

Unidad 4. Arrays en Java

Contenido

1	Actividad inicial introductoria.	1
2	Actividad inicial motivacional.	2
3	Introducción	2
4	Arrays	2
4.1	Representación en memoria	3
4.2	Declarar un array en java	5
4.3	Recorrido secuencial de arrays	7
4.4	Accediendo a las posiciones del array	8
5	Practica guiada. Notas Cornell	10
5.1.1.1	Practica Independiente	11
5.1.1.2	Actividad guiada Generando conocimiento por si mismos.	11
5.2	Array overflow y overrun	12
5.3	Haciendo más cosas con arrays	13
5.3.1.1	Practica independiente. Calcular el mínimo de temperaturas.	14
5.3.1.2	Practica independiente. Notas alumnos.	14
5.4	Arrays y el bucle for-each	15
5.4.1.1	Practica independiente for-each.	16
5.5	Limitaciones de las arrays	16
5.6	Operaciones con arrays. Array como parámetro y array devuelto en métodos. Tarea guiada	16
5.6.1	Tarea de Refuerzo	20
5.7	Arrays de objetos	21
5.8	Ejemplo Array de Objetos empleados.	23
5.8.1	Ejercicio.	26
6	Arrays multidimensionales.	27
6.1	Arrays rectangulares bidimensional. Matrices	28
6.2	Matrices irregulares (Jagged Arrays)	32
6.2.1	Ejercicio.	36
7	Abstrayendo matrices matemáticas con arrays y clases.	37
8	Bibliografía y referencias web	41

1 Actividad inicial introductoria.

Se pregunta a los alumnos que hacer para manejar 80 coches de nuestra clase Car. Que aporten ideas. ¿Debemos crear 80 variables?

2 Actividad inicial motivacional.

Se enseña un juego cuya estructura principal de datos es un array para explicar la importancia que tienen los arrays en todos los juegos de tablero

3 Introducción

La naturaleza de nuestras clases limita el número de acciones que puedes hacer fácilmente con ellos. Los algoritmos que hemos examinado hasta ahora todos han sido algoritmos secuenciales: algoritmos que pueden ser realizados para examinar cada elemento de datos una vez, en secuencia. Sin embargo hay un clase de algoritmos que se pueden realizar cuando se puede tener acceso a los datos elementos varias veces y en un orden arbitrario.

En este capítulo se examina un nuevo objeto denominado array que proporciona esta más información tipo flexible de acceso. El concepto de arrays de discos no es complejo, pero puede tomar un tiempo para que un programador principiante aprenda todas las diferentes formas en que puede ser un vector o matriz usado dependiendo de sus dimensiones. El capítulo comienza con una discusión general de arrays y luego nos movemos a las operaciones más comunes con arrays, así como técnicas avanzadas de arrays.

4 Arrays

Un array es una estructura flexible para almacenar una secuencia de valores del mismo tipo.

Array : Estructura que contiene varios valores del mismo tipo. Los valores almacenados en una matriz se denominan elementos. Los elementos individuales son se obtiene acceso mediante un índice entero.

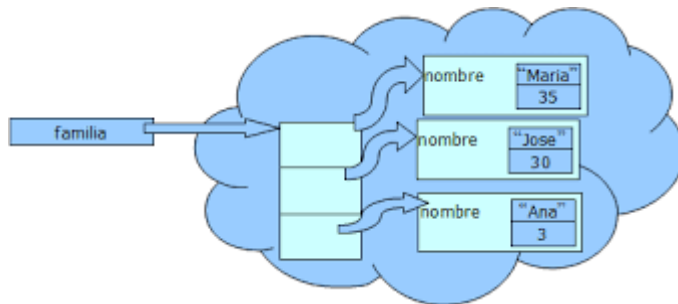
Si es de una dimensión uno le podemos llamar **vector** porque tiene la forma de un vector de matemáticas y están basados en ese concepto

Índice: Entero que indica la posición de un valor en una estructura de datos.

Puedes pensar en un array como en una calle con casas individuales. Cada calle tiene un numero donde vive familia. La calle contiene familias, el array va a

contener los tipos de datos que tu le digas, que podrían ser un objeto de tipo Familia.

Puedes pensar también en una familia en forma de Array de personas.

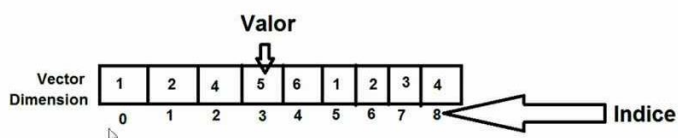


Indexación de base cero: Un esquema de numeración utilizado en java en el que una secuencia de valores se indexa a partir de 0 (elemento 0, elemento 1, elemento 2, etc.). Puede parecer más natural tener índices que comiencen con 1 en lugar de 0, pero Sun decidió que Java usaría el mismo esquema de indexación que se usa en C y C++.

Vectores

Varios Valores

Vector	1	2	4	5	6	1	2	3	4
Dimension									

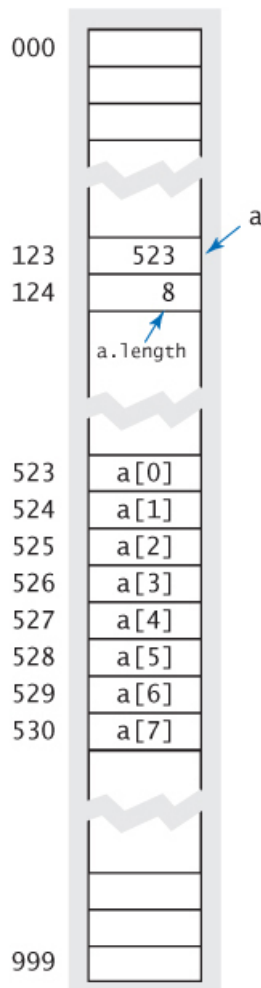


4.1 Representación en memoria

Los arrays son estructuras de datos fundamentales en el que tienen una correspondencia directa con los sistemas de memoria en prácticamente todas las computadoras. Los elementos de un array se almacenan consecutivamente en la memoria, de modo que es fácil acceder rápidamente a cualquier valor del array. De hecho, podemos ver la memoria misma como un array gigante. En las computadoras modernas, la memoria se implementa en hardware como una secuencia de ubicaciones de memoria, a cada una de las cuales se puede acceder rápidamente con un índice, la dirección de memoria. Cuando nos referimos a la memoria de la computadora, normalmente nos referimos al índice de una ubicación como su dirección. Es conveniente pensar en el nombre del array, digamos, a, como el almacenamiento de la dirección de memoria del

primer elemento de la matriz `a[0]`. A efectos ilustrativos, supongamos que el

La memoria del ordenador está organizada como 1.000 valores, con direcciones de 000 a 999. (Este modelo simplificado evita el hecho de que los elementos de la matriz pueden ocupar diferentes cantidades de memoria según su tipo, pero puede ignorar dichos detalles por el momento). Ahora, supongamos que una matriz de ocho elementos se almacena en las ubicaciones de memoria 523 a 530. En tal situación, Java almacenaría la dirección de memoria (índice) del primer elemento de matriz en otro lugar de la memoria, junto con la longitud de la matriz. Nos referimos a la dirección como un puntero y pensamos que apunta a la ubicación de la memoria a la que se hace referencia. Cuando especificamos `a[i]`, el compilador genera código que accede al valor deseado agregando el índice `i` a la dirección de memoria del array `a[]`. Por ejemplo, el código Java `a[4]` generaría código máquina que encuentra el valor en la ubicación de memoria $523 + 4 = 527$. Acceder al elemento `i` de una matriz es una operación eficiente porque simplemente requiere agregar dos enteros y luego hacer referencia a la memoria, solo dos operaciones elementales.



La ilustración muestra un rectángulo dividido en varias celdas. La celda en la parte superior está numerada "000" y la celda en la parte inferior está numerada "999". Los datos "523" en la celda numerada "123" están marcados como "a". Los datos "8" en la celda numerada "124" están marcados como "a.length". Las celdas

523 a 530 representan una matriz unidimensional y están etiquetadas como ""a[0]", "a[1]", "a[2]", "a[3]", "a[4]", "a[5]", "a[6]" y "a[7]".

