**Ingeniería en Desarrollo de Software**
3er semestre

Programa de la asignatura:
**Programación orientada a objetos I**

# Unidad 4. Arreglos

Clave:
**Ingeniería:** **TSU:**
15142316 / 16142316

**Universidad Abierta y a Distancia de México**

**Índice**

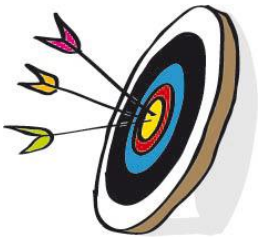
# Unidad 4. Arreglos

## Presentación de la unidad

Durante ésta, la cuarta unidad de la asignatura Programación Orientada a Objetos I, abordaremos el concepto arreglos. Identificarás una de las estructuras de datos más importante y básica, los arreglos. Conocerás su concepto, su declaración, manipulación y recorrido. De esta manera aprenderás a manejar varios datos unidos como un todo. Lo cual te servirá para el manejo de información obtenida de una base de datos, así como la estructuración de datos en tablas, que son temas que tratarás en la asignatura POO II.

En esta unidad es necesario que captures todos los programas de ejemplo, para que analices su sintaxis y así comprendas mejor los temas estudiados.

## Propósito



Al término de esta unidad lograrás:

- Distinguir los arreglos unidimensionales y multidimensionales.
- Manejar las operaciones básicas con arreglos.

## Competencia específica

Construir programas para el manejo de datos del mismo tipo (carácter, cadena, entero o decimal) a través del uso de arreglos aplicados en problemas matemáticos simples.

## 4.1. Arreglos unidimensionales

Como recordarás en unidades anteriores estudiaste variables que de acuerdo al tipo de datos con que se construyen, depende del tipo de información que podrán almacenar. En esta última unidad de POOI, se analizarán los arreglos que ayudan en el manejo de información que debe estar conjunta, es decir, los arreglos almacenan varios datos, como si fuesen muchas variables, pero estos datos están dentro de un solo lugar, como veremos a continuación.

Un **arreglo** es una estructura de datos que contiene una serie de variables que se encuentran en un solo lugar (ese lugar es el arreglo) a las que se accede mediante índices, o posiciones del arreglo.

Las variables contenidas en un arreglo, también llamadas los **elementos del arreglo**, son todas del mismo tipo, y este tipo se llama el **tipo de elemento del arreglo**. Un arreglo tiene un rango que determina el número de índices asociados a cada elemento del arreglo.

El rango de un arreglo también se conoce como las **dimensiones del arreglo**. Un arreglo con un rango de 1 se denomina **arreglo unidimensional**. Un arreglo con un rango mayor que 1 se llama un arreglo **multidimensional**. Los arreglos multidimensionales con un tamaño específico se refieren a menudo como arreglos de dos dimensiones, arreglos de tres dimensiones, y así sucesivamente.

Cada dimensión de un arreglo tiene una longitud asociada que es un número entero mayor o igual a cero. Las longitudes de las dimensiones no son parte del tipo del arreglo, sino que se establecen cuando una instancia del tipo arreglo se crea en tiempo de ejecución. La longitud de una dimensión determina el rango de validez de los índices de esa dimensión, por ejemplo para una dimensión de longitud N, los índices pueden variar de 0 a N-1. El número total de elementos de un arreglo es el producto de las longitudes de cada dimensión del arreglo. Si una o más de las dimensiones de un arreglo tienen una longitud de cero, se dice que el arreglo está vacío. El tipo de elemento de un arreglo

puede ser cualquier tipo de dato, incluyendo él mismo (Hejlsberg, 2011, p. 626); es decir, en lugar de cada posición contenga un dato, puede contener un arreglo completo.

En el siguiente subtema se expondrá cómo se declaran los arreglos, para que comiences a utilizarlos y en el subsecuente verás cómo manipularlos.

