Unidad 8

Estructura de almacenamiento. Arrays y cadenas de caracteres

Programación

Índice

Introducción

Hasta ahora hemos visto almacenar datos en variables. Declarábamos una variable y almacenábamos un solo valor, podíamos cambiarlo en el transcurso de nuestro programa y seguir trabajando con esa variable. ¿Y si necesitamos trabajar con más de un dato? Podemos usar esa misma variable e ir cambiando su dato constantemente, eso sería muy ineficiente. Otra opción sería crear tantas variables como necesitáramos, imaginaros que tengamos que trabajar con 10.000 datos diferentes, tendríamos que crear y manipular 10.000 variables, tampoco sería eficiente. Para ello vamos a introducir el concepto de contenedor de elementos, denominado Arrays que nos facilitará el trabajo cuando tengamos que operar con más de un dato.

Al finalizar esta unidad

- Conoceremos una tipología de estructura de almacenamiento, las arrays.
- Sabremos las principales operaciones con arrays
- Trabajaremos con las cadenas de caracteres y como se almacenan.
- Aprenderemos al tratamiento de tipos de datos y transformarlos.

8.1. Estructuras

Una forma de poder agrupar datos y estructurarlos, es definiendo clases para almacenar un conjunto de variables de diferente tipo que tienen una relación entre sí, denominado atributos o campos de clase.

En Java no existen estructuras definidas para agrupar un conjunto de una o más variables de distinto tipo como sucede en otros lenguajes de programación.

Ejemplo de estructura en C:

```
struct tipo_estructura
{
tipo_variable nombre_variable1;
tipo_variable nombre_variable2;
};
```

Equivalencia en java:

```
class tipo_estructura
{
tipo_variable nombre_variable1;
tipo_variable nombre_variable2;
}
```

8.2. Arrays

Un array es un objeto estático que almacena o contiene elementos de forma secuencial un número fijo de valores del mismo tipo.

Un array puede almacenar dos tipos de datos:

- **Primitivo:** Puede ser int, long, double, boolean, etc. Todos los elementos del array cuentan con un valor inicial, ya que son inicializados en el momento de su creación.
- **Abstracto o referencia:** Puede ser cualquier objeto. Cuando se crea el array, todos sus elementos son referencias a null, y por tanto deben crearse a parte.

8.2.1. Dimensiones de un array

Las dimensiones indican la profundidad de un array, vienen representadas por [ ] y podemos tener de dos tipos:

- **Unidimensional.** Los elementos son manejados de forma lineal, como una fila. En su declaración solo tendrá un par de [ ]
- **Multidimensional o Matriz.** Los elementos son manejados en forma de tabla, columnas y filas. En su declaración se añadirá tantos [ ] como dimensiones contenga. Lo más común es trabajar con dos dimensiones, pero se pueden emplear más.

