Capítulo 4

Almacenamiento de la información en estructuras de datos estáticas (Arrays)

4.1. Resultados de aprendizaje

6. Escribe programas que manipulen información, seleccionando y utilizando tipos avanzados de datos.

a) Se han escrito programas que utilicen arrays.

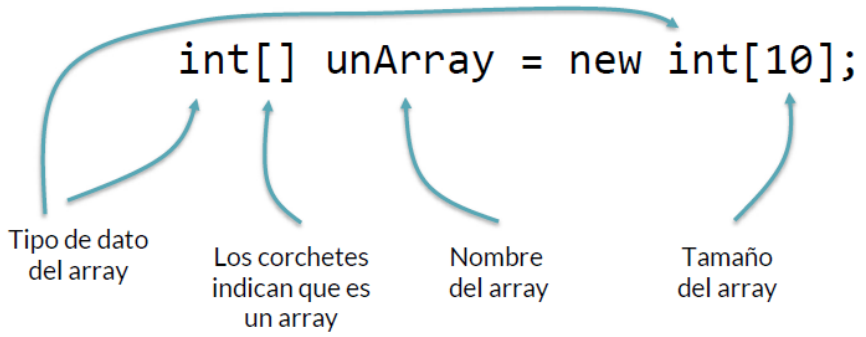
b) Se han reconocido las librerías de clases relacionadas con tipos de datos avanzados.

4.2. Arrays

En programación, una matriz o vector (denominados en inglés arrays) es una zona de almacenamiento continuo que contiene un conjunto de elementos del mismo tipo.

Los arrays se usan para guardar múltiples datos del mismo tipo (días de la semana, meses, ciudades, países, etc...)

Se accede a cada elemento del array mediante un número entero denominado índice. 0 es el índice del primer elemento y n-1 es el índice del último elemento, siendo n el número de elementos del array.



Para declarar un array se escribe:

```
1   tipo_de_dato[] nombre_del_array;
2   int[]listaNumeros;
```

Para crear un array se escribe:

```
1   nombre_del_array = new tipo_de_dato[dimensión];
2   listaNumeros = new int[5];
```

La declaración y la creación del array se puede hacer en una misma línea.

```
1   tipo-de-dato[] nombre_del_array = new tipo_de_dato[dimensión];
2   int[] listaNumeros = new int[5];
```

Para utilizar los elementos del array debemos indicar la posición que ocupan (de cero a n-1):

```
1  nombre_del_array[posicion] = valor;
2  listaNumeros[3] = 6;
```

Ejemplo:

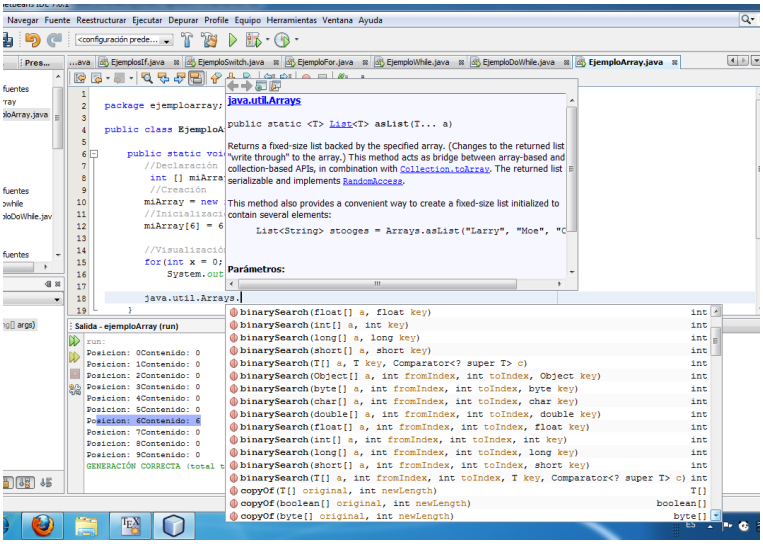
```
1  package ejemploarray;
2
3  public class EjemploArray {
4
5      public static void main(String[] args) {
6         //Declaración
7          int [] miArray;
8          //Creación
9         miArray = new int[10];
10        //Inicialización de un elemento
11        miArray[6] = 6;
12
13        //Visualización del contenido de todos los elementos
14        for(int x = 0; x < 10; x++)
15            System.out.println("Posicion: " + x + "Contenido: " +
                  miArray[x]);
16     }
17 }
```

