**El paquete java.lang**

**Clase String**

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

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

**CREAR UN STRING**

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

-         Utilizando una **cadena de caracteres** entre comillas:

String s1 = "abcdef";

-         Utilizando **operador de concatenación *+*** con dos o más objetos String:

String s2 = s1 + "ghij"; //s2 contiene "abcdefghij"

String s3 = s1 + s2 + "klm"; //s3 contiene " abcdefabcdefghijklm"

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"; |

**MÉTODOS DE LA CLASE STRING**

La clase String proporciona métodos para el tratamiento de las cadenas de caracteres: acceso a caracteres individuales, buscar y extraer una subcadena, copiar cadenas, convertir cadenas a mayúsculas o minúsculas, etc.

|  |  |
| --- | --- |
| **MÉTODO** | **DESCRIPCIÓN** |
| length() | Devuelve la longitud de la cadena |
| indexOf(‘caracter’) | Devuelve la posición de la primera aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra. |
| lastIndexOf(‘caracter’) | Devuelve la posición de la última aparición de *carácter* dentro del String. Devuelve -1 si no lo encuentra. |
| 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. |

Los puedes consultar todos en la API de Java:

<http://docs.oracle.com/javase/10/docs/api/index.html?java/lang/String.html>

Debemos recordar que:

**Los objetos String no son modificables**.

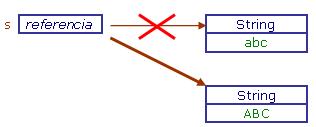
Por lo tanto, los métodos que actúan sobre un String con la intención de modificarlo lo que hacen es crear un nuevo String a partir del original y devolverlo modificado.

Por ejemplo: Una operación como convertir a mayúsculas o minúsculas un String no lo odificará, 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



El **recolector de basura** es el encargado de eliminar de forma automática los objetos a los que ya no hace referencia ninguna variable.

**EL OPERADOR DE CONCATENACIÓN +**

La clase 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, si tenemos dos String b y c:

String b = "Ordenador";

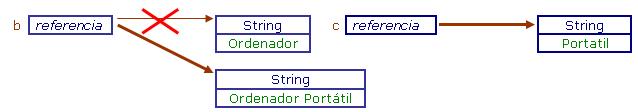
String c = " Portátil";



La operación:

b = b + c;

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



**Java StringBuilder - StringBuffer**

La clase String es una clase no modificable. Esto quiere decir que cuando se modifica un String se crea un nuevo objeto String modificado a partir del original y el recolector de basura es el encargado de eliminar de la memoria el String original.

Java proporciona la clase StringBuffer y a partir de Java 5 la clase StringBuilder para trabajar con cadenas de caracteres sobre las que vamos a realizar modificaciones frecuentes de su contenido.

La **diferencia entre StringBuffer y StringBuilder** es que los métodos de StringBuffer están sincronizados y los de StringBuilder no lo están. Por este motivo StringBuilder ofrece mejor rendimiento que StringBuffer y la utilizaremos cuando la aplicación tenga un solo hilo de ejecución.

En general decidiremos cuando usar String, StringBuilder o StringBuffer según lo siguiente:

-         Usaremos **String** si la cadena de caracteres no va a cambiar.

-         Usaremos **StringBuilder** si la cadena de caracteres puede cambiar y solamente tenemos un hilo de ejecución.

-         Usaremos **StringBuffer** si la cadena de caracteres puede cambiar y tenemos varios hilos de ejecución.

En esta entrada utilizaremos StringBuilder teniendo en cuenta que todo lo que se explica aquí es aplicable a StringBuffer.

**Constructores de la Clase StringBuilder**

Un objeto de tipo StringBuilder gestiona automáticamente su capacidad

– Se crea con una capacidad inicial.

– La capacidad se incrementa cuando es necesario.

