UNIVERSIDAD DE MÁLAGA

TEMA IV

Tipos de Datos Estructurados

Contenido del Tema

IV.1. Cadenas de caracteres.

- IV.1.1. Clases *String* y *StringBuilder*.
- IV.1.2. Asignación, comparación y paso de parámetros.

IV.2. Arrays.

- IV.2.1. Justificación.
- IV.2.2. Conceptos básicos.
- IV.2.3. Asignación, comparación y paso de parámetros.
- IV.2.4. Copia y duplicación de arrays.
- IV.2.5. Arrays Multidimensionales

Samsung TECH INSTITUTE

Tipos Estructurados

- Los tipos de datos vistos hasta ahora han sido los tipos básicos o simples:

`byte` (entero de 8 bits)	**`short`** (entero de 16 bits)
`int` (entero de 32 bits)	**`long`** (entero de 64 bits)
`float` (decimal de 32 bits)	**`double`** (decimal de 64 bits)
`char` (Unicode de 16 bits)	**`boolean`** (**`true`**, **`false`**)

- Para abordar problemas de mayor envergadura necesitamos tipos de datos más complejos (estructurados o compuestos). En este tema vamos a estudiar dos:
  - Cadenas de caracteres
  - Arrays
- En el Módulo 2 se tratarán más.

Tipos Estructurados

- Las variables de los tipos básicos almacenan “valores”. Por ejemplo, una variable de tipo **`int`** podrá almacenar un número entero.
- En cambio, las variables de los tipos estructurados (y de las clases en general, como se verá en el Módulo 2) almacenan realmente la dirección (referencia) en memoria donde se encuentran los datos (la dirección del objeto en sí).
- Esto tiene su importancia a la hora de entender la asignación, comparación y paso de parámetros, como ya veremos más adelante.

Cadenas de caracteres

- Las cadenas o secuencias de caracteres son frecuentes en los programas. Por ej. si manejamos una agenda personal, el nombre, la dirección, etc. son cadenas de caracteres.
- Las cadenas de caracteres literales (las únicas que hemos visto hasta ahora) se representan en Java como secuencias de caracteres Unicode encerradas entre comillas dobles.
- Para manipular variables y constantes que almacenan cadenas de caracteres se utilizan tres clases:
  - **`String`** - para cadenas inmutables
  - **`StringBuilder`** - para cadenas modificables
  - **`StringBuffer`** - para cadenas modificables (seguras ante hebras)
- Vamos a estudiar las dos primeras.

UNIVERSIDAD DE MÁLAGA

Clase String

- Una variable del tipo (clase) **`String`** puede almacenar una cadena de caracteres.

- Las variables de este tipo se pueden inicializar de dos formas diferentes:
  - ✓ **`String str = new String("¡Hola!");`**
  - ✓ **`String str = "¡Hola!";`**

Clase String

- Se puede leer de teclado una cadena de caracteres mediante el uso de Scanner:

```
Scanner teclado = new Scanner(System.in);
str = teclado.next();
      // se lee una cadena hasta separador
      // (blanco, tabulador o fin de linea)

str = teclado.nextLine();
      // se lee una cadena hasta fin de linea
```

- Se puede escribir por pantalla una cadena de caracteres mediante:

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

UNIVERSIDAD DE MÁLAGA

Clase String

- A una variable de tipo **`String`** se le puede asignar cadenas distintas durante su existencia.

```
String str = "¡Hola!";
...
str = "¡adios!";
```

- Una cadena de caracteres almacenada en una variable de tipo **`String`** no puede modificarse (crecer, cambiar un carácter, ...).

Métodos de String

- En el tema 3, estudiamos métodos estáticos (**`static`**) para aplicar diseño descendente. Su invocación se hacía usando su nombre (y los parámetros necesarios) directamente. Todos pertenecían a la clase que estuviéramos diseñando
- Los métodos que veamos en este tema 4, tanto para la clase **`String`** como para la **`StringBuilder`** (salvo que sean estáticos) deben invocarse poniendo primero la variable sobre la que queremos aplicar el método, seguida de un punto y finalmente el nombre del método (y los parámetros necesarios).
  - Esta notación ya la hemos usado con **Scanner**.
- Si se trata de un método estático, la invocación se realizará poniendo primero el nombre de la clase, seguido de un punto y finalmente el nombre del método (y los parámetros necesarios).
  - Esta notación ya la hemos usado con **System**.

Métodos de String

- Métodos de consulta:

```
int length()
char charAt(int pos)
int indexOf/lastIndexOf(char car)
int indexOf/lastIndexOf(String str)
int indexOf/lastIndexOf(char car, int desde)
int indexOf/lastIndexOf(String str, int desde)

   ...
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
IndexOutOfBoundsException
StringIndexOutOfBoundsException
```

Métodos de String

- Métodos que producen nuevos objetos **`String`**:

```
String substring(int posini, int posfin)
String substring(int posini)
String replace(String str1, String str2)
 String replaceFirst(String str1, String str2)
String concat(String s)   // también con +
String toUpperCase()
String toLowerCase()
static String format(String formato,...)
...
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
IndexOutOfBoundsException
StringIndexOutOfBoundsException
```

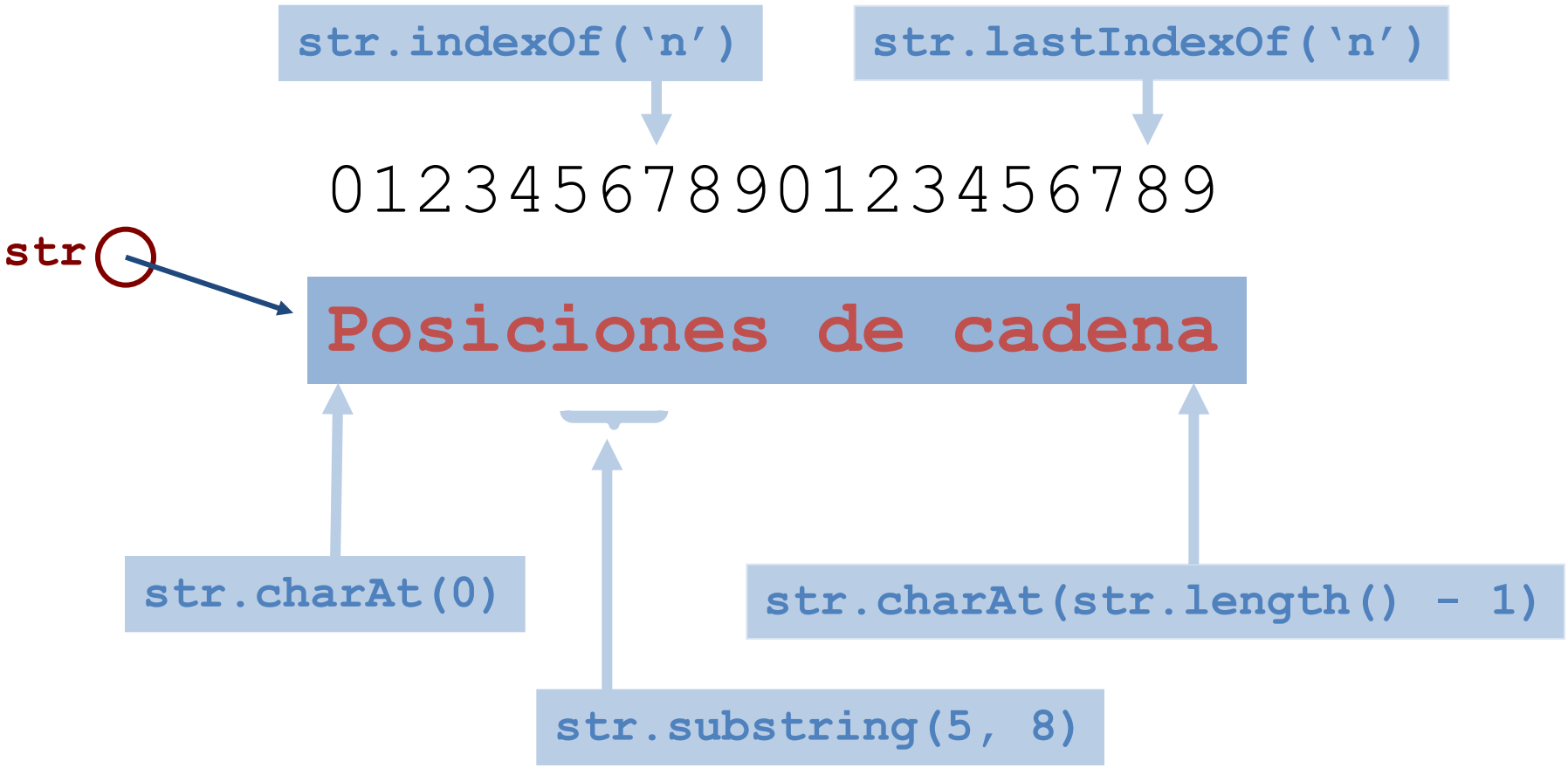
UNIVERSIDAD DE MÁLAGA

Métodos de String

- Comparación:

```
boolean equals(String str)
boolean equalsIgnoreCase(String str)
int compareTo(String str)
int compareToIgnoreCase(String str)
```

