

Unidad 5

Arrays y cadenas

Programación
1º D.A.M.

1

Contenido

1. Estructuras estáticas vs. dinámicas
2. Arrays
3. Cadenas de caracteres



2

1. Estructuras estáticas vs dinámicas

3

1. Estructuras estáticas vs dinámicas

■ Estructuras de almacenamiento de colecciones

■ Estáticas

- Tamaño conocido en tiempo de compilación
- Ejemplo
 - Arrays de reserva estática (compilación)

■ Dinámicas

- Tamaño variable en tiempo de ejecución
- Tamaño ilimitado
- Ejemplos
 - Pilas, colas, listas enlazadas, árboles, grafos, ...
- Implementación
 - C → Punteros
 - Java → Clases especiales

4

2. Arrays

1. Introducción
2. Manejo de arrays
3. Tipos de arrays
 1. Arrays unidimensionales
 2. Arrays bidimensionales
 3. Arrays multidimensionales
4. Arrays de objetos
5. Arrays y métodos
6. Clase Arrays
7. System.arraycopy

5

2.1. Introducción

- Tabla = Array
- Colecciones de datos homogéneos
 - Mismo tipo de datos
 - Un nombre
 - Almacenados en posiciones contiguas
- Índice
 - Permite acceso a un elemento del array
 - 0 a (Dimensión – 1)
- Número de elementos de una dimensión
 - Propiedad `length`
 - `nombre_array.length`

6

2.2. Manejo de arrays

■ Fases de manejo de un array

1. Definición
 - Creación de la referencia al array (identificador)
2. Instanciación
 - Creación del objeto (reserva de memoria)
3. Iniciación
 - Asignación de valores iniciales a sus posiciones
4. Acceso
 - Lectura y escritura de las posiciones del array

7

2.2. Manejo de arrays

1. Definición de un array

■ Creación de la referencia al array (identificador)

- `tipo[] identificador;`
 - `int[] valores;`
- `tipo identificador[]`
 - `int valores[];`

8

2.2. Manejo de arrays

2. Instanciación de un array

- Creación del objeto
(reserva de memoria)
 - `identificador = new tipo[tamaño]`
 - `valores = new int[10];`
- Posibilidad de hacer definición + instanciación
 - `tipo[] identificador = new tipo[tamaño];`
 - `int[] valores = new int[10];`

9

2.2. Manejo de arrays

3. Iniciación de un array

- Asignación de valores iniciales
 - `identificador = {lista de valores};`
 - `valores = {1,2,3,4,5,6,7,8,9,0};`
- Posibilidad de iniciar al declarar
 - Se reservan las posiciones correspondientes
 - `int[] valores = {1,2,3,4,5,6,7,8,9,0};`
- Asignación de un array a otro
 - `int b[];`
 - `b = valores;`

10

2.2. Manejo de arrays

4. Acceso a una posición

■ Uso de índices

- Valor entero entre 0 y tamaño dimensión-1
- Tantos índices como dimensiones

```

• valores[0] = 1;           // Primero
• System.out.println(valores[9]); // Último
• valores[10] = 4;         // ERROR!!!

```

11

2.3.1. Arrays unidimensionales

■ Definición + Instanciación

```

tipo[] nombre = new tipo[tam];
tipo nombre[] = new tipo[tam];

```

■ Acceso a los datos

```
nombre[indice];
```

- Índice → valor entre 0 y (tam - 1)

■ Carga de datos

■ Iniciación

```

tipo[] nombre = {lista_valores};
tipo nombre[] = {lista_valores};

```

■ Asignación

```
nombre[indice] = valor;
```

12

2.3.2. Arrays bidimensionales

- **Declaración + Instanciación**

```
tipo nombre[][] = new tipo[filas][columnas];
tipo[][] nombre = new tipo[filas][columnas];
```

- **Acceso a los datos**

```
nombre[indice_fila][indice_columna];
indice_fila    → valor entre 0 y (filas - 1)
indice_columna → valor entre 0 y (columnas - 1)
```

- **Carga de datos**

- **Iniciación**

```
tipo[][] nombre = {{lista_val_1}, {lista_val_2},
..., {lista_val_filas}}
```

- **Asignación**

```
nombre[indice_fila][indice_columna] = valor;
```

13

2.3.3. Arrays multidimensionales

- **Declaración + Instanciación**

```
tipo nombre[][]...[] = new tipo[t_dim1]...[t_dimN];
```

- **Acceso a los datos**

```
nombre[i_dim1][i_dim2]...[i_dimN];
i_dim1 → valor entre 0 y (t_dim1 - 1)
...
i_dimN → valor entre 0 y (t_dimN - 1)
```

