Strings - Manejo de Fechas

String

La clase String en Java es una clase fundamental que representa una secuencia de caracteres Unicode.

Las cadenas son inmutables en Java, lo que significa que una vez que se crea una cadena, no se pueden cambiar sus caracteres.

Aquí hay algunos de los métodos más comunes de la clase String:

- length(): Devuelve la longitud de la cadena.
- charAt(int index): Devuelve el carácter en la posición index de la cadena.
- substring(int startIndex) o substring(int startIndex, int endIndex): Devuelve una subcadena de la cadena original, que comienza en el índice startIndex y termina en el índice endIndex - 1.
- concat(String str): Concatena una cadena con otra cadena.
- replace(char oldChar, char newChar): Reemplaza todas las apariciones de oldChar en la cadena con newChar.
- toLowerCase() o toUpperCase(): Devuelve una versión en minúsculas o mayúsculas de la cadena, respectivamente.
- trim(): Devuelve una versión de la cadena sin espacios al comienzo o al final.

Estos son solo algunos de los métodos disponibles en la clase String.

La clase String es muy versátil y se utiliza comúnmente en Java para representar texto, incluyendo nombres de variables, nombres de métodos, mensajes de error y mucho más.

La clase String proporciona una amplia gama de métodos que permiten la comparación, concatenación, búsqueda y reemplazo de texto. Además, la clase String es un tipo de objeto en Java, lo que significa que se puede utilizar en conjunto con otros objetos para crear aplicaciones más complejas.

A continuación se describen en detalle los conceptos relacionados con la clase String en Java:

- Declaración: Las cadenas de caracteres se pueden declarar como objetos de la clase String o como literales de cadena usando comillas dobles.

- Concatenación: La concatenación de cadenas se puede realizar mediante el operador + o el método concat().
- Longitud: La longitud de una cadena se puede determinar usando el método length().
- Subcadena: Se puede extraer una parte de una cadena usando el método substring().
- Búsqueda: Se puede buscar un carácter o subcadena en una cadena usando el método indexOf().
- Comparación: Las cadenas se pueden comparar mediante el método equals() o compareTo().
- Mayúsculas/minúsculas: La conversión a mayúsculas o minúsculas se puede realizar usando los métodos toUpperCase() o toLowerCase().
- Reemplazo: Se puede reemplazar una subcadena en una cadena usando el método replace().
- División: Se puede dividir una cadena en partes usando el método split().
- Expresiones regulares: Java incluye soporte para expresiones regulares a través de la clase Pattern y Matcher.

Usos más comunes de la clase String:

- Representación de texto: es la forma más común de representar texto en Java. Por ejemplo, se puede utilizar para almacenar nombres de archivos, nombres de variables, mensajes de error y mucho más.

String nombre = "Ada Lovelace";

- Concatenación de cadenas: proporciona una serie de métodos que permiten concatenar dos o más objetos de tipo String para crear una nueva cadena.

Ejemplo:

String saludo = "Hola, " + nombre + "!";

- Búsqueda y reemplazo de texto: La clase String también proporciona métodos que permiten buscar y reemplazar texto dentro de un objeto de tipo String.

Ejemplo:

String texto = "Hola, mundo!";
texto = texto.replace("mundo", nombre);

- Comparación de cadenas: proporciona métodos que permiten comparar dos objetos de tipo String para determinar si son iguales o no.

Ejemplo:

```
String password = "secret";
if (password.equals("secret")) {
        System.out.println("La contraseña es correcta.");
}
```

- Conversión a otros tipos: también se puede utilizar para convertir otros tipos de datos a texto y viceversa.

Ejemplo:

```
int numero = 123;
String numeroComoTexto = Integer.toString(numero);
```

Estos son solo algunos ejemplos de cómo se puede utilizar la clase String en Java.

Sí, en Java existen otras formas de representar texto, como la clase **StringBuilder** y la clase **StringBuffer**. Sin embargo, la clase String es la más común y se utiliza en la mayoría de los casos debido a su simplicidad y facilidad de uso.

StringBuilder

La clase StringBuilder es una clase que permite crear y manipular objetos de tipo cadena en Java.

A diferencia de la clase String, que es inmutable, la clase StringBuilder es mutable, lo que significa que sus valores pueden ser modificados después de ser creados.

La clase StringBuilder es útil cuando se requiere hacer muchas operaciones de concatenación, ya que es más eficiente que la clase String en este tipo de operaciones. Además, la clase StringBuilder es más fácil de usar que otras clases para manipular texto, como la clase StringBuffer.

Ejemplo:

```
StringBuilder sb = new StringBuilder();
sb.append("Hello");
sb.append(" ");
sb.append("World");
System.out.println(sb.toString()); // imprime "Hello World"
StringBuilder sb = new StringBuilder("Hello");
sb.insert(5, " World");
System.out.println(sb.toString()); // imprime "Hello World"
```

Es importante tener en cuenta que, a diferencia de la clase String, la clase StringBuilder no es segura en cuanto a concurrencia, lo que significa que no es adecuada para usarse en entornos multithread. Para este tipo de entornos, se recomienda utilizar la clase StringBuffer.

