

# Capítulo 06



## Arreglos

Ania Cravero Leal

Departamento de Ingeniería de Sistemas  
Facultad de Ingeniería, Ciencias y Administración

“Proyecto financiado por el Fondo de Desarrollo Educativo de  
la Facultad de Ingeniería, Ciencias y Administración de la  
Universidad de La Frontera”

Versión  
1.0

**TEMARIO**

- 6.1 Conceptos Básicos
- 6.2 Arreglos Unidimensionales
  - 6.2.1 Declaración de arreglos
  - 6.2.2 Acceso a un arreglo
  - 6.2.3 Inicialización de arreglos
  - 6.2.4 Utilizando ciclos para acceder a un arreglo
  - 6.2.5 Búsquedas
  - 6.2.6 Ordenamiento
- 6.3 Arreglos Multidimensionales
  - 6.3.1 Fundamentos de arreglos multidimensionales
  - 6.3.2 Matrices
  - 6.3.3 Declaración de matrices
  - 6.3.4 Acceso a una matriz
- 6.4 Cadenas
  - 6.4.1 Fundamentos de cadenas
  - 6.4.2 Declaración de cadenas
  - 6.4.3 Lectura y escritura de cadenas
  - 6.4.4 Funciones propias para cadenas
- 6.5 Ejercicios Resueltos
- 6.6 Ejercicios Propuestos
- 6.7 Comentarios Finales

# Arreglos

---

Usamos un arreglo para procesar una colección de datos del mismo tipo, como una lista de temperaturas registradas durante un día o una lista de nombres obtenida desde un archivo de alumnos de un curso. En este capítulo te presentaremos los fundamentos de la definición y uso de arreglos en el lenguaje Java, y muchas de las técnicas básicas que se emplean para diseñar algoritmos y programas que usan arreglos.

## 6.1 Conceptos Básico

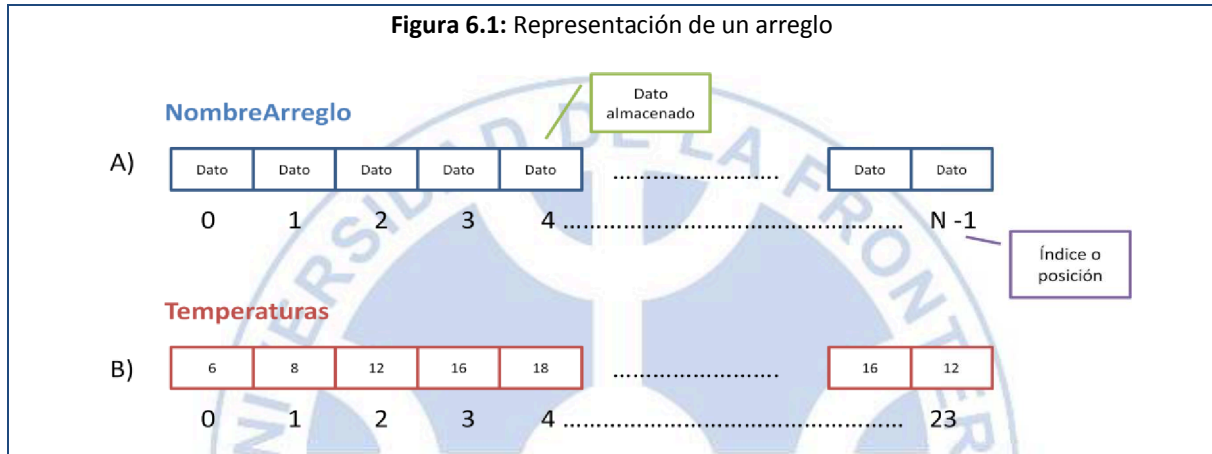
Supongamos que deseamos obtener las temperaturas del día viernes cada una hora, con el fin de determinar cuál fue la temperatura más alta, cuál fue la más baja, y cuál es el promedio de las mismas. No sabremos cuál es la temperatura más alta o más baja hasta que leamos todas las temperaturas, por tanto, debemos mantener en la memoria las 24 temperaturas para que, una vez determinada la temperatura más alta, cada temperatura se pueda comparar con ella.

Para retener las 24 temperaturas requerimos algo equivalente a 24 variables de tipo int. Podríamos usar 24 variables individuales, pero no es fácil seguir la pista a 24 variables, y probablemente más adelante quisiéramos cambiar nuestro programa para registrar temperaturas cada media hora, o en cada minuto. Sin lugar a dudas 48 o más variables será poco práctico. Un arreglo es la solución perfecta para este tipo de situaciones.

### a. Arreglo:

Un arreglo es una zona de memoria (variable) que puede almacenar un conjunto de N datos del mismo tipo.

Por lo tanto, un arreglo es una variable que dispone de una gran zona de memoria separada en N celdas del mismo tamaño. La figura 6.1 a) presenta una representación de la zona de memoria de un arreglo.



Observa que en la figura 6.1 b) está representado un arreglo con nombre Temperaturas que almacena las 24 temperaturas registradas del día viernes. Así tenemos que en la celda número 2 (índice 2) se ha registrado una temperatura de 12 °C.

**OJO:** No debemos confundir **índice** con **dato almacenado**. El índice indica sólo una posición de cada celda del arreglo, las que están numeradas desde el número cero hasta N-1. Así por ejemplo, si necesitamos un arreglo con 24 celdas, entonces estas serán automáticamente numeradas desde el cero hasta 23. Por otro lado, el dato almacenado puede ser de cualquiera de los tipos de datos que ya conocimos en el capítulo 5 de este libro. Por ejemplo, puede ser un número entero, un número con punto flotante, una letra, un carácter, etc.

## 6.2 Arreglos Unidimensionales

Un arreglo unidimensional es capaz de almacenar N datos del mismo tipo, tal y como hemos mostrado en la figura 6.1

### Ejemplo 6.1:

Algunos ejemplos de declaración de arreglos unidimensionales para el lenguaje Java son los siguientes:

```
char apellido[] = new
char[50];
```

//declaramos un arreglo que puede almacenar una secuencia de hasta 50 caracteres.

```
float peso[] = new
float[20];
```

//declaramos un arreglo que puede almacenar 20 números reales.

El ejemplo 6.1 muestra arreglos declarados utilizando varios tipos de datos y tamaños, para lenguaje Java.

### 6.2.1 Declaración de arreglos

Como toda variable, antes de utilizar un arreglo debemos declararlo. Es decir, debemos definir un nombre que lo representa, un tipo de dato, y la cantidad de celdas que tendrá. La figura 6.2 presenta un ejemplo de declaración en el lenguaje Java.

**Figura 6.2:** Ejemplo declaración de un arreglo unidimensional

Java

```
int temperatura[] = new int[24];
```

**Nombre arreglo:** Temperatura  
**Tipo:** int  
**Cantidad de celdas:** 24  
(numeradas de cero a 23)

Observa que en el lenguaje Java la forma general para declarar un arreglo es la siguiente:

**Tipo nombre[] = new tipo[numCeldas];**

**Nota:** El operador **new** de Java sirve para realizar las siguientes acciones:

- Separar memoria para el nuevo objeto que se almacenará. En la figura 6.2 **new** separa memoria del computador para almacenar un arreglo de 24 celdas de tipo entero.
- Invocar el método (función) de inicio de la clase llamado constructor. Esto debido a que en ocasiones Java requerirá crear un gran espacio de memoria, es decir, se debe construir el espacio.
- Retornar la referencia a un nuevo objeto. En Java los objetos almacenan referencias, a diferencia de las variables primitivas que almacenan datos.

