

Microfunciones de Cadenas

Recorridos, indices y construccion con StringBuilder (ES)

Introduccion a la Programacion

Unidad: Cadenas

Programa de demostracion (main)

```
1  public class EjemplosCadenas {
2      public static void main(String[] args){
3          System.out.println(EjemplosCadenas.primerPalabra("    hola mundo"));
4          System.out.println(EjemplosCadenas.ultimaPalabra("hola mundo    "));
5          System.out.println(EjemplosCadenas.contarPalabras("    uno    dos    tres    "));
6
7          System.out.println(EjemplosCadenas.encontrarNesima("abc—abc—abc","abc",2));
8          System.out.println(EjemplosCadenas.extraerEntre(
9              "NOMBRE:[Ana] AULA:[B—203]","NOMBRE:[ "," "]"));
10         System.out.println("— Construccion —");
11         System.out.println(EjemplosCadenas.rellenarIzquierda("42",5,'0'));
12         System.out.println(EjemplosCadenas.enmascararUsuarioCorreo("ana.soto@uni.edu"));
13         System.out.println(EjemplosCadenas.invertirMayusMinus("AbC xyz 123"));
14         System.out.println(EjemplosCadenas.normalizarEspacios("    hola    MUNDO    "));
15     }
16 }
```

Microfunciones de cadenas — índice

- ▶ `primeraPalabra(s)`: salta espacios iniciales y toma hasta el primer espacio.
- ▶ `ultimaPalabra(s)`: salta espacios finales y toma desde el último espacio.
- ▶ `contarPalabras(s)`: estado *dentro/afuera* para contar segmentos no vacíos.
- ▶ `encontrarNesima(texto, patron, n)`: usa `indexOf` y avanza sin solapes.
- ▶ `extraerEntre(s, ini, fin)`: ubica delimitadores y retorna subcadena interna.
- ▶ `rellenarIzquierda(s, ancho, relleno)`: construye con `StringBuilder`.
- ▶ `enmascararUsuarioCorreo(correo)`: preserva 1ª/última del usuario antes de @.
- ▶ `invertirMayusMinus(s)`: usa aritmética de caracteres (ASCII) para alternar caso.
- ▶ `normalizarEspacios(s)`: colapsa múltiples espacios y recorta extremos.

primeraPalabra(String) — ignora espacios iniciales

```
1 static String primeraPalabra(String s){
2     int i = 0, n = s.length();
3     // Saltar espacios del principio
4     while (i < n && s.charAt(i) == ' ') i++;
5     // Avanzar hasta el proximo espacio o fin
6     int j = i;
7     while (j < n && s.charAt(j) != ' ') j++;
8     return s.substring(i, j); // puede ser "" si no hay palabra
9 }
```

ultimaPalabra(String) — ignora espacios finales

```
1 static String ultimaPalabra(String s){
2     int j = s.length() - 1;
3     // Saltar espacios del final
4     while (j >= 0 && s.charAt(j) == ' ') j--;
5     // Retroceder hasta el espacio anterior o inicio
6     int i = j;
7     while (i >= 0 && s.charAt(i) != ' ') i--;
8     return s.substring(i + 1, j + 1); // "" si no hay palabra
9 }
```

contarPalabras(String) — colapsa mentalmente espacios

```
1  static int contarPalabras(String s){
2      int conteo = 0;
3      boolean dentro = false; // estoy dentro de una palabra?
4      for (int k = 0; k < s.length(); k++){
5          char c = s.charAt(k);
6          if (c != ' '){
7              if (!dentro){ conteo++; dentro = true; }
8          } else {
9              dentro = false; // cierre una palabra
10         }
11     }
12     return conteo;
13 }
```

encontrarNesima(String, String, int) — sin solapes

```
1  static int encontrarNesima(String texto, String patron, int n){
2      if (patron.length() == 0 || n <= 0) return -1;
3      int i = 0, m = patron.length(), cuenta = 0;
4      while (true){
5          int k = texto.indexOf(patron, i); // buscar desde i
6          if (k < 0) return -1;             // no hay mas
7          if (++cuenta == n) return k;      // n-esima
8          i = k + m;                       // avanzar sin solape
9      }
10 }
```

extraerEntre(String, String, String) — usa indices

```
1  static String extraerEntre(String s, String inicio, String fin){
2      int i = s.indexOf(inicio);
3      if (i < 0) return "";
4      i += inicio.length();
5      int j = s.indexOf(fin, i);
6      if (j < 0) return "";
7      return s.substring(i, j);
8  }
```


rellenarIzquierda(String,int,char) — construccion eficiente

```
1 static String rellenarIzquierda(String s, int ancho, char relleno){
2     int faltan = ancho - s.length();
3     if (faltan <= 0) return s;           // ya alcanza
4     StringBuilder sb = new StringBuilder(ancho);
5     for (int i = 0; i < faltan; i++) sb.append(relleno);
6     return sb.append(s).toString();
7 }
```

