FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma, Desarrollo de Aplicaciones Web**
MÓDULO:	**Programación**

UNIDAD 4:
CADENAS DE CARACTERES Y EXPRESIONES REGULARES

CONTENIDOS

AUTORES: **Fernando Rodríguez Alonso**
Sonia Pasamar Franco

ÍNDICE DE CONTENIDOS

1. CARACTERES

El tipo **carácter** representa un símbolo o grafema de un alfabeto o silabario utilizado para la forma escrita de un lenguaje natural.

1.1. TIPOS DE CARACTERES

Existen diferentes **tipos de caracteres**:

1) **Carácter Gráfico (Imprimible).** Representa un símbolo o grafema que se puede escribir, imprimir o visualizar en un medio de forma que pueda ser leído por un humano.
2) **Carácter de Control (No Imprimible).** No representa un símbolo escrito. Se utiliza como señal de control para causar efectos sobre los caracteres imprimibles.

Los **caracteres gráficos** se dividen también en varios tipos:

- Letra Mayúscula: `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
- Letra Minúscula: `a b c d e f g h i j k l m n o p q r s t u v w x y z`
- Número Arábigo: `0 1 2 3 4 5 6 7 8 9`
- Signo de Puntuación: `. , : ; ' " ( ) [ ] { } ¿ ? ¡ !`
- Signo Matemático: `¬ + - ± × ÷ < = >`
- Signo Monetario: `$ ¢ £ ¤ ¥ €`

Los **caracteres de control** más utilizados son:

- **Nulo (`\0`).** Se utiliza en muchos lenguajes de programación para indicar el final de una cadena de caracteres y para representar una referencia que no apunta a ningún objeto en memoria.
- **Aviso (`\a`).** Provoca que un dispositivo emita un aviso o alarma, como una campana, un pitido sonoro o un parpadeo de pantalla.
- **Retroceso (`\b`).** Borra el anterior carácter imprimido en pantalla.
- **Tabulador Horizontal (`\t`).** Mueve el cursor de impresión al siguiente límite de tabulación de la línea.
- **Salto de Línea (`\n`).** Mueve el cursor de impresión una línea abajo o al principio de la línea siguiente abajo. Se utiliza como marcador de fín de línea en sistemas operativos UNIX.
- **Salto de Página (`\f`).** Provoca que una impresora eyecte el papel hasta el principio de la página siguiente o que un terminal de vídeo (consola) limpie la pantalla.
- **Retorno de Carro (`\r`).** Mueve el cursor de impresión al comienzo de la línea, permitiendo la sobrescritura de caracteres. Se utiliza como marcador de fín de línea en sistemas operativos Mac.

La combinación **retorno de carro + salto de línea (`\r\n`)** se utiliza como marcador de fín de línea en sistemas operativos Windows y en protocolos de capa de aplicación, como FTP, SMTP y HTTP.

1.2. CODIFICACIONES DE CARACTERES

Inicialmente, los caracteres utilizados en informática fueron los determinados por el idioma inglés. Poco a poco, se introdujeron otros caracteres pertenecientes a otros idiomas (francés, alemán, castellano, etc.). Por último, en un esfuerzo global, se definió el estándar Unicode, que incluye 144.762 caracteres distribuidos en 159 conjuntos de símbolos denominados scripts (cada uno puede ser usado en uno o más sistemas de escritura).

Las **codificaciones de caracteres** más utilizadas son:

