

Strings (cadenas de caracteres)

Algoritmos y Programación III - UNPAZ

Strings

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

Strings

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

Ejemplos de strings:

- “pepe”

Strings

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

Ejemplos de strings:

- “pepe”
- “Hola, mundo!”

Strings

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

Ejemplos de strings:

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

Strings

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

Ejemplos de strings:

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

Strings

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

Ejemplos de strings:

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

Accediendo a un String

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

Accediendo a un 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.

Accediendo a un 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.
- La primera posición del **String** es la posición **cero**.

Accediendo a un 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.
- La primera posición del **String** es la posición **cero**.
- Ejemplo:

```
1 String fruta = "banana";  
2 char letra = fruta.charAt(1);  
3 System.out.println(letra);
```

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).

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:

```
1 String fruta = "banana";  
2 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:

```
1   String fruta = "banana";  
2   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:

```
1 String fruta = "banana";  
2 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**:

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

Sintaxis

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

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

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

Recorrido de un String

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

```
1  for(int i=0; i<fruta.length(); i=i+1)
2  {
3      System.out.println( fruta.charAt(i) );
4  }
```

Recorrido de un String

- El código anterior es equivalente a ...

```
1  int i = 0;
2  while( i < fruta.length() )
3  {
4      System.out.println( fruta.charAt(i) );
5      i=i+1;
6  }
```

Recorrido de un String

- 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:

```
1  int cont=0;
2  for(int i=0; i<fruta.length(); ++i)
3  {
4      if( fruta.charAt(i) == 'a' )
5          ++cont;
6  }
```

- ¿Qué hace este código?

El método indexOf

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

El método indexOf

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

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

El método indexOf

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

- Ejemplo:

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

- En cierto sentido, es el método opuesto de **charAt**.

El método indexOf

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

```
1   String fruta = "banana";  
2   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.

El método indexOf

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

```
1  int indice = fruta.indexOf('a', 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:

```
1 String fruta = "anana";  
2 fruta = "b" + fruta;  
3 fruta = fruta + "s";  
4 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:

```
1 String fruta = "anana";  
2 fruta = "b" + fruta;  
3 fruta = fruta + "s";  
4 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.

Los Strings no se comparan así nomás

- Para ver si dos **Strings** son iguales, se utiliza el método **equals**.

```
1 String nombre1 = "Alan Turing";  
2 String nombre2 = "Ada Lovelace";  
3  
4 if (nombre1.equals (nombre2))  
5     System.out.println ("Los nombres son iguales.");
```

Los Strings no se comparan así nomás

- **No** se debe utilizar el operador de comparación `==` en este caso!

Los Strings no se comparan así nomás

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

Los Strings no se comparan así nomás

- **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 **Strings** involucrados, y no las secuencias de caracteres en sí mismas.

Los Strings no se comparan así nomás

- **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 **Strings** 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.