

AGRUPACIÓN DE OBJETOS

Salvador López Mendoza

Octubre de 2022

ORDENAR NÚMEROS

PROBLEMA

Dados **tres** números encontrar un algoritmo que los ordene de menor a mayor.

¿Se puede usar el mismo algoritmo para ordenar **cinco** números?

¿Qué modificaciones se deben hacer al algoritmo para que pueda ordenar **cualquier cantidad** de números?

SECCIÓN ESCOLAR

PROBLEMA

En una institución educativa se requiere de un programa que ayude con el trabajo de atención a alumnos. Se necesita guardar y proporcionar toda la información relacionada con los alumnos de la institución. El programa debe ayudar a obtener y manejar datos acerca de los alumnos.

Los datos que debe tener, o en su caso calcular, son los datos personales del alumno, sus calificaciones, su promedio, la mayor calificación obtenida, los cursos en los que su calificación ha sido 10 y determinar el aprovechamiento de un alumno con respecto a su grupo.

METODOLOGÍA DE DISEÑO. OBJETOS

CANDIDATOS: Institución educativa, programa, trabajo, sección escolar, información, alumnos, datos, datos personales, calificaciones, promedio, cursos, grupo.

SELECCIONADOS: **Alumno, sección escolar**

METODOLOGÍA DE DISEÑO. COMPORTAMIENTO

ALUMNO: Guardar información, proporcionar información, obtener datos, calcular promedio, obtener calificación más alta, obtener cursos en los que su calificación es 10, determinar aprovechamiento respecto al grupo.

SECCIÓN ESCOLAR: Registrar la información de los alumnos, proporcionar información de los alumnos.

METODOLOGÍA DE DISEÑO. ESCENARIOS

Escenario principal:

- 1 Se muestra menú de opciones.
- 2 Usuario selecciona opción.
- 3 Dependiendo de la opción seleccionada, solicita más información.

Alumno
Nombre Dirección Carrera Calificaciones

LA CLASE ALUMNO

```
/**
 * Clase Alumno. Define alumnos de una institucion educativa
 *
 */
public class Alumno{
    private String nombre ;
    private String direccion ;
    private String carrera ;

    private int calif01, calif02, ..., calif36 ;
}
```


LA CLASE ALUMNO. MÉTODO PROMEDIO

```
/**  
 * Calcula el promedio de las calificaciones obtenidas  
 */  
public double promedio() {  
    return (calif01 + calif02 + ... + calif36) / 36 ;  
}
```

LA CLASE ALUMNO. MÉTODOS MODIFICADORES

```
public void asignarCal01(int calificacion) {  
    calif01 = calificación ;  
}
```

```
public void asignarCal02(int calificacion) {  
    calif02 = calificación ;  
}
```

...

```
public void asignarCal36(int calificacion) {  
    calif36 = calificación ;  
}
```

AGRUPACIÓN DE OBJETOS

Es frecuente encontrar problemas que requieren varios objetos de la misma clase y a los que se da un uso similar.

Se presenta la forma de agrupar objetos, y en general datos de cualquier tipo, en un objeto denominado **arreglo**.

Una vez creada la agrupación se analiza la forma de trabajar con cada elemento de ella o con todos como unidad.

DEFINICIÓN

Para facilitar la declaración y manipulación de grupos de datos del mismo tipo y relacionados entre sí se tiene el concepto de arreglo.

DEFINICIÓN

Un **arreglo** es *un objeto* que agrupa una cantidad fija y predeterminada de elementos del mismo tipo.

A diferencia de cualquier otro objeto, en Java los arreglos no responden a métodos.

DECLARACIÓN

La declaración de un arreglo requiere crear una referencia a él y luego llamar al constructor de la clase.