La clase StringBuilder proporcionan varios **constructores**, algunos de ellos son:

|  |  |
| --- | --- |
| **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 y le asigna el contenido del String s.  String s = "ejemplo";  StringBuilder sb = new StringBuilder (s); |

**Métodos de la Clase StringBuilder**

La clase StringBuilder proporcionan 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 lo encuentra. |
| lastIndexOf(‘caracter’) | Devuelve la posición de la última aparición de *carácter*. Devuelve -1 si no lo encuentra. |
| 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:

<http://docs.oracle.com/javase/10/docs/api/java/lang/StringBuilder.html>  
<http://docs.oracle.com/javase/10/docs/api/java/lang/StringBuffer.html>

**Ejemplo de uso de la clase StringBuilder:**

Vamos a escribir un método *separarMiles* que reciba un String que representa un número entero y devuelva un String con el mismo número al que se le añadirán los puntos separadores de millares.

Por ejemplo, si el método recibe el String "12345678" debe devolver el String "12.345.678"

Este problema lo podemos resolver de varias formas. En este caso la idea es darle la vuelta al número e insertar el primer punto en la cuarta posición del String, el siguiente punto 4 posiciones más adelante el siguiente otras 4 posiciones más adelante .... hasta llegar al final del número. De esta forma obtendremos grupos de 3 cifras separados por punto.

Finalmente le volvemos a dar la vuelta y ya lo tendremos.

Por ejemplo, si el String es:

"12345678"

Primero le damos la vuelta:

"87654321"

Ahora tenemos que insertar un punto donde está el 5. Nos queda:

"876.54321"

Insertamos otro punto cuatro posiciones más adelante, donde está el 2:

"876.543.21"

Ahora intentaríamos insertar otro punto cuatro posiciones más adelante pero como llegamos al final el proceso termina.

Si le damos la vuelta obtendremos el resultado:

"12.345.678"

/\*

\* Ejemplo de uso de StringBuilder

\* Separador de millares

\*/

package string8;

public class String8 {

public static void main(String[] args) {

String s = "1234567890";

s = separarMiles(s);

System.out.println(s);

}

public static String separarMiles(String s){

//creamos un StringBuilder a partir del String s

StringBuilder aux = new StringBuilder(s);

//le damos la vuelta

aux.reverse();

//variable que indica donde insertar el siguiente punto

int posicion = 3;

//mientras no lleguemos al final del número

while(posicion < aux.length()){

//insertamos un punto en la posición

aux.insert(posicion,'.');

//siguiente posición donde insertar

posicion+=4;

}

//le damos de nuevo la vuelta

aux.reverse();

//el StringBuilder se pasa a String y se devuelve

return aux.toString();

}

}

**Eficiencia de la Clase StringBuilder frente a la Clase String**  
  
Podemos comprobar que es más eficiente utilizar StringBuilder frente a String realizando la siguiente prueba:

Vamos a concatenar un número grande de cadenas de caracteres, por ejemplo 100000, y vamos a medir el tiempo que se emplea en hacerlo.

Lo vamos a realizar primero utilizando la clase String. A continuación, utilizando la clase StringBuilder y finalmente lo vamos a hacer utilizando StringBuilder pero asignando inicialmente memoria para la longitud final de la cadena resultante.

public class String3 {

public static void main(String[] args) {

String s = "cadena";

long t1, t2;

int n = 100000;

System.out.print("Concatenar " + n + " cadenas con String: ");

t1 = System.currentTimeMillis();

concatenar(s,n);

t2 = System.currentTimeMillis();

System.out.println((t2-t1) + " milisegundos");

System.out.print("Concatenar " + n + " cadenas con StringBuilder: ");

t1 = System.currentTimeMillis();

concatenar1(s,n);

t2 = System.currentTimeMillis();

System.out.println((t2-t1) + " milisegundos");

System.out.print("Concatenar " + n + " cadenas con StringBuilder Optimizado: ");

t1 = System.currentTimeMillis();

concatenar2(s,n);

t2 = System.currentTimeMillis();

System.out.println((t2-t1) + " milisegundos");

}

//método que concatena n cadenas usando la clase String

public static String concatenar(String s, int n) {

String resultado = s;

for (int i = 1; i < n; i++) {

resultado = resultado + s;