Existe una manera abreviada de declarar, crear y llenar un array:

```
1   int[] numeros={2, -4, 15, -25};
2
3   String[] nombres={"Juan", "José", "Miguel", "Antonio"};
```

Además de las operaciones que nosotros queramos codificar sobre un array, hay que tener en cuenta que Java nos proporciona muchos métodos dentro de la **clase Arrays**.

https://docs.oracle.com/javase/8/docs/api/index.html



https://codeloper.wordpress.com/2011/10/22/j2se-clase-arrays-metodos-fill-equals-so

4.2.1. Repetitivas para operar con los elementos de un array

Cualquier tipo de repetitiva sirve para realizar operaciones sobre los elementos de un array aunque normalmente se usan las repetitivas de tipo for.

Existe una **versión mejorada del bucle for** que nos permite iterar a través de un array sin la necesidad de un contador, con lo que se evita la posibilidad de salir mas allá de los limites del arreglo. Su sintaxis es:

```
1 for(parametro : nombreArray)
2 {
3   instrucción
4 }
```

También existe la interfaz **Iterator** que nos permite iterar sobre una colección de elementos utilizando los métodos hasNext() y next().

El método hasNext() retorna true en caso de haber más elementos y false en caso de llegar al final de iterator.

El método next(): retorna el siguiente elemento en la iteración.

4.2.2. Ejemplos

```
1   //CREACIÓN E INICIALIZACIÓN ELEMENTO A ELEMENTO
2     int[] unArray = new int[10];
3
4          unArray[0] = 100;
5          unArray[1] = 200;
6          unArray[2] = 300;
7          unArray[3] = 400;
8          unArray[4] = 500;
9          unArray[5] = 600;
10         unArray[6] = 700;
11         unArray[7] = 800;
12         unArray[8] = 900;
13         unArray[9] = 1000;
14
15         //RECORRIDO DE TODOS LOS VALORES DE UN ARRAY
16         for(int i = 0; i < unArray.length; i++) {
17           System.out.println(unArray[i]);
18         }
19  }
```

```
1   //CREACIÓN E INICIALIZACIÓN CON EL ATAJO { }
2   //En este caso, el tamaño lo determina la cantidad de valores que
        inicializamos
3     int[] unArray = {
4           100, 200, 300,
5           400, 500, 600,
6           700, 800, 900, 1000
```

```
7         };
8
9           //RECORRIDO DE TODOS LOS VALORES DE UN ARRAY
10      //CON EL BUCLE FOR MEJORADO
11          for(int i : unArray) {
12            System.out.println(i);
13          }
```