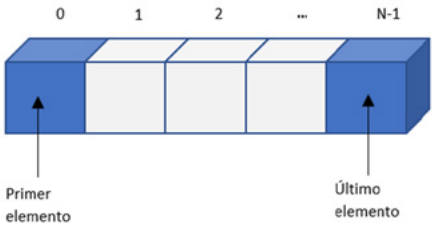

Imagen 1. Representación de un array Unidimensional

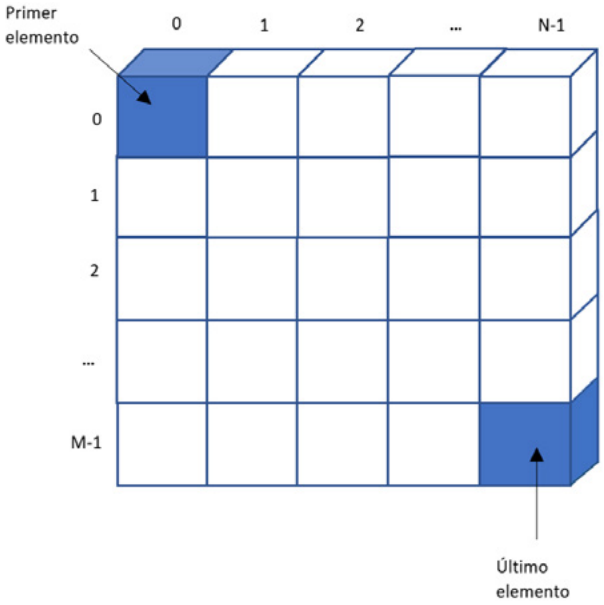

Imagen 2. Representación de un array Multidimensional

8.2.2. Declaración y creación

La declaración de un array es muy similar a una variable. Se indicará la visibilidad, el tipo de datos, tantos [ ] según las dimensiones que tenga y el nombre del array. Es posible alternar el orden de la declaración de la dimensión y el nombre.

Sintaxis

```
[Visibilidad] tipo[].. arrayl;
[Visibilidad] tipo arrayl [].;
```

Al crearlo siempre ira precedido por new, el tipo de dato, y tantos [1..N] según las dimensiones que tenga. Esta vez hay que indicar entre los corchetes la cantidad fija mayor igual a 1 del espacio que tendrá el array.

Sintaxis

```
= new tipo[1…..N]…;
```

Ejemplo de array para almacenar los datos de un cliente, tendrá dos columnas lógicas que representa el nombre y los apellidos, y cuatro filas para añadir cuatro clientes.

```
String [][] datosCliente = new String[4][2];
```

8.2.3. Inicialización y acceso

La inicialización de los elementos puede ser:

- **Directamente en la declaración.** El conjunto de elementos ira agrupado por { } y la cantidad total definida será el número de la capacidad del array.

  ```
  tipo[] nombre = {valor1, valor2}
  tipo[][] nombre = {{valor1, valor2},{valor3, valor4}}
  ```

- **Elemento a elemento.** Hay que acceder a cada posición e introducir su valor.

  ```
  nombre[0] = valor1;
  nombre[1] = valor2;
  nombre[n] = valor3;
  ```

El acceso a cada elemento del array se emplea un índice numérico que especifica la posición del elemento. El índice va desde la primera posición 0 hasta la última que será la cantidad total – 1.

8.2.4. Arrays y métodos

Como se vio en anteriores temas, podíamos invocar a un método y pasarle como argumentos varios parámetros, ahora podemos agrupar y pasar como parámetro un array. Igual que sucede cuando el método ha de retornar diferentes valores, podemos indicar que devuelva un array, de esta manera es posible devolver más de un valor.

8.2.5. Operaciones para realizar sobre un array estático

A continuación, se va a explicar diferentes operaciones más comunes sobre un array. En algunas operaciones se usará la librería java.util.Arrays que proporciona java para facilitar el uso de arrays.

En los diferentes ejemplos vamos a usar los siguientes array:

```
String [] amigos = {“Juan”,”Carla”,”María”,”Pepe”,”Antonio”};
String [][] amigosEdad = {{“Juan”,”20”},{“Carla”,”22”},{“María”,”23”},
{“Pepe”,”22”},{“Antonio”,”30”}};
```

Recorrer un array

Para recorrer un array tenemos que conocer previamente su capacidad total o longitud de la dimensión, para ello el array tiene un campo length que nos devuelve la longitud total. Una vez sabemos su longitud podemos recorrer todo el array mediante un bucle.

> **Empleando un bucle for**

```
for (int i=0;i<amigos.length;i++) {
amigos[i].toString();
}
```

El resultado

> **Empleando un bucle foreach**

```
for (String elemento: amigos) {
elemento.toString();
}
```

> **Recorrido de un array dimensional**

```
for (int i=0;i<amigosEdad.length;i++) {

  System.out.print(“Los datos de mi amigo nº: (“+ i + ”) “);

  for (int j=0;j<amigosEdad[i].length;j++) {
        System.out.print(amigosEdad[i][j] + “ “);
  }
  System.out.println();
}
```

El resultado: Los datos de mi amigo nº: (0) Juan 20 , Los datos de mi amigo nº: (1) Carla 22 , Los datos de mi amigo nº: (2) María 23 , Los datos de mi amigo nº: (3) Pepe 22, Los datos de mi amigo nº: (4) Antonio 30

Búsqueda de elementos

Para buscar elementos vamos a emplear el método .binarySearch(nombreArray, valor). Este método buscará el valor dentro del array y nos devolverá la posición en que se encuentra el elemento.

```
int posicionElemento = java.util.Arrays.binarySearch(amigos, "María");
System.out.println(posicionElemento);
```

El resultado: 2

Impresión de elementos

Se puede imprimir elemento a elemento usando el recorrido completo del array o usando el método .toString(array) pasando como argumento el nombre del array.

