APRENDERAPROGRAMAR.COM

¿QUÉ ES UNA INTERFAZ DE CLASE JAVA? CONCEPTO. SIGNATURA DE MÉTODOS Y CONSTRUCTORES. EJEMPLO CLASE STRING. (CU00648B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

Resumen: Entrega nº48 curso Aprender programación Java desde cero.

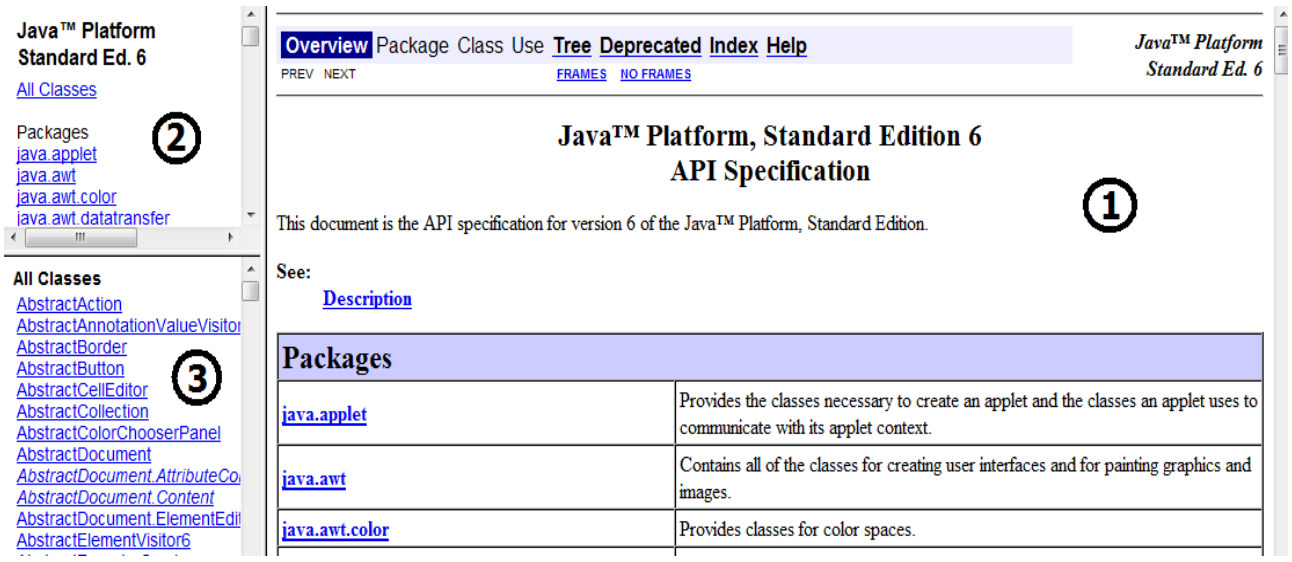
Autor: Alex Rodríguez

¿QUÉ ES UNA INTERFACE DE CLASE JAVA? CONCEPTO. EJEMPLO CON LA CLASE STRING.

Supongamos que queremos determinar si una cadena (String) comienza de una manera determinada, por ejemplo si empieza por 'caza'. Esta condición la cumplirían cadenas como "cazador", "cazaculebras" y "caza prohibida". Nosotros podríamos desarrollar un método que nos permita realizar esta determinación usando código propio.

Pero dado que queremos actuar sobre un objeto del API de Java (un String) y dado que queremos hacer algo que con toda seguridad es un problema que se le presenta con frecuencia a muchos programadores, lo lógico es consultar la documentación de la clase. Posiblemente ese método se encuentre disponible dentro de los métodos de la clase en el API de Java y nos podamos ahorrar tiempo y código si lo utilizamos.

Si en un buscador de internet introducimos el texto "api java **X**" donde X es la versión de java que estemos usando (por ejemplo "api java 9"), podemos acceder a una vista resumen de los paquetes y clases del API de Java. Otra forma de acceder es a través de BlueJ. Vamos al menú Help → Java Class Libraries (biblioteca de clases Java), y se nos abre el navegador en la página del API de Java de la versión de Java que esté usando BlueJ.



La información se suele distribuir en marcos. En uno de ellos (1) el detalle de la clase o paquete que tengamos seleccionado. En otro marco (2), el listado de paquetes (librerías). Y en otro marco (3), el listado de clases ("All Classes", todas las clases disponibles). Para buscar la documentación de una clase buscamos la clase en el listado de clases. Vamos a buscar la clase String y pulsamos sobre ella.

En la ventana de detalle veremos que la clase String tiene una extensa documentación. Prueba a buscar la clase ArrayList y échale un vistazo a su documentación. Verás que la documentación es un poco menos extensa, pero la estructura de la documentación es similar para todas las clases y suele comprender:

1.- El nombre de la clase y una descripción general.

2.- Lista breve de campos (atributos) de la clase (Fields).

3.- Lista breve de constructores de la clase.

4.- Lista breve de métodos de la clase.

5.- Lista detallada de los campos.

6.- Lista detallada de los constructores.

7.- Lista detallada de los métodos.

Toda esta información que describe qué hace la clase y cómo usarla (sin mostrar el código fuente o implementación) se denomina interfaz o interface de la clase. El código de implementación de la clase queda oculto (principio de ocultamiento de la información) y como programadores no vamos a tener acceso ni vamos a necesitar tener acceso a él.

La interfaz de clase nos muestra todos los constructores y métodos que se hayan definido como *public* en la clase. Por el contrario, no se van a mostrar aquellos constructores o métodos que se hayan definido como *private*. El motivo para ello es que los constructores o métodos *private* se consideran código auxiliar para su uso exclusivo dentro de la clase al que no se debe tener acceso desde fuera de ella.