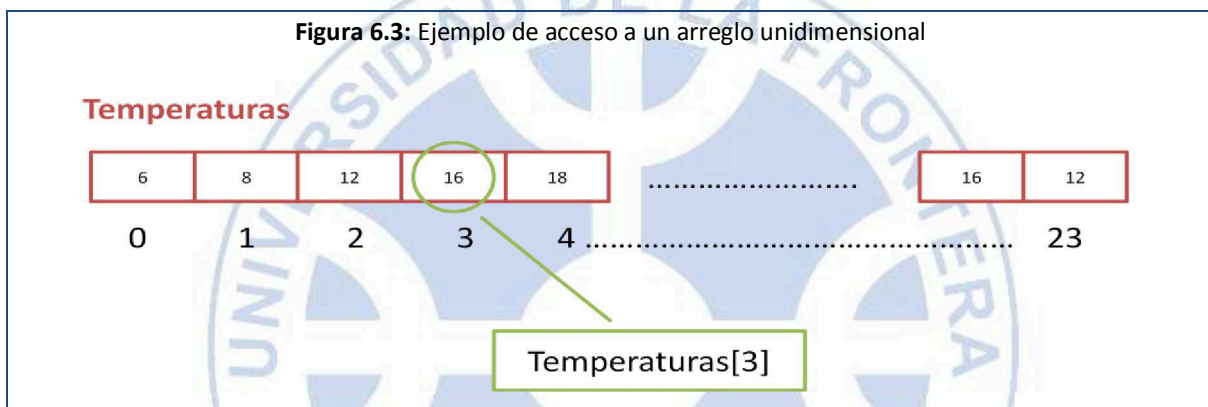
Entonces, ¿podemos decir que un arreglo es un objeto en Java?

**OJO!!:** En Java existe un tipo **string** que crea de manera automática un arreglo que puede almacenar hasta 256 caracteres. La forma de declarar el arreglo en Java es :

`String nombre[];`

### 6.2.2 Acceso a un arreglo

Para acceder al contenido de un arreglo debemos utilizar el nombre del arreglo y el índice o posición en donde está ubicado dicho dato. La figura 6.3 muestra un ejemplo de acceso al arreglo *Temperaturas*.



La figura anterior muestra el arreglo **Temperaturas** con 24 datos almacenados. El recuadro verde accede específicamente al dato ubicado en la posición 3, es decir, la temperatura 16 °C.

#### Ejemplo 6.2:

Este ejemplo muestra un programa en lenguaje Java, que accede a cada celda de un arreglo, tanto para almacenar los datos como para procesarlos.

**Lenguaje**

```
public class Arreglo {
    public static void main(String[] args) {
        int numeros[] = new int[4];
        numeros[0] = 10;
        numeros[1] = 2;
        numeros[2] = -5;
        numeros[3] = numeros[0] + numeros[2];
        System.out.println(numeros[0]);
        System.out.println(numeros[1] + " " + numeros[2]);
        System.out.println(numeros[2+1]);
    }
}
```

**Java:**

**numeros**

10	2	-5	5
0	1	2	3

Imprime por pantalla  
10 2 -5 5

Observemos que `numeros[2+1];`  
es lo mismo que `numeros[3];`



Observamos que para almacenar un dato en una celda específica del arreglo necesitamos el nombre del arreglo y la ubicación de la celda (índice). De tal manera que

```
numeros[2] = -5
```

le asigna el número -5 a la celda en posición dos, y que

```
numeros[3] = numeros[0] + numeros[2];
```

le asigna la suma del contenido de las celdas cero y dos a la celda número tres, es decir, asigna  $10 - 5$  que es 5.

También podemos ingresar los números desde el teclado. En tal caso, tendríamos lo siguiente :

**Lenguaje Java:**

```
numeros[0] = Integer.parseInt(In.readLine());
```

### 6.2.3 Inicialización de arreglos

Podemos inicializar un arreglo al declararlo. Al hacerlo, los datos que se almacenarán en cada celda se encierran en llaves y se separan con comas. Por ejemplo:

**Lenguaje Java:**

```
int[] numeros={2, -4, 15};
```

el número encerrado en los corchetes especifica cuántas celdas de memoria tiene el arreglo. Cuando se usan en otro lugar el programa, el número encerrado en los corchetes indica cuál celda es la que se está utilizando o accediendo.

**OJO!!** El índice entre corchetes no tiene que ser siempre una constante entera. Podemos usar cualquier expresión en los corchetes en tanto la evaluación de la expresión dé uno de los enteros de cero hasta uno menos del tamaño del arreglo. Por ejemplo, lo que sigue asigna el dato 23 a pesos[2]:

```
int n=1;
```

```
pesos[n+1]=23;
```

Aunque se vea diferentes, pesos[n+1] es lo mismo que pesos[2]. La razón es que  $n + 1$  es igual a 2.

La identidad de una celda específica, como pesos[i], la determina el valor de su índice, que en este caso es i. Así, podemos escribir programas que digan cosas como “hacer esto y lo otro con la celda i-ésima del arreglo”, donde el programa calcula el valor de i. Esto es muy utilizado cuando disponemos de muchas celdas en el arreglo y necesitamos realizar varias acciones con las mismas. En la siguiente sección te mostraremos ejemplos de ello.

**Nota:** El uso de *i* como índice lo usaremos debido a que es muy común que cualquier programador, o incluso en los libros, utilicen como índices las letras *i*, *j* o *k*.

Recuerda que el declarar variables con nombres inapropiados no es una buena práctica de programación, por tanto, esta sería una excepción.

#### 6.2.4 Utilizando ciclos para acceder a un arreglo

Cuando un arreglo dispone de varias celdas de memoria es muy común utilizar ciclos `for` para acceder a cada una de las celdas, ya sea para asignar datos, para leer datos desde el teclado, para procesar los datos o para mostrar cada dato almacenado en las distintas celdas de memoria.



Veamos algunos ejemplos prácticos.

### Ejemplo 6.3:

Este ejemplo muestra el promedio de 10 números ingresados en un arreglo de enteros.

Lenguaje Java:

```
import java.io.InputStreamReader;
import java.io.BufferedReader;

public class Promedio {
    public static void main(String[] args) {
        int numeros[] = new int[10];
        int suma = 0, i;
        float prom = 0;
        BufferedReader In = new BufferedReader (new InputStreamReader(System.in));
        try{
            for (i=0; i<10; i++){
                System.out.println(" Ingrese un número en la posicion: " + i);
                numeros[i] = Integer.parseInt(In.readLine());
                suma=suma + numeros[i];
            }
            prom =(float) suma / 10;
            System.out.println(" El promedio es " + prom);
        }catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

El índice i va obteniendo los valores desde el cero hasta 9 a medida que el ciclo for se ejecuta

Leemos un dato para la celda i. Cuando i es cero, el dato queda almacenado en numeros[0]

Para calcular el promedio debemos utilizar el conversor (float) ya que suma/10 entrega como resultado un entero y no un número real. (float) convertirá el resultado en un número real.

**También podríamos haber utilizado una constante para el tamaño del arreglo**

**Final int N = 10**

**Así podemos declarar el arreglo así:**

**int numeros[] = new int[N];**

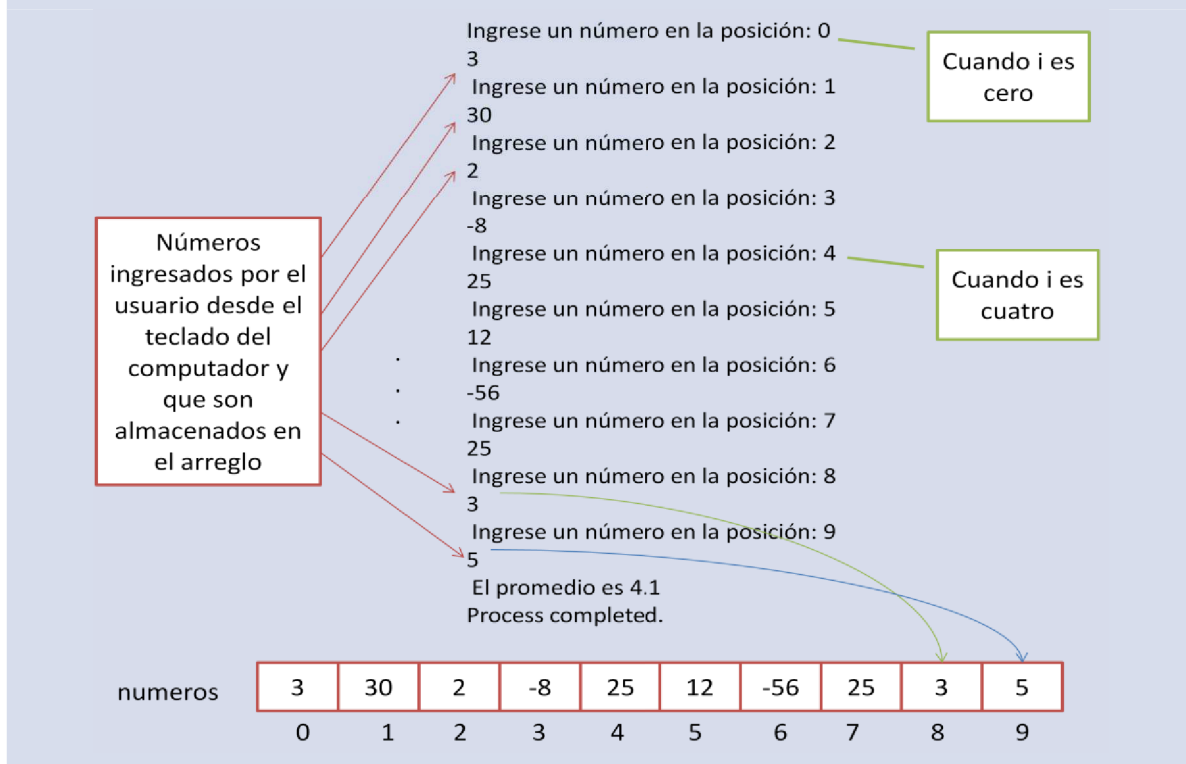
**Y el ciclo for sería:**

**for (i=0; i < N; i++)**

**La ventaja es que si necesitamos cambiar el tamaño del arreglo en cualquier momento, basta con modificar el valor de la constante N**



La figura siguiente muestra un ejemplo de ejecución del programa escrito en Java.



**Ejemplo 6.4:**

El siguiente programa escrito en Java muestra el mayor de 10 números ingresados por el usuario.