- **Carga de datos**

- **Iniciación**

```
tipo nombre[t_dim1][t_dim2]...[t_dimN] = {{{...}}}
```

- **Asignación**

```
nombre[i_dim1][i_dim2]...[i_dimN] = valor;
```

14

2.3.3. Arrays multidimensionales

- Sólo obligatoria la primera dimensión
 - Array bidimensional
 - Array de arrays unidimensionales


```
int[][] tabla = new int[2][];
tabla[0] = new int[3];
tabla[1] = new int[4];
```
 - Array tridimensional
 - Array de arrays bidimensionales
 - ...
- Cada fila distinto número de columnas

15

2.4. Arrays de objetos

- Definición
 - `NombreClase[] objetos;`
 - `NombreClase objetos[];`
- Instanciación del array
 - `objetos = new NombreClase[N];`
- Definición + Instanciación
 - `NombreClase[] objetos = new NombreClase[N];`
 - `NombreClase objetos[] = new NombreClase[N];`
- Instanciación de objetos del array
 - `objetos[i] = new NombreClase();`
- Tamaño del array
 - Atributo público `length`
 - `objetos.length`

16

2.5. Arrays y métodos

- Paso de la referencia al array (identificador)
 - Paso por referencia
 - Todos los arrays son objetos
 - Las modificaciones quedarán fuera también
- No se especifica el tamaño
 - Se puede conocer con la propiedad `length`
- Ejemplo

```
void rellena(double[] temperaturas)
```

17

2.5. Arrays y métodos

- Método `main`
 - `public static void main(String[] args)`
 - `args`
 - Array de cadenas de caracteres (`String`)
 - Lista de argumentos pasados al programa
 - `args.length`
 - Número de argumentos pasados al programa
 - `args[0]`
 - Primer argumento pasado al programa
 - `args[args.length - 1]`
 - Último argumento pasado al programa

18

2.6. Clase Arrays

- Clase estática del paquete `java.util`
 - Métodos estáticos para manejar arrays
 - `Arrays.metodo(parámetros);`
 - Métodos interesantes
 - `fill`
 - `equals`
 - `sort`
 - `binarySearch`

19

2.6. Clase Arrays

- Método `fill`
 - Rellena array con un valor
 - `int valores[] = new int[50];`
 - `Arrays.fill(valores, 2);`
 - Todas las posiciones tendrán el valor 2
 - `Arrays.fill(valores, 5, 8, -2)`
 - Desde índice 5 hasta el 7, valor -2

20

2.6. Clase Arrays

■ Método `equals`

- Devuelve true si dos arrays son iguales
 - Mismo tipo
 - Mismo tamaño
 - Contienen los mismos valores
- False en caso contrario
 - `Arrays.equals(array1, array2);`

21

2.6. Clase Arrays

■ Método `sort`

- Ordena un array en orden ascendente
 - `int v[]={7,8,2,0,9,1,4,3,2,6};`
 - `Arrays.sort(v);`
 - Ordena los valores en orden ascendente
 - `Arrays.sort(v, 2, 5);`
 - Ordena los valores del índice 2 al índice 4

22

2.6. Clase Arrays

■ Método `binarySearch`

■ Búsqueda rápida de un elemento

- Funciona en un array ordenado
- Resultados indefinidos en uno desordenado
- Devuelve el índice que ocupa el elemento

```
int v[]={7,8,2,0,9,1,4,3,2,6};
Arrays.sort(v);
System.out.println(Arrays.binarySearch(v,3));
Devuelve 4
```

23

2.7. `System.arraycopy`

■ Copia un array en otro

```
System.arraycopy(origen, desde1, destino,
                 desde2, cuánto)
```

- `origen`: array que se copia
- `desde1`: posición inicial que se copia
- `destino`: array donde se copia
- `desde2`: posición inicial en que se copia
- `cuánto`: cuántos elementos se copian

24

3. Cadenas de caracteres

1. Clase String
2. Clase StringBuffer
3. Clase StringTokenizer
4. Envoltorios y cadenas

25

3.1. Clase String

- Clase para crear objetos cadena
- Manejo de cadenas
 - Creación de la referencia
 - Instanciación del objeto
 - Manipulación
 - Asignación de valor
 - Búsqueda de subcadenas
 - Búsqueda de caracteres
 - Recorte de la cadena
 - ...