Creación de referencia:

```
tipoDeDato [] nombreDeArreglo ;
```

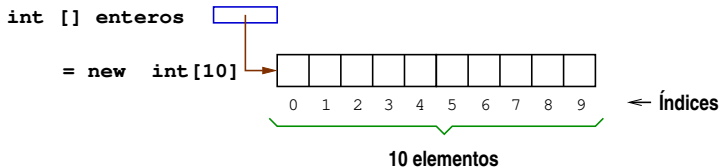
Llamada al constructor:

```
new tipoDeDato [tamaño];
```

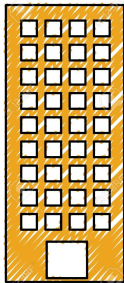
DECLARACIÓN (II)

Declaración y creación de un arreglo de diez enteros llamado **enteros**:

```
int[] enteros ;  
enteros = new int [10] ;
```



ARREGLOS. METÁFORA DEL EDIFICIO



Un arreglo se puede ver como un edificio de n pisos.

En cada piso hay gente que se encuentra ahí.

Para ir a un piso determinado, se dirige al elevador y se indica el piso deseado.

¿Cuántos pisos tiene el edificio?

¿Cómo se hace referencia a cada piso?

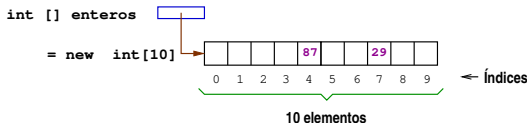
USO DE UN ARREGLO

Una vez definido y creado el arreglo se puede *usar cada elemento del mismo como una variable independiente*, del tipo especificado en la declaración del arreglo.

Para trabajar con cualquier dato del arreglo es necesario usar una variable o constante de tipo entero que representa la posición del elemento dentro del arreglo.

A este valor se denomina **índice**.

```
enteros[4] = 87 ;  
enteros[7] = enteros[4] / 3 ;
```



USO DE UN ARREGLO (II)

Los valores válidos para un índice son enteros entre 0, incluyéndolo, y uno menos que la cantidad de elementos.

Si n es el tamaño del arreglo

$$\text{índice} \in [0, n - 1]$$

El valor del índice puede ser el resultado de cualquier expresión que devuelva un entero en el rango permitido.

ADVERTENCIA (PARA PROGRAMAS EN JAVA)

El pretender acceder a un elemento del arreglo fuera de su rango permitido ocasiona la suspensión de la ejecución del programa con un mensaje de error.

TAMAÑO DEL ARREGLO

Al definir (crear) un arreglo, se establece la cantidad de elementos que tiene.

Un arreglo es una **estructura estática**, su tamaño no cambia.

En Java, la creación de un arreglo se define en dos etapas:

- Se define la referencia al arreglo.
- Se crea el arreglo y se establece la relación con la referencia.

La segunda etapa se puede realizar en cualquier momento.

Se puede aprovechar esta situación para definir la cantidad de elementos de un arreglo hasta el momento de ejecución.

TAMAÑO DEL ARREGLO (II)

En el caso más general, no se sabe de antemano cuántos elementos tiene un arreglo.

¿Cómo hacer programas que manejen un arreglo si no conocemos su tamaño?

Todo arreglo tiene en su estructura un **atributo público**. Es una constante y se llama **length**.

Debido a que es público se accede a él directamente (sin un método de por medio) para consultarse, más no para modificarse (por ser constante).

length indica la cantidad de elementos en el arreglo, pero no es un valor válido para un índice, el máximo valor para un índice es **length - 1**.

$$\text{indice} \in [0, \text{length} - 1]$$

MANEJO DE LOS ARREGLOS

En muchas ocasiones se necesita trabajar con todos los elementos de un arreglo, uno a la vez.

Para recorrer un arreglo se utiliza una instrucción de iteración.

- Lo más común es usar la instrucción **for**.
Generalmente se usa para recorrer todo el arreglo, desde el primer elemento hasta el último.
- Cuando solo se necesita usar un elemento del arreglo, se usa la instrucción **while** para buscarlo.
El recorrido del arreglo termina al encontrar al elemento deseado (o cuando se acabe el arreglo).

