Tema 5.-La clase Calendar

Contenido

1 Introducción

Un tipo de datos de gran importancia en las aplicaciones de gestión y el control de procesos son las fechas.

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*** o ***Calendar*** del paquete **java.util**.

Es por ello que vamos a estudiar estas clases para entender la información que gestionan y los métodos que permiten acceder a ellas.

Los atributos definidos para la clase Calendar contienen información relativa al año, mes, día del mes, día de la semana, día del año, hora, minuto, segundo y el número de milisegundos desde el 1 de enero de 1970 a las 0:00:00 (Epoch). Los atributos que contiene un objeto Calendar y los métodos que acceden a ellos están declarados como `protected`.

1.1 Fecha/Hora actual

Calendar es una clase abstracta, por lo que no podemos hacer un new de ella. La forma de obtener una instancia es llamando al método ***getInstance(),*** que nos devolverá un objeto de alguna clase hija de *Calendar* inicializada con la fecha/hora actual.

```
import java.util.Calendar;//siempre habrá que importarla

Calendar ahora = Calendar.getInstance();
System.out.println("Hoy es " + ahora.getTime());
```

La clase hija que realmente nos va a devolver este método es *GregorianCalendar*, correspondiente al calendario estándar para el mundo occidental. Sobre esta si podremos hacer `new GregorianCalendar()` si queremos instanciar un objeto directamente con valores nulos o con valores.

Para mostrar la fecha/hora por pantalla, podríamos intentar directamente

```
System.out.println("Hoy es " + ahora);
```

Si queremos copiar un objeto Calendar en otro nuevo tenemos que utilizar el método **clone()** que nos devolverá un objeto de la clase Object. Para convertirlo en Calendar hay que aplicar casting (Calendar)

```
Calendar nuevo = (Calendar) ahora.clone();
```

Si utilizamos la asignación los dos objetos estarán en la misma dirección y los cambios se producirán en ambas variables.

1.2 Leer los campos de la fecha/hora

Calendar tiene el método ***get(…)*** al que se pasa como parámetro la constante que indica el atributo que queremos obtener.

Los principales nombres de las constantes que indican los atributos que tiene definidos la clase Calendar son:

int	**AM_PM** Field number for get and set indicating whether the HOUR is before or after noon.
int	**DATE** Field number for get and set indicating the day of the month.
int	**DAY_OF_MONTH** Field number for get and set indicating the day of the month.
int	**DAY_OF_WEEK** Field number for get and set indicating the day of the week. (de 0 a 6)
int	**DAY_OF_YEAR** Field number for get and set indicating the day number within the current year.
int	**HOUR** Field number for get and set indicating the hour of the morning or afternoon. (0-11)
int	**HOUR_OF_DAY** Field number for get and set indicating the hour of the morning or afternoon. (0-23)
int	**MILLISECOND** Field number for get and set indicating the millisecond within the second.
int	**MINUTE** Field number for get and set indicating the minute within the hour.
int	**MONTH** Field number for get and set indicating the month. (de 0 a 11)
int	**PM** Value of the AM_PM field indicating the period of the day from noon to just before midnight.
int	**SECOND** Field number for get and set indicating the second within the minute.
int	**WEEK_OF_MONTH** Field number for get and set indicating the week number within the current month.
int	**WEEK_OF_YEAR** Field number for get and set indicating the week number within the current year.
int	**YEAR** Field number for get and set indicating the year

