

NOTAS – REPASO – EDITS:   
[UT1, UT3 Generación de números aleatorios] status: updated  
Falta: [UT4 Programación Orientada a Objetos, final]

# UT5 Arrays, Matrices y ArrayLists

**UT5: ARRAYS, MATRICES Y ARRAYLISTS**

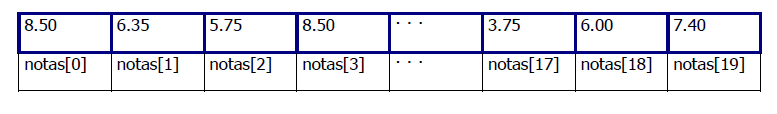
**1. ¿Qué es un array?**

Un array es una colección finita de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.

**Ej:** Se quieren guardar las notas de los 20 alumnos de una clase.

Gráficamente el array se puede representar de la siguiente forma:

Array notas:



Para acceder a cada elemento del array se utiliza el nombre del array y un índice que indica la posición que ocupa el elemento dentro del array. El índice se escribe entre corchetes.

El primer elemento del array ocupa la posición 0, el segundo la posición 1, etc. En un array de N elementos el último ocupará la posición N-1.

En el ejemplo, notas[0] contiene la nota del primer alumno y notas[19] contiene la del último.

El atributo **length** de un array contiene el tamaño del array independientemente de que tenga el valor por defecto u otro valor.

*(te devuelve el numero de elementos del array)*

**Los índices deben ser enteros no negativos.**

**1. Crear arrays unidimensionales**

Para crear un array se deben realizar dos operaciones:

1. Declaración

2. Instanciación

**1. Declaración de un array:** En la declaración se crea la referencia al array.

La referencia es el nombre del array en el programa. Se debe indicar el nombre del array y el tipo de datos que contendrá.

De forma general un array unidimensional se puede declarar de cualquiera de estas dos formas:

tipo [] nombreArray; o tipo nombreArray[];

**tipo:** indica el tipo de datos que contendrá. Un array puede contener elementos de tipo básico o referencias a objetos.

**nombreArray:** es la referencia al array.

**Ej:**

int [] ventas; //array de tipo int llamado ventas

double [] temperaturas; //array de tipo double llamado temperaturas

String [] nombres; //array de tipo String llamado nombres

**2. Instanciar un array:** Mediante la instanciación se reserva un bloque de memoria para almacenar todos los elementos del array.

La dirección, donde comienza el bloque de memoria, donde se almacenará el array se asigna al nombre. De forma general:

**nombreArray = new tipo[tamaño];**

**nombreArray**: es el nombre creado en la declaración.

**tipo**: indica el tipo de datos que contiene.

**tamaño**: es el número de elementos del array. Debe ser una expresión entera positiva. El tamaño no se puede modificar durante la ejecución del programa.

**new**: operador para crear objetos. Mediante new se asigna la memoria necesaria para ubicar el objeto. Java implementa los arrays como objetos.

**Ej**: ventas = new int[5]; //se reserva memoria para 5 enteros

Lo normal es que la declaración y la instanciación se hagan en una sola instrucción:

tipo [] nombreArray = new tipo[tamaño];

**Ej:** int [] ventas = new int[5];

El tamaño del array también se puede indicar durante la ejecución del programa, es decir, en tiempo de ejecución se puede pedir por teclado el tamaño del array y crearlo.

**Ej:**

……………..

Scanner teclado = new Scanner(System.in);

System.out.print("Número de elementos del array: ");

int numeroElementos = teclado.nextInt();

int [] ventas = new int[numeroElementos];

Si no hay memoria suficiente para crear el array, new lanza una excepción java.lang.OutOfMemoryError.



*Esquema importante: el int ventas va a tener una referencia a una dir de memoria; una vez que haga el new te reserva en memoria espacio para 5 elementos.*

|  |
| --- |
| **Diferencia entre la referencia y el contenido del array**  Debe quedar clara la diferencia entre la referencia (manejador del array o nombre del array) y el contenido del array.    El nombre del array contiene la dirección de memoria del contenido del array.  *(Por eso son contiguos, java para pasar al 2º elemento suma el tamaño del 1º.)* |

**3. Inicializar arrays unidimensionales**

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial. Los valores iniciales por defecto para un array en java son:

0 para arrays numéricos

'\u0000' (carácter nulo) para arrays de caracteres

false para arrays booleanos

null para arrays de String y de referencias a objetos.

También se pueden dar otros valores iniciales al array cuando se crea. Los valores iniciales se escriben entre llaves separados por comas y deben aparecer en el orden en que serán asignados a los elementos del array. El número de valores determina el tamaño del array.

**Ej:**

double [] notas = {6.7, 7.5, 5.3, 8.75, 3.6, 6.5};

boolean [] resultados = {true, false, true, false};

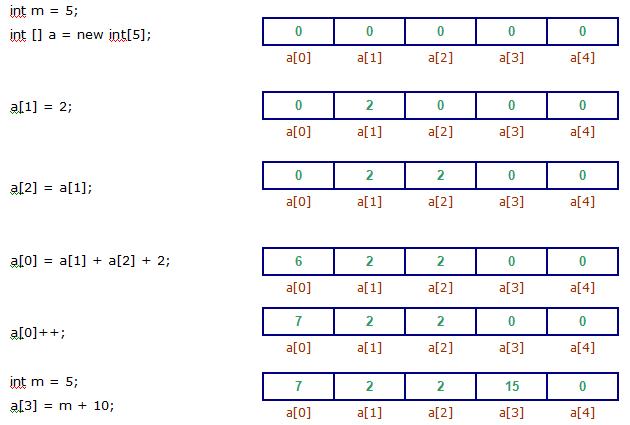
String [] dias = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"};

**4. Acceder a los elementos de un array**

Para acceder a cada elemento del array se utiliza el nombre del array y el índice que indica la posición que ocupa el elemento dentro del array. El índice se escribe entre corchetes. Se puede utilizar como índice un valor entero, una variable de tipo entero o una expresión de tipo entero.

Un elemento de un array se puede utilizar igual que cualquier otra variable.

Se puede hacer con ellos las mismas operaciones que se pueden hacer con el resto de variables (incremento, decremento, operaciones aritméticas, comparaciones, etc).



Si se intenta acceder a un elemento que está fuera de los límites del array (índice negativo o con un índice mayor que el último elemento del array) el compilador no avisa del error. El error se producirá durante la ejecución. En ese caso se lanza una excepción Java.lang.ArrayIndexOutOfBoundsException.

Se puede saber el número de elementos del array mediante el atributo **length**. Se puede utilizar length para comprobar el rango del array y evitar errores de acceso.