enmascararUsuarioCorreo(String) — preserva 1ra/ultima

```
1  static String enmascararUsuarioCorreo(String correo){
2      int arroba = correo.indexOf('@');
3      if (arroba <= 0) return correo; // sin usuario o sin '@'
4      String usuario = correo.substring(0, arroba);
5      String dominio = correo.substring(arroba); // incluye '@'
6      if (usuario.length() == 1) return "*" + dominio;
7      if (usuario.length() == 2) return usuario.charAt(0) + "*" + dominio;
8      StringBuilder sb = new StringBuilder(usuario.length() + dominio.length());
9      sb.append(usuario.charAt(0));
10     for (int i = 1; i < usuario.length() - 1; i++) sb.append('*');
11     sb.append(usuario.charAt(usuario.length() - 1)).append(dominio);
12     return sb.toString();
13 }
```

invertirMayusMinus(String) — aritmetica de caracteres

```
1  static boolean esMinuscula(char c){ return c>='a' && c<='z'; }
2  static boolean esMayuscula(char c){ return c>='A' && c<='Z'; }
3
4  static String invertirMayusMinus(String s){
5      StringBuilder sb = new StringBuilder(s.length());
6      for (int i = 0; i < s.length(); i++){
7          char c = s.charAt(i);
8          if (esMinuscula(c))      sb.append((char)(c - 'a' + 'A')); // min→MAY
9          else if (esMayuscula(c)) sb.append((char)(c - 'A' + 'a')); // MAY→min
10         else                    sb.append(c);                      // otros tal cual
11     }
12     return sb.toString();
13 }
```

normalizarEspacios(String) — colapsa y recorta

```
1  static String normalizarEspacios(String s){
2      StringBuilder out = new StringBuilder(s.length());
3      boolean espacioPrevio = false;
4      // Copiar dejando un solo espacio cuando hay varios
5      for (int i = 0; i < s.length(); i++){
6          char c = s.charAt(i);
7          if (c == ' '){
8              if (!espacioPrevio){ out.append(' '); espacioPrevio = true; }
9          } else { out.append(c); espacioPrevio = false; }
10     }
11     // Recortar borde inicial/final si quedo espacio
12     int ini = 0, fin = out.length();
13     while (ini < fin && out.charAt(ini) == ' ') ini++;
14     while (fin > ini && out.charAt(fin - 1) == ' ') fin--;
15     return out.substring(ini, fin);
16 }
```

ASCII 7-bit (0–127) en cuadrícula

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0x1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0x2	SP	!	"	#	\$	%	&	'	()	*	+	,	-	.	/
0x3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0x4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0x5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
0x6	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0x7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- ▶ SP = espacio (0x20). DEL = 0x7F.
- ▶ Bloques: dígitos '0'..'9' (0x30–0x39), mayúsculas 'A'..'Z' (0x41–0x5A), minúsculas 'a'..'z' (0x61–0x7A).
- ▶ Distancia fija: 'a' - 'A' = 32.
- ▶ Bases por fila (decimal): 0, 16, 32, 48, 64, 80, 96, 112.
- ▶ Offsets por columna (decimal): 0..15.

Tabla ASCII util (valores enteros)

Carácter	Código
'0'	48
'1'	49
'9'	57
'A'	65
'B'	66
'Z'	90
'a'	97
'b'	98
'z'	122

- ▶ Dígitos: '0' .. '9' \Rightarrow 48..57
- ▶ Mayúsculas: 'A' .. 'Z' \Rightarrow 65..90
- ▶ Minúsculas: 'a' .. 'z' \Rightarrow 97..122
- ▶ Distancia fija: 'a' - 'A' = 32

Atajo 1 — Un char es un numero

- ▶ Java promueve `char` a `int` en operaciones aritmeticas.
- ▶ Bloques contiguos: digitos 48–57, mayusculas 65–90, minusculas 97–122.
- ▶ Distancia fija: `'a' - 'A' = 32`. Permite mapear min↔MAY con sumas/restas.

Atajo 2 — $c - 'a' + 'A'$ (min→MAY)

```
1 char c = 'c'; // 99
2 char may = (char)('A' + (c - 'a')); // 65 + (99-97) = 67 -> 'C'
```

Formula general: $\text{may} = (\text{char})(\text{'A'} + (c - \text{'a'}))$, $\text{min} = (\text{char})(\text{'a'} + (C - \text{'A'}))$.

Atajo 3 — Cesar en 26 letras

```
1 int k = ((desp % 26) + 26) % 26; // normalizar negativo
2 char enc = (char)('A' + ((c - 'A' + k) % 26));
```

Pasos: offset desde 'A' \rightarrow sumar k \rightarrow modulo 26 \rightarrow volver a bloque.

Atajo 4 — Valor del dígito: $c - '0'$

```
1 int valor = c - '0'; // '7' - '0' = 55-48 = 7
2 cur = cur * 10 + valor; // parseo manual de "305"
```

Atajo 5 — Saltos y limites frecuentes

- ▶ **Avanzar sin solape:** `i = k + patron.length()` tras `indexOf`.
- ▶ **Bordes en substring:** usar `[ini, fin)` (fin excluido) con cuidado.
- ▶ **Relleno a izquierda:** calcular `faltan = ancho - s.length()` y construir con `StringBuilder`.
- ▶ **Colapso de espacios:** bandera `espacioPrevio` para no duplicar.

