DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROGRAMACIÓN

TEMA 4

ESTRUCTURAS DE ALMACENAMIENTO BÁSICAS

CONTENIDOS

4.1. Arrays o vectores.

4.2. Arrays multidimensionales o matrices.

4.3. API de manejo de arrays.

4.4. Cadenas de caracteres.

4.5. Operaciones básicas.

4.6. Enumerados.

4.7. Argumentos en línea de comandos.

4.1. ARRAYS O VECTORES.

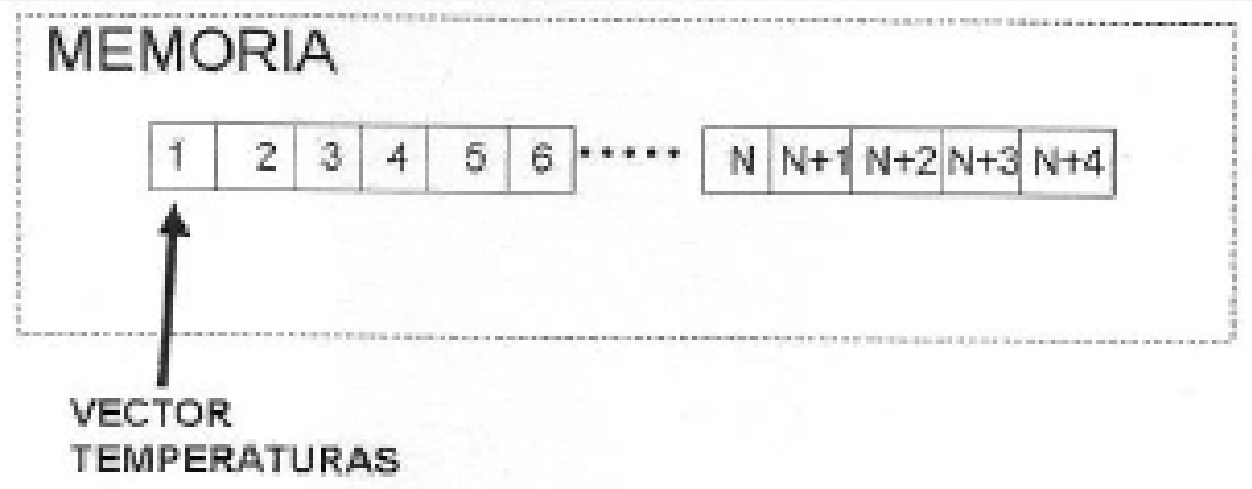
Hasta ahora, en los temas anteriores, no había una necesidad muy grande de almacenar muchos valores, se utilizaban variables para almacenar el color, la edad, la cantidad, etc.

Imaginemos que nos piden realizar un programa que almacene la temperatura de 100 ciudades y luego saque la temperatura media. No parece operativo tener 100 variables en nuestro programa, nada más que para escribir los nombres de las variables en el código, el número de líneas se dispara.

¿Y si nos dicen que se va a aumentar el número de ciudades a 200? La solución a esto es el uso de arrays o vectores (son sinónimos).

4.1. ARRAYS O VECTORES.

Un array se compone de una serie de posiciones consecutivas en memoria:



A los vectores se accede mediante un subíndice, si por ejemplo nuestro vector anterior se llama “temperaturas” y se quiere acceder a la posición N, habrá que escribir “temperaturas[N]” en el programa para obtener la información de esa posición de memoria.

N puede ser una variable o bien un valor concreto (literal).

4.1. ARRAYS O VECTORES.

Declaración de vectores.

En Java se pueden declarar vectores de dos formas diferentes. Para declarar nuestro vector de “temperaturas” se puede realizar de las siguientes formas:

byte[] temperaturas; *(formato preferido Java)*
byte temperaturas[]; *(formato heredado C)*

En ningún momento se ha dado el tamaño de la matriz, lo único que se ha especificado es el tipo de los elementos que va a albergar dicha matriz. Es decir, “temperaturas” es una referencia a una estructura donde se pueden guardar muchos valores del tipo definido (*byte*).

No existe ninguna restricción al tipo base de un array, puede ser un tipo primitivo, un enumerado o cualquier clase.

4.1. ARRAYS O VECTORES.

Creación de vectores.

Java trata a los vectores como si fuesen objetos, por lo tanto la creación de nuestro vector "temperaturas" será del siguiente modo:

temperaturas = new byte[100];

Una vez creado, el tamaño del vector es fijo (estático), no se puede cambiar.

