Arrays unidimensionales en Java

Concepto de Array o Arreglo:

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

Por ejemplo, supongamos que queremos guardar las notas de los 20 alumnos de una clase. Para almacenar las notas utilizaremos un array de 20 elementos de tipo double y en cada elemento del array guardaremos la nota de cada alumno

Podemos representar gráficamente el array de notas de la siguiente forma:

Array **notas**:

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 anterior, notas[0] contiene la nota del primer alumno y notas[19] contiene la del último

Los índices deben ser enteros no negativos.

1. CREAR ARRAYS UNIDIMENSIONALES

Para crear un array se deben realizar dos operaciones:

- Declaración
- Instanciación

Declarar de un array

En la declaración se crea la **referencia** al array.

La referencia será el nombre con el que manejaremos el 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 en java de cualquiera de estas dos formas:

```
tipo []  nombreArray;

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. Es el nombre que se usará en el programa para manejarlo.

Por ejemplo:

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

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

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

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 del array.

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 del array 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.

Por ejemplo:

```
ventas = new int[5]; //se reserva memoria para 5 enteros y
                                    //se asigna la dirección de inicio del array a ventas.
```

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

tipo [] nombreArray = new tipo[tamaño];

Por ejemplo: 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:

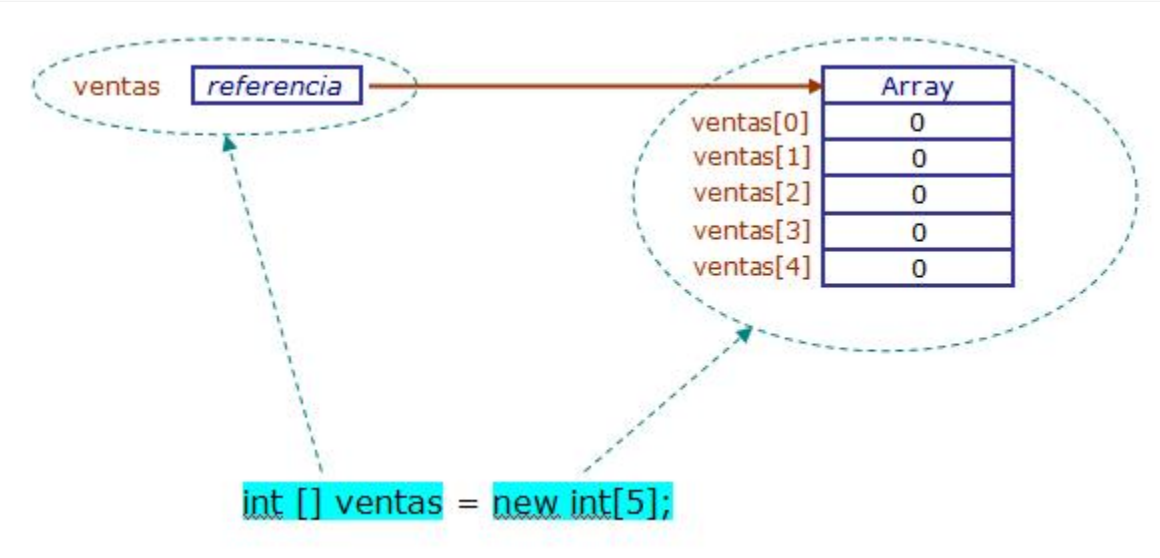
```
Scanner sc = new Scanner(System.in);
System.out.print("Número de elementos del array: ");
int numeroElementos = sc.nextInt();
int [] ventas = new int[numeroElementos];
```

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

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.



2. INICIALIZAR ARRAYS UNIDIMENSIONALES

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial:

Valores iniciales por defecto para un array en java:

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 podemos dar otros valores iniciales al array cuando se crea.

Los valores iniciales se escriben entre llaves separados por comas.

Los valores iniciales deben aparecer en el orden en que serán asignados a los elementos del array.

El número de valores determina el tamaño del array.

Por ejemplo:

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

se declara el array notas de tipo double, se reserva memoria para 6 elementos y se les asignan esos valores iniciales.

Ejemplos:

