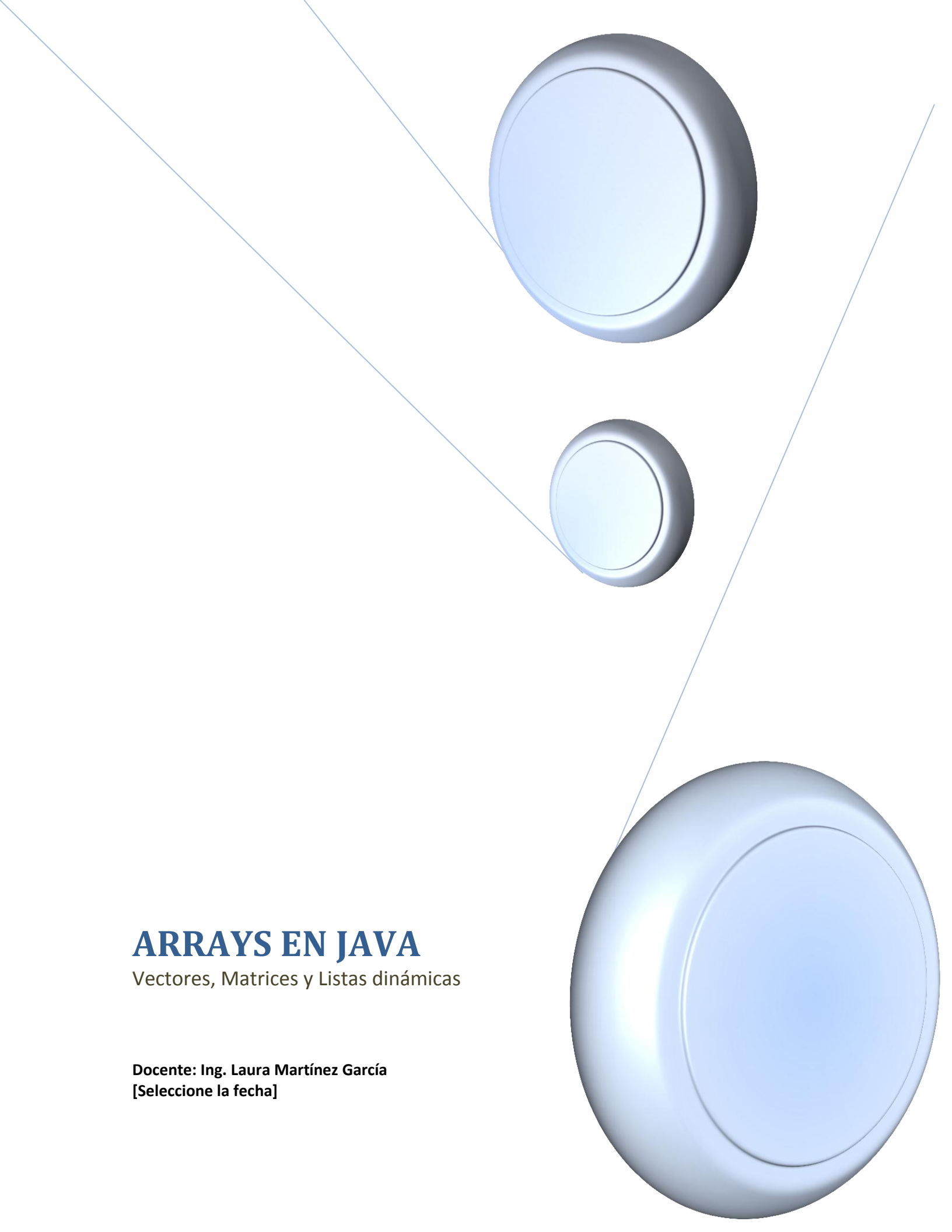


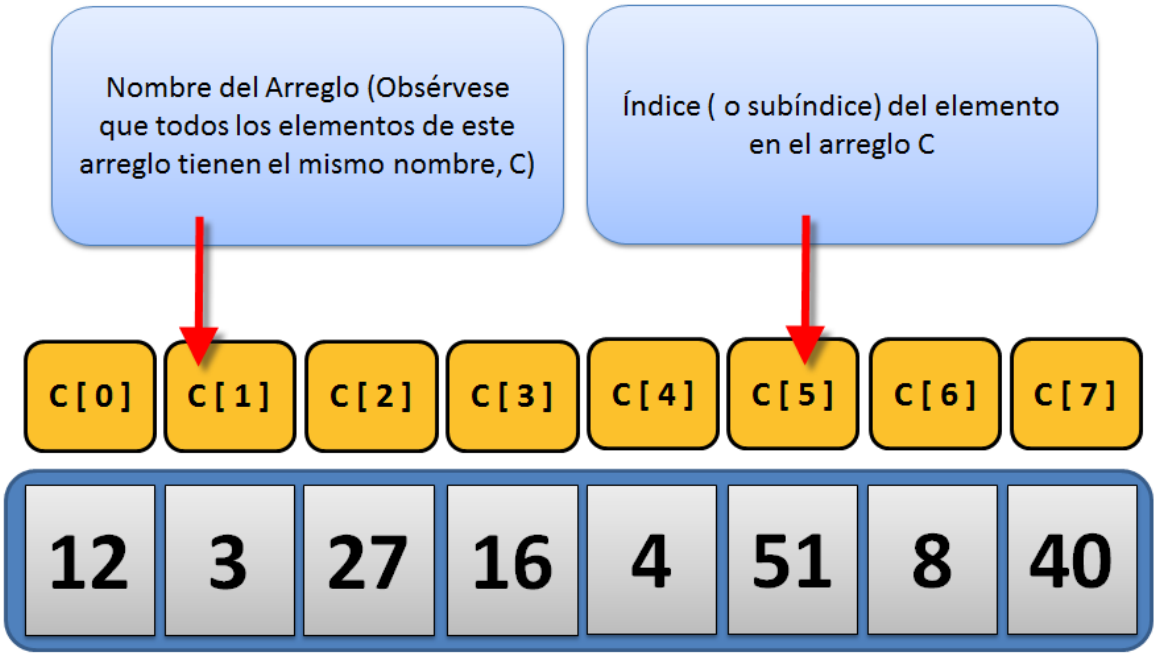
ARRAYS EN JAVA

Vectores, Matrices y Listas dinámicas

Docente: Ing. Laura Martínez García

[Seleccione la fecha]

Un array es una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar en tiempo de ejecución como puede producirse en otros lenguajes.



La declaración de un array en Java y su inicialización se realiza de la siguiente manera:

tipo_dato nombre_array[];
nombre_array = new tipo_dato[tamaño];

Por ejemplo, podríamos declarar un array de caracteres e inicializarlo de la siguiente manera:

char arrayCaracteres[];
arrayCaracteres = new char[10];

Los arrays se numeran desde el elemento cero, que sería el primer elemento, hasta el tamaño-1 que sería el último elemento. Es decir, si tenemos un array de diez elementos, el primer elemento sería el cero y el último elemento sería el nueve.

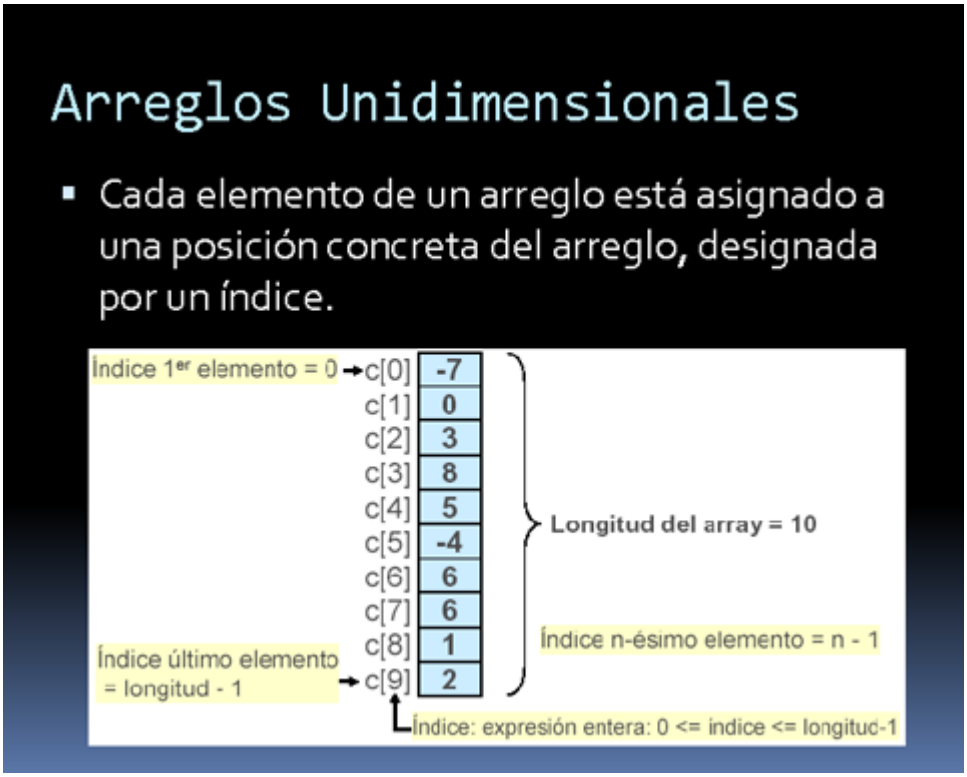
Para acceder a un elemento especifico utilizaremos los corchetes de la siguiente forma. Entendemos por acceso, tanto el intentar leer el elemento, como asignarle un valor.:

arrayCaracteres[numero_elemento];

