APRENDERAPROGRAMAR.COM

CLASE DATE DEL API JAVA. MÉTODOS BEFORE, AFTER, TOLOCALESTRING, TOGMTSTRING Y GETTIME. EJEMPLOS. (CU00924C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2029

Resumen: Entrega nº24 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra

INTRODUCCIÓN

A partir de la introducción de la versión Java 8, el manejo de las fechas y el tiempo ha cambiado en Java. Desde esta versión, se ha creado una nueva API para el manejo de fechas y tiempo en el paquete java.time, que resuelve distintos problemas que se presentaban con el manejo de fechas y tiempo en versiones anteriores. Sin embargo, nos podemos encontrar con la necesidad de tener que trabajar con código que usa versiones anteriores o que sigue usando la clase Date del paquete java.util.

Tener en cuenta que si se está usando una versión de Java igual o superior a la 8 no deben usarse estas clases sino las proporcionadas dentro del paquete java.time.

En el paquete java.util encontramos la clase Date, que representa una fecha con precisión de milisegundos. De ella heredan otras clases como Time o Timestamp que no veremos de momento (estas clases pertenecen al paquete java.sql, y son utilizadas precisamente cuando una aplicación Java se conecta a bases de datos con campos fecha de precisión nanosegundos).

DATE

La clase Date fue de las primeras en este paquete y como tal ha ido sufriendo cambios. Bastantes métodos están "deprecados". Un método deprecado (deprecated) es un método "obsoleto" de uso no recomendado. Estos métodos han sufrido mejoras o cambios que se reflejan en otros métodos o clases de uso recomendado en lugar del deprecated. Por compatibilidad se permite el uso de estos métodos, aunque se recomienda su sustitución por otros. El motivo por los que muchos de los métodos de esta clase son deprecated es sobre todo por temas de internacionalización en el formato de fechas que no se tuvieron en cuenta en las primeras versiones de Java. A pesar de todo, esta clase permite la interpretación de fechas como año, mes, día, hora, minutos y segundos y continua siendo bastante usada entre la comunidad de programadores.

Algunas consideraciones que debemos tener en cuenta cuando usemos esta clase es que en todos los métodos que acepten o devuelvan años, meses, días, horas, minutos y segundos se trabaja de esta manera:

- Un año "y" se representa por el entero y – 1.900. Por ejemplo el año 1982 se representaría por el entero 1982 – 1900 = 82. De este modo, 82 representa 1982 y 92 representa 1992.
- Los meses son representados desde 0 hasta 11, así Enero es 0 y Diciembre es 11.
- Los días son normalmente representados desde 1 al 31.
- Las horas desde 0 a 23.

- Los minutos van desde 0 a 59.
- Los segundos normalmente van desde 0 hasta 59. (Excepcionalmente pueden existir los segundos 60 y 61 para los años bisiestos).

EJEMPLO DE USO DE DATE

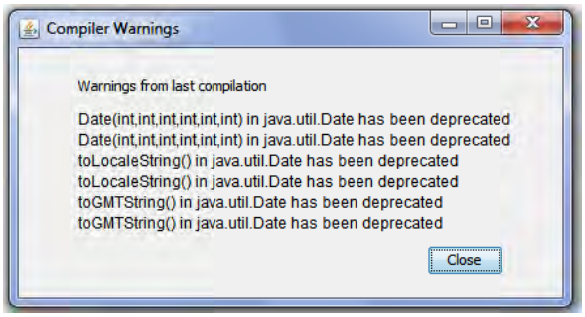
En el ejemplo que vamos a dar a continuación veremos cómo se puede usar la clase Date para representar una fecha determinada. Aprovechando que implementa la interfaz Comparable<Date>, compararemos 2 fechas para saber cuál va antes o después:

```
/* Ejemplo Clase Date aprenderaprogramar.com */
import java.util.Date;
public class Programa
{
   public static void main (String []args)
   {
      Date fecha1, fecha2;
      fecha1 = new Date(82,4,1,10,30,15);
      fecha2 = new Date(112,7,7,18,25,12);
      System.out.println("Fecha 1 Local: "+fecha1.toLocaleString());
      System.out.println("Fecha 2 Local: "+fecha2.toLocaleString());
      System.out.println("Fecha 1 en GMT: "+fecha1.toGMTString());
      System.out.println("Fecha 2 en GMT: "+fecha2.toGMTString());
      System.out.println("Fecha 1: "+fecha1.toString());
      System.out.println("Fecha 2: "+fecha2.toString());
      System.out.println("¿Es la fecha 1 posterior a la fecha 2?: "+fecha1.after(fecha2));
   }
}
```

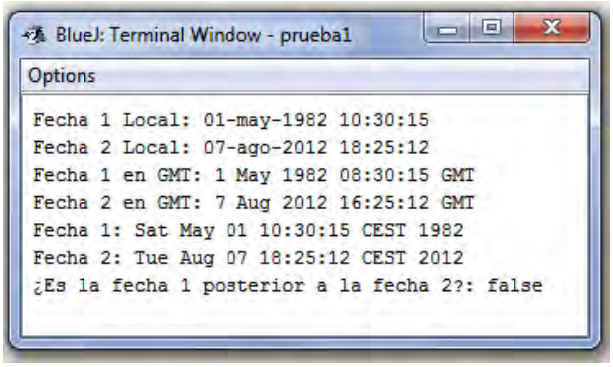
El constructor que hemos utilizado es uno de los disponibles, en concreto Date(int year, int month, int date, int hrs, int min, int sec) donde el primer número representa el año, el segundo el mes, el tercero el día, el cuarto las horas, el quinto el minuto y el sexto los segundos.