### 4.1.1. Declaración

Como se ha mencionado en el tema anterior los arreglos son un conjunto de elementos ordenados por su posición, y el contenido de éstos debe ser del mismo tipo de dato. Ahora bien, veamos cómo se programan los arreglos en JAVA.

En Java hay que declarar un *array* (o arreglo, son lo mismo) antes de poder utilizarlo. En la declaración es necesario incluir el nombre del *array* y el tipo datos que se van a almacenar en él (Froufe, 2009, p. 35).

La sintaxis general para declarar e instanciar un *array* es:

```
tipoDeDato[ ] identificadorDelArray = new tipoDeDato[tamañoDelArray]
```

Además, como ya se mencionó, se pueden usar en Java *arrays* de cualquier tipo de dato soportado, incluso los definidos por el usuario, como se muestra a continuación:

```
char x[ ];
int y[ ];
clsAlumnos objAlumnos[ ];
```

Haciendo referencia al subtema *2.1.1. Declaración y ámbito de variables* presentado durante la Unidad 2, se debe saber que la declaración de los *arrays* en Java se hace mediante una variable referenciada, es decir, no se guarda el valor del contenido del

*array*, lo que se guarda es una referencia de la locación de memoria donde están alojados dichos datos. Con lo que se acaba de ver, fácilmente se puede deducir que los *arrays* son objetos. Esto se puede analizar a continuación, a partir de lo que nos indica Froufe (2009, p. 35):

La memoria necesaria para almacenar los datos que componen el *array* se buscará en memoria dinámica a la hora de instanciar y crear realmente el *array*.

Al igual que con cualquier variable utilizada, en el lenguaje Java necesitan ser inicializadas a un valor. Bajo esta premisa, cuando se declara (crea) un *array* en Java se inicializa automáticamente, dependiendo de estos casos:

1. ***false*** cuando el tipo de dato es *boolean*.
2. ***null*** para un *array* de objetos de tipo *Object*.
3. El equivalente a **0** (vacío para *String*) para cualquier otro tipo de dato.

Cuando se hace la declaración de un *array* en Java se consideran dos posibles opciones: Declararlo vacío o Declarar colocando valores.

Veamos en el siguiente ejemplo:

- El código marcado en azul es un ejemplo de una **declaración de arreglo vacío**.
- Por su parte los códigos morado y rosa están **declarando un arreglo con valores**. Observa que el código morado está colocando los datos directamente, sin indicar el tamaño del arreglo, esto es posible porque las comas (,) indican la separación de los elementos del arreglo e indican su tamaño.
- El código en rosa, primero declara un arreglo vacío, pero en las siguientes instrucciones va llenando el arreglo con base en las posiciones de éste, recuerda que las posiciones inician en cero (0).

Ahora, la demostración del ejemplo:

```
//arreglo declarado en vacío
int números[] = new int[50];
```

```
//arreglo declarado con valores iniciales
String nombres[] = {“Juan”, “Pedro”, ”Luis”, ”Hugo”};
```

```
//al declarar un arreglo con valores, también podría escribirlo así:
String nombres[] = new String[4];
nombres[0] = new String(“Juan”);
nombres[1] = new String(“Pedro”);
nombres[2] = new String(“Luis”);
nombres[3] = new String(“Hugo”);
```

Una vez que se ha instanciado un *array*, se puede acceder a los elementos de éste utilizando un índice, de forma similar a la que se accede en otros lenguajes de programación. Los índices de un *array* siempre empiezan por 0 (cero) (Froufe, 2009, p. 35).