1) **ASCII.** Fue publicado inicialmente en 1967 y está basado en el alfabeto del idioma inglés. Utiliza 7 bits para representar cada carácter. Por tanto, codifica 128 caracteres diferentes.
2) **ISO/IEC 8859 (ASCII extendido).** Fue publicado inicialmente en 1987 con la intención de extender la codificación ASCII añadiendo más caracteres procedentes de otros alfabetos latinos. Utiliza 8 bits para representar cada carácter. Por tanto, codifica 256 caracteres diferentes. Aún asi, es bastante limitado para representar todos los posibles símbolos de todos los idiomas y alfabetos existentes.
3) **Unicode.** Fue publicado inicialmente entre 1991 y 1992 con el objetivo de facilitar el tratamiento informático y la transmisión y visualización de textos de numerosos idiomas y disciplinas técnicas. Define cada símbolo o carácter mediante un nombre o punto de código. Incluye 144.762 símbolos distribuidos en diferentes planos y conjuntos de símbolos. Contiene tres formas de codificación:
   - **UTF-8.** Codifica con un byte los caracteres definidos con ASCII y codifica con 2, 3 o 4 bytes el resto.
   - **UTF-16.** Codifica cada carácter con 2 bytes, permitiendo representar 65.536 símbolos, los cuales pertenecen al plano multilingual básico (BMP). Es compatible con ASCII.
   - **UTF-32.** Codifica cada punto de control con 4 bytes. Está pensado para los ideogramas de Asia oriental.

Java tiene un tipo carácter denominado **`char`**, que utiliza la codificación UTF-16 de Unicode y que también es considerado por el compilador como un tipo entero, ya que internamente cada carácter se almacena o codifica con un número o código.

2. CADENAS DE CARACTERES

Una **cadena de caracteres** es una secuencia ordenada que está formada por cero o más caracteres. También puede incluir secuencias de escape que indican caracteres de control y otros caracteres expresados en notación Unicode.

2.1. OPERACIONES DE CREACIÓN

Las **operaciones de creación de cadenas** se realizan mediante constructores y métodos estáticos de la clase **`String`**.

Crear una cadena vacía: **`String()`**

Inicializa una cadena recién creada con una secuencia de caracteres vacía.

Crear una cadena con un valor inicial: **`String(String original)`**

Inicializa una cadena recién creada con la misma secuencia de caracteres que la indicada por el parámetro. Como resultado, la cadena recién creada será una copia de la cadena original.

Crear una cadena con un formato:

```
static String format(String cadenaFormato,
                     Object... argumentos)
```

Devuelve una nueva cadena formateada usando la cadena con formato indicada y los argumentos asociados a los especificadores de formato incluidos en dicha cadena con formato.

El número de argumentos referenciados por los especificadores de formato (`%d`, `%f`, `%s`, etc…) es variable y puede ser cero. Si hay más argumentos que especificadores de formato en la cadena, los argumentos extra son ignorados.

Lanza la excepción `IllegalFormatException` si la cadena con formato contiene una sintaxis ilegal o algún especificador de formato incompatible con un argumento dado, hay argumentos insuficientes comparados con los especificadores de formato de la cadena, o se producen otras situaciones ilegales.

2.2. OPERACIONES DE CONSULTA

Las **operaciones de consulta de cadenas** se realizan mediante métodos de objeto de la clase **`String`**.

Obtener la longitud de una cadena: **`int length()`**

Devuelve la longitud de la cadena. Esta longitud es igual al número de caracteres Unicode que contiene la cadena.

Obtener un carácter de una cadena: **`char charAt(int posicion)`**

Devuelve el carácter de la posición indicada de la cadena. El rango de posiciones dentro de cualquier cadena comienza en `0` y termina en `length()-1`. El primer carácter de la cadena se encuentra en la posición 0, el segundo en la posición 1, y así sucesivamente.

Lanza la excepción `IndexOutOfBoundsException` si el parámetro posición es negativo o es mayor o igual que la longitud de la cadena.

Obtener una subcadena de una cadena: `String substring(int posicionInicial, int posicionFinal)`

Devuelve la subcadena indicada de la cadena. La subcadena comienza en el carácter de la posición `posicionInicial` (incluido) y termina con el carácter de la posición `posicionFinal-1` (excluido). Así, la longitud de la subcadena será `posicionFinal-posicionInicial`.

Lanza la excepción `IndexOutOfBoundsException` si `posicionInicial` es negativo, o `posicionFinal` es mayor que la longitud de la cadena, o `posicionInicial` es mayor que `posicionFinal`.

Buscar un carácter en una cadena: `int indexOf(char caracter)`

Si la cadena contiene el carácter indicado, devuelve la posición de la primera ocurrencia del carácter en la cadena. Es decir, cuando el carácter aparece más de una vez en la cadena, devuelve la menor de las posiciones de dichos caracteres encontrados en la cadena.

