

Notas técnicas de JAVA Nro. 8 –Tip en detalle

(Lo nuevo, lo escondido, o simplemente lo de siempre pero bien explicado)

Manejo de Fechas en Java: Consideraciones sobre clases Date Vs. Calendar

Tema: Date, Calendar

Descripción: Explica la manipulación y características de conversión que existe entre las clases Date y Calendar. Asimismo, sugiere soluciones para distintas necesidades de manejo de fechas.

Nivel: Avanzado

Fecha pub: Mayo 2005

"Notas Técnicas de JAVA" se envía con frecuencia variable y absolutamente **sin cargo** como un servicio a nuestros clientes. Contiene notas/recursos/artículos técnicos desarrollados en forma totalmente objetiva e independiente. Teknoda es una organización de servicios de tecnología informática y **NO comercializa hardware, software ni otros productos**. Si desea suscribir otra dirección de e-mail para que comience a recibir los tips envíe un mensaje desde esa dirección a develop@teknoda.com, indicando su nombre, empresa a la que pertenece, cargo y país.

Lista de Tips publicados hasta la fecha:

1. JAVA Basics: Cómo conformar un entorno de programación JAVA (serie de varios tips). Parte I: Selección e instalación de un IDE gratuito.
2. Una introducción a JDBC (Java Database Connectivity) (Acceso a bases de datos desde JAVA)
3. Manejo del error "Bad Magic Number"
4. Java Basics: Entendiendo la Java Virtual Machine
5. Organización de memoria en JAVA Vs. Modelo Tradicional
6. JAVA Basics: Entendiendo los applets
7. JAVA Basics: Diferencias conceptuales entre JavaBeans y Enterprise JavaBeans
8. Manejo de Fechas en Java: Consideraciones sobre clases Date Vs. Calendar

Próximos Tips:

Nivel Técnico avanzado

- JAVA Vs. C++

Nivel Básico

- JAVA Basics: Entendiendo los servlets
- JAVA Basics: Mitos y Verdades sobre JAVA

Tabla de contenido

- I. Introducción
- II. Características de la clase Date y Calendar
- III. Escenarios posibles para utilización de la clase Date y Calendar
- IV. Consideraciones Finales
- V. Dónde obtener información adicional

I. Introducción

Cuando se trabaja con fechas en Java, es usual que surjan dudas respecto de si conviene elegir la clase Date o la clase Calendar para manejar este tipo de dato. La diversidad de los constructores de cada una de estas clases, la forma de utilizar sus métodos para calcular diferencias de días, puede generar confusión a la hora de optar por una o por otra. De hecho, muchos terminan utilizando la clase de manejo de fecha que mejor conocen o una clase “Fecha” desarrollada especialmente, aunque esto fuera sub-óptimo.

La idea de este tip es explicar las características de ambas clases (Date y calendar), plantear los distintos escenarios para la utilización de las mismas, aprender los métodos más útiles y sugerir el código más adecuado para facilitar el mantenimiento de las aplicaciones.

II. Generalidades de la Clase Date y Calendar

La clase Date

La clase Date se remonta al primer entorno de desarrollo estandar de Java (JSDK), donde existía como *java.util.Date*. Esta clase ha sido utilizada por la mayoría de los programadores durante todos estos años para manipular fechas y hora. La clase *Date* permite representar un instante de tiempo específico como, **con precisión de milisegundos**; esto es, podemos extraer de ella año, mes, día, horas, minutos, segundos y milisegundos.

Antes del JDK 1.1 la clase *java.util.Date* tenía dos funciones adicionales a las que conocemos ahora, relacionadas a la interpretación de datos: el formateo (el aspecto con que se muestran los) y “parseo” (conversión de un “string” que contiene una fecha a *java.util.Date*).

La clase Date comenzó a “deprecarse” debido a las dificultades que presentaban los métodos de *Date* a la hora de “internacionalizar” los programas y la escasez de métodos para realizar operaciones. Por lo tanto, se necesitaba una nueva clase que se adaptara a la zona horaria donde correría la aplicación Java, e inclusive se adaptara al tipo de Calendario que perteneciera a ese lugar geográfico. Estas necesidades dieron origen, como veremos, a la creación de la clase *Calendar*.