```
package com.ufro.dis.libro.cap6;
import java.util.Scanner;

public class Main {

    final int N = 10;

    public static void main(String[] args) {
        int[] numeros = new int[N];
        int i;
        int max;
        Scanner intro = new Scanner(System.in);

        //Primero ingresamos los numeros
        for(i = 0; i < N; i++){
            System.out.print("Ingrese un número para la celda n°" + i + ": ");
            numeros[i] = intro.nextInt();
        }

        //Despues buscamos el mayor
        max = numeros[0];
        for(i = 1; i < N; i++){
            if(max < numeros[i]){
                //Actualizamos max con el mayor encontrado ahora
                max = numeros[i];
            }
        }
        System.out.print("El mayor de los números es: " + max);
    }
}
```

En primer lugar, debemos ingresar los números al arreglo. Para ello utilizamos un ciclo **for** con un índice *i* que tomará los valores desde el cero hasta N-1 (es decir 9) a medida que el ciclo se va ejecutando.

Después de ello, podemos buscar el número mayor. Como no sabemos en qué posición se encuentra exactamente utilizaremos una variable auxiliar (variable **max**) que se inicializará con el primer número almacenado en el arreglo (es decir *numero[0]*). A medida que avanza el ciclo **for**, debemos comparar el contenido de la variable **max** con la siguiente celda de memoria (la primera vez compara con *numero[1]*); en caso de que **max** contiene un número menor a la de la celda que actualmente está comparando (*if (max < numero[i])*) quiere decir que debemos actualizar **max** con el nuevo número que hasta ahora es el mayor. El proceso finaliza cuando comparamos **max** con la última celda del arreglo (*numeros[9]*), si el contenido de ésta es mayor que **max**, entonces **max** se vuelve a actualizar con el valor almacenado en *numero[9]*, en otro caso, **max** mantiene su valor actual. La siguiente figura muestra el proceso cuando *i* varía desde cero hasta N-1.

max 3 Inicializamos max con el primer número del arreglo

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

Cuando  $i = 1$ , comparamos max con la celda `numeros[1]`. Como 3 es menor que 30, entonces max se actualiza con el valor 30

max 30

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

Cuando  $i = 2$ , comparamos max con la celda `numeros[2]`. Como 30 no es menor que 2, entonces max mantiene su valor en 30

max 30

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

Cuando  $i = 3$ , comparamos max con la celda `numeros[3]`. Como 30 no es menor que -8, entonces max mantiene su valor en 30

max 30

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

Cuando  $i = 4$ , comparamos max con la celda `numeros[4]`. Como 30 no es menor que 25, entonces max mantiene su valor en 30

max 30

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

.....

Cuando  $i = 9$ , comparamos max con la celda `numeros[9]`. Como 30 no es menor que 5, entonces max mantiene su valor en 30

max 30

numeros	3	30	2	-8	25	12	-56	25	3	5
	0	1	2	3	4	5	6	7	8	9

Observamos que finalmente `max` almacena el dato 30, que justamente es el número mayor de los ingresados en el arreglo.

¿Qué debemos modificar del programa del ejemplo 6.4 para que muestre el número menor?

¿Es posible encontrar el número menor sin utilizar una variable auxiliar? Comenta con tus compañeros.

## 6.2.5 Búsquedas

Existen varios algoritmos para buscar un elemento en una lista, siendo los más conocidos, la búsqueda secuencial y la búsqueda binaria. Estos algoritmos permiten solucionar problemas del tipo, “encontrar los datos de un cliente”, “encontrar las características de un producto”, “encontrar un determinado número”, etc.

El siguiente ejemplo muestra un programa escrito en Java que permite buscar un número específico en una lista de números. El algoritmo que hemos aplicado es la búsqueda secuencial.

### Ejemplo 6.5:

El próximo programa escrito en Java permite buscar un número en el arreglo y decir si está o no está en el arreglo.

```
package com.ufro.dis.libro.cap6;
import java.util.Scanner;

public class Main {
    final int N = 10;

    public static void main(String[] args) {
        int[] numeros = new int[N];
        int i;
        int esta = 0; /* esta = 0 significa que supondremos que el número que desea
                       buscar el usuario no está en el arreglo */
        int NumBuscar;
        Scanner intro = new Scanner(System.in);

        //Primer ingreso de los números
        for(i = 0; i < N; i++){
            System.out.print("Ingrese un número para la celda n°" + i + ": ");
            numeros[i] = intro.nextInt();
        }
        // después el usuario ingresa el número que desea buscar
        System.out.print("Ingrese el número que desea buscar: ");
        NumBuscar = intro.nextInt();

