

"Tal vez la gratitud no sea la virtud más importante, pero sí es la madre de todas las demás".

Cicerón

Entre las clases de utilerías de la API de Java veremos las siguientes *String*, *StringBuilder*, y el trabajo con fechas mediante el uso de las APIs *Date*, *GregorianCalendar* y *SimpleDateFormat*. Mas adelante, después de ver los contenedores o *Collections* de Java echaremos un vistazo a la clase *Arrays* que nos ofrecerá funcionalidades tanto para tratar con arrays como con las *Collections*.

String Vs StringBuilder

Venimos usando la clase *String* desde comienzos del curso y, es ahora, una vez que hemos visto la POO, cuando nos encontramos en disposición de estudiar la funcionalidad asociada a este tipo de objetos.

En primer lugar, es conveniente decir que un objeto *String* representa o almacena a una cadena de caracteres. Adicionalmente, la cadena de caracteres almacenada es **immutable**. Esta característica de inmutabilidad, indica que ese valor no se puede cambiar una vez que se ha asignado al objeto tipo *String*. Algún tipo de modificación de la cadena de caracteres asignada a un objeto *String*, no modificará esa cadena de caracteres, sino que se creará otra cadena de caracteres después de realizar la modificación.

```
public class PruebasString{  
    Run | Debug  
    public static void main (String[] args){  
        String str1 = "My name is ";  
        System.out.println("Contenido de str1: " + str1);  
        String str2 = str1.concat("John Doe");  
        System.out.println("El contenido de str1 no varía " + str1);  
        System.out.println("La nueva cadena creada es referenciada por el objeto str2: " + str2);  
    }  
}
```

```
Contenido de str1: My name is  
El contenido de str1 no varía My name is  
La nueva cadena creada es referenciada por el objeto str2: My name is John Doe
```

Este ejemplo demuestra que la concatenación de la cadena "John Doe" al objeto **str1**, no modifica el contenido del objeto **str1**, el cual es inmutable; lo que se hace es crear un nuevo objeto de tipo *String* el cual es referenciado mediante el objeto **str2** y que contiene la concatenación del contenido de **str1** con la cadena "John Doe".

Como podemos suponer, este modo de trabajo de los tipos *String* no resulta eficiente en absoluto cuando se han de modificar las cadenas de caracteres con las que estamos trabajando, ya que se crean cadenas de caracteres auxiliares cada vez que se modifica un objeto *String*. Por ello, cuando se trabaja con cadenas de caracteres que requieren de modificaciones se recomienda trabajar con la clase *StringBuilder*. A continuación, podemos ver el ejemplo anterior utilizando la clase *StringBuilder*:

```
public static void main (String[] args){  
    StringBuilder str1 = new StringBuilder("My name is ");  
    System.out.println("Contenido de str1: " + str1);  
    str1.append("John Doe");  
    System.out.println("El contenido de str1 ha cambiado " + str1);  
}
```

```
Contenido de str1: My name is  
El contenido de str1 ha cambiado: My name is John Doe
```

Notad que el objeto **str1** recoge la modificación de añadir una nueva cadena de caracteres. No es necesario recoger la nueva cadena de caracteres formada en un nuevo objeto de tipo *StringBuilder*, ya que no se crean objetos intermedios.

Vemos a continuación los métodos más relevantes, tanto de la clase *String* como de la clase *StringBuilder*.

String

Existen muchas versiones sobrecargadas del constructor de la clase *String*. Nosotros vamos a ver 3 formas para obtener un objeto de este tipo:

1. Asignación de un literal:

```
String s = "My name is John Doe";
```

2. Creación de un objeto mediante el uso del operador **new**:

```
String s = new String ("My name is John Doe");
```

3. Mediante un array de char o de bytes:

```
char[] vStr = {'M', 'y', ' ', 'n', 'a', 'm', 'e', ' ', 'i', 's', ' ', 'J', 'o', 'h', 'n', ' ', 'D', 'o', 'e'};  
String s = new String (vStr);
```

```
byte[] vByte = {77, 121, 32, 110, 97, 109, 101, 32, 105, 115, 32, 74, 111, 104, 110, 32, 68, 111, 101};  
String s = new String (vByte);
```

Entre las utilidades de la clase *String* nos encontramos:

1. **int length()**: Retorna el número de caracteres de la cadena.

```
System.out.println("My name is John Doe".length()); // Retorna 19
```

