Nuevas clases temporales

A partir de Java 8, se han incluido en el java estándar una serie de clases para la gestión de fechas y horas. Estas clases se encuentran en el paquete java.time y otros subpaquetes contenidos en éste.

Estas nuevas clases ofrecen muchos métodos, no solo para trabajar con fechas y horas, sino también para el manejo de intervalos de tiempo e instantes. El tema del formateado de fechas/horas no se ha dejado de lado, y se ha incorporado también una nueva clase para gestionar este aspecto de forma sencilla y potente.

Para facilitar el estudio, vamos a dividir las clases en dos bloques:

- Clases para manejo de fechas/horas
- Clases para intervalos de tiempo

Comencemos pues con el estudio del primer grupo.

Clases para el manejo de fechas y horas

Antes de presentar las diferentes clases existentes para el manejo de fechas y horas, hay que resaltar el hecho de que, en todos los casos, los objetos creados son inmutables, es decir, una vez creados, sus valores de fecha/hora no pueden ser modificados.

Clase LocalDate

Empezamos hablando de esta clase cuyos objetos representan una fecha concreta. Esta clase, como el resto de las que vamos a presentar en este documento, no dispone de constructores públicos para la creación de objetos, por lo que tendremos que recurrir a métodos estáticos de la propia clase.

En el caso de LocalDate, estos son los métodos estáticos que nos permitirían crear un objeto LocalDate:

- static LocalDate now(). Crea un objeto asociado a la fecha actual del sistema
- static LocalDate of(int year, int month, int day). Crea un objeto asociado a una determinada fecha específica, cuyos valores se pasan como parámetro. Los valores de los meses van de 1 (enero) a 12 (diciembre). Si el valor de alguno de los parámetros está fuera de rango o el dia del mes no se corresponde con el valor indicado, se producirá una DateTimeException:

```
LocalDate ld=LocalDate.of(2017,2,30); //DateTimeException
```

Si mostramos por pantalla un objeto LocalDate se llamará al método toString(), que devuelve la fecha en formato: yyyy-mmm-dd:

```
LocalDate ld=LocalDate.of(2017,12, 30);
Systemout.println(ld);// muestra 2017-12-30
```

Una vez creado el objeto, podemos recurrir a los siguientes métodos para la manipulación de la fecha:

- getYear(), getMonth() y getDayOfMonth(). Devuelven, respectivamente, el año, mes y día del mes asociado a la fecha.
- plusYears(long y), plusMonths(long m) y plusDays(long d). Devuelven una copia de la fecha, correspondiente a los años, meses o días añadidos. Si se sobrepasa el número de días del mes o el número de meses del año, se incrementará el mes/año, según el caso:

  ```
  LocalDate ld=LocalDate.of(2017,12, 30);
  ld=ld.plus(2);
  System.out.println(ld); //muestra 2018-1-1
  ```

  Observa en el código anterior como la fecha original no se ha modificado, **se ha generado una nueva fecha** que se ha almacenado en la variable que apuntaba a la fecha original
- minusYears(long y), minusMonths(long m) y minusDays(long d). Devuelven una copia de la fecha, correspondiente a los años, meses o días restados. Si el resultado de la resta en el caso de los meses o días es inferior a 0, se reajustan los años/meses/dias de la fecha.
- withYear(int y), withMonth(int m) y withDayOfMonth(int d). Devuelven una copia de la fecha con el nuevo año, mes o día establecido, respectivamente. Se produce una DateTimeException si el mes o día del mes establecido no es válido.

Clase LocalTime

Representa una hora en concreto. Para crear un objeto LocalTime, recurriremos a alguno de los siguientes métodos estáticos:

- static LocalTime now(). Devuelve un objeto LocalTime con la hora actual
- static LocalTime of(int hour, int minute, int second). Devuelve un objeto LocalTime con la hora indicada. El valor de hora estará comprendido entre 0 y 23, mientras que los de minutos y segundos deberán estar comprendidos entre 0 y 59. Si alguno de los valores está fuera de rango se producirá un DateTimeException

Si mostramos por pantalla un objeto LocalTime se llamará al método toString(), que devuelve la hora en formato: hh:mm

Tras la creación del objeto, podemos utilizar los siguientes métodos para la manipulación de la hora:

- getHour(), getMinute() y getSecond(). Devuelven, respectivamente, la hora, minutos y segundos.
- plusHours(long h), plusMinutes(long m) y plusSeconds(long s). Devuelven un nuevo objeto LocalTime resultante de la suma de hora, minutos y segundo realizada:

  ```
  LocalTime lt=LocalTime.of(23, 50,40);
  System.out.println(lt.plusSeconds(800)); //muestra 00:04
  ```

- minusHours(long h), minusMinutes(long m) y minusSeconds(long s). Devuelven un nuevo objeto LocalTime resultante de restar le hora, minutos y segundos indicados.
- withHour(int h), withMinute(int m) y withSecond(int s). Devuelven una copia de la hora con la nueva hora, minuto o segundo establecido. Se producirá una excepción DateTimeException si el valor de alguno de los campos está fuera de rango.