El ejemplo anterior reserva en memoria 100 posiciones de tipo *byte*.

En los vectores, cuando se reservan N posiciones de memoria, los datos se almacenarán en las posiciones 0, 1, …, N-1 (*ArrayIndexOutOfBoundsException*). Si no se hace esto, el valor inicial por defecto de un array es *null*.

El tamaño también puede asignarse mediante una variable:

int v=100;

byte[] temperaturas;

temperaturas = new byte[v]; (O también, byte[] temperaturas = new byte[v];)

4.1. ARRAYS O VECTORES.

Inicialización de vectores.

Si no se especifica ningún valor, los elementos de un vector se inicializan automáticamente a unos valores predeterminados (variables numéricas a 0, objetos a *null*, booleanos a *false* y caracteres a '\u0000').

También es posible inicializarlo con los valores que desee el programador:

```
byte[] temperaturas = { 11, 12, 13, 14, 15 };
```

En este ejemplo se ha creado un vector de 5 posiciones de tipo *byte* con los valores especificados (no es necesario la instrucción *new*).

Otra forma sería, establecer los valores uno a uno utilizando los subíndices (sí es necesario la instrucción *new*):

```
temperaturas[0] = 11;   temperaturas[1]=12;   …
```

4.1. ARRAYS O VECTORES.

Ej.:

```
/* Quiero leer 10 notas ¿problema? */
/* Solución: utilizar 10 variables o utilizar arrays */

int nota;
int a,b,c,d,e,f,g,h,j,k;
for (int i=1; i<10; i++) nota=sc.nextInt(); // Sobreescribe los datos!!

/* Declaración de un array */

int[] notas; // Opción 1
//int notas[]; // Opción 2
notas=new int[5];
notas[0]=1;
notas[1]=2;
notas[2]=3;
notas[3]=4;
notas[4]=5;

/* Inicialiazación de datos */

int[] notas={1,2,3,4,5}; // Opción 1
int[] notas2=new int[]{1,2,3,4,5}; // Opción 2
String[] notas3={"APROBADO", "SUSPENSO"}; // Ejemplo array de String
```

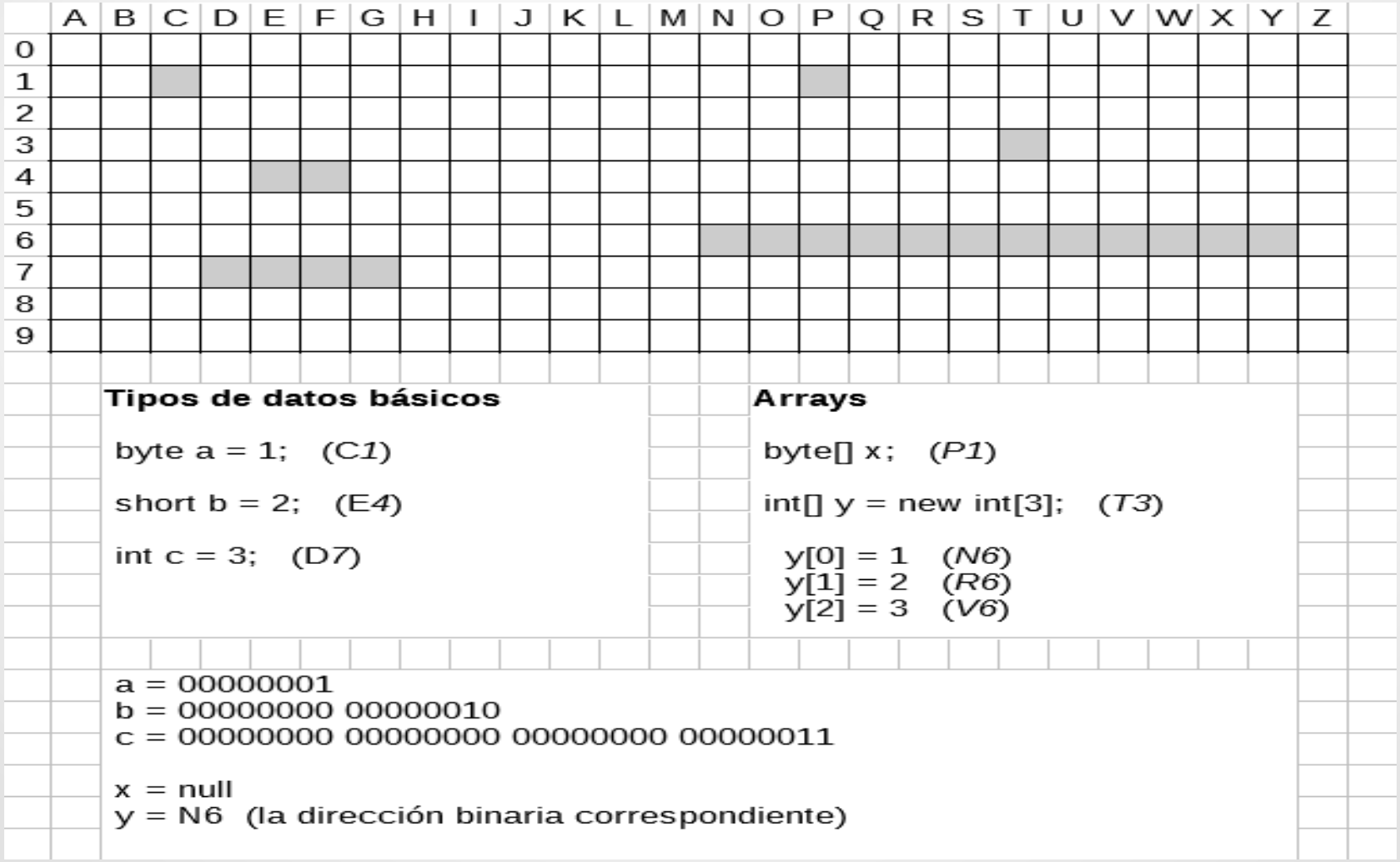
4.1. ARRAYS O VECTORES.

