Anexo 3: Fechas en Java

Contenido

1. Un ejemplo rápido

Veamos un ejemplo de cómo trabajar con fechas y horas utilizando la API Date-Time de Java:

- Una instancia de la clase LocalDate representa una fecha local sin hora;
- Una instancia de la clase LocalTime representa una hora local sin fecha;
- Una instancia de la clase LocalDateTime representa una fecha y hora locales;
- una instancia de la clase ZonedDateTime representa una fecha y hora con una zona horaria.

Una LocalDate y una LocalTime también se denominan parciales, ya que no representan un instante en la línea de tiempo; tampoco son conscientes de los cambios en el horario de verano.

Una ZonedDateTime representa un momento en el tiempo en una zona horaria determinada que se puede convertir a un instante en la línea de tiempo; es consciente de Daylight Savings de verano.

Por ejemplo, si se añaden 4 horas a una LocalTime de la 1:00 AM se obtendrá otra LocalTime de las 5:00 AM independientemente de la fecha y el lugar. Sin embargo, si añade 4 horas a una ZonedDateTime que representa la 1:00 AM del 9 de marzo de 2014, en la zona horaria de Chicago/America, te dará las 6:00 AM del 9 de marzo de 2014, en la misma zona horaria, ya que el reloj se adelanta 1 hora a las 2:00 AM de ese día debido al horario de verano.

Por ejemplo, las aplicaciones de líneas aéreas utilizarían instancias de la clase ZonedDateTime para almacenar la hora de salida y la hora de llegada de los vuelos.

En la API Date-Time, las clases que representan fecha, hora y datetime tienen un método now() que devuelve la fecha, hora o datetime actual, respectivamente.

El siguiente fragmento de código crea objetos datetime que representan una fecha, una hora y una combinación de ellas con y sin zona horaria:

```
LocalDate dateOnly = LocalDate.now();
LocalTime timeOnly = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();
ZonedDateTime dateTimeWithZone = ZonedDateTime.now();
```

LocalDate no tiene en cuenta la zona horaria. Será interpretado de manera diferente en diferentes zonas horarias en el mismo momento del tiempo.

Un objeto LocalDate se utiliza para almacenar un valor de fecha cuando la hora y la zona horaria no son importantes para dar un significado a la fecha, como una fecha de nacimiento, la fecha de publicación de un libro, etc.

Puede especificar los componentes de un objeto datetime utilizando el método de fábrica estático of(). El siguiente fragmento de código crea un LocalDate especificando los componentes de año, mes y día de una fecha:

```
// Crear un LocalDate que represente el 12 de enero de 1968
LocalDate miFechaNacimiento = LocalDate.of(1968, ENERO, 12);
```

Un LocalDate almacena un valor de sólo fecha sin hora ni zona horaria. Cuando se obtiene una LocalDate utilizando el método estático now(), se utiliza la zona horaria por defecto del sistema para obtener el valor de la fecha.

El listado 16-2 muestra cómo obtener la fecha, hora, datetime y datetime actuales con la zona horaria. También muestra cómo construir una fecha a partir del año, el mes del año y el día del mes. Puede obtener una salida diferente, ya que imprime los valores actuales de fecha y hora.

```
// Obtención de la fecha, hora y fecha-hora actuales
// y construcción de una fecha
//
// CurrentDateTime.java
package com.jdojo.datetime;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.ZonedDateTime;
import static java.time.Month.JANUARY;

public class CurrentDateTime {
      public static void main(String[] args) {
            // Get current date, time, and datetime
            LocalDate dateOnly = LocalDate.now();
            LocalTime timeOnly = LocalTime.now();
            LocalDateTime dateTime = LocalDateTime.now();
            ZonedDateTime dateTimeWithZone = ZonedDateTime.now();
            System.out.println("Current Date: " + dateOnly);
            System.out.println("Current Time: " + timeOnly);
            System.out.println("Current Date and Time: " + dateTime);
            System.out.println("Current Date, Time, and Zone: " +
            dateTimeWithZone);
            // Construct a birth date and time from datetime components
            LocalDate myBirthDate = LocalDate.of(1968, JANUARY, 12);
            LocalTime myBirthTime = LocalTime.of(7, 30);
            System.out.println("My Birth Date: " + myBirthDate);
            System.out.println("My Birth Time: " + myBirthTime);
      }
}
```

El programa utiliza cuatro clases para obtener una fecha local, una hora, una fecha-hora y una fecha-hora con una zona horaria. En la API Date-Time heredada, podría haber obtenido un resultado similar utilizando sólo la clase Calendar.