2. **Char charAt(int i)**: Retorna el carácter de la posición i.

```
System.out.println("My name is John Doe".charAt(9)); // Retorna el caracter 's'
```

3. **String substring(int i)**: Retorna una nueva cadena de caracteres que representa la cadena de caracteres desde la posición i hasta el final.

```
System.out.println("My name is John Doe".substring(11)); // Retorna la cadena de caracteres "John Doe"
```

4. **String substring(int i, int j)**: Retorna una nueva cadena de caracteres que representa la cadena de caracteres desde la posición i a la posición j.

```
System.out.println("My name is John Doe".substring(11, 15)); // Retorna la cadena de caracteres "John"
```

5. **String concat (String str)**: Retorna una nueva cadena concatenando la cadena del objeto *str* al final del objeto *String* actual con el que se invoca el método *concat*.

```
System.out.println("My name is ".concat("John Doe")); // Retorna la cadena de caracteres "My name is John Doe"
```

6. **int indexOf(String str)**: Retorna la posición en el *String* actual de la primera ocurrencia del *String str*.

```
System.out.println("My name is John Doe".indexOf("John")); // Retorna la posición 11
```

7. **int indexOf(String str, int i)**: Retorna la posición de la cadena de caracteres representada por el parámetro *str* en la cadena actual a partir de la posición i.

```
System.out.println("My name is John Doe".indexOf("o", 13)); // Retorna la posición 17
```

8. **int lastIndexOf(String str)**: Retorna la posición de la cadena de caracteres representada por el parámetro *str* en la cadena actual.

```
System.out.println("My name is John Doe".lastIndexOf("o")); // Retorna la posición 17
```

9. **boolean equals(Object o):** Compara la cadena de caracteres del objeto *String* actual con la cadena de caracteres del objeto *String* que se pasa por parámetro.

```
String str_1 = new String("My name is John Doe");
String str_2 = new String("My name is JOHN DOE");
System.out.println(str_1.equals(str_2)); // Retorna false
System.out.println(str_2.equals("My name is JOHN DOE")); // Retorna true
```

10. **boolean equalsIgnoreCase(String str):** Compara la cadena de caracteres del objeto *String* actual con la cadena de caracteres que se pasa por parámetro. En este caso la comparación se realiza sin considerar mayúsculas o minúsculas.

```
String str_1 = new String("My name is John Doe");
String str_2 = new String("My name is JOHN DOE");
System.out.println(str_1.equalsIgnoreCase(str_2)); // Retorna true
System.out.println(str_2.equalsIgnoreCase("My name is JOHN DOE")); // Retorna true
```

11. **int compareTo(String str):**

Compara la cadena de caracteres del objeto *String* actual con la cadena de caracteres que se pasa por parámetro, retornando:

<0 Si el contenido del objeto actual es menor que la cadena *str* que se pasa por parámetro.

0 Si el contenido del objeto actual es igual al valor de la cadena *str* que se pasa por parámetro.

>0 Si el contenido del objeto actual es mayor que la cadena *str* que se pasa por parámetro.

```
String str_1 = new String("abcd");
String str_2 = new String("efgh");
System.out.println(str_1.compareTo(str_2)); // Retorna <0 ya que str_1 < str_2
System.out.println(str_1.compareTo("abcd")); // Retorna 0 ya que str_1 == "abcd"
System.out.println(str_2.compareTo(str_1)); // Retorna >0 ya que str_2 > que str_1
```

12. **int compareToIgnoreCase(String str):** Compara la cadena de caracteres del objeto *String* actual con la cadena de caracteres que se pasa por parámetro, no teniendo en cuenta si hay mayúsculas o minúsculas en las cadenas de caracteres.

```
String str_1 = new String("abcd");
String str_2 = new String("efgh");
System.out.println(str_1.compareToIgnoreCase("abcd")); // Retorna 0 ya que str_1 == "abcd"
System.out.println(str_1.compareToIgnoreCase("ABCD")); // Retorna 0 ya que str_1 == "ABCD" (ignorando mayúsculas y minúsculas)
System.out.println(str_1.compareToIgnoreCase(str_2)); // Retorna <0 ya que str_1 < str_2
```

13. **String toLowerCase():** Pasa todos los caracteres del objeto actual a minúscula.

```
String str = new String("My name is John Doe");
System.out.println(str.toLowerCase()); // Retorna "my name is john doe"
```