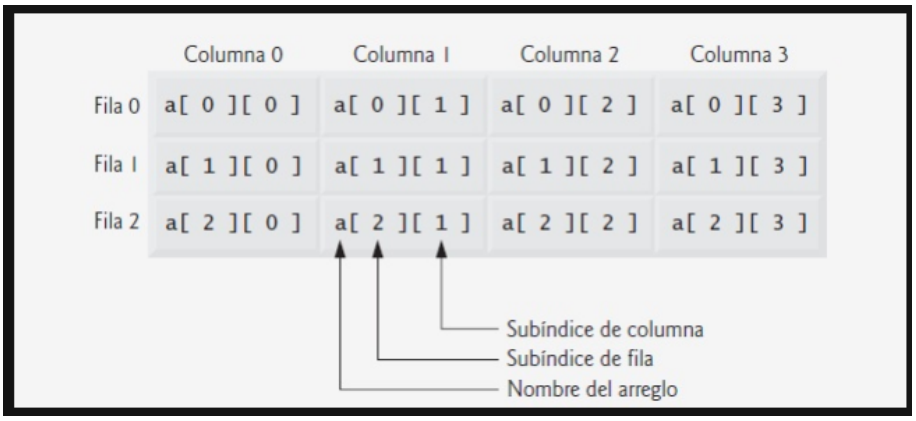
```
1           int[] array = new int[100];
2
3       for(int i = 0; i < array.length; i++) {
4         array[i] = aleatorio(100);
5       }
6
7       printArray(array);
8
9
10      //Podemos crear otro array, copia del primero, ampliando su longitud
11      int[] otroArray = Arrays.copyOf(array, 200);
12
13      //Podemos ordenar los elementos de un array
14      Arrays.sort(otroArray);
15      System.out.println("");
16      System.out.println("Array ordenado");
17      printArray(otroArray);
18
19      //Posición de un número aleatorio, si es que está
20      int num = aleatorio(100);
21      int pos = Arrays.binarySearch(array, num);
22      System.out.println("");
23      if (pos >= 0)
24        System.out.printf("El elemento " + num + " está contenido en el
              array, en la posición "+ pos);
25      else
26        System.out.println("El elemento " + num + " no está en el array");
27
28
29      //El método parallelsort realiza una ordenación más rápida para
            arrays muy largos
30      int[] arrayGrande = new int[123456];
31
32      //Este método sirve para inicializar un array con valores
33      //y usa una expresión lambda, que aprenderemos a crear en otro tema
34      Arrays.parallelSetAll(arrayGrande, i -> aleatorio(12345));
35      Arrays.parallelSort(arrayGrande);
36    }
37
38    /*
39     * ESTE MÉTODO DEVUELVE UN NÚMERO ALEATORIO ENTRE
40     * 0 y TOPE-1.
41     */
42    public static int aleatorio(int tope) {
43      Random r = new Random();
44      return r.nextInt(tope-1);
45    }
46
47    /*
48     * Este método imprime un array, indicando para
49     * cada elemento su valor y su posición
```

```
50     */
51   public static void printArray(int[] array) {
52     for(int i = 0; i < array.length; i++) {
53       System.out.print(array[i] + "[" + i +"]  ");
54       if (i > 0 && i % 10 == 0)
55         System.out.println("");
56     }
57     System.out.println("");
58   }
```

```
1   String[] palabras = new String[5]; // Declaración y creación
2
3        // Llenado
4        palabras[0] = "Hola";
5        palabras[1] = "Casa";
6        palabras[2] = "Coche";
7        palabras[3] = "Mar";
8        palabras[4] = "Libro";
9
10       // Visualización del contenido
11       // Por posición
12       for(int x = 0; x < palabras.length; x++)
13           System.out.print(palabras[x] + " ");
14       System.out.println();
15
16       // Cada palabra contenida en palabras
17       for(String palabra: palabras)
18          System.out.print(palabra + " ");
19       System.out.println();
20
21       // iterator
22       Iterator i = Arrays.asList(palabras).iterator();
23       while(i.hasNext()){
24           System.out.print(i.next() + " ");
25       }
```

4.3. Arrays multidimensionales

En Java es posible crear arrays con más de una dimensión, pasando de la idea de lista, vector o matriz de una sola fila a la idea de matriz de m x n elementos, estructuras tridimensionales, tetradimensionales, etc...



Un array bidimensional es aquel cuyos componentes son accesibles por medio de una pareja de índices que apunten a la fila y a la columna del componente requerido. Los arrays de este tipo son conocidos también con el nombre de matrices. Conceptualmente, podemos pensar en un array bidimensional como en una lista compuesta de filas y columnas, en donde para referirnos a una de ellas emplearemos un número para indicar la posición de fila y otro número para indicar la posición de la columna del componente deseado.

