BAILADEV

JAVA: STRING MÉTODOS

MÉTODO CONSTRUCTOR *(creación de un objeto de tipo cadena, llamando a la clase String)*

- **String** miCadena = **new String(“Mi cadena”)**;

EXCEPCIONES: Si nuestra variable de tipo **String** almacena un valor **null** y queremos hacer uso de algunos métodos propuestos en este documento, nos lanzará una excepción de tipo → **NullPointerException** proveniente de la clase **NullPointerException** que se encuentra en el paquete **java** subpaquete **lang** tal que el acceso completo al documento de la excepción es → **java.lang.NullPointerException**.

TIPO DE RETORNO: BOOLEAN

MÉTODO → contains()

Descripción

- Este método nos servirá para comprobar la existencia de una parte de la cadena o de toda si es el caso; para que pueda comprobar, necesita recibir una secuencia de caracteres a buscar en nuestra cadena base.
  Devolverá **true** si la secuencia de caracteres coincide con la cadena base, de lo contrario devolverá **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.contains(“*cadena*”)**

USO del MÉTODO

EJEMPLO VÁLIDO → “Hola”.contains(“Ho”) = true ← Devuelve
EJEMPLO VÁLIDO → “Hola”.contains(“la”) = true ← Devuelve
EJEMPLO VÁLIDO → “Hola”.contains(“e”) = false ← Devuelve

Argumentos: 1-requerido (tipo cadena) → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

MÉTODO → contentEquals()

Descripción

- Este método nos servirá para comprobar la existencia de toda la cadena, a diferencia del método **contains()** que solo buscará una parte coincidente, por lo que el método **contentEquals()** también buscará la misma igualdad en la clase **StringBuffer**, dando lugar a que se tenga dos comprobaciones, para que pueda comprobar, este método, al igual que el método **contains()**, necesita recibir una secuencia de caracteres a buscar en nuestra cadena base.
  Devolverá **true** si la secuencia de caracteres coincide tanto con la cadena base como en la clase **StringBuffer**, en palabras resumidas, debe ser la cadena a evaluar **EXACTAMENTE** igual a la cadena base, de lo contrario devolverá **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.contentEquals(“*cadena*”)**

USO del MÉTODO

EJEMPLO VÁLIDO → “Hola”.contentEquals(“Hola”) = true ← Devuelve
EJEMPLO VÁLIDO → “Hola”.contentEquals(“la”) = false ← Devuelve
EJEMPLO VÁLIDO → “Hola”.contentEquals(“e”) = false ← Devuelve
EJEMPLO VÁLIDO → “Perro”.contentEquals(“Perro”) = true ← Devuelve
EJEMPLO VÁLIDO → “Perro”.contentEquals(“perro”) = false ← Devuelve

Argumentos: 1-requerido (tipo cadena) → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

NOTA: La clase mencionada “**StringBuffer**” es como un **String**, solo que podremos modificarla ubicándola en el paquete **java.lang**, también tenemos la interfaz “**CharSequence**” que es una secuencia de caracteres legibles, que también lo vamos a poder encontrar en el paquete **java.lang**.

MÉTODO → endsWith()

Descripción

- Este método nos servirá para constatar de que la secuencia de caracteres termine con el de la cadena base.
  Si la cadena ingresada en su argumento coincide con el cierre o fin de la cadena base, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.endsWith("*cadena*")**

USO del MÉTODO

EJEMPLO VÁLIDO → "Hola".endsWith("Hola") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".endsWith("la") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".endsWith("ola") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".endsWith("Ho") = false ← Devuelve

Argumentos: **1-requerido (tipo cadena)** → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

MÉTODO → startsWith()

Descripción

- Este método nos servirá para constatar de que la secuencia de caracteres empiece con el de la cadena base.
  Si la cadena ingresada en su argumento coincide con el de apertura o inicio de la cadena base, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.startsWith("*cadena*")**

USO del MÉTODO

EJEMPLO VÁLIDO → "Hola".startsWith("Hola") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".startsWith("la") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".startsWith("ola") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".startsWith("Ho") = false ← Devuelve

Argumentos: **1-requerido (tipo cadena)** → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

MÉTODO → equals()

Descripción

- Este método nos servirá para comprobar dos cadenas, no importa si es creada de forma natural o por medio del objeto **String**, no tiene en cuenta cómo ha sido creada, sino su contenido, a diferencia del operador de igualdad **==**, que este sí le importa más cómo ha sido creada que su contenido.
  Si la cadena ingresada en su argumento coincide con la cadena base a evaluar, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.equals("*cadena*")**

USO del MÉTODO → Suponiendo lo siguiente: **String cadena= new String("Hola");**

EJEMPLO VÁLIDO → "Hola".equals("Hola") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola". equals("hola") = false ← Devuelve
EJEMPLO VÁLIDO → "Hola". equals(cadena) = true ← Devuelve
EJEMPLO VÁLIDO → "Hola" == cadena = false ← Devuelve

Argumentos: **1-requerido (tipo cadena)** → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

CONSEJO: Si vas a evaluar dos cadenas tenés que acostumbrarte a usar este método, en lugar de igualdad ya que vas a tener problemas de comparación, por más que el contenido sea igual.

