Programación

Estructuras de Almacenamiento
Cadenas de Caracteres y Arrays

Esta presentación ha sido elaborada a partir del material del módulo de Programación del IES El Majuelo (Gines)

ÍNDICE

1. Concepto de estructura

- Es un tipo de declaración que puede contener más de un dato.

- Ejemplos:
  - Arrays.
  - Listas enlazadas.
  - Pilas.
  - Colas.

2. Cadenas de caracteres

- No existe un tipo primitivo.
  - Se utiliza la clase `String`
    - Definida en el API de JAVA.
- Una cadena de caracteres es un objeto.
- Sus elementos están indexados
  - También en un *array*.

2. Cadenas de caracteres
2.1. Declaración

```
String cadena;
```

2. Cadenas de caracteres
2.2. Creación

```
String cadena = new String();
```

No resulta imprescindible hacerlo.

2. Cadenas de caracteres
2.3. Inicialización

```
String cadena = new String(“Cadena”);
```

o bien

```
String cadena = “Cadena”;
```

El valor almacenado en un `string` no puede cambiarse (lo que cambia es la referencia).

2. Cadenas de caracteres
2.4. Operaciones

- **`char charAt (int pos)`:** devuelve el carácter que está en la posición `pos`.
- **`boolean equals (String cadEnv)`:** compara el `string` con la cadena enviada (distingue entre minúsculas y mayúsculas).
- **`int compareTo (String cadEnv)`:** compara el `string` con la cadena enviada. Devuelve un 0 si son iguales, un número negativo si el `string` va antes (alfabéticamente) y un número positivo en caso contrario.
- **`int compareToIgnore (String cadena)`:** ídem pero sin distinguir minúsculas y mayúsculas.
- **`String toLowerCase()`:** devuelve el `string` convertido a minúsculas.
- **`String toUpperCase()`:** devuelve el `string` convertido a mayúsculas.

2. Cadenas de caracteres
2.4. Operaciones

- `String toString()`
- `String concat(String str)`
- `int length()`
- `String trim()`
- `boolean startsWith(String prefijo)`
- `boolean endsWith(String sufijo)`
- `String substring(int IndiceInicial, int Indicefinal)`
- `int indexOf(int car)`
- `int indexOf(String str)`
- `String replace(char car, char nuevoCar)`
- `static String valueOf(tipo dato)`
- `char[] toCharArray()`

2. Cadenas de caracteres
2.5. Conversiones

- Para convertir una variable de un tipo de datos simple (`char, boolean, int, long, float, double`) en una cadena `String` hay que utilizar el método **`valueOf()`** de la clase `String`.
- El proceso contrario se describe aquí:
  - `Boolean.`**`parseBoolean`**`(cadena_a_convertir);`
  - `Int.`**`parseInt`**`(cadena_a_convertir);`
  - `Long.`**`parseLong(`**`cadena_a_convertir);`
  - `Float.`**`parseFloat`**`(cadena_a_convertir);`
  - `Double.`**`parseDouble`**`(cadena_a_convertir);`
- Concatenación: operador '**+**'.

3. Arrays

- Es una estructura de datos donde toda la información que contiene es del mismo tipo.
- Propiedades:
  - Datos relacionados.
  - Tamaño de la estructura conocido de antemano (no modificable).
  - Acceso a los elementos a través de su posición.
- Tipos:
  - Unidimensionales.
  - Bidimensionales (“matrices”).
  - Multidimensionales.