Ej.:

```
/* Ejemplo de utilización de un array */

for (int i=0; i<10; i++) notas[i]=0;

/* Asignación de un array a otro ¿puntero C? */

int[] notas1={1,2,3,4,5};
int[] notas2;
notas2=notas1; // Asigna la referecia (no copia los valores)
System.out.println(notas2[0]);

/* Comparación de arrays */

System.out.println("Notas1 es igual a Notas2 ? ");
System.out.println(notas1==notas2); // Son iguales (compara la referencia)
int[] notas3={1,2,3,4,5};
System.out.println("Notas1 es igual a Notas3 ? ");;
System.out.println(notas1==notas3); // Son diferentes (compara la referencia, NO los valores)
```

4.1. ARRAYS O VECTORES.

Administración de memoria.



4.2. ARRAYS MULTIDIMENSIONALES O MATRICES.

Tratar con matrices en Java es parecido a tratar con vectores. Por ejemplo, una matriz de enteros de dos dimensiones de 5 filas y 8 columnas se crearía de la siguiente manera:

```
int[][] matriz = new int[5][8];
```

La inicialización del array en el momento de la declaración se hará del siguiente modo:

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

O posterior a la declaración:

```
matriz[0][0] = 1; matriz[0][1] = 4; ...
```

En el código anterior se ha creado un array de 2 filas y 3 columnas.

Los arrays pueden tener más de dos dimensiones y no tienen por qué ser "rectangulares":

```
int[][] tabla = new int [2][];
tabla[0] = new int[3];
tabla[1] = new int[4];
```

4.2. ARRAYS MULTIDIMENSIONALES O MATRICES.

El código siguiente muestra en la primera línea el número de filas de la matriz creada anteriormente (2) y la segunda línea, el número de columnas de la fila 0 de la matriz (3):

```
System.out.println(matriz.length);
System.out.println(matriz[0].length);
```

El acceso a la matriz se haría igual que con vectores. Imaginemos que queremos almacenar en cada celda de la matriz la suma de la posición de la fila y la columna. El resultado sería el siguiente:

```
for ( int i=0; i<5; i++ )
    for ( int j=0; j<8; j++ )
        matriz[i][j] = i+j;
```

4.2. ARRAYS MULTIDIMENSIONALES O MATRICES.

Ej.:

```
/* Array multidimensionales */

int[][] notas;
notas=new int[5][10];
notas[0][0]=10;

/* Ejemplo de utilización de un array multidimensional */

for (int i=0; i<5; i++)
    for (int j=0; j<10; j++) notas[i][j]=0;

/* Array multidimensionales "flexibilidad" */

int[][] notas;
notas=new int[5][];
notas[0]=new int[10];
notas[1]=new int[20];
notas[2]=new int[30];
notas[3]=new int[40];
notas[4]=new int[50];
```