Asignación de memoria

Cuando se utiliza la palabra clave new para crear una matriz, Java reserva suficiente espacio en la memoria para almacenar el número especificado de elementos. Este proceso se denomina **asignación de memoria**. Se requiere el mismo proceso para todas las variables que se utilizan en un programa (pero no se utiliza la palabra clave new con variables de tipos primitivos porque Java sabe cuánta memoria asignar). Llamamos la atención ahora porque es nuestra responsabilidad crear una matriz antes de acceder a cualquiera de sus elementos. Si no cumple con esta regla, obtendrá un error de variable no inicializada en tiempo de compilación.

4.2 Declarar un array en java

Supongamos que desea almacenar algunas lecturas de temperatura diferentes. Podría mantenerlos en una serie de variables:

```
double temperature1;  
double temperature2;  
double temperature3;
```

Esta no es una mala solución si tiene solo 3 temperaturas, pero suponga que necesita almacenar 3000 temperaturas. Entonces queríamos algo más flexible. En su lugar, puedes almacenar las temperaturas en un array.

Cuando se utiliza un array, primero es necesario declarar una variable para ella, por lo que tiene que saber qué tipo utilizar. El tipo dependerá del tipo de elementos que desee tener en el array. Para indicar que desee almacenar en un array, se indica el nombre de tipo con un conjunto de corchetes. Para las temperaturas, desea una secuencia de valores de tipo double, por lo que debe usar el tipo double[]. Por lo tanto, se puede declarar una variable para almacenar el array de la siguiente manera:

```
double[] temperature;
```

Los arrays son objetos, lo que significa que se deben crear con un new llamando al constructor. Simplemente declarando un variable no es suficiente para que el objeto exista. En este caso, se desea un array de tres valores double, que puede construir de la siguiente manera:

```
double[] temperature = new double[3];
```

Esta es una sintaxis ligeramente diferente de la que ha utilizado anteriormente al crear un nuevo objeto. Es una sintaxis especial solo para arrays. Observa como en el lado izquierdo no pone nada dentro de los corchetes, porque está describiendo un tipo. La temperatura variable puede hacer referencia a cualquier matriz de valores double, sin importar cuántos elementos tenga. En el lado derecho, sin embargo, se tiene que mencionar un número específico de elementos porque se le está pidiendo a Java que construya un objeto de matriz real y necesita saber cuántos elementos incluir.

La sintaxis general para declarar y crear un array es la siguiente:

```
<element type>[] <name> = new <element type>[<size>];
```

Se puede utilizar cualquier tipo como tipo de elemento, aunque los lados izquierdo y derecho de esta instrucción tienen que coincidir. Por ejemplo, cualquiera de las siguientes serían formas legales de construir una matriz:

```
int[] numbers = new int[10]; // un array de 10 ints  
char[] letters = new char[20]; // un array de 20 caracteres  
boolean[] flags = new boolean[5]; // un array de 5 booleanos  
String[] names = new String[100]; // un array de 100 cadenas  
Point[] points = new Point[50]; // un array de 50 puntos
```

Hay algunas reglas especiales que se aplican cuando se construye una matriz de objetos como una matriz de cadenas o una matriz de puntos, pero vamos a discutirlos más adelante en la unidad. Al ejecutar la línea de código para crear el array de temperaturas, Java construirá una matriz de tres valores doubles, con la variable temperature refiriéndose al array:



Como se puede ver, la variable `temperature` no es en sí mismo un array. En su lugar, almacena una referencia al array como en cualquier objeto Java. Los índices de arrays se indican entre corchetes. Para hacer referencia a un elemento individual del array, se combina el nombre de la variable que hace referencia al array (`temperature`) con un índice específico (`[0]`, `[1]` o `[2]`). Por lo tanto, hay un elemento conocido como `temperature[0]`, un elemento conocido como `temperature[1]` y un elemento conocido como `temperature[2]`.

En el diagrama del array de temperatura, cada uno de los elementos de la matriz se indica que tiene el valor `0.0`. Cada elemento se inicializa en un valor predeterminado, un proceso conocido como inicialización automática, tras

declarar y realizar el new del array.

Cuando Java realiza la inicialización automática, siempre se inicializa en el equivalente cero para el tipo. La tabla indica los valores equivalentes a cero para varios tipos. Observe que el equivalente cero para el tipo double es 0,0, por lo que los elementos del array se inicializaron en ese valor. Con los índices, puede almacenar los valores de temperatura específicos con los que desea trabajar:

```
temperature[0] = 74.3;  
temperature[1] = 68.4;  
temperature[2] = 70.3;
```

El código modifica el array para que tenga los siguientes valores:

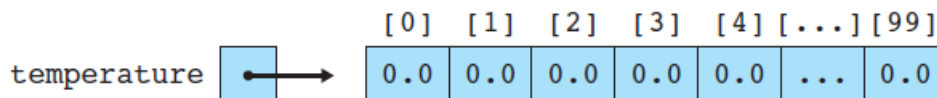


4.3 Recorrido secuencial de arrays

Obviamente, un array no es particularmente útil cuando tiene solo tres valores, pero se puede crear un array mucho más grande. Por ejemplo, podríamos crear una matriz de 100 temperaturas diciendo:

```
double[] temperature = new double[100];
```

Es prácticamente la misma línea de código que se ejecutó antes. La variable sigue siendo declarada como de tipo `double[]`, pero al construir el array se solicitan 100 elementos en lugar de 3, que construye una matriz mucho mayor:



Observa que el índice más alto es 99 en lugar de 100 debido a la indexación de base cero. No está restringido a valores literales simples dentro de los corchetes ya que se puede utilizar cualquier expresión de entero. Esto permite combinar arrays con bucles, lo que simplifica en gran medida el código que escribe. Por ejemplo, supongamos que deseamos leer una serie de temperaturas de un escáner. Puede leer cada valor individualmente, como en:

```
temperature[0] = input.nextDouble();
temperature[1] = input.nextDouble();
temperature[2] = input.nextDouble();
. . .
temperature[99] = input.nextDouble();
```

Pero como lo único que cambia de una instrucción a la siguiente es el índice, puede capturar este patrón en un bucle for con una variable de control que toma los valores de 0 a 99. Podemos usar expresiones para referenciar a una posición del array como en `temperature[i]`.

```
for (int i = 0; i < 100; i++) {
    temperature[i] = input.nextDouble();
}
```

Es momento de introducir la propiedad de arrays `length` que nos permite consultar la longitud del array, en este caso 100, con `temperature.length`. No necesitamos en código saber su longitud de antemano, sólo consultar la propiedad.

```
for (int i = 0; i < temperature.length ; i++) {
    temperature[i] = input.nextDouble();
}
```

Y como veis estamos recorriendo el array de manera secuencial, lo que se conoce como `recorrido del array`. Ejemplo de programa completo.

