Programación en Java:

String

1º DAM

Tipo primitivo char vs Clase Character

En Java contamos con dos posibles formas de manejar caracteres individuales: el tipo primitivo char y la clase Character

Los caracteres se declaran con comillas simple.

La clase Character tiene los siguientes métodos:

isUpper, isLower	isLetter
toUpperCase, toLowerCase	isDigit

Character Métodos

```java
char car_c = 'c';

Character.isUpperCase(car_c);   // False
Character.isLowerCase(car_c);   // True
Character.isDigit(car_c);       // False
Character.isLetter(car_c);      // True

char car_D = 'D';

Character.isUpperCase(car_D);   // True
Character.isLowerCase(car_D);   // False
Character.isDigit(car_D);       // False
Character.isLetter(car_D);      // True
```

```java
char car_9 = '9';

Character.isUpperCase(car_9);   // False
Character.isLowerCase(car_9);   // False
Character.isDigit(car_9);       // True
Character.isLetter(car_9);      // False

char car_$ = '$';
Character.isUpperCase(car_$);   // False
Character.isLowerCase(car_$);   // False
Character.isDigit(car_$);       // False
Character.isLetter(car_$);      // False
```

Cadenas: String

Para definir una cadena usamos comillas dobles.

Las cadenas son objetos INMUTABLES. Un objeto inmutable es aquel cuyo estado no se puede cambiar una vez construído.

```
h = "HOLA";
h = h + " Y ADIOS";
```

¿Cuántos objetos se crean con este código?

1. Objeto String "HOLA
2. La concatenación de la cadena con h + " Y ADIOS"
3. Finalmente, se crea otro objeto con el valor "HOLA Y ADIOS". La variable h ahora apunta a este nuevo objeto.

   Recuerda que no se modifica el contenido del objeto original, sino que se crean nuevos objetos con los valores actualizados.

String Declaración

Para crear una variable que contenga una cadena de texto podemos hacerlo de diferentes modos:

```
// Declaración e inicializacion de cadena vacía
String cadenaVacia_1 = "";
String cadenaVacia_2 = new String("");
String cadenaVacia_3 = new String();
```

String Métodos

La clase String tiene los siguientes métodos:

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
compareTo()	Compares two strings lexicographically	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
endsWith()	Checks whether a string ends with the specified character(s)	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Encodes this String into a sequence of bytes using the named charset.	byte[]
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
startsWith()	Checks whether a string starts with specified characters	boolean
subSequence()	Returns a new character sequence that is a subsequence of this sequence	CharSequence
substring()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the primitive value of a String object	String

String Métodos

- length(): devuelve el tamaño de la cadena de texto
- charAt(posicion): devuelve el carácter que se encuentra en la posición especificada (comenzando por el 0)

```
String cadena = "Cadena de texto uno";
int longitud = cadena.length();
char quintoCaracter = cadena.charAt(5);



System.out.println(
        String.format("La longitud de la cadena es %d", longitud)
        );
System.out.println(
        String.format("La letra en la posición quinta es %c", quintoCaracter)
        );
```

String Métodos

Concatenar:

- Al concatenar cadenas creamos nuevos elementos porque las cadenas en Java son Inmutables.
- Puede hacerse de dos formas:
    - concat(contenido)
    - operador **+**

```
String cadena1="Hola";
String cadena2 = cadena1+" y adiós";
String cadena3 = cadena1.concat(", buenos días");
System.out.printf("cadena1: %s %n", cadena1);
System.out.println("cadena1:" + cadena1);
System.out.printf("cadena2: %s %n", cadena2);
System.out.printf("cadena3: %s %n", cadena3);
```

String Métodos

Búsqueda:

- Si no se encuentra la cadena o valor, los métodos devuelven -1
- Si encuentra el valor devuelve la posición, comenzando en 0 y finalizando en n-1

indexOf(valor, posicion) *lastIndexOf(valor, posicion)*
indexOf(valor) *lastIndexOf(valor)*

valor a buscar
posición desde que empieza a buscar

```
String cadena = "Hello planet earth, a great planet.";
System.out.println(cadena.indexOf("planet"));
System.out.println(cadena.lastIndexOf("planet"));
System.out.println(cadena.indexOf("e", 5));
System.out.println(cadena.lastIndexOf("e",5));
```