```
//creación de un array de 4 elementos booleanos
boolean [] resultados = {true,false,true,false};
//creación de un array de 7 elementos de tipo String
String [] dias = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"};
```

3. ACCEDER A LOS ELEMENTOS DE UN ARRAY

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.

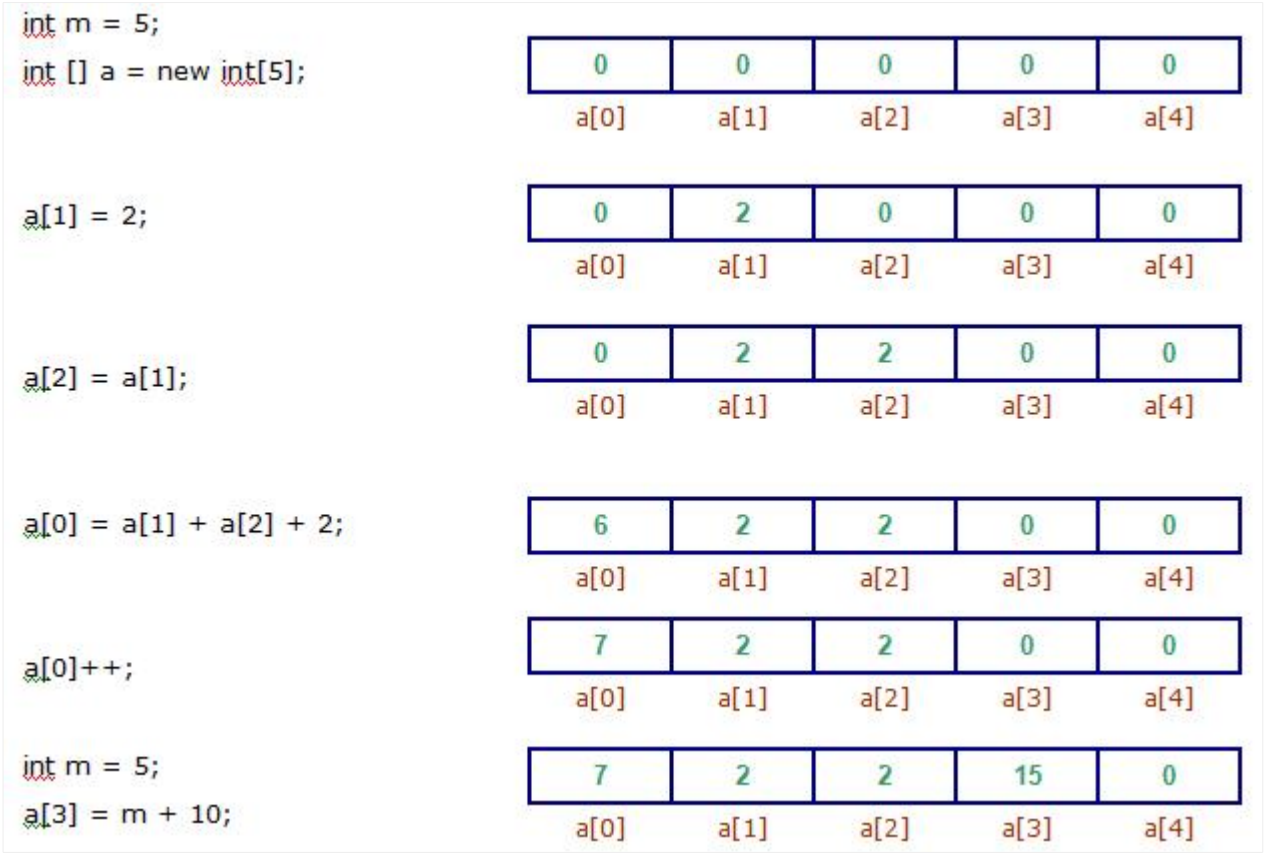
El índice se escribe entre corchetes.

Se puede utilizar como índice un valor entero, una variable de tipo entero o una expresión de tipo entero.

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**.

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).

Por ejemplo:



Si se intenta acceder a un elemento que está fuera de los límites del array (un elemento con índice negativo o con un índice mayor que el que corresponde al 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.

Podemos utilizar length para comprobar el rango del array y evitar errores de acceso.
Por ejemplo, para asignar un valor a un elemento del array que se lee por teclado:

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

int i, valor;

int [] a = new int[10];

System.out.print("Posición: ");

i = sc.nextInt();
System.out.print("Valor: ");
valor = sc.nextInt();
if (i>=0 && i < a.length) {
   a[i] = valor;
```

```
}
```

4. 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) utilizando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

Por ejemplo, el siguiente fragmento de programa Java declara un array de 7 elementos de tipo double y le asigna valores iniciales. A continuación recorre el array, utilizando la instrucción for, para mostrar por pantalla el contenido del array.

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos

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

    System.out.print(notas[i] + " "); //se muestra cada elemento del array

}
```

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

Por ejemplo:

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos

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

    System.out.print(notas[i] + " "); //se muestra cada elemento del array

}
```

Ejemplo de recorrido de un array en java:

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.

Este programa crea un array de elementos de tipo double que contendrá las notas de los alumnos. El tamaño del array será el número de alumnos de la clase.

Se realizan **3 recorridos** sobre el array, el primero para asignar a cada elemento las notas introducidas por teclado, el segundo para sumarlas y el tercero para mostrar los alumnos con notas superiores a la media.

```
import java.util.*;
public class Recorrido2 {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numAlum, i;
        double suma = 0, media;
        do {
            System.out.print("Número de alumnos de la clase: ");
            numAlum = sc.nextInt();
        } while (numAlum <= 0);
        double[] notas = new double[numAlum]; //se crea el array
        // Entrada de datos. Se asigna a cada elemento del array
        // la nota introducida por teclado
        for (i = 0; i < notas.length; i++) {
            System.out.print("Alumno " + (i + 1) + " Nota final: ");
            notas[i] = sc.nextDouble();
        }
        // Sumar todas las notas
        for (i = 0; i < notas.length; i++) {
            suma = suma + notas[i];
        }
        // Calcular la media
        media = suma / notas.length;
        // Mostrar la media
        System.out.printf("Nota media del curso: %.2f %n", media);
        // Mostrar los valores superiores a la 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]);
            }
        }
    }
}
```

Recorrer un Array en java con foreach. Bucle for para colecciones

A partir de java 5 se incorpora una instrucción for mejorada para recorrer arrays y contenedores en general.