### 4.1.2. Recorrido

Los elementos contenidos en los *arrays* están controlados por la posición que ocupen dentro del *array*. Así, en el siguiente ejemplo se tiene la declaración de un *array* de tipo *String* que contiene 4 elementos, refiriéndose a los nombres propios de 4 personas:

```
String nombres[ ] = {“Juan”, “Pedro”, ”Luis”, ”Hugo”};
```

Si, por alguna necesidad especial de la situación, se quisiera imprimir a pantalla el contenido del ***array* nombres** indicando la posición de cada nombre, bastaría con hacer lo siguiente:

```
System.out.println(nombres[0]+" está en la posición 0");
System.out.println(nombres[1]+" está en la posición 1");
System.out.println(nombres[2]+" está en la posición 2");
System.out.println(nombres[3]+" está en la posición 3");
```

Donde se indica al arreglo que se quiere imprimir el valor que está en la posición *i-ésima*, esta instrucción se le indica al lenguaje Java cuando se pone el nombre del arreglo con el cual se quiere trabajar, seguido de la posición que se desea recuperar (para impresión, operaciones, cálculos, entre otros) encerrado entre corchetes(*[ ]*). La salida de estas instrucciones escritas en lenguaje Java, al compilar y ejecutar el programa (se obvian las partes faltantes de la clase por cuestiones didácticas) sería la siguiente:

```
Juan está en la posición 0
Pedro está en la posición 1
Luis está en la posición 2
Hugo está en la posición 3
```

Imagina la cantidad de líneas necesarias para imprimir el nombre de todos los alumnos que pertenecen a una universidad cuya matrícula es un total de 5,000 alumnos. Como se podrá deducir fácilmente se necesitarían 5,000 líneas como las anteriores.

Para solucionar la cuestión anterior, de forma un tanto más sencilla, se utilizan ciclos que como ya se sabe, facilitan la ejecución de operaciones secuenciales y repetitivas pero necesarias, como en el caso de los *arrays*. Además de que los ciclos son controlados por índices (su comienzo y su fin) y ésta es la misma forma de acceder a los elementos de un *array*, como ya se había explicado. A continuación se presenta la misma impresión del contenido del *array* nombres, pero utilizando un ciclo *for*:

```
for(int i=0; i<4; i++){
System.out.println(nombres[i]+" está en la posición "+i);
}
```

Y la salida será:

```
Juan está en la posición 0
Pedro está en la posición 1
Luis está en la posición 2
Hugo está en la posición 3
```

Con base en las instrucciones mostradas con el uso del ciclo para recorrer un arreglo, se introdujeron algunas variaciones para poder hacer el recorrido funcional y se explica a continuación: cuando se declara un arreglo se indica el total de elementos que contendrá dentro de él (límite del arreglo); este mismo límite es el total de repeticiones que deberá ejecutar el ciclo *for*, indicado con i<4 (ya que si se indica un número mayor, Java lanzará una excepción indicando que se intenta acceder a una locación de memoria no reservada o inexistente). De igual manera el inicio del *arra*y debe comenzar por el número 0 (cero) y esto se indica en el ciclo *for* al hacer la declaración *i=0* que indica que desde esa posición comenzará a iterar.

Finalmente se le indica al arreglo que se quiere hacer el recorrido de sus elementos de uno en uno (aunque podría ser diferente, de dos en dos por ejemplo) para poder obtener el contenido total del *array*, en el ciclo *for* se le indica con la instrucción *i*++. Así, a la hora de colocar la línea de impresión `System.out.println(nombres[i]+" está en la posición "+i);` se hace referencia al elemento i-ésimo indicado entre corchetes, pero en lugar de indicarle un número fijo, se le indica la posición que se desea imprimir con la

variable *i* que a voluntad del programador iniciará, terminará y se irá incrementando según lo explicado de las partes que conforman al ciclo *for*.

### 4.1.3. Operaciones

Al referirse a operaciones con *arrays* no es en el sentido clásico de operaciones aritméticas sobre ellos (aunque se pueden hacer y se ejemplificará más adelante), sino a cómo se obtienen sus elementos, haciendo un profundo énfasis en la importancia que tienen los índices para ello.

