Tema 5. Estucturas de datos

Clases String, StringBuilder y StringTokenizer

1. LA CLASE STRING

Un String representa una **cadena de caracteres no modificable**.

Todos los literales de la forma "*cualquier texto entre comillas dobles*", es decir, literales entre comillas dobles que aparecen en un programa java, se implementan como objetos de tipo String.

Se pueden **crear un String** de varias formas, entre ellas:

- utilizando una **cadena de caracteres** entre comillas:
  ```
  String s1 = "abcdef";
  ```
- utilizando el **operador +** (operador concatenación) con dos objetos String:
  ```
  String s2 = s1 + "ghij";   //s2 contiene "abcdefghij"
  ```

Además la clase String proporciona varios **constructores**, entre ellos:

CONSTRUCTOR	DESCRIPCIÓN

String()	Constructor por defecto. El nuevo String toma el valor ""
	String s = new String(); //crea el string s vacío.
	Equivale a: String s = "";

String(String s)	Crea un nuevo String, copiando el que recibe como parámetro. String s = "hola"; String s1 = new String(s); //crea el String s1 y le copia el contenido de s
String(char[] v)	Crea un String y le asigna como valor los caracteres contenidos en el array recibido como parámetro. char [] a = {'a', 'b', 'c', 'd', 'e'}; String s = new String(a); //crea String s con valor "abcde"
String(char[] v , int pos, int n)	Crea un String y le asigna como valor los n caracteres contenidos en el array recibido como parámetro, a partir de la posición pos. char [] a = {'a', 'b', 'c', 'd', 'e'}; String s = new String(a, 1, 3); //crea String s con valor "bcd";

2. MÉTODOS DE LA CLASE STRING

MÉTODO	DESCRIPCIÓN
length()	Devuelve la longitud de la cadena
indexOf('caracter')	Devuelve la posición de la primera aparición de carácter. Devuelve -1 si no lo encuentra.
lastIndexOf('caracter')	Devuelve la posición de la última aparición de carácter
charAt(n)	Devuelve el carácter que está en la posición n
substring(n1,n2)	Devuelve la subcadena desde la posición n1 hasta n2 - 1
toUpperCase()	Devuelve la cadena convertida a mayúsculas
toLowerCase()	Devuelve la cadena convertida a minúsculas
equals(otroString)	Compara dos cadenas y devuelve true si son iguales
equalsIgnoreCase(otroString)	Igual que equals pero sin considerar mayúsculas y minúsculas
compareTo(OtroString)	Devuelve 0 si las dos cadenas son iguales. <0 si la primera es alfabéticamente menor que la segunda ó >0 si la primera es alfabéticamente mayor que la segunda.
compareToIgnoreCase(OtroString)	Igual que compareTo pero sin considerar mayúsculas y minúsculas.
valueOf(N)	Convierte el valor N a String. N puede ser de cualquier tipo.
isEmpty()	Devuelve true o false si el String está vacío o no.
trim()	Elimina los espacios en blanco al principio y al final del String.
replace(A, B)	Todas las ocurrencias de A en el String se sustituyen por B. A y B pueden ser un char o un String.
replaceAll(ExprRegular A, String B)	Todas las ocurrencias en el String que coinciden con la expresión regular se sustituyen por B.

Son solo algunos métodos de String. Los puedes consultar todos en la API de Java:

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/String.html

Los objetos String no son modificables.

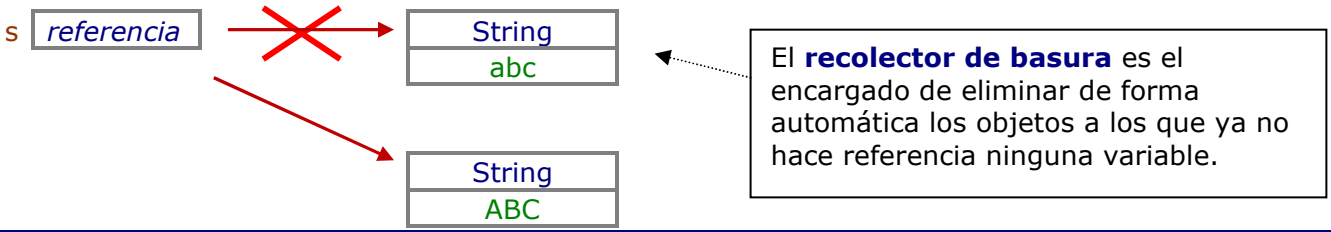
Los métodos que actúan sobre un String con la intención de modificarlo no lo modifican sino que devuelven un nuevo String con las modificaciones indicadas.