Permite acceder secuencialmente a cada elemento del array.

La sintaxis general es:

```
for(tipo nombreDeVariable : nombreArray){

....

}
```

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

nombreDeVariable: variable a la que en cada iteración se le asigna el valor de cada elemento del array. Esta definida dentro del for por lo que solo es accesible dentro de él.

nombreArray: es el nombre del array que vamos a recorrer.

Mediante este bucle solo podemos acceder a los elementos del array. No podemos hacer modificaciones en su contenido.

Ejemplo: El siguiente programa crea un array temperatura de 10 elementos. Lee por teclado los valores y a continuación los muestra por pantalla.

```
import java.util.*;
public class Recorrerforeach1 {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      double [] temperatura = new double[10];
      int i;
      for(i = 0; i<temperatura.length;i++){
         System.out.print("Elemento " + i + ": ");
         temperatura[i] = sc.nextDouble();
      }

      for(double t: temperatura){
         System.out.print(t + " ");
      }
      System.out.println();
   }
}
```

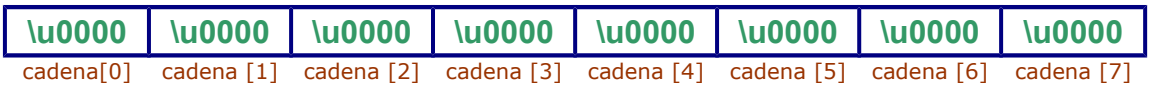
Arrays de caracteres en Java

Un array de caracteres es un array que contiene datos de tipo char.

Los arrays de caracteres en Java se crean de forma similar a un array unidimensional de cualquier otro tipo de datos.

A diferencia de los demás arrays, se puede mostrar el contenido completo de un array de caracteres mediante una sola instrucción print o printf.
Ejemplo: Array de 8 caracteres llamado cadena. Por defecto los elementos del array se inicializan con el carácter nulo (carácter \u0000 Unicode).
char [] cadena = new char[8];

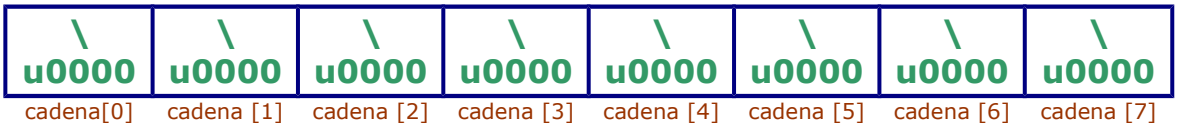
De forma gráfica el array de caracteres cadena se puede representar así:



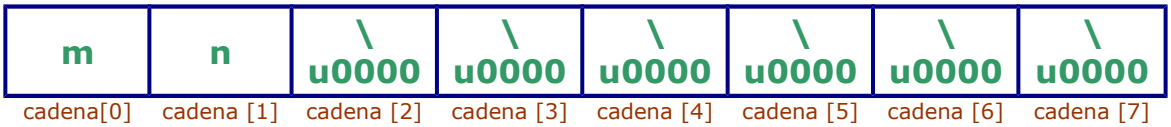
Para mostrar el contenido completo del array:
System.out.println(cadena);
Mostrará 8 caracteres nulos (en blanco)

Ejemplo: Array de 5 caracteres llamado vocales. Se asignan valores iniciales: a, e, i, o, u
char [] vocales = {'a', 'e', 'i', 'o', 'u'};

System.out.println(vocales);
Mostrará:
aeiou
El atributo length de un array de caracteres contiene el tamaño del array independientemente de que sean caracteres nulos u otros caracteres.
Por ejemplo:
char [] cadena = new char[8];



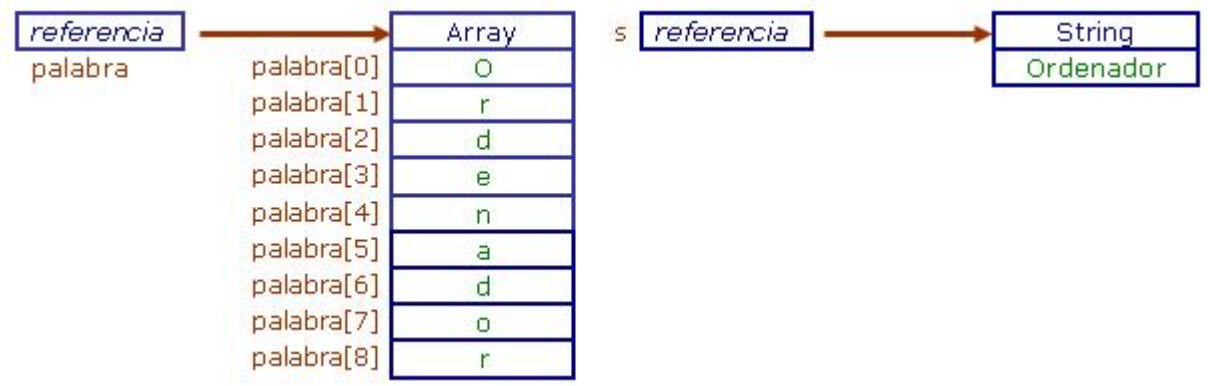
System.out.println(cadena.length);
Muestra: 8
cadena[0] ='m';
cadena[1] ='n';