4.4 Accediendo a las posiciones del array

Como se describe en la anterior sección, nos referimos a los elementos del array combinando el nombre de la variable que hace referencia al array con un índice entero entre corchetes:

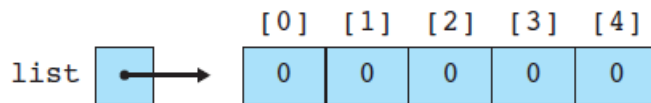
`<array variable>[<integer expression>]`

Se puede observar en esta descripción de sintaxis que el índice puede ser una expresión entera arbitraria. Para explotar esta capacidad, veamos cómo accederíamos a valores particulares en una matriz de enteros. Supongamos que construimos una matriz de longitud 5 y la llenamos con los primeros cinco enteros impares:

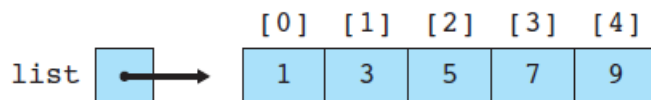
La primera línea de código declara una lista de variables de tipo `int[]` y hace que haga referencia a un matriz de longitud 5. Los elementos del array se

inician automáticamente en 0:

```
int[] list = new int[5];
```



La línea de asignación interior del bucle llena cada posición con el número impar en secuencia con la fórmula $2*i + 1$.



```
int[] list = new int[5];  
for (int i = 0; i < list.length; i++) {  
    list[i] = 2 * i + 1;  
}
```

Supongamos que queremos informar de los valores primero, medio y último de la lista. Mirando el diagrama anterior, podemos ver que se producen en los índices 0, 2 y 4, que significa que podríamos escribir el siguiente código:

```
// funciona solo para arrays de longitud 5  
System.out.println("primero = " + list[0]);  
System.out.println("medio = " + list[2]);  
System.out.println("final = " + list[4]);
```

Esto funciona cuando el array es de longitud 5, pero supongamos que cambiamos la longitud del array. Si el array tiene una longitud de 10, por ejemplo, este código informará de los incorrectos valores. Necesitamos modificarlo para incorporar `list.length`, al igual que al escribir el bucle de recorrido estándar.

Esto funciona cuando el array es de longitud 5, pero supongamos que cambiamos la longitud del array. Si el array tiene una longitud de 10, por ejemplo, este código informará de los incorrectos valores. Necesitamos modificarlo para incorporar `list.length`, al igual que al escribir el bucle de recorrido estándar.

El primer elemento del array siempre estará en el índice 0, por lo que la primera línea de código no necesita cambiar. Al principio podría pensar que podríamos arreglar la tercera línea de código reemplazando el **4** con `list.length`:

```
// No funciona porque el índice empieza en cero
System.out.println("final = " + list[list.length]);
```

El último valor estará en el índice `list.length - 1`. Podemos usar esta expresión directamente en nuestra declaración `println`:

```
// Este Código funciona
System.out.println("final = " + list[list.length - 1]);
```

Ejemplo ArrayImpares.java

```
package arrays;

import java.util.Scanner;

public class ArrayImpares {
    public static void main(String[] args) {

        Scanner console = new Scanner(System.in);
        System.out.print("Cuantos numeros impares a la lista? ");
        int numImpares = console.nextInt();
        int list[] = new int [numImpares];
        for (int i = 0; i <= numImpares-1; i++) {

            list[i] = 2*i+1;
            System.out.println("Numero impar posicion " + i + " y
valor:" + list[i]);

        }
    }
}
```

5 Practica guiada. Notas Cornell

1. Prueba y ejecuta el siguiente código.
2. **Notas Cornell**: comenta las líneas en verde y di que hacen.

ArrayRecorridoSecuencial.java

```
package arrays;
```

```
import java.util.*;

public class ArrayRecorridoSecuencial {

    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Para cuantos dias tomamos datos de
temperatura? ");
        int numDias = console.nextInt();
        int next = 0;
        float arrayDias[] = new float[numDias];
        int sum = 0;
        for (int i = 0; i < numDias; i++) {
            System.out.print("Dia " + (i + 1) + ", Introduce la
Temperatura mas alta ");
            next = console.nextInt();
            arrayDias[i] = next;
            sum += next;
        }

        double media = (double) sum / numDias;
        System.out.println();
        System.out.println("Media de temperatura = " + media);
    }
}
```

```
Para cuantos dias tomamos datos de temperatura? 4
Dia 1, Introduce la Temperatura mas alta 45
Dia 2, Introduce la Temperatura mas alta 40
Dia 3, Introduce la Temperatura mas alta 38
Dia 4, Introduce la Temperatura mas alta 41
```

Media de temperatura = 41.0

5.1.1.1 Practica Independiente

1. Añade las líneas de primer valor, valor medio y valor final al ejemplo `ArrayImpares.java`
2. ¿Que valor usarías para la posición del medio del array?
3. ¿Cómo modificarías este array añadiendo más código para que los números fueran pares, sin contar el cero?

5.1.1.2 Actividad guiada Generando conocimiento por si mismos.

1. Crea el siguiente programa y pruebalo

```
package arrays;

public class EjemploOutOfBounds {

    public static void main(String[] args) {

        String arrayString[] = {"Cadena 1", "Cadena 2", "Cadena 3",
                                "Cadena 4"};

        System.out.println("La posición 2 del array contiene" +
            arrayString[2]);

    }

}
```

Se ofrece en él una nueva forma de inicializar arrays. Con las llaves y con valores separados por comas se pueden inicializar arrays sin usar new. En este caso el array tendrá 4 posiciones, de la 0 a la 3.

```
String arrayString[] = {"Cadena 1", "Cadena 2", "Cadena 3", "Cadena 4"};
```

Prueba ahora a añadir esta línea de código y observa el resultado.

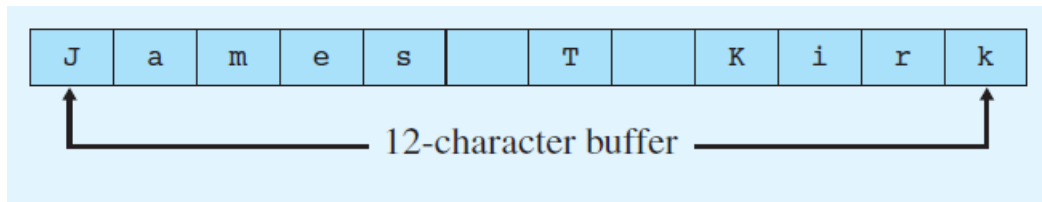
```
System.out.println("La posición 6 del array contiene" + arrayString[6]);
```

5.2 Array overflow y overrun

Buffer Overruns

Una de las primeras y aún más comunes fuentes de problemas de seguridad informática es una **overrun del búfer** (también conocido como **overflow/desbordamiento del búfer**). Una saturación del búfer es similar a una excepción de índice de matriz fuera de los límites (IndexOutOfBoundsException). Se produce cuando un programa escribe datos más allá de los límites del búfer reservado para esos datos.

Por ejemplo, podría tener espacio asignado para la cadena de 12 caracteres "James T Kirk":



Supongamos que indica al equipo que sobrescriba este búfer con la cadena "Jean Luc Picard". Hay 15 letras en el nombre de Picard, por lo que si escribe todos esos caracteres en el búfer, lo "invade" escribiendo tres caracteres adicionales:

Las últimas tres letras del nombre de Picard ("ard") se escriben en una parte de memoria que está más allá del final del búfer. Esta es una situación muy peligrosa, porque sobrescribirá cualquier dato que ya esté allí. Esto sería como un compañero de estudios agarrando tres hojas de papel de tu cuaderno y borrando cualquier cosa que hayas escrito en ellas. Es probable que tenga información útil escrita en esas hojas de papel, por lo que es probable que el desbordamiento cause un problema.

Las **saturaciones/overrun** del búfer a menudo se escriben como código de matriz. Usted puede preguntarse cómo tal programa malintencionado podría escribirse si el equipo comprueba los límites al tener acceso a una matriz. **La respuesta es que los lenguajes de programación más antiguos como C y C++ no comprueban los límites cuando se accede a una matriz.** En el momento en que se diseñó Java a principios de la década de 1990, el peligro de saturaciones de búfer era claro y los diseñadores del lenguaje **decidieron incluir la comprobación de límites de matriz** para que Java fuera más seguro. **Microsoft incluyó una comprobación de límites similares cuando diseñó el lenguaje C# a finales de la década de 1990.**

¿ Que le ha pasado a tu programa? ¿Qué ha hecho java como respuesta?

5.3 Haciendo más cosas con arrays

Aprovechando que tenemos guardados los datos de temperatura en el array vamos a hacer procesamiento de datos sobre ellos, volviendo a recorrer el array de manera secuencial. En este caso queremos calcular además de la media el máximo de temperaturas de todos los días introducidos.

