

ULP
Virtual

Tecnicatura Universitaria en Desarrollo de Software

Programación II

Guía 5.1

Arreglos



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS

ARREGLOS: VECTORES Y MATRICES

Un arreglo es un contenedor de objetos que tiene un número fijo de valores del mismo tipo. El tamaño del arreglo es establecido cuando el arreglo es creado y luego de su creación su tamaño es fijo, esto significa que no puede cambiar. Por eso decimos que un arreglo es una colección homogénea, finita e inmutable de datos. Cada una de los espacios de un arreglo es llamada elemento y cada elemento puede ser accedido por un índice numérico que arranca desde 0 hasta el tamaño menos uno. Por ejemplo, si tenemos un vector de 5 elementos mis índices serian: 0-1-2-3-4

Al igual que la declaración de otros tipos de variables, la declaración de un arreglo tiene dos componentes: el tipo de dato del arreglo y su identificador. El tipo de dato del arreglo se escribe como **tipo[]**, en donde, tipo es el tipo de dato de cada elemento contenido en él. Los arreglos pueden albergar tanto tipos primitivos como complejos. Los corchetes sirven para indicar que esa variable va a ser un arreglo de una dimensión. El tamaño del arreglo no es parte de su tipo (es por esto que los corchetes están vacíos).

Una vez declarado un arreglo hay que crearlo/dimensionarlo, es decir, hay que asignar al arreglo un tamaño para almacenar los valores. La creación de un arreglo se hace con el operador *new*. Recordemos que las matrices son bidimensionales por lo que tienen dos tamaños, uno para las filas y otro para las columnas de la matriz.

Declaración y creación de un Vector

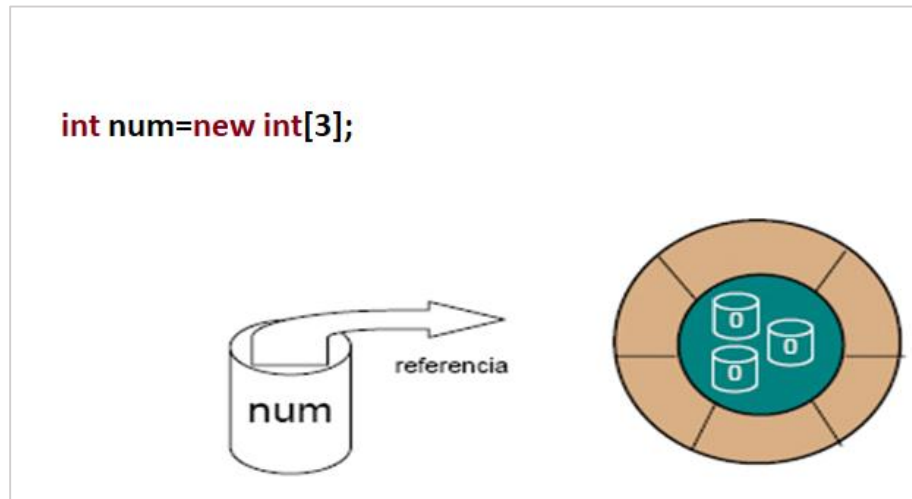
```
tipo[] arregloV = new tipo[Tamaño];
```

Declaración y creación de una Matriz

```
tipo[][] arregloM = new tipo[Filas][Columnas];
```

Una vez hayamos creado un arreglo, todas sus posiciones son inicializadas al valor por defecto del tipo de variable que albergue. Es decir, 0 o 0.0 si se trata de un número, false si se trata de un boolean y null si se trata de un tipo complejo.

Por ejemplo:



Existe una forma de crear un arreglo inicializando todas sus posiciones a un valor determinado, igualándolo a un listado de elementos separados por comas entre {}. El tamaño del arreglo será el número de elementos en el listado.