*(length - 1)*

Ej: Para asignar un valor a un elemento del array que se leen por teclado:

Scanner teclado = new Scanner(System.in);

int i, valor;

int [] a = new int[10];

System.out.print("Posición: ");

i = teclado.nextInt();

System.out.print("Valor: ");

valor = teclado.nextInt();

if (i>=0 && i < a.length)

a[i] = valor;

**5. Recorrer un array unidimensional**

Para recorrer un array se utiliza una instrucción iterativa (normalmente una instrucción for, aunque también puede hacerse con while o do while) usando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

**Ej**:

double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0};

for (int i = 0; i< 7; i++)

System.out.print(notas[i] + " ");

**Ej**: Programa que lee por teclado la nota de los alumnos de una clase y calcula la nota media del grupo. También muestra los alumnos con notas superiores a la media. El número de alumnos se lee por teclado.

import java.util.\*;

public class Ejemplo {

public static void main(String[] args) {

Scanner teclado= new Scanner(System.in);

int numAlum, i;

double suma = 0, media;

do {

System.out.print("Número de alumnos de la clase: ");

numAlum = teclado.nextInt();

} while (numAlum <= 0);

double[] notas = new double[numAlum];

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

System.out.print("Alumno " + (i + 1) + " Nota final: ");

notas[i] = teclado.nextDouble();

}

teclado.close();

for (i = 0; i < notas.length; i++)

suma = suma + notas[i];

media = suma / notas.length;

System.out.printf("Nota media del curso: %2f %n", media);

System.out.println("Listado de notas superiores a la media: ");

for (i = 0; i < notas.length; i++)

if (notas[i] > media)

System.out.println("Alumno numero " + (i + 1)+ " Nota final: " + notas[i]);

}

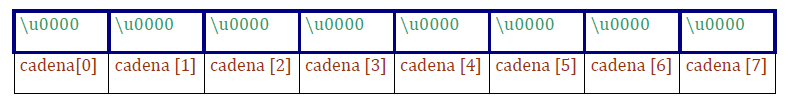
}

**6. Arrays de caracteres en Java**

Un array de caracteres en Java se crea de forma similar a un array de cualquier otro tipo de datos.

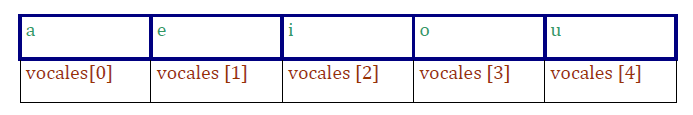
**Ej:** Array de 8 caracteres llamado cadena. Por defecto se inicializa con el carácter nulo.

char [] cadena = new char[8];



Ej: Array de 5 caracteres llamado vocales. Se asignan valores iniciales: a, e, i, o, u

char [] vocales = {'a', 'e', 'i', 'o', 'u'};



A diferencia de los demás arrays, se puede mostrar el contenido completo de un array de caracteres mediante una sola instrucción print o printf.

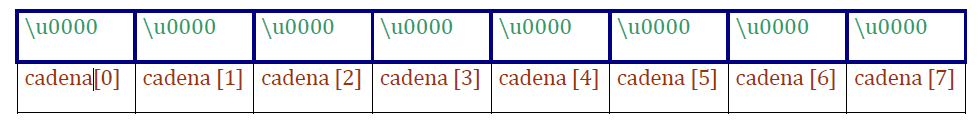
**Ej**:

System.out.println(cadena); //Muestra 8 caracteres nulos (en blanco)

System.out.println(vocales); //Muestra aeiou

El atributo length de un array de caracteres contiene el tamaño del array independientemente de que sean caracteres nulos u otros caracteres.

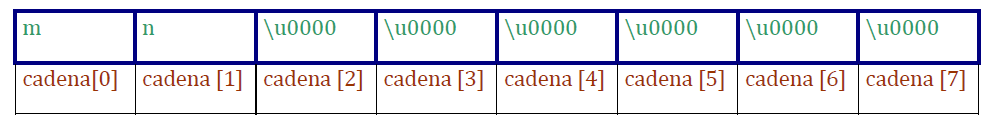
**Ej**: char [] cadena = new char[8];



System.out.println(cadena.length); // Muestra: 8

cadena[0] ='m';

cadena[1] ='n';



System.out.print(cadena);

System.out.print(cadena);

System.out.println(".");

Muestra: mnbbbbbbmnbbbbbb. //b representa los espacios en blanco

Se puede asignar un String a un array de caracteres mediante el método toCharArray() de la clase String.

Ej: String s = "Ordenador";



char [] palabra = s.toCharArray();

Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a palabra.



Se puede crear un String a partir de un array de caracteres.

**Ej:** char [] vocales = {'a', 'e', 'i', 'o', 'u'};

String s = new String(vocales);



Se crea un nuevo String con el contenido del array vocales y se asigna la dirección de memoria a s.



**7. Recorrer un array de caracteres unidimensional**

Se puede recorrer de forma completa utilizando una instrucción iterativa, normalmente un for.

**Ej:** char [] s = new char[10];

s[0]='a';

s[1]='b';

s[2]='c';

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

System.out.print(s[i]+ " "); //Muestra todos los caracteres del array,

// incluidos los nulos.

Si los caracteres no nulos se encuentran al principio del array se puede recorrer utilizando un while, mientras que no encontremos un carácter nulo.

**Ej**: char [] s = new char[10];

s[0]='a';

s[1]='b';

s[2]='c';

int i = 0;

while(s[i] != '\0'){

System.out.print(s[i]); // Muestra los caracteres del array hasta que

i++; // encuentra el primer nulo.

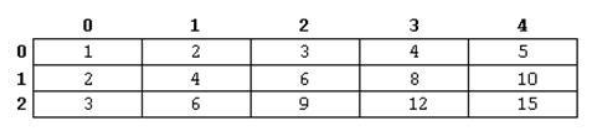
} // Ojo que si está lleno del todo da error

**8. Arrays Bidimensionales (Matrices)**

Un array puede tener más de una dimensión. El caso más general son los arrays bidimensionales, también llamados matrices o tablas. La dimensión de un array la determina el número de índices necesarios para acceder a sus elementos.

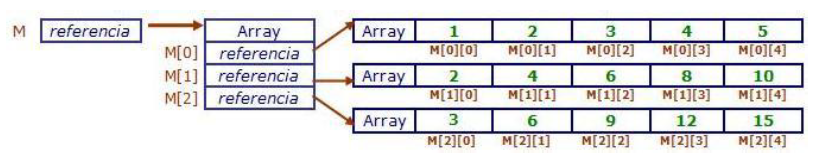
Una matriz necesita dos índices para acceder a sus elementos. Gráficamente se puede representar una matriz como una tabla de n filas y m columnas cuyos elementos son todos del mismo tipo.