Si añadimos estas líneas de código al final del ejemplo de temperaturas obtendremos el máximo. La línea clave es el if dentro del for. Si encontramos un día con temperatura mayor que máximo lo convertimos en el máximo:

```
float maximo=0;
int contarDias=0;
```

```
if (maximo < arrayDias[i] )
    maximo= arrayDias[i];
```

Además vamos a contar los días cuya temperatura sea menor de 20 grados.

```
if (arrayDias[i]<20)
    contarDias++;
```

```
float maximo=0;
int contarDias=0;
for (int i=1 ; i< numDias ; i++) {
```

```
    if (maximo < arrayDias[i] )
        maximo= arrayDias[i];
```

```
    if (arrayDias[i]<20)
        contarDias++;
```

```
    }
    System.out.println("El maximo de temperaturas de todos los dias
es: " + maximo);
```

5.3.1.1 Practica independiente. Calcular el mínimo de temperaturas.

1. Completa el ejemplo anterior para guardar también el mínimo.
2. Comenta el código anterior y el obtenido, especialmente los if.

5.3.1.2 Practica independiente. Notas alumnos.

Realiza un programa que pida por pantalla el numero de alumnos, y por cada uno de ellos recoja su nota. Además:

1. Calculamos la nota media
2. Calculamos máximo y mínimo
3. Numero de suspensos y de aprobados.
4. La media de los alumnos que están en posiciones impares del array 1,3,5

..., etc.

5.4 Arrays y el bucle for-each

Java 5 introdujo una nueva sentencia de bucle que simplifica ciertos bucles de arrays y colecciones que veremos más adelante. Se conoce como el bucle for mejorado o el bucle for-each. Se puede utilizar siempre que desee examinar cada valor de un array. Por ejemplo, en el programa Temperature2 podemos usar este bucle de la siguiente manera:

```
float maximo=0;
int contarDias=0;
for (int temperatura: arrayDias) {

    if (maximo < temperatura )
        maximo= temperatura;

    if (temperatura<20)
        contarDias++;
}

System.out.println("El maximo de temperaturas de todos los dias
es: " + maximo);
```

El nuevo bucle tiene la siguiente sintaxis

```
for ( variable declarada : array/colección) {

}

for (int temperatura: arrayDias)
```

Lo que hace este bucle es por cada elemento del array o colección, uno a uno va poniendo cada elemento en la variable temperatura y usando dentro del bucle. Imaginad que el array tiene la siguiente informacion. Este bucle sirve para recorrer un array ya lleno, no para llenar uno, el array debe estar lleno de antemano, es para recorridos.

45.2	37.4	31	28	26	24	33
------	------	----	----	----	----	----

En la primera iteración del array temperatura = 45.2

En la segunda iteración del array temperatura= 37.4

..

En la septima iteración y ultima temperatura = 33.

5.4.1.1 Practica independiente for-each.

1. Modifica el ejercicio de notas para usar el for-each en los recorridos.

5.5 Limitaciones de las arrays

Debe tener en cuenta algunas limitaciones generales de las matrices:

- No puede cambiar el tamaño de un array en medio de la ejecución del programa.
- No se pueden comparar arrays con el gual utilizando una simple condición ==. Recuerde que los arrays son objetos, por lo que si pregunta si un array es == a otra matriz, está preguntando si son el mismo objeto, no si almacenan los mismos valores.
- No puede imprimir una matriz utilizando una simple instrucción print o println. Obtendrá una salida extraña cuando lo haga.

Estas limitaciones han existido en Java desde que el lenguaje fue introducido por primera vez.

A lo largo de los años, Sun ha añadido nuevas clases a las bibliotecas de Java para abordarlas. La clase Arrays proporciona una solución para el segundo y el tercero Limitaciones. Puede comparar dos matrices para la igualdad llamando al método Arrays.equals y puede convertir una matriz en un texto útil equivalente llamando al método Arrays.toString. Ambos métodos se discutirán en detalle en la siguiente sección, cuando exploremos cómo están escritos.

La primera limitación es más difícil de superar. Dado que un array se asigna como un bloque contiguo de memoria, no es fácil hacerla más grande. Para hacer un array más grande, tendría que construir un nuevo array que sea mayor que la antigua y copiar los valores del array antiguo al nuevo. Java proporciona una clase denominada ArrayList que realiza esta operación de crecimiento automáticamente. También proporciona métodos para insertar y eliminar valores de la mitad de una lista.

5.6 Operaciones con arrays. Array como parámetro y array devuelto en métodos. Tarea guiada

Vamos a introducir dos operaciones básicas con arrays, copia y comparación.

En temas posteriores veremos ordenación y búsqueda junto a recursividad. Como ya hemos dicho no se puede comparar directamente un array debemos hacer una comparación con un método. Para la copia exactamente lo mismo.

Para copiar un array usaremos el método de los arrays `clone()`. Para comparar usaremos el método `equals` que hemos visto en anteriores temas

Igualmente podemos construir en nuestras clases nuestros propios métodos que reciban arrays como parámetros o devuelvan arrays.

Para copiar un array usamos el método `clone`. De este modo en el ejemplo hemos copiado `arrayString` en `arrayString2`.

```
String arrayString[] = {"Cadena 1","Cadena 2","Cadena 3" ,"Cadena 4"};
String arrayString2[];

arrayString2 = arrayString.clone();
```

Para comparar dos arrays deberíamos usar el método `equals`, pero no está implementado correctamente. Debemos implementarlo nosotros.

```
if (arrayString.equals(arrayString2)) {
```

Además en el ejemplo tenemos dos métodos:

`public static void muestraArray(String [] array)` que recibe un array por parámetro y lo muestra por pantalla.

`public static String [] modificaArray(String [] array)` recibe un array como parámetro y devuelve un array modificado. Fijaos como para devolver un array en un método se indica el tipo `String[]`.

`public static boolean arrayIguales(String[] array1, String[] array2)` que comprueba si dos arrays son iguales.

Vamos a explicar el código paso a paso:

```
package arrays;

public class OperacionesConArrays {

    public static boolean arrayIguales(String[] array1, String[] array2) {

        boolean resultado = true;

        // Si dos arrays son de distinta longitud son distintos
        if (array1.length != array2.length) {
            resultado = false;
        } else {
            for (int i = 0; i < array1.length; i++) {
                // Si dos arrays tienen un elemento diferente son distintos
                if (!array1[i].equals(array2[i])) {
                    resultado = false;
                    break;
                }
            }
        }
        return resultado;
    }

    public static String [] copiaArray(String[] array) {
        // creamos un array de la misma longitud del original
        String[] arrayCopiado = new String[array.length];

        for (int i = 0; i < array.length; i++) {
            //Copiamos elemento a elemento
            arrayCopiado[i] = array[i];
        }
        //devolvemos la copia
        return arrayCopiado;
    }

    public static void muestraArray(String [] array) {

        for (String s : array) {

            // mostramos cada element separado por |
            System.out.print(array[i] + "|");

        }

    }

    public static String [] modificaArray(String [] array) {
        // Creamos un array de la misma longitud
        // que el pasado como parametro
        String[] arrayModificado = new String[array.length];
    }
}
```

```
        for (int i=0; i<array.length; i++) {

            //copiamos cada element del array original
            // añadiendo modificado

            arrayModificado[i] = array[i] + " Modificado ";
        }
        //devolvemos el nuevo array
        return arrayModificado;
    }

    public static void main(String[] args) {

        String arrayString[] = {"Cadena 1", "Cadena 2", "Cadena 3",
        , "Cadena 4"};
        String arrayString2[];

        // CLONE NO FUNCIONAL PROBARLO
        arrayString2 = arrayString.clone();
        //

        // equals NO FUNCIONA PROBARLO
        // CAMBIARLO POR EL METODO QUE HEMOS CREADO

        if (arrayString.equals(arrayString2)) {

            System.out.println("arrayString y arrayString2 son iguales
con equals");
        } else {

            System.out.println("arrayString y arrayString2 son
distintos con equals");
        }

        if (OperacionesConArrays.arrayIguales(arrayString,
arrayString2)) {

            System.out.println("arrayString y arrayString2 son iguales
con nuestro método");
        } else {

            System.out.println("arrayString y arrayString2 son
distintos con nuestro método");
        }

        arrayString2 = OperacionesConArrays.modificaArray(arrayString);
        System.out.println("Mostramos arrayString2 despues de
modificarlo con el método modificaArray");
        OperacionesConArrays.muestraArray(arrayString2);

        if (OperacionesConArrays.arrayIguales(arrayString,
arrayString2)) {
```

```

        System.out.println("arrayString y arrayString2 son
iguales");
    } else {
        System.out.println("arrayString y arrayString2 son
distintos");
    }
}
}

