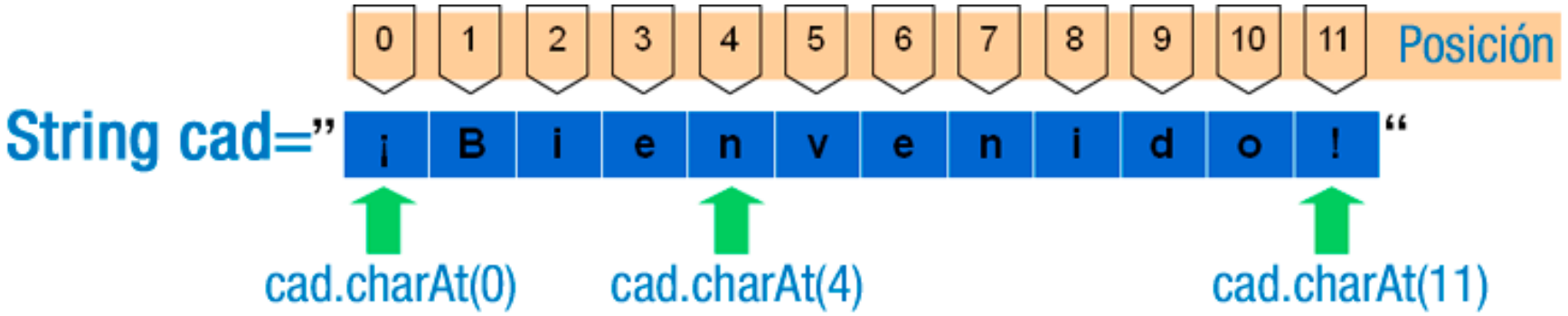
Unidad 4. Cadenas de caracteres y arrays



STRINGS EN JAVA

José L. Berenguel

Tabla de Contenidos

Cadenas de caracteres: String

- En Java, las cadenas de caracteres se emplean por medio de la clase ***String***.
- Los literales ***String*** se delimitan por comillas dobles.
- Podemos declarar variables ***String*** de la siguiente forma:

```
String nombre=”Jose Luis”; //declaración e inicialización implícita
String apellidos=new String(“García Sanz”); //inicialización explícita llamando al
constructor de la clase
```

- La clase ***String*** es **inmutable** lo que significa que cualquier modificación genera una nueva copia en memoria.

```
String nombre=”Jose Luis”; //declaración e inicialización implícita

//se crea un objeto String nuevo completamente cuya referencia se asigna a nombre.
//el anterior objeto String se destruirá por el recolector de basura
//si no hay más variables que lo referencien.
nombre = nombre + “ Berenguel”;
```

Operaciones con objetos String

- Concatenar dos cadenas: operador + y método ***concat()***
- Obtener el carácter en una posición: ***charAt()***
- Obtener una subcadena: ***substring()***
- Conversión de cadenas: ***valueOf()***
- Formato de cadenas: ***format()***
- Buscar una subcadena o un carácter: ***contains()***, ***indexOf()***
- Comprobar si empieza o termina con una secuencia de caracteres: ***startsWith()***, ***endsWith()***
- Reemplazar parte de una cadena por otra: ***replace()***, ***replaceAll()***
- Comparar cadenas: ***equals()***, ***equalsIgnoreCase()***, ***compareTo()***, ***compareToIgnoreCase()***
- Cambiar a mayúsculas/minúsculas: **toUpperCase()**, **LowerCase()**
- Elimina los espacios en blanco al inicio y final: ***trim()***

La clase StringBuilder

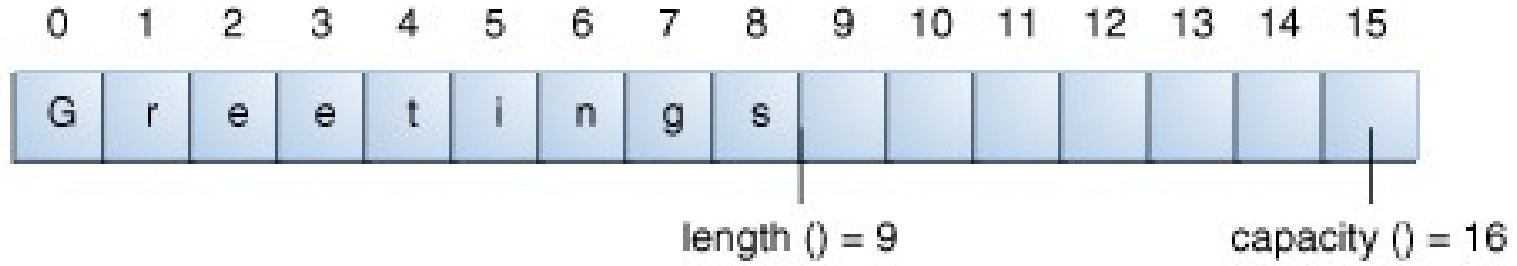
- La clase ***StringBuilder*** permite modificar la cadena de caracteres sin generar una nueva copia (es un objeto mutable) lo que optimiza el uso de la memoria.
- Para aplicaciones multihilo: ***StringBuffer***.
- No se permite la concatenación con + ni la declaración mediante literales con “ “, se debe usar el constructor y los métodos apropiados de la clase.
- ***StringBuilder*** contiene métodos básicos iguales a String.

