**Paquete java.time de Java 8: Fechas y horas**

*Por Diego Silva*  
Publicado en Diciembre 2014

En este artículo veremos sobre un nuevo paquete que existe en Java 8, se trata del paquete [java.time](http://docs.oracle.com/javase/8/docs/api/java/time/package-frame.html). 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, periodos, zonas de tiempo, etc. Están basados en el sistema de calendario ISO, el cual 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. Lo cual tiene sentido hacerlo enum porque siempre habrán siete días de la semana :). Este enum se llama [java.time.DayOfWeek](http://docs.oracle.com/javase/8/docs/api/java/time/DayOfWeek.html)

DayOfWeek lunes = DayOfWeek.MONDAY;

Este enum tiene algunos métodos interesantes que permite manipular días hacía 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. Para mi país probé con esto:

DayOfWeek lunes = DayOfWeek.MONDAY;

Locale l = new Locale("es","PE");

System.out.println("TextStyle.FULL:" + lunes.getDisplayName(TextStyle.FULL, l));

System.out.println("TextStyle.NARROW:" + lunes.getDisplayName(TextStyle.NARROW, l));

System.out.println("TextStyle.SHORT:" + lunes.getDisplayName(TextStyle.SHORT, l));

Deberían probar con Locale l = Locale.KOREA, es muy interesante lo que sale.

Para los meses, existe el enum [java.time.Month](http://docs.oracle.com/javase/8/docs/api/java/time/Month.html) 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));

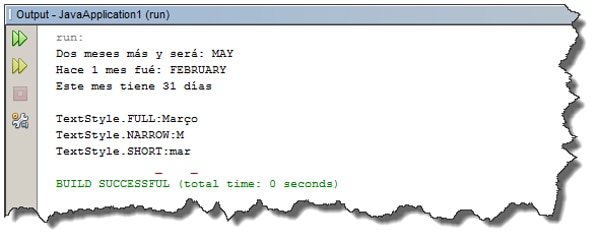
System.out.printf("Este mes tiene %s días %n ", mes.maxLength());

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));

.. y el resultado sería así:

  
**Clases de fecha**   
Las clases de fecha como el [java.time.LocalDate](http://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html) manejan la fecha, pero, a diferencia del java.util.Date, es que es solo 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);

Para representar el mes de un año específico, usamos la clase [java.time.YearMonth](http://docs.oracle.com/javase/8/docs/api/java/time/YearMonth.html) 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);

System.out.printf("Pero el mes %s tuvo %d días ¿Sabías que no es considerado bisiesto?%n",

mes, mes.lengthOfMonth());

La clase [java.time.MonthDay](http://docs.oracle.com/javase/8/docs/api/java/time/MonthDay.html) representa a un día de un mes en particular, tal como decir que el año nuevo es el 1 de enero.

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](http://docs.oracle.com/javase/8/docs/api/java/time/Year.html) 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](http://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html) 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 ¿Se entiende? 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. Normalmente lo manipularíamos con una cadena y de ahí hacemos raros algoritmos para saber si esa cadena está dentro de la hora actual.

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());

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](http://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html) 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 (o 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. Aquí muestro 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));

**Conclusión....**   
Ahora las tareas sobre manipulación de fechas que dejan los profesores serán mucho más fáciles de resolver :)

**Bibliografía**   
Me he basado este post de este tutorial [Standard Calendar](http://docs.oracle.com/javase/tutorial/datetime/iso/index.html), y podemos encontrar más sobre este API en el API de Java [java.time](http://docs.oracle.com/javase/8/docs/api/java/time/package-frame.html)

**Diego Silva.** En Noviembre de 2013 obtuvo el Oracle Certified Associate Java SE 7 Programmer.  
Ha Trabajado para el Estado peruano, desarrollando y llevando a producción sistemas de información en vía web. Actualmente trabaja en una empresa privada desarrollando aplicaciones Java Web con Oracle ADF y con la plataforma Liferay.

*Este artículo ha sido revisado por el equipo de productos Oracle y se encuentra en cumplimiento de las normas y prácticas para el uso de los productos Oracle.*