RECORRIDO DE UN ARREGLO

Asignar datos a un arreglo:

```
int n ;
Scanner in = new Scanner(System.in) ;

System.out.println("Escribe la cantidad de elementos que " +
                  "tiene el arreglo") ;
n = in.nextInt() ;

int [] numeros = new int [n] ; // n es valor entero positivo

for (int i = 0 ; i < n ; i++) {
    System.out.println("Escribe el valor del elemento " +
                      i + ":") ;
    numeros[i] = in.nextInt() ;
}
```

Recorre todo el arreglo. En cada lugar almacena un valor, el que haya indicado el usuario.

RECORRIDO DE UN ARREGLO (II)

Busca el primer elemento con valor negativo en los datos del arreglo:

```
int lugar = -1 ;    // Lugar que tiene el valor buscado
int indice = 0 ;    // Lugar en el que busca
boolean fin = false ; // Para saber si ha terminado
while (!fin) {      // No ha terminado
    if (numeros[indice++] < 0) {
        fin = true ;
        lugar = indice - 1; //lugar es la posición del elemento
    }
    if (indice == numeros.length) fin = true ;
} // Fin de while
if (lugar < 0) {
    System.out.println("No hay elementos con valor negativo");
} else {
    System.out.println("Hay un elemento con valor negativo," +
        " está en la posición " + lugar +
        " y tiene el valor " + numeros[lugar]);
}
```

MÉTODOS QUE DEVUELVEN MÁS DE UN VALOR

En **JAVA**, los métodos sólo devuelven un valor.

Son funciones del siguiente estilo: $f : A \times B \times C \times \dots \rightarrow X$

¿Cómo hacer para que devuelvan una colección de valores?

Por ejemplo, $f : A \times B \times C \rightarrow X \times X$.

Una solución: **devolver un arreglo**.

En el método se crea un arreglo local, con la cantidad de elementos que se quieran regresar, se llena con la información solicitada y se devuelve el arreglo completo.

Todos los datos que van en el arreglo que se regresa son del mismo tipo.

ARREGLOS COMO PARÁMETROS

Los arreglos pueden usarse como parámetros de los métodos, de la misma forma en que se usa cualquier otro tipo de dato.

La definición del parámetro formal se realiza de la misma manera en que se define un arreglo.

Al llamar al método que recibe un arreglo como parámetro sólo se escribe el nombre del arreglo, sin especificar los corchetes.

Dentro del método que recibe el arreglo como parámetro es posible modificar el contenido del arreglo.

Lo que se recibe es la referencia al arreglo, no el arreglo. A partir de la referencia se puede modificar el arreglo.

ARREGLOS DE OBJETOS

Los arreglos no están limitados a almacenar datos de tipos primitivos, en particular se puede tener un arreglo cuyos elementos sean **objetos** de cualquier clase.

La forma de declarar un arreglo de objetos es similar a la de cualquier otro arreglo:

```
Punto [] misPuntos ;  
misPuntos = new Punto[5] ;
```

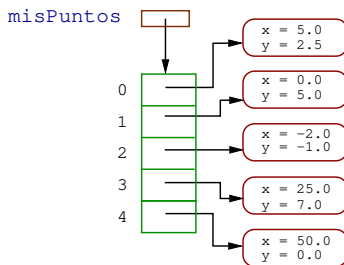
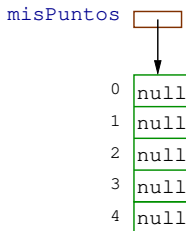
La semántica varía un poco con respecto a los arreglos de tipos primitivos.

Punto no es un tipo primitivo, se tiene un arreglo con espacio suficiente para cinco referencias, cada una a algún objeto.

No se define espacio para almacenar cinco objetos en el arreglo.