La siguiente figura representa un array M de 3 filas y 5 columnas:



Pero en realidad una matriz en Java es un array de arrays.

Gráficamente, podemos representar la disposición real en memoria del array anterior así:



La longitud del array M (M.length) es 3. Se corresponde con el número de filas del array bidimensional.

La longitud de cada fila del array (M[i].length) es 5. Se corresponde con el número de columnas de cada fila del array bidimensional.

Para acceder a cada elemento de la matriz se utilizan **dos índices**. El primero indica la fila y el segundo la columna.

**9. Crear matrices en java**

Se crean de forma similar a los arrays unidimensionales, añadiendo un índice.

**Ej**:

Matriz de datos de tipo int llamado ventas de 4 filas y 6 columnas.

int [][] ventas = new int[4][6];

Matriz de datos double llamado temperaturas de 3 filas y 4 columnas.

double [][] temperaturas = new double[3][4];

En Java se pueden crear arrays irregulares en los que el número de elementos de cada fila es variable. Solo es obligatorio indicar el número de filas.

**Ej**: int [][] m = new int[3][]; //crea una matriz m de 3 filas.

A cada fila se le puede asignar un número distinto de columnas:

m[0] = new int[3];

m[1] = new int[5];

m[2] = new int[2];

Gráficamente podemos representar la disposición real en memoria del array anterior así:



**10. Inicializar matrices**

Una matriz es un objeto, por tanto, cuando se crea, a sus elementos se les da automáticamente un valor inicial, al igual que a los array unidimensionales.

- 0 para arrays numéricos

- '\u0000' (carácter nulo) para arrays de caracteres

- false para arrays booleanos

- null para arrays de String y de referencias a objetos.

También es posible dar otros valores iniciales a la matriz cuando se crea. Los valores iniciales se escriben entre llaves separados por comas. Los valores que se le asignen a cada fila aparecerán a su vez entre llaves separados por comas.

El número de valores determina el tamaño de la matriz.

**Ej**: int [][] numeros = {{6,7,5}, {3, 8, 4}, {1,0,2}, {9,5,2}};

Se crea la matriz numeros de tipo int, de 4 filas y 3 columnas, y se le da esos valores iniciales.

**11. Recorrer matrices**

Para recorrer una matriz se anidan dos bucles for. En general, para recorrer un array multidimensional se anidan tantas instrucciones for como dimensiones tenga el array.

Se usa siempre length para obtener el número de columnas que tiene cada fila

for (i = 0; i < a.length; i++) { //número de filas

for (j = 0; j < a[i].length; j++) //número de columnas de cada fila

System.out.print(a[i][j] + " ");

System.out.println();

}

**Ej**: Programa que lee por teclado números enteros y los guarda en una matriz de 5 filas y 4 columnas. A continuación muestra los valores leídos, el mayor y el menor y las posiciones que ocupan.

import java.util.\*;

public class Bidimensional1 {

public static void main(String[] args) {

final int FILAS = 5, COLUMNAS = 4;

Scanner teclado = new Scanner(System.in);

int i, j, mayor, menor;

int filaMayor, filaMenor, colMayor, colMenor;

int[][] A = new int[FILAS][COLUMNAS];

System.out.println("Lectura de elementos de la matriz: ");

for (i = 0; i < FILAS; i++)

for (j = 0; j < COLUMNAS; j++) {

System.out.print("A[" + i + "][" + j + "]= ");

A[i][j] = teclado.nextInt();

}

System.out.println("valores introducidos:");

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

for (j = 0; j < A[i].length; j++)

System.out.print(A[i][j] + " ");

System.out.println();

}

mayor = A[0][0]; //se toma el primero como mayor y menor

menor = A[0][0];

filaMayor = filaMenor = colMayor = colMenor = 0;

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

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

if (A[i][j] > mayor) {

mayor = A[i][j];

filaMayor = i;

colMayor = j;

} else if (A[i][j] < menor) {

menor = A[i][j];

filaMenor = i;

colMenor = j;

}

}

}

teclado.close();

System.out.print("Elemento mayor: " + mayor);

System.out.println(" Fila: "+ filaMayor + " Columna: " + colMayor);

System.out.print("Elemento menor: " + menor);

System.out.println(" Fila: "+ filaMenor + " Columna: " + colMenor);

}

}

**ArrayList**

La clase ArrayList permite el almacenamiento de datos en memoria de forma similar a los arrays convencionales, pero con la gran ventaja de que el número de elementos que puede almacenar es dinámico. La cantidad de elementos que puede almacenar un array está limitada a la dimensión que se declara a la hora de crearlo o inicializarlo. Los ArrayList, en cambio, pueden almacenar un número variable de elementos sin estar limitados por un número prefijado.

**1. Declaración de un objeto ArrayList**

La declaración genérica de un ArrayList se hace con un formato similar al

siguiente:

ArrayList nombreDeLista;

De esta manera no se indica el tipo de datos que va a contener, cosa que suele ser recomendable, para que así se empleen las operaciones y métodos adecuados para el tipo de datos manejado.

Para especificar el tipo de datos que va a contener la lista se debe indicar entre los caracteres '<' y '>' la clase de los objetos que se almacenarán:

ArrayList<nombreClase> nombreDeLista;

En caso de almacenar datos de un tipo básico como char, int, double, etc, se debe especificar el nombre de la clase asociada: Character, Integer, Double, etc.

**Ej**: ArrayList<String> listaPaises;

ArrayList<Integer> edades;

**2. Creación de un ArrayList**

Para crear un ArrayList se puede seguir el siguiente formato:

nombreDeLista = new ArrayList();

Se puede declarar la lista a la vez que se crea:

ArrayList<nombreClase> nombreDeLista = new ArrayList<nombreClase> ();

**Ej**: ArrayList<String> listaPaises = new ArrayList<String>();

La clase ArrayList forma parte del paquete java.util, por lo que hay que incluir en la parte inicial del código la importación de ese paquete.

**3. Añadir elementos al final de la lista**

El método **add** posibilita añadir elementos. Los elementos que se van añadiendo, se colocan después del último elemento que hubiera en el ArrayList. El primer elemento que se añada se colocará en la posición 0.

**Ej**:

ArrayList<String> listaPaises = new ArrayList<String>();

listaPaises.add("España"); //Ocupa la posición 0

listaPaises.add("Francia"); //Ocupa la posición 1

listaPaises.add("Portugal"); //Ocupa la posición 2

**Ej**: Se pueden crear ArrayList para guardar datos numéricos de igual manera

ArrayList<Integer> edades = new ArrayList<Integer> ();

edades.add(22);

edades.add(31);

edades.add(18);

**4. Añadir elementos en cualquier posición de la lista**

También es posible insertar un elemento en una determinada posición desplazando el elemento que se encontraba en esa posición, y todos los siguientes, una posición más. Para ello, se emplea también el método **add** indicando como primer parámetro el número de la posición donde se desea colocar el nuevo elemento.