```

Resultado de la ejecución:

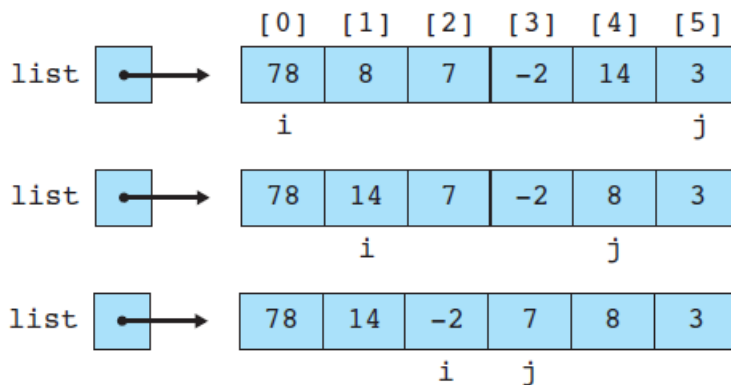
```

Mostramos arrayString
Cadena 1|Cadena 2|Cadena 3|Cadena 4|
Mostramos arrayString2
Cadena 1|Cadena 2|Cadena 3|Cadena 4|
arrayString y arrayString2 son distintos con equals
arrayString y arrayString2 son iguales con nuestro método
Mostramos arrayString2 despues de modificarlo con el método modificaArray
Cadena 1 Modificado |Cadena 2 Modificado |Cadena 3 Modificado |Cadena 4
Modificado |
arrayString y arrayString2 son distintos

```

5.6.1 Tarea de Refuerzo

1. Dado un array de float. Crear una clase OperacionesConArraysFloat que implemente:
 - a) Un método ArrayFloatIguales
 - b) Un método copiarArraysFloat
2. Construir una operación invierteArray que dado un array coloque sus elementos en posición contraria. Por ejemplo en un array de 6 elementos la posición 0 iría a 5 y viceversa. La 1 a 4 la 2 a 3.



3. Crear una clase genérica para ArraysGenéricos, que implementen dos métodos el ArrayGenericoIguales y el copiarArraysGenerico.

5.7 Arrays de objetos

Igual que podemos usar tipos primitivos con arrays, podemos almacenar objetos en arrays.

Todas los arrays que hemos visto hasta ahora han almacenado valores primitivos como valores int simples, pero puede tener matrices de cualquier tipo de Java. Sin embargo, los arrays de objetos se comportan de manera ligeramente diferente, porque los objetos se almacenan como referencias en lugar de como valores . La construcción de un array de objetos suele ser un proceso de dos pasos, porque normalmente tiene que construir tanto el array como los objetos individuales.

Para el siguiente ejemplo Point.java

```
package arraysdeobjetos.arraypuntos;

public class Point {

    private double x;
    private double y;

    public Point() {

        x=0;
        y=0;
    }

    public Point(double x, double y) {
        super();
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }
    public void setX(double x) {
        this.x = x;
    }
    public double getY() {
        return y;
    }
}
```

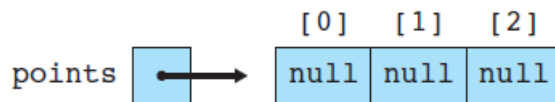
```
    }  
    public void setY(double y) {  
        this.y = y;  
    }  
  
    @Override  
    public String toString() {  
        return "Point [x=" + x + ", y=" + y + "]";  
    }  
}
```

Considere, por ejemplo, la siguiente declaración:

```
Point[] points = new Point[3];
```

Esto declara una variable denominada `points` que hace referencia a un array de longitud 3 que almacena referencias a objetos `Point`. La nueva palabra clave no construye ningún objeto `Point` real. En su lugar, construye un array de longitud 3, cada elemento de los cuales puede almacenar una referencia a un punto. Cuando Java construye el array, inicializa automáticamente estos elementos del array al equivalente cero para el tipo. El equivalente cero para todos los tipos de referencia es el valor especial `null`, que indica "ningún objeto":

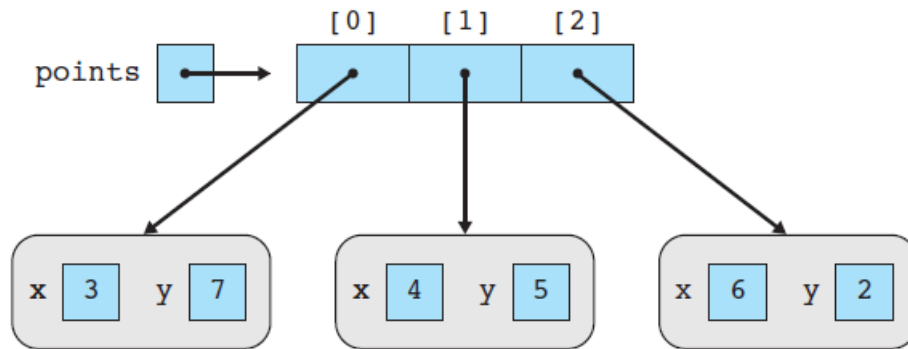
```
Point points [] = new Point[3];
```



Los objetos `Point` reales deben construirse por separado con la nueva palabra clave, como en:

```
Point[] points = new Point[3];  
points[0] = new Point(3, 7);  
points[1] = new Point(4, 5);  
points[2] = new Point(6, 2);
```

Después de ejecutar estas líneas de código, tendría objetos `Point` individuales referidos por los diversos elementos del array:



Podemos recorrer el array con un for como hemos visto al estilo foreach.

```
for(Point point : points ) {  
    System.out.println(point);  
}
```

```
Point [x=3.0, y=7.0]  
Point [x=4.0, y=5.0]  
Point [x=6.0, y=2.0]
```

O podemos acceder a las posiciones de manera aleatoria

```
System.out.println(points[1]);
```

```
Point [x=4.0, y=5.0]
```

5.8 Ejemplo Array de Objetos empleados

Vamos a generar un array de objetos leyendo los datos de consola con un bucle. Para ello vamos a usar la siguiente clase Persona, que tiene de atributos nombre, apellidos e id. El id lo pondremos automáticamente con un contador, los nombre y apellidos los leeremos de consola.

```
package arraysdeobjetos.empleados;
```

```
public class Empleado {
```

```
    int id;  
    String nombre;
```

```
String apellidos;

public Empleado () {

}

public Empleado (int id, String nombre, String apellidos) {

    this.id=id;
    this.nombre=nombre;
    this.apellidos=apellidos;

}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellidos() {
    return apellidos;
}

public void setApellidos(String apellidos) {
    this.apellidos = apellidos;
}

@Override
public String toString() {
    return "Empleado [id=" + id + ", nombre=" + nombre + ",
apellidos=" + apellidos + "];"
}

}
```

Para ello lo primero es crear el array de empleados y preguntarle al usuario el número de empleados que vamos a leer por pantalla para darle un tamaño al array.

```
Empleado empleados[];
```

```
System.out.println("Cuantos empleados desea para el array");
numEmpl= sc.nextInt();
```



```
empleados= new Empleado[numEmpl];
```

Lo siguiente es en un bucle for ir leyendo el nombre y los apellidos. De id para el empleado asignaremos la variable contador del bucle for que en este caso le hemos llamado id.

```
for (int id=0; id<numEmpl; id++) {  
    System.out.println("Escriba el nombre del empleado");  
    nombre= sc.next();  
  
    System.out.println("Escriba el primer apellidos del  
empleado");  
  
    apellidos= sc.next();  
  
    System.out.println("Escriba el segundo apellido del  
empleado");  
  
    apellidos= apellidos+ " " + sc.next();  
  
    Empleado emp = new Empleado(id, nombre, apellidos);  
    empleados[id] = emp;  
}
```

