CADENAS DE CARACTERES

Introducción

En los programas escritos hasta ahora hemos utilizado los tipos primitivos: char, byte, int, float, double y boolean. Estos bastan para implementar muchas aplicaciones, sobre todo las relacionadas con datos numéricos, pero proporcionan pocas herramientas para trabajar con texto. El tipo char, que almacena un solo carácter, es insuficiente para manejar textos complejos.

Por texto entendemos una palabra, una frase e incluso uno o más párrafos de cualquier longitud. En definitiva, un texto, como por ejemplo este mismo párrafo, es una secuencia de caracteres, de ahí que también se le denomine cadena de caracteres o, por economía del lenguaje, simplemente cadena.

Para manipular textos disponemos en la API de las clases Character y String -ambas ubicadas en el paquete java.lang-, que proporcionan multitud de funcionalidades para trabajar con un solo carácter la primera y con textos de cualquier longitud la segunda.

Tipo primitivo char

De forma general un carácter se define como una letra -de cualquier alfabeto-, un número, un ideograma o cualquier símbolo. En Java un carácter o literal carácter se escribe entre comillas simples (' '). Algunos ejemplos de ellos son: 'p', 'ñ', 'y', '7', 'O' '#'.

Unicode

Mediante un teclado podemos escribir ciertos caracteres, como 'a', pero esto no es posible para otros, como '♡'. Para solventar este problema, un conglomerado de empresas fundó el Unicode Consortium, un organismo que, mediante un comité técnico, diseñó y mantiene un estándar de codificación de caracteres denominado Unicode.

Este identifica cada carácter mediante un número entero único, llamado code point, cuyo valor se puede representar en decimal o en hexadecimal. Para evitar confusión cuando el code point se representa en hexadecimal se le antepone la secuencia U+ o \u, de forma general, aunque Java solo permite la segunda. Otra particularidad de la representación de un code point en hexadecimal

es que siempre se utilizan, como mínimo, 4 dígitos, completando con ceros por la izquierda si fuera necesario.

El esquema de codificación Unicode comprende un total de 1114112 posibles code points, que según la representación usada tomarán valores en el rango de Odec a 1114111dec, en decimal, o valores

en el rango de Ohex a 10ffffhex en hexadecimal. Lo que requiere un tamaño de 3 bytes para poder albergar todos los posibles valores.

A la hora de seleccionar un carácter es posible usar su codificación Unicode o el propio carácter si es posible escribirlo mediante el teclado. Por ejemplo, el carácter 'a' puede escribirse pulsando la tecla adecuada del teclado o mediante su code point en decimal (97)

o en hexadecimal (\u0061). Veamos la forma de asignar 'a' a una variable de tipo char,

```
char c;

c = 'a'; //directamente mediante el teclado

c = 97; //usando el code point en decimal
```

c = '\u0061'; //o con el code point en hexadecimal

La única forma de designar el carácter '♡' es mediante su code point.

char c = '\u2661'; //o bien, c = 9825;

System.out.println (c); //muestra un ♡

Para codificar cualquier code point necesitamos 3 bytes, por lo tanto, ¿cómo es posible que podamos asignar un code point a un tipo primitivo char (2 bytes)? La respuesta es que no todos los code points pueden asignarse a char. El problema surge porque, en un principio, los code points de Unicode usaban solamente 2 bytes para codificar todos los caracteres; por lo tanto, el tipo char se definió acorde a este tamaño. Con el tiempo, el tamaño del code point ha crecido para poder identificar la enorme cantidad de símbolos que se han ido añadiendo.

En consecuencia, solo los code points cuyo valor es inferior o igual a 65535 —\uFFFF—pueden asignarse a una variable de tipo char. Para valores mayores de code point hemos de utilizar variables de tipo

int, que tiene un tamaño de 4 bytes y dispone de espacio suficiente para albergar cualquier code point.

Para conocer la codificación de cualquier carácter necesitamos recurrir a las tablas diseñadas por el Unicode Consortium o bien a las bases de datos de caracteres. Estas tablas agrupan los caracteres según el alfabeto (latino, braille, etc.) o por el conjunto de símbolos al que pertenecen (matemáticos, jeroglíficos egipcios, etc.).

Por ejemplo, si deseamos trabajar en Java con el ideograma ♡,lo primero que tenemos que hacer es buscar su code point en las tablas de caracteres, donde encontramos que el code point que lo identifica es 9825 o \u2661.

Ejemplos de codificación Unicode

Carácter	Code point	
	decimal	hexadecimal
A	65	\u0041
a	97	\u0061
Ψ	936	\u03A8
♡	9825	\u2661
7	55	\u0037

Secuencias de escape

Un carácter precedido de una barra invertida (\) se conoce como secuencia de escape. Al igual que los caracteres escritos mediante la codificación Unicode, representan un único carácter, pero en este caso poseen un significado especial.