Métodos de String



UNIVERSIDAD DE MÁLAGA

Métodos de String

Concatenación

```
String nombre = "Juan";
String apellidos = "Pérez López";
String nombreCompleto = nombre + " " + apellidos;
```

Mayúsculas y minúsculas

```
String nombre = "Juan";
String nombreMay = nombre.toUpperCase();
String nombreMin = nombre.toLowerCase();
```

Métodos de String

El método format

- A partir de JDK 1.5.
- Permite construir salidas con formato.

```
String ej = "Cadena de ejemplo";
String s = String.format("La cadena %s mide %d", ej, ej.length());
System.out.println(s);
```

```
La cadena Cadena de ejemplo mide 17
```

- Formatos más comunes (se aplican con %):
  - s para cualquier objeto. Se aplica toString(). "%20s"
  - d para números sin decimales. "%7d"
  - f para números con decimales. "%9.2f"
  - b para booleanos "%b"
  - c para caracteres. "%c"

- Se pueden producir las excepciones:
  - MissingFormatArgumentException
  - IllegalFormatConversionException
  - UnknownFormatConversionException

Métodos de String

El método format

- Si el resultado de **`format`** es para mostrarlo por pantalla, podemos utilizar directamente **`printf(String formato, …)`**:

```
String ej = "Cadena de ejemplo";
System.out.printf("La cadena %s mide %d\n", ej, ej.length());
```

```
La cadena Cadena de ejemplo mide 17
```

Clase StringBuilder

- A una variable del tipo (clase) **`StringBuilder`** se le puede asignar cadenas distintas durante su existencia.

- Las variables de este tipo se inicializan de cualquiera de las formas siguientes:

✓
```
StringBuilder strB = new StringBuilder();
    // cadena vacía y capacidad inicial para 16 caracteres
```
✓
```
StringBuilder strB = new StringBuilder(10);
    // cadena vacía y capacidad inicial para 10 caracteres
```
✓
```
StringBuilder strB = new StringBuilder("hola");
    // cadena "hola" y capacidad inicial para 4+16 caracteres
```

UNIVERSIDAD DE MÁLAGA

Clase StringBuilder

- Una cadena de caracteres almacenada en una variable de tipo **`StringBuilder`** se puede ampliar, reducir y modificar mediante los métodos correspondientes.

- Cuando la capacidad de almacenamiento establecida se excede, se aumenta automáticamente.

Métodos de StringBuilder

- Métodos de consulta:

```
int length()
int capacity()
char charAt(int pos)
int indexOf/lastIndexOf(String str)
int indexOf/lastIndexOf(String str, int desde)
        (no tiene las versiones para char)
   ...
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
IndexOutOfBoundsException
StringIndexOutOfBoundsException
```

Métodos de StringBuilder

- Métodos para construir objetos **`String`**:

```
String substring(int posini, int posfin)
String substring(int posini)
String toString()
...
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
IndexOutOfBoundsException
StringIndexOutOfBoundsException
```

Métodos de StringBuilder

- Métodos para modificar objetos **`StringBuilder`**:

```
StringBuilder append(String str)
StringBuilder insert(int pos, String str)
void setCharAt(int pos, char car)
StringBuilder replace(int pos1,int pos2,
                      String str)
StringBuilder reverse()
...
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
IndexOutOfBoundsException
StringIndexOutOfBoundsException
```

Métodos de StringBuilder

- La clase **`StringBuilder`** no tiene definidos los métodos para realizar comparaciones que tiene la **`String`**.
- Pero se puede usar el método **`toString()`** para obtener un **`String`** a partir de un **`StringBuilder`** y poder usarlo para comparar.

```
StringBuilder sb1,sb2;
...
if (sb1.toString().equals(sb2.toString())) {
...
}
```

Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`.**

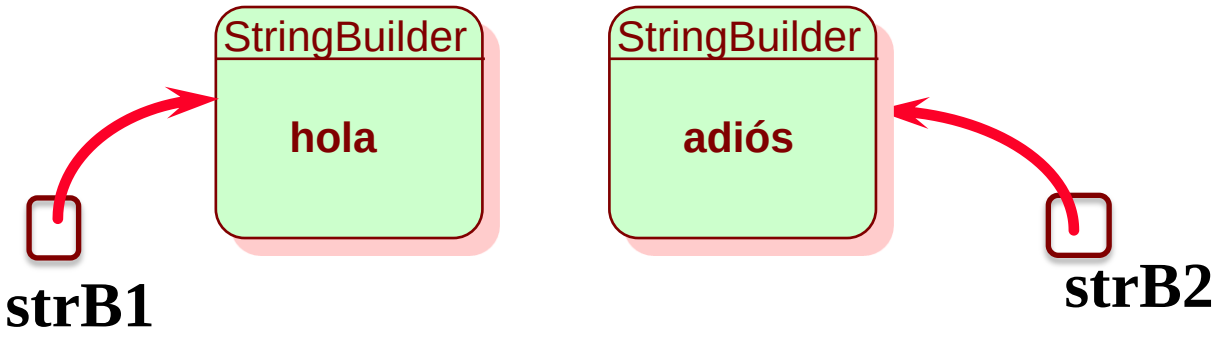
```
StringBuilder strB1= new Stringbuilder("hola");
StringBuilder strB2= new Stringbuilder("adiós");
...
strB1 = strB2;
```

- Como comentamos al principio del tema, las variables de tipos estructurados no almacenan el dato en sí, sino una referencia a la posición de memoria donde se encuentra ese dato (objeto).

Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`.**

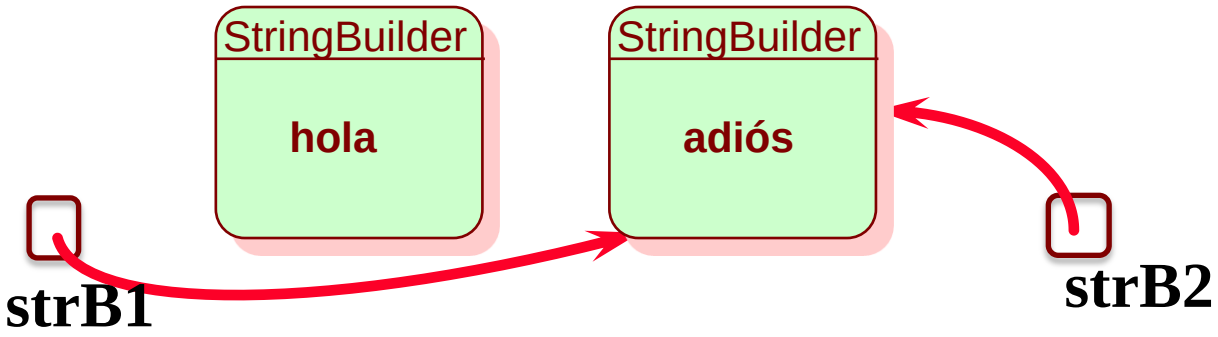
```
StringBuilder strB1= new Stringbuilder("hola");
StringBuilder strB2= new Stringbuilder("adiós");
...
strB1 = strB2;
```



Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`.**

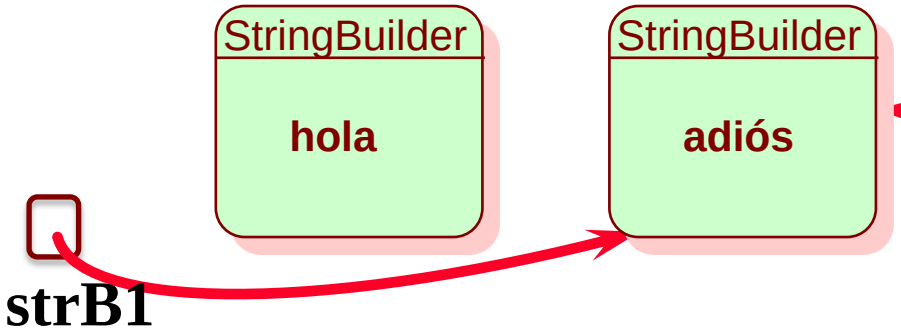
```
StringBuilder strB1= new Stringbuilder("hola");
StringBuilder strB2= new Stringbuilder("adiós");
...
strB1 = strB2;
```



Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`.**

```
StringBuilder strB1= new Stringbuilder("hola");
StringBuilder strB2= new Stringbuilder("adiós");
...
strB1 = strB2;
```

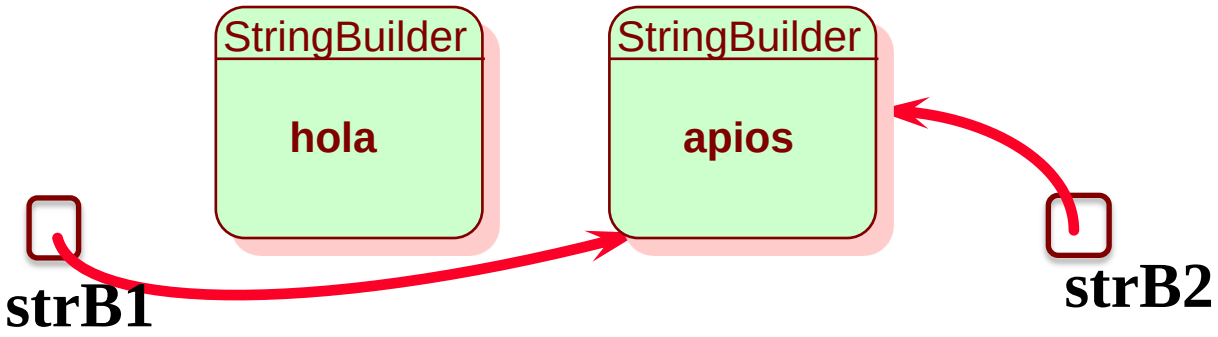


- El objeto "hola" ya no es accesible (Java dispone de un recolector de basura automático)

Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`.**

```
StringBuilder strB1= new Stringbuilder("hola");
StringBuilder strB2= new Stringbuilder("adiós");
...
strB1 = strB2;
```



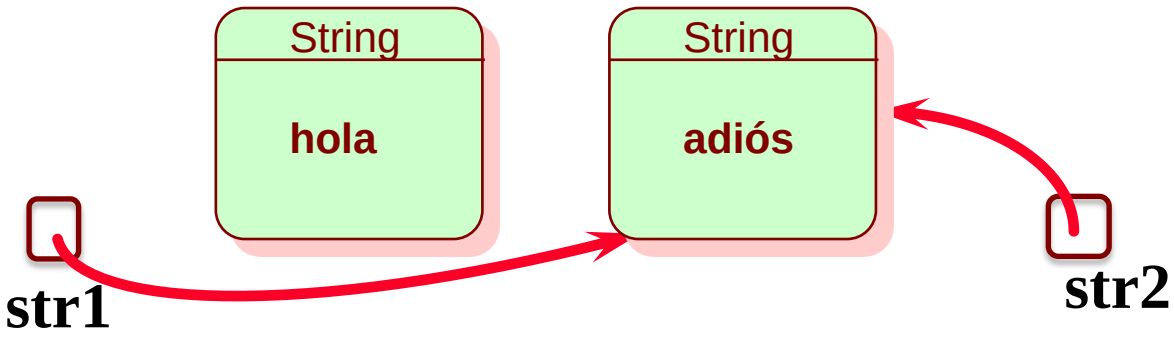
- Cualquier modificación en **`strB1`** implica una modificación en **`strB2`** (y viceversa)

```
strB1.replace(1,4,"pio");
```

Asignación

- Una variable de tipo **`String`** puede ser asignada a otra. Igual ocurre para el tipo **`StringBuilder`**.

```
String str1= new String("hola");
String str2= new String("adiós");
...
str1 = str2;
```



- Al tipo **`String`** no le es aplicable este último comentario (modificación), porque sus objetos son objetos inmutables

Comparación

- La clase **`String`** tiene varios métodos para realizar comparaciones:

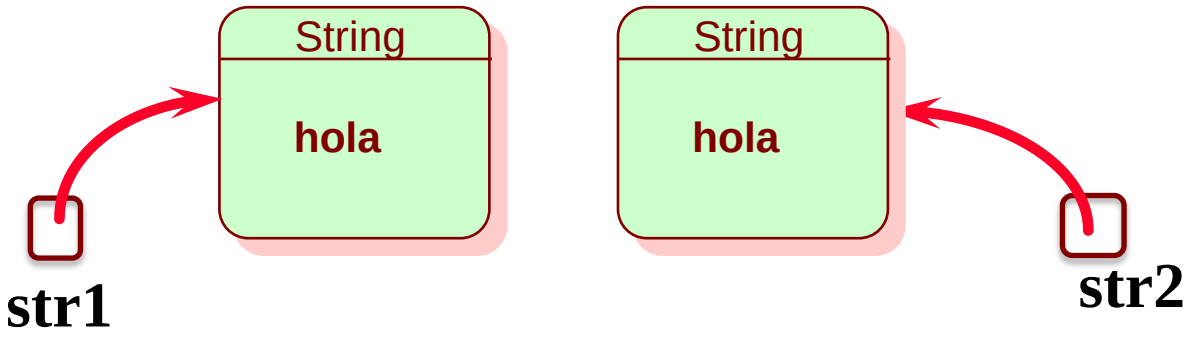
```
boolean equals(String str)
 // true o false

boolean equalsIgnoreCase(String str)
 // true o false, sin tener en cuenta may. y min.

int compareTo(String str)
 // negativo, 0, positivo

int compareToIgnoreCase(String str)
 // negativo, 0, positivo, sin tener en cuenta may. y min.
```

- La clase **`StringBuilder`** no los tiene definidos. Pero se puede usar el método **`toString()`** para obtener un **`String`** de un **`StringBuilder`** y poder comparar.

Comparación

¡Cuidado!

- No debemos comparar dos variables **`String`** o **`StringBuilder`** utilizando el operador == que se utiliza para tipos básicos.
- Si lo hiciéramos, estaríamos comparando referencias y no objetos.
- A veces puede funcionar igual por las optimizaciones del compilador



`str1 == str2`

Evalúa a **`false`**

`str1.equals(str2)`

Evalúa a **`true`**

Paso de parámetros

- Tanto los objetos **`String`** como los **`StringBuilder`** pueden pasarse como parámetros en la invocación a métodos. Un método también puede devolver un objeto de tipo **`String`** o **`StringBuilder`**.

- Recordemos que el paso de parámetros en Java es "Por Valor", esto es, se realiza una copia del valor del parámetro real en el parámetro formal correspondiente.