Ejemplos:

```
private int[] numeros = {1,2,3,4,5};  
private String[] cadenas = {"hola","adios"};  
private Integer[] ints = {new Integer(12), new Integer(98)};
```

Arreglo Bidimensional

```
int matriz[][]= {{1,2},{3,4},{5,6}};
```

matriz



1	2
3	4
5	6

ASIGNAR ELEMENTOS A UN ARREGLO

Cuando queremos ingresar un elemento en nuestro arreglo vamos a tener que elegir el subíndice en el que lo queremos guardar, es decir, en cual posición. Una vez que tenemos el subíndice

decidido tenemos que invocar nuestro vector por su nombre y entre corchetes el subíndice en el que lo queremos guardar.

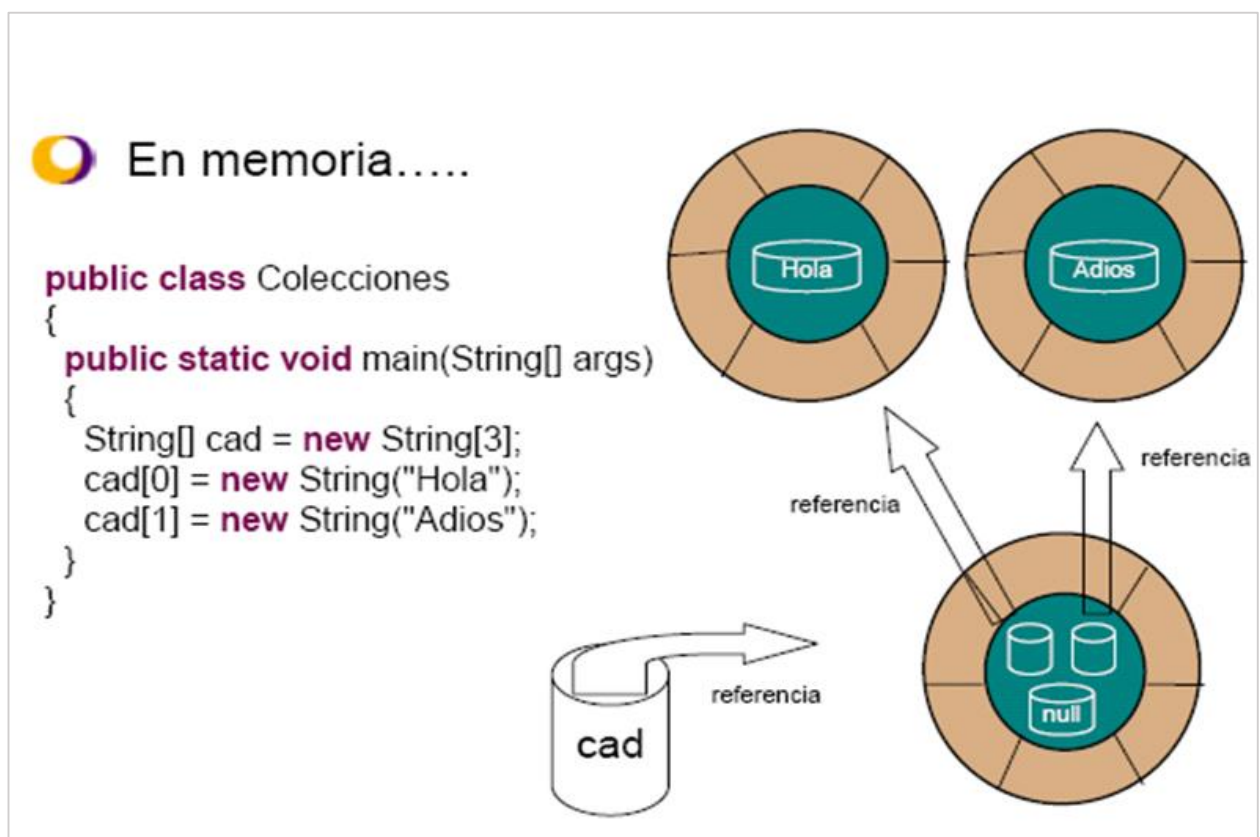
Después, pondremos el signo de igual (que es el operador de asignación) seguido del elemento a guardar. En las matrices vamos a necesitar dos subíndices y dos corchetes para representar la posición de la fila y la columna donde queremos guardar el elemento.

