7
30     System.out.println(texto1.indexOf("a",7+1)); // 9
31     System.out.println(texto1.lastIndexOf("a")); // 26
32     System.out.println(texto1.lastIndexOf("a",7)); // 7
33     System.out.println(texto1.length()); // 28
34     System.out.println(texto4.matches("T")); // false
35     System.out.println(texto4.matches("T.*")); // true
36     System.out.println(texto5.matches(".+\n.+")); // true
37     System.out.println(texto1.matches("[Txe].+")); // true
38     System.out.println(texto1.matches(".*[^wk].*")); // true
39     System.out.println(texto1.matches("[a-z]")); // false
40     System.out.println(texto1.matches(".*[a-x&&r-z].*")); // true
41     System.out.println(texto1.matches(".*[a-x&&[^acd]].*")); // true
42     System.out.println(texto1.matches("T?.+")); // true
43     System.out.println(texto6.matches(".*\\d.*")); // true
44     System.out.println(texto6.matches(".*\\D.*")); // true
45     System.out.println(texto6.matches(".*\\d{3}.*")); // false
46     System.out.println(texto6.matches(".*\\d{1,.}.*")); // true
47     System.out.println(texto6.matches(".*\\d{1,3}.*")); // true
48
49     System.out.println(texto1.replace('e','*'));// T*xto para hac*r las
pruebas
50     System.out.println(texto1.replace(" p","--"));// Texto--ara hacer
las--ruebas
51     System.out.println(texto1.replaceFirst("e","*"));// T*xto para hacer
las pruebas
52     System.out.println(texto1.substring(1,8)); // exto pa
```