CADENAS DE CARACTERES

Índice de contenido

1 LA CLASE String

Los strings son secuencias de caracteres, incluyendo letras (mayúsculas y minúsculas), signos, espacios, caracteres especiales, etc. En palabras simples, los strings sirven para guardar palabras, oraciones, etc.

En java los Strings son objetos, tipo String. Para crearlos, la manera más sencilla es:

```
String hola = "Hello World";
```

Los Strings, al ser objetos, también pueden ser inicializados como tales, a través del constructor:

```
String s = new String(); // String vacio
String s = new String("Hola"); // String "Hola"
```

Los Strings soportan el conjunto de caracteres Unicode. Para obtener información detallada sobre Unicode, visite www.unicode.org .

Para utilizarlos se puede “castear” un número entero (la representación decimal del carácter según UNICODE) a un “char”.

```
char c = (char) 36; // signo pesos
```

Además se pueden utilizar ciertos caracteres especiales, anteponiendo \:

- Barra invertida //
- Tabulación horizontal /t
- Salto de línea /n
- Comillas simples /'
- Comillas dobles /”
- …

La clase String es inmutable, por tanto una vez creado un objeto String no puede ser modificado. Hay algunos métodos en la clase String que parece que modifican el valor del objeto String, sin embargo lo que hacen es crear y devolver un nuevo objeto String con el resultado de la operación.

1.1 Operaciones especiales con cadenas

Dada la importancia de las cadenas en la programación, Java permite que a objetos de la clase String se pueda aplicar una sintaxis especial facilita su uso y legibilidad de código, para tres operaciones:

1. **Creación automática de nuevos objetos String a partir de literales alfanumérico.**

   Ya que por cada literal alfanumérico que haya en el código de un programa, Java construye un objeto String, esto permite que se pueda crear e inicializar un objeto String a partir de un literal.

   ```
   String s1=”Hola”;
   ```

```
  //Su equivalente explicito es:

  //String s1=new String(”Hola”);
```

2. **Concatenación de cadenas.**

   Permite aplicar el operador + para concatenar dos cadenas.

```
String s1=”Juan”;
String s2= “Luis”;
String s3 = “Juan”+”Luis”;
//su equivalente explicito es String s3 = s1.concat(”Luis”);
String s3 = “Juan”+ “ “+”Luis”;
//su equivalente explicito es String s3 = s1.concat(”
”).concat(“Luis);
```

3. **Contenación de cadenas con otros tipos de datos.**
   Cada vez que Java encuentra el operador + y uno de los operandos es String, convierte automáticamente a String el otro objeto operando.

```

int edad=23;
System.out.println(“La edad media es de “+ edad +” años”);
```

   Con edad lo que hace es convertir automáticamente el entero a su representación de cadena dentro de un objeto String, entonces esta cadena resultante se concatena.

   Nota: Cuando Java convierte datos en su representación de cadena durante la concatenación, lo hace llamando a una versión sobrecargada del método de conversión de cadenas valueOf().

   El método valueOf() está sobrecargado para los tipos primitivos y para el tipo Object.
   - Para los tipos primitivos valueOf() devuelve una cadena que contiene el texto equivalente legible por humanos del valor primitivo pasado como parámetro.
   - Para objetos valueOf() llama al método toString() de ese objeto.

   Se verá más adelante...

1.2 Metodos de la clase String

La clase String tiene una gran variedad de métodos útiles. La lista completa puede encontrarse en la documentación oficial (http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html).

A continuación explicamos los métodos que más se utilizan.

1.2.1 int length()

Sencillamente, retorna el largo (la cantidad de caracteres) de un String.

```
String texto1 = "Hola";
String texto2 = "(/.,;>.)";
```

```
String texto3 = "";
String texto4 = " ";
int n = texto1.length(); // 4
int m = texto2.length(); // 8
int p = texto3.length(); // 0
int q = texto4.length(); // 1
```

1.2.2 char charAt(int index)

Retorna el carácter que se encuentra en la posición dada por index. Si index es mayor o igual que el largo del String, el programa se caerá al ejecutar, aunque el compilador no le avisara al respecto (no es capaz de prever dicho comportamiento), y al compilar tampoco habrá errores. Por esta razón cada vez que utilice este método debe primero revisar el largo del String.

```
String texto = "¡Hola Juanito!";
char c1 = texto.charAt(0); // '¡'
char c2 = texto.charAt(1); // 'H'
char c3 = texto.charAt(13); // '!'
```

```
char c4 = texto.charAt(20); // error
char c5 = texto.charAt( texto.length() - 1 ); // '!'
```