        //después buscamos el número ingresado en el arreglo
        for(i = 0; i < N; i++){
            if(NumBuscar == numeros[i]){
                //cambiamos la variable esta a 1, que significa verdadero
                esta = 1;
                break;
            }
        }
        if(esta == 0){ //si mantuvo el valor cero quiere decir que el número no está
            System.out.print("El " + NumBuscar + " NO está en el arreglo");
        }else{
            System.out.print("El " + NumBuscar + " SI está en el arreglo");
        }
    }
}
```

La variable “esta” trabajará como una bandera. De esta forma cuando almacene 0 significa falso y cuando almacene 1 significa verdadero.

**Nota:** existen las variables booleanas, así que podríamos utilizar la variable de la siguiente manera:

`bool esta = false;`

Y dentro del if sería:

`esta = true;`

En primer lugar debemos ingresar los números al arreglo. Para ello utilizamos un ciclo *for* con la variable de control *i*, que hace de índice en cada celda del arreglo.

En segundo lugar, el usuario debe ingresar el número que desea buscar para saber si está o no está en el arreglo. Para ello, utilizamos la variable **NumBuscar** que almacena dicho número.

Para buscar el número (variable *NumBuscar*) en el arreglo, necesitaremos comparar dicho número con cada número almacenado en las celdas del arreglo. Así que nos conviene nuevamente utilizar un ciclo *for* con la variable de control *i* para recorrer todas las celdas del arreglo *numeros*.

La siguiente figura muestra el proceso de búsqueda cuando *i* varía de cero a N-1. Supondremos que el usuario ya ha ingresado los números al arreglo y que *NumBuscar* es 2.

Cuando  $i = 0$ , comparamos NumBuscar con la celda `numeros[0]`. Como 2 no es igual a 3, entonces la variable `esta` mantiene su valor en 0

NumBuscar	2	esta			0						
numeros	3	30	2	-8	25	12	-56	25	3	5	
	0	1	2	3	4	5	6	7	8	9	

Cuando  $i = 1$ , comparamos NumBuscar con la celda `numeros[1]`. Como 2 no es igual a 30, entonces la variable `esta` mantiene su valor en 0

NumBuscar	2	esta				0					
numeros	3	30	2	-8	25	12	-56	25	3	5	
	0	1	2	3	4	5	6	7	8	9	

Cuando  $i = 2$ , comparamos NumBuscar con la celda `numeros[2]`. Como 2 es igual a 2, entonces la variable `esta` cambia su valor a 1, y finaliza el ciclo al ejecutar `break`.

NumBuscar	2		esta		1						
numeros	3	30	2	-8	25	12	-56	25	3	5	
	0	1	2	3	4	5	6	7	8	9	

Por lo tanto `if (esta == 0)` será falso e imprimirá "El 2 SI está en el arreglo".

**OJO!!** Cuando escribimos `if (esta == 0)` es lo mismo que decir si *esta* es falso. Algunos programadores prefieren escribir la condición de la siguiente manera.

`if (!esta)`

Que también significa si *esta* es falso.

**RIESGO índice de arreglo fuera de intervalo:** El error de programación más común que se comete al usar arreglos es tratar de hacer referencia a un elemento inexistente del arreglo. Por ejemplo, considera la siguiente declaración de un arreglo

`int[] notas = new int[6];`

Al utilizar el arreglo *notas*, toda expresión que se use como índice deberá dar uno de los valores enteros 0 a 5 al evaluarse. Por ejemplo, si el programa contiene la sentencia `notas[i]`, la evaluación de *i* debe dar alguno de los enteros de 0 a 5. Si la evaluación de *i* da alguna otra cosa, será un error. En este caso, decimos que el índice está fuera del intervalo o simplemente que no es válido.

Un ejemplo de error sería si ejecutamos la siguiente instrucción

`notas[7] = 220;`

En este caso, estaríamos almacenado el dato 220 en una zona de memoria ubicada en la supuesta posición 7 del arreglo, y borraríamos lo que estaba almacenado allí, quizás por otra variable o por el propio sistema.

La siguiente figura representa la situación.



### 6.2.6 Ordenamiento

También existen algoritmos que permiten ordenar una lista de elementos. Algunos de los algoritmos más conocidos son, el de la burbuja, por selección, por inserción, por intercambio y el quicksort. Estos algoritmos permiten resolver problemas del tipo “mostrar los datos de los clientes en orden alfabético”, “mostrar las ventas de la semana de manera ascendente”, etc.

A continuación te mostramos un programa que permite ordenar números utilizando el famoso algoritmo de la burbuja.



### Ejemplo 6.6: Ordenamiento de la Burbuja

La **Ordenación de burbuja** (**Bubble Sort** en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el **método del intercambio directo**. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

El algoritmo básico es el siguiente:

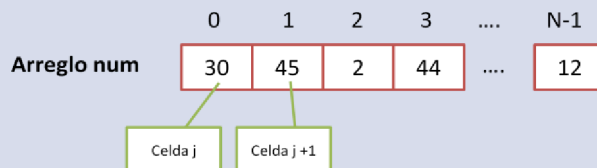
```

Algoritmo DeLaBurbuja
Para i de 2 hasta n hacer
  Para j de 0 hasta n-i hacer
    Si (num[j] > num[j+1]) entonces
      aux ← num[j]
      num[j] ← num[j+1]
      num[j+1] ← aux
    Fin si
  Fin para
Fin para
Fin algoritmo
  
```

Este ciclo **para** sirve para ejecutar el algoritmo **n** veces. Siendo **n** la cantidad de celdas del arreglo

Este ciclo **para** sirve para recorrer el arreglo

La estructura selectiva permite comparar las celdas **j** y **(j+1)**. Si la primera contiene un número mayor, se intercambian



Un ejemplo de ejecución del algoritmo se muestra a continuación.

Sea el arreglo num con 5 datos almacenados


	0	1	2	3	4
num	45	30	2	23	15

Cuando  $j$  es 0, se compara las celdas 0 y 1 puesto que  $j+1$  es 1. Como  $\text{num}[j] > \text{num}[j+1]$  es verdadero. Es decir,  $45 > 30$  entonces se intercambian los datos de las celdas.


	0	1	2	3	4
num	45	30	2	23	15

aux ← num[j]  


	0	1	2	3	4
num	30	30	2	23	15

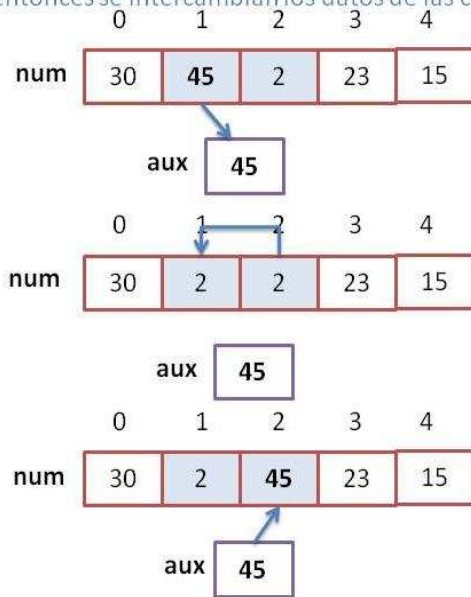
a[j] ← num[j+1]  


	0	1	2	3	4
num	30	45	2	23	15

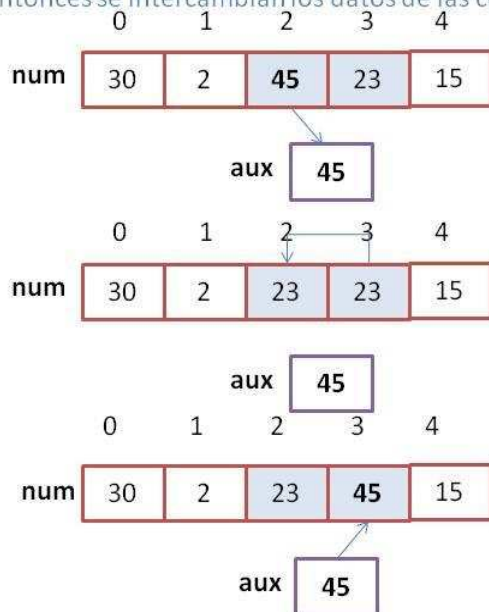
num[j+1] ← aux  




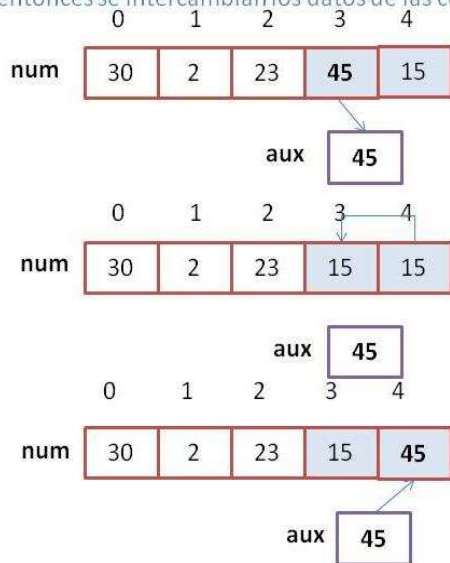
Cuando  $j$  es 1, se compara las celdas 1 y 2 puesto que  $j+1$  es 2.  
 Como  $\text{num}[j] > \text{num}[j+1]$  es verdadero. Es decir,  $45 > 2$   
 entonces se intercambian los datos de las celdas.



Cuando  $j$  es 2, se compara las celdas 2 y 3 puesto que  $j+1$  es 3.  
 Como  $\text{num}[j] > \text{num}[j+1]$  es verdadero. Es decir,  $45 > 23$   
 entonces se intercambian los datos de las celdas.



Cuando  $j$  es 3, se compara las celdas 3 y 4 puesto que  $j+1$  es 4.  
Como  $\text{num}[j] > \text{num}[j+1]$  es verdadero. Es decir,  $45 > 15$   
entonces se intercambian los datos de las celdas.



Observemos que el número **45**, que es el mayor de todo el arreglo, ha quedado almacenado en la última posición. Es como si una burbuja subiera.

Ahora quedará ejecutar el mismo proceso 4 veces más para que finalmente el arreglo quede totalmente ordenado.

El próximo programa escrito en Java permite ordenar los números de un arreglo de manera ascendente de acuerdo al método de ordenamiento de la Burbuja.