Por ejemplo: Una operación como convertir a mayúsculas o minúsculas un String no lo modificará sino que creará y devolverá un nuevo String con el resultado de la operación.

```
String s = "abc";
```


```
s = s.toUpperCase(); //convertir a mayúsculas el contenido del String s
```

En esta instrucción se crea un nuevo String con el contenido de s en mayúsculas. La dirección del nuevo String se le asigna a s.



3. CONCATENAR STRING

La clase String proporciona el operador + (concatenación) para unir dos o más String.

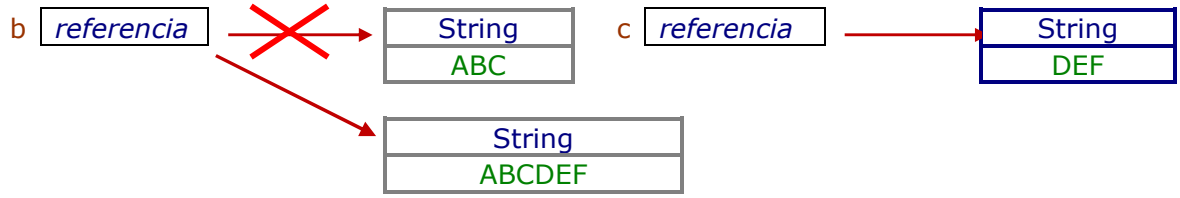
El resultado de aplicar este operador es un nuevo String concatenación de los otros.

Por ejemplo:

```
String b = "ABC";
String c = "DEF";
```

La operación: `b = b + c;`

crea un nuevo String (b + c) y le asigna su dirección a b:



4. EJEMPLOS DE USO DE STRING

Ejemplo 1: Programa que lea un texto y lo guarde en un String. El texto introducido no puede estar vacío, se debe introducir al menos un carácter. Elimina la última palabra del texto introducido y muestra el texto resultante. Las palabras del texto se introducirán separadas por espacios en blanco.

```
import java.util.Scanner;
public class String1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String texto;
        int pos;
        do {
            System.out.println("Introduce texto: ");
            texto = sc.nextLine();
        } while (texto.isEmpty()); //se debe introducir algo

        texto = texto.trim(); //elimina los posibles espacios en blanco
                              //al principio y al final del texto
        pos = texto.lastIndexOf(" "); //se busca el último espacio en blanco
        if (pos != -1) { //si lo hemos encontrado
            //se obtiene un String desde el principio del texto
            //hasta la posición anterior a ese espacio
            texto = texto.substring(0, pos);
        } else { //si no lo hemos encontrado significa que solo había una palabra
            texto = ""; //si solo había una palabra se la quitamos
        }
        System.out.println("Texto modificado: ");
        System.out.println(texto);
    }
}
```

Ejemplo 2: Programa que calcule el número de palabras que contiene un texto que se introduce por teclado. Para resolverlo escribe un método que reciba el texto y devuelva el número de palabras que contiene. Supondremos las palabras separadas por un espacio en blanco.

```
import java.util.Scanner;
public class String4 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String texto;
        System.out.println("Introduce texto: ");
        texto = sc.nextLine();
        System.out.println("El texto tiene " + contarPalabras(texto) + " palabras");
    }
    //método que recibe un String y devuelve el número de palabras que contiene
    //Suponemos que las palabras están separadas por un espacio en blanco
    public static int contarPalabras(String s) {
        int contador = 1, pos;
        s = s.trim(); //eliminar los posibles espacios en blanco al principio y al final
        if (s.isEmpty()) {
            contador = 0;
        } else {
            pos = s.indexOf(" ");
            while (pos != -1) {
                contador++;
                pos = s.indexOf(" ", pos + 1);
            }
        }
        return contador;
    }
}
```

5. LA CLASES STRINGBUILDER Y STRINGBUFFER

La clase String es una clase cuyos objetos no son modificables. Para **trabajar con cadenas de caracteres modificables**, Java proporciona las clases StringBuffer y StringBuilder.

StringBuilder y StringBuilder proporcionan los mismos métodos. Se diferencian en que los métodos de StringBuffer están sincronizados y los de StringBuilder no. Los métodos sincronizados se usan en aplicaciones multihilo y consumen más recursos que los métodos no sincronizados. Por este motivo StringBuilder ofrece mejor rendimiento que StringBuffer cuando la aplicación no es multihilo, que será nuestro caso durante este curso. Nosotros usaremos StringBuilder.