Por ejemplo, para acceder al tercer elemento lo haríamos de la siguiente forma:
arrayCaracteres[2];
char x = arrayCaracteres[2]; // Lectura de su valor.
arrayCaracteres[2] = 'b'; // Asignación de un valor.

Como se puede comprobar se pone el número dos, que coincide con el tercer elemento, ya que como dijimos anteriormente el primer elemento es el cero.

.length
Esta variable nos devuelve el número de elementos que posee el array. Hay que tener en cuenta que es una variable de solo lectura, es por ello que no podremos realizar una asignación a dicha variable.



Por ejemplo esto nos serviría a la hora de mostrar el contenido de los elementos de un array:

```
char array[];
array = new char[10];
for (int x=0;x<array.lenght;x++) {
    System.out.println (array[x]);
}
```

Matrices o Arrays de varios subindices

Podremos declarar arrays de varios subíndices, pudiendo tener arrays de dos niveles, que serían similares a las matrices, arrays de tres niveles, que serían como cubos y así sucesivamente, si bien a partir del tercer nivel se pierde la perspectiva geométrica.

Para declarar e inicializar un array de varios subíndices lo haremos de la siguiente manera:

tipo_dato nombre_array[][];
nombre_array = new tipo_dato[tamaño][tamaño];

De esta forma podemos declarar una matriz de 2x2 de la siguiente forma:

```
int matriz[][];

matriz = new int[2][2];
```

El acceso se realiza de la misma forma que antes:

```
int x = matriz[1][1]; // Para leer el contenido de un elemento

matriz[1][1] = x;     // Para asignar un valor.
```

Hay que tener en cuenta que para mostrar su contenido tendremos que utilizar dos bucles. Para saber el número de columnas lo haremos igual que antes mediante la variable length, pero para saber el número de filas que contiene cada columna lo tendremos que realizar de la siguiente manera:

```
matriz[numero_elemento].lenght;
```

Nuestra lectura de los elementos de una matriz quedaría de la siguiente forma:

```
int matriz[][];
matriz = new int[4][4];
for (int x=0; x < matrix.length; x++) {
   for (int y=0; y < matriz[x].length; y++) {
      System.out.println (matriz[x][y]);
   }
}
```

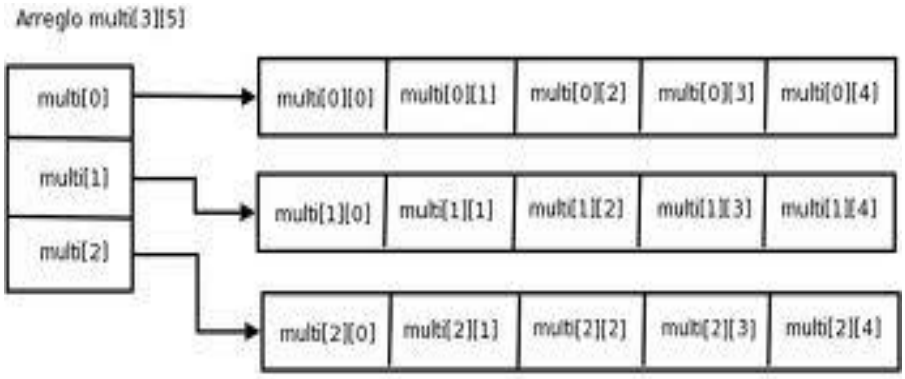


Diagrama Nº 6

Incialización de Arrays

Existe una forma de inicializar un array con el contenido, amoldándose su tamaño al número de elementos a los que le inicialicemos. Para inicializar un array utilizaremos las llaves de la siguiente forma:

tipo_dato array[] = {elemento1,elemento2,...,elementoN};

Así, por ejemplo, podríamos inicializar un array o una matriz:

char array[] = {'a','b','c','d','e'}; // Tenemos un array de 5 elementos.

int array[][] = { {1,2,3,4}, {5,6,7,8}}; // Tenemos un array de 4x4 elementos.