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

El resultado: Juan
Carla
María
Pepe
Antonio

```
System.out.println(java.util.Arrays.toString(amigos));
```

El resultado: [Juan, Carla, María, Pepe, Antonio]

Ordenación

Permite ordenar ascendentemente los valores de los elementos de un array.

```
java.util.Arrays.sort(amigos);
```

El resultado: [Antonio, Carla, Juan, María, Pepe]

Comparación

Internamente un array se representa como un objeto, guarda en su dirección de memoria la información que compone el array. Cuando se quiere comparar dos arrays, se puede mirar si apuntan a la misma dirección de memoria o si sus valores son iguales.

- **Mediante el operador "==".** Compara si dos arrays apuntan a la misma dirección memoria.
- **Mediante el método equals.** Compara si dos arrays tienen los mismos valores.

```
String [] amigosNuevos = amigos;
String [] amigosdeJuan = {"Juan","Carla","María","Pepe","Antonio"};
```

```
System.out.println("Los amigos nuevos apuntan a la misma dirección de memoria que amigos: "+ (amigosNuevos == amigos) + " Son los mismo que amigos: "+ java.util.Arrays.equals(amigos, amigosNuevos));
```

```
System.out.println("Los amigos de Juan apuntan a la misma dirección de memoria que amigos: " + (amigosdeJuan == amigos) + " Son los mismo que amigos: "+ java.util.Arrays.equals(amigos, amigosdeJuan));
```

El resultado:

Los amigos nuevos apuntan a la misma dirección de memoria que amigos: true Son los mismo que amigos: true

Los amigos de Juan apuntan a la misma dirección de memoria que amigos: false Son los mismo que amigos: true

Inicialización masiva

Es posible inicializar un array para que todos sus elementos tengan un valor definido. Para ello usaremos el método .fill(Array, Valor)

```
String [] datosClientes = new String[5];
java.util.Arrays.fill(datosClientes, "Sin datos");
```

El resultado: [Sin datos, Sin datos, Sin datos, Sin datos, Sin datos]

Copia

Si se quiere hacer una copia de un array previamente hay que determinar si se quiere hacer una copia deep o profunda o una copia shallow o poco profunda, dependiendo del tipo de sus elementos.

- **Una copia deep o profunda.** El array apuntará a una dirección de memoria diferente y tendrá una copia exacta de sus elementos. La copia será independiente del original, si se modifica el array original o la copia no tendrá afecto el cambio en ambas.
- **Una copia shallow o poco profunda.** Se hace copia de las direcciones de memoria donde apuntan los elementos del array. No es una copia independiente y cualquier modificación afecta a ambas.

Si el tipo de datos es primitivo hará una copia deep o profunda, en cambio, si el tipo de datos no es primitivo hará una copia shallow o poco profunda.

Para hacer una copia podemos usar el método clone(), java.util.Arrays.copyOf(array, longitud) o System.arraycopy(array origen, posición inicio, array copia, posición inicio, longitud).