Imagina que tenemos 2 arreglos unidimensionales, como sigue:

```
int[ ] a = new int[10];
int[ ] b = new int[10];
```

Y se quiere hacer una suma y multiplicación punto a punto (posición a posición) de cada uno de sus elementos, y almacenarlos en un tercer *array* para cada operación sugerida, luego imprimir el resultado de la acumulación de las operaciones. Esto se ejemplifica con la siguiente porción de código:

```
int c[ ] = new int[10]; //array que contendrá los resultados de la suma
int d[ ] = new int[10]; //array que contendrá los resultados de la multiplicación
int acumulacionSuma = 0;
int acumulacionMultiplicación = 0;
for(int i=0; i<10; i++){
c[i]=a[i]+b[i];
d[i]=a[i]*b[i];
acumulacionSuma += c[i];
acumulacionMultiplicacion *= d[i];
}
//imprimimos el resultado de sumar los arrays
for(int i=0; i<10; i++){
System.out.println(c[i]+",");
}
//imprimimos el resultado de multiplicar los arrays
for(int i=0; i<10; i++){
System.out.println(d[i]+",");
}
//imprimimos las acumulaciones
System.out.println("Acumulación de la suma es "+acumulacionSuma);
System.out.println("Acumulación de la multiplicación es "+acumulacionMultiplicacion);
//termina la ejecución
```

Nótese la importancia de los índices para poder acceder a los elementos de los *arrays* y de la misma manera para almacenar resultados.

Se puede utilizar un mismo contador para **n** *arrays*, siempre y cuando éstos sean del mismo tamaño, como se muestra en el ejemplo anterior, donde se sumó y multiplicó en un mismo ciclo. Sólo en las impresiones, dado que se buscaba que el resultado se viera de manera independiente, se utilizaron más **ciclos** *for*; tantos como **números** de **resultados** se quieran mostrar.

## 4.2. Arreglos multidimensionales

Los arreglos vistos en el tema anterior sólo son de una dimensión, es decir sólo se manejaban a manera de renglón (o fila).

En cambio los **arreglos multidimensionales** se manejan a manera de tabla, donde se tendrán filas y columnas, por lo que este tipo de arreglos ofrece un mejor manejo de datos cuando éstos deben manejar dimensiones, como la información mostrada en hojas de cálculo o tablas de bases de datos.

En los subtemas previos has visto cómo declarar y manipular arreglos unidimensionales, a continuación verás cómo hacerlo para los arreglos multidimensionales.

## 4.2.1. Declaración

Los arreglos multidimensionales tienen más de una dimensión. En Java, las **dimensiones** se manejan por medio de **corchetes** (muy parecido a los *arrays* unidimensionales), dentro de los cuales se escribe cada dimensión (cantidad de datos que guardará). Cada dimensión irá dentro de un corchete. La sintaxis general para declarar e instanciar un *array* es:

```
tipoDeDato[ ][ ][ ]... identificadorDelArray =
new tipoDeDato[tamañoDimensión][ tamañoDimensión][ tamañoDimensión]...
```

Recuerda que se pueden **definir** *arrays* de **cualquier tipo** de **dato**, y que todos los elementos que estén dentro del *array* serán del mismo tipo, también pueden crearse *arrays* de tipos de datos definidos por el programador como en el siguiente ejemplo, donde el último *array* declarado es de tipo **clsAlumno**, que el programador debió definir previamente.

```
char x[ ][ ];
int y[ ][ ][ ];
clsAlumnos objAlumnos[ ][ ];
```

Al igual que se **expuso** en los arreglos unidimensionales, para los de múltiples dimensiones, también la declaración de los *arrays* en Java se hace mediante una variable referenciada, es decir, no se guarda el valor del contenido del *array*, lo que se guarda es

una referencia de la locación de memoria donde están alojados dichos datos. Con lo que se acaba de ver, fácilmente se puede deducir que los *arrays* son objetos. Cuando se hace la declaración de un *array* en Java se consideran dos posibles opciones: Declararlo **vacío** o declarar **colocando valores**.