Cuando usar cada clase:

- Usaremos **String** si la cadena de caracteres no va a cambiar.
- Usaremos **StringBuilder** si la cadena de caracteres puede cambiar y la aplicación no es multihilo.
- Usaremos **StringBuffer** si la cadena de caracteres puede cambiar y la aplicación es multihilo.

Los objetos de tipo StringBuilder gestionan automáticamente su capacidad: Se crean con una capacidad inicial y la incrementan cuando es necesario.

Las clase StringBuilder proporciona varios **constructores:**

CONSTRUCTOR	DESCRIPCIÓN
StringBuilder()	Crea un StringBuilder vacío. StringBuilder sb = new StringBuilder ();
StringBuilder(int n)	Crea un StringBuilder vacío con capacidad para n caracteres.
StringBuilder(String s);	Crea un StringBuilder con valor inicial el String s. String s = "ejemplo"; StringBuilder sb = new StringBuilder(s); //crea StringBuilder sb y le asigna el contenido de s

6. MÉTODOS DE STRINGBUILDER

Las clases StringBuilder proporciona métodos para acceder y modificar la cadena de caracteres. Algunos de ellos son:

MÉTODO	DESCRIPCIÓN
length()	Devuelve la longitud de la cadena
append(X);	Añade X al final de la cadena. X puede ser de cualquier tipo
insert(posicion, X)	Inserta X en la posición indicada. X puede ser de cualquier tipo.
setCharAt(posicion, c)	Cambia el carácter que se encuentra en la posición indicada, por el carácter c.
charAt(posicion)	Devuelve el carácter que se encuentra en la posición indicada.
indexOf('caracter')	Devuelve la posición de la primera aparición de carácter. Devuelve -1 si no existe
lastIndexOf('caracter')	Devuelve la posición de la última aparición de carácter
substring(n1,n2)	Devuelve la subcadena (String) comprendida entre las posiciones n1 y n2-1. Si no se especifica n2, devuelve desde n1 hasta el final.
delete(inicio, fin)	Elimina los caracteres desde la posición inicio hasta fin-1.
reverse()	Invierte el contenido de la cadena
toString()	Devuelve el String equivalente.

Los puedes consultar todos en la API de Java:

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuilder.html
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StringBuffer.html

7. LA CLASE STRINGTOKENIZER

La clase StringTokenizer permite dividir un String en una serie de trozos (tokens) delimitados por caracteres específicos. Los delimitadores de los tokens por defecto son:

espacio en blanco, tabulador \t, salto de línea \n, retorno \r y avance de página \f

Para utilizarla es necesario hacer el import:

```
import java.util.StringTokenizer;
```

Un objeto StringTokenizer **se construye a partir de un objeto String**.

```
StringTokenizer st = new StringTokenizer("colores rojo, verde y azul");
```

Para obetener los tokens del String podemos utilizar los métodos hasMoreTokens() y nextToken().

- **hasMoreTokens()** devuelve true si hay más tokens que obtener en la cadena.
- **nextToken()** devuelve un String con el siguiente token.

Otro método importante es **countTokens()** que devuelve la cantidad de tokens que aun quedan por extraer.

Por ejemplo:

```
StringTokenizer st = new StringTokenizer("blanco, rojo, verde y azul");
while (st.hasMoreTokens()){
       System.out.println(st.nextToken());
}
```

La salida que se obtiene es:

```
blanco,
rojo,
verde
y
azul
```

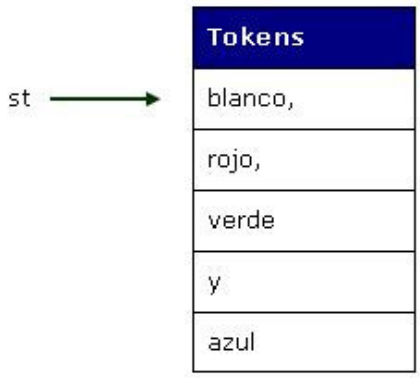
Se ha separado el String s en tokens separados por el delimitador por defecto, en este caso el espacio en blanco.

