APRENDERAPROGRAMAR.COM

CLASE CALENDAR Y GREGORIANCALENDAR DE JAVA. CONVERSIÓN DE FECHAS. EJEMPLOS. CAMBIOS DESDE JAVA 8 (CU00925C)

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

Fecha revisión: 2029

Resumen: Entrega nº25 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 las clases para el manejo de fechas y tiempo de 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 apartados anteriores del curso vimos cómo usar la clase Date de Java para manejar fechas. Pero Date tiene distintas limitaciones (de hecho vimos que gran parte de sus métodos están "deprecated"), ni contiene métodos para realizarle consultas. Date simplemente representa una fecha exacta. Calendar también representa una fecha pero su principal objetivo es que sea mutable, es decir, permitir cambiar su valor sin generar un nuevo objeto (cosa que no permite Date).

CALENDAR

La clase Calendar posee una gran cantidad de métodos para operar, consultar y modificar las propiedades de una fecha. Un aspecto principal es que es una clase abstracta y como tal posee algunos métodos que deben ser implementados por sus subclases.

Calendar se suele instanciar con su método getInstance() el cual nos crea un objeto de la clase conteniendo la fecha de ese momento. Así es muy típico el uso:

Calendar *ahoramismo* = ***Calendar****.getInstance();*

Calendar tiene 2 métodos de funcionamiento, lo que se llama "lenient" o "non-lenient" mode. Es decir modo permisivo o modo no permisivo. Por defecto se trabaja en modo permisivo y esto quiere decir que si configuramos un Calendar como el día 32 de Enero (lo cual sería un error), a la hora de formatear la fecha y por ejemplo imprimirla por pantalla se mostrará el 1 de Febrero. Es decir, con lenient mode Java trata de encontrar una fecha si le es posible aunque hayamos introducido un dato erróneo.

Si configuramos el Calendar en modo no permisivo, antes de calcular la fecha más asemejable lanzaría una excepción si algún parámetro sale de su rango permitido. El 32 de enero daría error.

El conjunto de métodos “set” permite establecer una fecha, mientras que los métodos “add” y “roll” permiten cambiar las fechas sumando o restando una cantidad. Estos dos últimos métodos fuerzan que los valores para los campos no sobrepasen el mínimo o el máximo del permitido según el calendario. También estos métodos suponen un recálculo inmediato de la fecha tras el cambio de sus valores, cosa que no ocurre con el uso de los métodos set.

Por ejemplo imaginemos que tenemos una fecha con los datos siguientes: 31 de Agosto de 2000. Si ahora llamáramos sobre ese objeto Calendar el método add(Calendar.MONTH, 13) lo que hacemos es añadir 13 meses más a la fecha, con lo que tendríamos la fecha 30 de Septiembre del 2001. Observamos que al añadir 13 meses nos vamos al mes siguiente del año siguiente, pero sin saltar ninguna excepción hemos pasado del día 31 al 30, esto es porque add como hemos dicho fuerza los valores para que no sobrepasen del mínimo o máximo del campo correspondiente, en este caso del campo de días.

GREGORIANCALENDAR

Esta clase es una clase concreta de Calendar y que por otro lado es el sistema de calendario estándar en el mundo, al menos en el mundo occidental que comprende Europa, Norte, Centro y SurAmérica y muchos otros países.

Por ello es la que vamos a utilizar en nuestro ejemplo a continuación, en donde vamos a ver cómo podemos trabajar con una fecha y realizar cambios sobre ella según nuestro calendario.