**Ej**: ArrayList<String> listaPaises = new ArrayList<String>();

listaPaises.add("España");

listaPaises.add("Francia");

listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal

listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal

Si se intenta insertar en una posición que no existe, se produce una excepción (IndexOutOfBoundsException)

**5. Suprimir elementos de la lista**

Si se quiere eliminar un determinado elemento de la lista se puede emplear el, o método **remove** al que se le puede indicar por parámetro un valor int con la posición a suprimir bien, se puede especificar directamente el elemento a eliminar si es encontrado en la lista.

**Ej**: ArrayList<String> listaPaises = new ArrayList<String>();

listaPaises.add("España");

listaPaises.add("Francia");

listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal

listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal

listaPaises.remove(2); //Eliminada Francia, queda: España, Italia, Portugal

listaPaises.remove("Portugal"); //Eliminada Portugal, queda: España, Italia

**6. Consulta de un determinado elemento de la lista**

El método **get** permite obtener el elemento almacenado en una determinada posición que es indicada con un parámetro de tipo int. Con el elemento obtenido se podrá realizar cualquiera de las operaciones posibles según el tipo de dato del elemento (asignar el elemento a una variable, incluirlo en una expresión, mostrarlo por pantalla, etc).

**Ej**: System.out.println(listaPaises.get(3)); // Mostraría: Portugal

**7. Modificar un elemento contenido en la lista**

Es posible modificar un elemento que previamente ha sido almacenando en la lista utilizando el método **set**. Como primer parámetro se indica, con un valor int, la posición que ocupa el elemento a modificar, y en el segundo parámetro se especifica el nuevo elemento que ocupará dicha posición sustituyendo al elemento anterior.

**Ej**: En la lista de países se desea modificar el que ocupa la posición 1 (segundo en la lista) por "Alemania".

listaPaises.set(1, "Alemania");

**8. Buscar un elemento**

La clase ArrayList facilita mucho las búsquedas de elementos gracias al método **indexOf** que retorna, con un valor int, la posición que ocupa el elemento que se indique por parámetro. Si el elemento se encontrara en más de una posición, este método retorna la posición del primero que se encuentre. El método **lastIndexOf** obtiene la posición del último encontrado.

En caso de que no se encuentre en la lista el elemento buscado, se obtiene el valor -1.

**Ej**: Comprobar si Francia está en la lista, y mostrar su posición.

String paisBuscado = "Francia";

int pos = listaPaises.indexOf(paisBuscado);

if(pos!=-1)

System.out.println(paisBuscado + " encontrado en la posición: "+pos);

else

System.out.println(paisBuscado + " no se ha encontrado");

**9. Recorrer el contenido de la lista**

Es posible obtener cada uno de los elementos de la lista utilizando un bucle con tantas iteraciones como elementos contenga, de forma similar a la usada con los arrays convencionales. Para obtener el número de elementos de forma automática se puede emplear el método **size**() que devuelve un valor int con el número de elementos que contiene la lista.

**Ej**: for (int i=0; i<listaPaises.size(); i++)

System.out.println(listaPaises.get(i));

También se puede emplear otro formato del bucle for (bucle **foreach**) en el que se va asignando cada elemento de la lista a una variable declarada del mismo tipo que los elementos del ArrayList.

**Ej**: for (String pais:listaPaises)

System.out.println(pais);

Si el ArrayList contiene objetos de tipos distintos o se desconoce el tipo se pondría.

for(Object o: nombreArrayList)

También es posible hacerlo utilizando un objeto **Iterator**. La ventaja de usar un Iterador es que no se necesita indicar el tipo de objetos que contiene el ArrayList. Iterator tiene como métodos:

**hasNext**: devuelve true si hay más elementos en el array.

**next**: devuelve el siguiente objeto contenido en el array

**remove**: borra el elemento que se está recorriendo dentro del iterador

**Ej**: ArrayList<Integer> nros = new ArrayList<Integer>();

//se insertan elementos

Iterator<Integer> it = nros.iterator();

//se crea el iterador it para el ArrayList nros

while(it.hasNext()) //mientras queden elementos

System.out.println(it.next()); //se obtienen y se muestran

**10. Otros métodos**

void **clear**(): Borra todo el contenido de la lista.

Object **clone**(): Retorna una copia de la lista.

boolean **contains**(Object elemento): Retorna true si se encuentra el elemento indicado en la lista, y false en caso contrario.

boolean **isEmpty**(): Retorna true si la lista está vacía.

**11. Ejemplos de uso de ArrayList**

**Ej**:

ArrayList<String> nombres = new ArrayList<String>();

nombres.add(“Ana”);

nombres.add(“Luisa”);

nombres.add(“Felipe”);

System.out.println(nombres); // [Ana, Luisa, Felipe]

nombres.add(1, “Pablo”);

System.out.println(nombres); // [Ana, Pablo, Luisa, Felipe]

nombres.remove(0);

System.out.println(nombres); // [Pablo, Luisa, Felipe]

nombres.set(0,”Alfonso”);

System.out.println(nombres); // [Alfonso, Luisa, Felipe]

String s = nombres.get(1);

String ultimo = nombres.get(nombres.size() – 1);

System.out.println(s + “ “ + ultimo); // Luisa Felipe

**Ej**: Escribe un programa que lea números enteros y los guarde en un ArrayList hasta que se lea un 0 y muestra los números leídos, su suma y su media.

import java.util.\*;

public class ArrayList2 {

public static void main(String[] args) {

Scanner teclado = new Scanner(System.in);

ArrayList<Integer> numeros = new ArrayList<Integer>();

int n;

do {

System.out.println(“Introduce números enteros. 0 para acabar: “);

System.out.println(“Numero: “);

n = teclado.nextInt();

if (n != 0)

numeros.add(n);

}while (n != 0);

System.out.println(“Ha introducido: “ + numeros.size() + “ números:”);

System.out.println(numeros); //Muestra el arrayList completo

Iterator<Integer> it = numeros.iterator();

while(it.hasNext())

System.out.println(it.next());

double suma = 0;

for(Integer i: numeros)

suma = suma + i;

System.out.println(“Suma: “ + suma);

System.out.println(“Media: “ + suma/ numeros.size());

}

}

**12. Copiar un ArrayList**

El nombre de un ArrayList contiene la referencia al ArrayList, es decir, la dirección de memoria donde se encuentra el ArrayList, igual que sucede con los arrays estáticos.