14. **String toUpperCase():** Transforma todos los caracteres del objeto actual a mayúsculas.

```
String str = new String("My name is John Doe");
System.out.println(str.toUpperCase()); // Retorna "MY NAME IS JOHN DOE"
```

15. **String trim():** Retorna la cadena de caracteres sin espacios al principio y al final de la cadena.

```
String str = new String(" My name is John Doe ");
System.out.println(str.trim()); // Retorna "My name is John Doe"
```

16. **String replace(String oldChar, String newChar):** Retorna la cadena de caracteres en la que se sustituye la subcadena *oldChar* por la cadena *newChar*.

```
String str = new String("My name is John Doe");
System.out.println(str.replace("John", "Michael")); // Retorna "My name is Michael Doe"
```

17. **String subSequence(int start_index, int last_index):** Genera una cadena de caracteres a partir de los índices especificados en los parámetros.

```
String str = new String("My name is John Doe");
System.out.println(str.subSequence(11, 15)); // Retorna "John"
```

18. **String[] Split(String regex)**: Divide la cadena de caracteres contenida en el objeto String actual tomando como carácter separador el indicado en la cadena de caracteres regex. (Este parámetro es en realidad una **expresión regular**. Para saber más sobre las expresiones regulares puedes ir al siguiente enlace: <https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#sum>).

```
String asignaturas = "Programación-Bases de Datos-Entornos de Desarrollo-Lenguajes de Marcas";
String[] vAsignaturas = asignaturas.split("-");
System.out.println(vAsignaturas[2]); // La salida es "Entornos de Desarrollo"
```

Ejercicio 1:

Partiendo de la cadena de caracteres “Me gusta programar en Java.”, aplicar los métodos expuestos más arriba para:

- A) Obtener la longitud de la cadena de caracteres.
- B) Obtener el carácter que ocupa la posición 5ª.
- C) Obtener la sub-cadena desde la posición 21 hasta el final.
- D) Obtener la sub-cadena indicada por los índices 8 y 16.
- E) Muestra por pantalla el resultado de concatenar la cadena “ (Casi siempre)”.
- F) En la nueva cadena de caracteres obtenida en E), recuperar la posición de la cadena “Casi”.
- G) Muestra por pantalla la cadena de caracteres original en mayúsculas.
- H) Compara la cadena de caracteres original con la cadena de caracteres en mayúsculas. Primero teniendo en cuenta la diferencia entre mayúsculas y minúsculas y por último ignorando esta diferencia.

Ejercicio 2:

Desarrollar una función que haga el reverso de una cadena de caracteres que se suministre por parámetro. Investiga que función de la API String (no explicada aquí) podría ser más útil para tal cometido.

Ej: “Esto es una PRUEBA” → “ABEURP una es otsE”

StringBuilder

De entre todos los métodos de la API de *StringBuilder* resaltaremos los siguientes:

StringBuilder(): Creación de un *StringBuilder* sin contenido.

```
StringBuilder strB = new StringBuilder();
strB.append("Estamos en el primer curso de DAW");
System.out.println(strB); // Muestra por pantalla "Estamos en el primer curso de DAW"
```

StringBuilder(String str): Creación de un *StringBuilder* inicializando su contenido con una cadena de caracteres.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW");
System.out.println(strB); // Muestra por pantalla "Estamos en el primer curso de DAW"
```

int length(): Devuelve la longitud de la cadena de caracteres almacenada en el objeto *StringBuilder* actual.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW");
System.out.println(strB.length()); // La salida es 33
```

StringBuilder append(*): Añade al final o inicializa un *StringBuilder* con una cadena de caracteres. * Se refiere todos los tipos primitivos Java, a la clase *String* o cualquier otro objeto que tenga redefinido el método *toString*.

```
StringBuilder strB = new StringBuilder();
strB.append("Estamos en el curso ");
strB.append(1);
strB.append(" de DAW.");
System.out.println(strB); // Muestra por pantalla "Estamos en el curso 1 de DAW"
```

char charAt(int index): Retorna el carácter de la posición pasada por parámetro en la cadena de caracteres almacenada en el *StringBuilder*.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW");
System.out.println(strB.charAt(11)); // Muestra por pantalla e
```

StringBuilder delete(int start, int end): Elimina una sub-cadena del *StringBuilder* actual.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW");
System.out.println(strB.delete(21, 27)); // La salida es "Estamos en el primer de DAW"
```