Paso de parámetros

- Con los tipos básicos esto implica una copia del valor almacenado.

```
int x = 3;
…
método(x);
```

```
private static void método(int y) {
  …
}
```

x

y

Paso de parámetros

- Cualquier modificación en el parámetro formal dentro del método NO afecta al parámetro real

```
int x = 3;
…
método(x);
```

```
private static void método(int y) {
  …
}
```

x

y

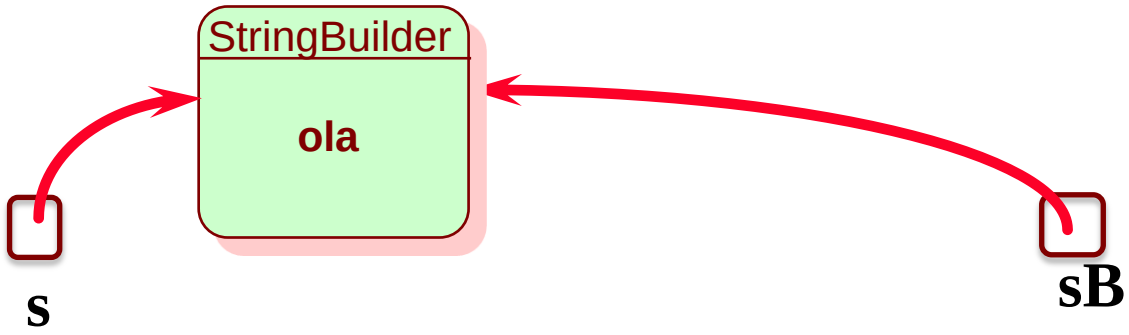
```
Y = 4;
```

Paso de parámetros

- En cambio, cuando tratamos con objetos, la copia es de la referencia, no del objeto referenciado.

```
StringBuilder s =
 new StringBuilder("ola");
…
método(s);
```

```
private static void método(StringBuilder sB){
  …
}
```

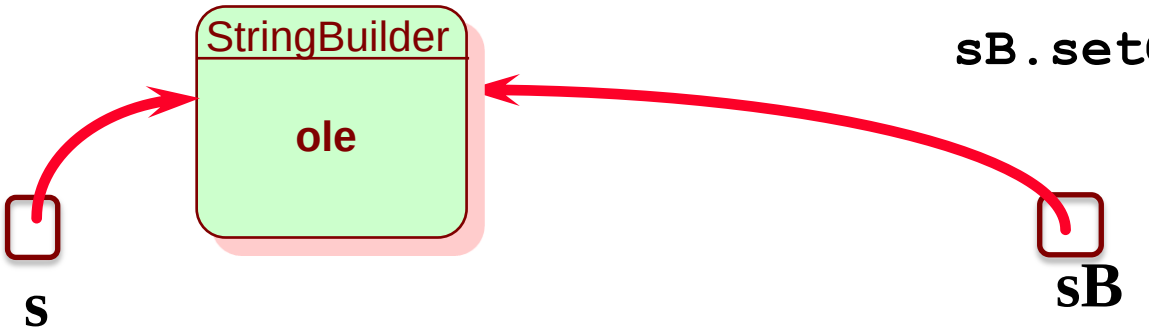


Paso de parámetros

- Cualquier modificación en el parámetro formal dentro del método SÍ afecta al objeto del parámetro real (siempre que sea modificable)

```
StringBuilder s =
 new StringBuilder("ola");
…
método(s);
```

```
private static void método(StringBuilder sB){
  …
}
```




```
sB.setCharAt(2,'e');
```

UNIVERSIDAD DE MÁLAGA

Ejemplos

Ejemplo 1.

Un palíndromo es una palabra que expresa lo mismo leída de izquierda a derecha que de derecha a izquierda.

solos, ana, kayak, narran, acurruca

Se necesita un algoritmo que lea una palabra y determine si es o no un palíndromo.

UNIVERSIDAD DE MÁLAGA

Ejemplos

```
public static void main(String[] args) {
  Scanner teclado = new Scanner(System.in);
  String palabra;

  System.out.print("Introduzca palabra: ");
  palabra = teclado.next();

  if (esPalindromo(palabra)) {
    System.out.println("SI es palíndromo");
  } else {
    System.out.println("NO es palíndromo");
  }
  teclado.close();
}
```

UNIVERSIDAD DE MÁLAGA

Ejemplos

```
public static void main(String[] args) {
  Scanner teclado = new Scanner(System.in);
  String palabra;

  System.out
  palabra = t

  if (esPalin
    System.ou
  } else {
    System.ou
  }
  teclado.clo
}
```

```
private static boolean esPalindromo(String palabra) {
  int i,j;

  i = 0;
  j = palabra.length() - 1;
  while ((i < j) && (palabra.charAt(i) == palabra.charAt(j))) {
    i++;
    j--;
  }
  return (i >= j);
}
```

Ejemplos

Ejemplo 2:

Programa que lee una palabra (formada por letras minúsculas), y escribe su plural según las siguientes reglas:

- Si acaba en vocal se le añade la letra 's'.
- Si acaba en consonante se le añaden las letras 'e‘ y ‘s'. Si la consonante es la letra 'z', se sustituye por la letra 'c‘.
- Suponemos que la palabra introducida es correcta.

UNIVERSIDAD DE MÁLAGA

Ejemplos

```
public static void main(String[] args) {
  Scanner teclado = new Scanner(System.in);
  StringBuilder palabra;

  System.out.print("Introduzca palabra: ");
  palabra = new StringBuilder(teclado.next());

  construirPlural(palabra);

  System.out.println("Su plural es: " + palabra);
  teclado.close();
}
```

Ejemplos

```
public static vo
  Scanner teclad
  StringBuilder

  System.out.pri
  palabra = new

  construirPlura

  System.out.pri
  teclado.close(
}
```

```
private static void construirPlural(StringBuilder palabra) {
  if (palabra.length() > 0) {
    if (esVocal(palabra.charAt(palabra.length()- 1))) {
      palabra.append('s');
    } else {
      if (palabra.charAt(palabra.length()- 1) == 'z') {
         palabra.setCharAt(palabra.length()- 1,'c');
      }
      palabra.append("es");
    }
  }
}
```

```
private static bool esVocal(char car) {
   return car == 'a'|| car == 'e' || car == 'i' || car == 'o' || car == 'u';
}
```

UNIVERSIDAD DE MÁLAGA

Arrays. Justificación

- Una empresa tiene 20 agentes (identificados por números del 1 al 20)
- Los agentes cobran comisión sobre la parte de sus ventas que excede los 2/3 del promedio de ventas del grupo.
- Se necesita un algoritmo que lea el valor de las operaciones comerciales de cada agente e imprima el número de identificación de aquellos que deban percibir comisión así como el valor correspondiente a sus ventas.
- Consideraciones:
    - Se necesita almacenar durante la ejecución del programa los valores de las ventas de cada agente: para el cálculo del promedio y para la comparación de niveles. Por tanto, **necesitamos 20 variables.**
    - Se hará un **procesamiento similar sobre los datos de cada agente.**

UNIVERSIDAD DE MÁLAGA

Arrays. Justificación

```
import java.util.Scanner;

public class Principal {

  static final double PORCION = 2.0/3.0;

  public static void main(String[] args) {
    Scanner teclado = new Scanner(System.in);

    double ventas1,ventas2,ventas3, ventas4, ..., ventas20;
    double suma, umbral;
```

Arrays. Justificación

```
import java.util.Scanner;

pub
    suma = 0;
    System.out.print("Introduzca  ventas agente 1: ");
    ventas1 = teclado.nextDouble();
    suma += ventas1;
    System.out.print("Introduzca ventas agente 2: ");
    ventas2 = teclado.nextDouble();
    suma += ventas2;
    System.out.print("Introduzca ventas agente 3: ");
    ventas3 = teclado.nextDouble();
    suma += ventas3;
    System.out.print("Introduzca ventas agente 4: ");
    ventas4 = teclado.nextDouble();
    suma += ventas4;
    ...
    System.out.print("Introduzca ventas agente 20: ");
    ventas20 = teclado.nextDouble();
    suma += ventas20;

    umbral= PORCION *(suma/20);
```

Arrays. Justificación

```
import java.ut

pub

    suma =
    System.
    ventas1
    suma +=
    System.
    ventas2
    suma +=
    System.
    ventas3
    suma +=
    System.
    ventas4
    suma +=
    ...
    System.
    ventas2
    suma +=

    umbral=
```

```
    if (ventas1>umbral)
    {
        System.out.println("Ventas Agente 1: " + ventas1);
    }
    if (ventas2>umbral)
    {
        System.out.println("Ventas Agente 2: " + ventas2);
    }
    if (ventas3>umbral)
    {
        System.out.println("Ventas Agente 3: " + ventas3);
    }
    if (ventas4>umbral)
    {
        System.out.println("Ventas Agente 4: " + ventas4);
    }
    ...
    if (ventas20>umbral)
    {
        System.out.println("Ventas Agente 20: " + ventas20);
    }
    teclado.close();
}
```

Arrays. Justificación

¿Qué ocurriría si en lugar de 20 tenemos 200 agentes ?

Necesitaríamos:

- 200 variables.
- 200 lecturas y sumas.
- Habría 200 sentencias `if` en el código.

¿Qué ocurriría si el número de agentes varía a lo largo de la vida útil del programa ?

Necesitaríamos modificar el código continuamente

Arrays. Justificación

¿Qué ocurriría si en lugar de 20 tenemos 200 agentes ?

