La clase String

Hemos aprendido a diferenciar entre clase y objetos, a acceder desde un objeto a los miembros datos y a las funciones miembro. Vamos a utilizar clases importantes en el lenguaje Java y a crear objetos de dichas clases. Empezaremos por la clase *String* una de las más importantes del lenguaje Java. Más adelante, volveremos a estudiar otros ejemplos para que el lector se acostumbre a crear sus propias clases.

La clase *String*

Dentro de un objeto de la clases *String* o *StringBuffer*, Java crea un array de caracteres de una forma similar a como lo hace el lenguaje C++. A este array se accede a través de las funciones miembro de la clase.

Los strings u objetos de la clase *String* se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

Java crea un objeto de la clase *String* automáticamente.

Para crear un string explícitamente escribimos

```
String str=new String("El primer programa");
```

También se puede escribir, alternativamente

```
String str="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

```
String str="";
String str=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase *String*. Sin embargo,

```
String str;
```

está declarando un objeto *str* de la clase *String*, pero aún no se ha creado ningún objeto de esta clase.

Cómo se obtiene información acerca del string

Una vez creado un objeto de la clase *String* podemos obtener información relevante acerca del objeto a través de las funciones miembro.

Para obtener la longitud, número de caracteres que guarda un string se llama a la función miembro *length*.

```
String str="El primer programa";
int longitud=str.length();
```

Podemos conocer si un string comienza con un determinado prefijo, llamando al método *startsWith*, que devuelve **true** o **false**, según que el string comience o no por dicho prefijo

```
String str="El primer programa";
boolean resultado=str.startsWith("El");
```

En este ejemplo la variable resultado tomará el valor **true**.

De modo similar, podemos saber si un string finaliza con un conjunto dado de caracteres, mediante la función miembro *endsWith*.

```
String str="El primer programa";
boolean resultado=str.endsWith("programa");
```

Si se quiere obtener la posición de la primera ocurrencia de la letra p, se usa la función *indexOf*.

```
String str="El primer programa";
int pos=str.indexOf('p');
```

Para obtener las sucesivas posiciones de la letra p, se llama a otra versión de la misma función

```
pos=str.indexOf('p', pos+1);
```

El segundo argumento le dice a la función *indexOf* que empiece a buscar la primera ocurrencia de la letra p a partir de la posición *pos+1*.

Otra versión de *indexOf* busca la primera ocurrencia de un substring dentro del string.

```
String str="El primer programa";
int pos=str.indexOf("pro");
```

Vemos que una clase puede definir varias funciones miembro con el mismo nombre pero que tienen distinto número de parámetros o de distinto tipo.

Comparación de strings

La comparación de strings nos da la oportunidad de distinguir entre el operador lógico **==** y la función miembro *equals* de la clase *String*. En el siguiente código

```
String str1="El lenguaje Java";
String str2=new String("El lenguaje Java");
if(str1==str2){
        System.out.println("Los mismos objetos");
}else{
        System.out.println("Distintos objetos");
}
if(str1.equals(str2)){
        System.out.println("El mismo contenido");
}else{
        System.out.println("Distinto contenido");
}
```

Esta porción de código devolverá que *str1* y *str2* son distintos objetos pero con el mismo contenido. *str1* y *str2* ocupan posiciones distintas en memoria pero guardan los mismos datos.

Cambiemos la segunda sentencia y escribamos

```
String str1="El lenguaje Java";
String str2=str1;
System.out.prinln("Son el mimso objeto "+(str1==str2);
```

Los objetos *str1* y *str2* guardan la misma referencia al objeto de la clase *String* creado. La expresión *(str1==str2*) devolverá **true**.

Asi pues, el método *equals* compara un string con un objeto cualquiera que puede ser otro string, y devuelve **true** cuando dos strings son iguales o **false** si son distintos.

```
String str="El lenguaje Java";
boolean resultado=str.equals("El lenguaje Java");
```

La variable *resultado* tomará el valor **true.**

La función miembro *comapareTo* devuelve un entero menor que cero si el objeto string es menor (en orden alfabético) que el string dado, cero si son iguales, y mayor que cero si el objeto string es mayor que el string dado.

```
String str="Tomás";
int resultado=str.compareTo("Alberto");
```

La variable entera *resultado* tomará un valor mayor que cero, ya que Tomás está después de Alberto en orden alfabético.

```
String str="Alberto";
int resultado=str.compareTo("Tomás");
```

La variable entera *resultado* tomará un valor menor que cero, ya que Alberto está antes que Tomás en orden alfabético.

Extraer un substring de un string

En muchas ocasiones es necesario extraer una porción o substring de un string dado. Para este propósito hay una función miembro de la clase *String* denominada *substring*.

Para estraer un substring desde una posición determinada hasta el final del string escribimos

```
String str="El lenguaje Java";
String subStr=str.substring(12);
```

Se obtendrá el substring "Java".

Una segunda versión de la función miembro *substring*, nos permite extraer un substring especificando la posición de comienzo y la el final.

```
String str="El lenguaje Java";
String subStr=str.substring(3, 11);
```

Se obtendrá el substring "lenguaje". Recuérdese, que las posiciones se empiezan a contar desde cero.

Convertir un número a string

Para convertir un número en string se emplea la función miembro estática *valueOf* (más adelante explicaremos este tipo de funciones).

```
int valor=10;
String str=String.valueOf(valor);
```

La clase *String* proporciona versiones de *valueOf* para convertir los datos primitivos: **int**, **long**, **float**, **double**.