La sintaxis para crear arrays multidimensionales es:

```
1
2  Tipo_de_variable[ ][ ] [ ] ....  Nombre_del_array = new
       Tipo_de_variable[dimensión1][dimensión2]....[dimensiónN];
```

Ejemplo:

```
1  package arraysmultidimensionales;
2
3  public class ArraysMultidimensionales {
4
5      public static void main(String[] args) {
6         char[][] arrayBidimensional = new char[3][6];
7
8         arrayBidimensional[0][0] = 'n';
9         arrayBidimensional[0][1] = 'i';
10        arrayBidimensional[0][2] = 'e';
11        arrayBidimensional[0][3] = 'v';
```

```
12          arrayBidimensional[0][4] = 'e';
13          arrayBidimensional[0][5] = 's';
14
15          arrayBidimensional[1][0] = 'r';
16          arrayBidimensional[1][1] = 'u';
17          arrayBidimensional[1][2] = 'i';
18          arrayBidimensional[1][3] = 'z';
19
20          for(int fila = 0; fila < 3; fila++)
21          {
22              for(int columna = 0; columna < 6; columna ++)
23                  System.out.print(arrayBidimensional[fila][columna]);
24              System.out.println();
25          }
26      }
27  }
```

Una matriz bidimensional puede tener varias filas, y en cada fila no tiene por qué haber el mismo número de elementos o columnas. Por ejemplo, podemos declarar e inicializar la siguiente matriz bidimensional

```
1       double[][] matriz={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};
```

La primer fila tiene cuatro elementos 1,2,3,4.

La segunda fila tiene dos elementos 5,6.

La tercera fila tiene seis elementos 7,8,9,10,11,12.

La cuarta fila tiene un elemento 13.

```
1   //No tiene porqué ser cuadrado
2       int[][] bidimensional = new int[10][20];
3       final int TOPE = 100;
4
5       for(int i = 0; i < bidimensional.length; i++) {
6         for(int j = 0; j < bidimensional[0].length; j++) {
7           bidimensional[i][j] = aleatorio(TOPE);
8         }
9       }
10
11
12      for(int i = 0; i < bidimensional.length; i++) {
13        for(int j = 0; j < bidimensional[0].length; j++) {
14          System.out.print(bidimensional[i][j]+ "\t");
15        }
16        System.out.println("");
17      }
18
19    }
20
21    /*
22     * ESTE MÉTODO DEVUELVE UN NÚMERO ALEATORIO ENTRE
23     * 0 y TOPE-1.
24     */
25    public static int aleatorio(int tope) {
26      Random r = new Random();
```

```
27      return r.nextInt(tope-1);
28    }
```

```
int[][][] arreglo3 = new int[2][3][5]{
                                        {
                                          { 1,2,3,4,5 },
                                          { 2,1,3,4,5 },
                                          { 1,4,3,2,5 }
                                        },
                                        {
                                          { 5,4,3,2,1 },
                                          { 3,4,2,5,1 },
                                          { 1,5,4,2,3 }
                                        }
                                      };
```

4.4. Funciones

Ahora que ya sabemos trabajar con arrays vamos a retomar el tema de **varargs** en las funciones.

Desde hace algunas versiones, Java incluye la opción de usar varargs para indicar que un método recibirá un número arbitrario de argumentos de un tipo. Estos son útiles cuando no sabemos a priori la cantidad de argumentos que recibiremos. Para usarlos usaremos la sintaxis de tres puntos seguidos (...), justo después del tipo de dato, y separados por un espacio del nombre del argumento.

Dentro del método, un varargs se trata igual que un array.

Un método que reciba varios argumentos de diferentes tipos, y además, un varargs , debe incluir este varargs como el último en orden de recepción. De otra forma, sería imposible identificar el número de argumentos recibidos.

```
1  public int sumar(int... numero)
2  {
3      int resultado = 0;
4      for(int i = 0; i < numero.length; i++)
5      {
6          resultado += numero[i];
7      }
8      return resultado;
9  }
10
11
12 System.out.println(sumar(3,5));
13 System.out.println(sumar(3,5,7));
14 System.out.println(sumar(3,5,6,5));
```