StringBuffer

La clase StringBuffer es una clase de Java que permite crear y manipular objetos de tipo cadena. A diferencia de la clase String, que es inmutable, la clase StringBuffer es mutable, lo que significa que sus valores pueden ser modificados después de ser creados.

La clase StringBuffer es similar a la clase StringBuilder en muchos aspectos, incluyendo su facilidad de uso y eficiencia en operaciones de concatenación. Sin embargo, a diferencia de la clase StringBuilder, la clase StringBuffer es segura en cuanto a concurrencia, lo que significa que se puede utilizar en entornos multithread sin problemas de seguridad.

Ejemplo:

```
StringBuffer sb = new StringBuffer();
sb.append("Hello");
sb.append(" ");
sb.append("World");
System.out.println(sb.toString()); // imprime "Hello World"
```

```
StringBuffer sb = new StringBuffer("Hello"); sb.insert(5, " World");
System.out.println(sb.toString()); // imprime "Hello World"
```

En general, la clase StringBuffer es una opción adecuada cuando se requiere manipular texto en entornos multithread, mientras que la clase StringBuilder es una opción adecuada para entornos singlethread y operaciones de concatenación. Sin embargo, en muchos casos, la clase String es suficiente y es la opción más sencilla y fácil de usar.

Manejo de fechas

En Java, el manejo de fechas se realiza a través de la clase java.util.Date y su subclase java.util.Calendar. Estas clases permiten representar fechas y horas, y realizar operaciones como la obtención de la fecha y hora actual, la comparación de fechas, la adición o sustracción de días, meses o años, entre otras.

Algunos ejemplos de uso de la clase java.util.Date:

```
Date fechaActual = new Date();
System.out.println(fechaActual); // imprime la fecha y hora actual
```

```
Date fechaFutura = new Date(System.currentTimeMillis() + 1000 * 60 * 60 * 24 * 7);
System.out.println(fechaFutura); // imprime la fecha y hora actual más una semana
```

```
if (fechaActual.before(fechaFutura)) {
        System.out.println("La fecha actual es antes que la fecha futura");
}
```

Algunos ejemplos de uso de la clase java.util.Calendar:

```
Calendar cal = Calendar.getInstance();
System.out.println(cal.getTime()); // imprime la fecha y hora actual
cal.add(Calendar.DATE, 7);
System.out.println(cal.getTime()); // imprime la fecha y hora actual más una semana
if (cal.before(Calendar.getInstance())) {
        System.out.println("La fecha actual es antes que la fecha futura");
}
```

En Java 8 y versiones posteriores, se introdujo una nueva API de fechas y horas llamada java.time que proporciona una forma más fácil y moderna de trabajar con fechas y horas. La API de java.time ofrece una gran cantidad de clases y métodos para representar, manipular y formatear fechas y horas, incluyendo LocalDate, LocalTime, LocalDateTime entre otros.

A continuación, se muestran algunos ejemplos de uso de la API de java.time:

```
LocalDate fechaActual = LocalDate.now();
System.out.println(fechaActual); // imprime la fecha actual
LocalDate fechaFutura = fechaActual.plusDays(7);
System.out.println(fechaFutura); // imprime la fecha actual más una semana

if (fechaActual.isBefore(fechaFutura)) {
        System.out.println("La fecha actual es antes que la fecha futura"); }
```

LocalDateTime fechaHoraActual = Local

Algunos ejemplos de uso de la API de java.time:

- ➢ Obtención de la fecha y hora actual:

```
LocalDateTime fechaHoraActual = LocalDateTime.now(); System.out.println(fechaHoraActual);
```

- ➢ Adición o sustracción de días, meses o años:

```
LocalDate fechaActual = LocalDate.now();
LocalDate fechaFutura = fechaActual.plusDays(7).plusMonths(2).plusYears(1);
System.out.println(fechaFutura);
```

- ➢ Comparación de fechas:

```
LocalDate fecha1 = LocalDate.of(2020, Month.JANUARY, 1);
LocalDate fecha2 = LocalDate.of(2021, Month.JANUARY, 1);
if (fecha1.isBefore(fecha2)) {
        System.out.println("fecha1 es antes que fecha2");
}
```

- Formateo de fechas:

```
LocalDate fecha = LocalDate.now();
DateTimeFormatter formato = DateTimeFormatter.ofPattern("dd/MM/yyyy");
String fechaFormateada = fecha.format(formato);
System.out.println(fechaFormateada);
```

- Conversión de fechas:

```
LocalDate fecha = LocalDate.now();
Date fechaVieja = Date.from(fecha.atStartOfDay().atZone(ZoneId.systemDefault()).toInstant());
System.out.println(fechaVieja);
```

Estos son solo algunos ejemplos de uso de la API de java.time, pero hay muchas otras características y métodos que puedes explorar para mejorar tus habilidades en el manejo de fechas en Java.

Lectura requerida:
Sznajdleder, Pablo Augusto: El gran libro de Java a Fondo Editorial Marcombo.
Ceballos, Francisco Javier: Java 2. Curso de Programación, Editorial Ra-Ma

Lectura ampliatoria:
https://www.oracle.com/ar/java/