EJEMPLO DE USO DE CALENDAR Y GREGORIANCALENDAR

En el ejemplo que vamos a dar a continuación veremos cómo se puede usar la clase Calendar y GregorianCalendar para representar una fecha determinada y poder modificarla según el calendario gregoriano:

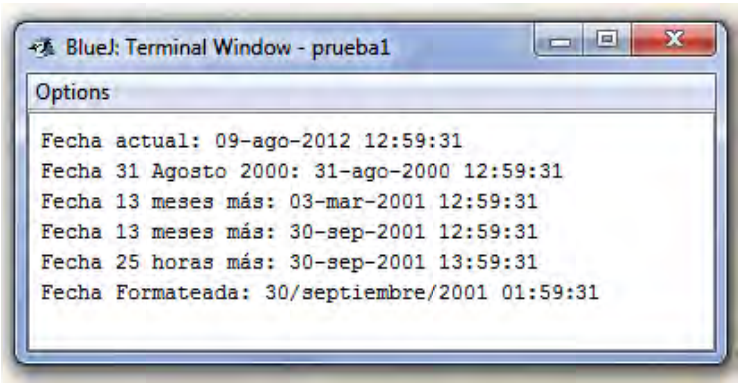
```
/* Ejemplo Clase Calendar y GregorianCalendar aprenderaprogramar.com */

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;

public class Programa
{
   public static void main (String []args)
   {
      Calendar c1 = GregorianCalendar.getInstance();
      System.out.println("Fecha actual: "+c1.getTime().toLocaleString());
      c1.set(2000, Calendar.AUGUST, 31);
      System.out.println("Fecha 31 Agosto 2000: "+c1.getTime().toLocaleString());
      c1.set(Calendar.MONTH, 13);
      System.out.println("Fecha 13 meses más: "+c1.getTime().toLocaleString());
      /* como podremos observar no nos imprimirá la fecha deseada, hemos escrito una incoherencia*/
      c1.set(2000, Calendar.AUGUST, 31);
```

```
        /* volvemos a la fecha anterior usando set y empleamos el método add */
        c1.add(Calendar.MONTH, 13); /* Añadir 13 meses */
        System.out.println("Fecha 13 meses más: "+c1.getTime().toLocaleString()); /*ahora sí es correcto*/
        c1.roll(Calendar.HOUR, 25); /* Añadir 25 horas */
        System.out.println("Fecha 25 horas más: "+c1.getTime().toLocaleString());    /*uso de roll*/
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MMMMM/yyyy hh:mm:ss");
        System.out.println("Fecha Formateada: "+sdf.format(c1.getTime()));
    }
}
```

En este caso no mostramos diagrama de clases, ya que tan solo hemos usado nuestra clase principal Programa. El método getTime() nos permite obtener un objeto Date a partir del objeto Calendar y al objeto Date le podemos pedir una representación tipo String con el método toLocaleString().

Al compilar nuestro Programa en BlueJ nos saldrá probablemente un mensaje alertando del uso de métodos deprecados. No le daremos importancia ahora a este hecho.

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



Como podemos observar en la salida, primero hemos impreso la fecha actual del sistema ya que al inicializar nuestro objeto GregorianCalendar con la instrucción: *GregorianCalendar.getInstance();* ésta toma por defecto los valores locales y actuales del sistema.

Posteriormente observamos cómo alteramos la fecha con el método set indicándole que queremos establecer que el mes para esa fecha sea el mes 13. Observamos que si utilizamos set de esta manera (errónea, pues no existe ningún mes que sea el 13) lo que hace Java es tratar de buscar la fecha que considera que se ajusta a lo que hemos indicado, pero no siempre lo hará bien o como nosotros pudiéramos esperar. En este caso al indicar que el mes sea el 13 Java nos muestra la fecha 3 de marzo del 2001 lo cual es un resultado "imprevisible" ya que no corresponde a sumarle 13 meses ni a establecer el mes 3.

Si escribiéramos 3 en lugar de 13, es decir c1.set(Calendar.MONTH, 3); el resultado obtenido sería 01-may-2000 17:22:37, es decir, hemos establecido que el mes de la fecha es el mes 3 (siendo 0 enero, 1 febrero, 2 marzo, 3 abril y 4 mayo). ¿Por qué nos devuelve mayo si el mes 3 es abril? Porque Java busca el mes que le hemos indicado (abril) y comprueba que abril tiene 30 días, como nuestra fecha es día 31

interpreta que hemos cometido un error y busca la fecha más aproximada posible y nos devuelve el 31 de mayo de 2000. Realmente se trata de una incoherencia por nuestra parte tratar de representar un día 31 de abril porque ese día no existe.