Ejemplo: A continuación se desarrolla una clase que implementa tres métodos el primero genera "n" números enteros, el segundo los imprime y el último obtiene el promedio de estos "n" números.

```
class Arreglo01 {
 private int valor[];
 private void inicia(int n) {
   valor = new int[n];
   for (int i = 0; i < n; i++ )
      valor[i] = (int)(Math.random()*1000.0);
 } //inicia

 private double promedio() {
   int n = valor.length;
   int suma = 0;
   for(int i = 0; i < n; i++)
     suma += valor[i];
   return suma/n;
 }

 public void imprime(int n ) {
   inicia(n);
   for (int i = 0; i < n; i++ )
     System.out.println(valor[i] + " ");
   System.out.println("El promedio de " + n + " números es " + promedio());
   System.out.println();
 } //imprime
}
```

```
class Prueba_Arreglo01 {
  public static void main(String[] arg) {
    Arreglo01 arreglo = new Arreglo01();
    arreglo.imprime(10);
    arreglo.imprime(20);
  }
}
```

Ejemplo: Se definen dos arreglos "nombre" y "calificación"; la primera con elementos String y la segunda con elementos int. Se introducen los nombres y las calificaciones obtenidas, se imprime la lista y, posteriormente se imprime el nombre y calificación de los alumnos que alcanzaron 90 o más.

```
class Alumnos{
   private String[] nombre = {"CARLOS MANUEL", "CESAR", "VLADIMIR","ULISES",
                              "ARMANDO", "ROXANA PATRICIA", "ROSA MARIA",
                              "CARLOS EDUARDO", "LUIS ALBERTO", "JOSE ANTONIO",
                              "DORA EVELIA", "BALDERAS HUGO", "JOSE CARLOS",
                              "JUAN JOSE", "ANGEL GUSTAVO", "JOSE ISMAEL",
                              "ALEXANDRA", "SUSANA", "ANA KARINA", "DIANA LAURA",
                              "EDSON", "EMILIO MARCOS"};

    private int[] calificacion = {68,91,73,80,64,53,82,70,91,65,59,88,57,66,72,
                                  53,50,56,70,66,70,94};
    public void listaCompleta(){
       int n = nombre.length;
       for(int i = 0; i< n; i++)
          System.out.println(nombre[i] + "    " + calificacion[i]);
    }//listaCompleto

    //Imprime altos promedios
    public void altosPromedios(){
       int n = nombre.length;
       System.out.println("imprime nombre y calificacion alumnos de alto promedio");
       for(int i = 0; i< n; i++)
         if(calificacion[i] >= 90)
           System.out.println(nombre[i] + "    " + calificacion[i]);
    }//altosPromedios
}
```

```
public class Prueba_Alumnos {

    public static void main(String[] args) {
      Alumnos misAlumnos = new Alumnos();
      misAlumnos.listaCompleta();
      misAlumnos.altosPromedios();
    } //main

}
```

ARRAYLIST

Las aplicaciones frecuentemente necesitan almacenar un grupo de datos en un sólo objeto. Los arrays sirven bien para este propósito, pero algunas veces necesitamos incrementar o reducir dinámicamente el número de elementos del array, o hacer que contenga distintos tipos de datos. Esto es común entre las aplicaciones como las tiendas online. Un cliente añade una mercancía a su carro de la compra, y detrás de la escena, los ítems son almacenados y eliminados automáticamente. Para esta clase de grupos de datos crecientes, podemos usar la clase Vector o la reciente clase ArrayList del paquete java.util .

ArrayList es una de las muchas clases del Collection Framework, que proporciona un conjunto de interfaces y clases bien-diseñados para almacenar y manipular grupos de datos como una sola unidad, una colección.