En este caso también no hay diagrama de clases, ya que tan solo hemos usado nuestra clase principal Programa.

Al compilar nuestro Programa en BlueJ nos saldrá probablemente un mensaje alertando del uso de métodos deprecados como ya hemos comentado anteriormente.



El resultado de ejecución del programa nos devuelve la siguiente salida:



Como podemos observar en la salida, imprimimos las fechas en varios formatos tanto la fecha 1, como la fecha 2. El primero es el formato local donde se representa en este caso las fechas con el formato del país donde nos encontremos. Esto se debe a que en tiempo de ejecución se recupera el formato de fecha correspondiente al ordenador donde se esté ejecutando la aplicación mediante una propiedad de la máquina virtual de java. Si estamos en España, recuperaremos la hora de España y si estamos en México la de México, en Chile la de Chile, etc.

En algunos casos nos interesa crear una fecha partiendo de un String, para ello podemos utilizar el constructor utilizando la cadena String, pero si la fecha introducida en este caso no sigue el formato local deberemos de utilizar la clase DateFormat para parsear (transformar) el String y poder crear la fecha Date correspondiente.

El segundo par de formatos es el formato GMT donde podemos observar que se muestra con 2 horas menos (diferencia entre la hora local de España que es donde hemos hecho el programa y la hora GMT). Mientras que el formato tercero es el que por defecto muestra Date.

Por último observamos cómo podemos comparar 2 fechas, utilizando los métodos before o after (en este caso) además podríamos haber utilizado por supuesto el compareTo.

Un ejercicio interesante puede ser el calcular la diferencia en días entre 2 fechas dadas, así vamos a realizar este ejercicio:

```
/* Ejemplo Clase Date aprenderaprogramar.com */
import java.util.Date;
public class Programa
{
   public static void main (String []args)
   {
      Date fecha1, fecha2;
      long diferencia = 0;
      fecha1 = new Date(112,7,1,10,30,15);
      fecha2 = new Date(112,7,7,18,25,12);
      diferencia = fecha2.getTime()-fecha1.getTime();
      System.out.println("Diferencias en dias: "+diferencia/(3600000*24));
   }
}
```

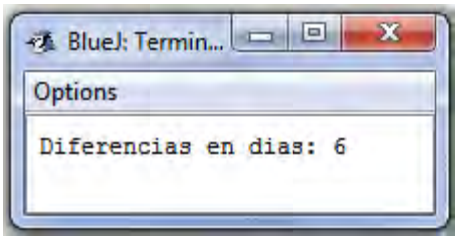
Las fechas creadas las interpretamos así:

(112,7,1,10,30,15): (año, mes, día, hora, minutos, segundos): año 1900+112 = 2012, mes 7 que es agosto (el mes cero es enero), día 1, hora 10, minutos 30, segundos 15.

Por tanto la primera fecha es el 1 de agosto de 2012 a las 10 horas 30 minutos 15 segundos.

(112,7,7,18,25,12): de la misma forma llegamos a la conclusión de que la fecha representada es el 7 de agosto de 2012 a las 18 horas 25 minutos 12 segundos.

Entre el 1 de agosto y el 7 de agosto hay 6 días (7-1 = 6). Si quisiéramos obtener los días totales ambos inclusive bastaría con sumarle uno al resultado. Al obtener la salida del programa efectivamente vemos que la diferencia en días es la siguiente:



El método getTime() aplicado sobre un objeto object devuelve un entero largo (long) que representa el número de milisegundos desde el 1 de enero de 1970 a las 0 horas GMT (este es un valor de referencia que utiliza Java, podría haber sido otra fecha, pero los desarrolladores de Java eligieron esta). Para pasar de milisegundos a segundos hemos de dividir por 1000 y para pasar de segundos a días hemos de dividir por 24 horas * (3600 segundos/hora). Resulta una operación: numero / (3600000 *24) de forma que el valor de milisegundos numero queda transformado en días.

En resumen lo que hacemos es obtener la diferencia en milisegundos entre las dos fechas y transformarlo a días. El resultado lo almacenamos en un tipo long (entero largo) de forma que si lleva decimales los decimales quedan truncados.

Nota:

No todos los métodos de la clase Date están deprecados pero sí bastantes de ellos, por eso se recomienda utilizar para versiones de Java previas a la 8 la clase **DateFormat** para formatear y parsear cadenas String a fechas y usar la clase **Calendar** para conversiones de fechas y sus campos, cuestión que no vamos a abordar ahora. Para versiones Java a partir de la 8 se recomienda usar las clases del paquete java.time, cuestión que tampoco vamos a abordar ahora.

Próxima entrega: CU00925C

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:
http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180