- **Método clone()**

  La invocación de este método se indica desde el array origen a copiar.

  ```
  String [] amigosCopia = amigos.clone();
  amigosCopia[0] = "Elena";
  ```

```
System.out.println(java.util.Arrays.toString(amigos));
System.out.println(java.util.Arrays.toString(amigosCopia));
```

Resultado: [Antonio, Carla, Juan, María, Pepe]
[Elena, Carla, Juan, María, Pepe]

```
Amigo [] amigosNoPrimitivo = {new Amigo("Juan",22), new Amigo("Elena",25)};
Amigo [] copiaAmigosNoPrimitivo = amigosNoPrimitivo.clone();

System.out.println(amigosNoPrimitivo[0].getNombre());
System.out.println(copiaAmigosNoPrimitivo[0].getNombre());

amigosNoPrimitivo[0].setNombre("Carolina");

System.out.println(amigosNoPrimitivo[0].getNombre());
System.out.println(copiaAmigosNoPrimitivo[0].getNombre());
```

Resultado: Juan
Juan
Carolina
Carolina

- **java.util.Arrays.copyOf** (array, longitud)

Directamente se invoca al método pasando como parámetro el array origen y la longitud hasta donde hará la copia.

```
String[] amigosCopia = java.util.Arrays.copyOf(amigos, amigos.length);

System.out.println(java.util.Arrays.toString(amigos));
System.out.println(java.util.Arrays.toString(amigosCopia));
```

Resultado: [Antonio, Carla, Juan, María, Pepe]
[Antonio, Carla, Juan, María, Pepe]

- **System.arraycopy** (array origen, posición inicio, array copia, posición inicio, longitud)

A diferencia del anterior, previamente es necesario tener los dos arrays, el de origen y la copia. Se invoca al método pasando como parámetros el array origen y su posición inicial donde comenzará a copiar, el array de copia y su posición inicial donde comenzará la copia y la longitud hasta donde hará la copia.

```
String [] amigosCopia = new String[amigos.length];

System.arraycopy(amigos, 0, amigosCopia, 0, amigos.length);

System.out.println(java.util.Arrays.toString(amigos));
System.out.println(java.util.Arrays.toString(amigosCopia));
```

Resultado: [Antonio, Carla, Juan, María, Pepe]
[Antonio, Carla, Juan, María, Pepe]

Inserción de elementos

Operar con arrays estáticos facilita su operativa para trabajar con un conjunto de elementos que ya sabemos al principio, en cambio presenta un gran inconveniente cuando no podemos conocer el tamaño inicial. Si queremos insertar nuevos elementos, hay que declarar un nuevo array incrementando su tamaño, copiar los elementos anteriores y añadir el nuevo elemento en la posición que corresponda.

```
int[] numVisitas = {10, 15, 16};


System.out.println(java.util.Arrays.toString(numVisitas));


int posElemento = 1;
int valorElemento = 31;


int[] nuevoNumVisitas = new int[numVisitas.length+1];
System.arraycopy(numVisitas, 0, nuevoNumVisitas, 0, posElemento);
nuevoNumVisitas[posElemento] = valorElemento;
System.arraycopy(numVisitas, posElemento, nuevoNumVisitas, posElemento+1, numVisitas.
length-posElemento);


System.out.println(java.util.Arrays.toString(nuevoNumVisitas));
```

Resultado: [10, 15, 16]

[10, 31, 15, 16]

Eliminación de elementos

Igual que sucedía con la inserción ahora se presenta al revés, cada vez que queramos eliminar un elemento, hay que declarar un nuevo array decrementando su tamaño, copiar los elementos anteriores a excepción del elemento a borrar.

```
int[] numVisitas = {10, 15, 16};


System.out.println(java.util.Arrays.toString(numVisitas));


int posElemento = 1;
int[] nuevoNumVisitas = new int[numVisitas.length-1];
System.arraycopy(numVisitas, 0, nuevoNumVisitas, 0, posElemento);
System.arraycopy(numVisitas, posElemento+1, nuevoNumVisitas, posElemento, nuevoNum-
Visitas.length-posElemento);
System.out.println(java.util.Arrays.toString(nuevoNumVisitas));
```

Resultado: [10, 15, 16]

[10, 16]

8.3. Cadenas de caracteres

Las cadenas de caracteres es un conjunto de caracteres para formar una palabra o frase de longitud finita, puede contener cualquier digito o carácter y su representación es un array de caracteres.

8.3.1. Declaración, creación e inicialización

Una cadena de caracteres suele venir representada por un array de tipo char. Sin embargo, java proporciona el tipo String para facilitar su uso.