Como todos son atributos **static** hay que acceder poniendo `obj.get(Calendar.PARAM)`. Aparte de los campos evidentes por su nombre, tenemos campos que puede ser útiles, como el día de la semana (Lunes a Domingo), el número de semana del año, el número de semana del mes, etc. Un ejemplo más o menos completo puede ser el siguiente:

```
Locale locale = Locale.getDefault();
// Locale locale = Locale.GERMAN;
Calendar hoy = Calendar.getInstance();
System.out.println(locale);
System.out.println("Año : " + hoy.get(Calendar.YEAR));
System.out.println("Mes (0 es Enero): " + hoy.get(Calendar.MONTH));
System.out.println("Mes (String): "
      + hoy.getDisplayName(Calendar.MONTH, Calendar.SHORT, locale));
System.out.println("Día del Mes : " + hoy.get(Calendar.DAY_OF_MONTH));
System.out.println("Día de la Semana (1 is Domingo): "+ hoy.get(Calendar.DAY_OF_WEEK));
System.out.println("Día de la Semana (String): "
            + hoy.getDisplayName(Calendar.DAY_OF_WEEK, Calendar.LONG, locale));
System.out.println("Semana del Año : " + hoy.get(Calendar.WEEK_OF_YEAR));
System.out.println("Semana del Mes : " + hoy.get(Calendar.WEEK_OF_MONTH));
System.out.println("Día del Año : " + hoy.get(Calendar.DAY_OF_YEAR));
System.out.println("24-hour reloj : " + hoy.get(Calendar.HOUR_OF_DAY));
System.out.println("12-hour reloj : " + hoy.get(Calendar.HOUR));
System.out.println("AM/PM : " + hoy.get(Calendar.AM_PM));
System.out.println("AM/PM : "
      + hoy.getDisplayName(Calendar.AM_PM, Calendar.LONG, locale));
System.out.println("Minutos : " + hoy.get(Calendar.MINUTE));
System.out.println("Segundos : " + hoy.get(Calendar.SECOND));
System.out.println("MiliSegundos : " + hoy.get(Calendar.MILLISECOND));
```

Hay campos que no requieren explicación detallada, como año, día del mes, minuto, etc. Pero para el día de la semana, el mes o si es am/pm que nos devuelven un número entero hay que entender lo que significan. En el día de la semana el 0 corresponde a Domingo y el 6 a Sábado, en el mes el 0 corresponde a Enero y el 11 a Diciembre, en am/pm el 0 corresponde a **am** y el 1 a **pm**. Para obtener un texto más legible y no decirle al usuario, por ejemplo, que está en el mes 0, Calendar tiene un método ***getDisplayName(…)*** que nos devuelve un texto legible para mes, día de la semana o AM/PM. Este método admite **tres** parámetros:

- Primer parámetro: Campo del que queremos la cadena visible, *Calendar.MONTH, Calendar.DAY_OF_WEEK, Calendar.AM_PM,…*
- Si queremos una representación larga o corta. Por ejemplo, para Enero podrían ser sólo tres letras "*Ene*" o bien "*Enero*" con todas sus letras. Para indicar esto debemos pasar como segundo parámetro una de las constantes *Calendar.SHORT* o *Calendar.LONG*
- Tercer parámetro: El *Locale* en el que queremos el texto. Este *Locale* de alguna forma es en qué idioma lo queremos. En nuestro caso, el *Locale* por defecto es español y el mes me devolvería *Enero*, pero si usamos un Locale en Alemán, nos devolvería "*Januar*"

El Locale por defecto del sistema operativo se puede obtener con

```
Locale locale = Locale.getDefault();
```

y para otros idiomas, pueden usarse las constantes definidas en Locale

```
Locale locale = Locale.GERMAN;
```

1.3 Establecer los valores de los atributos de un objeto Calendar

Los atributos de un objeto Calendar se pueden asignar mediante los métodos ***set().*** Los valores que se establezcan con set no se actualizan hasta que tenga que ser utilizado a través de un método ***get()*** o se tenga que calcular con alguna operación a partir de su valor en milisegundos de la fecha, esto se realiza mediante los métodos ***add()*** y ***roll().***

El modo por defecto en el que trabaja Calendar es permisivo (ajusta el efecto de los métodos a una fecha posible), en el modo no-permisivo cuando un método no genera una fecha correcta, lanza una excepción.