```
package com.ufro.dis.libro.cap6;
import java.util.Scanner;

public class Main {
    final int N = 10;

    public static void main(String[] args) {
        int[] numeros = new int[N];
        int i;
        int j;
        int aux;
        Scanner intro = new Scanner(System.in);

        //Primero ingresamos los numeros
        for(i = 0; i < N; i++){
            System.out.print("Ingrese un número para la celda n°" + i + ": ");
            numeros[i] = intro.nextInt();
        }

        //después ordenamos los números
        for (i=2; i <= N; i++){
            for (j =0; j < N-i ; j++){
                if ( numeros[j] > numeros[j+1]){
                    // intercambiamos las variables
                    aux = numeros [j];
                    numeros [j] = numeros [j+1];
                    numeros [j+1] = aux;
                }
            }
        }

        //Finalmente mostramos los números ordenados
        for (i=0; i < N; i++){
            System.out.println("numeros [" + i +"] =" + numeros[i] );
        }
    }
}
```

¿Cómo podemos ordenar de mayor a menor?

Modifica el programa anterior para solucionar el problema.

## 6.3 Arreglos Multidimensionales

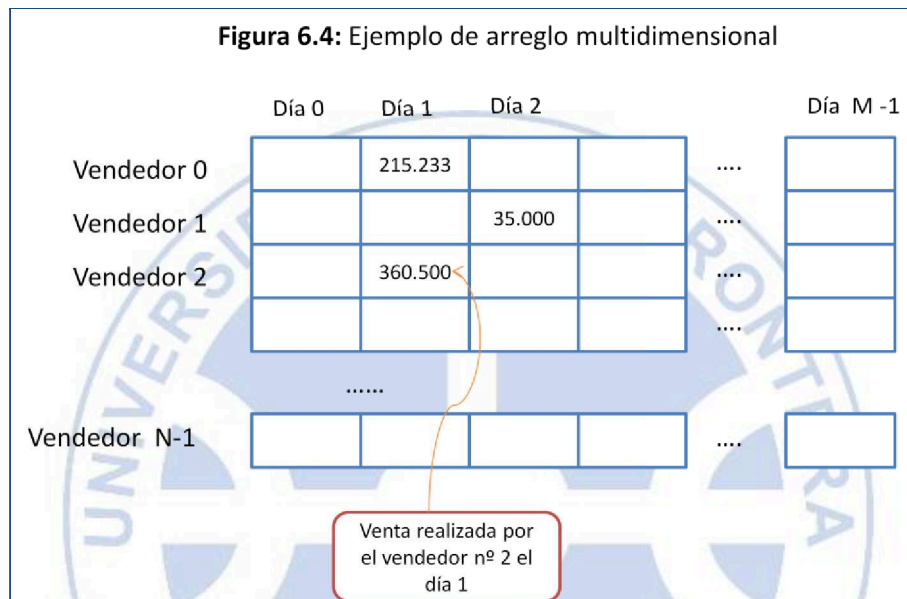
Java permite declarar arreglos con más de un índice. En esta sección describiremos estos arreglos multidimensionales.

### 6.3.1 Fundamentos de arreglos multidimensionales

A veces es útil disponer de un arreglo con más de un índice, ya que nos da la posibilidad de manipular información que requiere dos o más dimensiones. Por ejemplo, si una empresa desea registrar las ventas

diarias, por un mes, de cada vendedor de la tienda. En este caso necesitamos dos dimensiones: día y vendedor, y los datos que almacenaríamos en el arreglo son las ventas.

La figura 6.4 muestra un esquema que representa el arreglo bidimensional para registrar las ventas de los vendedores de la empresa.



Observa que la figura 6.4 muestra un arreglo con dos dimensiones, el número del vendedor para las filas y el número del día del mes para las columnas. En general la empresa dispone de N vendedores y el mes es de M días; y las ventas realizadas, que son los datos, son almacenados en las celdas. Así tenemos que la venta del vendedor 2 el día 1 es de \$360.500.

A este tipo de arreglo se le llama comúnmente matriz, ya que para acceder a los valores almacenados necesitamos conocer el índice de la fila y el índice de la columna de cada celda específica.



**Ejemplo 6.7:**

Ahora te mostraremos algunos ejemplos de declaraciones de matrices utilizando distintos tipos de datos y dimensiones.

En Java

```
float promedios[][] = new
float[10][25];
```

```
long pesos[][] = new long
[5][33];
```

**6.3.2 Matrices****a. Matriz:**

Una matriz es un arreglo bidimensional que almacena de manera general  $N \times M$  datos del mismo tipo.  $N$  corresponde al número de filas y  $M$  corresponde al número de columnas.

**6.3.3 Declaración de matrices**

Para declarar una matriz necesitamos un tipo de dato, un nombre que la represente, y la definición de su tamaño. El tamaño de la matriz está determinado por la cantidad de filas y columnas. La figura 6.5 muestra un ejemplo de declaración en Java.

**Figura 6.5:** Ejemplos de declaración de una matriz

Java

```
int ventas [][] = new int[6][30];
```

**Nombre de la matriz:** Temperatura

**Tipo:** Int

**Cantidad de filas:** 64 (numeradas de cero a 5)

**Cantidad de columnas:** 30 (numeradas de cero a 29)

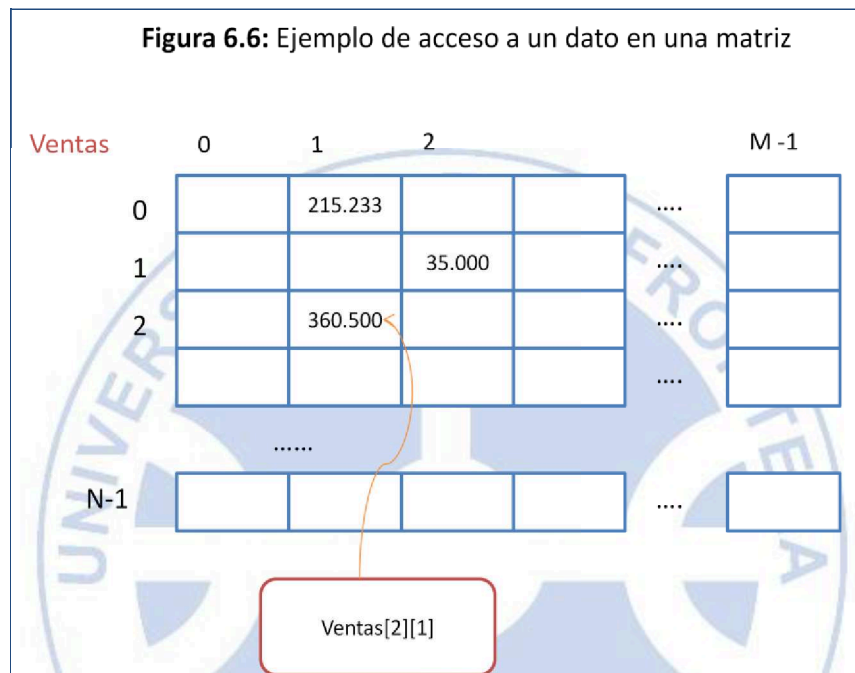
Observa que en Java, las matrices se declaran de la siguiente forma general

**Tipo nombreMatriz [][] = new tipo [numFilas][numColumnas];**

El ejemplo 6.6 presenta algunos ejemplos de declaraciones de.

### 6.3.4 Acceso a una matriz

Para acceder al contenido de una celda específica de una matriz debemos hacerlo mediante su nombre, posición de la fila, y posición de la columna. La figura 6.6 muestra un ejemplo de acceso.



Observa que `Ventas[2][1]` representa el contenido de la celda ubicada en la fila 2 y columna 1, es decir, los 360.500.

Nuevamente, y al igual que los arreglos unidimensionales, se deberá utilizar índices (por lo general, *i*, *j* y *k*) para representar la posición en la que se encuentra la celda que se está accediendo desde un programa. La única diferencia, es que ahora necesitaremos dos índices, el primer para determinar la posición de la fila y el segundo para determinar la posición de la columna.

El ejemplo siguiente muestra un programa (en los tres lenguajes) que declara y accede a una matriz.

**Ejemplo 6.8:**

El siguiente programa permite ingresar desde el teclado las ventas de una semana para los 10 vendedores de una empresa. Luego el programa calculará el promedio de ventas semanal par cada vendedor.

**Lenguaje Java**