Si lo representamos de forma gráfica, el funcionamiento es el siguiente:

La instrucción

```
StringTokenizer st = new StringTokenizer("blanco, rojo, verde y azul");
```

Produce un resultado que podemos representar de la siguiente forma:



Se separa el String original en tokens.

st apunta al primer token obtenido

A continuación la instrucción

```
st.hasMoreTokens()
```

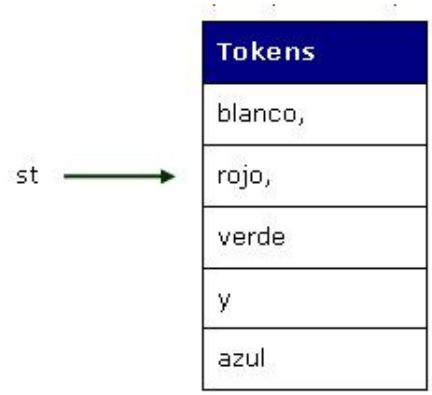
comprueba si hay tokens que extraer.

En este caso st apunta a un token por lo tanto hasMoreTokens() devuelve *true*.

A continuación la instrucción:

`st.nextToken()` que se encuentra dentro de `System.out.println`

obtiene el token al que apunta st `("blanco,")` y avanza al siguiente.



El String `"blanco, "` se muestra por pantalla.

El ciclo while repite el proceso.

Cuando se obtiene el último *token* `("azul")` y st avanza, st apunta ahora al siguiente token después de *azul*. Como ya no hay más tokens la condición `while(st.hasMoreTokens()){` será *false*.

Si queremos utilizar otros delimitadores distintos, podemos especificarlo cuando se crea el objeto StringTokenizer.

Por ejemplo, para indicar que los delimitadores son la coma y el espacio en blanco:

```
StringTokenizer st = new StringTokenizer("colores rojo, verde y azul", ", ");
```

La ejecución del while anterior obtendría la salida:

```
colores
rojo
verde
y
azul
```

La coma no aparece ya que se ha especificado como delimitador y los delimitadores no aparecen. Si queremos que los delimitadores aparezcan como *tokens* se debe escribir *true* como tercer argumento en el constructor:

```
StringTokenizer st = new StringTokenizer("colores rojo, verde y azul", ", ", true);
```

En este caso la salida es:

```
colores

rojo
,

verde

y

azul
```

Debemos tener en cuenta que si indicamos delimitadores ya no tendremos el espacio en blanco como delimitador por defecto. Si queremos seguir utilizándolo hay que indicarlo junto al resto de delimitadores.

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/StringTokenizer.html

8. EJEMPLOS DE USO DE STRING, STRINGBUILDER Y STRINGTOKENIZER

Ejemplo 1: Programa que lea un texto y elimine los espacios en blanco sobrantes del texto introducido. Los espacios sobrantes pueden estar al principio, al final o entre las palabras del texto. Entre una palabra y otra de la cadena final solo habrá un espacio en blanco. Para resolverlo escribe un método que reciba un String con el texto y devuelva un String con el texto modificado.

```
import java.util.Scanner;
import java.util.StringTokenizer;
public class String9 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String s;
        System.out.println("Introduce texto: ");
        s = sc.nextLine();
        System.out.println("Texto introducido: ");
        System.out.println(s);
        s = eliminarBlancos(s);
        System.out.println("Texto modificado: ");
        System.out.println(s);
    }
    //método que recibe un String y devuelve otro sin los espacios en blanco sobrantes
    public static String eliminarBlancos(String s) {
        //Creamos un StringBuilder vacío para ir formando el String resultante
        StringBuilder sb = new StringBuilder();
        //Creamos un StringTokenizer a partir del String que recibe el método
        //En este caso el delimitador (separador de tokens) será el espacio en blanco
        //que es el delimitador por defecto. Por eso no es necesario indicarlo
        StringTokenizer st = new StringTokenizer(s);
        //Se recorre el StringTokenizer
        //En este caso cada token obtenido es una palabra del String
        while (st.hasMoreTokens()) { //mientras haya tokens
            sb.append(st.nextToken()); //se extrae el token y se añade al StringBuilder
            sb.append(" "); //a continuación de cada palabra se añade un espacio en blanco
        }
        //se convierte el Stringbuilder a String
        //se elimina el espacio en blanco añadido después de la última palabra
        //y se devuelve
        return sb.toString().trim();
    }
}
```