1.2.3 boolean startsWith(String s)

Devuelve true si el String comienza con el String s entregado como parámetro. La coincidencia debe ser exacta (se distingue mayúsculas de minúsculas, por ejemplo, y los caracteres con tilde no son iguales a los que no lo llevan).

```
String texto = "¡Hola Juanito!";
boolean b1 = texto.startsWith("¡"); // true
boolean b2 = texto.startsWith("¡Hola ") // true
boolean b3 = texto.startsWith("¡hola Juanito!") // false
boolean b4 = texto.startsWith("¡Hola Juanito! ") // false
boolean b5 = texto.startsWith("¡Hola Juanito!") // true
```

1.2.4 boolean startsWith(String s, int k)

Como el método anterior, pero comienza a revisar desde el índice (posición) k, sin importar lo que haya antes. Si k == 0, entonces el método es equivalente al anterior.

```
String texto = "¡Hola Juanito!";
boolean b1 =
boolean b2 = texto.startsWith(" Juanito!", 5); // true
boolean b3 = texto.startsWith("a", 7); // false
boolean b4 =texto.startsWith("¡", 20); // false
```

1.2.5 int indexOf(char c)

Devuelve un entero que indica la posición donde aparece por primera vez el carácter c. Si el carácter no aparece en todo el String, devuelve -1.

```
String texto = "¡Hola Juanito!";
int n = texto.indexOf('o'); // 2
int m = texto.indexOf(' '); // 5
int p = texto.indexOf('¡'); // 0
int q = texto.indexOf('z'); // -1
```

1.2.6 int indexOf(String s)

Similar al anterior, solo que devuelve la posición donde aparece por primera vez un determinado String.

```
String texto = "En la arboleda un arbolito";
int n = texto.indexOf("arbol"); // 6
int m = texto.indexOf("árbol"); // -1
int p = texto.indexOf("boli"); // 20
int q = texto.indexOf("bol"); // 8
int indexOf(char c, int fromIndex) y int indexOf(String s, int
fromIndex)
```

Como el método indexOf(String s), pero comienza a buscar desde la posición fromIndex.

int lastIndexOf(char c), int lastIndexOf(String s), int lastIndexOf(char c, int fromIndex), int lastIndexOf(String s, int fromIndex)

Como indexOf(char c), pero en vez de buscar la primera aparición del char c, busca la última. Retorna -1 si no lo encuentra. Se definen de manera análoga a los anteriores

1.2.7 boolean equals(String s)

Retorna true cuando ambos String son exactamente iguales. Esto quiere decir que el largo de ambos es el mismo, y que en cada posición encontramos el mismo carácter.

```
String texto1 = "casa";
String texto2 = "caza";
String texto3 = "casi";
String texto4 = "casa";
String texto5 = "CaSa";
boolean b1 = texto1.equals(texto3); //false
boolean b2 = texto4.equals(texto1); //true
boolean b3 = texto3.equals(texto1); //false
boolean b4 = texto1.equals(texto5); //false
```

1.2.8 boolean equalsIgnoreCase(String s)

Retorna true cuando ambos String son iguales, sin importar mayúsculas y minúsculas.

```
String texto1 = "casa";
String texto2 = "caza";
String texto3 = "casi";
String texto4 = "casa";
String texto5 = "CaSa";
boolean b1 = texto1.equalsIgnoreCase(texto3); //false
boolean b2 = texto4.equalsIgnoreCase(texto1); //true
boolean b3 = texto3.equalsIgnoreCase(texto1); //false
boolean b4 = texto1.equalsIgnoreCase(texto5); //true
```

1.2.9 int compareTo(String s)

Este método compara lexicográficamente dos String. Parte en el primer par de caracteres hasta encontrar un par de char distintos. Si encuentra dos caracteres distintos, retorna la diferencia entre ellos (recuerde que char funciona como un número entero). Si uno de los String comienza con el otro, retorna el número de caracteres adicionales que tiene. Si ambos String son iguales, retorna 0.

```
String texto = "radiografía";
int n = texto.compareTo("radio"); // 6
int m = texto.compareTo("radiología"); // -5
int p = texto.compareTo("rabia"); // 2
int q = texto.compareTo("ralentizar"); // -8
int r = texto.compareTo("radiografia"); // 132
int s = texto.compareTo("radiografía"); // 0
```

1.2.10 String substring(int inicio, int fin)

Retorna una parte del String: desde la posición inicio hasta la posición fin.

```
String texto = "desenfreno";
String subTexto = texto.substring(5, 10); // "freno"
```

1.2.11 String replaceAll(String aReemplazar , String reemplazo)

Reemplaza todas las ocurrencias de aReemplazar por reemplazo.

```
String t1 = "El auto volaba rápido";
String t2 = t1.replaceAll("auto", "avión")); // "El avión volaba
rápido"
```

1.2.12 Formateado de String

Para producir una salida con strings y numbers formateados se puede utilizar printf() o format().