String Métodos

Contiene/Empieza o termina:

- Devuelve true si la palabra/letra a buscar se encuentra en la cadena
- Puede buscarse al principio/fin o en cualquier posición

```
boolean contiene    = cadena.contains("de");    //True

//  Similar a
contiene = cadena.indexOf("de")!=-1;

boolean empieza     = cadena.startsWith("Cad"); //True

//  Similar a
contiene = cadena.substring(0, "Cad".length()).equals("Cad");

boolean termina     = cadena.endsWith("uno");   // True

//  Similar a
contiene = cadena.indexOf("uno", (cadena.length()-"uno".length()-1))!=-1;
```

String Métodos

Reemplazar:

- Reemplazar es similar al anterior en el sentido de que cambia la expresión indicada por otra, pero no modifica la cadena de texto original, sino que crea una nueva.

  - *replace(elementoBuscado, cadenaDeReemplazo)*
  - *replaceAll(elementoBuscar, cadenaDeReemplazo)*

- Pueden utilizarbuscarse y reeemplazarse caracteres (char) o palabras (CharSequence, String)

```
String cadena = "Hello planet earth, a great planet.";
String cadena2 = cadena.replaceAll("et", "ET");
System.out.println(cadena);
System.out.println(cadena2);
```

String Métodos

Formato:

- *toUpperCase()*: convierte a MAYÚSCULAS 2.
- *toLowerCase()*: convierte a minusculas
- *trim()* ⇒ elimina espacios iniciales y finales (no intermedios)

```
String iesNombre = "     IeS JaCaRanDa     ";

String IES             = iesNombre.toUpperCase();
String ies             = iesNombre.toLowerCase();
String iesSinEspacios  = iesNombre.trim();
```

```
▸ iesNombre     "  IeS JaCaRanDa  " (id=36)
▸ IES           "  IES JACARANDA  " (id=37)
▸ ies           "  ies jacaranda  " (id=38)
▸ iesSinEspaci  "IeS JaCaRanDa" (id=39)
```

String Métodos

Subcadenas:

- *substring (posicionInicial):* recorta la cadena desde la posición establecida (incluida) hasta el final.
- *substring (posicionInicial, posicionFinal):* igual que el anterior hasta el valor de la posicionFinal -1 ⇒ Generan una nueva cadena, no modifica la original

```
String cadena = "Hello planet earth, great planet.";
System.out.println(cadena.substring(20,25));
// Devuelve great
System.out.println(cadena.substring(20));
// Devuelve great planet.
```

String Métodos

Comparación de cadenas:

- *==* : realiza la comparación a nivel de objeto. Es decir, determina si ambos objetos son iguales.
- *equals:* compara los caracteres dentro del objeto String. Su contenidos

```
String c1 = "Hola";
String c2 = new String(c1);
System.out.println(c1 + " == " + c2 + " -> " + (c1 == c2));
System.out.println(c1 + " equals " + c2 + " -> " +
c1.equals(c2));
```

String Métodos

Comparación de cadenas:

compareTo(String cad)

Devuelve:

- 0 si son iguales
- <0 Si la primera es menor que la segunda
- >0 Si la primera es mayor que la segunda

```
String string1 = "Holaaaa";
String string2 = "Holaaaa";
System.out.println(string1.compareTo(string2));
```

String Métodos

División:

```
String[] split(String regex)
String[] split(String regex, int
limiteDeSeparaciones)
```

El método devuelve una tabla de subcadenas utilizando el delimitador definido en regex.

```
String str = "Los lunes me encanta Java";
String[] words = str.split(" ");
String dirección = "192.168.0.1:8080
String [] partes = str.split(":");
```

String Métodos

1. ¿QUé haría este código?

```
String strMain ="Odio los lunes de Enero";
String[] tabla = strMain.split("\\s");
System.out.println(Arrays.toString(tabla));
```

1. Utiliza el método split para filtrar todos los números de una cadena
2. Imagina que queremos obtener los nombres de usuarios a partir de sus correos. Tenemos algo así

```
String correos =
"usuario1@dominio.com,usuario2@dominio.com,usuario3@dominio2.com,usuario4@dominio.net"
```

y queremos devolver una tabla con usuario1, usuario2, usuario3 y usuario4

String Expresiones Regulares

¿Qué es una expresión regular?