}

return resultado;

}

//método que concatena n cadenas usando la clase StringBuilder

public static String concatenar1(String s, int n) {

StringBuilder resultado = new StringBuilder(s);

for (int i = 1; i < n; i++) {

resultado.append(s);

}

return resultado.toString();

}

//método optimizado que concatena n cadenas usando la clase StringBuilder

//se crea un StringBuilder inicial con el tamaño total del String resultante

public static String concatenar2(String s, int n) {

StringBuilder resultado = new StringBuilder(s.length() \* n);

for (int i = 0; i < n; i++) {

resultado.append(s);

}

return resultado.toString();

}

}

La salida del programa dependerá del ordenador que utilicemos. En cualquier caso nos debe mostrar que concatenar utilizando String es más lento que si utilizamos StringBuilder.   
En mi caso el resultado obtenido ha sido este:

Concatenar 100000 cadenas con String: 304435 milisegundos

Concatenar 100000 cadenas con StringBuilder: 15 milisegundos

Concatenar 100000 cadenas con StringBuilder Optimizado: 1 milisegundos

**Clase StringTokenizer**

La clase StringTokenizer sirve para separar una cadena de caracteres en una serie de elementos o **tokens.**

Se incluye en el paquete java.util

Los tokens se separan mediante delimitadores.

**Los delimitadores por defecto son:**

espacio en blanco

tabulador \t

salto de línea \n

retorno \r

avance de página \f

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

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. Lanza una excepción del tipo NoSuchElementException si no hay más tokens.

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

**Ejemplo** de uso de StringTokenizer:

String s = "blanco, rojo, verde y azul";

StringTokenizer st = new StringTokenizer(s);

while (st.hasMoreTokens())

System.out.println(st.nextToken());

La salida que se obtiene por pantalla es:

blanco,

rojo,

verde

y

azul

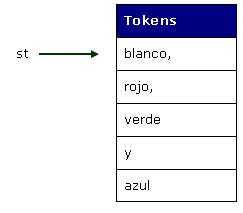
Se ha separado el String s en tokens separados por el delimitador por defecto. En este caso el espacio en blanco.

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

La instrucción

StringTokenizer st = new StringTokenizer(s);

Produce lo siguiente:

[](https://1.bp.blogspot.com/-7gtDi0BqI9g/UV1tBg2sepI/AAAAAAAAALQ/uLRmSSpBda4/s1600/java-stringtokenizer.JPG)

Se separa el String en tokens y el objeto st *apunta* al primero.

A continuación

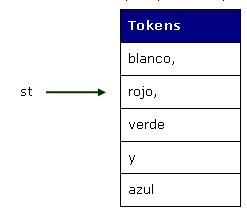
st.hasMoreTokens()

comprueba si hay tokens que extraer. En este caso st apunta a un token por lo tanto hasMoreTokens() devuelve true.

A continuación

st.nextToken()

Obtiene el token al que apunta st (blanco,) y avanza al siguiente.

[](https://1.bp.blogspot.com/-C3zxf57QDRE/UV1tLrYojMI/AAAAAAAAALY/WM3y8DdmT_k/s1600/java-stringtokenizer-2.JPG)

El ciclo while repite el proceso.

Cuando se alcanza el último token y se avanza el siguiente la condición del while será false.

Los **delimitadores** se pueden especificar 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 aparezcan 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

**Clases Envolventes**

Java es un lenguaje de programación orientado a objetos. Un programa Java debe contener objetos y las operaciones entre ellos. La única excepción a esto en un programa Java son los tipos de datos primitivos (int, double, char, etc.)

Los tipos de datos primitivos no son objetos, pero en ocasiones es necesario tratarlos como tales. Por ejemplo, hay determinadas clases que manipulan objetos (ArrayList, HashMap, …). Para poder utilizar tipos primitivos con estas clases Java provee las llamadas clases envolventes también llamadas clases contenedoras o wrappers.

