TEMA 5: Arrays

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:

Arraynotas:

8.50	6.35	5.75	8.50	. . .	3.75	6.00	7.40
notas[0]	notas[1]	notas[2]	notas[3]	. . .	notas[17]	notas[18]	notas[19]

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

Los índices deben ser enteros no negativos.

2. Crear arrays unidimensionales

Para crear un array se deben realizar dos operaciones:

1. Declaración
2. Instanciación

1. Declarar 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:

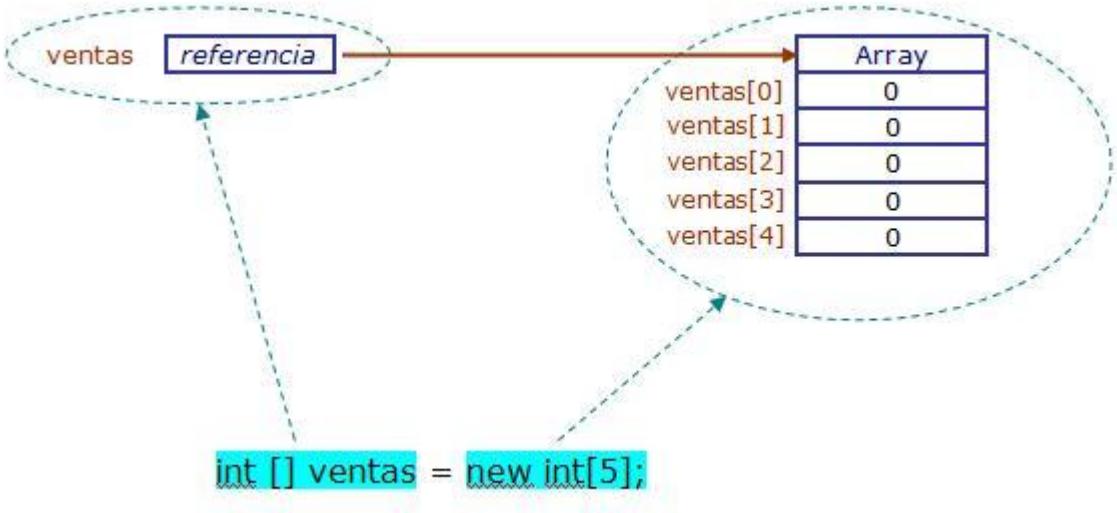
tipo [] nombreArray = new tipo[tamaño];

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.



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).

Ej:

```
int m = 5;
int [] a = new int[5];
```

0	0	0	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

```
a[1] = 2;
```

0	2	0	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

```
a[2] = a[1];
```

0	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

```
a[0] = a[1] + a[2] + 2;
```

6	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

```
a[0]++;
```

7	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

```
int m = 5;
a[3] = m + 10;
```

7	2	2	15	0
a[0]	a[1]	a[2]	a[3]	a[4]

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.

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] + " ");
```

Para evitar errores de acceso al array es recomendable utilizar length para recorrer el array completo.

```
Ej: double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0};
    for (int i = 0; i< notas.length; 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();
            }
            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 efecto se inicializa con el carácter nulo.

```
char [] cadena = new char[8];
```

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

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

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

a	e	i	o	u
vocales[0]	vocales [1]	vocales [2]	vocales [3]	vocales [4]

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];

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.println(cadena.length); // Muestra: 8
cadena[0] ='m';
cadena[1] ='n';
```

m	n	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.print(cadena);
System.out.print(cadena);
System.out.println(".");
```

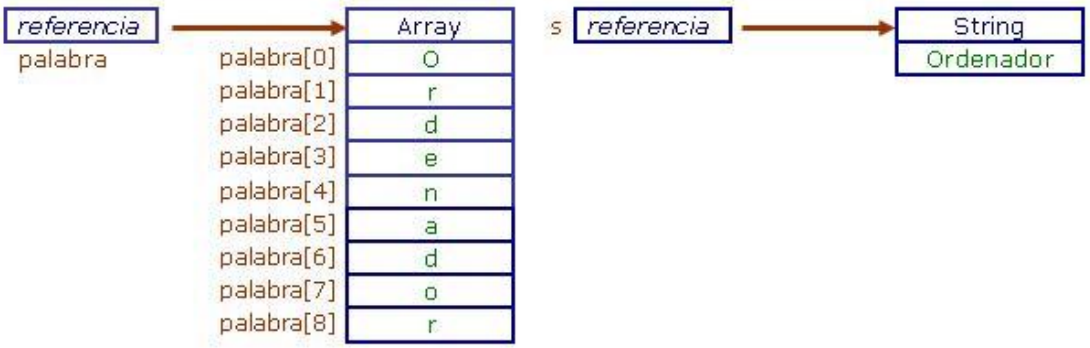
Muestra: mnbbbbbbmnbbbbbb. //b representa los espacios en blanco

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();

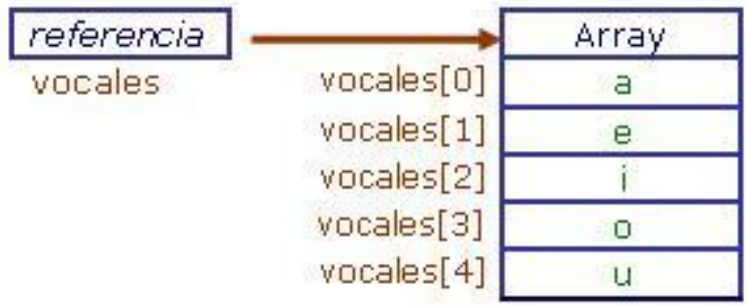
Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a palabra.