La clase StringBuilder

- El constructor por defecto ***StringBuilder()*** crea un objeto con una cadena vacía y una capacidad de 16 elementos.
- El constructor ***StringBuilder(String s)*** crea un objeto cuya cadena se inicializa al valor del parámetro y con una capacidad extra de 16 elementos.

```
StringBuilder hello=new StringBuilder(); //StringBuilder vacío con capacidad 16

//añadimos 9 caracteres al inicio.
hello.append("Greetings");
```



Expresiones regulares

- Permite comprobar si **una cadena de caracteres cumple un patrón** establecido.
- El patrón se diseña a través de símbolos y caracteres especiales. Ejemplo número binario: **“[01]+”**
- **Reglas generales** para construir una expresión regular:
  - Conjunto de símbolos fijo. Se indican dichos símbolos en el patrón. Ejemplo 3 aes: **“aaa”**.
  - Opcionalidad de simbolos. Con los corchetes se Indica que puede aparece solo uno de los símbolos. Ejemplo: **[“abc”]**, **“aa[xy]aa”**.
  - Con el guion se indica cualquier carácter entre la letra inicial y final. Ejemplo: **“[a-z]” “[A-Z]” “[a-zA-Z]” “[0-9]”**.

Expresiones regulares

- Las **repeticiones del patrón** se indican con operadores de cuantificación:
  - Interrogación (**?**). Un patrón aparece 1 o ninguna vez: **“a?”**, .
  - Asterisco (*****). Un patrón aparece 0 o más veces. Ejemplo: **“a*”**,
  - Suma (**+**). Un patrón debe aparecer al menos una vez. Ejemplo: **“a+”**.
  - Llaves **{min,max}**. Podemos indicar el número mínimo y máximo de veces que un patrón puede aparecer. Ejemplo: **“a{1,4}”**, **a{2,}**, **a{5}**.
  - Barra (|). Una de entre varias opciones. Ejemplo: **“a|e|i|o|u”**, **“este|oeste|norte|sur”**.
  - Paréntesis **( )**. Permiten crear **grupos de patrones** sobre los que aplicar los operadores de cuantificación. Ejemplo: **“(#[01]){2,3}”**.

Expresiones regulares

- En Java las expresiones regulares se usan con las clases ***Pattern*** y ***Matcher*** del paquete ***java.util.regex***.
  - ***Pattern***. Se utiliza para definir y procesar el patrón o expresión regular mediante los símbolos apropiados. Verifica que el patrón está bien construido.
  - ***Matcher***. Comprueba si una cadena sigue un patrón definido.

```
Pattern patronBinario = Pattern.compile(“[01]+”);
Matcher m = patronBinario.matcher(“00001010”);

if(m.matches()){
    System.out.println(“Sí, es un número binario”);
}else{
    System.out.println(“No, no es un número binario”);
```

Expresiones regulares

- La clase ***Matcher*** contiene el resultado del patrón y dispone de varios métodos para analizar la forma en la que la cadena cumple el patrón.
  - ***matches()***. Devuelve *true* si toda la cadena cumple el patrón.
  - ***lookingAt()***. Devuelve *true* si el patrón ha encajado al comienzo de la cadena (esta puede contener caracteres adicionales).
  - ***find()***. Devuelve *true* si encuentra una coincidencia del patrón en algún lugar de la cadena (***start()*** y ***end()*** para saber la posición inicial y final de la coincidencia). Se puede volver a llamar a ***find()*** para encontrar la coincidencia siguiente.
  - ***reset()***. Reinicia el método ***find()*** para volver a la primera coincidencia.

Expresiones regulares

- La clase ***Matcher*** permite acceder a los grupos de patrones indicados con los paréntesis con el método ***group()***.
- El grupo 0 hace referencia a toda la cadena por lo que el primer grupo individual será el 1.

```
//Patrón con tres grupos: letra inicial, dígitos, letra final
Pattern patronDNINIE = Pattern.compile("([XY]?)([0-9]{1,9})([A-Za-z])");

Matcher m = patronDNINIE.matcher(“X123456789Z Y00110011M 999999T”);

while(m.find()){
     System.out.println(“Letra inicial (opcional):” + m.group(1));
     System.out.println("Número:" + m.group(2));
     System.out.println("Letra NIF:" + m.group(3));
}
```

Expresiones regulares

- **Otros símbolos** para construir expresiones regulares más **complejas**:
  - Negación (**^**).
    - Justo después de un [significa que se admitirá cualquier símbolo distinto a los indicados. Ejemplo: **“[^abc]”**.
    - Al principio de la expresión regular indica comienzo de línea o entrada. Se combina con **$** para indicar fin de línea o de entrada (**modo multilínea**). Ejemplo: **“^[01]+$”**.
  - Punto (**.**). Simboliza cualquier carácter.
  - **“\\d”**. Un dígito numérico. Equivale a **“[0-9]”**.
  - **“\\D”**. Cualquier carácter excepto un dígito numérico. Equivale a **“[^0-9]”**.
  - **“\\s”**. Un espacio en blanco (incluye tabulaciones, saltos de línea y otras formas de espacio).
  - **“\\S”**. Cualquier cosa excepto un espacio en blanco.
  - **“\\w”**. Cualquier carácter que podrías encontrar en una palabra.

Expresiones regulares

- Secuencias de escape. En caso de que sea necesario utilizar un carácter con significado (llave, corchete, paréntesis, …) en el patrón se debe escapar.
- Se antepone **** al símbolo que queramos escapar. Ejemplo: **“\\(“ “\\[“ “\\]”**.
- A excepción de las comillas que se escapan con una barra \. Ejemplo: **“\””**.

Unidad 4. Cadenas de caracteres y arrays

DUDAS Y PREGUNTAS