**Ejemplo de declaración de un arreglo vacío**

```
int números[ ][ ] = new int[3][3];
```

En la estructura general, si pudiera observarse la reserva de memoria que hace Java sobre la memoria principal de la computadora, sería como sigue:

```
[Null][Null][Null]
[Null][Null][Null]
[Null][Null][Null]
```

**Declaración colocando valores**

```
int numeros2[ ][ ] = {{1, 2, 3},
                      {4, 5, 6},
                      {7, 8, 9}}
```

Observa que la estructura que se muestra, coincide perfectamente con las dimensiones declaradas en el arreglo *intnumeros[ ][ ] = new int[3][3];* donde el número contenido en el primer corchete corresponde al número de filas que contendrá el *array*, por ende, el número contenido en el segundo corchete corresponde al número de columnas que

contendrá el *array*, de esta manera se analizará una estructura diferente, como la que sigue:

```
int números[ ][ ] = new int[5][3];
```

Quedaría así:

```
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
[ ][ ][ ]
```

Declararlo con valores iniciales es un poco diferente de la declaración para *arrays* unidimensionales, pues se debe especificar explícitamente el contenido de cada dimensión del *array*.

```
String nombres[ ] = {{“Juan”, “Pedro”, ”Luis”, ”Hugo”},
{“Ana”, “Paulina”, “Alan”, “Blanca”}};
Sería lo mismo escribirlo así:
String nombres[ ][ ] = new String[2][4];
nombres[0][0] = new String(“Juan”);
nombres[0][1] = new String(“Pedro”);
nombres[0][2] = new String(“Luis”);
nombres[0][3] = new String(“Hugo”);
nombres[1][0] = new String(“Ana”);
nombres[1][1] = new String(“Paulina”);
nombres[1][2] = new String(“Alan”);
nombres[1][3] = new String(“Blanca”);
```

## 4.2.2. Recorrido

El recorrido de los elementos de un *array* multidimensional es de la misma manera que en un arreglo unidimensional, sólo agregando un corchete por cada dimensión que se añada, considerando que se puede acceder a los valores de los elementos de un arreglo bidimensional a través del nombre del arreglo y dos subíndices. Los subíndices deben escribirse entre corchetes y representan la posición del elemento en el arreglo. Así, es posible referirse a un elemento del arreglo escribiendo el nombre del arreglo y los subíndices del elemento entre corchetes. Los valores de los subíndices empiezan en cero para el primer elemento, hasta el tamaño del arreglo menos uno.

Tómese como ejemplo el siguiente código, donde se recorre e imprime el *array* bidimensional llamado **nombres**:

```
for(int i=0; i<2;i++){ //contador encargado de recorrer las filas
        for(int j=0; j<4; j++){ //contador encargado de recorrer las columnas
                System.out.println(nombres[i][j]+" está en la posición "+i+"-"+j+" del arrray");
}
}
```

La salida esperada para esta porción de código será:

```
Juan está en la posición 0-0 del array
Pedro está en la posición 0-1 del array
Luis está en la posición 0-2 del array
Hugo está en la posición 0-3 del array
Ana está en la posición 1-0 del array
Paulina está en la posición 1-1 del array
Alan está en la posición 1-2 del array
Blanca está en la posición 1-3 del array
```

Observa el aumento de los contadores y la manera en que van incrementándose, ya que ahí es donde radica la importancia del recorrido de los arreglos, en la posición de sus contadores.