El método printf() de System.out, devuelve un objeto ***PrintStream.***

```
System.out.printf("The value of the float " +
                  "variable is %f, while " +
                  "the value of the " +
                  "integer variable is %d, " +
                  "and the string is %s",
```

```
                                floatVar, intVar, stringVar);
```

La clase String proporciona el método static format() que devuelve un objeto ***String***.

public static String format(String format, Object... args)

```
String fs;
fs = String.format("The value of the float " +
                   "variable is %f, while " +
                   "the value of the " +
                   "integer variable is %d, " +
                   " and the string is %s",
                   floatVar, intVar, stringVar);
System.out.println(fs);
```

Este método permite obtener una cadena que puede ser reutilizada.

1.3 Conversiones entre String y otros tipos de datos

1.3.1 Wrappers

Los Wrappers (envoltorios) son clases diseñadas para ser un complemento de los tipos primitivos.

En efecto, los tipos primitivos son los únicos elementos de Java que no son objetos. Esto tiene algunas ventajas desde el punto de vista de la eficiencia, pero algunos inconvenientes desde el punto de vista de la funcionalidad.

Las clases Wrapper también proporcionan métodos para realizar otras tareas con lo tipos primitivos, tales como conversión con cadenas de caracteres en uno y otro sentido.

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Existe una clase Wrapper para cada uno de los tipos primitivos numéricos que extienden de la clase Number, esto es, existen las clases Byte, Short, Integer, Long, Character, Float y Double (obsérvese que los nombres empiezan por mayúscula, siguiendo la nomenclatura típica de Java).

1.3.2 Conversiones de otros tipos de datos a String

1. **conversión implícita**, al encontrarse la cadena en un contexto en que se requiere un string.

   int i;
   String s1= “ “+ i;

2. método static de la clase **String valueOf()**

   String s2= String.valueOf(i);

3. Utilizar el método toString de la clase

Ejemplo:

```
public class ToStringDemo {

    public static void main(String[] args) {
        double d = 858.48;
        String s = Double.toString(d);

        int dot = s.indexOf('.');

        System.out.println(dot + " digits " +
            "before decimal point.");
        System.out.println( (s.length() - dot - 1) +
            " digits after decimal point.");
    }
}
```

Método toString() de la clase Object. Redefinición para las clases del programador

Todas las clases implementan el método toString() porque está definido en la clase Object. Las clases han de sobreescribir este método ya toString() que es la manera de obtener la representación en cadena legible por humanos de objetos de clases creadas por el programador.

Es decir implementar toString() en una nueva clase es devolver un objeto String que contenga la cadena legible que describa adecuadamente al objeto de la clase. Además permite así que mediante la cadena resultante se integre totalmente el objeto en el entorno de programación de Java (pudiéndose utilizar por ejemplo en sentencias println()).

1.3.3 Conversión de String a otros tipos de datos.

Se puede utilizar los métodos de los Wrappers para convertir a tipos de datos primitivos. Por ejemplo Integer.parseInt o Float.parseFloat

1.4 Más métodos que manipulan String

Otros métodos de Class Strings para manipular Strings

Method	Description
`public` String `trim()`	Devuelve una copia del objeto this String, sin los espacios en blanco al inicio y al final de la cadena.

`public String toLowerCase()` `public String toUpperCase()`	Returns a copy of this string converted to lowercase or uppercase. If no conversions are necessary, these methods return the original string.
`String[] split(String regex)` `String[] split(String regex, int limit)`	Searches for a match as specified by the string argument (which contains a regular expression) and splits this string into an array of strings accordingly. The optional integer argument specifies the maximum size of the returned array. Regular expressions are covered in the lesson titled "Regular Expressions."
`CharSequence subSequence(int beginIndex, int endIndex)`	Returns a new character sequence constructed from beginIndex index up until endIndex - 1.