Necesitaríamos:

- 200 variables.
- 200 lecturas y sumas.
- Habría 200 sentencias `if` en el código.

SOLUCIÓN: Usar un nuevo tipo de datos.
El tipo ARRAY

¿Qué ocurriría si el número de agentes varía a lo largo de la vida útil del programa ?

Necesitaríamos modificar el código continuamente

Arrays. Conceptos básicos

- Un array es un objeto que almacena una colección de un **número fijo de elementos** pertenecientes a un **mismo tipo de datos**, llamado tipo Base.
- Sintaxis:
  - Declaración

```
int    [] listaEnteros;
String [] listaCadenas;
```

  - Creación e Inicialización (se puede hacer en el momento de la declaración)

```
listaEnteros = new int[10];
listaCadenas  = new String[t];
char [] vocales = {'a','e','i','o','u'};
```

- La longitud se guarda en una variable **length** que sólo se puede consultar.
- Los elementos de un array se inicializan con los valores por defecto (**`false '\u0000' 0 +0.0F +0.0D null`**) (excepto al usar arrays literales)

Arrays. Conceptos básicos

componentes array nombre

- Un array es un objeto que almacena una colección de un **número fijo de elementos** pertenecientes a un **mismo tipo de datos**, llamado tipo Base.
- Sintaxis:
  - Declaración

```
int    [] listaEnteros;
String [] listaCadenas;
```

componentes tamaño

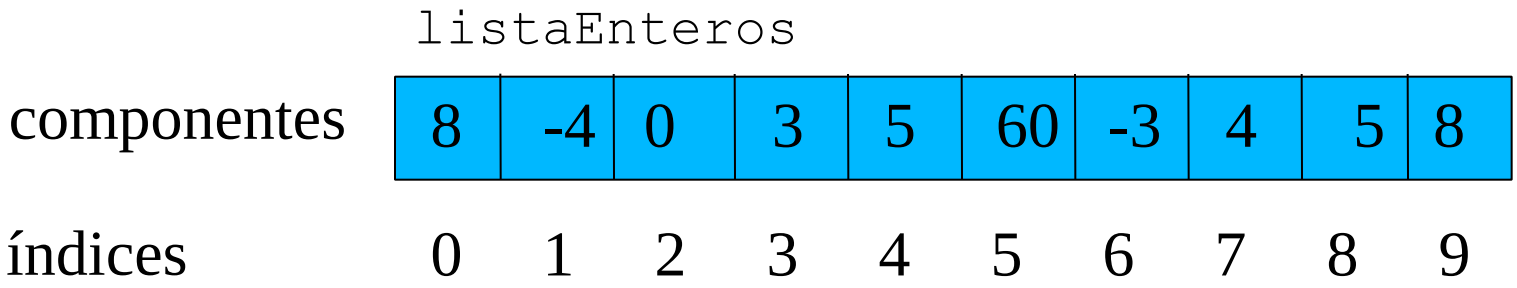
  - Creación e Inicialización (se puede hacer en el momento de la declaración)

```
listaEnteros = new int[10];
listaCadenas  = new String[t];
char [] vocales = {'a','e','i','o','u'};
```

array literal

- La longitud se guarda en una variable l[illegible] puede consultar.
- Los elementos de un array se inicializan con los valores por defecto (**`false '\u0000' 0 +0.0F +0.0D null`**) (excepto al usar arrays literales)

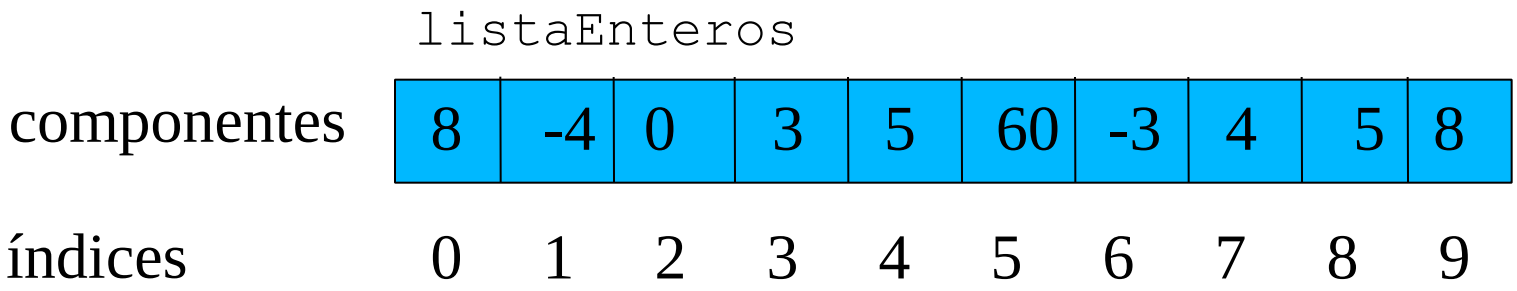
Arrays. Conceptos básicos



- El primer elemento de un array ocupa la posición 0
- El último elemento ocupa la posición Número de Elementos–1

```
listaEnteros.length - 1
```

Arrays. Conceptos básicos



- La operación básica al usar arrays es el **acceso a un elemento** individual del array. Se usará el **operador [ ]**

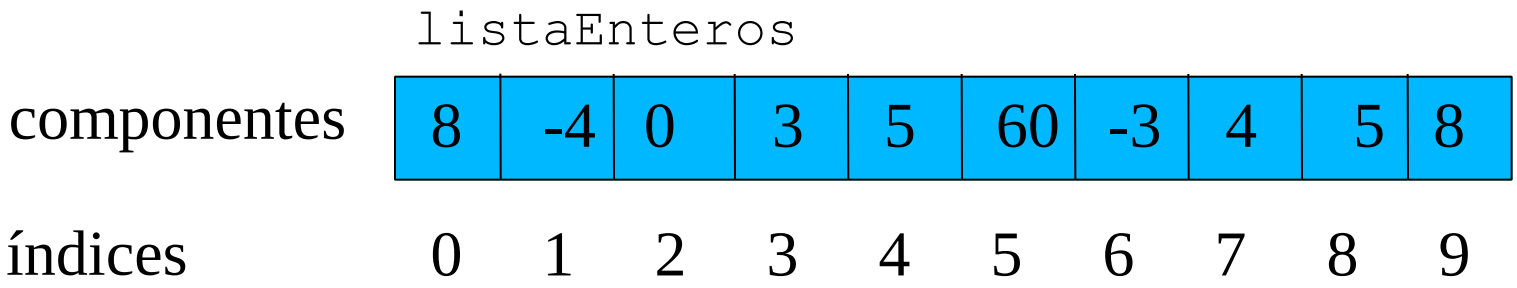
  Así: `listaEnteros[4]` vale 5

  `listaEnteros[1]` vale −4

- El elemento accedido puede ser utilizado como cualquier variable del tipo base del array, por ejemplo:

```
listaEnteros[2]++;
System.out.print(listaEnteros[7]);
```

Arrays. Conceptos básicos



- Es posible que el índice de acceso al elemento sea una expresión que se evalúe a un número entero. Por ejemplo:

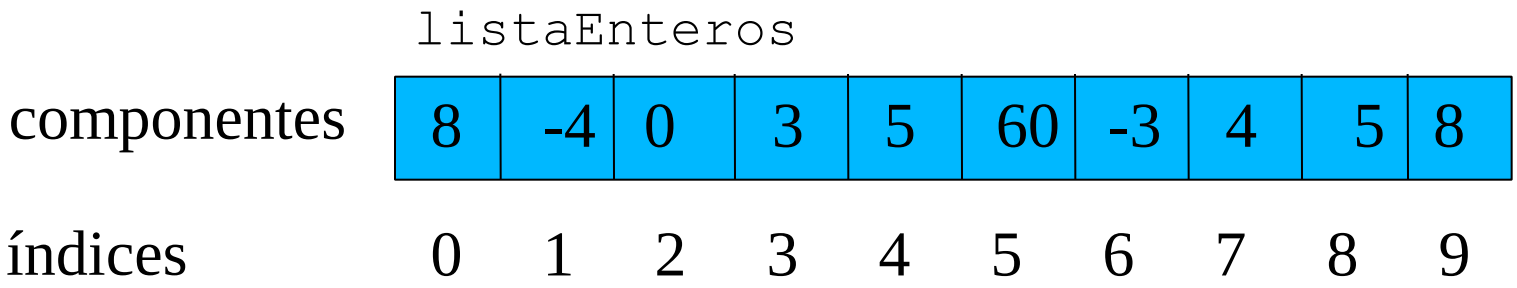
```
listaEnteros[i], listaEnteros[i*2], ...
```

```
for (int i = 0; i < listaEnteros.length; i++) {
     listaEnteros[i] = teclado.nextInt();
}
```