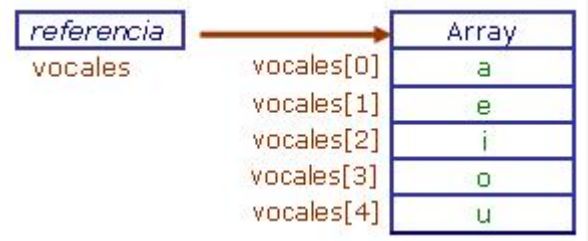
System.out.println(cadena.length);
Muestra: 8
System.out.print(cadena);
System.out.print(cadena);
System.out.println(".");
Mostrará:
mn*bbbbbb*mn*bbbbbb*.
Los espacios en blanco se han representado por el carácter *b*

Se puede asignar un String a un array de caracteres mediante el método toCharArray() de la clase String.
Ejemplo:
String s = "Ordenador";

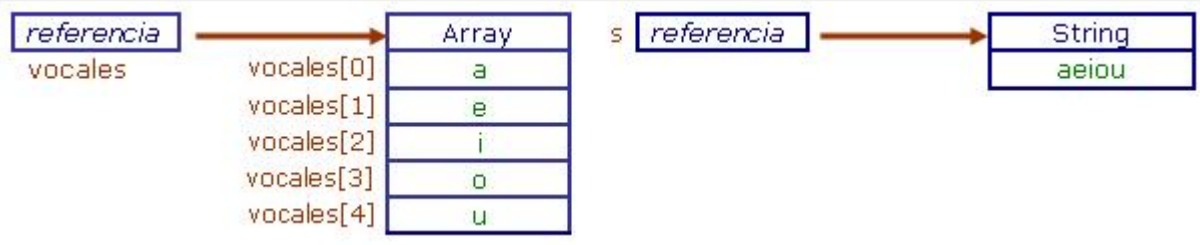
char [] palabra = s.toCharArray();
Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a *palabra*.



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



RECORRER UN ARRAY DE CARACTERES UNIDIMENSIONAL

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

```
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]+ " ");
```

Mostrará 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.
Por ejemplo:

```
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]);
      i++;
}
```

Muestra los caracteres del array hasta que encuentra el primer nulo.

Matrices en Java

Un array en Java 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 vectores que hemos visto en otra entrada anterior son arrays unidimensionales porque solo utilizan un índice para acceder a cada elemento.

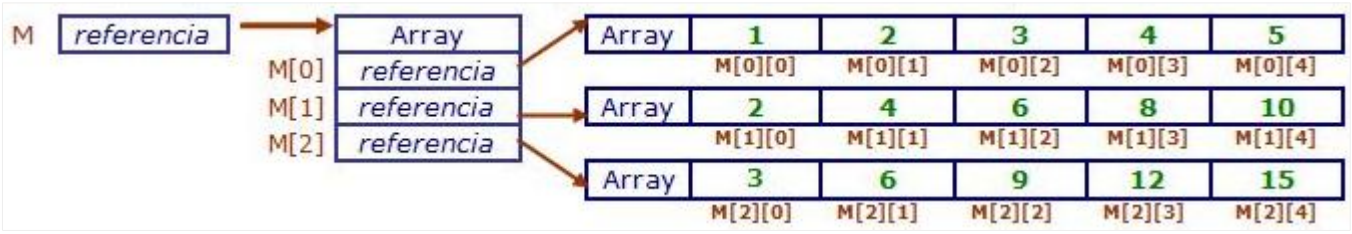
Una matriz necesita dos índices para acceder a sus elementos. Gráficamente podemos representar una matriz como una tabla de n filas y m columnas cuyos elementos son todos del mismo tipo.

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.

CREAR MATRICES EN JAVA

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

Por ejemplo:

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.

Por ejemplo:

int [][] m = new int[3][];

crea una matriz m de 3 filas.

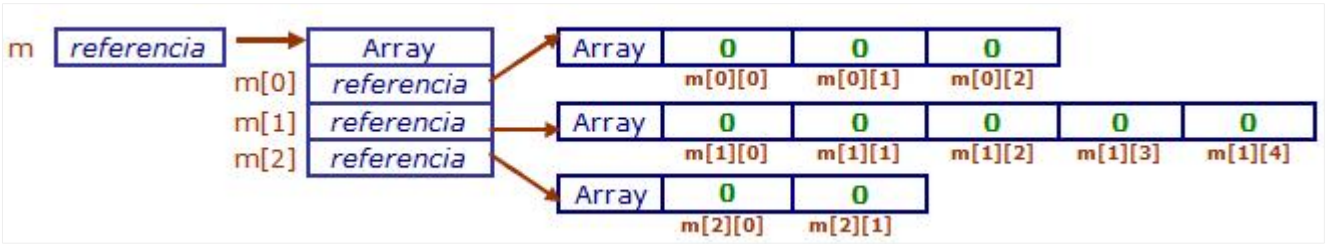
A cada fila se le puede asignar un número distinto de columnas:

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:



INICIALIZAR MATRICES

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial:

- 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 podemos dar otros valores iniciales al array 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.

