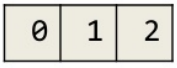
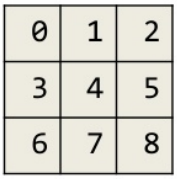
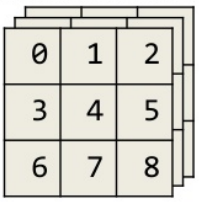
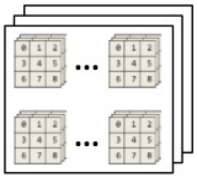
Anexo UD04: Cheatsheet Strings en Java

Dimensions	Example	Terminology
1		Vector
2		Matrix
3		3D Array (3rd order Tensor)
N		ND Array

1. **Introducción**
   1. 1. Construyendo `String`
   1. 2. Operando con Métodos de la clase `String`
   1. 3. Ejemplo de todos los métodos de `String`
   1. 4. Arrays de `String`
   1. 5. Los `String` son inmutables
   1. 6. `String` en Argumentos de Línea de Comandos
   1. 7. Concatenar cadenas en `Java`
      1. 7. 1. Operador `+`
      1. 7. 2. Método `concat()`
      1. 7. 3. Método `append` de `StringBuilder`
      1. 7. 4. ¿Cuándo usar cada uno?
2. **Fuentes de información**

1. Introducción

Desde el punto de vista de la programación diaria, uno de los tipos de datos más importantes de Java es **String**. *String* define y admite cadenas de caracteres. En algunos otros lenguajes de programación, una cadena o string es una matriz o array de caracteres. Este no es el caso con Java. En Java, **los String son objetos**.

En realidad, has estado usando la clase String desde el comienzo del curso, pero no lo sabías. Cuando crea un literal de cadena, en realidad está creando un objeto String. Por ejemplo, en la declaración:

```
System.out.println("En Java, los String son objetos");
```

La clase String es bastante grande, y solo veremos una pequeña parte aquí.

1.1. Construyendo `String`

Puede construir un `String` igual que construye cualquier otro tipo de objeto: utilizando new y llamando al constructor *String*. Por ejemplo:

```
String str = new String("Hola");
```

Esto crea un objeto *String* llamado *str* que contiene la cadena de caracteres “Hola”. También puedes construir una *String* desde otro *String*. Por ejemplo:

```
String str = new String("Hola");
String str2 = new String(str);
```

Después de que esta secuencia se ejecuta, str2 también contendrá la cadena de caracteres “Hola”. Otra forma fácil de crear una cadena se muestra aquí:

```
String str = "Estoy aprendiendo sobre String en JavadesdeCero.";
```

En este caso, *str* se inicializa en la secuencia de caracteres “*Estoy aprendiendo sobre String en JavadesdeCero.*”. Una vez que haya creado un objeto String, puede usarlo en cualquier lugar que permita una cadena entrecomillada. Por ejemplo, puede usar un objeto String como argumento para *println()*, como se muestra en este ejemplo:

```
// Uso de String
class DemoString
{
    public static void main(String args[])
    {
        //Declaración de String de diferentes maneras
        String str1=new String("En Java, los String son objetos");
        String str2=new String("Se construyen de varias maneras");
        String str3=new String(str2);

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);

    }
}
```

La salida del programa se muestra a continuación:

```
En Java, los String son objetos
Se construyen de varias maneras
Se construyen de varias maneras
```

1.2. Operando con Métodos de la clase `String`

La clase String contiene varios métodos que operan en cadenas. Aquí se detallan todos los métodos:

- `int length()`: Devuelve la cantidad de caracteres del String.

```
"Javadesdecero.es".length(); // retorna 16
```

- `Char charAt(int i)`: Devuelve el carácter en el índice *i*.

```
System.out.println("Javadesdecero.es".charAt(3)); // retorna 'a'
```

- `String substring(int i)`: Devuelve la subcadena del i-ésimo carácter de índice al final.

```
"Javadesdecero.es".substring(4); // retorna desdecero.es
```