Una expresión regular es un patrón para encontrar una cadena en el texto.

Las expresiones regulares se escriben con letras y números, así como con metacaracteres, que son caracteres que tienen un significado especial en la sintaxis de las expresiones regulares.

```
String regex = "java"; // The pattern is "java";
```

```
String regex = "\\d{3}"; // The pattern is three digits;
```

String Expresiones Regulares

¿Cómo se usan?

1. Crear expresión regular
2. Compilamos ese patrón. Para ello usamos el método estático compile de la clase **Pattern (definimos el patrón)**

```
public static Pattern compile (String literal)
```

```
public static Pattern compile (String literal, int flags)
```

1. Verificamos si una cadena cumple el patrón con el método matcher devolviendo un objeto tipo **Matcher (buscamos el patrón)**

```
Matcher matcher = pattern.matcher("Aquí busco patrón");
```

String Expresiones Regulares

```
public static Pattern compile (String literal, int flags);
```

Donde:

- Literal Se corresponde con la expresión regular
- flags puede tener los siguientes valores:
  - `Pattern.CASE_INSENSITIVE` Para no diferenciar entre mayúsculas y minúscúlas
  - `Pattern.LITERAL` - Caracteres especiales no tendrán ningún significado especial.

String Expresiones Regulares

Ejemplo:

Buscaremos cadenas que empiecen por p pudiendo seguirle una o varias qs:

```
String regex = "pq*";
Pattern patron = Pattern.compile(regex);
Matcher busca = patron.matcher("pqq");
boolean encontrado = busca.matches();
System.out.println(encontrado?
        "Sigue patrón": "No sigue patrón");
```

String Expresiones Regulares

Es tu turno:

Investiga qué es una expresión regular y completa la tabla.

Expresión	Descripción
.	
^expresión	
expresión$	
[abc]	
[abc][12]	
[^abc]	
[a-z1-9]	
A|B	
\d	
\D	
\s	
\S	
\w	
\W	
\b	
*	
?	

String Expresiones Regulares

Cuantificadores para una expresión regular

Tenemos caracteres especiales que nos van a indicar el número de repeticiones de la expresión, la siguiente tabla muestra los caracteres:

Cuantificador	Descripción
n+	Encuentra cualquier string con al menos un «n»
n*	Encuentra cero o más ocurrencias de n
n?	Encuentra en el string la aparición de n cero o una vez
n{x}	Encuentra la secuencia de n tantas veces como indica x.
n{x,}	Encuentra una secuencia de X tantas veces como indica n

String Expresiones Regulares

Metacaracteres en una expresión regular

Metacaracter	Descripción
|	Símbolo para indicar OR.
.	Encuentra cualquier carácter
^	Sirve para hacer match al principio del string
$	Hace match al final de un String
\d	Encuentra dígitos
\s	Busca un espacio
\b	Hace match al principio de una palabra.
\uxxxx	Encuentra el carácter Unicode especificado por el número hexadecimal xxxx

Expresión regular	Descripción
`.`	Hace match con cualquier caracter
`^regex`	Encuentra cualquier expresión que coincida al principio de la línea.
`regex$`	Encuentra la expresión que haga match al final de la línea.
`[abc]`	Establece la definición de la expresión, por ejemplo la expresión escrita haría match con a, b o c.
`[abc][vz]`	Establece una definición en la que se hace match con a, b o c y a continuación va seguido por v o por z.
`[^abc]`	Cuando el símbolo ^ aparece al principio de una expresión después de [, negaría el patrón definido. Por ejemplo, el patrón anterior negaría el patrón, es decir, hace match para todo menos para la a, la b o la c.
`[e-f]`	Cuando hacemos uso de -, definimos rangos. Por ejemplo, en la expresión anterior buscamos hacer match de una letra entre la e y la f.
`Y|X`	Establece un OR, encuentra la Y o la X.

String: Caracteres especiales

Carácter	Nombre
`\b`	Borrado a la izquierda
`\n`	Nueva línea
`\r`	Retorno de carro
`\t`	Tabulador
`\f`	Nueva página
`\'`	Comilla simple
`\"`	Comilla doble
`\\`	Barra invertida

https://refactorizando.com/expresiones-regulares-java/


¿Dudas?