Por ejemplo:

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 asignan esos valores iniciales.

Asignando valores iniciales se pueden crear también matrices irregulares.

int [][] a = {{6,7,5,0,4}, {3, 8, 4}, {1,0,2,7}, {9,5}};

Crea una matriz irregular de 4 filas. La primera de 5 columnas, la segunda de 3, la tercera de 4 y la cuarta de 2.

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.

Ejemplo de recorrido de una matriz en Java:

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 Bidimensional2 {

  public static void main(String[] args) {
    final int FILAS = 5, COLUMNAS = 4;
    Scanner sc = 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] = sc.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 = menor = A[0][0];//se toma el primero como mayor y menor
    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);
```

```
    }
}
```

En este ejemplo se han utilizado dos formas distintas para recorrer la matriz:

- utilizando en el for el número de filas y columnas
- utilizando en el for el atributo length

Para recorrer arrays irregulares como el siguiente:

```
int [][] a = {{6,7,5,0,4}, {3, 8, 4}, {1,0,2,7}, {9,5}};
```

Usaremos 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();
}
```

Java ArrayList. Estructura dinámica de datos

DECLARACIÓN Y CREACIÓN DE UN ARRAYLIST

De forma general un ArrayList en Java se crea de la siguiente forma:

ArrayList nombreArray = new ArrayList();

Esta instrucción crea el ArrayList *nombreArray* vacío.

Un arrayList declarado así puede contener objetos de cualquier tipo.

Por ejemplo:

ArrayList a = new ArrayList();

a.add("Lenguaje");
a.add(3);
a.add('a');
a.add(23.5);

Los elementos del arrayList a son:
"Lenguaje" 2 'a' 23.5

Es decir, **un ArrayList puede contener objetos de tipos distintos**.

En este ejemplo, el primer objeto que se añade es el String "Lenguaje". El resto no son objetos. Son datos de tipos básicos pero el compilador los convierte automáticamente en objetos de su **clase envolvente** (clase contenedora o wrapper) antes de añadirlos al array.

Un array al que se le pueden asignar elementos de distinto puede tener alguna complicación a la hora de trabajar con él. Por eso, una alternativa a esta declaración es indicar el tipo de objetos que contiene. En este caso, el array solo podrá contener objetos de ese tipo.

De forma general:

ArrayList<tipo> nombreArray = new ArrayList<tipo>();

tipo debe ser una clase. Indica el tipo de objetos que contendrá el array.

No se pueden usar tipos primitivos. Para un tipo primitivo se debe utilizar su clase envolvente.

Por ejemplo:

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

Crea el array *numeros* de enteros.

MÉTODOS DE ARRAYLIST

Algunos métodos que proporciona ArrayList son:

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.
remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

Los puedes consultar todos en:

http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

RECORRER UN ARRAYLIST

Podemos recorrerlo de forma clásica con un **bucle for:**

```
for(int i = 0;i<array.size();i++){
        System.out.println(array.get(i));
}
```

Con un **bucle foreach:**

Si suponemos el array de enteros llamado *numeros:*

```
for(Integer i: numeros){
        System.out.println(i);
}
```

Si el array contiene objetos de tipos distintos o desconocemos el tipo:

```
for(Object o: nombreArray){
         System.out.println(o);
}
```

Utilizando un **objeto Iterator**.

http://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

La ventaja de utilizar un Iterador es que no necesitamos indicar el tipo de objetos que contiene el array.

Iterator tiene como métodos:

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

next: devuelve el siguiente objeto contenido en el array.

Ejemplo:

```
ArrayList<Integer> numeros = new ArrayList<Integer>();
.....
//se insertan elementos
.....

Iterator it = numeros.iterator(); //se crea el iterador it para el array numeros
while(it.hasNext())                    //mientras queden elementos
      System.out.println(it.next());  //se obtienen y se muestran
```

EJEMPLOS DE USO DE ARRAYLIST

Ejemplo 1:

```
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
```

Ejemplo 2: 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.

