Strings (cadenas de caracteres)

Algoritmos y Programación III - UNPAZ





El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.



El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.

Ejemplos de strings:

• "pepe"



El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.

- "pepe"
- "Hola, mundo!"



El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.

- "pepe"
- "Hola, mundo!"
- "253" (notar que no es un int)



El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.

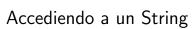
- "pepe"
- "Hola, mundo!"
- "253" (notar que no es un int)
- "2+2" (notar que no es el **int** 4!)





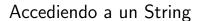
El tipo de datos **String** representa las secuencias de caracteres, junto con algunas operaciones sencillas para trabajar con estas secuencias.

- "pepe"
- "Hola, mundo!"
- "253" (notar que no es un int)
- "2+2" (notar que no es el **int** 4!)
- "" (string vacío)



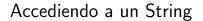


• El método **charAt** permite acceder a los caracteres que conforman el **String**.



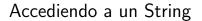


- El método charAt permite acceder a los caracteres que conforman el String.
- Recibe como parámetro un int que indica la posición, y retorna un char con el caracter ubicado en la posición indicada.





- El método charAt permite acceder a los caracteres que conforman el String.
- Recibe como parámetro un int que indica la posición, y retorna un char con el caracter ubicado en la posición indicada.
- La primera posición del **String** es la posición cero.





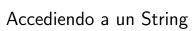
- El método charAt permite acceder a los caracteres que conforman el String.
- Recibe como parámetro un int que indica la posición, y retorna un char con el caracter ubicado en la posición indicada.
- La primera posición del **String** es la posición cero.

Ejemplo:

```
String fruta = "banana";

char letra = fruta.charAt(1);

System.out.println(letra);
```





 Si el parámetro que se pasa a charAt no corresponde a un índice (posición) válido del String, se genera una excepción (error en tiempo de ejecución).

Accediendo a un String



- Si el parámetro que se pasa a charAt no corresponde a un índice (posición) válido del String, se genera una excepción (error en tiempo de ejecución).
- Ejemplo:

```
String fruta = "banana";
char letra = fruta.charAt(11);
```

Accediendo a un String



- Si el parámetro que se pasa a charAt no corresponde a un índice (posición) válido del String, se genera una excepción (error en tiempo de ejecución).
- Ejemplo:

```
String fruta = "banana";

char letra = fruta.charAt(11);
```

Este código genera la siguiente excepción:
 Exception in thread "main" java.lang.StringIndexOutOfBoundsException:
 String index out of range: 11



La longitud de un String

 La función length permite consultar la longitud de un String, y retorna un int como resultado.

La longitud de un String



 La función length permite consultar la longitud de un String, y retorna un int como resultado.

```
Ejemplo:
```

```
String fruta = "banana";
System.out.println(fruta.length());
```

• ¿Qué muestra por consola este código?

Sintaxis



 Observar la sintaxis que utilizamos para acceder a los métodos de la clase String:

```
unString.charAt(3);
unString.length();
```

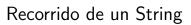
Sintaxis



 Observar la sintaxis que utilizamos para acceder a los métodos de la clase String:

```
unString.charAt(3);
unString.length();
```

• Estas funciones son métodos de la clase **String**, que se ejecutan sobre una instancia de la clase.





 Es habitual recorrer un String de izquierda a derecha (secuencialmente), accediendo a cada char de la secuencia:

```
for(int i=0; i<fruta.length(); i=i+1)
{
    System.out.println( fruta.charAt(i) );
}</pre>
```

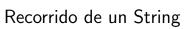




• El código anterior es equivalente a ...

```
int i = 0;
while( i < fruta.length() )

{
    System.out.println( fruta.charAt(i) );
    i = i+1;
}</pre>
```





• Los elementos del tipo de datos **char** se notan entre apóstrofes simples.

Recorrido de un String



- Los elementos del tipo de datos char se notan entre apóstrofes simples.
- Ejemplo:

```
int cont=0;
for(int i=0; i<fruta.length(); ++i)

fif( fruta.charAt(i) == 'a' )
++cont;
}
</pre>
```

¿Qué hace este código?



 Dado un char, el método indexOf encuentra el índice donde aparece ese caracter en el String por primera vez.



 Dado un char, el método indexOf encuentra el índice donde aparece ese caracter en el String por primera vez.

Ejemplo:

```
String fruta = "banana";
int indice = fruta.indexOf('a');
```



 Dado un char, el método indexOf encuentra el índice donde aparece ese caracter en el String por primera vez.

```
Ejemplo:
```

```
String fruta = "banana";

int indice = fruta.indexOf('a');
```

En cierto sentido, es el método opuesto de charAt.



 Dado un char, el método indexOf encuentra el índice donde aparece ese caracter en el String por primera vez.

```
Ejemplo:
```

```
String fruta = "banana";
int indice = fruta.indexOf('a');
```

- En cierto sentido, es el método opuesto de charAt.
- Si el **char** que se pasa como parámetro no está en el **String**, entonces **indexOf** retorna -1 como resultado.



 Una segunda versión de indexOf toma como segundo parámetro un int que indica desde qué índice de la cadena se debe comenzar la búsqueda.

int indice = fruta.indexOf($^{1}a^{1}$, 2);



Los Strings son inmutables

 Algo muy particular con respecto a los Strings es que "no se los puede modificar". Es decir, no es posible alterar su contenido.



Los Strings son inmutables

- Algo muy particular con respecto a los Strings es que "no se los puede modificar". Es decir, no es posible alterar su contenido.
- Obviamente, si quiero modificar el string que está guardado en una variable, siempre puedo cambiarlo por otro:

```
String fruta = "anana";

fruta = "b" + fruta;

fruta = fruta + "s";

String otro = fruta.toUpperCase();
```

 Los métodos toUpperCase y toLowerCase convierten un String a mayúsculas y minúsculas, respectivamente.



Los Strings son inmutables

- Algo muy particular con respecto a los Strings es que "no se los puede modificar". Es decir, no es posible alterar su contenido.
- Obviamente, si quiero modificar el string que está guardado en una variable, siempre puedo cambiarlo por otro:

```
String fruta = "anana";

fruta = "b" + fruta;

fruta = fruta + "s";

String otro = fruta.toUpperCase();
```

- Los métodos toUpperCase y toLowerCase convierten un String a mayúsculas y minúsculas, respectivamente.
- Suelen generar confusión, porque pareciera que modifican (mutan) el **String** sobre el que actúan, pero esto no es así.



• Para ver si dos **String**s son iguales, se utiliza el método **equals**.

```
String nombre1 = "Alan Turing";
String nombre2 = "Ada Lovelace";

if (nombre1.equals (nombre2))
System.out.println ("Los nombres son iguales.");
```



• No se debe utilizar el operador de comparación == en este caso!



 No se debe utilizar el operador de comparación == en este caso! (por qué no?)



- No se debe utilizar el operador de comparación == en este caso! (por qué no?)
- Las variables nombre1 y nombre2 contienen la posición de memoria de los **String**s involucrados, y no las secuencias de caracteres en sí mismas.



- No se debe utilizar el operador de comparación == en este caso! (por qué no?)
- Las variables nombre1 y nombre2 contienen la posición de memoria de los **String**s involucrados, y no las secuencias de caracteres en sí mismas.
- Entonces, al comparar nombre1 == nombre2, estamos determinando si están ubicadas en la misma posición de memoria, y no su contenido. Podría haber dos cadenas iguales ubicadas en diferentes posiciones de memoria. En ese caso serían iguales pero la comparación == nos daría falso.