Si la cadena no contiene el carácter indicado, devuelve `-1`.

Buscar un carácter en una cadena: `int lastIndexOf(char caracter)`

Si la cadena contiene el carácter indicado, devuelve la posición de la última ocurrencia del carácter en la cadena. Es decir, cuando el carácter aparece más de una vez en la cadena, devuelve la mayor de las posiciones de dichos caracteres encontrados en la cadena.

Si la cadena no contiene el carácter indicado, devuelve `-1`.

Buscar una subcadena en una cadena: `int indexOf(String subcadena)`

Si la cadena contiene la subcadena indicada, devuelve la posición de la primera ocurrencia de la subcadena en la cadena. Es decir, cuando la subcadena aparece más de una vez en la cadena, devuelve la menor de las posiciones de dichas subcadenas encontradas en la cadena.

Si la cadena no contiene la subcadena indicada, devuelve `-1`.

Buscar una subcadena en una cadena: `int lastIndexOf(String subcadena)`

Si la cadena contiene la subcadena indicada, devuelve la posición de la última ocurrencia de la subcadena en la cadena. Es decir, cuando la subcadena aparece más de una vez en la cadena, devuelve la mayor de las posiciones de dichas subcadenas encontradas en la cadena.

Si la cadena no contiene la subcadena indicada, devuelve `-1`.

Determinar si una cadena tiene un prefijo: `boolean startsWith(String prefijo)`

Devuelve verdadero si la cadena empieza con el prefijo indicado. Si el prefijo es la cadena vacía, también devuelve verdadero.

Devuelve falso en caso contrario.

Determinar si una cadena tiene un sufijo: `boolean endsWith(String sufijo)`

Devuelve verdadero si la cadena termina con el sufijo indicado. Si el sufijo es la cadena vacía, también devuelve verdadero.

Devuelve falso en caso contrario.

Comparar dos cadenas igualmente: `boolean equals(String otraCadena)`

Realiza la comparación de igualdad entre la cadena y la otra cadena indicada por parámetro.

Devuelve verdadero si la otra cadena no es `null` y representa la misma secuencia de caracteres que la cadena. Es decir, dos cadenas serán iguales cuando tienen el mismo número de caracteres y tienen el mismo carácter en cada posición de las mismas.

Devuelve falso en caso contrario.

Comparar dos cadenas lexicográficamente: `int compareTo(String otraCadena)`

Realiza la comparación lexicográfica entre la cadena y la otra cadena indicada por parámetro. La comparación se basa en los códigos Unicode de cada carácter de las cadenas.

Si la cadena precede lexicográficamente a la otra cadena, devuelve un número entero negativo.

Si la cadena es igual a la otra cadena, devuelve cero.

Si la cadena sucede lexicográficamente a la otra cadena, devuelve un número entero positivo.

2.3. OPERACIONES DE MANIPULACIÓN

Las **operaciones de manipulación de cadenas** se realizan mediante métodos de objeto de la clase **`String`**.

Convertir una cadena a letras minúsculas: `String toLowerCase()`

Convierte todos los caracteres de la cadena a letras minúsculas usando las reglas de la región geográfica, política o cultural (idioma y país) utilizada por el entorno de desarrollo.

Devuelve una nueva cadena con todos los caracteres de la cadena convertidos a letras minúsculas.

Convertir una cadena a letras mayúsculas: `String toUpperCase()`

Convierte todos los caracteres de la cadena a letras mayúsculas usando las reglas de la región geográfica, política o cultural (idioma y país) utilizada por el entorno de desarrollo.

Devuelve una nueva cadena con todos los caracteres de la cadena convertidos a letras mayúsculas.

Concatenar dos cadenas en otra cadena: `String concat(String otraCadena)`

Realiza la concatenación o unión de la cadena con la otra cadena indicada por parámetro.

Si la longitud de la otra cadena es `0`, devuelve la propia cadena.

Si la longitud de la otra cadena es mayor que `0`, devuelve una nueva cadena formada por los caracteres de la cadena seguidos de los caracteres de la otra cadena.

