Gestión de 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 las clases para el manejo de fechas y tiempo de java.util.

Gestión de Fechas en versiones anteriores a JAVA 8

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.

La clase Date de Java representa una fecha. Pero Date tiene distintas limitaciones (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 crea un objeto de la clase conteniendo la fecha de ese momento. Así es muy típico el uso:

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

Calendar tiene 2 modos de funcionamiento: "lenient" o "non-lenient" mode. Es decir modo permisivo o modo no permisivo.

En modo permisivo es el usado por defecto y esto quiere decir por ejemplo, que si se configura un Calendar con el día 32 de Enero (lo cual sería un error), a la hora de formatear la fecha e imprimirla por pantalla mostrará el 1 de Febrero. Es decir, con lenient mode Java trata de encontrar una fecha si le es posible aunque se haya introducido un dato erróneo.

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

El conjunto de métodos "set" permiten establecer una fecha.

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.

Ejemplos: Dada la siguiente fecha: 31 de Agosto de 2000

a) Sumar 13 meses a la fecha

b) Mostrar el resultado

c) Realizar conclusiones

```
Calendar calendario = Calendar.getInstance();
calendario.set(2000, 7, 31);
calendario.add(Calendar.MONTH, 13);
System.out.println(calendario.getTime().toLocaleString());
// observe que en la sentencia anterior toLocaleString() es deprecated
// conservada por compatibilidad con la versión 1.1 de Java
// ACtualmente una alternativa válida es usar SimpleDateFormat
SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy hh:mm:ss");
System.out.println(sdf.format(calendario.getTime()));
```

Conclusiones: al añadir 13 meses sobre la fecha inicial se salta al mes siguiente del año siguiente, pero sin generar ninguna excepción hemos pasado del día 31 al 30. add fuerza los valores para que no sobrepasen del mínimo o máximo del campo correspondiente, en este caso del campo de días.

Otro ejemplo:

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

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

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

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

System.out.println("Some Date : " + sameDate.getTime());
```

GREGORIANCALENDAR

Esta clase es una subclase 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.

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

Cuyo resultado será:

```
run:
Fecha : Mon Feb 22 23:11:44 ART 2010
BUILD SUCCESSFUL (total time: 0 seconds)
```

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 establecer el mes de Febrero, el valor asignado debe ser 1, en vez de un 2 que es los que nos dictaría la costumbre. Para evitar estas confusiones, siempre es bueno usar las constantes que define la clase Calendar, como Calendar.FEBRUARY usada en el ejemplo precedente.

El siguiente ejemplo muestra el uso de otras constantes:

Leer campos de una fecha:

Observe el siguiente ejemplo:

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

Calendar today = Calendar.getInstance();

System.out.println("Año : " + today.get(Calendar.YEAR));

System.out.println("Mes (0 es Enero y 11 Diciembre): " + today.get(Calendar.MONTH));

System.out.println("Mes (Como Cadena): "+ today.getDisplayName(Calendar.MONTH, Calendar.SHORT, locale));

System.out.println("Dia del Mes : " + today.get(Calendar.DAY_OF_MONTH));

System.out.println("Dia de la Semana (1 es Domingo): "+ today.get(Calendar.DAY_OF_WEEK));

System.out.println("Day de la Semana (Cadena): "+ today.getDisplayName(Calendar.DAY_OF_WEEK, Calendar.LONG,locale));

System.out.println("Semana del Año : " + today.get(Calendar.WEEK_OF_YEAR));

System.out.println("Semana del Mes : " + today.get(Calendar.WEEK_OF_MONTH));

System.out.println("Dia del Año : " + today.get(Calendar.DAY_OF_YEAR));

System.out.println("Hora (24 hs) : " + today.get(Calendar.HOUR_OF_DAY));

System.out.println("Hora (AM y PM) : " + today.get(Calendar.HOUR));

System.out.println("AM/PM : " + today.get(Calendar.AM_PM));

System.out.println("AM/PM : "+ today.getDisplayName(Calendar.AM_PM, Calendar.LONG, locale));

System.out.println("Minutos : " + today.get(Calendar.MINUTE));

System.out.println("Segundos : " + today.get(Calendar.SECOND));

System.out.println("Milisegundos : " + today.get(Calendar.MILLISECOND));
```

Mediante el uso del método get() y utilizando las constantes de Calendar es posible obtener diversos valores de los campos que constituyen la fecha gestionada por el calendario.

Normalmente los campos devolverán en general números. Por ejemplo, el día de la semana, el mes o si es am/pm devuelven un número entero. El 0 corresponde a Domingo y el 6 a Sábado, el 0 corresponde a Enero y el 11 a Diciembre, el 0 corresponde a am y el 1 a pm.

Para obtener un texto más legible, Calendar tiene un método getDisplayName() que devuelve un texto legible para mes, día de la semana o AM/PM. Este método admite tres parámetros

- Campo del que se desea obtener la cadena visible, por ejemplos Calendar.MONTH, Calendar.DAY_OF_WEEK, Calendar.AM_PM, etc
- Representación larga o corta. Por ejemplo, para Enero podrían ser sólo tres letras Ene o bien Enero con todas sus letras. Para indicar esto se debe pasar como segundo parámetro una de la constante Calendar.SHORT o Calendar.LONG.
- El Locale en el que se desea el el texto. El Locale indica el idioma en el cual se desea representar el valor.