- `String substring(int i, int j)`: Devuelve la subcadena del índice *i* a *j-1*.

```
"Javadesdecero.es".substring(4,9); // retorna desde
```

- `String concat(String str)`: Concatena la cadena especificada al final de esta cadena.

```
String s1 = "Java";
String s2 = "desdeCero;
String salida = s1.concat(s2); // retorna "JavadesdeCero"
```

- `int indexOf(String s)`: Devuelve el índice dentro de la cadena de la primera aparición de la cadena especificada.

```
String s = "Java desde Cero";
int salida = s.indexOf("Cero"); // retorna 11
```

- `int indexOf(String s, int i)`: Devuelve el índice dentro de la cadena de la primera aparición de la cadena especificada, comenzando en el índice especificado.

```
String s = "Java desde Cero";
int salida = s.indexOf('a',2); //retorna 3
```

- `int lastIndexOf(int ch)`: Devuelve el índice dentro de la cadena de la última aparición de la cadena especificada.

```
String s = "Java desde Cero";
int salida = s.lastIndexOf('a'); // retorna 3
```

- `boolean equals(Objeto otroObjeto)`: Compara este String con el objeto especificado.

```
Boolean salida = "Java".equals("Java"); // retorna true
Boolean salida = "Java".equals("java"); // retorna false
```

- `boolean equalsIgnoreCase(String otroString)`: Compares string to another string, ignoring case considerations.

```
Boolean salida= "Java".equalsIgnoreCase("Java"); // retorna true
Boolean salida = "Java".equalsIgnoreCase("java"); // retorna true
```

- `int compareTo(String otroString)`: Compara dos cadenas lexicográficamente.

```java
int salida = s1.compareTo(s2); // donde s1 y s2 son strings que se comparan
/*
 Esto devuelve la diferencia s1-s2. Si:
    salida < 0 // s1 es menor que s2
    salida = 0 // s1 y s2 son iguales
    salida > 0 // s1 es mayor que s2
 */
```

- `int compareToIgnoreCase(String otroString)`: Compara dos cadenas lexicográficamente, ignorando las consideraciones *case*.

```java
int salida = s1.compareToIgnoreCase(s2); // donde s1 y s2 son strings que se comparan
/*
 Esto devuelve la diferencia s1-s2. Si:
    salida < 0 // s1 es menor que s2
    salida = 0 // s1 y s2 son iguales
    salida > 0 // s1 es mayor que s2
*/
```

Importante En este caso, no considerará el *case* de una letra (ignorará si está en mayúscula o minúscula).

- `String toLowerCase()`: Convierte todos los caracteres de String a minúsculas.

```java
String palabra1 = "HoLa";
String palabra2 = palabra1.toLowerCase(); // retorna "hola"
```

- `String toUpperCase()`: Convierte todos los caracteres de String a mayúsculas.

```java
String palabra1 = "HoLa";
String palabra2 = palabra1.toUpperCase(); // retorna "HOLA"
```

- `String trim()`: Devuelve la copia de la cadena, eliminando espacios en blanco en ambos extremos. No afecta los espacios en blanco en el medio.

```java
String palabra1 = " Java desde Cero ";
String palabra2 = palabra1.trim(); // retorna "Java desde Cero"
```

- `String replace(char oldChar, char newChar)`: Devuelve una nueva cadena al reemplazar todas las ocurrencias de oldChar con newChar.

```java
String palabra1 = "yavadesdecero";
String palabra2 = palabra1.replace('y' ,'j'); //retorna javadesdecero
```

Ejemplo palabra1 sigue siendo yavadesdecero y palabra2, javadesdecero

- `String replaceAll(String regex, String replacement)`: devuelve una cadena que reemplaza toda la secuencia de caracteres que coinciden con la expresión regular `regex` por la cadena de reemplazo `replacement`.

```java
String str = "Ejemplo con espacios en blanco y tabs";
String str2 = str.replaceAll("\\s", ""); //retorna Ejemploconespaciosenblancoytabs
```

Ampliación Otras expresiones regulares (entre otras muchísimas):

- `\w` Cualquier cosa que sea un carácter de palabra
- `\W` Cualquier cosa que no sea un carácter de palabra (incluida la puntuación, etc.)
- `\s` Cualquier cosa que sea un carácter de espacio (incluido el espacio, los caracteres de tabulación, etc.)