En la actualidad, la clase Date posee varios constructores y métodos “getters” y “setters” deprecados que tienen sus equivalentes en la clase *Calendar*. Sin embargo, **la clase Date en si misma, no se encuentra deprecada por ahora.**

La clase Calendar

La clase *Calendar* es abstracta y se considera como base para convertir un conjunto de campos enteros como YEAR (año), MONTH (mes), DAY (día), HOUR (hora) a una instancia heredada de la clase *Calendar*.

Una subclase de *Calendar* representa una fecha, de acuerdo a las reglas de un calendario específico. La especificación provee una subclase concreta de *Calendar*: **GregorianCalendar**. Otras subclases podrían representar varios tipos de calendarios lunares usados en diferentes lugares del mundo.

III. Escenarios posibles para la utilización de la clase Date y/o Calendar .

Escenario 1: Problemas de actualización de una aplicación Java ya desarrollada

En un futuro cercano, la deprecación (casi un 90%) de los distintos métodos de *Date* llevará a la caducidad de *Date* como clase, por lo tanto todos aquellos sistemas que trabajen con JVM y entornos actualizados no reconocerán a *Date* como clase propia de Java. Este escenario hace inevitable **el traslado de esta clase a *Calendar***, la cual aporta amplitud en la manipulación de diferentes calendarios y soporta acciones que se realizan con objetos de tipo *Date*.

Sin embargo, al encarar este cambio, aparece una dificultad al actualizar los tipos de los objetos del sistema, dado **que no existe una forma inmediata para transformar un objeto de tipo *Date* a un objeto de tipo *Calendar***: (es decir, no hay un “casteo” válido ni tampoco método alguno proporcionado por Sun)

Para poder lograrlo, es necesario aplicar “receta” que a continuación desarrollamos.

Solución para el Escenario 1: Transformación de un objeto Date a un objeto Calendar y seteos de un objeto Calendar

Suponemos las siguientes consideraciones...

- El entorno actual ya no posee a *Date* como clase propia de JAVA.
- No tendremos instaladas versiones anteriores a la actual para poder utilizar los métodos deprecados (debido a que se podría poner en peligro la funcionalidad de todo el sistema)
- Se pretende transformar los objetos *Date* a objetos *Calendar*

Pasos a seguir en dicha transformación:

1. Convertir un objeto *Date* a *String*.
2. Parsear el *String*, separando las variables día, año, fecha, hora, etc.
Por ejemplo: `String strYear = strDate.substring(strDate.length() - 4, strDate.length());`
3. Tomar las variables obtenidas como *String* que representan el día y el mes. Convertirlas al formato entero que las representa. (Se aconseja utilizar las tablas de equivalencias que se presentan a continuación)

Por ej: Si el objeto *Date* posee el siguiente formato: `Thu Oct 14 16:14:45 GMT-03:00 2004`

La variable representativa:

- Mes (en este ejemplo: `Oct`), no puede ser convertida a `int` o `Integer` directamente. Es necesario hacer una tabla de equivalencias según corresponda en el calendario gregoriano y la región.

Mes nro.	Nombre del mes
0	Enero
1	Febrero
2	Marzo
3	Abril
4	Mayo
5	Junio
6	Julio
7	Agosto
8	Septiembre
9	Octubre
10	Noviembre
11	Diciembre

La asignación del mes obtenido en Date a su equivalente en formato **int** se puede realizar a través de:

- una secuencia de if anidados ó
 - un ciclo *Switch* ó
 - la creación de CONSTANTES o LITERALES (ésta es la más aconsejada)
4. Tomar las variables obtenidas como String que representan la hora ej: **16:14:45** (sin la zona horaria **GMT-03:00**) y convertir las horas, minutos y segundos a int.
 5. Crear una variable Calendar y “setearla” con los datos obtenidos.

Como la clase Calendar es Abstracta, **no se puede hacer un new de esta clase**, en consecuencia se instancia de la siguiente manera:

```
Calendar fecha1 = Calendar.getInstance();
```

Posteriormente se utilizan algunos de los siguientes métodos de seteos, para obtener un objeto representativo de lo deseado:

void	set (int year, int month, int date) Setea los valores para los campos de año, mes y fecha .
void	set (int year, int month, int date, int hour, int minute) Setea los valores para los campos de año, mes, fecha, hora y minutos.
void	set (int year, int month, int date, int hour, int minute, int second) Setea los valores para los campos de año, mes, fecha, hora, minutos y segundos.