- Si se intenta acceder a una posición no válida el sistema lanza una excepción:

```
ArrayIndexOutOfBoundsException
```

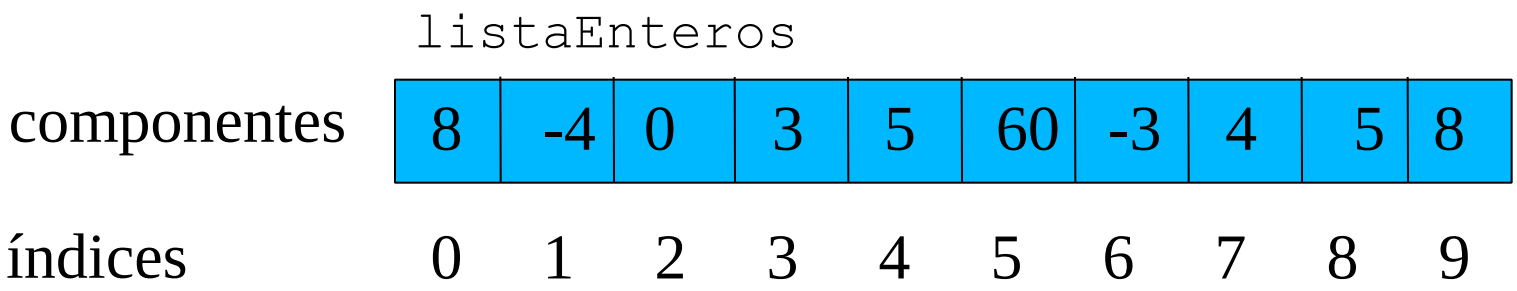
Arrays. Conceptos básicos



- Para recorrer un array se pueden utilizar las estructuras iterativas estudiadas (while, do while, for)
- Además, es posible utilizar una sintaxis alternativa (pero no se puede modificar el contenido de las celdas), denominada "for-each"

```
for (int e : listaEnteros) {
      System.out.println(e);
}
```

Arrays. Conceptos básicos



- Para mostrar por pantalla el contenido de un array, además de hacerlo elemento a elemento con un bucle, también se puede utilizar el método **toString** de la clase **Arrays** (de java.util)

```
System.out.println(Arrays.toString(listaEnteros));
```

mostrará por pantalla:

[8,-4,0,3,5,60,-3,4,5,8]

- Para leer un array de teclado no se puede hacer como un todo (hay que hacerlo elemento a elemento con un bucle)

Comparación

- La clase **Arrays** (de java.util) tiene un método para comprobar si dos arrays son iguales:

  ```
  boolean equals(Tipo[] a1, Tipo[] a2)
  ```

- También tiene un método para comprobar si un array contiene un determinado elemento (si el array está ordenado):

  ```
  int binarySearch(Tipo[] a, Tipo valor)
  ```

- También tiene un método para ordenar un array:

  ```
  void sort(Tipo[] a)
  ```

- Para utilizarlos se debe poner delante **`Arrays.`** Ej:
  - `if (Arrays.equals(array1,array2)) ...`
  - `indice = Arrays.binarySearch(array,v);`
  - `Arrays.sort(array)`

Asignación y Paso de Parámentros

- Una variable de tipo array puede ser asignada a otra, siempre que tengan el mismo tipo base.

- De igual forma, una variable de tipo array puede utilizarse como parámetro real en la invocación a un método, siempre que el parámetro formal sea un array con el mismo tipo base.

- Un método también puede devolver un array.

Asignación y Paso de Parámentros

- Al tratarse de objetos, hay que tener en cuenta las mismas consideraciones que se explicaron con las cadenas de caracteres (una variable almacena realmente una referencia al objeto en sí).
- Los arrays son objetos modificables (mutables).

Asignación y Paso de Parámentros

- El método **`main`** de la Clase Distinguida recibe como parámetro un array de **`String`**:

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

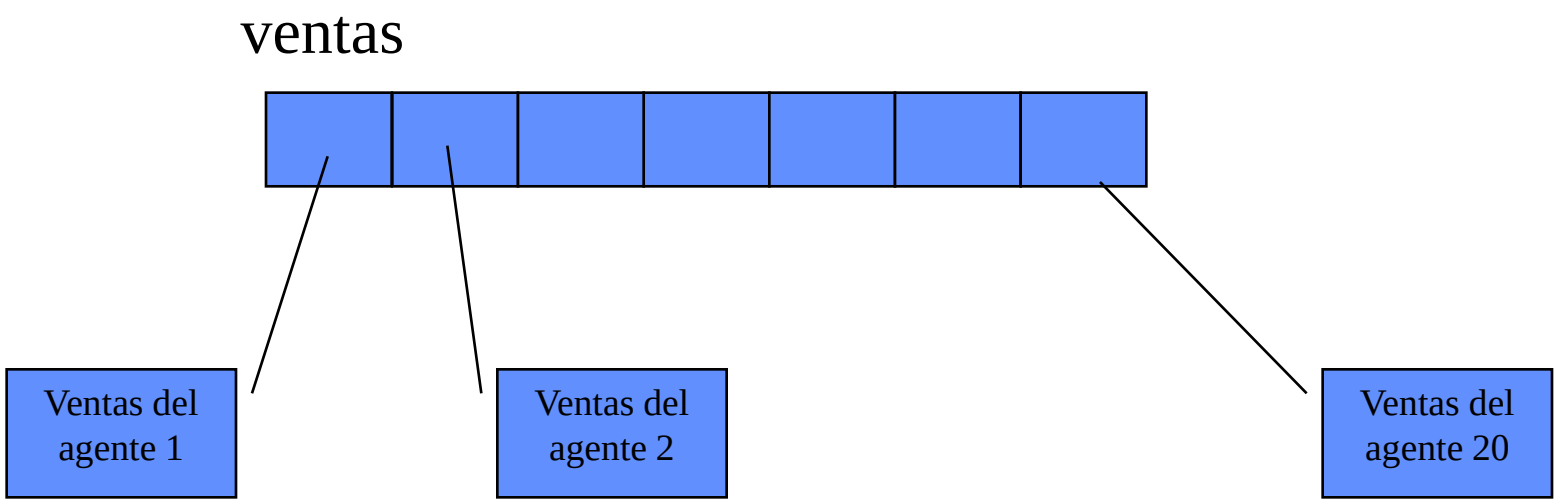
- ¿Cómo se pasan argumentos en IntelliJ IDEA?
- ¿Cómo se convierten a int, double, …?

```
public class Principal {
    public static void main(String[] args) {
        int x = Integer.parseInt(args[0]);
        double y = Double.parseDouble(args[1]);
          ...
    }
}
```

Ya se verán en el Módulo 2

Ejemplo

- Retomemos el ejemplo de los agentes de ventas.
- La idea de los arrays es almacenar en una misma variable (de tipo estructurado) diferentes componentes **del mismo tipo.**



UNIVERSIDAD DE MÁLAGA

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final double PORCION = 2.0/3.0;
  static final int NUM_AGENTES = 20;

  public static void main(String[] args) {
    double [] ventas = new double[NUM_AGENTES];
    double umbral;
    Scanner teclado = new Scanner(System.in);

    leer(ventas,teclado);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);
    teclado.close();
  }
```

Ejemplo

```
import java.ut
public class P
  static final
  static final

  public stati
    double [] 
    double umbral;
    Scanner teclado = new Scanner(System.in);

    leer(ventas,teclado);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);
    teclado.close();
  }
```

```
private static void leer(double[] ventas, Scanner teclado) {

  for (int i = 0; i < NUM_AGENTES; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) +": ");
    ventas[i] = teclado.nextDouble();
  }

}
```

Ejemplo

```
import java.ut
public class P
  static final
  static final

  public stati
    double [] 
    double umbral;


    leer(ventas);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);

  }
```

```
private static void leer(double[] ventas) {
  Scanner teclado = new Scanner(System.in);
  for (int i = 0; i < NUM_AGENTES; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) +": ");
    ventas[i] = teclado.nextDouble();
  }
  teclado.close();
}
```

Ejemplo

