

Strings

Creación, concatenación, métodos y ejemplos

Cátedra 10

Otoño 2005

La clase String permite a Java trabajar con textos y manipularlos fácilmente.

1 Lectura/escritura de strings

Este programa lee una palabra, la guarda en una variable de la clase String y luego la escribe seguida de un mensaje.

```
String nombre;
con.print ("Cómo te llamas:");
nombre=con.readLine();
con.print (nombre+" es un nombre hermoso");
```

Los objetos de la clase String sirven para almacenar y manipular cadenas (secuencias) de caracteres. Un String puede contener letras (mayúsculas, minúsculas), cifras, símbolos (! " \$ % ...) y espacios. Ejemplos de objetos String son:

"Ser un romántico viajero", "Cueeeek", "edad =" y "188"

Java hace una distinción entre el String "188" y el int 188. Reconoce el primero como una "palabra" y el segundo como un número.

Por ejemplo, son incorrectos:

```
int x="4944"; // debería ser int x=4944;
String k=47; // debería ser String k="47";
```

Al igual que los tipos primitivos (int y double), los String pueden ser inicializados al momento de ser declarados:

```
int edad=42;
String nombre="Mary Jane";
```

Esta última instrucción denota la creación de un objeto String que contiene el nombre Mary Jane. Esta es la única clase de Java que permite crear objetos sin usar new. También pueden leerse desde el teclado

```
String dirección;
con.print ("Ingresa dónde vives: ");
dirección=con.readLine();
```

2 Algunos métodos útiles

2.1 length()

El largo de un String se puede obtener mediante su método length:

```
String animal="Perro";
int largo=animal.length();
```

asigna el valor 5 a largo. Es importante notar que todos los caracteres se cuentan para calcular el largo, incluyendo los espacios. El String vacío ("") tiene largo 0.

2.2 compareTo

El orden en que aparecen las palabras en el diccionario se llama "orden lexicográfico". Por ejemplo, "Amelia" es lexicográficamente menor que "Bernardo" y este que "Carlos". En caso de dos palabras que empiecen con la misma letra, Java compara la segunda letra (carácter) para ver qué palabra es menor. Si ambas letras son iguales, sigue con la tercera y así sucesivamente. Para Java las minúsculas son "mayores" que la mayúsculas. De hecho, el orden de los caracteres es

espacio<...<0<1<2...9<...<A<B<C<...<Y<Z<...<a<b<...<z<...

Así se tiene que

Asno<Burro, asno>Burro, asno>asna, U>Colo-Colo, lata<latas, etc.

Java permite comparar dos strings mediante el método compareTo. Este recibe un String como parámetro y devuelve un valor entero:

```
int menor=nombre1.compareTo(nombre2);
```

El valor que devuelve compareTo es menor que 0 si nombre1 es menor que nombre2, 0 si son iguales y mayor que cero en caso contrario. También se puede ver directamente si dos Strings son iguales con la función equals:

```
if (nombre.equals("Pedro")) ...
```

2.3 substring

El método substring recibe dos parámetros (i, j) y devuelve un nuevo string formado por los caracteres i a j-1. El índice i va de 0 al largo del String menos uno. Por ejemplo,

```
String nombre="Catarata";
String sub=nombre.substring(2,6); // en sub queda "tara"
```

2.4 indexOf

El método indexOf devuelve a partir de qué letra de un string aparece un segundo string. Por ejemplo,

```
String nombre="Juan Enrique Pardo";
int j=nombre.indexOf ("Enrique");
// j vale 5 (posición de la E de Enrique)
con.print (nombre.indexOf("ua"));
// escribe 1
int p="gusano".indexOf("sana");
// asigna -1 a p, pues sana no aparece en gusano
```

Existe una versión de indexOf, que recibe además un entero. La búsqueda del segundo string se comienza a realizar a partir de la letra cuyo indice esta dado por el entero. Por ejemplo, si el string s es Ananas, s.indexOf("an") es 0, s.indexOf("an",1) es 2, s.indexOf("an",2) es 2 y s.indexOf(s2,s.indexOf(s2)+1); devuelve la segunda ocurrencia de s2 en s.

3 Concatenación

Los strings se pueden concatenar (pegar) con el operador +. Nuevamente es importante recalcar que los strings son la única clase que soportan un operador. No sumes tortugas, fracciones ni consolas con +. Por ejemplo,

```
"hola" + "juan" es "holajuan"
"hola" + " " + "juan" es ("hola" + " ") + "juan" u "hola juan"
```

Además, Java permite la siguiente expresión:

```
String nombre="Lucifer "+666; // en nombre queda "Lucifer 666"
```

En este caso Java transforma el 666 a un String y luego lo concatena. Esto permite convertir fácilmente un número en un String:

```
int x=4368;
String dirección="" + x;
```

Por ejemplo, considera el siguiente programa que dice cuantas cifras tiene un número:

```
con.print ("Ingresa un número");
int x=con.readInt();
String s="" + x;
con.print("El número "+x+" tiene "+s.length()+" cifras");
```

Debes ser cuidados al usar el operador +. A veces significa suma (de enteros o de reales) y a veces concatenación de strings.

4 Conversión

Acabamos de ver cómo convertir un entero o double en String. El siguiente código convierte un string en double:

```
String spi="3"+" .1415926";
double pi=new Double(spi).doubleValue();
```

Si se desea convertir el string en entero:

```
con.print ("Ingresa un número entero o fin para terminar");
String entrada=con.readLine();
int num;
if (!entrada.equals("fin"))
    num=Integer.parseInt(entrada);
```

AL MARGEN

char

Existe un tipo similar a int llamado char. Un char es un tipo entero que va de -32000 hasta 32000 (16 bits). Se pueden usar de la misma forma que los ints, pero son muy útiles para trabajar con caracteres. Esto se debe a que muchos métodos están sobrecargados de forma que cuando reciben un char c, lo interpretan como la letra (caracter) cuyo código es c.

Por ejemplo,

```
char c=65; // 65 es el código de la A
con.print(c); // escribe 'A'
```

Otra forma de escribir esto es

```
char c='A'; // guardo en c el código de A
con.print(c);
```

Si el string s contiene **calculador**, al concatenarle el char c, se obtiene **calculadorA**, pues el operador + interpreta el char de la misma forma que con.print. El método charAt(i) de la clase String permite obtener el carácter i-ésimo de un string