```
package com.ufro.dis.libro.cap6;

import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Ventas{
    public static void main (String [] args) {
        final int VENDEDORES =10;
        final int DIAS =7;
        int ventas[] []= new int[VENDEDORES][DIAS];
        int suma =0, i,j;
        float promedio =0;
        BufferedReader In =new BufferedReader (new InputStreamReader(System.in));

        try{
            for (i=0; i< VENDEDORES; i++){
                System.out.println("Ingrese las ventas de la semana para el vendedor: " + i);
                for (j=0; j < DIAS; j++){
                    System.out.println("Ingrese ventas para el día: " + j);
                    ventas[i][j] = Integer.parseInt(In.readLine());
                }
            }
        } catch (Exception e){
            e.printStackTrace();
        }
        for (i=0; i < VENDEDORES; i++){
            suma = 0;
            for (j =0; j < DIAS ; j++){
                suma = suma + ventas[i][j];
            }
            promedio = (float ) suma /DIAS;
            System.out.println("El promedio de ventas para el vendedor " + i + "es " + promedio);
        }
    }
}
```

Modifica el programa anterior (escoge algún lenguaje) para responder las siguientes preguntas:

- Qué vendedor vende más en la semana
- Qué día de la semana se logra la mayor venta (considere la venta de un día la suma de ventas década vendedor para ese día)
- Qué vendedores lograron ventas superiores a \$100.000. Indique el día en que lo logra.

## 6.4 Cadenas

Hasta ahora te hemos presentado ejemplos de arreglos y matrices que procesan números. En este apartado te mostraremos que los arreglos y matrices también pueden almacenar caracteres. A este tipo de vectores le llaman cadenas.

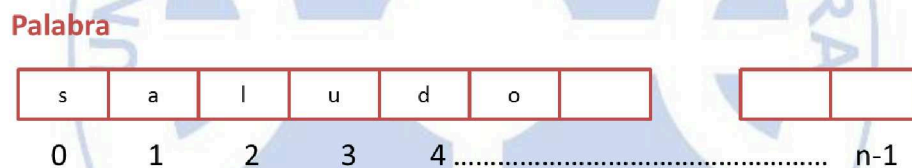
### 6.4.1 Fundamentos de cadenas

#### a. Cadena

Una cadena es un vector que almacena N caracteres, por lo tanto cada celda puede almacenar un dato de tipo char.

Las cadenas se almacenan en un arreglo de caracteres. La figura 6.7 muestra una representación de un arreglo que almacena una cadena de caracteres

**Figura 6.7:** Ejemplo de cadena



Para manejar las cadenas contamos con bibliotecas que disponen de las funciones necesarias para su funcionamiento.

En Java las cadenas son objetos de las clases predefinida String o StringBuffer, que están incluidas en el paquete java.lang.\*

**String:** Java crea un arreglo de caracteres o una cadena. A estas cadenas accedemos a través de las funciones que poseen esta clase.

### 6.4.2 Declaración de cadenas

- Cadenas tipo *char* para Java:

Una variable de cadena es sólo un arreglo de caracteres. Por tanto, la siguiente declaración de un arreglo nos proporciona una variable de cadena capaz de almacenar un valor de cadena de tipo *char* con nueve o menos caracteres.

En la declaración de un arreglo de caracteres sería de la siguiente manera.

```
char [] palabra = new char[10];
```

Podemos inicializar una cadena al declararla, como se muestra a continuación:

```
char miMensaje[] = {'H','o','l','a',' ','a','l','u','m','n','o','s'};
```

Observa que la cadena que se asigna a la variable *miMensaje* no necesita llenar todo el arreglo.

Cuando inicializamos una cadena se puede omitir el tamaño el arreglo

Una cadena es una variable, pero también es un arreglo, por lo tanto cada celda posee un índice y podemos utilizarlo como cualquier otro arreglo. Revisemos el siguiente ejemplo:

**Nota:** Debemos tener cuidado al asignar una cadena a un arreglo de caracteres, ya que el uso del símbolo = es **ilegal** para este caso. Es decir

```
char cadena[20];
```

```
cadena = "hola amigos";
```



ilegal!!

Traerá un error ya que no será posible asignar cada letra en las celdas del arreglo.

Esto se debe a que cadena es un puntero implícito y recibirá como dato sólo direcciones de memoria. Pero este tema no será abordado en este libro.

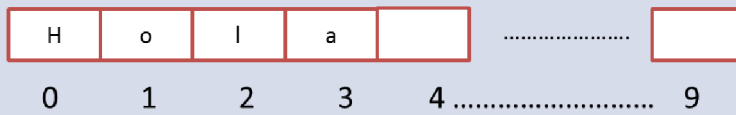
### Ejemplo 6.9:

Supongamos que nuestro programa contiene la siguiente declaración de una cadena:

```
char palabra[10]= {'h','o','l','a'}
```

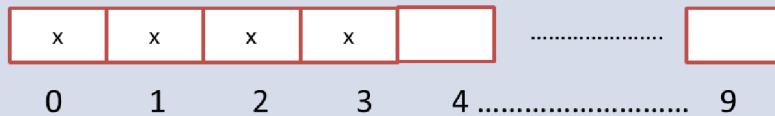
Al declarar palabra como se muestra arriba, nuestro programa tiene las siguientes celdas con datos almacenados: palabra[0], palabra[1], palabra[2], palabra[3].

#### Palabra



Por ejemplo, el siguiente código modificará el valor actual de la cadena palabra para convertirla en una nueva cadena que contiene en todas las celdas ocupadas la letra 'x'.

#### Palabra



```
import java.io.*;
public class CadenaChar {
    public static void main (String [] args) {
        char palabra[] = {'h','o','l','a'};
        int i;
        for (i=0; i<4; i++)
            palabra[i] = 'x';
    }
}
```



### 6.4.3 Lectura y escritura de cadenas

- **La clase string de Java:**

En Java las cadenas son objetos de las clases predefinida **String** o **StringBuffer**, que están incluidas en el paquete **java.lang.\***

Esto quiere decir que siempre que aparecen conjuntos de caracteres entre comillas dobles, el compilador de Java crea automáticamente un objeto **String**. El compilador es más eficiente y usa un objeto **StringBuffer** para construir cadenas a partir de las expresiones, creando el **String** final sólo cuando es necesario.

Los caracteres de las cadenas tienen un índice que indica su posición. El primer carácter de una cadena tiene el índice 0, el segundo el 1, el tercero el 2 y así sucesivamente; tal y como lo hemos visto con los arreglos numéricos, sólo que en esta caso cada celda almacena sólo un carácter.

Los strings u objetos de la clase **String** se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

En este caso, Java crea un objeto de la clase **String** automáticamente.

Para crear un string explícitamente escribimos

```
String frase=new String("El primer programa");
```

También se puede escribir de manera alternativa

```
String frase="El primer programa";
```

**Nota:** También podemos asignar una cadena de manera directa a la variable, tal y como lo hacemos con las variables comunes (float, int, double, etc.)

```
String frase;
```

```
frase="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

```
String frase="";
```

```
String frase=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase *String*. Sin embargo,

**String str;**

está declarando un objeto **frase** de la clase **String**, pero aún no se ha creado ningún objeto de esta clase, es decir, no se ha construido el objeto en la memoria.

Para ingresar una cadena en Java desde el Buffer, debemos utilizar el método `readLine()`. Observemos el siguiente ejemplo escrito en JCreator.

#### Ejemplo 6.10:

El siguiente programa permite al usuario ingresar una cadena desde el Buffer y luego la muestra por pantalla.

```
import java.io.*;

public class CadenaBuffer {
    public static void main (String [] args) {
        String cadena;
        BufferedReader In =new BufferedReader (new InputStreamReader(System.in));

        try{
            System.out.print(" Ingrese una cadena:");
            cadena = In.readLine();
            System.out.println(" Su cadena es " + cadena);
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

**Nota:** Observemos que al leer una cadena desde el Buffer no debemos utilizar alguna función de conversión de datos como lo hacíamos con las variables de tipo `int`, `float`, `double`, etc. Esto se debe a que el Buffer siempre contendrá datos de tipo alfanuméricos.

**Nota:** Podemos acceder a los caracteres de una variable de tipo **string** tal y como lo hicimos con los arreglos de tipo **char**, por lo que las variables de tipo **string** poseen todas las ventajas de los arreglos de caracteres, además de otras ventajas que los arreglos de caracteres no tienen.