ARREGLOS DE OBJETOS (II)

```
Punto [] misPuntos ;  
misPuntos = new Punto[5] ;  
  
misPuntos[0] = new Punto(5.0, 2.5) ;  
misPuntos[1] = new Punto(0.0, 5.0) ;  
misPuntos[2] = new Punto(-2.0, -1.0) ;  
misPuntos[3] = new Punto(25.0, 7.0) ;  
misPuntos[4] = new Punto(50.0, 0.0) ;
```



ARREGLOS DE CADENAS DE CARACTERES

Los arreglos no están limitados a almacenar datos de tipos primitivos, en particular se puede tener un arreglo cuyos elementos sean cadenas (**Strings**).

La forma de declarar un arreglo de cadenas es similar a la de cualquier otro arreglo:

```
String[] diasHabiles = new String[5] ;
```

ARREGLOS DE CADENAS DE CARACTERES (II)

La semántica varía un poco con respecto a los arreglos de tipos primitivos.

String no es un tipo primitivo, se tiene un arreglo con espacio suficiente para cinco referencias, cada una a alguna cadena, no para cinco cadenas.

¿Qué pasa si después de declarar el arreglo **diasHabiles** se intenta lo siguiente?

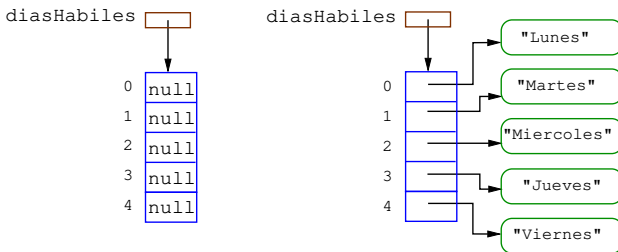
```
System.out.println("Primer dia es: " + diasHabiles[0]) ;
```

Se genera un error, pues no hay objetos asociados (referenciados) a cada elemento del arreglo.

Se debe crear un objeto (String) para cada elemento del arreglo.

ARREGLOS DE CADENAS DE CARACTERES (III)

```
String [] diasHabiles ;  
diasHabiles = new String[5] ;  
  
diasHabiles[0] = "Lunes" ;  
diasHabiles[1] = "Martes" ;  
diasHabiles[2] = "Miercoles" ;  
diasHabiles[3] = "Jueves" ;  
diasHabiles[4] = "Viernes" ;
```



PARÁMETRO DEL MÉTODO MAIN

Firma del método `main`:

```
public static void main(String [] param)
```

El método `main` tiene un parámetro:

Se llama `param` y es un **arreglo de String**.

¿Cómo usar ese parámetro?

De la misma forma que cualquier arreglo.

¿Qué cosas están en el parámetro de `main`?

Cadenas de caracteres que indiquemos al invocar el programa.

Generalmente se usan esas cadenas para enviar información al programa, por ejemplo opciones.

ARREGLOS DE VARIAS DIMENSIONES

Los arreglos representan una estructura lineal.

Ejemplo:

Un vector (x_1, x_2, \dots, x_n) se puede representar como un arreglo:

```
double [] x = new double[n] ;
```

Aquí a cada componente del vector le corresponde un elemento del arreglo.

ADVERTENCIA

En el vector $i \in [1, n]$; en el arreglo $i \in [0, n - 1]$.

No todos los problemas que necesitan de la agrupación de objetos son (conceptualmente) estructuras lineales.

ARREGLOS DE VARIAS DIMENSIONES (II)

MATRICES Estructuras de dos dimensiones, en muchas ocasiones representan sistemas de ecuaciones.

$$A_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

Es una estructura que tiene m renglones y n columnas.

```
double [][] matrizA = new double [m][n] ;
```

Para acceder al elemento que está en el renglón i , columna j :

```
System.out.println(matrizA[i][j]) ;
```

Con $i \in [0, m - 1]$ y $j \in [0, n - 1]$.

ARREGLOS DE VARIAS DIMENSIONES (III)

TABLEROS DE JUEGO Permiten describir la colocación de piezas mediante un sistema de coordenadas (renglón, columna).