Clase LocalDateTime

Representa una combinación de fecha y hora. Para crear un objeto de este tipo utilizaremos los métodos estáticos:

- static LocalDateTime now(). Crea un objeto LocalDateTime asociado a la fecha y hora actuales
- static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute). Crea un objeto LocalDateTime con los datos de fecha y hora proporcionados. Si alguno está fuera de rango, se producirá una DateTimeException

Esta clase incluye también todos los métodos get, plus y with que hemos visto en las dos clases anteriores. Y, aparte de estos, disponemos también de los métodos:

- LocalDate toLocalDate(). Devuelve un objeto LocalDate con la parte de la fecha asociada al objeto.
- LocalTime toLocalTime(). Devuelve un objeto LocalTime con la parte de la hora asociada al objeto.

Clase Instant

Un objeto Instant representa un instante en la línea de tiempo, tomando como referencia las 00:00:00 horas del 1-1-1970.

Existen diversos métodos estáticos para crear un objeto Instant:

- static Instant now(). Crea un objeto asociado al instante de tiempo actual.
- static Instant ofEpochMillis(long milis). Crea un objeto del instante asociado a los milisegundos indicados.
- static Instant ofEpochSecond(long sec). Igual que el anterior, pero indicando los segundos transcurridos desde la fecha de referencia.

Al presentar un objeto de este tipo, el formato devuelto por el método toString() es de la forma: yyyy-mm-ddThh:mm:ssZ:

```
Instant it=Instant.ofEpochSecond(70000);
System.out.println(it); //muestra 1970-01-01T19:26:40Z
```

La clase proporciona los siguientes métodos para la manipulación del instante de tiempo:

- long getEpochSecond(). Devuelve los segundos transcurridos desde la fecha de referencia.

- long toEpochMillis(). Devuelve los milisegundos transcurridos desde la fecha de referencia.
- plusMillis(long m) y plusSeconds(long s). Devuelven un nuevo objeto Instant resultante de añadir los milisegundos o segundos indicados, respectivamente.
- minusMillis(long m) y minusSeconds(long s). Devuelven un nuevo objeto Instant resultante de restar los milisegundos o segundos indicados, respectivamente.

Formateado de fechas

La nueva clase DateTimeFormatter del paquete java.time.format permite aplicar un determinado formato a los objetos LocalDate, LocalTime y LocalDateTime estudiados anteriormente. Las tres clases disponen de un método *format* que tiene la siguiente firma:

String format(DateTimeFormatter format).

A partir del objeto DateTimeFormatter que se proporciona como parámetro devuelven una cadena de caracteres, resultante de aplicar el formato indicado sobre el objeto de fecha/hora correspondiente.

Por tanto, lo primero será crear un objeto DateTimeFormatter, para lo cual utilizaremos alguno de los siguientes métodos estáticos:

- static DateTimeFormatter ofLocalizedDateTime(FormatStyle dateTimeStyle). Devuelve un objeto DateTimeFormatter según el estilo indicado por el objeto FormatStyle. FormatStyle, que se encuentra también en el paquete java.time.format, proporciona una serie de constantes de estilo predefinido, concretamente, las constantes FUL, LONG, MEDIUM y SHORT. La siguiente instrucción crea un objeto DateTimeFormatter para aplicar un formato de fecha hora largo:

  ```
  DateTimeFormatter dtf;
  dtf=DateTimeFormatter.ofLocalizedDateTime(FormatStyle.FULL);
  ```

- ofLocalizedDate(FormatStyle dateStyle) y ofLocalizedTime(FormatStyle timeStyle). Igual que el anterior, aunque solo proporcionan formato para fecha y hora, respectivamente.
- static DateTimeFormatter ofPattern(String pattern). Crea un objeto DateTimeFormatter a partir del patrón de formato indicado como parámetro. En la ayuda oficial de Oracle sobre la clase DateTimeFormatter tienes una tabla con los diferentes caracteres de formato que se pueden utilizar para construir el patrón de formato. Por ejemplo, si quisieramos formatear una fecha/hora de manera que se pueda presentar de la forma dia/mes/año - horas:minutos:segundos, esta sería la instrucción para la creación del objeto:

  ```
  DateTimeFormatter dtf;
  dtf=DateTimeFormatter.ofPattern("dd/MM/YYYY - HH:mm:ss");
  ```

Una vez creado el DateTimeFormatter, si quisiéramos cambiar la localización, deberíamos utilizar el método:

- DateTimeFormatter withLocale(Locale locale). Devuelve una copia del objeto DateTimeFormatter, asociado a la localización proporcionada como parámetro