Ejemplo 2: Escribe un **método** que reciba como entrada un String que contiene caracteres separados por espacios en blanco, comas y guiones y devuelva un String con los guiones y las comas sustituidos por espacios. Por ejemplo, si el método recibe el String "ab c,de-f-g h" devolverá el String "ab c de f g h".

```
public static String metodo(String s) {
        //Creamos un StringBuilder vacío para ir formando el String resultante
        StringBuilder sb = new StringBuilder();
        //Creamos un StringTokenizer a partir del String que recibe el método
        //En este caso los delimitadores son la coma, el guión y el espacio en blanco
        StringTokenizer st = new StringTokenizer(s, " ,-");
        while (st.hasMoreTokens()) { //mientras haya más tokens
            sb.append(st.nextToken()); //extraemos un token y lo añadimos al StringBuilder
            sb.append(" "); //a continuación se añade un espacio en blanco
        }
        //se convierte el Stringbuilder a String
        //se elimina el espacio en blanco añadido después de la última palabra
        //y se devuelve
        return sb.toString().trim();
    }
```

9. DIVIDIR UN STRING UTILIZANDO EL MÉTODO SPLIT

El método split de la clase String es la alternativa a usar StringTokenizer para separar cadenas.
Este método divide el String en cadenas según la **expresión regular** que recibe.
El método split devuelve un array de String con las partes obtenidas del String original.

Ejemplo 1: Se dispone de un String que contiene palabras separadas por espacios en blanco. Se desea obtener las palabras que contiene el String por separado.

```
String str = "blanco rojo amarillo verde azul marrón naranja negro";
String[] cadenas = str.split(" ");
for (int i = 0; i < cadenas.length; i++) {
     System.out.println(cadenas[i]);
}
```

Salida por pantalla:

```
blanco
rojo
amarillo
verde
azul
marrón
naranja
negro
```

En este ejemplo se ha indicado al método Split que el separador de palabras es el espacio en blanco. De la misma forma podemos indicar que separe el String por cualquier otro carácter. Por ejemplo, si queremos que separe el String anterior por el carácter 'a' escribiríamos:

```
String[] cadenas = str.split("a");
```

Los String obtenidos ahora son:

```
bl
nco rojo
m
rillo verde
zul m
rrón n
r
nj
 negro
```

Hay una serie de caracteres que no podemos usar directamente como separadores. Estos caracteres son: `\ ^ $ . | ? * + ( ) { } [`

Estos caracteres se denominan **metacaracteres** y tienen un significado especial en las expresiones regulares.

Si queremos indicar que las palabras del String están separadas por alguno de estos caracteres debemos "escaparlo" escribiendo delante \\

Ejemplo 2: Se dispone de un String que contiene palabras separadas por puntos. Se desea obtener las palabras que contiene el String por separado.

```
String str = "blanco.rojo.amarillo.verde.azul.marrón.naranja.negro";
String[] cadenas = str.split("\\.");
for (int i = 0; i < cadenas.length; i++) {
        System.out.println(cadenas[i]);
}
```

Salida por pantalla:

```
blanco
rojo
amarillo
verde
azul
marrón
naranja
negro
```

Ejemplo 3: Si queremos indicar que son varios los caracteres separadores lo tenemos que indicar escribiendo los separadores entre corchetes.

Se dispone de un String que contiene palabras separadas por guiones, comas, puntos y espacios en blanco. Se desea obtener las palabras que contiene el String por separado.

```
String str = "blanco-rojo,amarillo verde-azul.marrón-naranja,negro";
String[] cadenas = str.split("[-,. ]");
for (int i = 0; i < cadenas.length; i++) {
     System.out.println(cadenas[i]);
}
```

Salida por pantalla:

```
blanco
rojo
amarillo
verde
azul
marrón
naranja
negro
```

En este ejemplo el método Split recibe la expresión regular `[-,. ]`

Entre corchetes se indican los caracteres que queremos usar como separadores.

En este caso el carácter punto ya no es necesario escaparlo.

Ejemplo 4: Utilizando expresiones regulares podemos escribir separadores más complejos. Por ejemplo, podemos indicar que los separadores serán los siguientes:

- El carácter e seguido de una s o una m, es decir, cada vez que aparezca la secuencia "es" ó "em"
- La secuencia "un"

Esto lo indicaremos mediante la expresión regular:

```
(e[s|m])|(un)
```

```
String str = "Se muestra un ejemplo de como funciona split";
String[] cadenas = str.split("(e[s|m])|(un)");
for (int i = 0; i < cadenas.length; i++) {
     System.out.println(cadenas[i]);
}
```

Salida por pantalla:

```
Se mu
tra
 ej
plo de como f
ciona split
```