3. Arrays
3.1. Declaración

```
tipo identificador[];
```

O bien

```
tipo [] identificador;
```

```
tipo identificador [][];
```

etc…

3. Arrays
3.2. Creación

```
identificador = new tipo[Elementos];
identificador = new tipo[Filas][Columnas];
```

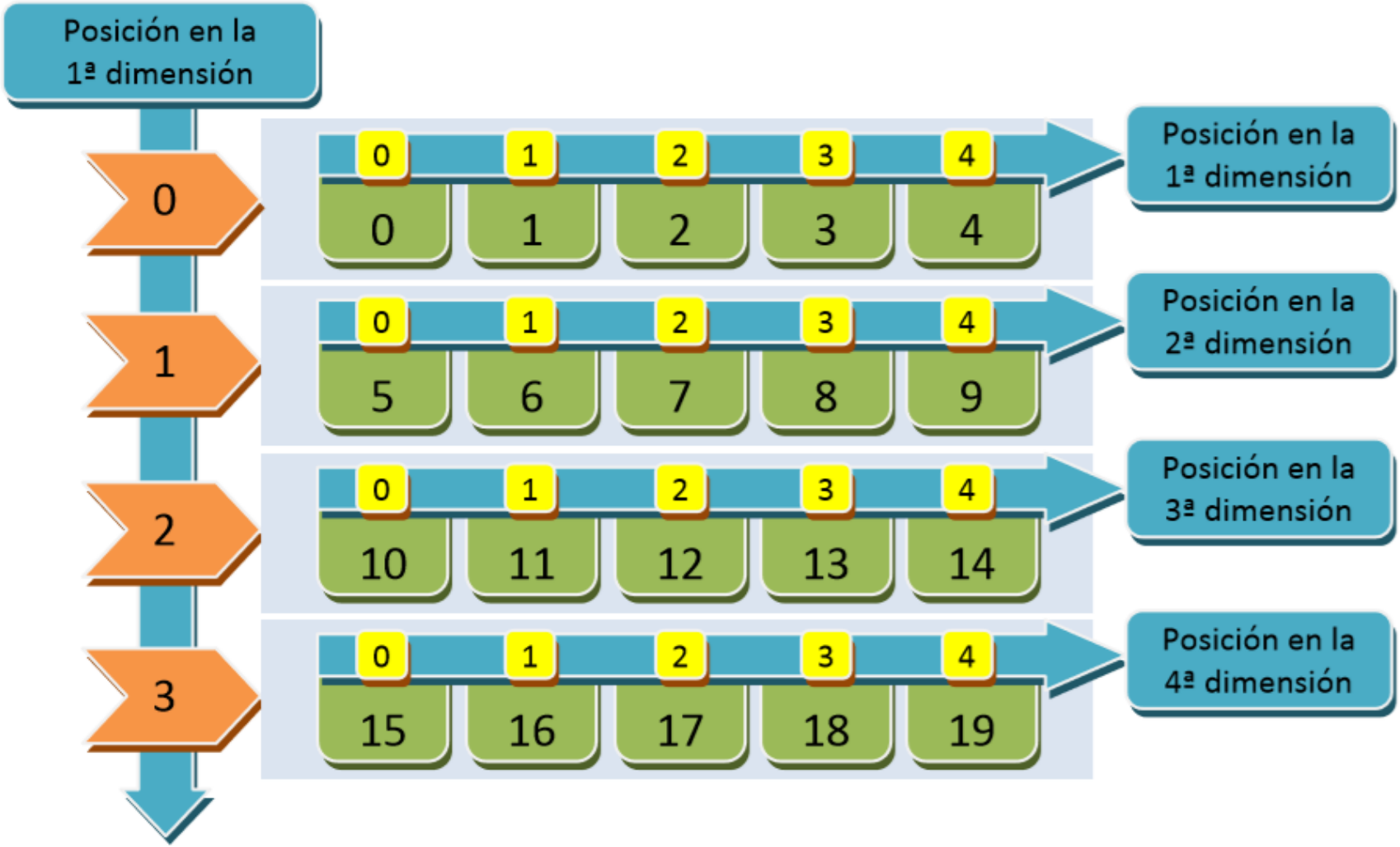
etc…

Pasos 1 y 2 juntos:

```
tipo [] identificador = new tipo[Elementos];
```

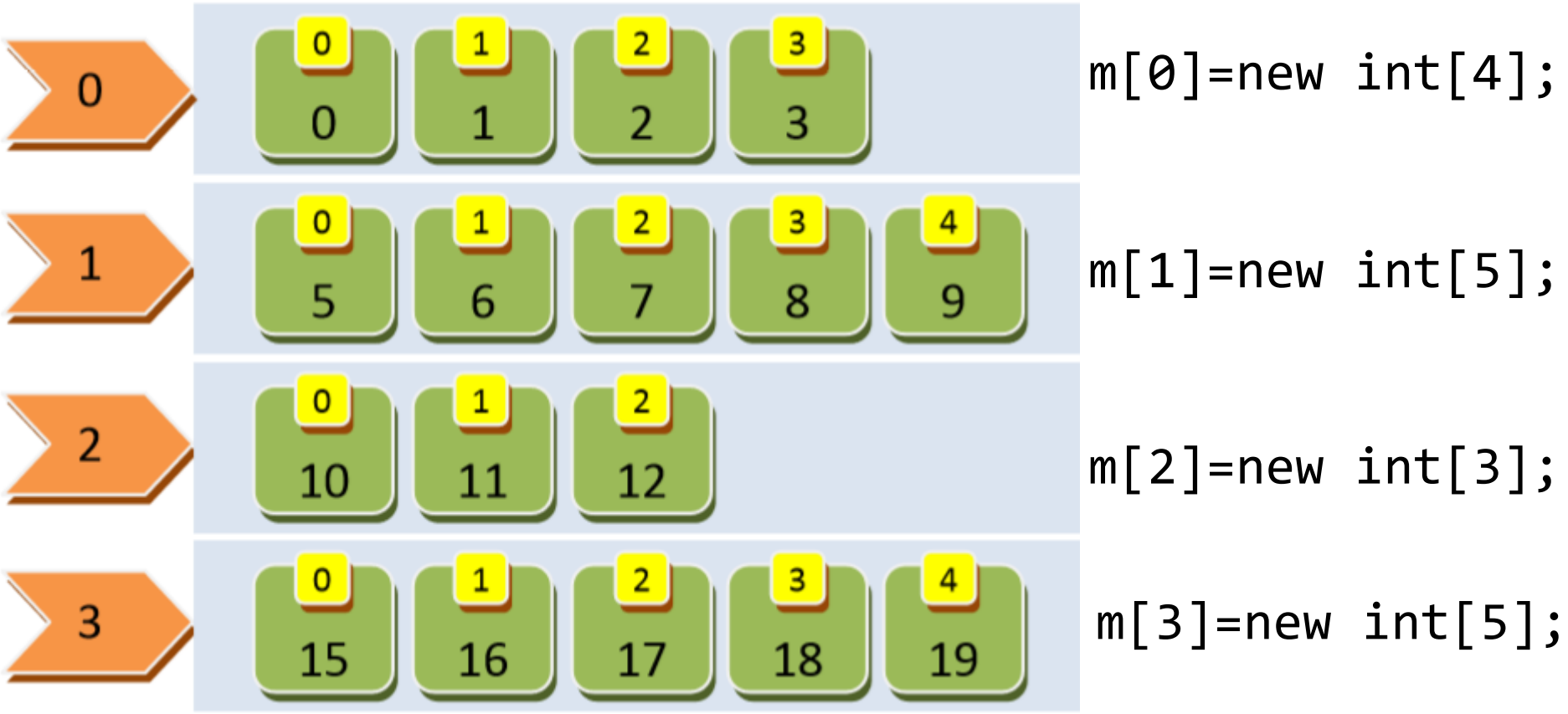
3. Arrays
3.2. Creación

```
int[][] matriz = new int[4][5];
```



3. Arrays
3.2. Creación

```
int[][] m = new int[4][];
```



3. Arrays
3.3. Inicialización

Vector:

```
int vector[] = {1, 2, 3, 5, 7};
String lista_Nombres[] = {“Maria”, “David”};
```

Matriz:

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

JAVA determina el tamaño del *array* en función de los valores asignados y hace la reserva de memoria sin tener que hacer new.

3. Arrays
3.4. Acceso a elementos

- Los elementos del *array* se identifican por la posición que ocupa que va desde la posición cero, que sería el primer elemento, hasta *tam-1*, que sería el último elemento.
- El tamaño del vector se puede conocer utilizando su propiedad *length*.
  - **`vector.length`**.
  - **`matriz.length`** (filas).
  - **`matriz[pos].length`** (columnas de la fila `pos`).