Fíjate que podemos tener problemas si tenemos activo el lenient mode. Para desactivarlo procedemos de esta manera: c1.setLenient(false);

Una vez desactivado al tratar de ejecutar una instrucción como c1.set(Calendar.MONTH, 3); Java detecta que estamos tratando de representar un día 31 de abril y al no existir tal día en lugar de buscar algo asemejable nos devuelve un error (java.lang.IllegalArgumentException: MONTH).

Observamos en la salida que utilizando el método add habiendo inicializado otra vez previamente nuestro calendar con la fecha del 31 de agosto del 2000, si sumamos 13 meses nos vamos al 30 de septiembre del 2001, lo cual es correcto. Además el método add nos ha corregido el desvío automáticamente ya que en un Calendario Gregoriano Septiembre tiene 30 días y no 31.

Posteriormente vemos el uso del método roll que también añade valores a cierto parámetro, en este caso hemos tratado de añadir 25 horas. ¿Por qué no nos ha añadido 25 horas? Porque para roll el número máximo de horas que se pueden añadir es 23 (si añadiéramos 24 obtendríamos exactamente la misma fecha inicial sin cambios). Con roll, a diferencia de con el método add, no modificamos los parámetros más largos permisibles para un campo. Por tanto no podemos añadir más de 23 horas tratando de modificar una hora ni más de 11 meses tratando de modificar un mes. Es decir, nosotros al añadir 25 horas no aumentamos en un día y una hora con el método roll. Sin embargo con add sí podríamos sumar a nuestro calendario 25 horas porque add no establece una limitación en cuanto al valor máximo admisible.

Si hubiéramos escrito c1.roll(Calendar.HOUR, 2); entonces sí se añadirían correctamente dos horas obteniendo como fecha 30-sep-2001 20:05:21.

Para finalizar, comentar el uso de DateFormat que como indicamos en apartados anteriores del curso es la clase utilizada para formatear fechas.

Al igual que Calendar, DateFormat es una clase Abstracta y por eso hemos tenido que utilizar una subclase concreta de esta. En este caso hemos usado SimpleDateFormat, a la cual en el constructor le decimos como deseamos el patrón de fechas:

new SimpleDateFormat("dd/MMMMM/yyyy hh:mm:ss");

En este caso le hemos introducido que nuestro formato de fecha es dd referente a los días en este caso “30”, MMMMM para que escriba el nombre de los meses en este caso “septiembre”, yyyy para los años en este caso 2001 y hh:mm:ss para las horas, minutos y segundos respectivamente, en nuestro caso 01:59:31.

A la hora de la impresión o el formateo se utiliza el método format pasándole como argumento la fecha deseada como objeto Date, por ejemplo: *sdf.format(c1.getTime());*

CONCLUSIONES

Hemos visto el uso de Calendar y GregorianCalendar así como también someramente el formateo de fechas con DateFormat y SimpleDateFormat. Hemos visto cómo podemos fácilmente cambiar un objeto que represente una fecha y las consecuencias que tiene dependiendo del método utilizado y de tener o no activo el lenient mode. El uso de estas clases sólo está indicado si trabajamos con versiones de Java anteriores a la 8. Si trabajamos con Java 8 o superior, se recomienda el uso de las clases incluidas dentro del paquete java.time para el manejo de tiempos y fechas (clases como Clock, Duration, Instant, LocalDate, LocalTime, MonthDay, OffsetDateTime, OffsetTime, Period, Year, YearMonth, ZonedDateTime, ZoneId y ZoneOffset).

Próxima entrega: CU00926C

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