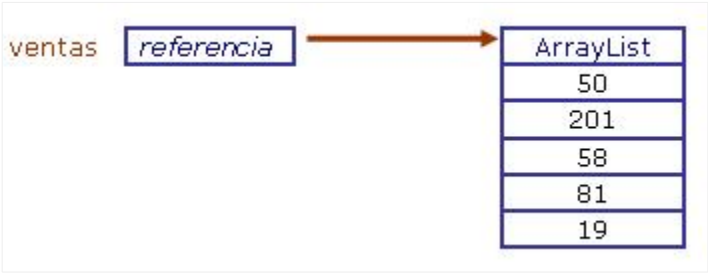
```
import java.util.*;

public class ArrayList2 {

   public static void main(String[] args) {
      Scanner sc = 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 = sc.nextInt();
         if (n != 0)
            numeros.add(n);
      }while (n != 0);

      System.out.println("Ha introducido: " + numeros.size() + " números:");
```

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

        //recorrido usando un iterador para mostrar un elemento por línea
        Iterator it = numeros.iterator();
        while(it.hasNext())
            System.out.println(it.next());

        //recorrido usando foreach para sumar los elementos
        double suma = 0;
        for(Integer i: numeros){
            suma = suma + i;
        }
        System.out.println("Suma: " + suma);
        System.out.println("Media: " + suma/numeros.size());
    }
}
```

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.

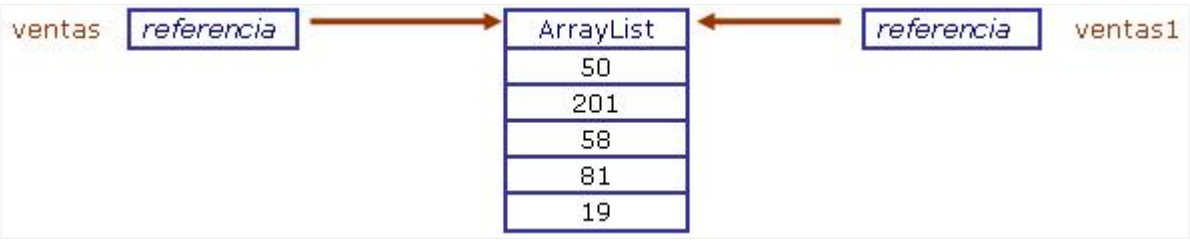
Si disponemos de un ArrayList de enteros llamado ventas:



La instrucción:

ArrayList<Integer> ventas1 = ventas;

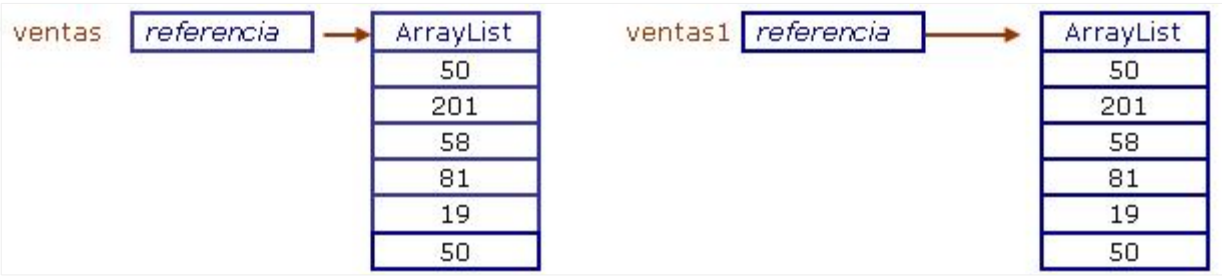
No copia el array ventas en el nuevo array ventas1 sino que **crea un alias**:



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

Para hacer una copia podemos hacerlo 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);



ARRAYLIST COMO PARÁMETRO DE UN MÉTODO

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

Ejemplo: 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) {
      // Crea una lista para el resultado del método
      ArrayList<String> resultado = new ArrayList<String>();
      // Recorre la lista de nombres en orden inverso
      for (int i = nombres.size() - 1; i >= 0; i--) {
          // Añade cada nombre al resultado
          resultado.add(nombres.get(i));
      }
      return resultado;
  }
}
```

ARRAYS BIDIMENSIONALES UTILIZANDO ARRAYLIST

Un ArrayList es un array unidimensional, pero con ellos podemos *simular* arrays de dos o más dimensiones anidando ArrayLists.

Para crear una matriz lo que creamos es un ArrayList cuyos elementos son a su vez ArrayList. Esto se puede extender sucesivamente y obtener arrays de más dimensiones.

Ejemplo:

Programa que lee las notas de 10 alumnos y las guarda en un ArrayList Bidimensional. Cada alumno tiene un número indeterminado de notas. La lectura de notas de cada alumno acaba cuando se introduce un número negativo. Finalmente se muestran todas las notas de todos los alumnos.

```
public static void main(String args[]){

    Scanner sc = new Scanner(System.in);
    final int numAlumnos = 10;  //número de alumnos
    int i, j, nota, cont = 1;

    //crear un ArrayList bidimensional de enteros vacío
    //Realmente se crea un ArrayList de ArrayLists de enteros
    ArrayList<ArrayList<Integer>> array = new ArrayList<ArrayList<Integer>>();
```

```
    //Se leen las notas de cada alumno.
    System.out.println("Introduzca notas. <0 para acabar");
    for(i=0;i<numAlumnos;i++){
       cont = 1;
       System.out.println("Alumno " + (i+1) + ": ");
       System.out.print("Nota " + cont + ": ");
       nota = sc.nextInt();

      //para cada alumno se añade una nueva fila vacía
      //esto es necesario porque el arrayList se crea vacío
       array.add(new ArrayList<Integer>());

       while(nota>=0){
          array.get(i).add(nota); //en la fila i se añade un nueva nota
          cont++;
          System.out.print("Nota " + cont + ": ");
          nota = sc.nextInt();
       }
    }

    //Mostrar todas las notas
    System.out.println("Notas de alumnos");
    for(i=0;i<array.size();i++){  //para cada alumno (para cada fila)
       System.out.print("Alumno " + i + ": ");
       for(j=0;j<array.get(i).size();j++){  //se recorre todas la columnas de la fila
          System.out.print(array.get(i).get(j) + " "); //se obtiene el elemento i,j
       }
       System.out.println();
    }
  }
```