### 4.2.3. Operaciones

Como los *arrays* unidimensionales y multidimensionales son tan parecidos en su método de acceso (que de hecho es idéntico), se ejemplifica los *arrays* multidimensionales de la misma manera que se hizo con los *arrays* unidimensionales, sólo modificando sus dimensiones:

Imagina que tenemos 2 arreglos unidimensionales, como sigue:

```
int[ ][ ] a = new int[10][10]; //en lugar de ser 10 números ahora serán 100
int[ ][ ] b = new int[10][10]; //en lugar de ser 10 números ahora serán 100
```

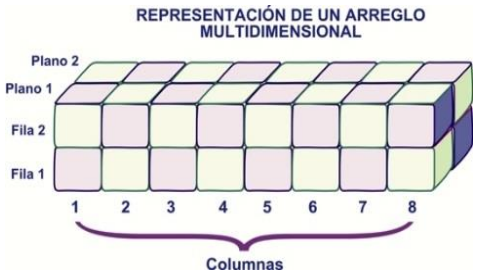
Recuerda que el número total de elementos de un *array* multidimensional es la multiplicación de cada dimensión contra todas las restantes, y se quiere hacer una suma y multiplicación punto a punto (posición a posición) de cada uno de sus elementos y almacenarlos en un tercer *array* para cada operación sugerida, luego imprimir el resultado de la acumulación de las operaciones. Observa un ejemplo con la siguiente porción de código:

```
int c[ ] = new int[10][10]; //array que contendrá los resultados de la suma
int d[ ] = new int[10][10]; //array que contendrá los resultados de la multiplicación
int acumulacionSuma = 0;
int acumulacionMultiplicación = 0;
for(int i=0; i<10; i++){ //acceso a las filas
for(int j=0; j<10, j++){ //acceso a las columnas
c[i][j]=a[i][j]+b[i][j];
d[i][j]=a[i][j]*b[i][j];
acumulacionSuma += c[i][j];
acumulacionMultiplicacion *= d[i][j];
}
}
//imprimimos el resultado de sumar los arrays
for(int i=0; i<10; i++){ //acceso a las filas
for(int j=0; j<10, j++){ //acceso a las columnas
System.out.println(c[i][j]+",");
}
System.out.println();//retorno de carro para darle forma de matriz
}
//imprimimos el resultado de multiplicar los arrays
for(int i=0; i<10; i++){//acceso a las filas
for(int j=0; j<10, j++){ //acceso a las columnas
System.out.println(d[i][j]+",");
}
System.out.println();//retorno de carro para darle forma de matriz
}
//imprimimos las acumulaciones
System.out.println("Acumulación de la suma es "+acumulacionSuma);
System.out.println("Acumulación de la multiplicación es "+acumulacionMultiplicacion);
//termina la ejecución
```

Nota la importancia de los índices para poder acceder a los elementos de los *arrays* y de la misma manera para almacenar resultados.

Se puede utilizar un mismo contador para n *arrays*, siempre y cuando éstos sean del mismo tamaño, como en el ejemplo anterior donde se sumó y multiplicó en un mismo ciclo. Sólo en las impresiones como se buscaba que el resultado se viera de manera independiente, se utilizaron más ciclos *for*; tantos como números de resultados se quiera mostrar.

## Cierre de la unidad




Has concluido la cuarta y última unidad del curso. En esta unidad has estudiado los arreglos unidimensionales y los arreglos multidimensionales.

Dentro del tema de arreglos se ha visto cómo se declaran arreglos vacíos o con datos, y a partir de ello, se determinó que lo más importante para recorrer arreglos es el manejo de las posiciones.

Con estos temas has concluido el estudio de esta asignatura, si tienes dudas respecto a ellos, te sugerimos volver a revisar el material, para que estés listo(a) para continuar con la asignatura POOII, para la cual lo aprendido aquí, te será de gran utilidad.

## Para saber más

Consulta la página oficial del lenguaje Java, donde podrás encontrar manuales de referencia sobre arreglos. Disponible en: http://www.java.com/es/

## Fuentes de consulta


- Froufe, A. (2009). *Java 2. Manual de usuario y tutorial*. México: Alfaomega, Ra-Ma.

- Hejlsberg, A., Torgersen, M., Wiltamuth, S. y Golde, P. (2011). *C# Programming Language (Covering C# 4.0).* (4a. ed.). (Microsoft .NET Development Series). Estados Unidos: Addison-Wesley.