**Ej**: Se tiene un ArrayList de enteros llamado ventas.



La instrucción ArrayList<Integer> ventas1 = ventas; no copia el array ventas en el nuevo array ventas1 sino que crea una referencia.



De esta forma se tiene dos formas de acceder al mismo ArrayList: mediante la referencia ventas y mediante la referencia ventas1.

Para hacer una copia se puede hacer de forma manual elemento a elemento o se puede pasar la referencia del ArrayList original al constructor del nuevo. Shallow & Deep copy ver.

ArrayList<Integer> ventas1 = new ArrayList<Integer>(ventas);



**13. ArrayList como parámetro de un método**

Un ArrayList puede ser usado como parámetro de un método. Además un método también puede devolver un ArrayList mediante la sentencia return.

**Ej**: Método que recibe un ArrayList de String y lo modifica invirtiendo su contenido.

import java.util.\*;

public class ArrayList4 {

public static void main(String[] args) {

ArrayList<String> nombres = new ArrayList<String>();

nombres.add("Ana");

nombres.add("Luisa");

nombres.add("Felipe");

nombres.add("Pablo");

System.out.println(nombres);

nombres = invertir(nombres);

System.out.println(nombres);

}

public static ArrayList<String> invertir(ArrayList<String> nombres) {

ArrayList<String> resultado = new ArrayList<String>();

for (int i = nombres.size() - 1; i >= 0; i--)

resultado.add(nombres.get(i));

return resultado;

}

}

Como ejercicio, hacer un nuevo método invertir, que modifique el ArrayList que se recibe en lugar de devolver una copia:

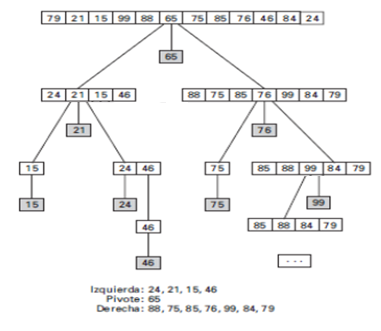
public static void invertir(ArrayList<String> nombres)

# UT5 Algoritmos de Ordenación de Arrays

**Intercambio**: Consiste en comparar el primer valor con el resto de las posiciones posteriores, cambiando el valor de las posiciones en caso de que el segundo sea menor que el primero comparado. Después la segunda posición con el resto de posiciones posteriores y así sucesivamente hasta que el array se ordena. Se realizan muchas pasadas sobre el array, por lo que es lento y costoso.

**Burbuja**: Consiste en comparar el primero con el segundo, si el segundo es menor que el primero se intercambian los valores. Después el segundo con el tercero y así sucesivamente, cuando no haya ningún intercambio, el array estará ordenado. Lo peor de este método de ordenación, es que tiene una complejidad de **O(n2)** haciendo que cuantos más valores a ordenar tengamos, mayor tiempo tardará en ordenar.

**Quicksort**: Consiste en ordenar un array mediante un pivote, que tomaremos como un punto intermedio en el array. Es como si se ordenaran pequeños trozos del array, haciendo que a la izquierda estén los menores a ese pivote y en la derecha los mayores a este. Después se vuelve a calcular el pivote de trozos de listas. Usa recursividad. Le pasamos el array, su posición inicial y su posición final como parámetro. Tiene una complejidad de **O(n log2 n),** haciendo que mejore el rendimiento aun teniendo muchos valores que ordenar.



**Sort**: Consiste en usar el método **sort** de **java.util.Arrays**.

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html>

Para ejecutarlo escribimos:

**Arrays.sort (array a ordenar);**

Simplemente insertamos como parámetro el array que queremos ordenar.

Por ejemplo, dado el siguiente array de Strings:

String [] nombres = {"juan", "pedro", "ana", "maria", "felipe", "luis", "eduardo"};

Con el método sort, se ordena ascendentemente con la instrucción:

**Arrays.sort(nombres);**

Para ordenar un array de forma descendente (de mayor a menor) hay que indicarlo utilizando el método **reverseOrder()** de la clase **Collections**.

Para utilizar reverseOrder es necesario incluir el import:

**import java.util.Collections;**

Por ejemplo, para ordenar el array nombres de forma descendente escribimos la instrucción Arrays.sort de la siguiente forma:

**Arrays.sort(nombres, Collections.reverseOrder());**

Con Arrays.sort podemos ordenar arrays de cualquier tipo de datos. Por ejemplo, para ordenar un array de enteros:

int [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};

Arrays.sort(numeros);

// Mostrarlo ordenado

for (int n : numeros) {

System.out.println(n);

}

*Collections.reverseOrder()* solo funciona para arrays de objetos. Por este motivo si queremos ordenar de forma descendente arrays de tipos de datos simples debemos utilizar la **clase envolvente** equivalente al tipo de dato básico.

Por ejemplo, para ordenar un array de enteros de forma descendente hay que declararlo de tipo Integer en lugar de int.

Integer [] numeros = {3, 5, 1, 2, 1, 7, 0, -1};

Arrays.sort(numeros, Collections.reverseOrder());

for (int n : numeros) {

System.out.println(n);

}

**ALGORITMOS DE BÚSQUEDA BINARIA EN ARRAYS**

Es un [algoritmo de búsqueda](https://es.wikipedia.org/wiki/Algoritmo_de_búsqueda" \t "Algoritmo de búsqueda) que encuentra la posición de un valor en un [array ordenado](https://es.wikipedia.org/w/index.php?title=Arreglo_ordenado&action=edit&redlink=1" \t "Arreglo ordenado (aún no redactado)).

Compara el valor con el elemento que tengamos en medio del array, si no son iguales, la mitad en la cual el valor no puede estar es eliminada y la búsqueda continúa en la mitad restante hasta que el valor se encuentre.

Tienes un ejemplo en la implementación del algoritmo QuickSort de los ejemplos.

