Paquete java.time de Java 8: Fechas y horas

Java 8 introduce una nueva API para fechas y horas que es thread-safe, más fácil de leer y más amplia que la API anterior.

Esta API se introduce para cubrir los siguientes inconvenientes de la vieja API de fecha y hora:

1. java.util.Date no es thread-safe por lo que los desarrolladores tienen que hacer frente a problemas de concurrencia durante el uso de la fecha. La nueva API de fecha y hora es inmutable y no tiene métodos setter.
2. Dificultad para manejar zona-horaria.

En java 8 se crea un nuevo paquete para el manejo de fechas, se trata del paquete java.time. Este paquete es una extensión a las clases java.util.Date y java.util.Calendar que vemos un poco limitado para manejo de fechas, horas y localización.

Las clases definidas en este paquete representan los principales conceptos de fecha - hora, incluyendo instantes, fechas, horas, períodos, zonas de tiempo, etc. Están basados en el sistema de calendario ISO, el cual es el calendario mundial *de-facto* que sigue las reglas del calendario Gregoriano.

Enumerados de mes y de día de la semana

Existe un enum donde se definen todos los días de la semana. Este enum se llama java.time.DayOfWeek y contiene algunos métodos interesantes que permiten manipular días hacia adelante y hacia atrás:

```
DayOfWeek lunes = DayOfWeek.MONDAY;
System.out.printf("8 días será: %s%n",lunes.plus(8));
System.out.printf("2 días antes fue: %s%n",lunes.minus(2));
```

Además, con el método getDisplayName() se puede acceder al texto que corresponde a la fecha, dependiendo del Locale actual, o el que definamos.

```
DayOfWeek lunes = DayOfWeek.MONDAY;
Locale l = new Locale("es","ES");
System.out.printf("TextStyle.FULL:%s%n",lunes.getDisplayName(TextStyle.FULL,l));
System.out.printf("TextStyle.NARROW:%s%n",lunes.getDisplayName(TextStyle.NARROW,l));
System.out.printf("TextStyle.SHORT:%s%n",lunes.getDisplayName(TextStyle.SHORT,l));
```

Para los meses, existe el enum java.time.Month que básicamente hace lo mismo:

```
Locale l = new Locale("pt"); //probamos con portugues
Month mes = Month.MARCH;
System.out.printf("Dos meses más y será: %s%n", mes.plus(2));
System.out.printf("Hace 1 mes fué: %s%n", mes.minus(1));
System.out.printf("Este mes tiene %s días %n ", mes.maxLength());
System.out.printf("TextStyle.FULL:%s%n",mes.getDisplayName(TextStyle.FULL, l));
System.out.printf("TextStyle.NARROW:%s%n",mes.getDisplayName(TextStyle.NARROW, l));
System.out.printf("TextStyle.SHORT:%s%n",mes.getDisplayName(TextStyle.SHORT, l));
```

Clases de fecha

Las clases de fecha como el java.time.LocalDate manejan la fecha, pero, a diferencia del java.util.Date, sólo trabaja fecha, y no hora. Esto nos permitirá manipular la fecha para registrar fechas específicas

como el día de cumpleaños o de matrimonio. Aquí unos ejemplos:

```
LocalDate date = LocalDate.of(1999, Month.AUGUST, 23);
DayOfWeek dia=date.getDayOfWeek();
System.out.printf("El día que conocí a quien es mi esposa fue el %s y fue un %s
%n",date,dia);

LocalDate fechaActual = LocalDate.now();
System.out.printf("La fecha actual es %s, mes: %s, año:%s%n",
      fechaActual,fechaActual.getMonth(),fechaActual.getYear());
System.out.printf("Hace una semana fue %s%n",fechaActual.minusDays(7));
System.out.printf("Dentro de 2 meses será %s%n%n",fechaActual.plusMonths(2));
```

Para representar el mes de un año específico, usamos la clase java.time.YearMonth y también podemos obtener la cantidad de días de ese mes, sobre todo cuando jugamos con los bisiestos.

```
YearMonth mes = YearMonth.now();
System.out.printf("Este mes es %s y tiene %d días%n", mes,  mes.lengthOfMonth());
mes = YearMonth.of(2004, Month.FEBRUARY);
System.out.printf("El mes %s tuvo %d días,%n", mes, mes.lengthOfMonth());
mes = YearMonth.of(2002, Month.FEBRUARY);
System.out.printf("el mes %s tuvo %d días,%n", mes, mes.lengthOfMonth());
mes = YearMonth.of(2000, Month.FEBRUARY);
System.out.printf("el mes %s tuvo %d días%n", mes, mes.lengthOfMonth());
mes = YearMonth.of(1800, Month.FEBRUARY);
```