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

Veamos algunos ejemplos:

```
char c = '\'';

System.out.println (c); //muestra una comilla simple:

C = '\"' ;

System.out.println (c); //muestra una comilla doble: "
```

c = '\t'; //tabulador. Al ser invisible lo representamos con |_____|

System.out.println("1" + c + "2"); //muestra "1|_____|2" ;

Conversión char → int

Cada code point no es más que un número entero, representado en decimal o en hexadecimal. El hecho de que un carácter se identifique con un número crea una estrecha relación entre el tipo char y el tipo int. Es posible asignar un valor entero a una variable

de tipo char (siempre y cuando el valor del entero esté comprendido entre 0 y 65535) y asignar un carácter a una variable de tipo int, ya que Java se encarga de realizar las conversiones oportunas (el tipo int representa los números en decimal por defecto).

Veamos algunos ejemplos:

char c = '\'';

System.out.println (c); //muestra una comilla simple:

C = '\"' ;

System.out.println (c); //muestra una comilla doble: "

c = '\t'; //tabulador. Al ser invisible lo representamos con |_____|

System.out.println("1" + c + "2"); //muestra "1|_____|2" ;

Clasificación de caracteres

Un carácter puede clasificarse dentro de algunos de los grupos siguientes:

- Dígitos: este grupo está formado por los caracteres '0', '1' ..., '9'.
- Letras: formado por todos los elementos del alfabeto, tanto en minúscula ('a', 'b'...) como en mayúscula ('A', 'B'...).
- Caracteres blancos: como el espacio o el tabulador, entre otros.
- Otros caracteres: signos de puntuación, matemáticos, etcétera.

Los **métodos de Character** para verificar si un carácter pertenece a alguno de estos grupos devuelven un booleano: true en caso de que pertenezca o false en caso contrario.

Esto métodos son:

■boolean isDigit (char c): indica si el carácter c es un dígito. Devuelve true en caso afirmativo y false en caso contrario.

■ boolean isLetter (char c): determina si el carácter pasado como parámetro es una letra (minúscula o mayúscula)

■ boolean isLetterOrDigit (char c): indica si el carácter es una letra o un dígito

■ boolean isLowerCase (char c): especifica si c es una letra y, además, está en minúscula.

■ boolean isUpperCase (char c): funciona igual que el método anterior, pero indicando si el carácter es una letra mayúscula.

■ boolean isSpaceChar (char c): devolverá true si el carácter utilizado como parámetro de entrada es el espacio (' ')

■ boolean isWhitespace (char c): amplía el método anterior y determina si el carácter pasado es cualquier carácter blanco. Los caracteres que hacen que el método devuelva true son:

- Espacio en blanco ('_'): se teclea mediante la barra espaciadora.

- Retorno de carro ('\r'): dependiendo del sistema operativo, este carácter tendrá distinto comportamiento al imprimirse.

- Nueva línea ('\n'): es el carácter que se consigue al pulsar la tecla Intro.

- Tabulador ('\t'): equivale a varios espacios en blanco. Se genera con la tecla Tab.

- Otros: existen otros caracteres considerados blancos como el tabulador vertical, el separador de ficheros, etc., aunque están en desuso.

Conversión

Los métodos que realizan conversiones son aquellos que devuelven transformado el valor que se les pasa como parámetro, normalmente un carácter, en otro carácter o en un valor de un tipo distinto.

Char toLowerCase (char c): si el carácter pasado es una letra, lo devuelve convertido a minúscula.

Char toUpperCase(char c): similar al anterior método, pero convierte el carácter, si es una letra, a mayúscula

Clase String

Las cadenas, conjuntos secuenciales de caracteres, se manipulan mediante la clase String, que funciona de forma dual.Por un lado, de manera

general, tiene un funcionamiento no estático; pero a su vez, dispone de algunos métodos que sí lo son.

Una variable de tipo String almacenará una cadena de caracteres, que provendrá e la manipulación de otra cadena o de un literal. Un literal cadena consiste en un texto entre comillas dobles ("). Es posible utilizar cualquier carácter, incluidos los codificados mediante Unicode y las secuencias de escape.

Inicialización de cadenas

```
String cad = "Mi perro \"Perico\" es de color blanco";

System.out.print (cad);
```

Comparación

Los operadores de comparación disponibles para números y caracteres

igual (==), menor que () y mayor que (>) no se encuentran disponibles directamente para comparar cadenas de caracteres, pero en su lugar disponemos de métodos de la clase String que realizan las comparaciones oportunas.