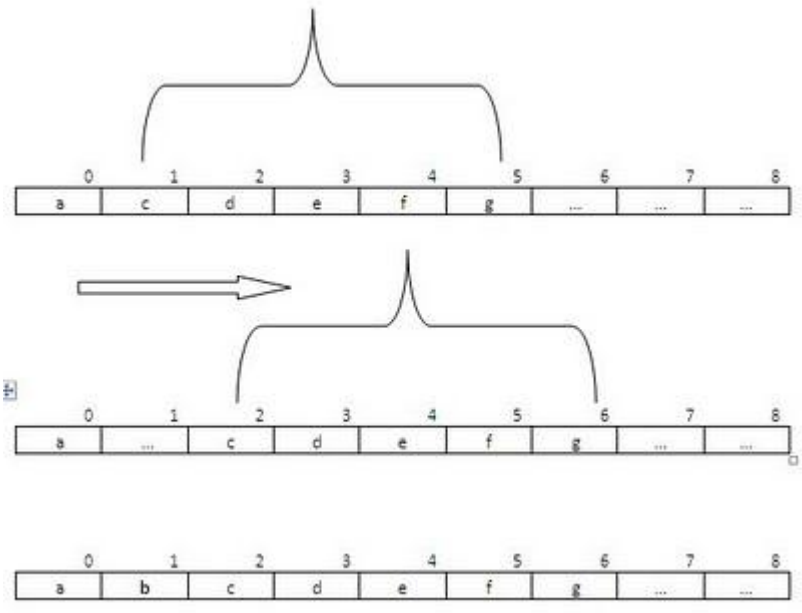
La clase ArrayList permite el almacenamiento de datos en memoria de forma similar a los arrays convencionales, pero con la gran ventaja de que el número de elementos que puede almacenar es dinámico. La cantidad de elementos que puede almacenar un array convencional está limitada por el número que se indica en el momento de crearlo o inicializarlo. Los ArrayList, en cambio, pueden almacenar un número variable de elementos sin estar limitados por un número prefijado. Un ArrayList contiene tantos objetos como necesitemos y tiene varios constructores, dependiendo de cómo necesitemos construir el ArrayList . Los siguientes dos constructores nos ayudarán a empezar:

ArrayList() construye un ArrayList con capacidad cero por defecto, pero crecerá según le vayamos añadiendo:

```
ArrayList al = new ArrayList();
```

ArrayList(int initialCapacity) construye un ArrayList vacío con una capacidad inicial especificada:

```
ArrayList al2 = new ArrayList(5);
```



Declaración de un objeto ArrayList

ArrayList<*nombreClase***>** *nombreDeLista***;**

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

```
1.ArrayList<String> listaPaises;
2.ArrayList<Integer> edades;
3.ArrayList<Persona> p;
```

Creación de un ArrayList

Para **crear un ArrayList** se puede seguir el siguiente **formato**:

nombreDeLista = **new ArrayList();**

Como suele ser habitual, se puede declarar la lista a la vez que se crea:

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

Añadir elementos al final de la lista

El **método** add de la clase ArrayList 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.
Ejemplos:

```
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
```

Insertar elementos en una determinada posición

Con los arrayList 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:

void **add(**int *posición***,** Object *elementoAInsertar***);**

Ejemplo:

```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal");
//El orden hasta ahora es: España, Francia, Portugal
listaPaises.add(1, "Italia");
//El orden ahora es: España, Italia, Francia, Portugal
```

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

Suprimir elementos de la lista

Si se quiere que un determinado elemento se elimine 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.

Object **remove(**int *posición***)**
boolean **remove(**Object *elementoASuprimir***)**

Se puede ver en el siguiente ejemplo los dos posibles usos:

```
ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal");
//El orden hasta ahora es: España, Francia, Portugal
listaPaises.add(1, "Italia");
//El orden ahora es: España, Italia, Francia, Portugal
listaPaises.remove(2);
//Eliminada Francia, queda: España, Italia, Portugal
listaPaises.remove("Portugal");
//Eliminada Portugal, queda: España, Italia
```

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:

Object **get(**int *posición***)**

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). Por ejemplo:

```
System.out.println(listaPaises.get(3));
//Siguiendo el ejemplo anterior, mostraría: Portugal
```

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.

Object **set(**int *posición,* Object *nuevoElemento***)**

Por ejemplo, si en el ejemplo de la lista de países se desea modificar el que ocupe la posición 1 (segundo en la lista) por "Alemania":

```
listaPaises.set(1, "Alemania");
```

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.

int **indexOf(**Object *elementoBuscado***)**

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.

Ejemplo que comprueba si Francia está en la lista, y muestra su posición.

```
String paisBuscado = "Francia";
int pos = listaPaises.indexOf(paisBuscado);
if(pos!=-1)
System.out.println(paisBuscado + " se ha encontrado en la posición: "+pos);
else
System.out.println(paisBuscado + " no se ha encontrado");
```

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

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 empleada 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.

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

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

```
for(String pais:listaPaises)
System.out.println(pais);
```

(Como implementación de la clase Collection, los ArrayList puede recorrerse utilizando iterator)

Otros métodos de interés

- 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 encuenta el elemento indicado en la lista, y false en caso contrario.
- boolean **isEmpty**(): Retorna true si la lista está vacía.
- Object[] **toArray**(): Convierte la lista a un array
- Arrays.sort(v) ordena los elementos del vector.
- Arrays.equals(v1,v2) comprueba si dos vectores son iguales.
- Arrays.fill(v,val) rellena el vector v con el valor val.
- Arrays.binarySearch(v, k) busca el valor k dentro del vector v (que previamente ha de estar ordenado.