**Ejemplo 6.11:**

El siguiente programa permite ingresar una frase y luego se mostrará carácter a carácter.

```
import java.io.*;

public class MostrarCarac {
    public static void main (String [] args) {
        BufferedReader In =new BufferedReader (new InputStreamReader(System.in));
        String cadena;
        int i;

        try{
            System.out.print(" Ingrese una cadena:");
            cadena = In.readLine();
            for (i=0; i < cadena.length(); i++)
                System.out.println(" carater " + i + " es " + cadena.charAt(i));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

Length() y charAt(i) son métodos (funciones) que describiremos en la siguiente sección.

Observamos que `cadena.charAt(i)` es el elemento `i` de la cadena completa, por tanto podemos acceder a cada carácter al igual que los arreglos de tipo `char`. Utilizamos el método `charAt(i)` ya que en Java no se permite el uso de `cadena[i]` como en el caso de los arreglos de tipo `char`.



#### 6.4.4 Funciones propias para cadenas

Las bibliotecas nos aportan funciones para trabajar sobre las cadenas, y saber por ejemplos, la cantidad total de caracteres ingresados por un usuario, la cantidad de vocales que tiene una frase, entre otras que veremos a continuación.

- **Métodos (funciones) para Java:**

En Java existen métodos (o funciones) que permiten manipular cadenas. La tabla 6.3 presenta algunos ejemplos.

**Tabla 6.3:** Ejemplo de Métodos para manipulación de cadenas en Java

Funciones	Descripción
<b>c1.concat(c2)</b>	Concatena las cadenas c1 y c2
<b>c1.equals(c2)</b>	Devuelve true en caso de que las cadenas c1 y c2 son iguales
<b>c1.equalsIgnoreCase(c2)</b>	Devuelve true en caso de que las cadenas c1 y c2 son iguales, no importando si están escritas en minúsculas o mayúsculas
<b>c1.trim()</b>	Elimina los espacios del principio y fin de la cadena
<b>c1.length()</b>	Devuelve la cantidad de caracteres que hay en c1

Observemos los siguientes ejemplos que utilizan algunas de estos métodos.

**Ejemplo 6.12:**

El siguiente programa permite al usuario ingresar una cadena y luego la muestra al revés. Para ello utilizaremos el método **length()** que devuelve la cantidad de caracteres de una cadena, así sabremos cuál es la última posición de la misma.

```
import java.io.*;

public class MostrarCaracReves {
    public static void main (String [] args) {
        BufferedReader In =new BufferedReader (new InputStreamReader (System.in));
        String cadena;
        int i;

        try{
            System.out.print(" Ingrese una cadena:");
            cadena = In.readLine();
            for (i=cadena.length()-1; i>=0; i--)
                System.out.print(cadena.charAt(i));
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

**Ejemplo 6.13:**

El próximo programa compara dos cadenas ingresadas por el usuario. Para ello, utilizaremos el método **equals()** que devuelve verdadero si dos cadenas son iguales.

```
import java.io.*;

public class comparaCadenas {
    public static void main (String [] args) {
        BufferedReader In =new BufferedReader (new InputStreamReader (System.in));
        String cadena1, cadena2;

        try{
            System.out.print(" Ingrese una cadena:");
            cadena1 = In.readLine();
            System.out.print(" Ingrese otra cadena:");
            cadena2 = In.readLine();
            if (cadena1.equals(cadena2))
                System.out.println("las cadenas son iguales");
            else
                System.out.println("las cadenas son distintas");
        }
        catch (Exception e){
            e.printStackTrace();
        }
    }
}
```

## 6.5 Ejercicios Resueltos

### Ejercicio 6.1:

Cree un programa que genere 20 números de Fibonacci. Para ello, debe almacenar los números en un arreglo.

Los números de Fibonacci se generan en forma sucesiva de acuerdo a la siguiente regla:

Fibo(1) = 1

Fibo(2) = 1

Fibo(3) = Fibo(1) + Fibo(2)

Fibo(4) = Fibo(2) + Fibo(3)

.....

Fibo(n) = Fibo(n-2) + Fibo(n-1)

Por lo tanto, necesitamos un arreglo de 20 celdas de tipo entero para almacenar dichos números. Las dos primeras celdas se inicializan en 1, y luego en las otras celdas (celda i) se almacena la suma de los números ubicados en las dos celdas anteriores, es decir, la celda i-1 y la celda i-2.

Lenguaje Java:

```
import java.io.*;

public class Fibonacci {
    public static void main (String [] args) {
        int fibonacci[] = new int[20];
        int i;

        // primero inicializamos las dos primeras celdas con 1
        fibonacci[0] = 1;
        fibonacci[1] = 1;

        // luego le asignamos los números a las otras celdas que quedan
        for (i=2; i<20; i++)
            fibonacci[i] = fibonacci[i-2] + fibonacci[i-1];
        // finalmente mostramos los números por pantalla
        for (i=2; i<20; i++)
            System.out.println(" Fibonacci  " + i + " es " + fibonacci[i]);
    }
}
```



### Ejercicio 6.2:

Una persona posee una cuenta de ahorro en el banco “Ahorra Feliz”. Para mantener información detallada de su cuenta le solicita a usted crear un programa en Java que permita por medio de **un arreglo**:

- Ingresar los abonos mensuales a la cuenta de ahorro durante el año.
- Mostrar el mes en que se ingresó el mayor abono a la cuenta.
- Mostrar el promedio de su cuenta de ahorro en el año.
- Mostrar la varianza de la cuenta de ahorro.
- Mostrar la desviación estándar de su cuenta de ahorro anual.

Algunos antecedentes para calcular el **Promedio, Varianza y Desviación estándar**:

Esta medida nos permite determinar el promedio aritmético de fluctuación de los datos respecto a su punto central o media. La desviación estándar nos da como resultado un valor numérico que representa el promedio de diferencia que hay entre los datos y la media. Para calcular la desviación estándar basta con hallar la raíz cuadrada de la varianza, por lo tanto su ecuación sería:  $S = \sqrt{S^2}$

Para comprender el concepto de las medidas de distribución vamos a suponer que la persona que solicita el programa desea saber qué tanto varían los abonos mensuales en su cuenta de ahorro. Como ejemplo, supongamos que los abonos en dólares para los primeros 5 meses son los que se muestran en la siguiente fórmula que calcula el promedio:

$$\bar{X} = \frac{490 + 500 + 510 + 515 + 520}{5} = \frac{2535}{5} = 507$$

La varianza sería:

$$S_X^2 = \frac{\sum_{i=1}^n (X_i - \bar{X})^2}{n - 1}$$

$$S^2 = \frac{(490 - 507)^2 + (500 - 507)^2 + (510 - 507)^2 + (515 - 507)^2 + (520 - 507)^2}{(5 - 1)}$$

$$S^2 = \frac{(-17)^2 + (-7)^2 + (3)^2 + (8)^2 + (13)^2}{4} = \frac{289 + 49 + 9 + 64 + 169}{4} = \frac{580}{4} = 145$$

Por lo tanto la desviación estándar de los 5 primeros abonos sería:  $S = \sqrt{145} = 12.04 \cong 12$

Con lo que concluiríamos que el abono promedio de los 5 primeros meses es de 507 dólares, con una tendencia a variar por debajo o por encima de dicho abono en 12 dólares.