Asignación de un Vector

```
vector[0] = 5;
```

Asignación de una Matriz

```
matriz[0][0] = 6;
```



Esta forma de asignación implica asignar todos los valores de nuestro arreglo de uno en uno, esto va a conllevar un trabajo bastante grande dependiendo del tamaño de nuestro arreglo.

Entonces, para poder asignar varios valores a nuestro arreglo y no hacerlo de uno en uno usamos un bucle Para. El bucle Para, al poder asignarle un valor inicial y un valor final a una variable, podemos adaptarlo fácilmente a nuestros arreglos. Ya que, pondríamos el valor inicial de nuestro arreglo y su valor final en las respectivas partes del Para. Nosotros, usaríamos la variable creada en el Para, y la pasaríamos a nuestro arreglo para representar todos los subíndices del arreglo, de esa manera, recorriendo todas las posiciones y asignándole a cada posición un elemento.

Para poder asignar varios elementos a nuestra matriz, usaríamos dos bucles Para anidados., ya que un Para recorrerá las filas (variable i) y otro las columnas (variable j).

Asignación de un Vector

En este ejemplo, llenaremos las posiciones del arreglo con números enteros ingresados por teclado.

```
Scanner leer=new Scanner(System.in);  
for (int i = 0; i < 5; i++) {  
    System.out.println("Ingrese un nro");  
    vector[i] =leer.nextInt();  
}
```

Asignación de una Matriz

```
Scanner leer=new Scanner(System.in);  
for (int i = 0; i < 3; i++) {  
    for (int j = 0; j < 3; j++) {  
        System.out.println("Ingrese un nro");  
        matriz[i][j] = leer.nextInt();  
    }  
}
```

Para conocer la longitud de un arreglo, podremos acceder a su atributo público: *length*.

Un arreglo al no ser dinámico, no podremos: ni eliminar posiciones, ni insertar posiciones. El borrado será lógico, como igualar las posiciones a null, a -1, etc. Dependerá del desarrollador.

Ejemplo

```
public class Colecciones
{
    // Recorrido.
    for(int i=0; i<saludos.length; i++)
    public static void main(String[] args)
    {
        System.out.println(saludos[i]);

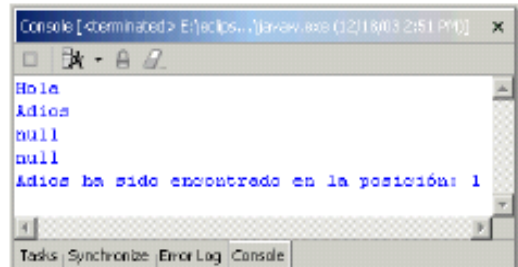
        // Creación e inicialización.
        String[] saludos = new String[4];

        // Búsqueda.
        boolean sw = false;
        for(int i=0; i<saludos.length; i++)
        {
            // Inserción.
            saludos[0] = new String("Hola");
            saludos[1] = new String("Adios");
            saludos[2] = new String("Hello");
            saludos[3] = new String("GoodBye");

            if(saludos[i] != null && saludos[i].equals("Adios"))
            {
                System.out.println("Adios ha sido
                    encontrado en la posición: " + i);
                sw = true;
                break;
            }
        }

        // Extracción.
        String tmp = saludos[2];

        // Borrado.
        saludos[2] = null;
        saludos[3] = null;
    }
}
```



Sentencia for/in:

Básicamente se trata de una simplificación a la hora de codificar. Es decir, al final, el compilador convierte el código en una sentencia **for** convencional.