- `\S` Cualquier cosa que no sea un carácter de espacio (incluidas letras y números, así como puntuación, etc.)

Debe escapar de la barra invertida si desea que `\s` alcance el motor de expresiones regulares, lo que da como resultado `\\s`).

Más información sobre expresiones regulares en java: https://www.vogella.com/tutorials/JavaRegularExpressions/article.html

1.3. Ejemplo de todos los métodos de `String`

```java
// Código Java para ilustrar diferentes constructores y métodos
// de la clase String.

class DemoMetodosString
{
    public static void main (String[] args)
    {
        String s= "JavadesdeCero";
        // o String s= new String ("JavadesdeCero");

        // Devuelve la cantidad de caracteres en la Cadena.
        System.out.println("String length = " + s.length());

        // Devuelve el carácter en el índice i.
        System.out.println("Character at 3rd position = "
                + s.charAt(3));

        // Devuelve la subcadena del carácter índice i-ésimo
        // al final de la cadena
        System.out.println("Substring " + s.substring(3));

        // Devuelve la subcadena del índice i a j-1.
        System.out.println("Substring  = " + s.substring(2,5));

        // Concatena string2 hasta el final de string1.
        String s1 = "Java";
        String s2 = "desdeCero";
        System.out.println("String concatenado  = " +
                s1.concat(s2));

        // Devuelve el índice dentro de la cadena de
        // la primera aparición de la cadena especificada.
        String s4 = "Java desde Cero";
        System.out.println("Índice de Cero: " +
                s4.indexOf("Cero"));

        // Devuelve el índice dentro de la cadena de
        // la primera aparición de la cadena especificada,
        // comenzando en el índice especificado.
        System.out.println("Índice de a  = " +
                s4.indexOf('a',3));

        // Comprobando la igualdad de cadenas
        Boolean out = "Java".equals("java");
        System.out.println("Comprobando la igualdad: " + out);
        out = "Java".equals("Java");
        System.out.println("Comprobando la igualdad: " + out);

        out = "Java".equalsIgnoreCase("jaVA ");
        System.out.println("Comprobando la igualdad: " + out);

        int out1 = s1.compareTo(s2);
        System.out.println("Si s1 = s2: " + out1);

        // Conversión de cases
        String palabra1 = "JavadesdeCero";
        System.out.println("Cambiando a minúsculas: " +
                palabra1.toLowerCase());

        // Conversión de cases
        String palabra2 = "JavadesdeCero";
```

```
        System.out.println("Cambiando a MAYÚSCULAS: " +
                palabra1.toUpperCase());

        // Recortando la palabra
        String word4 = " JavadesdeCero ";
        System.out.println("Recortando la palabra: " + word4.trim());

        // Reemplazar caracteres
        String str1 = "YavadesdeCero";
        System.out.println("String Original: " + str1);
        String str2 = "YavadesdeCero".replace('Y' ,'J') ;
        System.out.println("Reemplazando Y por J -> " + str2);

        // Reemplazar todos los carácteres
        String strAll = "Ejemplo con espacios en blanco y tabs";
        System.out.println("String Original: " + strAll);
        String strAll2 = strAll.replaceAll("\\s", "");
        System.out.println("Eliminando todos los espacios en blanco -> " + strAll2);
    }
}
```