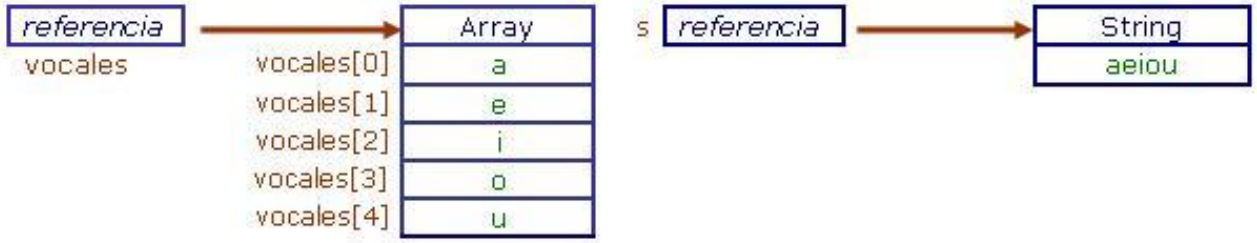
Se puede crear un String a partir de un array de caracteres.

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.
    }
```

8. Arrays Bidiminesionales (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.

Los arrays unidimensionales porque solo utilizan un índice para acceder a cada elemento.

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:

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

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.

La longitud de cada fila del array (M[i].length) es 5.

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.

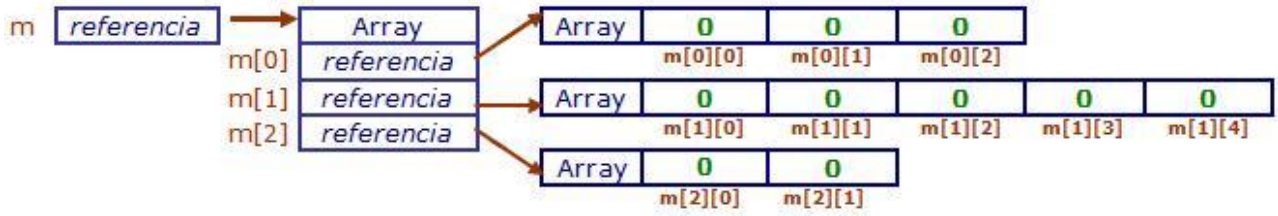
A cada fila se le puede asignar un número distinto de columnas:

```
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 anidad tantas instrucciones for como dimensiones tenga el array.

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 Bidimensional {
   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;
                     }
                }
            }
            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);
        }
    }
```

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();
}
```

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 limitado 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();
```

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

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. En primer elemento que se añada se colocará en la posición 0.

Ej:
```
ArrayList<String> listaPaises = new ArrayList();
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();
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();
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 método remove al que se le puede indicar por parámetro un valor int con la posición a suprimir, o bien, se puede especificar directamente el elemento a eliminar si es encontrado en la lista.

Ej:
```
ArrayList<String> listaPaises = new ArrayList();
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: nombreArray)
      System.out.println(o);
```

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.

```
Ej: ArrayList<Integer> nros = new ArrayList<Integer>();
    //se insertan elementos
    Iterator 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 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/ numerous.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.

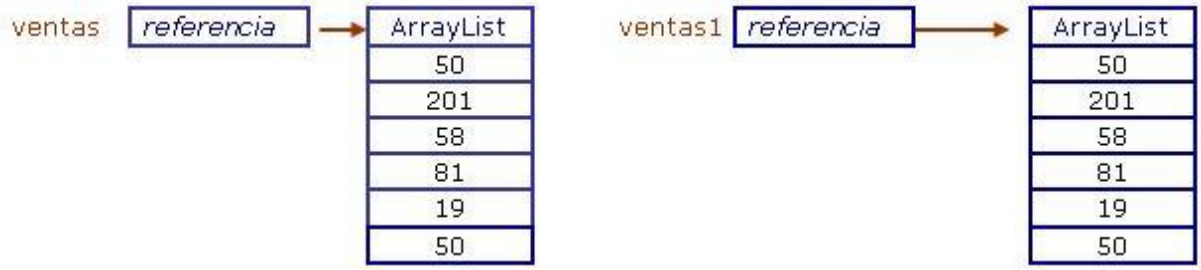
Ej: Se tiene un ArrayList de enteros llamado ventas.

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.

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;
    }
}
```