MÉTODO → equalsIgnoreCase()

Descripción

- Este método nos servirá para comprobar dos cadenas, no importa si es creada de forma natural o por medio del objeto **String**, no tiene en cuenta cómo ha sido creada, sino su contenido, a diferencia del operador de igualdad **==**, que este sí le importa más cómo ha sido creada que su contenido.
  Este método a diferencia de **equals()**, lo que hará será **IGNORAR** las mayúsculas y minúsculas.
  Si la cadena ingresada en su argumento coincide con la cadena base a evaluar, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.equalsIgnoreCase(*"cadena"*)**

USO del MÉTODO → Suponiendo lo siguiente: String cadena= new String("hoLa");

EJEMPLO VÁLIDO → "Hola".equalsIgnoreCase("hola") = true ← Devuelve
EJEMPLO VÁLIDO → "HOLA".equalsIgnoreCase("hola") = true ← Devuelve
EJEMPLO VÁLIDO → "HOLA".equalsIgnoreCase("hoLA") = true ← Devuelve
EJEMPLO VÁLIDO → "HOLA".equalsIgnoreCase(cadena) = true ← Devuelve

Argumentos: **1-requerido (tipo cadena)** → Espera recibir como argumento una cadena/secuencia de caracteres a evaluar con la cadena base, donde esta secuencia es el criterio de búsqueda sobre la cadena base.

CONSEJO: Si vas a evaluar dos cadenas tenés que acostumbrarte a usar este método, en lugar de igualdad ya que vas a tener problemas de comparación, por más que el contenido sea igual.

MÉTODO → isEmpty()

Descripción

- Este método nos servirá para comprobar si una cadena se encuentra vacía o no.
  Si la cadena se encuentra vacía, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.isEmpty()**

USO del MÉTODO

EJEMPLO VÁLIDO → "Hola".isEmpty() = false ← Devuelve
EJEMPLO VÁLIDO → " ".isEmpty() = false ← Devuelve
EJEMPLO VÁLIDO → "".isEmpty() = true ← Devuelve

CONSEJO: Tenés que saber lo siguiente, si nuestra cadena llegase a contener espacios en blanco, esto quiere decir que tiene contenido, por lo que nos devolvería **false**; por lo que su longitud debe ser **0** al momento de fijarnos la cantidad de caracteres que esta dispone para que sea **true**.

MÉTODO → matches()

Descripción

- Este método nos servirá para comprobar si una cadena coincide con una expresión regular (**REGEX**) dada, esta expresión es una simple cadena que contará con símbolos especiales que harán avanzada nuestra forma de buscar. El resultado generado es el mismo que **Pattern.matches(regex, "cadena")**.
  Si la expresión regular dada coincide con la cadena base, nos devolverá **true**, de lo contrario **false**.

FÓRMULA DEL MÉTODO: ***miCadena*.matches(*"REGEX"*)**

USO del MÉTODO → *Usamos ".*[…].*" para preguntarle a la cadena si tiene esos caracteres*

EJEMPLO VÁLIDO → "Hola".matches(".*[hHa].*") = true ← Devuelve
EJEMPLO VÁLIDO → "Hola".matches(".*[zA1].*") = false ← Devuelve
EJEMPLO VÁLIDO → "Hola".matches(".*[za1].*") = true ← Devuelve

Argumentos: **1-requerido (tipo cadena)** → Espera recibir como argumento una cadena con símbolos **REGEX** que evaluará con la cadena base, donde este **REGEX** será el criterio de búsqueda sobre la cadena base.

CONSEJO: Las expresiones regulares son algo un poco complejas, por lo que hay que estudiarlas a parte ya que dependiendo del lenguaje utilizado su aplicación puede que sea distinta.

MÉTODO → regionMatches()

Descripción

- Este método nos servirá para comprobar si dos regiones de cadenas coinciden o son iguales. Este método tiene dos variantes de este método, es decir, una es un método de prueba que distingue entre mayúsculas y minúsculas y la otra ignora el método que distingue entre mayúsculas y minúsculas. Este método hará comparación de dos subcadenas de tipo **String**.

TOMA COMO BASE STR1 → Como punto de partida debemos hacer uso de una cadena base para que podamos comparar, para esto tomaremos una cadena declarada del objeto String.

FÓRMULA DEL MÉTODO: (OPC 1) ***STR_1.***regionMatches***(INDEX1, STR2, INDEX2 , CARACTERES, LONGITUD);***
(OPC 2) ***STR_1.***regionMatches***(IGNORAR, INDEX1, STR2, INDEX2 , CARACTERES, LONGITUD);***

USO del MÉTODO

String cadena1 **= new String("Hola") | "Hola";**
String cadena2 **= new String("HOLA") | "HOLA";**

cadena1.regionMatches(0, cadena2, 1, 3) = false ← Devuelve
cadena1.regionMatches(true, 0, cadena2, 0, 3) = true ← Devuelve

Argumentos: **1-IGNORAR MAYÚSCULAS Y MINÚSCULAS - OPCIONAL (tipo boolean)** → nos devolverá **true** si estas subcadenas representan secuencias de caracteres que son iguales, ignorando mayúsculas y minúsculas si y solo si el **IGNORE CASE** es **true**.
1-INDEX1 - REQUERIDO (tipo entero) → Vamos a especificarle su índice inicial, es decir, desde la posición de qué carácter va a comenzar a comparar.
2-STR2 - REQUERIDO (objeto String) → Vamos a tomar otra cadena para hacer comparación con la cadena STR_1, ambas deben ser del objeto String.
3-INDEX2 - REQUERIDO (tipo entero) → Vamos a especificarle su índice inicial, es decir, desde la posición de qué carácter va a comenzar a comparar.
4-LONGITUD - REQUERIDO (tipo entero) → Vamos a especificar la cantidad de caracteres que vamos a estar comparando.

Más información:

- Repositorio del JDK (**importante**): https://github.com/openjdk/jdk
- https://openjdk.org/projects/jdk/
- https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/java/lang/String.html
- https://docs.oracle.com/javase/8/docs/api/java/lang/CharSequence.html
- https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuffer.html
- https://www.w3schools.com/java/java_ref_string.asp