StringBuilder deleteCharAt(int index): Elimina el carácter posicionado en el índice pasado por parámetro.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW");
System.out.println(strB.deleteCharAt(20)); // La salida es "Estamos en el primercurso de DAW"
```

StringBuilder insert(int offset, *): Inserta la representación en cadena de caracteres del objeto o tipo primitivo pasado en el segundo parámetro a partir del índice pasado en el primer parámetro.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW.");
System.out.println(strB.insert(33, new StringBuilder(" Genial!!!"))); // La salida es "Estamos en el primer curso de DAW Genial!!!."
```

StringBuilder replace(int begin, int end, String str): Reemplaza la sub-cadena de caracteres enmarcada entre los índices *begin* y *end* por la cadena de caracteres suministrada en el tercer parámetro.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW.");
System.out.println(strB.replace(14, 19, "segundo")); // La salida es "Estamos en el segundor curso de DAW."
```

String substring(int start): Retorna un *String* conformado por la sub-cadena desde la posición *start* hasta el final de la cadena de caracteres.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW.");
System.out.println(strB.substring(14)); // La salida es "primer curso de DAW."
```

String substring(int start, int end): Retorna un *String* conformado por la sub-cadena desde la posición *start* hasta la posición *end*.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW.");
System.out.println(strB.substring(14, 20)); // La salida es "primer"
```

StringBuilder reverse(): Retorna un *StringBuilder* reemplazando la cadena de caracteres del *StringBuilder* original por esa misma cadena dada la vuelta.

```
StringBuilder strB = new StringBuilder("Estamos en el primer curso de DAW.");
System.out.println(strB.reverse()); // La salida es ".WAD ed osruc remirp le ne somatsE"
```

Ejercicio 3:

Trabajando con el API de *StringBuilder* y partiendo de la cadena de caracteres **"Hello Java world."**, realizar las siguientes operaciones:

1. **Eliminar** la cadena de caracteres **"Java"** y mostrar la cadena de caracteres del *StringBuilder* resultante.
2. Sobre el *StringBuilder* resultante del apartado anterior **insertar** la cadena **"my "** en la posición 6. Mostrar la cadena de caracteres resultante.

3. **Añadir** al final de la cadena de caracteres del *StringBuilder* resultante del apartado 2 la cadena de caracteres: " I am xxxx.". Donde xxxx es mi nombre de pila. Mostrar la cadena de caracteres resultante.
4. En el *StringBuilder* del apartado anterior **reemplazar** vuestro nombre de pila por vuestro primer apellido.
5. **Obtener la subcadena** entre las posiciones 0 y 5, **creando** un nuevo *StringBuilder* con ella. Hacer el **reverso** de este nuevo substring y **añadela** de nuevo al *StringBuilder* del apartado 4 a partir de la posición 0. Mostrar la cadena de caracteres resultante.

Fechas

Vamos a trabajar con fechas y tiempos de una manera práctica, a través de ejemplos sobre las operaciones más comunes que se realizan con fechas y tiempos:

Ejemplo 1- Convertir Date a String

```
SimpleDateFormat sdf = new SimpleDateFormat("dd/M/yyyy");
String date = sdf.format(new Date());
System.out.println(date); // El resultado es "16/4/2020"
```

Nota que es el método *format* de *SimpleDateFormat* el que "formatea" una fecha "en bruto" al formato que se le especifica en su creación.

Ejercicio 4.1

Visita la API *SimpleDateFormat* y haz un pequeño resumen de los patrones de los formatos de las fechas. ¿Qué se muestra por pantalla con un objeto *Date* sin formato?

Ejemplo 2- Convertir String a Date

```
SimpleDateFormat sdf = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
String dateInString = "16-04-2025 10:20:56";
Date date = sdf.parse(dateInString);
System.out.println(date); //La salida es Wed Apr 16 10:20:56 CEST 2025
```

Nota que es el método *parse* de la API de *SimpleDateFormat* el que convierte una cadena de caracteres que es una fecha al formato especificado en la creación de la clase.

Ejercicio 4.2

¿Qué pasa si el contenido de la cadena de caracteres no es una fecha o no coincide con el formato especificado? ¿Cómo solucionarlo? Implementa la solución para el código del ejemplo 2.

Ejemplo 3- Obtener la fecha y hora actuales y mostrarlos en un formato determinado.

