Para qué sirve *SimpleDateFormat*

La clase *SimpleDateFormat* nos ayuda a mostrar las fechas en el formato que queramos o a reconstruirlas a partir de una cadena de texto.

Convertir un *Date* a *String*

Un uso simple de *SimpleDateFormat* para escribir una fecha en pantalla puede ser este

import java.text.SimpleDateFormat;

...

SimpleDateFormat formateador = new SimpleDateFormat("dd/MM/yyyy");

// Esto muestra la fecha actual en pantalla, más o menos así 26/10/2006

System.out.println(formateador.format(new Date()));

Por supuesto, podemos jugar con la máscara y ponerla a nuestro gusto, dentro de las posibilidades que nos ofrece la API de *SimpleDateFormat*.

SimpleDateFormat formateador = new SimpleDateFormat("'Hoy es' EEEEEEEEE dd 'de' MMMMM 'de' yyyy");

SimpleDateFormat formatea = new SimpleDateFormat("dd/MM/yyyy");

SimpleDateFormat forma = new SimpleDateFormat("'Ingresaste' EEEEEEEEE dd 'de' MMMMM 'de' yyyy");

DateFormat formatoFecha=DateFormat.getDateInstance(DateFormat.FULL);

En la [API de *SimpleDateFormat*](http://java.sun.com/j2se/1.5.0/docs/api/java/text/SimpleDateFormat.html) podemos ver todas las opciones para la máscara de fecha y hora.

Obtener un *Date* a partir de un *String*

Para reconstruir una fecha a partir de la cadena de texto, podemos usar la misma clase *SimpleDateFormat*. Por ejemplo, si un usuario escribe una fecha con este formato en un **JTextField**, podemos leerla así

import java.text.SimpleDateFormat;

...

JTextField textField = new JTextField();

...

SimpleDateFormat formateador = new SimpleDateFormat("dd/MM/yyyy");

try

{

Date fecha = formateador.parse(textField.getText());

}

catch (ParseException e)

{

// Error, la cadena de texto no se puede convertir en fecha.

}

**Una cosa interesante es que a la hora de convertir un *String* a *Date*, la clase *SimpleDateFormat* es "indulgente", es decir, si metemos mal el *String*, trata de corregirlo. Por ejemplo, si introducimos una fecha "32 de Enero", no obtendremos error, sino que se arreglará y nos devolverá un Date correspondiente a "1 de Febrero". Podemos eliminar este comportamiento con el método *setLenient(false)*;**

SimpleDateFormat formateador = new SimpleDateFormat("dd/MM/yyyy");

formateador.setLenient(false);

de esta forma, el "32 de Enero" dará un error.

Otro aspecto importante a tener en cuenta es nunca enviar

SimpleDateFormat formateador = new SimpleDateFormat("dd/mm/yyyy");

en vez de

SimpleDateFormat formateador = new SimpleDateFormat("dd/MM/yyyy");

debido a que mm se refiere a minutos y no meses.

Locale

SimpleDateFormat trabaja por defecto en el idioma del sistema operativo. Por ejemplo, si nuestro sistema operativo es español y ejecutamos el siguiente código

SimpleDateFormat dateFormat = new SimpleDateFormat("EEEE MMMM d HH:mm:ss z yyyy");

System.out.println(dateFormat.format(new Date()));

obtendremos la siguiente salida

martes junio 18 07:39:34 CEST 2013

es decir, EEEE corresponde al día de la semana y lo obtenemos en español, mientras que MMMM corresponde al mes y lo obtenemos también en español (M da el mes con una cifra, por ejemplo, 6. MM lo da con dos, es decir, 06. MMM da tres letras del mes, es decir jun. MMMM da todas las letras del mes necesarias. De igual forma sucede con E, EE, EEE y EEEE para el día de la semana).

De igual forma, el método parse() de SimpleDateFormat espera en ese idioma por defecto (español en este caso) el día de la semana, mes, etc en el caso de que estén con texto y no con cifras. El siguiente código funcionará correctamente

dateFormat.parse("martes junio 18 07:39:34 CEST 2013");

con dateFormat definido como en el trozo de código anterior.

Podemos cambiar este idioma por defecto en el momento de construir el SimpleDateFormat, pasando un segundo parámetro con el Locale desdeado. Por ejemplo, si con el sistema operativo en español usamos el Locale.UK

SimpleDateFormat dateFormat = new SimpleDateFormat("EEEE MMMM d HH:mm:ss z yyyy", Locale.UK);

String fecha = dateFormat.format(new Date());

el código de salida será

Tuesday June 18 07:45:18 CEST 2013

y nuestro ejemplo de parse anterior

dateFormat.parse("martes junio 18 07:39:34 CEST 2013");

fallará por tener los textos en español (martes y junio).

Aparte de los Locale definidos directamente el Locale, como Locale.UK, Locale.US, Locale.GERMAN, ... podemos construir nuestro propio Locale pasando dos parámetros al constructor de Locale, uno de idioma y otro de país, por ejemplo

Locale espanol = new Locale("es","ES");

El método Locale.getAvailableLocales() nos permite saber que combinaciones de idioma/país son válidas. El siguiente código las muestra todas

Locale[] locales = Locale.getAvailableLocales();

for (Locale local:locales){

System.out.println(local.getLanguage() + ", "+local.getCountry());

}