26

3.1. Clase String

■ Manejo de cadenas (String)

■ Creación de referencia + instanciación

```
String palabra = "Sergio";

char[] cadena = {'S', 'e', 'r', 'g', 'i', 'o'};
String nombre = new String(cadena);

String valor = new String("");
```

27

3.1. Clase String

■ Comparación de objetos String

■ Dados dos objetos cadena1, cadena2

- `cadena1.equals(cadena2)`
 - `true` si `cadena1` y `cadena2` son iguales
- `cadena1.equalsIgnoreCase(cadena2)`
 - Como el anterior, ignorando mayúsculas y minúsculas
- `cadena1.compareTo(cadena2)`
 - Diferencia numérica alfabética $|cadena1 - cadena2|$
 - 0 si son iguales (alfabéticamente – ASCII)
- `cadena1.compareToIgnoreCase(cadena2)`
 - Como el anterior, ignorando mayúsculas y minúsculas

28

3.1. Clase String

■ `String.valueOf`

■ Método estático

- Convierte a cadena algo que no lo es

```
String numero = String.valueOf(12345);
```

```
String fecha = String.valueOf(new Date());
```

- Existe el análogo en otras muchas clases

29

3.1. Clase String

■ Algunos métodos de variables String

■ `length()`

- Longitud de la cadena (número de caracteres)

■ `concat(otra)`

- Devuelve la cadena original con la cadena "otra" al final

■ `charAt(índice)`

- Carácter ubicado en una posición (índice de 0 a `length() - 1`)

■ `substring(desde, hasta)`

- Subcadena desde el índice "desde" hasta el "hasta - 1"

■ `indexOf(...)`

- Posición (índice) de un carácter en una cadena
- Posición (índice) de una subcadena en una cadena
- Posición (índice) de un carácter en una cadena desde un índice
- Posición (índice) de una subcadena en la cadena desde un índice

30

3.1. Clase String

■ Algunos métodos de variables String

- `lastIndexOf(...)`
 - Como `indexOf` pero comenzando la búsqueda por atrás
- `endsWith(texto)`
 - True si la cadena termina con el texto indicado
- `startsWith(texto)`
 - True si la cadena comienza por el texto indicado
- `replace(carácter1, carácter2)`
 - Sustituye las apariciones del carácter1 por carácter2
 - Proporciona el resultado, no hace el cambio en la cadena
- `replaceAll(cadena1, cadena2)`
 - Sustituye las apariciones de la cadena1 por cadena2
 - Proporciona el resultado, no hace el cambio en la cadena

31

3.1. Clase String

■ Algunos métodos de variables String

- `toUpperCase()`
 - Devuelve la misma cadena convertida a mayúsculas
- `toLowerCase()`
 - Devuelve la misma cadena convertida a minúsculas
- `toArray()`
 - Obtiene un array de caracteres a partir de la cadena
- `split(patión)`
 - Divide una cadena en partes según un patrón

32

3.2. Clase StringBuffer

- Permite crear cadenas mutables
 - Se puede modificar su contenido
 - Crear la cadena
 - Añadir elementos a la cadena
 - Igual que con '+', pero mucho mejor rendimiento
 - Eliminar elementos de la cadena
 - Modificar elementos de la cadena

33

3.2. Clase StringBuffer

- Algunos métodos
 - `append(elemento)`
 - Añade al final de la cadena el "elemento"
 - El "elemento" puede ser de distintos tipos
 - `delete(comienzo, fin)`
 - Elimina la subcadena "comienzo – fin"
 - `insert(...)`
 - Inserta un elemento a partir de una posición
 - `replace(origen, fin, cadena)`
 - Reemplaza las posiciones de origen a fin por "cadena"

34

3.2. Clase StringBuffer

■ Algunos métodos

- `capacity()`
 - Devuelve la capacidad actual
- `toString()`
 - Obtiene el String correspondiente

■ Otros métodos

- `charAt`, `deleteCharAt`, `ensureCapacity`, `indexOf`, `lastIndexOf`, `length`, `reverse`, `setCharAt`, `substring`, `trimToSize`, ...

35

3.3. Clase StringTokenizer

■ Divide una cadena en tokens

- Token: subcadena entre caracteres separadores
 - El separador puede ser cualquier cadena dada

■ Manejo

- Inclusión de `java.util.StringTokenizer`
- Creación

```
StringTokenizer st = new StringTokenizer(cadena, cadSeparador);
```

■ Manipulación

- `st.hasMoreTokens()`
 - Indica si hay más tokens o no
- `st.nextToken()`
 - Devuelve el siguiente token

36

3.4. Envoltorios y cadenas

■ Envoltorios o Wrappers

- Clases asociadas a los tipos de datos básicos
 - `Integer`, `Float`, `Double`, `Character`, ...
- Proporcionan funcionalidad asociada a su tipo
- Parte de dicha funcionalidad incluye
 - Conversión desde el tipo básico a cadena (`String`)
 - Método `toString`
 - `Integer.toString`, `Float.toString`, ...
 - Conversión de cadena (`String`) al tipo básico
 - Método `parse<Tipo>`
 - `Integer.parseInt`, `Float.parseFloat`, ...

37

Unidad 5

Arrays y cadenas

Programación
1º D.A.M.

38