Fijaos como al final, tras leer todos los datos creamos el empleado y lo asignamos al array

```
Empleado emp = new Empleado(id, nombre, apellidos);  
    empleados[id] = emp;
```

Una vez el array ha sido creado lo recorreremos para leer sus elementos.

```
for (Empleado empl : empleados) {  
    System.out.println(empl);  
}
```

El ejemplo completo del programa ArrayEmpleados quedaría así.

```
package arraysdeobjetos.empleados;  
  
import java.util.Scanner;
```

```
public class ArrayEmpleados {  
  
    public static void main(String[] args) {  
  
        int numEmpl=0;  
        String nombre="";  
        String apellidos="";  
        Scanner sc = new Scanner(System.in);  
  
        Empleado empleados[];  
  
        System.out.println("Cuantos empleados desea para el array");  
        numEmpl= sc.nextInt();  
        empleados= new Empleado[numEmpl];  
  
        for (int id=0; id<numEmpl; id++) {  
  
            System.out.println("Escriba el nombre del empleado");  
            nombre= sc.next();  
  
            System.out.println("Escriba el primer apellidos del  
empleado");  
  
            apellidos= sc.next();  
  
            System.out.println("Escriba el segundo apellido del  
empleado");  
  
            apellidos= apellidos+ " " + sc.next();  
  
            Empleado emp = new Empleado(id, nombre, apellidos);  
            empleados[id] = emp;  
  
        }  
  
        for (Empleado empl : empleados) {  
            System.out.println(empl);  
        }  
    }  
}
```

5.8.1 Ejercicio

1. Añadid los atributos edad(int) y salario (double) a Persona, modificar el constructor con parámetros con estos dos nuevos atributos. Añadid también los getters y los setters para estos atributos.
2. Modificar ArrayPersona para leer estos dos nuevos atributos
3. Crear un método en ArrayPersona para calcular la media de edad y la

media de sueldo, mediaEdad, y mediaSueldo que reciban el array de Empleados.

6 Arrays multidimensionales.

Los ejemplos de arrays en las secciones anteriores involucraban lo que se conoce como matrices unidimensionales (una sola fila o una sola columna de datos). A menudo, querrá almacenar datos de manera multidimensional. Por ejemplo, es posible que desee almacenar una tabla bidimensional de datos que tenga filas y columnas. Afortunadamente, se puede crear arrays de muchas dimensiones arbitrarias:

- double: un double
- double[]: array de doubles de una dimensión
- double[][]: array de doubles de dos dimensiones
- double[][][]: Colección de doubles de tres dimensiones

Los arrays de más de una dimensión se denominan matrices multidimensionales. Arrays multidimensional. Un array de arrays, a cuyos elementos se accede con varios índices.

	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I

`a[0][2]='C'`

```
char a[][];           //Declara
a = new char[3][3]    //Crea
a[0][0]='A';          //Inicializa
...
```

		0	1	2
0	0	a	b	c
1	0	d	e	f
2	0	g	h	i

`a[0][2][1]='l'`

```
char a[][][];         //Declara
a = new char[3][3][3] //Crea
a[0][0][0]='a'
...
```

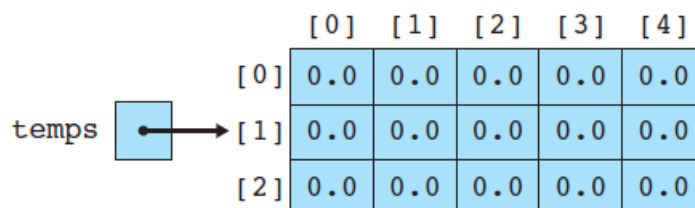
6.1 Arrays rectangulares bidimensional. Matrices

A los arrays de dos dimensiones les solemos llamar matrices porque tienen la forma de una matriz matemática, y están basados en ese concepto.

El uso más común de una matriz multidimensional es una matriz bidimensional de cierto ancho y alto. Por ejemplo, supongamos que en tres días separados tomaste una serie de cinco lecturas de temperatura. Puede definir una matriz bidimensional que tenga tres filas y cinco columnas de la siguiente manera:

```
double[][] temps = new double[3][5];
```

Observa que en los lados izquierdo y derecho de esta instrucción de asignación, debe usar un doble conjunto de corchetes. Al describir el tipo de la izquierda, debe dejar en claro que esto no es solo una secuencia unidimensional de valores, que sería de tipo doble [], sino una cuadrícula bidimensional de valores, que es de tipo doble []. A la derecha, al construir la matriz, debe especificar las dimensiones de la tabla. La convención normal es enumerar la fila primero seguida de la columna. La matriz resultante se vería así:



	[0]	[1]	[2]	[3]	[4]
[0]	0.0	0.0	0.0	0.0	0.0
[1]	0.0	0.0	0.0	0.0	0.0
[2]	0.0	0.0	0.0	0.0	0.0

Al igual que con las matrices unidimensionales, los valores se inicializan a 0.0 y los índices

comience con 0 tanto para filas como para columnas. Una vez que haya creado una matriz de este tipo, puede hacer referencia a elementos individuales proporcionando números de fila y columna específicos (en ese orden). Por ejemplo, para establecer el cuarto valor de la primera fila en 98.3 y para

establecer el primer valor de la tercera fila en 99.4, diría:

```
temps[0][3] = 98.3; // cuarto valor de la primera fila
temps[2][0] = 99.4; // primer valor de la tercera fila
```

Es útil pensar en esto de una manera gradual, comenzando con el nombre del arreglo. Por ejemplo, si desea hacer referencia al primer valor de la tercera fila, lo obtiene mediante los pasos siguientes:

temps toda la tabla

temps[2] toda la tercera fila

temps[2][0] el primer elemento de la tercera fila

Puede pasar matrices multidimensionales como parámetros del mismo modo que pasa matrices unidimensionales. Sin embargo, debe tener cuidado con el tipo. Para pasar la cuadrícula de temperatura, tendría que usar un parámetro de tipo `double[][]` (con ambos conjuntos de corchetes). Por ejemplo, aquí hay un método que imprime la tabla. Para recorrer una matriz como esta necesitaremos dos bucles anidados

```
public static void print(double[][] grid) {
    for (int i = 0; i < grid.length; i++) {
        for (int j = 0; j < grid[i].length; j++) {
            System.out.print(grid[i][j] + " ");
        }
        System.out.println();
    }
}
```

Observe que para preguntar por el número de filas se pide `grid.length` y para pedir el número de columnas que solicita la `grid[i].length`. Cada fila es un array en una matriz multidimensional

El método `Arrays.toString` mencionado anteriormente en este capítulo funciona en matrices multidimensionales, pero produce un resultado deficiente. Cuando se usa con las temperaturas de matriz anteriores, se puede usar para producir resultados como los siguientes:

```
[[D@14b081b, [D@1015a9e, [D@1e45a5c]
```

Esto se debe a que `Arrays.toString` funciona concatenando las representaciones `String` de los elementos de la matriz. En este caso los elementos son matrices propias, por lo que no se convierten en `Strings` correctamente. Para corregir el problema, puede utilizar un método diferente, denominado `Arrays.deepToString`, que devolverá mejores resultados para matrices multidimensionales:

```
System.out.println(Arrays.deepToString(temps));
```

La llamada produce el siguiente resultado:

```
[[0.0, 0.0, 0.0, 98.3, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0],  
[99.4, 0.0, 0.0, 0.0, 0.0]]
```

Con nuestro método `print` el resultado sería:

```
ArrayMultidimensional.print(temps);
```

```
0.0 0.0 0.0 98.3 0.0  
0.0 0.0 0.0 0.0 0.0  
99.4 0.0 0.0 0.0 0.0
```

Las matrices pueden tener tantas dimensiones como desee. Por ejemplo, si desea un cubo tridimensional de 4 por 4 por 4 de enteros, dirías:

```
int[][][] numbers = new int[4][4][4];
```

La convención normal sería asumir que este es el número plano seguido del número de fila seguido del número de columna, aunque se puede usar cualquier convención que desee siempre que tu código esté escrito de manera consistente.