Dividir una cadena en varias subcadenas: `String[] split(String expresionRegular)`

Realiza la división o partición de la cadena en varias subcadenas usando como separador la expresión regular indicada por parámetro.

Devuelve un vector de cadenas formado por las subcadenas que se extraen al partir la cadena con la expresión regular. Si se generan subcadenas vacías al final, éstas no se incluirán en el vector de cadenas resultante.

Por ejemplo, la cadena `"boo:and:foo"` se divide de forma diferente según la expresión regular usada:

EXPRESIÓN REGULAR	RESULTADO
`":"`	`{ "boo", "and", "foo" }`
`"o"`	`{ "b", "", ":and:f" }`

Lanza la excepción `PatternSyntaxException` si la sintaxis de la expresión regular es inválida.

Eliminar espacios de una cadena: **`String trim()`**

Si la cadena es la cadena vacía, devuelve la propia cadena. Si el primer carácter y el último carácter de la cadena son ambos caracteres distintos del espacio en blanco (`' '`), también devuelve la propia cadena.

Si todos los caracteres de la cadena son espacios en blanco, devuelve una cadena vacía.

En cualquier otro caso, devuelve una nueva cadena que será una subcadena de la cadena, pero sin incluir todos sus espacios en blanco iniciales y finales. Se considera espacio en blanco a cualquier carácter menor o igual que `'\u0020'`.

Por ejemplo, al recortar la cadena `"  boo and foo  "`, se genera el resultado `"boo and foo"`.

Reemplazar un carácter de la cadena: **`String replace(char caracterBuscar, char caracterReemplazo)`**

Si el carácter a buscar no aparece ninguna vez en la cadena, devuelve la propia cadena.

En caso contrario, devuelve una nueva cadena con los caracteres de la cadena, pero en la que cada aparición u ocurrencia del carácter a buscar se ha cambiado por el carácter de reemplazo.

Reemplazar una subcadena de la cadena: **`String replace(String subcadenaBuscar, String subcadenaReemplazo)`**

Si la subcadena a buscar no aparece ninguna vez en la cadena, devuelve la propia cadena.

En caso contrario, devuelve una nueva cadena con los caracteres de la cadena, pero en la que cada aparición u ocurrencia de la subcadena a buscar se ha cambiado por la subcadena de reemplazo. El proceso de reemplazo se realiza desde el principio hasta el final de la cadena.

2.4. OPERACIONES DE CONVERSIÓN CON TIPOS NUMÉRICOS

Las **operaciones de conversión de tipos numéricos** (**`char`**, **`int`** y **`double`**) **a cadenas** se realizan mediante métodos estáticos de la clase **`String`**.

Convertir un carácter a cadena: **`static String valueOf(char caracter)`**

Devuelve una cadena de longitud 1 conteniendo el carácter indicado por parámetro.

Convertir un número entero a cadena: **`static String valueOf(int numEntero)`**

Devuelve una cadena que representa el número entero indicado por parámetro en base `10`.

Convertir un número real a cadena: **`static String valueOf(double numReal)`**

Devuelve una cadena que representa el número real indicado por parámetro en base `10` y con al menos 1 dígito para la parte fraccionaria.

Las **operaciones de conversión de cadenas a tipos numéricos** (**`int`** y **`double`**) se realizan mediante métodos estáticos de las clases envoltorio correspondientes (**`Integer`** y **`Double`**).

Convertir una cadena a número entero: **`static int parseInt(String cadena)`**

Convierte la cadena en base `10` a un número entero con signo. El primer carácter de la cadena puede ser `'-'` o `'+'`, para indicar un valor negativo o positivo. Los restantes caracteres deben ser dígitos numéricos (`0-9`).

Devuelve un número entero como resultado de la conversión de la cadena en base `10`.

Lanza la excepción `NumberFormatException` si la cadena contiene algún carácter no válido que impida generar el número entero.

Convertir una cadena a número real: **`static int parseDouble(String cadena)`**

Convierte la cadena con notación decimal o científica a un número real con signo.