La API Date-Time es muy completa. Abarca unas 80 clases y unos 1.000 métodos. Permite representar y manipular fechas y horas utilizando diferentes escalas y diferentes sistemas de calendario. Existen varias normas locales y una norma universal (ISO-8601) se han utilizado para el cronometraje. Para aprovechar la API Date-Time, es necesario comprender la historia del cronometraje. Las siguientes secciones le ofrecen un breve resumen de las distintas formas de medir el tiempo mediante sistemas de calendario y las normas de fecha y hora ISO-8601. Si conoce bien estos temas, puede saltarse estas secciones y continuar desde la sección "Explorar la API Fecha-Hora".

1.1. Explorar la API Date-Time

Al principio, explorar la API Date-Time resulta intimidante, ya que contiene muchas clases con numerosos métodos.

Aprender la convención de nomenclatura de los métodos ayudará enormemente a comprender la API.

La API se ha diseñado cuidadosamente para que los nombres de las clases y sus métodos sean coherentes e intuitivos.

Los métodos que comienzan con el mismo prefijo realizan tareas similares. Por ejemplo, un método of() de una clase se utiliza como método de fábrica estático para crear un objeto de esa clase.

Todas las clases, interfaces y enums de la API Date-Time se encuentran en el paquete java.time y en cuatro de sus subpaquetes como se indica en la siguiente tabla:

Package	Description
`java.time`	Contains frequently used classes. The `LocalDate`, `LocalTime`, `LocalDateTime`, `ZonedDateTime`, `Period`, `Duration`, and `Instant` classes are in this package. Classes in this package are based on ISO standards.
`java.time.chrono`	Contains classes supporting non-ISO calendar systems, for example, Hijrah calendar, Thai Buddhist calendar, etc.
`java.time.format`	Contains classes for formatting and parsing dates and times.
`java.time.temporal`	Contains classes for accessing components of dates and times. It also contains classes that act like datetime adjusters.
`java.time.zone`	Contains classes supporting time zones and zone rules.

En las siguientes secciones se explican los prefijos utilizados en los nombres de métodos de la API Fecha-Hora con sus significados y ejemplos.

1.1.1. Los métodos ofXxx()

Las clases de la API Date-Time no proporcionan constructores públicos para crear sus objetos. Permiten crear objetos mediante métodos de fábrica (factory) estáticos denominados "of" o "ofXxx" (donde Xxx se sustituye por una descripción de los parámetros). El siguiente fragmento de código muestra cómo crear objetos de la clase LocalDate:

```
LocalDate ld1 = LocalDate.of(2021, 5, 2); // 2021-05-02
LocalDate ld2 = LocalDate.of(2021, Month.JULY, 4); // 2021-07-04
LocalDate ld3 = LocalDate.ofEpochDay(2002); // 1975-06-26
LocalDate ld4 = LocalDate.ofYearDay(2014, 40); // 2014-02-09
```

1.1.2. Los métodos from()

Un método from() es un método estático de fábrica, similar a un método of(), que se utiliza para derivar un objeto datetime a partir del argumento especificado. A diferencia de un método of(), un método from() requiere conversión de datos en el argumento especificado.

Para entender lo que hace un método from(), piense en su nombre como un método deriveFrom(). Mediante un método from(), se obtiene un nuevo objeto datetime a partir del argumento especificado. El siguiente fragmento de código muestra cómo derivar un LocalDate de un LocalDateTime:

```
LocalDateTime ldt = LocalDateTime.of(2021, 5, 2, 15, 30); // 2021-05-02T15:30
LocalDate ld = LocalDate.from(ldt); // 2021-05-02
```

1.1.3. Los métodos withXxx()

La mayoría de las clases de la API son inmutables. No tienen métodos setXxx(). Si desea cambiar un campo de un objeto datetime, por ejemplo, el valor del año en una fecha necesita buscar un método con prefijo "with". Un método withXxx() devuelve una copia del objeto con el campo especificado cambiado.

Suponga que tiene un objeto LocalDate y desea cambiar su año. Para ello debe utilizar el método withYear(int newYear) de la clase LocalDate. El siguiente fragmento de código muestra cómo obtener una LocalDate a partir de otra LocalDate con el año cambiado:

```
LocalDate ld1 = LocalDate.of(2021, Month.MAY, 2); // 2021-05-02
LocalDate ld2 = ld1.withYear(2014); // 2014-05-02
```

Puede obtener una nueva LocalDate a partir de una LocalDate existente cambiando varios campos encadenando las llamadas al método withXxx(). El siguiente fragmento de código crea una nueva LocalDate a partir de una LocalDate existente cambiando el año y el mes:

LocalDate ld3 = LocalDate.of(2021, 5, 2); // 2021-05-02

LocalDate ld4 = ld3.withYear(2024).withMonth(7); // 2024-07-02

1.1.4. Los métodos getXxx()

Un método getXxx() devuelve el elemento especificado del objeto. Por ejemplo, el método getYear() de la clase LocalDate devuelve la parte del año de la fecha. El siguiente fragmento de código muestra cómo obtener el año, mes y día de un objeto LocalDate:

```
LocalDate ld = LocalDate.of(2021, 5, 2);
int año = ld.getYear();             // 2021
Month mes = ld.getMes();            // Month.MAYO
int día = ld.getDayOfMonth();       // 2
```

1.1.5. Los métodos toXxx()

Un método toXxx() convierte un objeto en un tipo Xxx relacionado. Por ejemplo, el método toLocalDate() de la clase LocalDateTime devuelve un objeto LocalDate con la fecha del objeto LocalDateTime original. Estos son algunos ejemplos de uso de los métodos toXxx():

```
LocalDate ld = LocalDate.of(2021, 8, 29); // 2021-08-29
// Convierte la fecha a días de época. Los días de época son el número de
// días transcurridos desde 1970-01-01 a una fecha. Una fecha anterior a
// 1970-01-01 devuelve un entero negativo.
long epochDays = ld.toEpochDay();         // 18868
// Convierte una LocalDateTime en una LocalTime utilizando el método
// toLocalTime()
LocalDateTime ldt = LocalDateTime.of(2021, 8, 29, 16, 30);
LocalTime lt = ldt.toLocalTime();         // 16:30
```

1.1.6. Los métodos atXxx()

Un método atXxx() permite crear un nuevo objeto datetime a partir de un objeto datetime existente, proporcionando alguna información adicional. Contraste el uso de un método atXxx() con el de un método withXxx(); el primero permite crear un nuevo tipo de objeto proporcionando información adicional, mientras que el segundo permite crear una copia de un objeto modificando sus campos.

Suponga que tiene la fecha 2021-05-02. Si desea crear una nueva fecha de 2021-07-02 (con el mes cambiado a 7), usaría el método withXxx(). Si quiere crear una fecha de 2021-05-02T15:30 (añadiendo la hora 15:30), utilizaría un método atXxx(). Algunos ejemplos de uso de métodos atXxx() son:

```
LocalDate ld = LocalDate.of(2021, 5, 2);  // 2021-05-02
LocalDateTime ldt1 = ld.atStartOfDay();   // 2021-05-02T00:00
LocalDateTime ldt2 = ld.atTime(15, 30);   // 2021-05-02T15:30
```

Los métodos atXxx() admiten el patrón constructor. El siguiente fragmento de código muestra cómo utilizar un patrón constructor para construir una fecha local:

```
// Usar un patrón constructor para construir una fecha 2021-05-22
LocalDate d1 = Year.of(2021).atMonth(5).atDay(22);
// Utiliza un método de fábrica of() para construir una fecha 2021-05-22
LocalDate d2 = LocalDate.of(2021, 5, 22);
```

1.1.7. Los métodos plusXxx() y minusXxx()

Un método plusXxx() devuelve una copia de un objeto añadiendo un valor especificado. Por ejemplo, el método plusDays(long days) de la clase LocalDate devuelve una copia del objeto LocalDate añadiendo el número de días especificado.

Un método minusXxx() devuelve una copia de un objeto restándole un valor especificado. Por ejemplo, el método minusDays(long days) de la clase LocalDate devuelve una copia del objeto LocalDate restándole el número de días especificado:

```
LocalDate ld = LocalDate.of(2021, 5, 2); // 2021-05-02
LocalDate ld1 = ld.plusDays(5); // 2021-05-07
FechaLocal ld2 = ld.plusMonths(3); // 2021-08-02
FechaLocal ld3 = ld.plusWeeks(3); // 2021-05-23
FechaLocal ld4 = ld.minusMonths(7); // 2011-10-02
FechaLocal ld5 = ld.menosSemanas(3); // 2021-04-11
```

1.1.8. Métodos multipliedBy(), dividedBy() y negated()

La multiplicación, la división y la negación no tienen sentido en fechas y horas. Son aplicables a los tipos datetime que denotan una cantidad de tiempo como Duration y Period. Las duraciones y los periodos pueden sumarse y restarse. La API admite duraciones y periodos negativos:

```
Duración d = Duration.ofSeconds(200); // PT3M20S (3 minutos y 20 segundos)
Duración d1 = d.multiplicadoPor(2); // PT6M40S (6 minutos y 40 segundos)
Duración d2 = d.negated(); // PT-3M-20S (-3 minutos y -20 segundos)
```