Atajo 6 — Pizarron numerico

```
1 // ASCII clave
2 'A'=65, 'Z'=90, 'a'=97, 'z'=122, '0'=48, '9'=57
3 // 'c' -> 'C'
4 'A' + ('c' - 'a') = 65 + (99-97) = 67 = 'C'
5 // Cesar: 'Z' + 2 -> 'B'
6 'A' + ((( 'Z'-'A') + 2) % 26) = 'B'
7 // Dígito: '7' -> 7
8 '7' - '0' = 55 - 48 = 7
```

Atajo 7 — ¿Cuándo NO usar estos atajos?

- ▶ Texto no-ASCII (acentos, ß, etc.): preferir `Character.toUpperCase/LowerCase`.
- ▶ Reglas locales (p. ej. turco): usar siempre `Locale.ROOT`.
- ▶ Comparaciones robustas: considerar normalización Unicode (NFC) si hay acentos.

Panorama de codificaciones

- ▶ **ASCII (US-ASCII, 7-bit)** — repertorio básico.
Referencia
- ▶ **ISO-8859-1 (Latin-1, 8-bit)** — Europa occidental.
Referencia
- ▶ **Windows-1252 (8-bit)** — “comillas inteligentes”, “€” y guiones tipográficos.
Referencia
- ▶ **UTF-8 (Unicode)** — 1–4 bytes, estándar de la web.
RFC 3629 (Unicode)
- ▶ **UTF-16 (Unicode)** — 2/4 bytes, pares sustitutos; notas sobre BOM.
FAQ UTF/BOM

Ejemplos prácticos (bytes por codificación)

Carácter	Unicode	ISO-8859-1	Win-1252	UTF-8
A	U+0041	41	41	41
ñ	U+00F1	F1	F1	C3 B1
€	U+20AC	(no)	80	E2 82 AC
—	U+2014	(no)	97	E2 80 94

En ISO-8859-1, “€” y “—” no existen; en Windows-1252 son 0x80 y 0x97; en UTF-8 usan 3 bytes.

Trampas comunes (codificaciones)

- ▶ **Mojibake** al abrir CP1252 como UTF-8: aparecen “Ã±”, “â€””, “â€œ”...
- ▶ “Comillas inteligentes” (U+2018..U+201D) y guiones (–, —) no existen en ISO-8859-1.
- ▶ **BOM**: UTF-8 puede tener BOM; UTF-16 casi siempre. Evita ambigüedad especificando el charset.
- ▶ En Java, especifica siempre el charset al leer/escribir:

```
1 Files.readString(path, java.nio.charset.StandardCharsets.UTF_8);  
2 Files.writeString(path, texto, java.nio.charset.StandardCharsets.UTF_8);
```


Glosario (núcleo)

- ▶ **Inmutabilidad**: `String` no cambia; nuevas cadenas en cada “modificación”.
- ▶ **StringBuilder**: construir en bucle; `length()` vs `capacity()`.
- ▶ **Índices y rangos**: base 0; `substring(i,j)` usa `[i, j)`.
- ▶ **indexOf/lastIndexOf**: devuelven índice o `-1`; `indexOf(pat,from)` y avance sin solape.
- ▶ **equals vs ==**: contenido vs referencia. **compareTo**: orden lexicográfico.

Glosario ampliado (I)

- ▶ **Punto de código:** identificador Unicode de un carácter (p.ej., U+00E1 = á).
- ▶ **Unidad de código:** bloque físico de codificación (UTF-8: byte; UTF-16: char).
- ▶ **Par sustituto:** dos char para un símbolo fuera del BMP (ej.,).
- ▶ **Grafema:** “carácter visible” para el usuario; puede ser varios puntos de código.
- ▶ **Carácter de control:** no imprimible (LF, CR, TAB); afecta lectura y saltos de línea.
- ▶ **CR/LF/CRLF:** fin de línea en Mac/Unix/Windows; unificar antes de procesar.
- ▶ **Whitespace:** conjunto de espacios en blanco (más que ' ').

Glosario ampliado (II)

- ▶ **Centinela:** valor especial; `indexOf` usa `-1` para “no encontrado”.
- ▶ **Off-by-one:** error de límites; recordar que `substring(i,j)` usa `[i,j)`.
- ▶ **Idempotente:** aplicar dos veces no cambia
(`normalizar(normalizar(s)) == normalizar(s)`).
- ▶ **Sensibilidad a Locale:** resultados dependen del idioma; para código, `Locale.ROOT`.
- ▶ **U+FFFD ():** carácter de reemplazo al decodificar bytes inválidos.
- ▶ **Endianness:** orden de bytes (LE/BE) en UTF-16/32; el BOM lo indica.