En notación decimal, el primer carácter de la cadena puede ser `'-'` o `'+'`, para indicar un valor negativo o positivo, seguido de al menos un dígito numérico (`0-9`) para la parte entera, un punto (`'.'`) y terminando con al menos un dígito (`0-9`) para la parte fraccionaria.

En notación científica, el primer carácter de la cadena puede ser `'-'` o `'+'`, para indicar un valor negativo o positivo, seguido de un dígito numérico (`0-9`), un punto (`'.'`) y al menos un dígito (`0-9`) para la mantisa, seguido de la letra `'E'` o `'e'`(base 10) y terminando con un signo `'-'` o `'+'` y al menos un dígito (`0-9`) para el exponente.

Devuelve un número real como resultado de la conversión de la cadena con notación decimal o científica.

Lanza la excepción `NumberFormatException` si la cadena contiene algún carácter no válido que impida generar el número real.

3. CASOS PRÁCTICOS

3.1. PASO A MAYÚSCULAS

El siguiente ejemplo muestra una forma de pasar a mayúsculas las letras minúsculas de una cadena de caracteres. Para ello, tras solicitar al usuario la cadena a transformar, se recorre carácter a carácter y cuando se encuentre uno que corresponda con una minúscula, se reemplaza por su correspondiente mayúscula utilizando el código ASCII.

```
import entrada.Teclado;

public class Mayusculas {

    public static void main(String[] args) {
        int i;
        char caracter;
        String cadena;
        String cadenaMayusculas = "";
        cadena = Teclado.leerCadena("Introduce cadena: ");
        for(i = 0; i < cadena.length(); i++) {
            caracter = cadena.charAt(i);
            if(caracter >= 'a' && caracter <= 'z') {
                caracter = (char)(caracter - 32);
            }
            else if(caracter == 'ñ') {
                caracter = 'Ñ';
            }
            cadenaMayusculas = cadenaMayusculas + caracter;
        }
        System.out.println(cadenaMayusculas);
    }
}
```

3.2. OPERACIONES DE OBTENCIÓN

El siguiente ejemplo muestra el uso de diferentes métodos de la clase String para obtener información de una cadena:

- Carácter que se encuentra en una posición.
- Subcadena delimitada por posiciones.
- Posición de un carácter dado.

```java
public class MetodosObtencion {

    public static void main(String[] args) {
        int posicion;
        char caracter;
        String subcadena;
        String cadena = "Fundamentos de algoritmia.";
        //Posición de caracteres
        caracter = cadena.charAt(3);
        System.out.println("El carácter en la posición 3 es " + caracter);
        //Subcadenas
        subcadena = cadena.substring(15);
        System.out.println("Cadena desde posición 15: " + subcadena);
        subcadena = cadena.substring(12, 14);
        System.out.println("Cadena entre posiciones 12 y 14: " + subcadena);
        //Búsqueda de caracteres
        posicion = cadena.indexOf('a');
        System.out.println("La primera ocurrecia de a es: " + posicion);
        posicion = cadena.lastIndexOf('a');
        System.out.println("La última ocurrecia de a es: " + posicion);
    }

}
```

3.3. OPERACIONES DE COMPARACIÓN

El siguiente ejemplo muestra el uso de métodos de comparación de la clase String:

- Comparación de igualdad.
- Comparación lexicográfica, según la tabla ASCII.

```java
import entrada.Teclado;

public class MetodosComparacion {

    public static void main(String[] args) {
        String cadena1;
        String cadena2;
        cadena1 = Teclado.leerCadena("Introduce cadena: ");
        cadena2 = Teclado.leerCadena("Introduce otra cadena: ");
        if(cadena1.equals(cadena2)) {
            System.out.println("Las dos cadenas son iguales");
        }
        else {
            System.out.println("Las dos cadenas son distintas");
        }
        System.out.println("ORDEN SEGÚN ASCII:");
        if(cadena1.compareTo(cadena2)<0) {
            System.out.println(cadena1);
            System.out.println(cadena2);
        }
        else {
            System.out.println(cadena2);
            System.out.println(cadena1);
        }
    }

}
```