■ boolean equals(String otra): compara la cadena que invoca el método con otra.

El resultado de la comparación se indica devolviendo true o false, según sean iguales o distintas.

■ boolean equalsIgnoreCase (String otraCadena): funciona igual que equals() pero sin distinguir mayúsculas de minúsculas al realizar la comparación.

■ boolean regionMatches (int inicio, String otracad, int inicioOtra, int longitud): compara dos fragmentos de cadenas: el primero corresponde a la cadena invocante y comienza en el carácter con índice inicio; y el segundo corresponde a la cadena otraCad y comienza en el carácter con indice inicioOtra. Ambos fragmentos tendrán la longitud indicada. El método devuelve true o false para indicar si las regiones coinciden.

■ boolean regionMatches (boolean ignora, int ini, String otracad, int iniotra, int longitud): hace lo mismo que el método anterior con la

diferencia de que, si el valor del parámetro que ignora es true, la comprobación se realiza considerando iguales las mayúsculas y minúsculas.

Comparación alfabética

Otra forma de comparar dos cadenas es alfabéticamente, es decir, según el orden de un diccionario. Una cadena se considera alfabéticamente menor que otra si va antes en un diccionario

■ int compareTo(String cadena): compara alfabéticamente la cadena invocante y la que se pasa como parámetro, devolviendo un entero cuyo valor determina el orden de las cadenas de la forma:

- 0: si las cadenas comparadas son exactamente iguales.
- negativo: si la cadena invocante es menor alfabéticamente que la cadena pasada como parámetro, es decir, va antes por orden alfabético.
- positivo: si la cadena invocante es mayor alfabéticamente que la cadena pasada,

■ int compareToIgnoreCase (String cadena): realiza una comparación alfabética sin distinguir entre letras mayúsculas ni minúsculas.

Concatenación

El operador + sirve para unir o concatenar dos cadenas.

Obtención de caracteres

Todos los caracteres que forman una cadena pueden ser identificados mediante la posición que ocupan, al igual que los elementos de una tabla

Para conocer qué carácter se encuentra en una posición determinada de una cadena disponemos de:

■char charAt(int posicion): devuelve el carácter que ocupa el índice *posición* en la cadena que invoca el método.

Obtención de una subcadena

Una subcadena es un fragmento de una cadena, es decir, un subconjunto de caracteres contiguos de una cadena

Los métodos que llevan a cabo esto son:

- String substring(int inicio): devuelve la subcadena formada desde la posición inicio hasta el final de la cadena. Lo que se devuelve es una copia y la cadena invocante no se modifica.

- String substring (int inicio, int fin): hace lo mismo que la anterior, devolviendo la subcadena comprendida entre los índices inicio y el anterior a fin.

A veces una cadena leída del teclado o de algún fichero viene acompañada de una serie de espacios en blanco ('') y tabuladores ('\t', que representaremos |____|') al comienzo o al final de la cadena. Para eliminar estos caracteres blancos:

■ String strip(): devuelve una copia de la cadena eliminando los caracteres blancos del principio y del final. La cadena invocante no se modifica.

Tradicionalmente se ha usado el método trim () para esta operación, pero el resultado no es idéntico. Mientras strip () elimina los espacios blancos, trim() elimina todos los caracteres no imprimibles.

■ String stripLeading(): igual que strip () pero solo elimina los espacios en blanco del principio.

■ String stripTrailing (): solo elimina los espacios en blanco del final.

Longitud de una cadena

Como hemos visto, en ciertos métodos es necesario utilizar algunos índices para localizar los caracteres que forman una cadena. Para evitar el uso de un índice que se encuentre fuera de rango, existe:

■ int length(): devuelve el número de caracteres (longitud) de una cadena.

Búsqueda

Dentro de una cadena, entre los caracteres que la forman, es posible buscar un carácter o una subcadena. Disponemos de métodos que realizan la búsqueda de izquierda a derecha, o en sentido contrario a partir de una posición dada.

■ int indexOf (int c): busca la primera ocurrencia del carácter c en la cadena invocante empezando por el principio. Si lo encuentra, devuelve su índice, o -1 en caso contrario.

■ int indexOf (String cadena): sirve para buscar la primera ocurrencia de una cadena.

■int indexOf(int c, int inicio): busca la primera ocurrencia del carácter c, pero en lugar de comenzar a buscar en la posición 0, lo hace desde la

posición inicio en adelante. Devuelve el índice del elemento buscado si lo encuentra o -1 en caso contrario.

■ int indexOf (String cadena, int inicio): busca la primera ocurrencia de cadena a partir de la posición inicio.

■ int lastIndexOf (int c): devuelve el índice de la última ocurrencia de c, o -1 en el caso de que no se encuentre.