# UT5 Ejercicios Arrays 1. Entrega 1

**1.-** Escribe el valor final de las variables:

**a)** double[] A= new double[3];

double x;

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

A[i] = i\*3;

x = A[0]+A[1]+A[2];

El valor de x es \_\_\_\_ 0+3+6

El valor de A[0] es \_\_\_\_0

El valor de A[1] es \_\_\_\_ 3

El valor de A[2] es \_\_\_\_ 6

**b)** int[] B = new int[5];

B[4]=1;

B[B[4]]=2;

B[B[B[4]]]=0;

B[B[B[B[4]]]]=3;

B[B[B[B[B[4]]]]]=4;

El valor de B[0] es \_\_\_\_3

El valor de B[1] es \_\_\_\_2

El valor de B[2] es \_\_\_\_0

El valor de B[3] es \_\_\_\_4

El valor de B[4] es \_\_\_\_1

*se va sustituyendo en las matrices*

B→ 0 0 0 0 0

0 1 2 3 4

**c) int[] C= new int[4];**

**int y;**

**for (int i=C.length-1; i>=0; i--)**

**C[i] = i/2;**

**y = C[0]+C[1]+C[2]+C[3];**

El valor de y es \_\_\_\_2

El valor de C[0] es \_\_\_\_(0)/2

El valor de C[1] es \_\_\_\_(1)/2

El valor de C[2] es \_\_\_\_(2)/2

El valor de C[3] es \_\_\_\_(3)/2

**2.-** Escribe el código en Java de los siguientes procedimientos:

a) Un procedimiento que recibe un array de enteros y le asigna a cada componente los valores 0,3,6,..,3\*(n-1) donde n es el indice del array.

b) Un procedimiento que recibe un array de enteros y le asigna a cada componente los valores n-1, n-2, ...,2,1,0 donde n es la longitud del array.

c) Un procedimiento que recibe un array de valores booleanos y le asigna de manera intercalada los valores true y false.

**3.-** Escribe el código en Java de los siguientes ejercicios:

tienen que ser static

a) El método public int cuentaCeros ( int arr[ ] ) que recibe un array de enteros y devuelve el número de ceros que se encuentran dentro del array.

b) El método public int sumaPares ( int arr[ ] ) que recibe un array de enteros y devuelve el resultado de sumar sólo los números pares que hay dentro del array.

c) El método public int cuentaRepeticiones ( int arr[ ], int x ) que recibe un array de enteros y un valor entero x. El método devuelve el número de veces que se repite el valor de x en el array.

d) El método public void sustituye (int arr[ ], int viejo, int nuevo) que recibe un array de enteros y dos valores enteros viejo y nuevo. El método debe reemplazar todos los valores viejo del array por el valor de nuevo.

e) El método public void intercambia (int arr[ ], int x, int y), que recibe un array de enteros y dos valores enteros que corresponden a dos componentes del array e intercambie los valores de las componentes del array. El ejercicio se hará teniendo en cuenta que x e y son el contenido de dos posiciones del array y después se hará teniendo en cuenta que x e y son las posiciones del array.

f) El método public void invierte (int arr[ ]), que recibe un array de enteros e invierte la secuencia de valores del array.

g) El procedimiento public void rotaDerecha (int arr[ ]), que recibe un array de enteros y mueve a cada elemento una posición adelante, colocando el último valor del array en la primera componente del array resultante.

h) El método public boolean iguales (int a1[ ], int a2[ ]), que recibe dos arrays de enteros y devuelve true si los dos arrays contienen la misma secuencia de valores y false de otra manera**.**

**4.-** Programa Java que lea por teclado 10 números enteros y los guarde en un array. A continuación, calcula y muestra por separado la media de los valores positivos y la de los valores negativos.

**5.-** Programa Java que lea 10 números enteros por teclado y los guarde en un array.

Calcula y muestra la media de los números que estén en las posiciones pares del array. Considerar la primera posición del array (posición 0) como par.

**6.-** Haz un programa en Java para leer la altura de N personas y calcular la altura media. Calcular cuántas personas tienen una altura superior a la media y cuántas tienen una altura inferior a la media. El valor de N se pide por teclado y debe ser entero positivo.

**7.-** Haz un programa que cree un array de números y otro de String de un número de elementos que pondrás en una constante, que contendrán:

El primero, notas entre 0 y 10 (debemos controlar que inserte una nota válida), pudiendo tener decimales.

En el segundo, se insertarán los nombres de alumnos.

Después, se creará un tercer array de Strings, donde se insertará la nota en forma de cadena, de la siguiente forma:

Si la nota está entre 0 y 4,99 , será un suspenso

Si está entre 5 y 6,99 , será un bien.

Si está entre 7 y 8,99 será un notable.

Si está entre 9 y 10 será un sobresaliente.

Finalmente, mostrar por pantalla el nombre de cada alumno, su nota numérica y su nota alfabética usando los tres arrays.

# UT5 Ejercicios Arrays 1. Entrega 2

**1.** Haz un programa que rellene un array con los 100 primeros números enteros y los muestre en pantalla en orden inverso al orden en que se han introducido (orden inverso).

**2.** Haz un programa que cree un array de 10 posiciones de números aleatorios entre 1 y 100. Posteriormente se pedirá por teclado una posición y se mostrará en pantalla qué valor tiene esa posición.

Math.random() → Devuelve un numero entre 0.0 y 1.0

Math.floor(Math.random()\*6) → Devuelve un numero entre 0 y 5

**3.** Haz un programa que cree un array cuyo tamaño se pedirá por teclado y cuyo contenido serán números aleatorios entre 1 y 300. Posteriormente se creará otro array que guardará aquellos números del primer array que acaben en un dígito que se indicará por teclado (se debe controlar que se introduce un numero correcto). Finalmente, mostrar por pantalla los dos arrays.

**4.** Haz un programa que calcule la letra de un DNI. Se pedirá el DNI por teclado y devolverá la letra correspondiente al DNI. Para buscar la letra, se calcula el resto de dividir el número de dni entre 23, y con el resultado que estará entre 0 y 22, se busca en el array de caracteres la letra correspondiente en esa posición.

El orden de los caracteres es: T, R, W, A, G, M, Y, F, P, D, X, B, N, J, Z, S, Q, V, H, L, C, K, E

**5.** Haz un programa que pida un número por teclado y después diga si el número introducido es capicúa o no.

**6.** Haz un programa que genere 100 números aleatorios entre 0 y 10 y cree un histograma con las frecuencias de cada número. Para representar las barras del histograma se utilizará secuencias ‘\*’.

Por ejemplo, la secuencia: 1, 1, 3, 4, 1, 3, 1, 2, generaría la siguiente salida:

1: \*\*\*\*

2: \*

3: \*\*

4: \*

# UT5 Ejercicios Matrices (Arrays Bidimensionales)

**1.** Crear dos matrices de dimensión 4x4 de enteros (valores aleatorios de 0 a 9) y obtener una tercera matriz correspondiente a la suma de las dos.

**2.** Escribir un programa que cree una matriz entera de tamaño 4x5 (valores aleatorios de 0 a 9) y un valor entero (también aleatorio), y muestre en pantalla la posición de la primera coincidencia del valor en la matriz.

**3.** Dada una matriz de 8x8, hacer un programa que cree una matriz idéntica y la escriba en pantalla. *(con métodos ha dicho)*

**4.** Escribir un programa que cree una matriz entera de tamaño 6x8 (valores aleatorios de 1 a 100) y obtenga y escriba en pantalla el valor máximo y mínimo de toda la matriz.