Sugerencia:

Manejar los meses del año como mes actual - 1 para los setteos. Si se quiere cargar abril, cargarlo como 4 -1 (el 4 se lo obtendría mediante métodos aplicados a la instancia Calendar, por dicho motivo se hizo explícita la operación algebraica).

Tener en cuenta que Enero corresponde a los valores 0 u 12 en el argumento correspondiente a 12. Calendar tiene como punto inicial el valor 0 del calendario por dicho motivo existe año,día y mes cero.

Ejemplo1:

```
Calendar cal = Calendar.getInstance();
cal.set(2005, 0, 0 );
System.out.println(cal.getTime());
```

Por consola:

```
Fri Dec 31 12:35:21 GMT-03:00 2004
```

Ejemplo2:

```
Calendar cal = Calendar.getInstance();
cal.set(2005, 0, 20 );
System.out.println(cal.getTime());
```

Por consola:

```
Thu Jan 20 12:42:01 GMT-03:00 2005
```

Ejemplo3:

```
Calendar cal = Calendar.getInstance();
cal.set(0, 12, 20 );
System.out.println(cal.getTime());
```

Por consola:

```
Thu Jan 20 12:49:32 GMT-03:00 1
```

Ejemplo4:

```
Calendar cal = Calendar.getInstance();
cal.set(2005, 12, 20 );
System.out.println(cal.getTime());
```

Por consola:

```
Fri Jan 20 12:42:44 GMT-03:00 2006
```

Prestar atención a las salidas por consola.

Escenario 2: Realización de operaciones con fechas en una aplicación Java

En las aplicaciones, es habitual tener la necesidad de realizar operaciones con las fechas para saber, por ej: cuántos días hace que se realizó un pedido o cuántos días faltan para el vencimiento de una factura, etc. Por lo tanto observamos que es muy común sumar y restar fechas.

La clase que nos provee estas funcionalidades es **Calendar**. Como consecuencia de ello, si tenemos una aplicación que utiliza la clase Date debemos transformar sus objetos a *Calendar*. (Ver Receta para el Escenario1: Transformación de un objeto Date a un objeto Calendar)

Tenemos en cuenta las siguientes consideraciones:

- El entorno actual es igual o superior a j2dsk 1.2, por lo tanto permite trabajar con la clase *Calendar* .
- No tendremos instaladas versiones anteriores a la actual para poder utilizar los métodos deprecados (debido a que se podría poner en peligro la funcionalidad de todo el sistema)

Para realizar operaciones con las fechas tenemos el siguiente método:

Abstract void	<u>add</u> (int field, int amount) Función aritmética de la fecha
---------------	--

Este método sirve para sumar (como su nombre lo indica), pero también para restar cantidades a fechas.

Solución para el Escenario 2: Ejecución de operaciones (suma y diferencias) en un objeto Calendar

Suma y Resta de fechas:

Este concepto puede diferir de la situación en la que se esté trabajando.

La operación conceptualmente puede ser:

- la sumatoria/resta de las cantidades de días de ambas fechas desde el punto cero del año, comienzo del mismo.
- la suma/resta de los días entre la fecha1 y la fecha2.

Con el método `fechaEjemplo.get(Calendar.DAY_OF_YEAR)` se obtienen la cantidad de días transcurridos en el año hasta la fechaEjemplo. Devolveria un número dentro de los 365 días del año.

Con el método `fecha1.get(Calendar.DATE)` se obtiene la cantidad de días transcurridos en el mes de la fecha. Devolveria un número dentro de los 30,31 o 28 días del mes.

A continuación como hacer sumatorias y restas según los 2 criterios enunciados anteriormente:

1.

- Al primer argumento del método se necesita pasarle el parámetro *Calendar.DATE*, el cual nos devuelve por cantidad de días.

- Al segundo parámetro del método, debemos pasarle al segundo: *fecha1.get(Calendar.DATE)* que nos devuelve el nro del día.

```
fecha2.add(Calendar.DATE, fecha1.get(Calendar.DATE));
```

2.