Podemos inicializar arrays de múltiples dimensiones con literales igualmente. En el ejemplo anterior tenemos 4 filas que representan 4 semanas, y cada fila tendrá 7 temperaturas, una por día de la semana.

```
double [][] tempsCuatroSemanas = {{30,28,27,20,25,26,27},  
                                     {20,21,24,25,26,23,24},  
                                     {20,21,24,25,26,23,24},  
                                     {20,21,24,25,26,23,24}};
```

Para recorrer una matriz como la anterior necesitaremos dos bucles anidados, uno para recorrer cada fila, y otro interno para recorrer cada columna. En el siguiente ejemplo vamos a calcular las medias semanales de temperatura.

```
for (int i=0; i<tempsCuatroSemanas.length ; i++) {  
    double media=0.0;  
    for (int j=0; j<tempsCuatroSemanas[i].length;j++) {  
        media= media+ tempsCuatroSemanas[i][j];  
    }  
}
```

```
media= media/7;

System.out.println("La media de la semana " + i + " es " + media);

}
```

El ejemplo completo ArrayMultidimensional.java

```
package arraysmultidimensionales;

import java.util.Arrays;

public class ArrayMultidimensional {

    public static void print(double[][] grid) {
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[i].length; j++) {
                System.out.print(grid[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {

        double[][] temps = new double[3][5];
        temps[0][3] = 98.3; // cuarto valor de la primera fila
        temps[2][0] = 99.4; // primer valor de la tercera fila

        System.out.println(temps);
        System.out.println(Arrays.deepToString(temps));

        ArrayMultidimensional.print(temps);

        double [][] tempsCuatroSemanas = {{30,28,27,20,25,26,27},
            {20,21,24,25,26,23,24},
            {20,21,24,25,26,23,24},
            {20,21,24,25,26,23,24}};

        for (int i=0; i<tempsCuatroSemanas.length ; i++) {

            double media=0.0;

            for (int j=0; j<tempsCuatroSemanas[i].length;j++) {

                media= media+ tempsCuatroSemanas[i][j];
```

```

    }

    media= media/7;

    System.out.println("La media de la semana " + i + " es " +
media);

}

}

}

```

6.2 Matrices irregulares (Jagged Arrays)

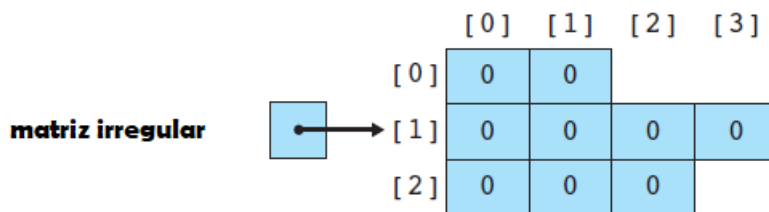
Los ejemplos anteriores han involucrado tablas rectangulares que tienen un número fijo de filas y columnas. También es posible crear una matriz irregular, donde el número de columnas varía de una fila a otra.

Para construir una matriz irregular, se divide la creación en dos pasos: crea primero la matriz para mantener las filas y, a continuación, crea cada fila individual. Por ejemplo, para construir una matriz que tiene dos elementos en la primera fila, cuatro elementos en la segunda fila y tres elementos en la tercera fila, puede decir:

```

int[][] matrizirregular = new int[3][];
matrizirregular [0] = new int[2];
matrizirregular [1] = new int[4];
matrizirregular [2] = new int[3];

```



Podemos explorar esta técnica escribiendo un programa que produzca las filas de lo que se conoce como *triángulo de Pascal*. Los números en el triángulo tienen muchas propiedades matemáticas útiles. Por ejemplo, la fila n del triángulo de Pascal contiene los coeficientes obtenidos al expandir:

$$(x + y)^n$$

Estos son los resultados para n entre 0 y 4:

$$(x + y)^0 = 1$$

$$(x + y)^1 = x + y$$

$$(x + y)^2 = x^2 + 2xy + y^2$$

$$(x + y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

$$(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$$

Si sacas solo los coeficientes, obtienes lo siguiente:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Este es el triángulo de Pascal. Una de las propiedades del triángulo es que, dada cualquier fila, puede usarla para calcular la siguiente fila. Por ejemplo, comencemos con la última fila del triángulo anterior:

```
1 4 6 4 1
```

Podemos calcular la siguiente fila sumando pares de valores adyacentes juntos. Entonces, sumamos el primer par de números (1 + 4), luego el segundo par de números (4 + 6), y así sucesivamente:

$$\begin{array}{ccccccc} (1 + 4) & (4 + 6) & (6 + 4) & (4 + 1) \\ \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} & \underbrace{\hspace{1cm}} \\ 5 & 10 & 10 & 5 \end{array}$$

Luego ponemos un 1 al frente y al reverso de esta lista de números, y terminamos con la siguiente fila del triángulo:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Esta propiedad del triángulo proporciona una técnica para calcularlo. Podemos construirlo fila por fila, calculando cada nueva fila a partir de los valores de la fila anterior. En otras palabras, vamos a escribir un bucle como este (suponiendo que tengamos una matriz bidimensional llamada triángulo en la que almacenar la respuesta):

Lo primero es el método para rellenar

```
public static void rellena(int[][] triangulo) {
    for (int i = 0; i < triangulo.length; i++) {
        triangulo[i] = new int[i + 1];
    }
}
```

```

        triangulo[i][0] = 1;
        triangulo[i][i] = 1;
        for (int j = 1; j < i; j++) {
            triangulo[i][j] = triangulo[i - 1][j - 1] +
triangulo[i - 1][j];
        }
    }
}

```

Sabemos que el primer y ultimo elemento de cada fila es uno

```

triangulo[i][0] = 1;
triangulo[i][i] = 1;

```

El resto de elementos de cada fila se rellena usando la fila anterior

```

triangulo[i][j] = triangulo[i - 1][j - 1] + triangulo[i - 1][j];

```


```

1 4 6 4 1
1 5 10 10 5 1

```

5 es = 1+4 es decir $\text{triangulo}[5][1] = \text{triangulo}[4][0] + \text{triangulo}[4][1]$

10 es = 4+6 es decir $\text{triangulo}[5][2] = \text{triangulo}[4][1] + \text{triangulo}[4][2]$ y así sucesivamente para cada elemento

triangulo 

	[0]	[1]	[2]	[3]	[4]	[5]
[0]	1					
[1]	1	1				
[2]	1	2	1			
[3]	1	3	3	1		
[4]	1	4	6	4	1	
[5]	1	5	10	10	5	1

Pintar el triángulo es algo que hemos visto anteriormente:

```

public static void print(int[][] triangulo) {
    for (int i = 0; i < triangulo.length; i++) {
        for (int j = 0; j < triangulo[i].length; j++) {
            System.out.print(triangulo[i][j] + " ");
        }
        System.out.println();
    }
}

```

El resultado para n=11

```

public static void main(String[] args) {
    // TODO Auto-generated method stub

    int[][] triangulo = new int[11][];
    rellena(triangulo);
    print(triangulo);
}

```

es:

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1

```

TrianguloPascal.java

```
package arraysmultidimensionales;
```

```

public class TrianguloPascal {

    public static void rellena(int[][] triangulo) {
        for (int i = 0; i < triangulo.length; i++) {
            triangulo[i] = new int[i + 1];
            triangulo[i][0] = 1;
            triangulo[i][i] = 1;
            for (int j = 1; j < i; j++) {
                triangulo[i][j] = triangulo[i - 1][j - 1] +
triangulo[i - 1][j];
            }
        }
    }
}

```