■ int lastIndexOf (String cadena): funciona igual que el anterior, pero buscando la última ocurrencia de cadena.

■ int lastIndexOf (int c, int inicio): la búsqueda se realiza desde el final al inicio de la cadena, comenzando en la posición inicio.

■ int lastIndexOf (String cadena, int inicio): devuelve la posición de cadena en la cadena invocante, comenzando en el índice inicio y buscando desde el final hacia el principio. En caso de no encontrar nada, devuelve -1.

Comprobaciones

Es posible realizar ciertas comprobaciones con una cadena de caracteres, como por ejemplo si está vacía, si contiene cierta subcadena, si comienza con un determinado prefijo o si termina con un sufijo dado, entre otras. Por regla general, los métodos que realizan estas comprobaciones devuelven un booleano que indica el éxito o el fracaso de la consulta.

Cadena vacía

Una cadena vacía es aquella que no está formada por ningún carácter, y se representa mediante “ ”(comillas dobles seguidas de otras comillas dobles). No contiene ningún carácter, es decir, su longitud es 0. Para asignar la cadena vacía a una variable,

String cad = “”;

El método para comprobar si una variable contiene la cadena vacía es:

- boolean isEmpty(): indica mediante un booleano true, si la cadena está vacía, o false en caso contrario.

Contiene

Si necesitamos comprobar si una cadena contiene otra subcadena,

■ boolean contains (CharSequence subcadena): devuelve true si en la cadena invocante se encuentra subcadena en cualquier posición.

Prefijos y sufijos

Los prefijos y sufijos no son más que subcadenas que van al principio o al final de una cadena respectivamente. Un ejemplo de prefijo en Java para la palabra programación es prog. En las cadenas de caracteres podemos comprobar si comienzan o terminan con un prefijo o sufijo dado. Para ello disponemos de los siguientes métodos:

■ boolean startsWith (String prefijo): comprueba si la cadena que invoca el método comienza con la cadena prefijo que se pasa como parámetro.

■ boolean startsWith (String prefijo, int inicio): hace lo mismo que el método anterior, comenzando la comprobación en la posición inicio.

■ boolean endsWith (String sufijo): indica si la cadena termina con el sufijo que le pasamos como parámetro.

Conversión Cadenas

Una cadena puede transformarse sustituyendo todas las letras que la componen a minúsculas o a mayúsculas, lo que resulta útil a la hora de procesar, por ejemplo, valores que provienen de un formulario y que cada usuario puede escribir de una forma u otra.

- String toLowerCase(): devuelve una copia de la cadena donde se han convertido todas las letras a minúsculas.
- String toUpperCase(): similar al método toLowerCase (), convierte todas las letras a mayúsculas.

El método replace () permite sustituir todas las ocurrencias de un carácter de una cadena por otro que se pasa como parámetro.

- String replace (char original, char otro): devuelve una copia de la cadena invocante donde se han sustituido todas las ocurrencias del carácter original por otro.

■String replace (CharSequence original, CharSequence otra): cambia todas las ocurrencias de la cadena original por la cadena otra.

Separación en partes

Una cadena se puede descomponer en partes si definimos un separador.

Por ejemplo, podemos descomponer la frase: «En un lugar de La Mancha», formada por seis palabras separadas por espacios, en una tabla de seis elementos de tipo String. Para ello utilizaremos el siguiente método:

■ String[] split (String separador): devuelve las subcadenas resultantes de dividir la cadena invocante con el separador pasado como parámetro. La subcadenas resultantes de la división se devuelven como una tabla de String.

Cadenas y tablas de caracteres

Existe una innegable relación entre las cadenas, clase String, y las tablas de caracteres, char [], hasta el punto de que en algunos lenguajes de programación no existe el tipo cadena, sino tablas de caracteres. En Java, ambas, cadenas y tablas de caracteres, pueden convertirse sin problema

unas en otras. En aquellas ocasiones en que interese manipular o cambiar de lugar los caracteres dentro de una cadena, resulta más cómodo trabajar con una tabla, cuyo acceso a los elementos es directo. Además, las cadenas en Java no se pueden modificar una vez creadas. Cuando modificamos una cadena lo que ocurre es que se crea una cadena nueva donde se incluyen las modificaciones. Afortunadamente, este proceso es transparente al programador.

El método que crea una tabla de caracteres tomando como base una cadena es:

■ char [] tocharArray (): crea y devuelve una tabla de caracteres con el contenido de la cadena desde la que se invoca, a razón de un carácter en cada elemento.

El método que realiza el proceso inverso, crear una cadena tomando como base una tabla de caracteres, es:

■ static String valueOf(char[] tabla) ;devuelve un String con el contenido de la tabla de caracteres.