> ***El conjunto de signaturas de métodos y constructores públicos de una clase constituyen su interfaz o interface. En esencia, la interface es una abstracción que consiste en que conocemos la signatura de los métodos (qué hacen) pero no su implementación (cómo lo hacen). Muchas veces se hace referencia a la implementación como "parte privada de una clase" para distinguirla de la parte pública (interfaz). Esta forma de trabajar se dice que hace uso del principio de ocultamiento de la información y se ha demostrado que es beneficiosa para una buena programación.***

Veamos en síntesis (no la veremos de forma completa porque resulta realmente extensa) los contenidos que ofrece la documentación de la clase String. Ten en cuenta que puede haber algunos cambios según la versión de Java que se emplee.

Nombre de la clase y descripción general

java.lang

Class String

java.lang.Object
  └ **java.lang.String**

All Implemented Interfaces:
Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

The `String` class represents character strings. All string literals in Java programs, such as `"abc"`, are implemented as instances of this class. Strings are constant; their values cannot be changed after they are created. String buffers support mutable strings. Because String objects are immutable they can be shared. For example:

```
String str = "abc";
```

Continúa comentarios, ejemplos y cuestiones relevantes de la clase…

Since:
JDK1.0

See Also:
`Object.toString()`, `StringBuffer`, `StringBuilder`, `Charset`, Serialized Form

Lista breve de campos (Field Summary)

Field Summary	
`static` Comparator<String>	**CASE_INSENSITIVE_ORDER** A Comparator that orders `String` objects as by `compareToIgnoreCase`.

Lista breve de constructores (Constructor Summary)

Constructor Summary
String() Initializes a newly created `String` object so that it represents an empty character sequence.
String(char[ ] value) Allocates a new `String` so that it represents the sequence of characters currently contained in the character array argument.

Continúa constructores de la clase…

Lista breve de métodos

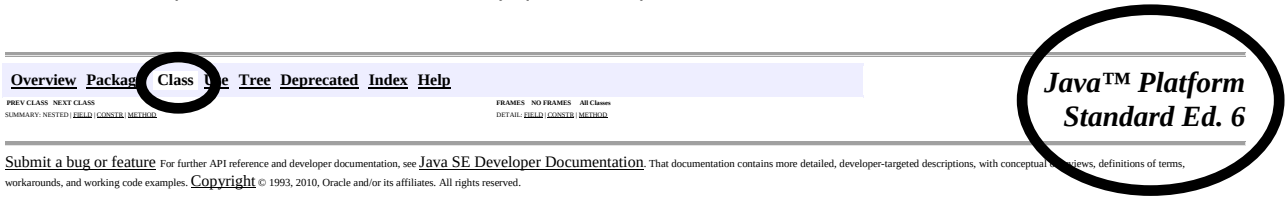
Method Summary	
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
int	**length**() Returns the length of this string.
boolean	**startsWith**(String prefix) Tests if this string starts with the specified prefix.
boolean	**startsWith**(String prefix, int toffset) Tests if the substring of this string beginning at the specified index starts with the specified prefix.
String	**substring**(int beginIndex) Returns a new string that is a substring of this string.

String	**substring**(int beginIndex, int endIndex) Returns a new string that is a substring of this string.
	Continúa métodos de la clase…

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Detalle de campos (Field Detail), constructores (Constructor Detail) y métodos (Method Detail)

En la parte inferior de la documentación se encuentra información de detalle (más extensa) sobre los elementos que se muestran en las listas breves o sumarios. Podemos acceder a ellos pulsando sobre el elemento correspondiente de la lista breve o bien recorriendo completamente la documentación. En distintas partes (encabezado, final) de la documentación nos aparece la información de a qué versión de Java corresponde esa documentación y qué es lo que se documenta:



Hemos señalado dos cosas a las que debemos de prestar atención: a qué corresponde la documentación que estamos consultando. Corresponde a una clase y a Java Platform Standard Ed. **X** (Java SE **X**) donde la X es la versión de Java. Si no realizamos bien las búsquedas puede ocurrir que estemos consultando la documentación de algo que no sea una clase, o de una distribución de Java que no sea aquella con la que estemos trabajando. A nivel profesional, es posible que nos veamos en la tesitura de tener que trabajar en algunos proyectos con Java SE 6 y en otros con Java SE 12 o posteriores. Hemos de prestar la atención necesaria para diferenciarlos y para no confundirnos.

Hemos dicho que dada una interfaz de una clase no vamos a necesitar ver el código de implementación de la clase. ¿Es esto siempre cierto? Digamos que sí siempre que la interface esté bien redactada y documentada, como es el caso de la biblioteca estándar Java. Si estamos trabajando en una empresa y nos facilitan una interfaz de clase desarrollada por otras personas y esta interfaz es pobre, incompleta o mal redactada, es posible que nos veamos obligados a solicitar consultar el código fuente para entender la clase. En proyectos grandes, es frecuente que sea necesario definir las interfaces de las clases antes incluso de que exista el código de implementación. El sentido que tiene esto es que podamos desarrollar código incluyendo las formas de invocación previstas en la signatura de las clases, independientemente de que el código de esas clases tarde en desarrollarse o, una vez desarrollado, vaya sufriendo cambios o mejoras. **La idea que subyace al concepto de interfaz es la abstracción:** saber qué hace y cómo usar una clase, pero "olvidarnos" de su implementación (por implementación entendemos el código completo que define una clase). La distinción entre interfaz e implementación es un concepto clave en programación orientada a objetos.

Próxima entrega: CU00649B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188