Otra característica es que permite manejar la hora en diferentes zonas horarias:

```
Calendar spanishToday = Calendar.getInstance(TimeZone.getDefault());
Calendar canadianToday = Calendar.getInstance(TimeZone.getTimeZone("Canada/Central"));

System.out.println("Hora en España "+spanishToday.get(Calendar.HOUR_OF_DAY));
System.out.println("Hora en Canadá "+canadianToday.get(Calendar.HOUR_OF_DAY));
```

Para obtener un Calendar en una fecha hora concreta tenemos dos opciones. Una de ellas, sabiendo que lo que realmente vamos a obtener es un GregorianCalendar, es hacer un new GregorianCalendar(...) pasando como parámetros el año, mes, día, hora, minuto, segundo. No es necesario pasar todo, ya que GregorianCalendar tiene varios constructores con menos parámetros

```
Calendar mismaFecha = new GregorianCalendar(2010, Calendar.FEBRUARY, 22, 23, 11, 44);
System.out.println("Some Date : " + mismaFecha.getTime());
```

Es importante tener en cuenta un detalle. Para Calendar los meses van de 0 a 11, es decir, 0 es Enero y 11 es Diciembre. Por ello, en el parámetro correspondiente al mes, si queremos meter Febrero, debemos meter un 1, en vez de un 2 que es los que nos dictaría la lógica. Para evitar estas confusiones, siempre es bueno usar las constantes que nos define la clase Calendar, como Calendar.FEBRUARY en el ejemplo.

La otra opción para obtener un Calendar en una fecha/hora concreta es llamar a su método set() para fijar los campos que queramos cambiar. El método set() admite dos parámetros, uno para identificar el campo concreto a cambiar (año, mes, día, hora, minuto, segundo, milésimas de segundo) y un segundo parámetro que sería el valor a poner. El siguiente código muestra bastantes de las posibilidades

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

mismaFecha.set(Calendar.YEAR, 2010);
// Month. 0 is January, 11 is November
mismaFecha.set(Calendar.MONTH, Calendar.AUGUST);
mismaFecha.set(Calendar.DAY_OF_MONTH, 23);

// Either 12-hour clock plus AM/PM
mismaFecha.set(Calendar.HOUR, 10);
mismaFecha.set(Calendar.AM_PM, Calendar.PM);
// or 24-hour clock
mismaFecha.set(Calendar.HOUR_OF_DAY, 22);

mismaFecha.set(Calendar.MINUTE, 36);
mismaFecha.set(Calendar.SECOND, 22);
mismaFecha.set(Calendar.MILLISECOND, 123);

System.out.println("Nueva fecha : " + mismaFecha.getTime());
```

Calendar tiene definidas constantes para todos los nombres de los posibles campos y son estas constantes las que pasamos como primer parámetro. Adviértase que para el mes nuevamente hemos usado las constantes definidas como Calendar.AUGUST, en vez de directamente un número de mes (7 para Agosto), que puede llevar a confusión.

Aparte de los campos evidentes, vemos por ejemplo que la hora se puede fijar de dos formas:

- Pasando una hora de 0 a 11 y pasando el valor AM/PM
- Pasando una hora de 0 a 24.
- HOUR_OF_DAY
- AM_PM + HOUR

No los mostramos en el ejemplo, pero hay más campos, como día de la semana, semana del año, semana del mes, etc. El día podría fijarse con cualquiera de estas combinaciones (ver API de Calendar).

- YEAR + MONTH + DAY_OF_MONTH
- YEAR + MONTH + WEEK_OF_MONTH + DAY_OF_WEEK
- YEAR + MONTH + DAY_OF_WEEK_IN_MONTH + DAY_OF_WEEK
- YEAR + DAY_OF_YEAR
- YEAR + DAY_OF_WEEK + WEEK_OF_YEAR

Fijando los valores de cualquiera de esas combinaciones quedaría perfectamente determinada la fecha. En el ejemplo no hemos metido tantas variantes, nos hemos ido a la más sencilla que es meter YEAR + MONTH + DAY_OF_MONTH

1.4 Días entre dos fechas

Si queremos calcular la diferencia entre dos fechas concretas, desgraciadamente Java no nos ofrece métodos directos para hacerlo.

Calendar tiene un método ***getTimeInMilis()*** que nos devuelve el número de milisegundos que han pasado desde el 1 de Enero de 1970 a las 00:00:00 hasta la fecha/hora representada por nuestra instancia de Calendar. Si tenemos dos objetos fecha/hora como Calendar, la diferencia entre ellas en milisegundos se puede calcular fácilmente:

```
Calendar unDia = Calendar.getInstance();
unDia.set(Calendar.MONTH, Calendar.MARCH);

Calendar otroDia = Calendar.getInstance();
otroDia.set(Calendar.MONTH, Calendar.FEBRUARY);

long milisec = unDia.getTimeInMillis() - otroDia.getTimeInMillis();
```

A partir de aquí es fácil convertir esos milisegundos de diferencia a cualquier otra unidad que nos interese, como número de días, de horas, etc. Por ejemplo, para pasar los milisegundos a días debemos

- dividir por 1000 para pasar los milisegundos a segundos
- después dividir por 60 para pasar los segundos a minutos
- después dividir por 60 para pasar los minutos a horas
- después dividir por 24 para pasar las horas a días

El siguiente código nos daría el número de días entre ambas fechas (por cuestiones de redondeo)

```
long milisec = unDia.getTimeInMillis() - otroDia.getTimeInMillis();
long days = (long) (milisec/(1000*60*60*24.)+0.1);//por redondeo
System.out.println("Días : " + days);
```

1.5 Sumar y restar fechas

Calendar tiene el método ***add(…, …)*** que permite sumar y restar valores a una fecha concreta. Este método admite dos parámetros:

- El campo (año, mes, día, hora, minuto, segundo), identificado por una de las constantes ya conocidas, al que queremos sumar o restar un valor
- Valor a sumar o restar. Si el valor es positivo, se suma, si el valor es negativo, se resta.

Veamos un poco de código sencillo que no requiere explicación

```
Calendar hoy = Calendar.getInstance();
hoy.add(Calendar.DAY_OF_MONTH, 20);
System.out.println("Hoy más 20 días : " + hoy.getTime());

hoy = Calendar.getInstance();
hoy.add(Calendar.DAY_OF_MONTH, -20);
System.out.println("Hoy menos 20 días : " + hoy.getTime());
```

1.6 Comparar fechas

Calendar permite comparar fechas, indicándonos si una es anterior o posterior a otra. Los métodos son ***before()*** y ***after()*** para saber si nuestra fecha es anterior o posterior a otra que tengamos devolviendo un booleano. Adicionalmente, el método ***compareTo()*** devuelve un número negativo, cero o positivo según nuestra fecha sea anterior, igual o posterior a la fecha/hora que se pase por parámetro.

El método compareTo() es útil para ordenar Calendar que estén en un array por medio de clases como Arrays.sort().

Veamos un trozo de código sencillo:

```
Calendar hoy = Calendar.getInstance();
Calendar despues = Calendar.getInstance();
despues.add(Calendar.HOUR_OF_DAY, 2);

Calendar antes = Calendar.getInstance();
antes.add(Calendar.HOUR_OF_DAY, -5);
```

```
System.out.println("Hoy es después de hoy+2horas " + hoy.after(despues));
System.out.println("Hoy es antes de hoy+2horas " + hoy.before(despues));
System.out.println("Hoy es después de hoy-5horas " + hoy.after(antes));
System.out.println("Hoy es antes de hoy-5horas " + hoy.before(antes));
```

En `hoy` obtenemos la fecha/hora actual. Sumamos un par de horas para obtener la fecha/hora posterior en `despues` y restamos 5 días para obtener una fecha/hora anterior a la actual en `antes`. Luego simplemente hacemos llamadas a `hoy.after(…) y hoy.before(…)` sacando el resultado (un boolean) por pantalla. La salida de este código es:

```
Hoy es después de hoy+2horas false
Hoy es antes de hoy+2horas true
Hoy es después de hoy-5horas true
Hoy es antes de hoy-5horas false
```

1.7 Mostrar fechas

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:

```
SimpleDateFormat sdf = 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:

```
System.out.println(sdf.format(c1.getTime()));
```

Letter	Date or Time Component	Presentation	Examples
G	designa la Era	Text	AD
y	Año	Year	1996; 96
Y	Año	Year	2009; 09
M	Mes del Año (context sensitive)	Month	July; Jul; 07
L	Mes del Año (standalone form)	Month	July; Jul; 07
w	Semana del Año	Number	27
W	Semana del Mes	Number	2
D	Día del Año	Number	189
d	Día del Mes	Number	10
F	Día de la Semana en el Mes	Number	2
E	Nombre del día de la semana	Text	Tuesday; Tue
u	Número del día en la semana (1 = Lunes, ..., 7 = Domingo)	Number	1
a	Am/pm marker	Text	PM
H	Hora en el día (0-23)	Number	0
k	Número de hora en el día (1-24)	Number	24
K	Hora según am/pm (0-11)	Number	0

h	Número de hora según am/pm (1-12)	Number	12
m	Minuto	Number	30
s	Segundo	Number	55
S	Millisegundo	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
X	Time zone	ISO 8601 time zone	-08; -0800; -08:00

El método ***getTime()*** devuelve el objeto ***Calendar*** convertido a objeto ***Date***, lo que nos permite una salida legible por pantalla al pasarlo en *System.out.println()*, ese es el motivo por el que lo hemos usado junto con **SimpleDateFormat**.

2 Lista completa de métodos.

Result.	Método	Descripción
Calendar (static)	**getInstance()**	Proporciona la fecha actual del ordenador en un objeto **Calendar**.
int	**get(**Calendar.d**)**	Obtiene el valor del atributo declarado en ***d***
String	**getDisplayName(**Calendar.d, Calendar.long, Locale l**)**	Obtiene el nombre asociado al valor del atributo declarado en ***d***, corto o largo si ***long*** vale SHORT o LONG y en el idioma de ***l***.
void	**set(**Calendar.d, int valor**)**	Cambia el contenido del atributo declarado en ***d*** con el valor de ***valor***.
void	**set(**int y, int m, int d**)**	Cambia el contenido de la fecha, el año por ***y***, el mes por ***m*** y el día por ***d***.
void	**set(**int y, int m, int d, int h, int min, int s**)**	Cambia el contenido de la fecha, el año por ***y***, el mes por ***m*** y el día por ***d,*** la hora por ***h***, los minutos por ***min*** y los segundos por ***s***.
void	**add(**Calendar.d, int valor**)**	Añade al contenido del atributo declarado en ***d*** el valor de ***valor***. Puede que la fecha resultante no sea válida y en ese caso ajusta a una fecha correcta.
boolean	**before(**Calendar data**)**	Devuelve **true** si la fecha que invoca es anterior a data, **false** en caso contrario.
boolean	**after(**Calendar data**)**	Devuelve **true** si la fecha que invoca es posterior a data, **false** en caso contrario.
int	**compareTo(**Calendar data**)**	Compara las dos fechas y devuelve **-1** si la fecha que invoca el método es menor que *data*, devuelve **0** si son iguales y devuelve **1** si la fecha que invoca al método es mayor que *data*.
Date	**getTime()**	Obtiene un objeto de la clase **Date** a partir del objeto **Calendar**. Se utiliza para aplicar el formato de un objeto de la clase **SimpleDateFormat**("formato")
void	**setTime(**Date d**)**	Guarda en el objeto que invoca el método la conversión a Calendar del objeto **d** de tipo Date.
long	**getTimeInMilis()**	Devuelve un long con el número de milisegundos desde el 1 de enero de 1970, negativo si es anterior.
void	**clear()**	Pone sin valor todos los atributos de la fecha
Calendar Object	**(Calendar) x.clone()**	Crea un Objeto a partir de x Calendar
String	**toString()**	Muestra todo el contenido del objeto que invoca el método