X		
	O	

0,0	0,1	0,2
1,0	1,1	1,2
2,0	2,1	2,2

```
Ficha [][] tableroGato = new Ficha[3][3] ;  
    tableroGato[0][0] = new Ficha(X) ;  
    tableroGato[1][1] = new Ficha(O) ;
```

Es responsabilidad del programador convertir las expresiones comunes de los jugadores a las posiciones de la matriz.

Por ejemplo: *arriba a la derecha* corresponde a la posición *renglón 0, columna 2*.

ARREGLOS DE VARIAS DIMENSIONES (IV)

Definición general de un arreglo (matriz) de dos dimensiones:

```
tipoDato [][] arregloBiDimension = new tipoDato[m][n] ;
```

Lo que se define es un arreglo cuyos elementos son arreglos del tipo de dato especificado.

Al igual que en el caso de los arreglos de una dimensión, al crear matrices de objetos se está creando una estructura para contener referencias a esos objetos.

Hay que crear cada objeto y asignarlo a la referencia correspondiente para poder usar esos elementos.

ARREGLOS DE VARIAS DIMENSIONES (V)

Al definir un arreglo cuyos elementos son arreglos, se define un arreglo de referencias a arreglos. Al momento de crear el arreglo de dos dimensiones no se han creado los objetos (arreglos) a que se hace referencia en cada elemento.

Cada referencia a un arreglo puede ser a un arreglo de tamaño distinto.

Si tenemos arreglos cuyos elementos son arreglos, ¿podemos tener arreglos cuyos elementos son matrices?

¡Se tiene una estructura de tres dimensiones!

¿Cuál es el límite?

ORDENAR NÚMEROS

Diseñar un algoritmo que ordene n números enteros.

- Dados dos números enteros a y b :

```
Si ( $a < b$ )  
    entonces  $a$  es el primero y  $b$  el segundo;  
    en caso contrario  $b$  es el primero y  $a$  es el segundo.
```

- Dados tres números enteros a , b y c .

```
Si ( $a < b$ )  
    entonces si ( $a < c$ )  
        Entonces  $a$  es el primero;  
    ...
```

Todavía se puede expresar el algoritmo (y el programa que lo implemente) con relativa facilidad.

- Dados 87 números enteros ...

ORDENAR NÚMEROS (II)

Los datos que se quieren ordenar están en un arreglo.


La idea es intercambiar los elementos del arreglo de tal forma que, al finalizar el proceso, queden ordenados de acuerdo a su posición en el arreglo: el valor más pequeño será el primer elemento del arreglo, el segundo valor más pequeño será el segundo elemento del arreglo, y así sucesivamente.

PROPUESTA

Encontrar al valor más pequeño de todo el arreglo y colocarlo al principio del arreglo.

Tomar al resto de los elementos del arreglo y realizar el mismo proceso.

ORDENAR NÚMEROS (III)

0	125	 Posición
1	43	
2	2436	
3	876	
4	-2	
5	9876	
6	-543	
7	54	
8	432	
9	-765	

0	-765
1	125
2	2436
3	876
4	43
5	9876
6	-2
7	54
8	432
9	-543

ORDENAR NÚMEROS (IV)

0	-765		0	-765
1	125	← Posición	1	-543
2	2436		2	2436
3	876		3	876
4	43		4	125
5	9876		5	9876
6	-2		6	43
7	54		7	54
8	432		8	432
9	-543		9	-2

ORDENAR NÚMEROS. TIEMPO

Para el algoritmo anterior, se ha medido el tiempo que se tarda en ordenar las distintas cantidades de valores.

Se espera que el tiempo necesario para resolver el problema aumente conforme aumenta la cantidad de datos.

Cantidad	Tiempo	Porcentaje
10	0:00.10	
100	0:00.05	50 %
1,000	0:00.08	160 %
10,000	0:00.25	312.5 %
100,000	0:13.62	5448 %
1,000,000	21:24.6	9431.71 %

¿Cómo es el factor de crecimiento?