Esta función se emplea mucho cuando programamos applets, por ejemplo, cuando queremos mostrar el resultado de un cálculo en el área de trabajo de la ventana o en un control de edición.

Convertir un string en número

Cuando introducimos caracteres en un control de edición a veces es inevitable que aparezcan espacios ya sea al comienzo o al final. Para eliminar estos espacios tenemos la función miembro *trim*

```
String str="  12 ";
String str1=str.trim();
```

Para convertir un string en número entero, primero quitamos los espacios en blanco al principio y al final y luego, llamamos a la función miembro estática *parseInt* de la clase *Integer* (clase envolvente que describe los números enteros)

```
String str="  12 ";
int numero=Integer.parseInt(str.trim());
```

Para convertir un string en número decimal (**double**) se requieren dos pasos: convertir el string en un objeto de la clase envolvente *Double*, mediante la función miembro estática *valueOf*, y a continuación convertir el objeto de la clase *Double* en un tipo primitivo **double** mediante la función *doubleValue*

```
String str="12.35 ";
double numero=Double.valueOf(str).doubleValue();
```

Se puede hacer el mismo procedimiento para convertir un string a número entero

```
String str="12";
int numero=Integer.valueOf(str).intValue();
```

Métodos Principales

Para poder aplicar estos métodos es necesario crear un objeto String. Además de estos métodos, la clase String cuenta con otros muchos. Consultar la API para más información.

- **int length():** devuelve la longitud de la String, incluyendo espacios en blanco. La longitud siempre es una unidad mayor que el índice asociado al último carácter de la String.

  Ejemplo:

  ```
  String s="cucu";
  int longitud=s.length();
  System.out.println(longitud);
  ```

  Por consola:

  4

- **int indexOf(String str, int indice):** devuelve el índice en el que aparece por primera vez la String del primer argumento en la que se aplica el método, a partir del índice especificado en el segundo argumento. Recordar que una String está indexada. Si el índice a partir del que se inicia la búsqueda no existe o la String no aparece, devuelve –1. **MUY USADO.**

  Ejemplo:

  ```
  String str="expedicion";
  System.out.println(str.indexOf("x",str.length()));
  ```

  Por consola:

  –1 porque la búsqueda se inicia a partir de un índice que no existe ya que el índice mayor es la longitud de la String –1.

- **int indexOf(char ch):** devuelve el índice en el que aparece por primera vez el carácter que se le pasa al argumento. Si no se encuentra el carácter devuelve –1. Se observa que el nombre de este

método es igual al anterior aunque su número de argumentos es distinto además de su tipo. A esto, en Java, se le llama **sobrecarga de métodos: mismo nombre pero distinto nº de argumentos o distinto tipo de argumentos o distinto orden**. Ir a la API para comprobar que hay más con este mismo nombre. Este concepto se tratará más en profundidad en temas posteriores.

- **String replace (char viejoChar, char nuevoChar):** cambia el carácter asociado al primer argumento por el que se le pasa al segundo, de la String sobre la que se aplica el método generando una nueva. La String sobre la que se aplica el método no cambia, simplemente se crea otra nueva en base a la String sobre la que se aplica el método.

  Ejemplo:

```
String s="cucu";
String s1=s.replace('u','o');
System.out.println(s);
System.out.println(s1);
```

  Por consola:

  cucu

  coco

- **String toLowerCase():** devuelve una nueva String convirtiendo todos los caracteres de la String sobre la que se aplica el método, en minúsculas.
- **String toUpperCase():** devuelve una nueva String convirtiendo todos los caracteres de la String sobre la que se aplica el método, en mayúsculas.
- **boolean equals(String str):** investiga si dos String tienen los mismos caracteres y en el mismo orden. Si es así devuelve true y si no false. **MUY USADO**
- **boolean equalsIgnoreCase(String str):** investiga si dos String tienen los mismos caracteres y en el mismo orden sin tener en cuenta las mayúsculas. Si es así devuelve true y si no false. **MUY USADO**
- **boolean startsWith(String str):** devuelve true si la String sobre la que se aplica comienza por la del argumento; false si esto no ocurre.
- **boolean startsWith(String str, int indice):** devuelve true si la String sobre la que se aplica comienza por la del argumento a partir de un determinado índice asociado al segundo argumento; false si esto no ocurre.
- **boolean endsWith(String str):** devuelve true si la String sobre la que se aplica acaba en la del argumento; false si esto no ocurre.
- **String trim():** devuelve una String en base a la que se le pasa al argumento, pero sin espacios en blanco al principio ni al final. No elimina los espacios en blanco situados entre las palabras.

  Ejemplo:

```
String str=" hola que tal ";
System.out.println(str.length());
String strBis=str.trim();
System.out.println(strBis.length());
```

Por consola:

14

12

- **String substring(int indiceIni, int indiceFin):** devuelve una String obtenida a partir del índice inicial incluido y del índice final excluido; es decir, se comporta como un intervalo semiabierto [indiceIni, indiceFin). Si el índice final sobrepasa la longitud de la String, lanza una IndexOutOfBoundsException. **MUY USADO.**

  Ejemplo:

```
String str="cucurrucucupaloma";
System.out.println(str.substring(4,9));
```

  Por consola:

  rrucu

- **char charAt (int indice):** devuelve el carácter asociado al índice que se le pasa como argumento de la String sobre la que se aplica el método. Si el índice no existe se lanza una StringIndexOutOfBoundsException que hereda de IndexOutOfBoundsException. **MUY USADO.**