Cada tipo primitivo tiene su correspondiente clase envolvente:

|  |  |
| --- | --- |
| **Tipo primitivo** | **Clase Envolvente** |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |
| char | Character |
| boolean | Boolean |

Estas clases proporcionan métodos que permiten manipular el tipo de dato primitivo como si fuese un objeto.

Las conversiones entre los tipos primitivos y sus clases envolventes son automáticas. No es necesario hacer un casting. Para realizarlas se utiliza el **Boxing/Unboxing**.

* **Boxing**: Convertir un tipo primitivo en su clase Wrapper.
* **Unboxing**: Convertir un objeto de una clase Wrapper en su tipo primitivo.

**Ejemplo de Boxing:**

double x = 29.95;

Double y;

y = x; // boxing

**Ejemplo de Unboxing:**

double x;

Double y = 29.95;

x = y; // unboxing

**[Clase Integer](https://www.blogger.com/blogger.g?blogID=3756682660522124367)**

En la siguiente tabla aparecen algunos métodos de la clase Integer. El resto de clases envolventes correspondientes a tipos primitivos numéricos tienen métodos similares.

|  |  |
| --- | --- |
| Integer(int valor)  Integer(String valor) | Constructor a partir de un int  Integer n = new Integer(20);  Constructor a partir de un String  String s = "123456";  Integer a = new Integer(s); |
| int intValue()  float floatValue()  double doubleValue()  … | Devuelve el valor equivalente  Integer n = new Integer(30);  int x = n.intValue();  double y = n.doubleValue(); |
| int parseInt(String s) | Método estático que devuelve un int a partir de un String.  String s = "123456";  int z = Integer.parseInt(s); |
| String toBinaryString(int i) String toOctalString(int i) String toHexString(int i) | Métodos estáticos que devuelven un String con la representación binaria, octal o hexadecimal del número.  int numero = 12;  String binario = Integer.toBinaryString(numero); |
| Integer valueOf(String s) | Método Estático. Devuelve un Integer a partir de un String.  Integer m = Integer.valueOf("123"); |

API de la clase Integer: <http://docs.oracle.com/javase/10/docs/api/java/lang/Integer.html>

**Clase Character**

Provee una serie de métodos para manipular los datos de tipo char. En la siguiente tabla aparecen algunos de estos métodos.

|  |  |
| --- | --- |
| Character(char c) | Constructor a partir de un char  char car = 'x';  Character a = new Character(car); |
| char charValue() | Devuelve el char equivalente  Character n = new Character('q');  char c = n.charValue(); |
| boolean isLowerCase(char ch) boolean isUpperCase(char ch)  boolean isDigit(char ch)  boolean isLetter(char ch) | Comprueba si es un carácter en minúsculas.  Comprueba si es un carácter en mayúsculas.  Comprueba si es un dígito (carácter del 0 al 9).  Comprueba si es una letra.  Todos son estáticos.  if(Character.isUpperCase(c)){    .....  } |
| char toLowerCase(char ch) char toUpperCase(char ch) | Devuelve el char en mayúsculas.  Devuelve el char en minúsculas.  Métodos estáticos.  char car = 'u';  System.out.println(Character.toUpperCase(car)); |
| Character valueOf(char c) | Método Estático. Devuelve un Character a partir de un char.  Character m = Character.valueOf('a'); |

API de la clase Character: <http://docs.oracle.com/javase/10/docs/api/java/lang/Character.html>

Ejemplo de uso de la clase Character: programa que lee un texto por teclado y muestra cuántos dígitos y letras contiene.

import java.util.Scanner;

public class MainJavaApplication {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String texto;

int cuentaCifras = 0, cuentaLetras = 0;

System.out.println("Introduce texto ");

texto = sc.nextLine();

for (int i = 0; i < texto.length(); i++) {

if (Character.isDigit(texto.charAt(i))) {

cuentaCifras++;

} else if (Character.isLetter(texto.charAt(i))) {

cuentaLetras++;

}

}

System.out.println("El texto contiene " + cuentaCifras + " dígitos");

System.out.println("El texto contiene " + cuentaLetras + " letras");

}

}