- Java no permite el acceso a posiciones no definidas en un *array.* Salta la excepción **`ArrayIndexOutOfBoundsException`**.

3. Arrays
3.5. Recorrido

- Vector:
  - Bucle de recorrido (de inicio a fin o viceversa).
- Matriz:
  - Recorrido por filas o por columnas (un bucle para cada criterio), dos bucles anidados.

- Ambos sirven tanto para tipos básicos (`int`) como para referenciados (objetos).

3. Arrays
3.6. Búsqueda y ordenación

- Búsqueda lineal:
  - *Array* no ordenado.
    - Recorrido completo.
  - *Array* ordenado.
    - Recorrido parcial.
- Búsqueda binaria (vector):
  - Tiene que estar ordenado.
- Ordenación.
  - Diversos métodos.

3. Arrays
3.7. Clase *Arrays*

- API de Java: **`import java.util.Arrays`**
- Métodos **estáticos** para manipular tablas y operar sobre ellas.
- Simplifica el trabajo con tablas.

3. Arrays
3.7. Clase *Arrays*

- Arrays.**toString**(t): String
- Arrays.**fill**(t, valor): void
- Arrays.**fill**(t, posInicio, posFin, valor): void
- Arrays.**equals**(tA, tB): boolean
- Arrays.**binarySearch**(t, valor): int
- Arrays.**binarySearch**(t, posInicio, posFin, valor): int
- Arrays.**sort** (t): void
- Arrays.**copyOf**(tOrigen, longitud): t[longitud]
- Arrays.**copyOfRange** (tOrigen, posInicio, posFin): t[]