Sumar y restar fechas:

Calendar tiene un método add() que permite sumar y restar campos 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 resta un valor
- Valor a sumar o restar. Si el valor es positivo, se suma, si el valor es negativo, se resta.

El siguiente ejemplo muestra las dos operaciones:

```
Calendar today = Calendar.getInstance();
today.add(Calendar.DAY_OF_MONTH, 20);
System.out.println("Fecha al sumar 20 días : " + today.getTime());

today = Calendar.getInstance();
today.add(Calendar.DAY_OF_MONTH, -20);
System.out.println("Fecha al restar 20 días : " + today.getTime());
```

Comparar fechas:

Calendar permite comparar fechas, lo cual permite indicar si una fecha es anterior o posterior a otra fecha. Para ello utiliza los métodos son before() y after(). Adicionalmente, el método compareTo() devuelve un número negativo, cero o positivo si la fecha es anterior, igual o posterior respectivamente a la fecha con la cual se compara.

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

Ejemplo:

```
Calendar today = Calendar.getInstance();
Calendar after = Calendar.getInstance();
after.add(Calendar.HOUR_OF_DAY, 2);

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

System.out.println("Today es mayor que today+2horas: " + today.after(after));
System.out.println("Today es menor que today+2horas " + today.before(after));
System.out.println("Today es mayor que today-5hours " + today.after(before));
System.out.println("Today es menir que today-5hours " + today.before(before));
```

Días entre dos fechas:

Java no ofrece métodos útiles para hacerlo. Se puede realizar una aproximación de la siguiente forma: Calendar posee el método getTimeInMilis() que 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 una instancia de Calendar. Si se tienen dos fecha/hora de tipo Calendar, la diferencia entre ellas en milisegundos se puede calcular fácilmente:

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

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

long milisec = aDay.getTimeInMillis() - otherDay.getTimeInMillis();
long days = milisec / 1000 / 60 / 60 / 24;
System.out.println("Días : " + days);
```

Observe que es posible convertir esos milisegundos de diferencia a cualquier otra unidad que nos interese, como número de días, de horas, etc. En el ejemplo anterior, para pasar los milisegundos a días se procede de la siguiente manera:

- 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

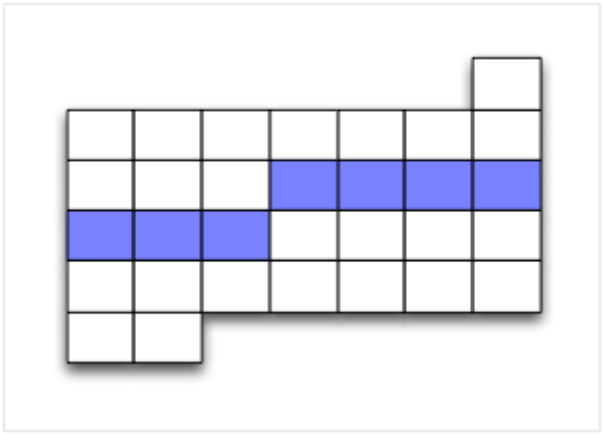
Java 8

Una clase importante y que viene a substituir al Calendario es LocalDate que permite definir fechas y trabajar con ellas de una forma bastante más directa que el Calendario.

```
1  package com.arquitecturajava.fechas;
2
3  import java.time.LocalDate;
4  import java.time.temporal.ChronoField;
5
6  public class PrincipalLocalDate {
7
8   public static void main(String[] args) {
9
10  LocalDate finAño=LocalDate.of(214, 12, 31);
11
12  LocalDate navidad=finAño.minusDays(6);
13  System.out.println(navidad.get(ChronoField.DAY_OF_MONTH));
14
15  }
16 }
```

El programa imprimirá 25 por pantalla.

Otra de las clases que me ha parecido que aporta es la clase **Period que define un intervalo de tiempo entre dos fechas** y nos permite trabajar con ese intervalo de forma sencilla.



Ejemplificamos su uso:

```java
package com.arquitecturajava.fechas;

import java.time.LocalDate;
import java.time.Period;

public class PrincipalPeriodo {

public static void main (String[] args) {

LocalDate fechaA = LocalDate.of(1978, 8, 26);
LocalDate fechaB = LocalDate.of(1988, 9, 28);
Period period = Period.between(fechaA, fechaB);
System.out.printf("Periodo %s y %s"
+ "hay %d años, %d meses"
+ " y %d dias%n", fechaA, fechaB,
period.getYears(),
period.getMonths(),
period.getDays());
}
}
```

Esto nos imprimirá el siguiente mensaje por pantalla:

Periodo 1978-08-26 y 1988-09-28hay 10 años, 1 meses y 2 dias

Con lo cual se agrega esta funcionalidad que no existía con Calendar.