Metodos para buscar caracteres y subcadenas en un String

Method	Description
`int indexOf(int ch)` `int lastIndexOf(int ch)`	Returns the index of the first (last) occurrence of the specified character.
`int indexOf(int ch, int fromIndex)` `int lastIndexOf(int ch, int fromIndex)`	Returns the index of the first (last) occurrence of the specified character, searching forward (backward) from the specified index.
`int indexOf(String str)` `int lastIndexOf(String str)`	Returns the index of the first (last) occurrence of the specified substring.
`int indexOf(String str, int fromIndex)` `int lastIndexOf(String str, int fromIndex)`	Returns the index of the first (last) occurrence of the specified substring, searching forward (backward) from the specified index.
`boolean contains(CharSequence s)`	Returns true if the string contains the specified character sequence. **Note:** `CharSequence` is an interface that is implemented by the `String` class. Therefore, you can use a string as an argument for the `contains()` method.

Métodos para reemplazar caracteres o subcadenas en un String

Method	Description
`String replace(char oldChar, char newChar)`	Returns a new string resulting from replacing all occurrences of oldChar in this string with

	newChar.
`String replace(CharSequence target, CharSequence replacement)`	Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.
`String replaceAll(String regex, String replacement)`	Replaces each substring of this string that matches the given regular expression with the given replacement.
`String replaceFirst(String regex, String replacement)`	Replaces the first substring of this string that matches the given regular expression with the given replacement.

Ejemplo:

```
public class Filename {
    private String fullPath;
    private char pathSeparator,
                 extensionSeparator;

    public Filename(String str, char sep, char ext) {
        fullPath = str;
        pathSeparator = sep;
        extensionSeparator = ext;
    }

    public String extension() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        return fullPath.substring(dot + 1);
    }

    // gets filename without extension
    public String filename() {
        int dot = fullPath.lastIndexOf(extensionSeparator);
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(sep + 1, dot);
    }

    public String path() {
        int sep = fullPath.lastIndexOf(pathSeparator);
        return fullPath.substring(0, sep);
    }
}

Esta clase crea un objeto de la clase anterior y llama a sus métodos.

public class FilenameDemo {
    public static void main(String[] args) {
        final String FPATH = "/home/user/index.html";
        Filename myHomePage = new Filename(FPATH, '/', '.');
        System.out.println("Extension = " + myHomePage.extension());
        System.out.println("Filename = " + myHomePage.filename());
        System.out.println("Path = " + myHomePage.path());
    }
}
```

Métodos para comparar cadenas y subcadenas

Method	Description
`boolean endsWith(String suffix)` `boolean startsWith(String prefix)`	Returns `true` if this string ends with or begins with the substring specified as an argument to the method.
`boolean startsWith(String prefix, int offset)`	Considers the string beginning at the index `offset`, and returns `true` if it begins with the substring specified as an argument.
`int compareTo(String anotherString)`	Compares two strings lexicographically. Returns an integer indicating whether this string is greater than (result is > 0), equal to (result is = 0), or less than (result is < 0) the argument.
`int compareToIgnoreCase(String str)`	Compares two strings lexicographically, ignoring differences in case. Returns an integer indicating whether this string is greater than (result is > 0), equal to (result is = 0), or less than (result is < 0) the argument.
`boolean equals(Object anObject)`	Returns `true` if and only if the argument is a `String` object that represents the same sequence of characters as this object.
`boolean equalsIgnoreCase(String anotherString)`	Returns `true` if and only if the argument is a `String` object that represents the same sequence of characters as this object, ignoring differences in case.
`boolean regionMatches(int toffset, String other, int ooffset, int len)`	Tests whether the specified region of this string matches the specified region of the String argument. Region is of length `len` and begins at the index `toffset` for this string and `ooffset` for the other string.
`boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)`	Tests whether the specified region of this string matches the specified region of the String argument. Region is of length `len` and begins at the index `toffset` for this string and `ooffset` for the other string. The boolean argument indicates whether case should be ignored; if true, case is ignored when comparing

	characters.
`boolean matches(String regex)`	Tests whether this string matches the specified regular expression. Regular expressions are discussed in the lesson titled "Regular Expressions."