4.3. API DE MANEJO DE ARRAYS.

La API de Java proporciona la clase Arrays (*java.util*) para realizar múltiples operaciones con arrays. Las más interesantes son: rellenar el array, buscar si un dato está en el array y ordenar los datos de un array.

```
int [] tabla = new int[20];
Arrays.fill (tabla, 7); //rellena todo el array con el número 7
Arrays.fill (tabla, 4, 11, 7); // rellena de la posición 4 a la 10 con el número 7
Arrays.sort (tabla); //ordena el array en orden ascendente
Arrays.sort (tabla, 6, 16); // ordena sólo una parte del array
int pos = Arrays.binarySearch(tabla, 23);
//busca un dato en un array, el array debe estar previamente ordenado, devuelve un número positivo con la posición del elemento o un número negativo con la posición esperada (si no está)
```

4.3. API DE MANEJO DE ARRAYS.

Ej.:

```
/* Funciones de arrays */

int notas[]=new int[10];
System.out.println(notas.length);
for (int i=0; i<notas.length; i++) notas[i]=i;

/* Clase Arrays y métodos (mirar la API) */

int notas[]=new int[10];
Arrays.fill(notas,10);
Arrays.fill(notas,3,5,10);

//~ Arrays.equals();
//~ Arrays.sort();
//~ Arrays.copyOf();
//~ Arrays.clone();

for (int i=0; i<notas.length; i++) System.out.println(notas[i]);
```

4.4. CADENAS DE CARACTERES.

Una cadena de caracteres es un vector o array de elementos de tipo *char*.

```
char[] nombre1 = { 'p', 'e', 'p', 'e' };
char[] nombre2 = { 112, 101, 112, 101 };
char[] nombre3 = new char[4];
```

En el código anterior las variables *nombre1* y *nombre2* contienen exactamente lo mismo, ya que internamente Java almacena los caracteres con sus símbolos ASCII correspondientes (a la 'p' le corresponde el 112 y a la 'e' el 101). La variable *nombre3* se ha creado como una cadena de 4 caracteres que no ha sido inicializada y, por lo tanto, sus 4 posiciones contendrán el valor '\0'.

Nota: Las cadenas de caracteres en Java se tratan como objetos de la clase String.

4.5. OPERACIONES BÁSICAS.

Cuando tratamos con arrays aparecen operaciones básicas muy habituales con las que tenemos que estar familiarizados:

- *Búsqueda*
- *Inserción*
- *Borrado*
- *Ordenación*

```
public static void mostrar(int[] t) {
    System.out.println("");
    for (int i=0; i<t.length; i++) {
        System.out.print(t[i]+" ");
    }
    System.out.println("");
    System.out.println("");
}
```

4.5. OPERACIONES BÁSICAS.

```
// Búsqueda secuencial de un elemento

int[] t={1,2,3,4,0};
mostrar(t);

int key=3;
int pos=-1;
for (int i=0; i<t.length; i++) {
    if (t[i]==key) {
        pos=i;
        break;
    }
}
if (pos!=-1) {
    System.out.println("Numero "+key+" en posicion "+pos);
} else {
    System.out.println("Numero "+key+ " no existe!!");
}
```

4.5. OPERACIONES BÁSICAS.

```
// Inserción de un elemento por posición

int[] t={1,2,3,4,0};
mostrar(t);

int key=7;
int pos=2;
for (int i=t.length-1; i>pos; i--) {
    t[i]=t[i-1];
}
t[pos]=key;

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Inserción de un elemento ordenado

int[] t={1,3,5,7,0};
mostrar(t);

int key=4;
int pos=-1;
do {
    pos++;
} while (pos<t.length && t[pos]!=0 && t[pos]<key);
if (pos<t.length) {
    for (int i=t.length-1; i>pos; i--) {
        t[i]=t[i-1];
    }
    t[pos]=key;
}

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Borrado de un elemento por posición

int[] t={1,2,3,4,0};
mostrar(t);

int pos=2;
for (int i=pos; i<t.length-1; i++) {
    t[i]=t[i+1];
}
t[t.length-1]=0;

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Borrado de un elemento dado

int[] t={1,2,3,4,0};
mostrar(t);

int key=3;
int pos=-1;
do {
    pos++;
} while (pos<t.length && t[pos]!=key);
if (pos<t.length) {
    for (int i=pos; i<t.length-1; i++) {
        t[i]=t[i+1];
    }
    t[t.length-1]=0;
}

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Ordenación (Selección)

int[] t={4,1,3,2,5};
mostrar(t);

int pos;
for (int i=0; i<t.length-1; i++) {
    pos=i;
    for (int j=i+1; j<t.length; j++) {
        if (t[j]<t[pos]) {
            pos=j;
        }
    }
    int aux=t[i];
    t[i]=t[pos];
    t[pos]=aux;
}

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Ordenación (Burbuja)

int[] t={4,1,3,2,5};
mostrar(t);

for (int i=0; i<t.length-1; i++) {
    for (int j=t.length-1; j>i; j--) {
        if (t[j-1]>t[j]) {
            int aux=t[j-1];
            t[j-1]=t[j];
            t[j]=aux;
        }
    }
}

mostrar(t);
```