La clase java.time.MonthDay representa a un día de un mes en particular.

```
MonthDay dia=MonthDay.of(Month.FEBRUARY, 29);
System.out.printf("El día %s %s es válido para el año
            2010%n",dia,dia.isValidYear(2010)?"":"no"); //la respuesta será NO
```

Y la clase java.util.Year nos permite manipular y conocer sobre un año en específico, sin importar el día o mes.

```
Year año = Year.now();
System.out.printf("Este año es %s y %s es bisiesto%n",año,año.isLeap()?"sí":"no");
```

Clase de Hora

La clase java.time.LocalTime es similar a las otras cosas que comienza con el prefijo Local, pero se centra únicamente en la hora. Esta clase es muy útil para representar horas y tiempos de un día, tales como la hora de inicio de una película o el horario de atención de una biblioteca. Se centra únicamente en la hora de un día cualquiera, pero no en una fecha específica. Con el java.util.Date solo podemos manipular la hora de un día de un año en especial, de una zona de horario en especial, pero con el LocalTime solo nos centramos en la hora en sí, sin importar que día sea.

Aquí un pequeño ejemplo de su uso:

```
LocalTime justoAhora = LocalTime.now();
System.out.printf(“En este momento son las %d horas con %d minutos y %d segundos\n”
      ,justoAhora.getHour(),justoAhora.getMinute(),justoAhora.getSecond());

enOchoHoras = justoAhora.plusHours(8);
System.out.printf("Dentro de ocho horas serán las %d horas con %d minutos y %d
segundos\n",enOchoHoras.getHour(),enOchoHoras.getMinute(),enOchoHoras.getSecond());
```

Como se puede ver, no tiene nada que ver la fecha, solo se manipuló la hora.

La clase de hora/fecha

La clase java.time.LocalDateTime manipula la fecha y la hora sin importar la zona horaria. Esta clase es usada para representar la fecha (año, mes, día) junto con la hora (hora, minuto, segundo, nanosegundo) y es - en efecto - la combinación de LocalDate y LocalTime. Esta clase puede ser usada para especificar un evento, tal como la final de Champions League 2014 en la hora local del evento.

Además del método now que viene en cada clase vista hasta ahora, la clase LocalDateTime tiene varios métodos of (métodos con prefijo of) que crean una instancia de LocalDateTime. También hay un método from que convierte una instancia de otro formato de tiempo a la instancia LocalDateTime. También hay métodos para agregar y quitar horas, minutos, días, semanas y meses. He aquí algunos ejemplos:

```
LocalDateTime ahora = LocalDateTime.now();
System.out.printf("La hora es: %s%n", ahora);
LocalDateTime algunDia = LocalDateTime.of(1976, Month.MARCH, 27, 6, 10);
System.out.printf("Yo nací el %s%n", algunDia);
System.out.printf("Hace seis meses fue %s%n", LocalDateTime.now().minusMonths(6));
```

La clase para dar formato a las fechas

La clase java.time.DateTimeFormatter se utiliza para dar formato a las fechas. He aquí algunas de las letras que se usan como patrones:

- **y**, nos permite acceder al año en formato de cuatro o dos dígitos (2014 o 14).
- **D**, nos permite obtener el número de día del año (225).
- **d,** al contrario del anterior nos devuelve el número del día del mes en cuestión (27).
- **L**, nos ayuda a obtener el mes del año en forma numérica, **M** nos da el mes en texto.
- **H**, nos da la hora.
- **s**, nos da los segundos.
- **m**, nos permite obtener los minutos.
- **a**, nos da el am o pm de la hora.
- **z,** nos permite acceder al nombre de la zona horaria.

Ejemplos:

```
DateTimeFormatter fechaFormateada = DateTimeFormatter.ofPattern("yyyy/LL/dd");
System.out.println(fechaFormateada.format(LocalDate.now()));
System.out.println(LocalDate.parse("2014/11/15", fechaFormateada));

DateTimeFormatter zonaHoraria = DateTimeFormatter.ofPattern("d MMMM, yyyy h:mm a");
System.out.println(ZonedDateTime.now().format(zonaHoraria));

String fechaInicial = “1906-12-31”;
LocalDate fechaTomada = LocalDate.parse(fechaInicial);
System.out.printf("Fecha: %s%n",fechaTomada);
String fechaInicialHora = "1906-12-31T12:05";
LocalDateTime fechaHoraTomada = LocalDateTime.parse(fechaInicialHora);
System.out.printf("Fecha/Hora: %s%n",fechaHoraTomada);
DateTimeFormatter df = DateTimeFormatter.ofPattern("dd MMM yyyy");
System.out.printf("%s con nuevo formato: %s",fechaTomada,df.format(fechaTomada));
```

La clase para períodos de tiempo

La clase java.time.Period define un intervalo de tiempo entre dos fechas y nos permite trabajar con él de forma sencilla.

```
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());
```