```
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Date date = new Date();
System.out.println(dateFormat.format(date)); // La salida es 2020/04/16 18:46:37
```

Ejercicio 4.3

Modifica el código del ejemplo 3 para mostrar sólo la fecha. Ahora muestra sólo la hora.

Ejemplo 4- Obtener partes de una fecha y hora (GregorianCalendar)

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd HH:mm:ss");
Calendar calendar = new GregorianCalendar(2025,1,28,13,24,56);

int year      = calendar.get(Calendar.YEAR);
int month     = calendar.get(Calendar.MONTH); // Jan = 0, dec = 11
int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
int dayOfWeek = calendar.get(Calendar.DAY_OF_WEEK);
int weekOfYear = calendar.get(Calendar.WEEK_OF_YEAR);
int weekOfMonth = calendar.get(Calendar.WEEK_OF_MONTH);


int hour      = calendar.get(Calendar.HOUR); // Reloj de 12 horas
int hourOfDay = calendar.get(Calendar.HOUR_OF_DAY); // Reloj de 24 horas
int minute    = calendar.get(Calendar.MINUTE);
int second    = calendar.get(Calendar.SECOND);
int millisecond = calendar.get(Calendar.MILLISECOND);

System.out.println(sdf.format(calendar.getTime()));

System.out.println("year \t\t: " + year);
System.out.println("month \t\t: " + month);
System.out.println("dayOfMonth \t: " + dayOfMonth);
System.out.println("dayOfWeek \t: " + dayOfWeek);
System.out.println("weekOfYear \t: " + weekOfYear);
System.out.println("weekOfMonth \t: " + weekOfMonth);

System.out.println("hour \t\t: " + hour);
System.out.println("hourOfDay \t: " + hourOfDay);
System.out.println("minute \t\t: " + minute);
System.out.println("second \t\t: " + second);
System.out.println("millisecond \t: " + millisecond);
```

La salida del código de la izquierda es:



```
year      : 2025
month     : 1
dayOfMonth : 28
dayOfWeek  : 6
weekOfYear : 9
weekOfMonth : 4
hour      : 1
hourOfDay  : 13
minute    : 24
second    : 56
millisecond : 0
```

Ejercicio 4.4

Comprueba la salida del código anterior en caso de que el *GregorianCalendar* **calendar** se crease sin parámetros, es decir, si se usa el constructor por defecto de *GregorianCalendar* para la variable **calendar**.

Ejemplo 5- Comparación de fechas. Método 1: Uso de *compareTo* de la API de la clase *Date*.

```
try{
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    Date date1 = sdf.parse("2020-12-31");
    Date date2 = sdf.parse("2021-01-31");
    System.out.println("date1 : " + sdf.format(date1));
    System.out.println("date2 : " + sdf.format(date2));

    if (date1.compareTo(date2) > 0) {
        System.out.println("Date1 es posterior Date2");
    } else if (date1.compareTo(date2) < 0) {
        System.out.println("Date1 es anterior Date2");
    } else if (date1.compareTo(date2) == 0) {
        System.out.println("Date1 es igual a Date2");
    } else {
        System.out.println("No es posible que estes aquí!");
    }
} catch (java.text.ParseException e){
    System.out.println("Error");
}
```

Ejemplo 6- Comparación de fechas. Método 2: Uso de *after*, *before* e *equals* de la API de la clase *Date*.

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
Date date1 = sdf.parse("2019-12-31");
Date date2 = sdf.parse("2020-01-31");

System.out.println("date1 : " + sdf.format(date1));
System.out.println("date2 : " + sdf.format(date2));

if (date1.after(date2)) {
    System.out.println("Date1 is after Date2");
}

if (date1.before(date2)) {
    System.out.println("Date1 is before Date2");
}

if (date1.equals(date2)) {
    System.out.println("Date1 is equal Date2");
}
```

La salida del código de la izquierda es:



```
date1 : 2019-12-31
date2 : 2020-01-31
Date1 is before Date2
```

Ejercicio 5

Dada la cadena de caracteres **sFechas**, en el que se guardan fechas separadas por comas con el formato días/mes/año, generar un array **vFechas** en el que cada posición del array será un objeto de tipo *Date* con formato *yyyy-MM-dd*, con todas las fechas de **sFechas**. Dicho array deberá ser ordenado de manera ascendente.

sFechas = "21/04/2015, 12/06/1955, 26/01/1985, 18/06/1936, 12/10/1492, 25/12/2000, 02/05/2018, 13/10/1975, 01/03/1991, 20/10/1972, 17/04/2020"