4.5. OPERACIONES BÁSICAS.

```
// Ordenación (Inserción)

int[] t={4,1,3,2,5};
mostrar(t);

for (int i=1; i<t.length; i++) {
    int aux=t[i];
    int j=i-1;
    while (j>=0 && aux<t[j]) {
        t[j+1]=t[j];
        j--;
    }
    t[j+1]=aux;
}

mostrar(t);
```

4.6. ENUMERADOS.

Los enumerados son conjuntos de valores constantes para los que no existe un tipo predefinido.

Se definen de la siguiente forma:

```
enum DiaSemana {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES}
enum TurnoClase {MAÑANA, TARDE}
enum Semaforo {VERDE, AMARILLO, ROJO}
```

Su uso sería, por ejemplo:

```
DiaSemana hoy = DiaSemana.VIERNES;
DiaSemana ayer = DiaSemana.JUEVES;
if (hoy!=ayer)
    System.out.println("Hoy es: "+hoy);
```

4.6. ENUMERADOS.

```
/* ENUMERADOS */

/* El tipo básico "enum" define un conjunto de constantes que se representan
 * como identificadores únicos. Son de tipo referencia, al igual que las clases.
 * Los tipos "enum" son implícitamente "final", no se pueden modificar.
 * Los tipos "enum" son implícitamente "static" y no pueden ser locales.
 * No se pueden crear objetos de tipos enumerados (new).
 * En Java, los tipos "enum" se definen internamente como clases "Enum". */

private enum DIAS {LUNES,MARTES,MIERCOLES,JUEVES,VIERNES,SABADO,DOMINGO}
private enum ESTADO {APAGADO,ENCENDIDO}
private enum COLORES {ROJO,VERDE,AZUL}

/* La siguiente declaración sería equivalente pero no nos ofrece
 * las ventajas de los enumerados, aunque permite comparaciones con
 * enteros que los enumerados no permiten!! */

private final static int ROJO=1;
private final static int VERDE=2;
private final static int AZUL=3;
```

4.6. ENUMERADOS.

```
/* EJEMPLOS */

System.out.println("");

DIAS dia=DIAS.LUNES;

if (dia==DIAS.LUNES) System.out.println("El dia es: "+dia);

System.out.println(dia.name());     // Método de la clase Enum
System.out.println(dia.ordinal());  // Médoto de la clase Enum

switch (dia) {
    case LUNES:
        System.out.println("El dia es: "+dia);
        break;
    case MARTES:
        System.out.println("El dia es: "+dia);
        break;
    /* ... */
}

for (int i=0; i<DIAS.values().length; i++) {    // Recorrido del enumerado como array
    System.out.println(DIAS.values()[i].name()+" "+DIAS.values()[i].ordinal());
}

System.out.println("");
```

4.7. ARGUMENTOS EN LÍNEA DE COMANDOS.

```java
public static void main(String[] args)
{
    /* USO DE ARGUMENTOS EN LINEA DE COMANDOS */

    /* En muchos sistemas, es posible pasar argumentos desde la línea de comandos
     * a una aplicación, para lo cual se incluye un parámetro de tipo String[] en
     * la lista de parámetros de la función "main".
     * Por convención, este parámetro se llama "args".
     * Cuando se ejecuta una aplicación con el comando "java", se pasan los
     * argumentos de línea de comandos que aparecen detrás del nombre de la
     * aplicación, al método "main" en forma de objetos String en el array "args".
     * El número de argumentos pasados se obtiene del tamaño del array.
     * */

    System.out.println("");

    for (int i=0; i<args.length; i++ )
        System.out.println(args[i]);

    System.out.println("");
}
```

FIN