Creado el DateTimeFormatter podemos aplicarlo para formatear cualquier objeto fecha/hora de los estudiados. El siguiente código presenta la fecha y hora actuales formateadas de la forma indicada en el ejemplo anterior:

```
DateTimeFormatter dtf;
dtf=DateTimeFormatter.ofPattern("dd/MM/YYYY - HH:mm:ss");
LocalDateTime ldt=LocalDateTime.now();
System.out.println(ldt.format(dtf));
```

Clases para intervalos de tiempo

En el paquete java.time encontramos dos clases para manejar intervalos de tiempo, estas son Duration y Period. Vamos a analizar cada una de ellas y el contexto de utilización de las mismas

Clase Period

Un objeto Period representa un intervalo de tiempo basado en fecha, del tipo 2 años 4 meses y 10 días. Como el resto de clases que estamos estudiando, Period no dispone de constructores públicos por lo que para crear un objeto de esta clases recurriremos a los siguientes métodos estáticos:

- static Period of(int years, int months, int days). Crea un objeto Period a partir de los años, meses y días indicados. Los valores pueden ser negativos.
- static Period ofYears(int years). Crea un objeto Period representado por los años indicados.
- static Period ofMonths(int months). Crea un objeto Period representado por los meses indicados.
- static Period ofDays(int days). Crea un objeto Period representado por los días indicados.

En el caso de la clase Period, el método *toString()*, llamado cuando se presenta un objeto por pantalla, devuelve una cadena de caracteres con el periodo de tiempo formateado de la siguiente manera: PañosYmesesMdiasD:

```
Period p=Period.of(3, 20, 5);
System.out.println(p); //muestra P3Y20M5D
```

En cuanto a los métodos que proporciona la clase Period para la manipulación de periodos de tiempo, tenemos los siguientes:

- int getYears(), getMonths() y getDays(). Devuelven los años, meses y dias, respectivamente, asociados al periodo.
- Period plusYears(long years), plusMonths(long months) y plusDays(long days). Devuelven un nuevo objeto Period resultante de la suma del número de años, meses o días indicados.

- Period minusYears(long years), minusMonths(long months) y minusDays(long days). Devuelven un nuevo objeto Period resultante de la resta del número de años, meses o días indicados.
- Period withYears(long years), withMonths(long months) y withDays(long days). Devuelven un nuevo objeto Period resultante de establecer el número de años, meses o días indicados.
- Period normalized(). Devuelve un nuevo objeto Period con los años y meses normalizados, es decir, reajustando los valores de años y meses de manera que el valor de estos últimos no sea superior a 11. Para comprender su funcionamiento, veamos el siguiente código de ejemplo:

```
Period p=Period.ofYears(10);
p=p.plusMonths(20);
System.out.println(p); //sin normalizar muestra P10Y20M
System.out.println(p.normalized()); //normalizado muestra P11Y8M
```

Clase Duration

Representa un intervalo temporal medido con unidades horarias, al estilo 2 horas 20 minutos 40 segundos.

Para crear un objeto Duration emplearemos los siguientes métodos estáticos de la clase:

- static Duration ofDays(long days). Crea un objeto Duration basado en la cantidad de días indicado, aunque internamente, cada día es considerado como 24 horas.
- static Duration ofHours(long hours). Crea un objeto Duration basado en la cantidad de horas indicada.
- static Duration ofMinutes(long minutes). Crea un objeto Duration basado en la cantidad de minutos indicado.
- static Duration ofSeconds(long seconds). Crea un objeto Duration basado en la cantidad de segundos indicado.

El método toString() de Duration devuelve una representación del objeto en la forma: PThorasHminutosMsegundosS:

```
Duration d=Duration.ofDays(2);
System.out.println(d); //muestra PT48H
```

Los siguientes métodos de la clase Duration nos permitirán manipular los datos del objeto:

- long getSeconds(). Devuelve la duración del intervalo en segundos
- long toDays(), toHours() y toMinutes(). Devuelve la duración del intervalo en días, horas o minutos, respectivamente.

```
Duration d=Duration.ofDays(2);
System.out.println(d.getSeconds());//muestra 172800
System.out.println(d.toDays()); //muestra 2
System.out.println(d.toHours()); //muestra 48
System.out.println(d.toMinutes()); //muestra 2880
```

- Duration plusDays(long days), plusHours(long hours), plusMinutes(long minutes) y plusSeconds(long seconds). Devuelven una copia del objeto Duration con los días, horas, minutos o segundos añadidos.
- Duration minusDays(long days), minusHours(long hours), minusMinutes(long minutes) y minusSeconds(long seconds). Devuelven una copia del objeto Duration con los días, horas, minutos o segundos restados
- Duration withSeconds(long seconds). Devuelve una copia del objeto Duration con la cantidad de segundos establecidos:

```
Duration d=Duration.ofDays(2);
System.out.println(d.withSeconds(100)); //muestra PT1M40S
```