Sintaxis:

```
for (tipo variable:coleccion) {  
    sentencias;  
}
```

Por ejemplo:

```
int lista[] = {24,73,95,16,78};  
for (int nro:lista){  
    System.out.println(nro);  
}
```

Equivale a:

```
int lista[] = {24,73,95,16,78};  
for (int i=0; i < lista.length; i++){  
    System.out.println(lista[i]);  
}
```

VECTORES Y MATRICES EN MÉTODOS:

Los arreglos se pueden pasar como parámetros a un método (función o procedimiento) del mismo modo que pasamos variables, poniendo el tipo de dato delante del nombre del vector o matriz, pero deberemos sumarle los corchetes para representar que es un vector o matriz. Sin embargo, hay que tener en cuenta que la diferencia entre los arreglos y las variables, es que los arreglos siempre se pasan por referencia ya que Java los maneja como objetos.

```
public static void llenarVector(int[] vector){  
}  
  
public static void mostrarMatriz(int[][] matriz){  
}
```

A diferencia de Pseint, en Java si podemos devolver un vector o una matriz en método para usarla en otro momento. Lo que hacemos es poner como tipo de dato de retorno del método, el tipo de dato que tendrá el vector y así poder devolverlo.

```

public static int[] devolverVector(){

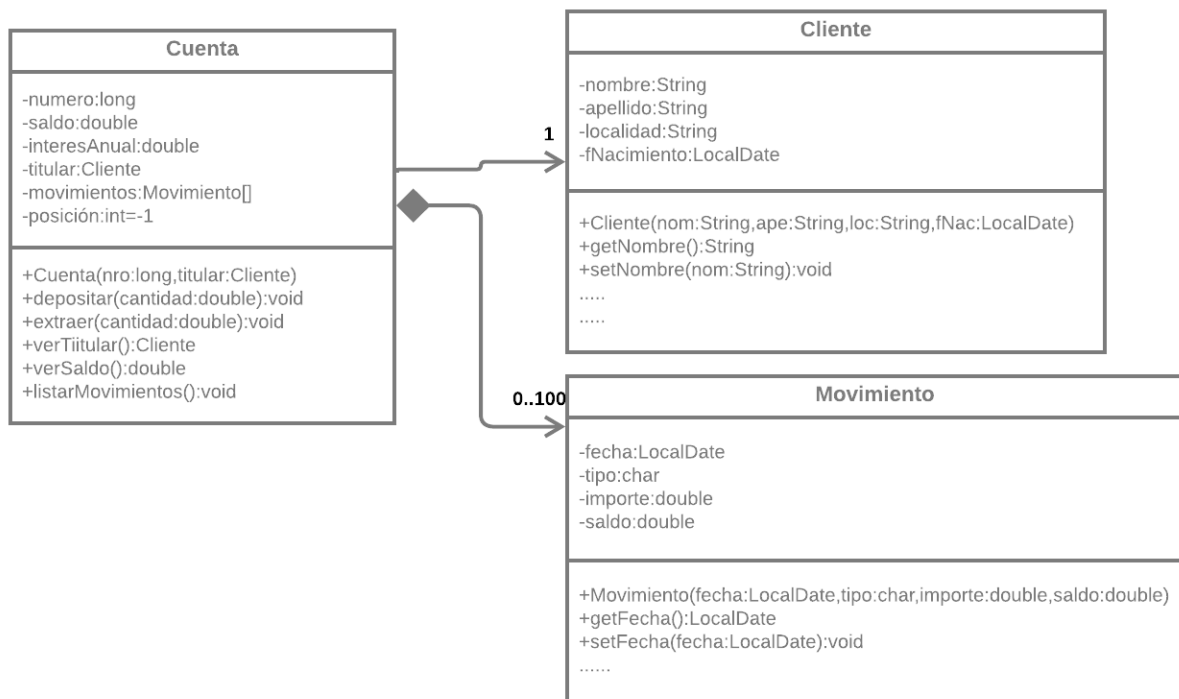
int[] vector = new int[5];

return vector; }

```

Veamos esto en un ejemplo completo:

Utilizaremos para este ejemplo el modelo de Cuenta, Movimiento y Cliente que utilizamos en la guía de relaciones entre clases, pero con algunos cambios.