Una solución general del problema sería el siguiente:

```
import java.io.*;
import java.util.Scanner;

public class Banco {
    public static void main(String[] args) {
        int N = 12;
        int[] cuenta = new int[N];
        float prom, vari;
        int i, mayor, mes, suma;
        Scanner intro = new Scanner(System.in);

        // primero ingresamos el ahorro por teclado
        for (i=0; i<N; i++){
            System.out.println("Ingrese abono para el mes " + (i+1));
            cuenta[i] = intro.nextInt();
        }

        // luego podemos encontrar el mes en que se ingresó el mayor ahorro
        mayor = cuenta[0];
        mes=0;

        for (i=1; i<N; i++){
            if (cuenta[i] > mayor){
                mayor = cuenta[i];
                mes = i;
            }
        }
        System.out.println("el mes en que se ingresó el mayor ahorro es " + mes);

        //luego calculamos el promedio
        suma = 0;
        for (i=1; i<N; i++)
            suma = suma + cuenta[i];

        prom = (float) suma /N;
        System.out.println("el promedio de dinero en el cuenta de ahorro para el año es " + prom);

        //calculamos la varianza
        for (i=0; i<N; i++)
            suma = suma + (cuenta[i]-(int) prom)* (cuenta[i]-(int) prom);
        //o bien suma = suma + (cuenta[i]-prom)* (cuenta[i]-prom)
        vari= (float) suma /N-1;
        System.out.println("la varianza es " + vari);

        //finalmente la desviación estándar
        System.out.println("y la desviación estándar es " + Math.sqrt(vari));
    }
}
```

**Ejercicio 6.3:**

Una empresa de servicios desea almacenar la cantidad de reclamos que realizan los clientes durante una semana. Para ello dispone de una tabla de valores como la que sigue:

Ejemplo de tabla de reclamos:

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
10:00-13:00	4	6	2	34			
13:00-15:00	–	10		7			
15:00-20:00			29				

La cantidad de reclamos son almacenados por día de la semana de acuerdo a la jornada laboral. Por ejemplo, para el día martes en la jornada laboral de 10:00 a 13:00 hrs se registraron 6 reclamos. Con estos datos, el gerente general requiere que cree un programa que pueda realizar las siguientes actividades:

- Ingresar la cantidad de reclamos por día y jornada laboral.
- Mostrar el promedio de reclamos para la jornada laboral de 15:00 a 20:00 hrs.
- Mostrar el día de la semana que tuvo más reclamos.

Una solución en lenguaje Java sería la siguiente:

```
import java.io.*;
import java.util.Scanner;

public class Reclamos {

    public static void main(String[] args) {
        final int jornadas = 3;
        final int dias = 7;
        int[][] reclamos = new int[jornadas][dias];
        int i, j;
        int suma = 0;
        int mayor = 0;
        int dia = 0;
        Scanner intro = new Scanner(System.in);

        // primero ingresamos los reclamos por teclado
        for (j=0; j < dias; j++){
            System.out.println("Ingrese los reclamos para el día " + (int)(j+1));
            for (i=1; i< jornadas; i++){
                System.out.println("Ingrese el reclamo para la jornada " + (int)(i+1));
                reclamos[i][j] = intro.nextInt();
            }
        }
    }
}
```

```

// luego calculamos el promedio de reclamos para la jornada de 15:00 a 20:00 hrs.
// es decir la jornada número 2
suma = 0;
for (i=0; i< dias; i++)
    suma =suma + reclamos[2][i]; //jornada =2
System.out.println("El promedio de reclamos para la jornada de 15:00 a 20:00 hrs"
    + "es: " + (float) (suma/dias));

//finalmente calculamos el día de la semana que tuvo más reclamos
for (j= 1; j <dias; j++){
    suma =0;
    for (i= 0; i< jornadas; i++)
        suma= suma + reclamos[i][j];
    if (mayor < suma){
        mayor = suma;
        dia = j;
    }
}
System.out.println("El dia de la semana que tuvo más reclamos fue " + dia);
}
}

```



## 6.6 Ejercicios Propuestos

### Ejercicio 6.4:

Cree un programa, que permita a un usuario realizar las siguientes acciones con una matriz de 10 x 10

- a). Ingresar sólo números positivos entre 0 y 250
- b). Calcular el promedio de los números para una fila que ingresa el usuario
- c). Sumar los números de la diagonal.

### Ejercicio 6.5:

Un banco registra el saldo mensual de la cuenta corriente de un cliente en un arreglo. Se pide crear un programa para realizar las siguientes acciones:

- a) Ingresar los saldos por mes
- b) Mostrar los meses en que el saldo es negativo
- c) Mostrar el mes en que se tiene el mayor saldo
- d) Calcular un promedio de saldos para el año

### Ejercicio 6.6:

En un arreglo de 30 celdas se almacena las ventas diarias logradas por un vendedor de artículos de cocina durante el mes de abril. La empresa requiere un programa en Java que permita responder a los siguientes requerimientos por medio de funciones:

- a) Ingresar las ventas diarias del vendedor
- b) Mostrar el día del mes en que logró la mayor venta
- c) Calcular el total de ventas del mes
- d) Mostrar los días del mes en el que se logró ventas inferiores a los \$10.000
- e) Calcular la remuneración del empleado para el mes de abril si se sabe que se le paga una comisión del 1% de las ventas totales logradas.

**Ejercicio 6.7:**

Se ingresan a un arreglo las ventas diarias (total \$ de cada día) que realiza un vendedor de una tienda de CDs, se pide crear un programa que permita:

- Ingresar las ventas diarias del vendedor realizadas durante un mes (30 días) al arreglo. (Nota: recuerde validar que los números ingresados sean mayores o iguales a cero)
- Calcular el promedio de ventas logradas durante los primeros 15 días del mes.
- Mostrar las ventas diarias mayores a \$345.000
- Mostrar el día en que logró la mayor venta.

**Ejercicio 6.8:**

Un método clásico para identificar los números primos existentes en una secuencia de números que va desde 2 a N es la llamada Criba de Eratóstenes.

El algoritmo usado para este propósito va marcando (eliminando) todos los múltiplos de 2, 3, 4, 5 y así sucesivamente hasta que se encuentre el primer número no marcado que es mayor que la raíz cuadrada de N.

Ejemplo: suponga una secuencia de números desde 2 a 20.

- Se toma el número 2, dado que es el primer número no marcado.
- Marcar todos los múltiplos de 2 a partir de  $2^2$ .
- Se toma al 3, dado que es el siguiente número no marcado.
- Marcar todos los múltiplos de 3 a partir de  $3^2$ .
- Se toma el número 5 como el siguiente número. Pero  $5^2$  es mayor que 20. Entonces el algoritmo aquí se detiene y todos los números no marcados son los números primos.

Gráficamente : (E significa eliminado (marcado) )

a)



2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

b)

2	3	E	5	E	7	E	9	E	11	E	13	E	15	E	17	E	19	E
---	---	---	---	---	---	---	---	---	----	---	----	---	----	---	----	---	----	---

c)



2	3	E	5	E	7	E	9	E	11	E	13	E	15	E	17	E	19	E
---	---	---	---	---	---	---	---	---	----	---	----	---	----	---	----	---	----	---



d)

2	3	E	5	E	7	E	E	E	11	E	13	E	E	E	17	E	19	E
---	---	---	---	---	---	---	---	---	----	---	----	---	---	---	----	---	----	---

e)

2	3	E	5	E	7	E	E	E	11	E	13	E	E	E	17	E	19	E
---	---	---	---	---	---	---	---	---	----	---	----	---	---	---	----	---	----	---

Escriba un programa que permita solucionar el problema, para una secuencia que va desde 2 hasta cualquier N.

**Ejercicio 6.9:**

Dado un arreglo llamado PROM, mantiene los promedios (valores reales), de un curso que posee N alumnos, escriba un programa que entregue:

- a) El promedio de las notas
- b) El mayor y el menor promedio
- c) La cantidad de promedios en [4.5 - 6.0]

**Ejercicio 6.10:**

Dado un mensaje se debe calcular su costo para enviarlo por telégrafo. Para esto se sabe que las letras cuestan, cada una, \$10. Los caracteres especiales que no sean letras cuestan \$30 y los dígitos tienen un valor de \$20 cada uno. Los espacios no tienen valor.

Restricciones:

- El mensaje es una cadena
- Las letras ñ, á, é, í, ó, ú se consideran caracteres especiales.

Un ejemplo de ejecución del programa es:

**Entrada:** Feliz cumpleaños

**Salida:** Valor del mensaje \$ 170

Crear un programa que permita calcular el costo del mensaje dado un texto ingresado por el usuario.



## 6.7 Comentarios Finales

En este capítulo te hemos presentado los fundamentos de la definición y uso de arreglos en Java.

En este capítulo, los arreglos los hemos clasificado en arreglos unidimensionales, arreglos multidimensionales y cadenas. Para cada caso, hemos descrito las principales características con ejemplos básicos que te ayudan a comprender de mejor manera estos elementos.

Finalmente, te hemos entregado un set de ejercicios resueltos con el fin de reforzar el desarrollo de programas que utilizan arreglos. También hemos incorporado un set de ejercicios propuestos con el fin de que practiques y comentes las dudas con tu profesor y compañeros.