La forma más común de declarar e inicializar una cadena de caracteres es mediante el uso de comillas dobles "" para encerrar la secuencia de caracteres.

Un ejemplo de inicialización:

```
char [] array = {'H','o','l','a'};
String cadena = "Hola";
String arrayACadena = String.copyValueOf(array);
```

8.3.2. Operaciones

El tipo String contiene bastantes métodos para operar con las cadenas de caracteres. A continuación, se detallan los métodos más importantes:

Método	Definición
charAt(int posición)	Devuelve el carácter situado en la posición pasada por parámetro.
compareTo(String cadena)	Devuelve un valor negativo, un valor positivo o cero. Un valor negativo si el String precede a la cadena pasada por parámetro, un valor positivo indica lo contrario y cero si son iguales.
compareToIgnoreCase(String cadena)	Igual al anterior, pero sin tener en cuenta mayúsculas y minúsculas
concat(String cadena)	Une el String con la cadena pasada por parámetro. Es lo mismo que usar el operador +
contains (String cadena)	Devuelve verdadero o falso si el String contiene la cadena pasada por parámetro
indexOf(String cadena)	Devuelve la posición de la primera ocurrencia de la cadena pasada por parámetro.
lastIndexOf(String cadena)	Similar a la anterior, pero empezando por el final.
isEmpty()	Devuelve verdadero o falso si el String contiene algún carácter.
length()	Devuelve el tamaño de la cadena.
replace(String cadenaBuscada, String remplazo)	Devuelve la cadena sustituyendo las partes que coincide con la cadenaBuscada

Método	Definición
substring(int posiciónInicial, int posiciónFinal)	Devuelve la cadena comprendida entre las posiciones indicadas por parámetros.
toCharArray()	Devuelve el array de caracteres de la cadena.
toLowerCase()	Devuelve una cadena en minúscula.
toUpperCase()	Devuelve una cadena en mayúscula.
trim()	Devuelve una cadena sin espacios al principio y final.

Imagen 3. Métodos más importantes que contiene String

Ejemplos,

```
String ejemploCadenaString = “ Ejemplo cadena “;

System.out.println(ejemploCadenaString.trim());
System.out.println(ejemploCadenaString.toLowerCase());
System.out.println(ejemploCadenaString.toUpperCase());
System.out.println(ejemploCadenaString.length());
System.out.println(ejemploCadenaString.substring(0, 8));
System.out.println(ejemploCadenaString.replace(“a”,”1”));
System.out.println(ejemploCadenaString.concat(“contenido nuevo”));
System.out.println(ejemploCadenaString.compareToIgnoreCase(“ Ejemplo cadena “));
```

Resultado:

Ejemplo cadena

ejemplo cadena

EJEMPLO CADENA

16

Ejemplo

Ejemplo c1den1

Ejemplo cadena contenido nuevo

8.3.3. Conversiones

A partir de una cadena de caracteres que contienen dígitos podemos convertirlo a un tipo primitivo de uso numérico o viceversa.

Las conversiones se realizan usando la clase envoltorio de cada tipo.

> **Convertir de String a tipo primitivo.** ← Siempre clase Wrapp + .parse[claseWrapp]()

```
Byte.parseByte(cadena);
Short.parseShort(cadena);
Integer.parseInteger(cadena);
Long.parseLong(cadena);
Float.parseFloat(cadena);
Double.parseDouble(cadena);
```

> **Convertir de String a clase envoltorio** ← Siempre es clase Wrapp + valueOf();

```
Byte.valueOf(cadena);
Short.valueOf(cadena);
Integer.valueOf(cadena);
Long.valueOf(cadena);
Float.valueOf(cadena);
Double.valueOf(cadena);
```

> **Convertir un valor numérico a cadena**

```
String.valueOf(numero);
```

Ejemplos de conversión.

```
int valorNumerico = 100;
String cadenaNumerica = String.valueOf(valorNumerico);

int entero = Integer.parseInt(cadenaNumerica);
float decimal = Float.parseFloat(cadenaNumerica);
double doble = Double.parseDouble(cadenaNumerica);
```

www.universae.com