```

    }
}

public static void print(int[][] triangulo) {
    for (int i = 0; i < triangulo.length; i++) {
        for (int j = 0; j < triangulo[i].length; j++) {
            System.out.print(triangulo[i][j] + " ");
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    // TODO Auto-generated method stub

    int[][] triangulo = new int[11][];
    rellena(triangulo);
    print(triangulo);
}
}

```

6.2.1 Ejercicio

Crea un programa que lea para cada semana las horas de trabajo de un trabajador. Unas semanas trabajará mas días otros menos como en el siguiente ejemplo:

```

8 8 8 8 8
8 4 8 4 8 4 4
8 4 8 4 8
3 0 0 8 6 4 4
8 8
0 0 8 8
8 8 4 8 4

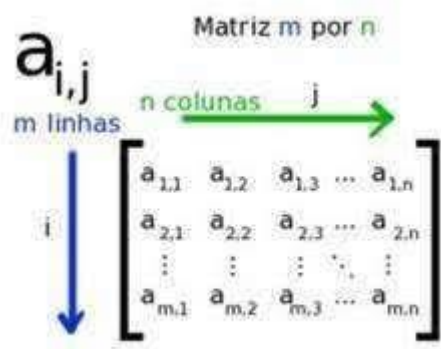
```

Crear la clase MatrizHoras.java. El programa preguntará el numero total de semanas. Y para cada semana los días que trabaja, para rellenar la matriz irregular desde la consola.

- 1) Crear un método rellena() que devolverá una matriz irregular rellena
- 2) Crea un método totalHoras() que reciba la matriz y devuelva el numero total de horas.
- 3) Crear un método que pinte para cada semana la media de horas trabajadas y el numero de días, pintaMedias().
- 4) Crea un método que calcule el numero total de días trabajados diasTrabajados() usando las propiedades length de los arrays.
- 5) Crear un método que calcule la media de horas diarias, mediaTotal() que recibirá como parámetro la matriz irregular.

7 Abstrayendo matrices matemáticas con arrays y clases.

En este apartado vamos a realizar una abstracción de una matriz matemática usando arrays y clases en Java. Hemos de tener en cuenta que en las matrices matemáticas los índices comienzan en uno y no en cero como en los arrays en Java.



Se proporciona esta clase Matriz, con unas operaciones que iremos realizando en clase. Definimos como propiedades la dimensión en filas X, y la dimensión en columnas y. Y definimos una propiedad matriz privada para guardar los datos.

```
private int dimX;
private int dimY;
private double matriz[][];
```

Para conservar los índices de la matriz matemática escribimos en el array Java en un posición menos. Por ejemplo, el índice 1 y 2, `setCell(1,2)` iría a la matriz Java en la posición `matriz[1-1][2-1]`, `matriz[0][1]`. Estamos abstrayendo la realidad y representando matrices matemáticas con arrays Java de dos dimensiones.

```
public double getCell(int x, int y) {
    return matriz[x-1][y-1];
}

public void setCell(int x, int y, double valor) {
    matriz[x-1][y-1]= valor;
}
```

Nuestra tarea será implementar todas las operaciones de matrices que se explicarán en clase, y están marcadas con implementar en un comentario.

```
package matrizmatematica;

import java.util.Scanner;

public class Matriz {

    private int dimX;
    private int dimY;
    private int dimen = 0;

    private double matriz[][];

    public Matriz(int dimX, int dimY) {

        this.dimX = dimX;
        this.dimY = dimY;

        matriz = new double[dimX][];

        for (int i = 0; i < dimX; i++) {

            matriz[i] = new double[dimY];

        }

    }

    public double getCell(int x, int y) {

        return matriz[x - 1][y - 1];

    }

    public void setCell(int x, int y, double valor) {

        matriz[x - 1][y - 1] = valor;

    }

    public int getXDimen() {

        return dimX;

    }

    public int getYDimen() {

        return dimY;

    }

    public static Matriz CrearAPartirDeArray(double[][] matrizInicial) {

        Matriz matriz = new Matriz(matrizInicial.length,
matrizInicial[0].length);

        for (int i = 0; i < matriz.getXDimen(); i++) {

            for (int j = 0; j < matriz.getYDimen(); j++) {
```

```
        matriz.setCell(i + 1, j + 1, matrizInicial[i][j]);
    }
}
return matriz;
}

//Implementar
public Matriz clonarMatriz() {
    Matriz clonada = new Matriz(this.dimX, this.dimY);

    return clonada;
}
//Implementar
public void inicializarACero() {
}
//Implementar
public void leerMatrizPantalla() {
    Scanner sc = new Scanner(System.in);
    System.out.println("Introduzca la matriz por pantalla termino a
termino");
}

//Implementar
public void mostrarEnPantalla() {
}

//Implementar
public Matriz multiplicarMatrices(Matriz m) {
    Matriz mRes = new Matriz(this.dimX, m.dimY);

    return mRes;
}

//Implementar
public void multiplicarMatrizPorEscalar(double value) {
}

//Implementar
public Matriz devuelveFila(int fila) {
    Matriz matrizFila = new Matriz(1, this.dimY);

    return matrizFila;
}

//Implementar
public Matriz devuelveColumna(int columna) {
```

```
        Matriz matrizColumna = new Matriz(this.dimX, 1);

        return matrizColumna;
    }
    //Implementar
    public Matriz matrizTranspuesta() {

        Matriz matrizTrans = new Matriz(this.dimY, this.dimX);

        return matrizTrans;
    }
    //Implementar
    public static Matriz matrizIdentidad(int dimX, int dimY) {

        Matriz matrizIden = new Matriz(dimX, dimY);

        return matrizIden;
    }
    //Implementar
    public Matriz intercambiarFilas(int i, int j) {

        Matriz identTransf = intercambiaColumnaIdentidad(i, j);

        return identTransf.multiplicarMatrices(this);
    }
    //Implementar
    public Matriz transformacionFila(int fila, double valor) {

        Matriz identTransf = Matriz.matrizIdentidad(this.dimX,
this.dimX);

        return identTransf.multiplicarMatrices(this);
    }
    //Implementar
    public Matriz transformacionFilaSuma(int i, int j, double valor) {

        Matriz identTransf = Matriz.matrizIdentidad(this.dimX,
this.dimX);

        return identTransf.multiplicarMatrices(this);
    }
    //Implementar
    private Matriz intercambiaColumnaIdentidad(int i, int j) {

        Matriz ident = Matriz.matrizIdentidad(this.dimX, this.dimX);

        Matriz identTrans = new Matriz(this.dimX, this.dimX);

        return identTrans;
    }
    //Implementar
    public Matriz intercambiarColumnas(int i, int j) {
```



```
        Matriz identTransf = Matriz.matrizIdentidad(this.dimX,
this.dimX);

        return this.multiplicarMatrices(identTransf);
    }
    //Implementar
    public Matriz transformacionColumna(int columna, double valor) {

        Matriz identTransf = Matriz.matrizIdentidad(this.dimX,
this.dimX);

        return this.multiplicarMatrices(identTransf);
    }
    //Implementar
    public Matriz transformacionColumnaSuma(int i, int j, double valor) {

        Matriz identTransf = Matriz.matrizIdentidad(this.dimX,
this.dimX);

        return this.multiplicarMatrices(identTransf);
    }

    //Rekursivo
    //Implementar
    public double calculaDeterminante() {

        return 0.0;
    }

    //Implementar
    public double calculaAdjunto(int x, int y) {

        double resultadoComponente = 0.0;

        return resultadoComponente;
    }
}
```

8 Bibliografía y referencias web

Referencias web

Tutoriales de Java arquitectura Java

<https://www.arquitecturajava.com/>

Tutoriales Java geeksforgeeks

<https://www.geeksforgeeks.org/>

Tutoriales Java Baeldung

<https://www.baeldung.com/>

Bibliografía

Programación, Alfonso Jiménez Pérez, Francisco Manuel Pérez Montes, Paraninfo, 1ª edición, 2021

Entornos de Desarrollo, Maria Jesus Ramos Martín, Garceta 2ª Edición, 2020

PROGRAMACIÓN EN LENGUAJES ESTRUCTURADOS, Enrique Quero Catalinas y José López Herranz, Paraninfo, 1997.