- Al primer argumento del método pasarle el parámetro *Calendar.DATE*, el cual nos devuelve por cantidad de días.
- Al segundo parámetro del método, debemos pasarle al segundo: *fecha1.get(Calendar.DAY_OF_YEAR)* que nos devuelve el nro del día del año.

```
fecha2.add(Calendar.DATE, fecha1.get(Calendar.DAY_OF_YEAR));
```

3. Si deseamos realizar una resta, debemos agregar simplemente un signo "-" delante del segundo parámetro.

```
fecha2.add(Calendar.DATE, - fecha1.get(Calendar.DATE));
```

```
fecha2.add(Calendar.DATE, - fecha1.get(Calendar.DAY_OF_YEAR));
```

A continuación se presenta un ejemplo, donde se trabaja con fechas pertenecientes al mismo mes:

```
public class Ejemplo {
    public static void main(String[] args)
    {
        int dia = 2;
        int mes = 10;
        int anio = 2004;

        Calendar fecha1 = Calendar.getInstance();
        fecha1.set( anio, mes - 1, dia);
        Calendar fecha2 = Calendar.getInstance(); //fecha de hoy

        System.out.print("Fecha Formada:  ");
        System.out.println("Fecha " + fecha1.get(5)+ "Mes " +
            fecha1.get(2)+ "Año" + fecha1.get(1));

        System.out.print("Antes de procesar (fecha actual):  ");
        System.out.println("Fecha " + fecha2.get(5)+ "Mes " +
            fecha2.get(2)+ "Año" + fecha2.get(1));

        fecha2.add(Calendar.DATE, fecha1.get(Calendar.DATE));

        System.out.print("se SUMA:" + fecha1.get(Calendar.DATE)+ "y el
            resultado es");
```

```

System.out.println("Fecha " + fecha2.get(5)+ "Mes " +
                  fecha2.get(2)+ "Año" + fecha2.get(1));

fecha2.add(Calendar.DATE, - fecha1.get(Calendar.DATE));
                //- para indicar resta

System.out.print("se SUMA:"+(- fecha1.get(Calendar.DATE))+"y el
resultado es");
System.out.println("Fecha " + fecha2.get(5)+ "Mes " +
                  fecha2.get(2)+ "Año" + fecha2.get(1));
    }
}

```

Por consola se verá:

Fecha Formada:	Fecha 2	Mes 9	Año 2005		
Antes de procesar (fecha actual):	Fecha 13	Mes 9	Año 2004		
se SUMA:	2	y el resultado es	Fecha 15	Mes 9	Año 2004
se SUMA:	-2	y el resultado es	Fecha 13	Mes 9	Año 2004

IV. Consideraciones Finales

Cuando trabajamos con base de datos a través de JDBC (Java DabaBase Connectivity) es muy posible que necesitemos almacenar fechas. Para ello tenemos tres clases :

- *java.sql.Date*. Hereda de *java.util.Date*, por lo tanto, tiene una precisión de milisegundos. De todas maneras, su formato por defecto sólo muestra el día, mes y año en la salida.
- *java.sql.Time* y *java.sql.Timestamp*. Estas clases heredan de *java.util.Date* por lo tanto es fácil realizar conversiones de *java.sql.Date* a *java.sql.Time* o *Timestamp*. Para realizar las conversiones debemos pasar todo a milisegundos y así luego poder utilizar el constructor deseado.

Ejemplo de los constructores de *java.sql.Time* y *Timestamp*

- `Timestamp(long millis)`
- `Time(long time)`

En cambio, si no partimos de los constructores mencionados anteriormente, *Date* nos provee del método `getTime()` que retorna el objeto *java.sql.Date* en milisegundos.

La clase *Time* posee un formato para representar los datos de horas, minutos, segundos y milisegundos. En cambio *Timestamp* representa estos mismos datos y además los nanosegundos.

V. Dónde obtener información adicional

Sitio de sun: <http://java.sun.com>

Developer Forums: <http://forum.java.sun.com>

Especificación de Clase Calendar: <http://java.sun.com/j2se/1.4.2/docs/api/java/util/Calendar.html>

Applet que realiza la conversión de fechas entre distintos calendarios: <http://web.meson.org/calendars/>

Información sobre los distintos tipos de calendarios: <http://web.meson.org/calendars/calinfo.html>

Copyright 2005 Teknoda S.A. Mayo 2005. JAVA es marca registrada de Sun. SAP, R/3 y ABAP son marcas registradas de SAP AG. AS/400 es marca registrada de IBM. Todas las marcas mencionadas son marcas registradas de las empresas proveedoras.

La información contenida en este artículo ha sido recolectada en la tarea cotidiana por nuestros especialistas a partir de fuentes consideradas confiables. No obstante, por la posibilidad de error humano, mecánico, cambios de versión u otro, Teknoda no garantiza la exactitud o completud de la información aquí volcada.

Dudas o consultas develop@teknoda.com