ArrayList de Objetos en Java

En este punto vamos a escribir un programa Java que crea un ArrayList de Objetos de tipo Coche. El programa pide por teclado los datos de los coches y los guarda en el array. A continuación utilizará el ArrayList para mostrar por pantalla lo siguiente:

- Todos los coches introducidos.
- Todos los coches de una marca determinada.
- Todos los coches con menos de un número determinado de Kilómetros.
- El coche con mayor número de Kilómetros.
- Todos los coches ordenados por número de kilómetros de menor a mayor.

Primero creamos la clase Coche:

```
//Clase Coche
public class Coche {
    private String matricula;
    private String marca;
    private String modelo;
    private int Km;

    public int getKm() {
        return Km;
    }

    public void setKm(int Km) {
        this.Km = Km;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getMatricula() {
        return matricula;
    }
```

```
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("\nMatrícula: ");
        sb.append(matricula);
        sb.append("\nMarca: ");
        sb.append(marca);
        sb.append("\nModelo: ");
        sb.append(modelo);
        sb.append("\nKm: ");
        sb.append(Km);
        return sb.toString();
    }
}
```

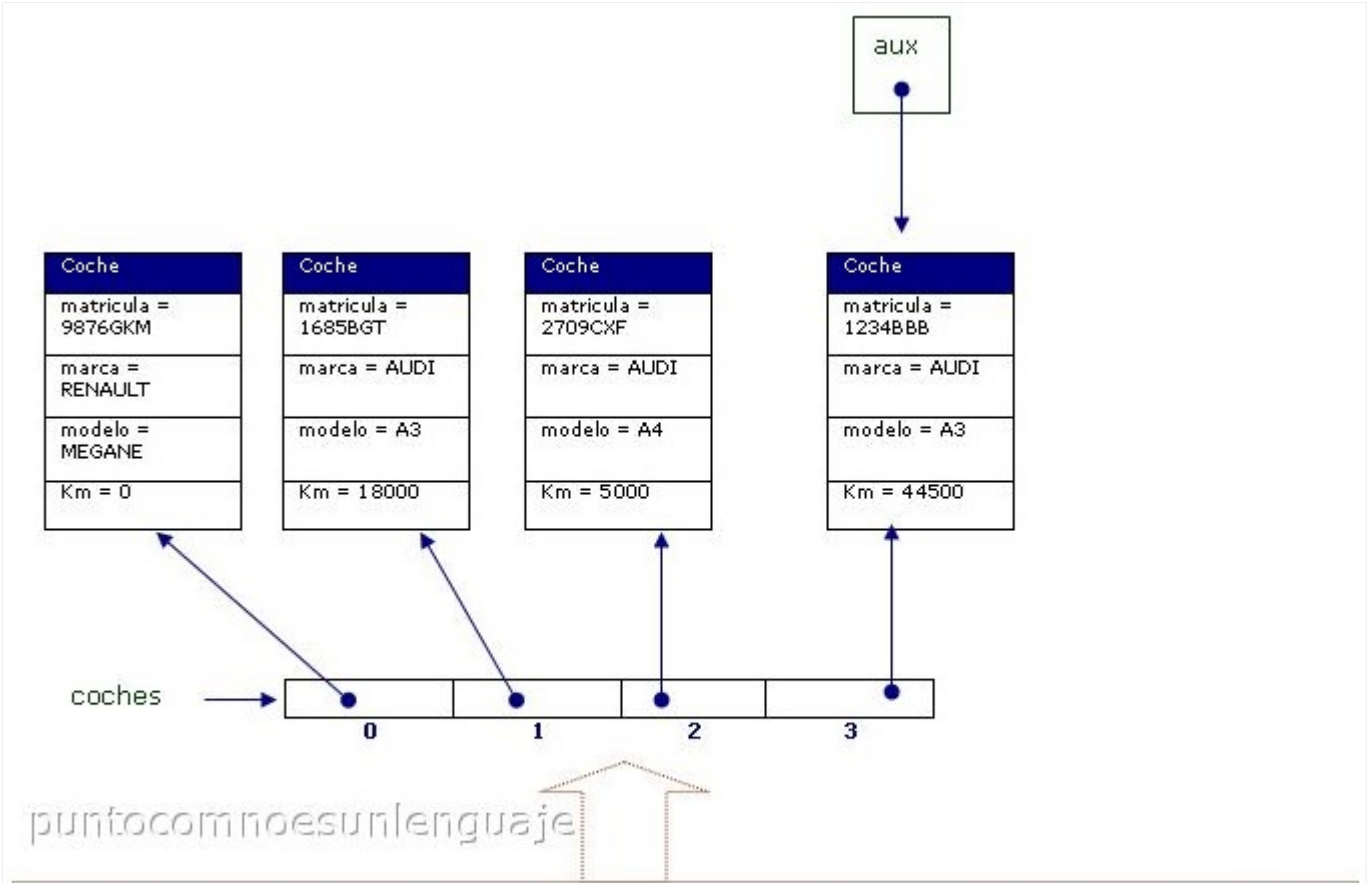
A continuación creamos la clase principal del proyecto:

```
//clase principal
public class Basico1 {

    //Se crea un ArrayList para guardar objetos de tipo Coche.
    static ArrayList<Coche> coches = new ArrayList<Coche>();

    static Scanner sc = new Scanner(System.in);
```

```
  //método main
  public static void main(String[] args) {
      leerCoches();
      System.out.println("\nCoches introducidos:");
      mostrarCoches();
      mostrarPorMarca();
      mostrarPorKm();
      System.out.println("\nCoche con mayor número de Km: " + mostrarMayorKm());
      System.out.println("\nCoches ordenados de menor a mayor número de Km");
      mostrarOrdenadosPorKm();
  } //fin método main

    //Método para leer coches e introducirlos en el array
    public static void leerCoches(){
      //Declaración de variables para leer los datos de los coches
      String matricula;
      String marca;
      String modelo;
      int Km;
      //Variable auxiliar que contendrá la referencia a cada coche nuevo.
      Coche aux;
      int i, N;
      //se pide por teclado el número de coches a leer
      do {
         System.out.print("Número de coches? ");
         N = sc.nextInt();
      } while (N < 0);
      sc.nextLine(); //limpiar el intro
      //lectura de N coches
      for (i = 1; i <= N; i++) {
         //leer datos de cada coche
         System.out.println("Coche " + i);
         System.out.print("Matrícula: ");
         matricula = sc.nextLine();
         System.out.print("Marca: ");
         marca = sc.nextLine();
         System.out.print("Modelo: ");
         modelo = sc.nextLine();
```

```
        System.out.print("Número de Kilómetros: ");
        Km = sc.nextInt();
        sc.nextLine(); //limpiar el intro
        aux = new Coche(); //Se crea un objeto Coche y se asigna su referencia a aux
        //se asignan valores a los atributos del nuevo objeto
        aux.setMatricula(matricula);
        aux.setMarca(marca);
        aux.setModelo(modelo);
        aux.setKm(Km);

       //se añade el objeto al final del array
        coches.add(aux);
    }
  } //fin método leerCoches()
```