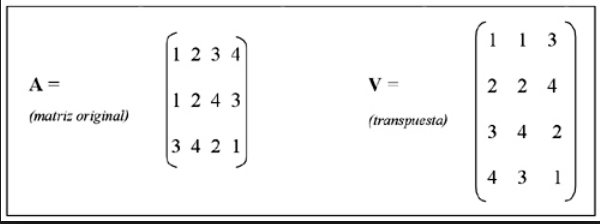
**5.** Dada una matriz de dimensión 5x5:

* Elevar al cuadrado los elementos situados por encima de la diagonal principal.
* Sumar los elementos situados por debajo de la diagonal principal.
* Sumar 1 a elementos de la diagonal principal.
* Multiplicar por 2 los elementos de la diagonal inversa.

**6.** Escribir un programa que lea una matriz de 4x5 de números enteros (valores aleatorios de 0 a 9), calcule la suma de cada fila y de cada columna y muestre por pantalla la nueva tabla, incluyendo las sumas de las filas como una sexta columna y de las columnas como una quinta fila. No se crea una nueva matriz más grande, solo se imprime.

**7.** Hacer un programa que transponga una matriz de 4x4.

* Primero haciendo el cambio sobre la propia matriz.
* Segundo generando una nueva matriz.



**8.** Se dispone de tres matrices unidimensionales. La primera de ellas contiene los nombres de pila de un grupo de N personas. La segunda matriz contiene los primeros apellidos y la tercera los segundos apellidos.

Crear una nueva matriz unidimensional que contenga para cada persona, en cada celda el nombre junto con los dos apellidos.

Crea otra matriz (bidimensional ahora) en la que en la primera fila tengamos los nombres, en la segunda fila los primeros apellidos y en la tercera los segundos apellidos.

**9.** En un campeonato de baloncesto intervienen 20 equipos de 10 miembros cada uno. Se quiere hacer un programa que lleve a cabo las siguientes operaciones:

**a.** Leer y almacenar la altura de cada jugador en la estructura de datos que consideres más adecuada.

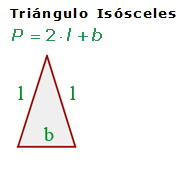
**b.** Determinar qué equipo tiene mayor altura media.

**c.** Formar una selección de 20 miembros integrada por el jugador más alto de cada equipo, guardándola en la estructura de datos que consideres más adecuada.

# UT5 Ejercicios ArrayList

**1.-** Haz una clase para trabajar con triángulos isósceles (2 lados iguales). Dicha clase tendrá como atributos la base y la altura que serán privados. También tendrá como mínimo un constructor y dos métodos para calcular el área y el perímetro de un triángulo, y todos aquellos métodos que sean necesarios para el buen funcionamiento del programa.

Fórmula a aplicar:



perimetro = 2 \* lado + base, siendo lado la medida del lado repetido

lado^2 = (base/2)^2 + altura^2

lado = Raiz Cuadrada((base/2)^2 + altura^2)

Crea una clase Principal que cree un Array/ArrayList de triángulos en el que se realicen las siguientes operaciones:

1. Añadir un triángulo.
2. Calcular e imprimir el triángulo con el área más grande.
3. Calcular e imprimir el triángulo con el perímetro más pequeño.
4. Imprimir la altura, la base, el área y el perímetro de un triángulo en concreto.
5. Imprimir la altura, la base, el área y el perímetro de todos los triángulos.

**2.-** Dada la clase Viaje siguiente, escribir un método denominado uneViaje que reciba como parámetros dos tipos Viaje y devuelva un nuevo objeto de esa misma clase con:

el origen del primero, el destino del segundo y como distancia la suma de las distancias de los dos viajes originales, si el destino del primero coincide con el origen del segundo, sino se cumple dicha condición se asignará -1 a la distancia.

class Viaje {

private String origen;

private String destino;

private double distancia;

}

*Nota*: La función para comparar cadenas es: cadena1.compareTo(cadena2) devuelve 0 si las dos cadenas son iguales, <0 si la cadena1 < cadena2 y >0 si cadena1>cadena2.

Crea una clase Principal, en la que se defina un array/ArrayList de 10 objetos Viaje e imprima por pantalla el resultado de usar el método uneViaje, con una componente y la siguiente (0 y 1, 1 y 2, 2 y 3, … ,9 y 10).

3. Un punto en el plano se puede representar mediante un par de coordenadas x e y, ambas tomando valores en el conjunto de los números reales. En Java podemos representar un punto en el plano mediante una instancia de la siguiente clase:

public class Punto {

private double x;

private double y;

/\* métodos \*/

/\* A completar en el resto de apartados \*/

}

En Java, el constructor de la clase sirve para inicializar los valores de los atributos a la hora de instanciarse un objeto de la misma en tiempo de ejecución. Programa un constructor para la clase Punto. Debe recibir como parámetros las dos coordenadas del punto.

El método toString() tiene un significado especial en los objetos Java. Es el método que se utiliza para obtener una representación como cadena de texto de dicho objeto.

Programa el método toString() para que devuelva una cadena de texto con la representación del punto con el siguiente formato: (x, y)

Donde x e y deben ser reemplazados por sus respectivos valores. El prototipo de dicha función se especifica a continuación:

public String toString() {

/\* ... \*/

}

**Pista**

Recuerda que el operador "+", aplicado a cadenas de texto, permite concatenarlas. Si se concatenan números a cadenas, Java los convierte automáticamente a cadenas de texto para realizar la concatenación.

Crea un paquete llamado *paqPunto* para guardar en él la clase punto y la clase Prueba que se te pide más adelante.

Programa una clase llamada **Prueba** que tenga un método main para probar el código anterior. Este método ***debe recibir como argumentos*** de línea de comandos las coordenadas x e y, crear un nuevo objeto Punto con dichas coordenadas e imprimir en su salida estándar (la pantalla en este caso) la representación textual de dicho objeto.

El programa debe comprobar que el número de argumentos de línea de comandos recibido sea el correcto.

**Pista**

El método parseDouble de la clase [Double](http://docs.oracle.com/javase/7/docs/api/java/lang/Double.html) transforma una cadena de texto a un tipo primitivo double.

Es habitual que los atributos de un objeto se declaren como privados (private) para evitar accesos incontrolados desde otros puntos del código. Si es necesario que desde el exterior se acceda a estos atributos, se proporcionan métodos cuyo nombre empieza, por convenio, con las cadenas "get" (acceso en lectura) y "set" (acceso en escritura).