Salida:

```
String length = 13
Character at 3rd position = a
Substring adesdeCero
Substring = vad
String concatenado = JavadesdeCero
Índice de Cero: 11
Índice de a = 3
Comprobando la igualdad: false
Comprobando la igualdad: true
Comprobando la igualdad: false
Si s1 = s2: -26
Cambiando a minúsculas: javadesdecero
Cambiando a MAYÚSCULAS: JAVADESDECERO
Recortando la palabra: JavadesdeCero
String Original: YavadesdeCero
Reemplazando Y por J -> JavadesdeCero
String Original: Ejemplo con espacios en blanco y tabs
Eliminando todos los espacios en blanco -> Ejemploconespaciosenblancoytabs
```

1.4. Arrays de `String`

Al igual que cualquier otro tipo de datos, los String se pueden ensamblar en arrays. Por ejemplo:

```
// Demostrando arrays de String
class StringArray
{
    public static void main (String[] args)
    {
       String str[]={"Java", "desde","Cero"};

        System.out.println("Array Original: ");
        for (String s : str)
            System.out.print(s+ "");
        System.out.println("\n");

        //Cambiando un String
        str[1]="Curso";
        str[2]="Online";

        System.out.println("Array Modificado: ");
        for (String s : str)
            System.out.print(s+ "");
        System.out.println("\n");
    }
}
```

Se muestra el resultado de este programa:

```
Array Original:
JavadesdeCero

Array Modificado:
JavaCursoOnline
```

1.5. Los `String` son inmutables

El contenido de un objeto String es inmutable. Es decir, una vez creada, la secuencia de caracteres que compone la cadena **no se puede modificar**. Esta restricción permite a Java implementar cadenas de manera más eficiente. Aunque esto probablemente suene como un serio inconveniente, no lo es.

Cuando necesite una cadena que sea una variación de una que ya existe, simplemente cree una nueva cadena que contenga los cambios deseados. Como los objetos String no utilizados se recolectan de forma automática, ni siquiera tiene que preocuparse por lo que sucede con las cadenas descartadas. Sin embargo, debe quedar claro que las variables de referencia de cadena pueden, por supuesto, cambiar el objeto al que hacen referencia. Es solo que el contenido de un objeto `String` específico no se puede cambiar después de haber sido creado.

Para comprender completamente por qué las cadenas inmutables no son un obstáculo, utilizaremos otro de los métodos de String: `substring()`. El método `substring()` devuelve una nueva cadena que contiene una parte especificada de la cadena invocadora. Como se fabrica un nuevo objeto String que contiene la subcadena, la cadena original no se altera y la regla de inmutabilidad permanece intacta. La forma de `substring( )` que vamos a utilizar se muestra aquí:

```
String substring(int beginIndex, int endIndex)
```

Aquí, `beginIndex` especifica el índice inicial, y `endIndex` especifica el punto de detención. Aquí hay un programa que demuestra el uso de `substring( )` y el principio de cadenas inmutables:

```
// uso de substring()
class SubString {
    public static void main (String[] args){
        String str="Java desde Cero";

        //Construyendo un substring
        String substr=str.substring(5,15);

        System.out.println("str: "+str);
        System.out.println("substr: "+substr);
    }
}
```

Salida:

```
str: Java desde Cero
substr: desde Cero
```

Como puede ver, la cadena original `str` no se modifica, y `substr` contiene la subcadena.

1.6. `String` en Argumentos de Línea de Comandos

Ahora que conoce la clase String, puede comprender el parámetro *args* en main() que ha estado en cada programa mostrado hasta ahora. Muchos programas aceptan lo que se llaman **argumentos de línea de comandos**. Un argumento de línea de comandos es la información que sigue directamente el nombre del programa en la línea de comando cuando se ejecuta.