Un banco de la ciudad, nos pide que desarrollemos una aplicación que permita registrar una Cuenta con su Cliente titular y todos los Movimientos que se realicen en dicha Cuenta como: extracciones y depósitos. Para ello hemos realizado el siguiente modelo en el que tenemos una clase Cuenta con los atributos: numero, saldo, interesAnual, el Cliente titular de la Cuenta, una lista de Movimientos sobre ella, supondremos que una Cuenta puede registrar hasta 100 movimientos y una variable atributo posición que nos servirá como índice en el arreglo de movimientos; como verán, está inicializado en -1 indicando que al crearse una Cuenta no hay movimientos. La clase Cuenta además posee un constructor que permite instanciar una Cuenta con su número y titular y una serie de métodos especiales que se detallan a continuación:

- **depositar(cantidad)** → Recibe la cantidad de dinero que incrementará el saldo de la cuenta y registra el movimiento “D”(Depósito) en el arreglo en la posición +1.
- **extraer(cantidad)** → Recibe la cantidad de dinero a extraer que disminuirá el saldo de la cuenta y registra el movimiento “E” (Extracción) en el arreglo en la posición +1. Ojo!!! Solo extraerá dinero si hay saldo suficiente.
- **verTitular():Cliente**→Retornará el Cliente titular de la cuenta

- **verSaldo():double**→Retornará el saldo actual de la cuenta.
- **listarMovimientos():void**→Mostrará por consola el detalle de todos los movimientos.

C:/Users/Usuario/Documents/NetBeansProjects/EjemploCompleto/src/ejemplocompleto/Cliente.java

```
1 package ejemplocompleto;
2 import java.time.LocalDate;
3
4 public class Cliente {
5
6     private String nombre;
7     private String apellido;
8     private String localidad;
9     private LocalDate fNacimiento;
10
11     public Cliente(String nombre, String apellido, String localidad,
12         LocalDate fNacimiento) {
13         this.nombre = nombre;
14         this.apellido = apellido;
15         this.localidad = localidad;
16         this.fNacimiento = fNacimiento;
17     }
18
19     public String getNombre() {
20         return nombre;
21     }
22
23     public void setNombre(String nombre) {
24         this.nombre = nombre;
25     }
26
27     public String getApellido() {
28         return apellido;
29     }
30
31     public void setApellido(String apellido) {
32         this.apellido = apellido;
33     }
34
35     public String getLocalidad() {
36         return localidad;
37     }
38
39     public void setLocalidad(String localidad) {
40         this.localidad = localidad;
41     }
42     public LocalDate getfNacimiento() {
43         return fNacimiento;
44     }
45     public void setfNacimiento(LocalDate fNacimiento) {
46         this.fNacimiento = fNacimiento;
47     }
48 }
```

```
1 package ejemplocompleto;
2 import java.time.LocalDate;
3
4 public class Movimiento {
5     private LocalDate fecha;
6     private char tipo;
7     private double importe;
8     private double saldo;
9
10    public Movimiento(LocalDate fecha, char tipo, double importe, double saldo){
11        this.fecha = fecha;
12        this.tipo = tipo;
13        this.importe = importe;
14        this.saldo = saldo;
15    }
16
17    public LocalDate getFecha() {
18        return fecha;
19    }
20
21    public void setFecha(LocalDate fecha) {
22        this.fecha = fecha;
23    }
24
25    public char getTipo() {
26        return tipo;
27    }
28
29    public void setTipo(char tipo) {
30        this.tipo = tipo;
31    }
32
33    public double getImporte() {
34        return importe;
35    }
36
37    public void setImporte(double importe) {
38        this.importe = importe;
39    }
40
41    public double getSaldo() {
42        return saldo;
43    }
44    public void setSaldo(double saldo) {
45        this.saldo = saldo;
46    }
47 }
```