Programa en la clase Punto los siguientes métodos, que devuelven el valor de las coordenadas x e y. El prototipo de dichos métodos se muestra a continuación:

public double getX() {

/\* ... \*/

}

public double getY() {

/\* ... \*/

}

public void setX(double x) {

/\* ... \*/

}

public void setY(double y) {

/\* ... \*/

}

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

**Cálculo de distancias.**

En este apartado vamos a implementar dos métodos auxiliares que nos permitirán calcular distancias entre puntos.

Programa un método de la clase Punto que devuelva la distancia del punto al origen de coordenadas. El prototipo del método se muestra a continuación:

public double distanciaAlOrigen() {

/\* completar \*/

}

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

Programa un método en Punto que devuelva la distancia entre el punto representado por la instancia actual del objeto y otra instancia de Punto que se recibe como parámetro. El prototipo del método se muestra a continuación:

public double calcularDistancia(Punto otroPunto) {

/\* completar \*/

}

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

**Cálculo de cuadrante.**

Programa un método de la clase Punto que devuelva el cuadrante en el que se encuentra el punto.

Devuelve 0 si está en el origen de coordenadas o sobre alguno de los ejes.

Devuelve 1 si está en el primer cuadrante (x e y positivos).

Devuelve 2 si está en el segundo cuadrante (x negativo e y positivo).

Devuelve 3 si está en el tercer cuadrante (x e y negativos).

Devuelve 4 si está en el cuarto cuadrante (x positivo e y negativo).

El prototipo del método se muestra a continuación:

public int calcularCuadrante() {

/\* completar \*/

}

Modifica el código de tu clase de prueba para comprobar que su comportamiento es correcto.

**Cálculo del punto más cercano.**

En este apartado tienes que basarte en los métodos de los apartados anteriores.

Programa un método en Punto que reciba como parámetro un array de objetos de la clase Punto y devuelva una referencia al objeto de dicho array que esté más cercano al punto actual. El prototipo del método se muestra a continuación:

public Punto calcularMasCercano(Punto[] otrosPuntos) {

/\* completar \*/

}

Modifica el código de tu clase Prueba para comprobar que su comportamiento es correcto.

**La clase Triangulo.**

Un triángulo está plenamente definido por sus tres vértices. Estos vértices se pueden representar como objetos de la clase Punto.

**Declaración de la clase.**

En este apartado tienes que basarte en los métodos de los apartados anteriores.

Programa la clase Triangulo con sus atributos y un constructor que reciba como parámetro tres puntos del plano, así como sus geter, seter y toString. Guárdala en un paquete que llamarás paqTriangulo, en él guardarás también la clase **Prueba** donde probarás el método que se describe a continuación.

**Longitud de los lados.**

Programa el método calcularLongitudLados() de la clase Triangulo, que debe devolver un array de tres posiciones, cada una de las cuales debe ser la longitud de uno de los lados del triángulo. El prototipo del método se muestra a continuación:

public double[] calcularLongitudLados() {

/\* completar... \*/

}

Crea un ArrayList con algunos triángulos que pedirás al usuario para guardarlos en el programa e imprímelos. (no hace falta menú, solo es para repasar conceptos vistos hasta ahora)

EJERCICIO DE COMPOSICIÓN DE OBJETOS

**4**. Se pide hacer una aplicación en java que represente una finca mediante una clase.

De la finca se quiere guardar su nombre y el número de metros cuadrados que posee.

Además, se quiere guardar información sobre la casa y la parcela que la componen.

De la casa se quiere saber los metros cuadrados, el número de plantas y el número de habitaciones que posee.

De la parcela se quiere saber también el número de metros cuadrados que tiene y si se dedica a explotación agrícola y/o ganadera.

Una vez que se haya modelizado todo mediante clases, se añadirá un menú que permita:

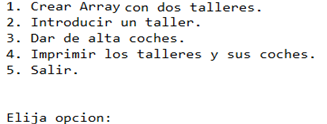
* Crear una nueva finca y guardarla.
* Mostrar los datos de una finca dado el nombre de la misma, que se pedirá al usuario.
* Mostrar todas las fincas con todos sus datos.

El ejercicio lo harás usando un arrayList para almacenar cada una de las fincas.

**5.-** Realiza un programa que simule el funcionamiento de dos talleres de reparación de coches. Lo primero que hará el programa será crear un Array con dos talleres.

Cada taller tendrá nombre, CIF y dirección, que se pedirán al usuario. Cada vez que llega un coche nuevo se da de alta en uno de los dos talleres (se le pregunta al usuario a qué taller va el coche. Para cada coche se pide su matrícula y una descripción de la tarea a realizar en él, así como el NIF del dueño.

Se pide realizar un menú con las siguientes opciones:



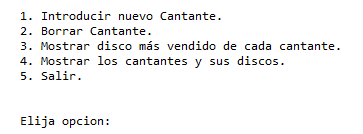
**6.-** Ejercicio para guardar información sobre cantantes famosos. Crearemos una clase denominada ListaCantantes que disponga de un atributo de tipo ArrayList para guardar cantantes. La clase debe tener un método que permita añadir objetos de tipo CantanteFamoso en la lista, otro método para eliminar un cantante y dos métodos para mostrar el disco más vendido de cada cantante y todos los discos de cada cantante (todo son opciones del menú que se muestra más abajo).

Cada objeto de tipo CantanteFamoso tendrá como atributos:

* nombre (String)
* discos (lista con los discos)
* y los métodos para obtener y establecer los atributos.

En el atributo discos, se guardará un listado de todos los discos editados por el cantante. Para cada uno de los discos se dispondrá de su título y el número de ejemplares vendidos.

Crea una clase principal que muestre un menú, con las siguientes opciones:



**7.-** Definir los atributos de las clases de objetos necesarias para almacenar la información relativa a unas muestras de alcoholemia recogidas en un análisis preventivo de la Dirección General de Tráfico, de forma que:

* 1. Para cada conductor interesa tener su nombre, DNI. y una colección de referencias a las muestras que se le han tomado durante el período de estudio.
  2. Cada muestra tendrá los siguientes datos: Día y hora de la muestra, código del puesto de control preventivo (dos letras y dos números), matrícula del vehículo y la tasa de alcohol espirado en aire (entre 0 y 2.5 mg/l).

Hacer una simulación del funcionamiento de estas clases usando el siguiente menú:

1. Introducir un nuevo conductor.  
2. Introducir una muestra.  
3. Mostrar los datos de un conductor.  
4. Mostrar las muestras de un conductor.  
5. Mostrar los datos de todos los conductores.  
6. Salir.

Elige una opción:

Hacerlo con Arrays y ArrayList.

Al introducir una muestra, se elige de qué conductor por su dni.  
Los datos del conductor se muestran buscando por dni.