Para acceder a los argumentos de la línea de comandos dentro de un programa Java es bastante fácil: se almacenan como cadenas en la matriz String pasada a `main()`. Por ejemplo, el siguiente programa muestra todos los argumentos de línea de comandos con los que se llama:

```java
// Mostrando Información de Línea de Comando
class DemoLC{
    public static void main (String[] args){
        System.out.println("Aquí se muestran "+ args.length
                           + " argumentos de línea de comando.");
        System.out.println("Estos son: ");
        for (int i=0; i<args.length; i++){
            System.out.println("arg["+i+"]: "+args);
        }
    }
}
```

Si `DemoLC` se ejecuta de esta manera,

```
java DemoLC uno dos tres
```

verá la siguiente salida:

```
Aquí se muestran 3 argumentos de línea de comando.
Estos son:
arg[0]: uno
arg[1]: dos
arg[2]: tres
```

Observe que el primer argumento se almacena en el índice 0, el segundo argumento se almacena en el índice 1, y así sucesivamente.

Para tener una idea de la forma en que se pueden usar los argumentos de la línea de comandos, considere el siguiente programa. Se necesita un argumento de línea de comandos que especifique el nombre de una persona. Luego busca a través de una matriz bidimensional de cadenas para ese nombre. Si encuentra una coincidencia, muestra el número de teléfono de esa persona.

```java
class Telefono {
    public static void main (String[] args){
        String numeros[][]={{ "Alex", "123-456"},
                            { "Juan", "145-478"},
                            { "Javier", "789-457"},
                            { "Maria", "784-554"}
                            };
        int i;
        if (args.length != 1){
            System.out.println("Ejecute así: java Telefono <nombre>");
        } else {
            for (i = 0; i < numeros.length; i++) {
                System.out.println(numeros + ": " + numeros);
                break;
            }
            if (i == numeros.length){
                System.out.println("Nombre no encontrado.");
            }
        }
    }
}
```

Aquí hay una muestra de ejecución:

```
java Telefono Alex
Alex: 123-456
```

1.7. Concatenar cadenas en `Java`

1.7.1. Operador `+`

```java
String s1 = "Hola,";
String s2 = "cómo estas?";
String s3 = s1 + s2;
System.out.println("String 1: " + s1);
System.out.println("String 2: " + s2);
System.out.println("Cadena resultante: " + s3);
```

salida:

```
String 1: Hola,
String 2: cómo estas?
Cadena resultante: Hola,cómo estas?
```

1.7.2. Método `concat()`

```java
String s1 = "Hola,";
String s2 = "cómo estas?";
String s3 = s1.concat(s2);
System.out.println("String 1: " + s1);
System.out.println("String 2: " + s2);
System.out.println("Cadena resultante: " + s3);
```

salida:

```
String 1: Hola,
String 2: cómo estas?
Cadena resultante: Hola,cómo estas?
```

1.7.3. Método `append` de `StringBuilder`

```java
String s3 = (new StringBuilder()).append("Hola,").append("cómo estas?").toString();
System.out.println("Cadena resultante: " + s3);
```

salida:

```
Cadena resultante: Hola,cómo estas?
```

1.7.4. ¿Cuándo usar cada uno?

Si usas la concatenación de Strings en un bucle, por ejemplo algo similar a esto:

```java
String s = "";
for (int i = 0; i < 100; i++) {
    s += ", " + i;
}
System.out.println(s);
```

Para el caso anterior, lo mejor seria utilitzar el método `append` de `StringBuilder` en lugar del operador `+` porque es mucho más rápido y consume menos memoria.

Versión con append:

```java
StringBuilder s = new StringBuilder();
for (int i = 1; i < 100; i++) {
    s.append(", ").append(i);
}
System.out.println(s);
```

Si solo tienes una sentencia similar a esta:

```java
String s = "1, " + "2, " + "3, " + "4, " ...;
```

Entonces puedes usar el operador `+` sin problemas, porque el compilador usará `StringBuilder` automáticamente.

2. Fuentes de información

- https://javadesdecero.es/clases/string/
- https://guru99.es/java-string-replace-method/
- https://stackoverflow.com/questions/5455794/removing-whitespace-from-strings-in-java