Podemos representar de forma gráfica el contenido del ArrayList según se van introduciendo los objetos:



Aspecto del ArrayList coches. Suponiendo que ya se habían insertado antes 3 objetos, añadir uno nuevo al final supone asignar la referencia del nuevo objeto (aux) al último elemento.

```
  //Método para mostrar todos los coches
  public static void mostrarCoches(){
     for(int i = 0; i< coches.size(); i++)
        System.out.println(coches.get(i));  //se invoca el método toString de la clase Coche
  }

  //Método para mostrar todos los coches de una marca que se pide por teclado
  public static void mostrarPorMarca(){
     String marca;
     System.out.print("Introduce marca: ");
     marca = sc.nextLine();
     System.out.println("Coches de la marca " + marca);
     for(int i = 0; i<coches.size(); i++){
        if(coches.get(i).getMarca().equalsIgnoreCase(marca))
           System.out.println(coches.get(i));
     }
  }

  //Método para mostrar todos los coches con un número de Km inferior
  //al número de Km que se pide por teclado
  public static void mostrarPorKm(){
     int Km;
     System.out.print("Introduce número de kilómetros: ");
     Km = sc.nextInt();
     System.out.println("Coches con menos de " + Km + " Km");
     for(int i = 0; i<coches.size(); i++){
        if(coches.get(i).getKm() < Km)
           System.out.println(coches.get(i));
     }
  }

  //Método que devuelve el Coche con mayor número de Km
  public static Coche mostrarMayorKm(){
     Coche mayor = coches.get(0);
     for(int i = 0; i < coches.size(); i++){
```

```
        if(coches.get(i).getKm() > mayor.getKm())
            mayor = coches.get(i);
      }
      return mayor;
   }


   //Método que muestra los coches ordenados por número de Km de menor a mayor
   public static void mostrarOrdenadosPorKm(){
      int i, j;
      Coche aux;
      for(i = 0; i< coches.size()-1; i++)
         for(j=0;j<coches.size()-i-1; j++)
            if(coches.get(j+1).getKm() < coches.get(j).getKm()){
               aux = coches.get(j+1);
               coches.set(j+1, coches.get(j));
               coches.set(j, aux);
            }
      mostrarCoches();
   }
} //fin de la clase principal
```