Ejemplo:

```
public class RegionMatchesDemo {
  public static void main(String[] args) {
    String searchMe = "Green Eggs and Ham";
    String findMe = "Eggs";
    int searchMeLength = searchMe.length();
    int findMeLength = findMe.length();
    boolean foundIt = false;
    for (int i = 0; i <= (searchMeLength - findMeLength); i++) {
       if (searchMe.regionMatches(i, findMe, 0, findMeLength)) {
          foundIt = true;
          System.out.println(searchMe.substring(i, i + findMeLength));
          break;
       }
    }
    if (!foundIt)
      System.out.println("No match found.");
  }
}
```

1.5 CARÁCTERÍSTICAS DE STRING

1. **Diferencias entre el método equals y el operador ==**

```
String cad1=”PUBLICIDAD”;
String cad2= new String(”PUBLICIDAD”);
if(cad1.equals(cad2))
      System.out.print(“SON IGUALES”);
else
      System.out.print(“SON DIFERENTES”);

if(cad1==cad2)
      System.out.print(“SON IGUALES”);
else
      System.out.print(“SON DIFERENTES”);
```

En el primer caso compara las dos secuecias de caracteres. → “SON IGUALES”
En el segundo caso compara las referencias. → “SON DIFERENTES”

Si hacemos:
cad1=cad2

```
if(cad1==cad2)
  System.out.print(“SON IGUALES”);
else
```

```
System.out.print("SON DIFERENTES");
```

Se obtendrá "SON IGUALES"

2. Cuando se compara un objeto String con un literal alfanumérico también se ha de utilizar el método equals(), en lugar del operador ==.
(En muchos casos es indistinto utilizar el operador o el método, pero hay situaciones en que no lo resultados pueden ser inesperados. La siguiente explicación ejemplifica esto)

Java asegura que todas las cadenas literales iguales usan un único objeto (por lo que en estos casos se podrían utilizar operadores de comparación, pero no es aconsejable)

```
String c1 = "Hola";
String c2 = "Hola";
if (c1==c2)
   System.out.println("c1==C2 es verdad");
else
   System.out.println("c1==C2 es falso");
if (c1.equals(c2))
   System.out.println("c1.equals(c2) es verdad");
else
   System.out.println("c1.equals(c2) es falso");
if(c1=="Hola")
   System.out.println("c1==\"Hola\" es verdad");
else
   System.out.println("c1==\"Hola\" es falso");

String c3 = "HolaAdios";

System.out.println("c1 = "+ c1);
System.out.println("c3 = "+ c3);
System.out.println("c1+\"Adios\" = "+ c1+"Adios");

if (c3==c1+"Adios")
   System.out.println("c3==c1 es verdad");
else
   System.out.println("c3==c1 es falso");//ya que no estamos comparando dos
   //literales que si compartirian el mismo objeto
```

3. La clase String es inmutable, por tanto una vez creado un objeto String no puede ser modificado. Hay algunos métodos en la clase String que parece que modifican el valor del objeto String, sin embargo lo que hacen es crear y devolver un nuevo objeto String con el resultado de la operación.

4. El método toString() lo contienen todas la clases de java ya que lo heredan de la clase Object (que devuelve *clase@hasdcode*).
Es una buena práctica de programación sobreescribir este método en cada clase que creemos de manera que ofrezcamos una cadena de caracteres entendible por un humano que represente el

contenido de un objeto de la clase.
Por ejemplo para la clase Alumno puede la cadena resultante podría ser:
“nombre: Juan Perez ciclo=DAW curso=1”

Otro ejemplo para un punto en un eje de coordenadas sería:

```
public class Punto {
     int x; int y;
     public Punto(int x, int y) {
          this.x = x; this.y = y;
     }
     public String toString() {
          return “Punto[“ + x + ”,” + y +”]”;
     }
     public static void main(String args[]) {
          Punto punto = new Punto(2,3);
          System.out.println( "visualizar datos
          del punto”+ punto );
     }
}
```

Un punto importante a tener en cuenta es que hay métodos, tales como ***System.out.println()***, que exigen que su argumento sea un objeto de la clase ***String***. Si no lo es, habrá que utilizar algún metodo que lo convierta en ***String***.
“***locale”*** es la forma que java tiene para adaptarse a las peculiaridades de los idiomas distintos del inglés: acentos, caracteres especiales, forma de escribir las fechas y las horas, unidades monetarias, etc.

2 LA CLASE StringBuilder

Una vez que se crea un objeto String , su contenido ya no puede variar. Ahora hablaremos sobre las características de la clase StringBuilder para crear y manipular información de cadenas dinámicas; es decir, cadenas que pueden modif carse. Cada objeto StringBuilder es capaz de almacenar varios caracteres especificados por su capacidad. Si se excede la capacidad de un objeto StringBuilder , ésta se expande de manera automática para dar cabida a los caracteres adicionales. La clase StringBuilder también se utiliza para implementar los operadores + y += para la concatenación de objetos String .

https://docs.oracle.com/javase/tutorial/java/data/buffers.html