```
import java.ut
public class P
  static final
  static final

  public stati
    double [] 
    double umbral;

    leer(ventas);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);

  }
```

```
private static void leer(double[] ventas) {
  Scanner teclado = new Scanner(System.in);
  for (int i = 0; i < ventas.length; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) +": ");
    ventas[i] = teclado.nextDouble();
  }
  teclado.close();
}
```

Ejemplo

```
import java.ut

public class P
  static final
  static final

  public stati
    double [] ventas = new
    double umbral;

    leer(ventas);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);

  }
```

```
private static void leer(double[] ventas) {
  Scanner teclado = new Scanner(System.in);
  for (int i = 0; i < ventas.length; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) + ": ");
    ventas[
  }
  teclado.
}
```

```
private static double Media(double[] ventas) {
  double suma = 0.0;
  for (int i = 0; i < ventas.length; i++) {
    suma += ventas[i];
  }
  return suma/ventas.length;
}
```

Ejemplo

```
import java.ut

public class P
  static final
  static final

  public stati
    double [] ventas = new
    double umbral;

    leer(ventas);
    umbral= PORCION * Media(ventas);
    imprimir(ventas, umbral);

  }
```

```
private static void leer(double[] ventas) {
  Scanner teclado = new Scanner(System.in);
  for (int i = 0; i < ventas.length; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) + ": ");
    ventas[
  }
  teclado.
}
```

```
private static double Media(double[] ventas) {
  double suma = 0.0;
  for (double v : ventas){
    suma += v;
  }
  return suma/ventas.length;
}
```

UNIVERSIDAD DE MÁLAGA

Ejemplo

```
import java.ut

public class P
  static final
  static final

  public stati
    double [] ventas = new
    double

    Leer(v
    umbral
    imprim

  }
```

```
private static void leer(double[] ventas) {
  Scanner teclado = new Scanner(System.in);
  for (int i = 0; i < ventas.length; i++) {
    System.out.print("Introduzca ventas del agente " + (i+1) +": ");
    ventas[
  }
  teclado.
}
```

```
private static double Media(double[] ventas) {
  double suma = 0.0;
  for (double v : ventas){
    suma += v;
  }
```

```
private static void imprimir(double[] ventas, double umbral) {
  for (int i = 0; i < ventas ventas.length; i++){
    if (ventas[i] > umbral) {
        System.out.println("Ventas del Agente " + (i+1) +": " +ventas[i]);
    }
  }
}
```

UNIVERSIDAD DE MÁLAGA

Copia de Arrays

- Para la copia eficiente de componentes de un array a otro Java tiene el método **`arraycopy`** de la clase **`System`**.

```
public static
   void arraycopy(Object arrayOrigen,
                  int primÍndiceOrigen,
                  Object arrayDestino,
                  int primÍndiceDestino,
                  int númeroDeCompCopia)
```

UNIVERSIDAD DE MÁLAGA

Copia de Arrays

```
char[] arrayOrigen =
  {'d','e','s','c','a','f','e','i','n','a','d','o'};
char[] arrayDestino = new char[7];

System.arraycopy(arrayOrigen, 3, arrayDestino, 0, 7);
```

`arrayDestino` contendrá: `{'c','a','f','e','i','n','a'}`

Duplicación de Arrays

- Se puede utilizar el método **`copyOf`** de la clase **`Arrays`** (de java.util).
- Si la nueva longitud especificada es diferente de la del array original:
  - Elimina elementos si es menor
  - Añade valores por defecto si es mayor

```
public static
   TipoBase[] copyOf(TipoBase[] arrayOriginal,
                     int nuevaLongitud)
```

Duplicación de Arrays

```
int[] arrayOrigen = {12,4,8,3,5,203,28};
```

```
int[] arrayDestino = Arrays.copyOf(arrayOrigen,3);
```

`arrayDestino` contendrá: `{12,4,8}`

```
int[] arrayDestino = Arrays.copyOf(arrayOrigen,10);
```

`arrayDestino` contendrá: `{12,4,8,3,5,203,28,0,0,0}`

UNIVERSIDAD DE MÁLAGA

Ejemplo

```
. . .
numElem = 0;
System.out.print("Introduzca secuencia de números enteros (0 para terminar): ");
elem = teclado.nextInt();
while (elem != 0) {
        if (numElem == a.length) {
                a = Arrays.copyOf(a,numElem*2);
        }
        a[numElem] = elem;
        numElem++;
        elem = teclado.nextInt();
}
a = Arrays.copyOf(a,numElem);
. . .
```

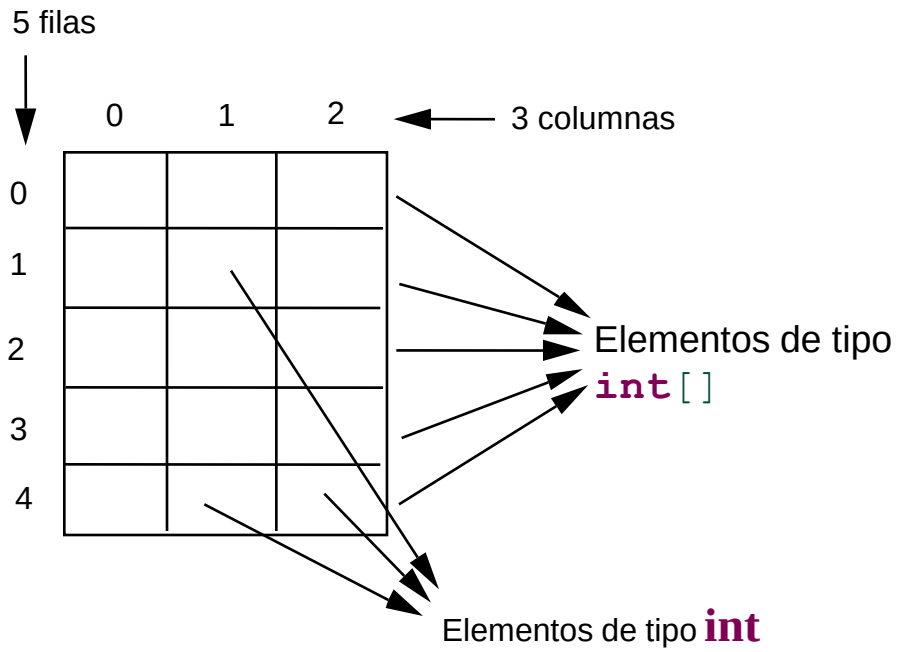
Arrays multidimensionales

- El tipo base de un array puede ser tanto simple como estructurado.
- Ya hemos visto ejemplos de arrays de cadenas de caracteres (parámetro de **`main`**).
- Si el tipo base de un array es otro array, conseguimos un array bidimensional.

Arrays multidimensionales

- Ejemplo:

```
int [][] m = new int [5][3];
```



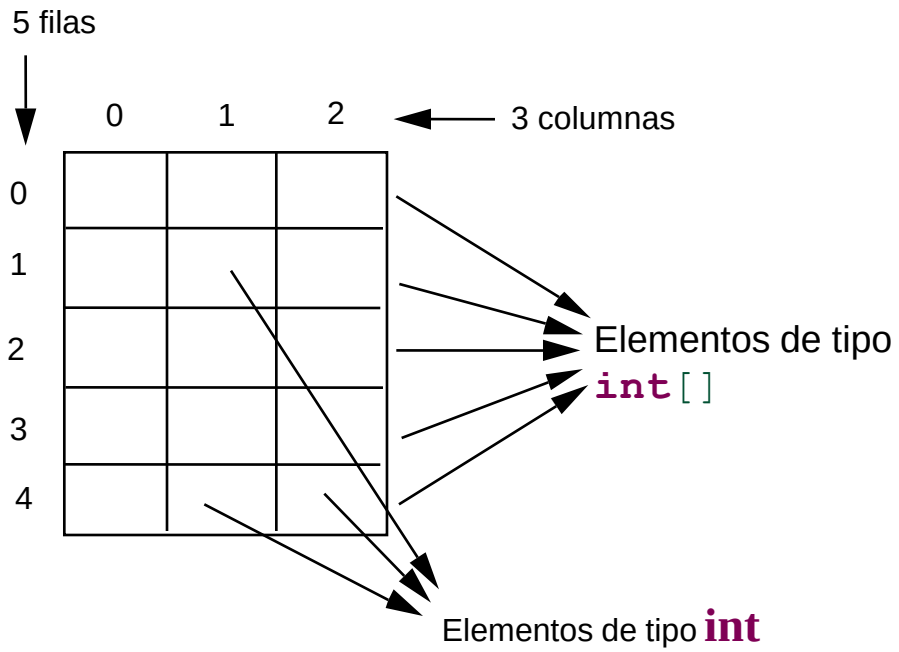
- Para acceder a un elemento de m:

```
m[i]   obtendremos un array de 3 enteros
```

Arrays multidimensionales

- Ejemplo:

```
int [][] m = new int [5][3];
```



- Para acceder a un elemento de m:

```
m[i][j] accederemos al elemento de índice j del
        array obtenido al acceder al índice i del
        array m (un int)
```

Arrays multidimensionales

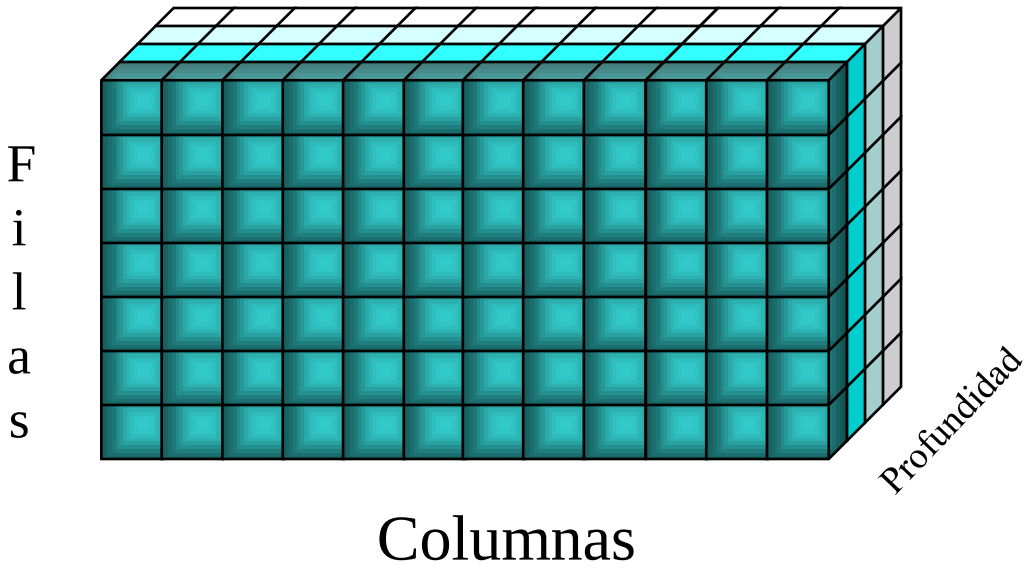
- Al crear un array bidimensional con el operador **`new`** sólo hace falta fijar el tamaño de la primera dimensión, dejando la segunda para más adelante (aunque esto no es lo más usual)

```
int[][] matriz = new int[4][];
for (int i = 0; i < matriz.length; i++) {
   matriz[i] = new int[i + 5];
   for (int j = 0; j < matriz[i].length; j++) {
         matriz[i][j] = i + j;
   }
}
```

Arrays multidimensionales

- También podemos crear arrays de 3 o más dimensiones.

```
double [][][]cubo = new double[4][7][12];
```



UNIVERSIDAD DE MÁLAGA

Ejemplo

- Ejemplo: Algoritmo que lee una matriz 6x8 de enteros (fila a fila), la almacena en un array bidimensional "a", deja los resultados de las sumas de los elementos de cada fila en un vector "b" y las sumas de los elementos de cada columna en un vector "c". Finalmente imprime los 3 arrays con el formato siguiente:

```
a a a a a a a a b
a a a a a a a a b
a a a a a a a a b
a a a a a a a a b
a a a a a a a a b
a a a a a a a a b
c c c c c c c c
```

UNIVERSIDAD DE MÁLAGA

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final int FILAS = 6;
  static final int COLUMNAS = 8;
  public static void main(String[] args) {
    int [][] a = new int[FILAS][COLUMNAS];
    int [] b = new int[FILAS];
    int [] c = new int[COLUMNAS];
    leerMatriz(a);
    for (int fi = 0; fi < FILAS; fi++){
      b[fi]= sumarFila(a[fi]);
    }
    for (int co = 0; co < COLUMNAS; co++){
      c[co]= sumarCol(a,co);
    }
    for (int fi = 0; fi < FILAS; fi++){
      escribirFila(a[fi]);
      System.out.println(b[fi]);
    }
    escribirFila(c);
}
```

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final int FILA
  static final int COLU
  public static void ma
    int [][] a = new in
    int [] b = new int[
    int [] c = new int[
    leerMatriz(a);
    for (int fi = 0; fi
      b[fi]= sumarFila(
    }
    for (int co = 0; co
      c[co]= sumarCol(a,co);
    }
    for (int fi = 0; fi < FILAS; fi++) {
      escribirFila(a[fi]);
      System.out.println(b[fi]);
    }
    escribirFila(c);
}
```

```
private static void leerMatriz(int[][] a) {
  Scanner teclado = new Scanner(System.in);
  System.out.println("Introduzca valores fila a fila:");
  for (int fi = 0; fi < FILAS; fi++) {
    for (int co = 0; co < COLUMNAS; co++) {
      a[fi][co] = teclado.nextInt();
    }
  }
  teclado.close();
}
```

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final int FILA
  static final int COLU
  public static void ma
    int [][] a = new in
    int [] b = new int[
    int [] c = new int[
    leerMatriz(a);
    for (int fi = 0; fi
      b[fi]= sumarFila(
    }
    for (int co = 0; co
      c[co]= sumarCol(a,co);
    }
    for (int fi = 0; fi < FILAS; fi++){
      escribirFila(a[fi]);
      System.out.println(b[fi]);
    }
    escribirFila(c);
}
```

```
private static void leerMatriz(int[][] a) {
  Scanner teclado = new Scanner(System.in);
  Syst
  for
    fo
    }
  }
  teclado.close();
}
```

```
private static void escribirFila(int[] fila) {
  for (int co = 0; co < COLUMNAS; co++){
    System.out.print(fila[co] + " ");
  }
}
```

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final int FILA
  static final int COLU
  public static void ma
    int [][] a = new in
    int [] b = new int[
    int [] c = new int[
    leerMatriz(a);
    for (int fi = 0; fi
      b[fi]= sumarFila(
    }
    for (int co = 0; co
      c[co]= sumarCol(a,co);
    }
    for (int fi = 0; fi < FILAS; fi++){
      escribirFila(a[fi]);
      System.out.println(b[fi]);
    }
    escribirFila(c);
}
```

```
private static void leerMatriz(int[][] a) {
  Scanner teclado = new Scanner(System.in);
  Syst
  for
    fo
    }
  }
  teclad
}
```

```
private static void escribirFila(int[] fila) {
  for (int co = 0; co < COLUMNAS; co++) {
}
```

```
private static int sumarFila(int[] fila) {
  int resultado = 0;

  for (int co = 0; co < COLUMNAS; co++){
    resultado += fila[co];
  }
  return resultado;
}
```

Ejemplo

```
import java.util.Scanner;
public class Principal {
  static final int FILA
  static final int COLU
  public static void ma
    int [][] a = new in
    int [] b = new int[
    int [] c = new int[
    leerMatriz(a);
    for (int fi = 0; fi
      b[fi]= sumarFila(
    }
    for (int co = 0; co
      c[co]= sumarCol(a,co);
    }
    for (int fi = 0; fi < FILAS; fi++)
      escribirFila(a[fi]);
      System.out.println(b[fi]);
    }
    escribirFila(c);
}
```

```
private static void leerMatriz(int[][] a) {
  Scanner teclado = new Scanner(System.in);
  Syst
  for
    fo
      }
  }
  teclad
}
```

```
private static void escribirFila(int[] fila) {
  for (int co = 0; co < COLUMNAS; co++) {
}
```

```
private static int sumarFila(int[] fila) {
  int
  for
    re
  }
  retu
}
```

```
private static int sumarCol(int[][] a, int co) {
  int res=0;

  for (int fi=0; fi < FILAS; fi++) {
    res += a[fi][co];
  }
  return res;
}
```