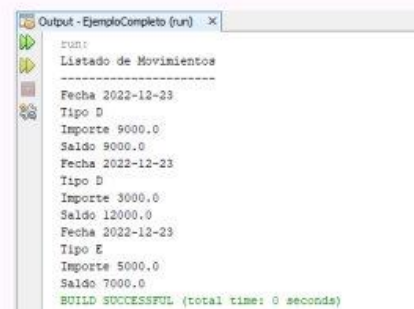
```
1 package ejemplocompleto;
2 import java.time.LocalDate;
3
4 public class Cuenta {
5     private long numero;
6     private double saldo;
7     private double interesAnual;
8     private Cliente titular;
9     private Movimiento movimientos[];
10    private int posicion = -1;
11
12    public Cuenta(long numero, Cliente titular) {
13        this.movimientos = new Movimiento[100];
14        this.numero = numero;
15        this.titular = titular;
16    }
17    //-----
18    public void depositar(double cantidad) {
19
20        this.saldo+=cantidad;
21        this.posicion++;
22        if(posicion < 100){
23            this.movimientos[posicion]=new Movimiento(LocalDate.now(), 'D', cantidad, this.saldo);
24        } else {
25            System.out.println("No se pueden registrar más movimientos");
26        }
27    }
28    //-----
29    public void extraer(double cantidad) {
30
31        this.saldo-=cantidad;
32        if(this.saldo >= cantidad){
33
34            this.posicion++;
35            if(this.posicion < 100){
36
37                this.movimientos[posicion]=new Movimiento(LocalDate.now(), 'E', cantidad, this.saldo);
38            } else {
39
40                System.out.println("No se pueden realizar más movimientos");
41            }
42        } else {
43
44            System.out.println("No hay saldo suficiente");
45        }
46    }
47
48
49
50    }
51
52
53
```

```
54     public Cliente verTitular(){
55
56         return this.titular;
57     }
58     public double verSaldo(){
59
60         return this.saldo;
61     }
62
63     public void listarMovimientos(){
64
65         System.out.println("Listado de Movimientos ");
66         System.out.println("-----");
67         //El arreglo tiene 100 posiciones,
68         //pero puede que no todas tengan movimientos
69
70         for(Movimiento movi:movimientos){
71
72             if(movi !=null){
73                 System.out.println("Fecha "+movi.getFecha());
74                 System.out.println("Tipo "+movi.getTipo());
75                 System.out.println("Importe "+movi.getImporte());
76                 System.out.println("Saldo "+movi.getSaldo());
77             }
78         }
79     }
80 }
```

Ahora desde el método main de la clase principal del proyecto, crearemos una Cuenta obviamente con saldo 0 (cero) para un Cliente Juan López, nacido el 20 de abril de 1976, en la ciudad de San Luis; luego haremos un depósito de 9000 pesos en dicha Cuenta, luego otro depósito por 3000 pesos; luego una extracción de 5000 peso y por último pediremos el listado de movimientos.

C:/Users/Usuario/Documents/NetBeansProjects/EjemploCompleto/src/ejemplocompleto/EjemploCompleto.java

```
1
2 package ejemplocompleto;
3
4 import java.time.LocalDate;
5 import java.time.Month;
6 import java.time.temporal.ChronoUnit;
7 import java.util.*;
8
9
10 public class EjemploCompleto {
11
12
13     public static void main(String[] args) {
14
15         //Crearemos un cliente
16         Cliente juan=new Cliente("Juan","López","San Luis",
17             LocalDate.of(1976, Month.APRIL, 20));
18
19         Cuenta cuental=new Cuenta(101,juan);
20
21         cuental.depositar(9000);
22         cuental.depositar(3000);
23         cuental.extraer(5000);
24
25         cuental.listarMovimientos();
26
27
28
29
30
31
32     }
33
34 }
```



Output - EjemploCompleto (run) x

```
run:
Listado de Movimientos
-----
Fecha 2022-12-23
Tipo D
Importe 9000.0
Saldo 9000.0
Fecha 2022-12-23
Tipo D
Importe 3000.0
Saldo 12000.0
Fecha 2022-12-23
Tipo E
Importe 5000.0
Saldo 7000.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

BIBLIOGRAFIA:

Christopher Alexander, “A Pattern Language”, 1978

Erich Gamma, “Design Patterns: Elements of Reusable OO Software”

Martin Fowler, “Analysis Patterns: Reusable Object Models”, Addison Wesley,
1997

Kathy Sierra, “OCA/OCP JAVA SE 7 PROGRAMMER I & II STUDY GUIDE (EXAMS 1Z0-803 & 1Z0-8)”, McGraw Hill, 2014