Anexo. Tipos para trabajar con fechas, horas y duraciones

Autor: Antonia M. Reina
Revisión:

1. Introducción

En Java 8 se define un nuevo conjunto de tipos para trabajar con fechas, horas, instantes temporales y duraciones, que se han incluido en el paquete `java.time`. Los tipos nuevos, más interesantes desde el punto de vista de la asignatura son los siguientes:

- `LocalDate`, tipo inmutable para representar fechas (sin zona horaria), es decir, solo con día, mes y año. Por ejemplo, 03-12-2015
- `LocalTime`, tipo inmutable para representar horas sin fecha (ni zona horaria), solo con hora, minutos, segundos y nanosegundos (si son necesarios). Por ejemplo, 10:10:12.99.
- `LocalDateTime`, tipo inmutable para representar fechas con horas (pero sin zona horaria), con día, mes, año, horas, minutos, segundos y nanosegundos. Por ejemplo, 03-12-2015 10:10:12.99.
- `Duration`, tipo inmutable que representa una cantidad temporal que se mueve en el rango de nanosegundos, segundos, minutos, horas o días.
- `Period`, tipo inmutable que representa una cantidad temporal que se mueve en el rango de años, meses o días.

En el resto de secciones del anexo, se verán el conjunto de métodos más interesantes desde el punto de vista de la asignatura.

2. Creación de fechas y horas con métodos de factoría

Para crear objetos de tipo `LocalDate`, `LocalTime` y/o `LocalDateTime` la API de Java proporciona métodos de factoría. Para trabajar de manera uniforme, los tres tipos proporcionan distintas versiones del método of para crear objetos:

TIPO	MÉTODOS DE FACTORÍA
LocalDate	static LocalDate of(int year, int month, int dayOfMonth) static LocalDate of(int year, Month month, int dayOfMonth)
LocalTime	static LocalTime of(int hour, int minute) static LocalTime of(int hour, int minute, int second) static LocalTime of(int hour, int minute, int second, int nanoOfSecond)
LocalDateTime	static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute) static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second) static LocalDateTime of(int year, int month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond) static LocalDateTime of(int year, Month month, int dayOfMonth, int hour, int minute) static LocalDateTime of(int year, Month month, int dayOfMonth, int hour, int minute, int second) static LocalDateTime of(int year, Month month, int dayOfMonth, int hour, int minute, int second, int nanoOfSecond) static LocalDateTime of(LocalDate date, LocalTime time)

Así `LocalDate`, tiene dos versiones del método `of`: una que toma el año, el mes y el día; y otra que toma el año, el mes (pero usando el enumerado `Month`[1]), y el día. Note que el orden en el que aparecen los parámetros está relacionado con el orden en que se escriben las fechas en inglés.

Algunos ejemplos de uso de estos métodos son:

```
LocalDate fecha2 = LocalDate.of(2014, Month.MAY,23);
LocalDate fecha3 = LocalDate.of(2014, 5,23);

LocalTime hora2 = LocalTime.of(11,00);
LocalTime hora3 = LocalTime.of(10,59,59);

LocalDateTime fechaHora2 = LocalDateTime.of(2014,Month.MAY,23,11,00);
LocalDateTime fechaHora3 = LocalDateTime.of(2014, Month.MAY,23,10, 59, 59);
LocalDateTime fechaHora4 = LocalDateTime.of(2014,5,23,10, 59, 59);
LocalDateTime fechaHora5 = LocalDateTime.of(fecha3, hora3);
```

[1] https://docs.oracle.com/javase/8/docs/api/java/time/Month.html

Además del método `of`, los tres tipos proporcionan el método de factoría `now`, que permite crear un objeto de tipo `LocalDate`, `LocalTime` o `LocalDateTime` con la fecha, hora o fecha y hora del sistema en el momento en que se invoca al método. Por ejemplo,

```
LocalDate fecha1 = LocalDate.now();
LocalTime fecha1 = LocalTime.now();
LocalDateTime fechahora1 = LocalDateTime.now();
```

3. Creación de cantidades temporales con métodos de factoría

Para crear objetos que representan una cantidad de tiempo se usan métodos de factoría de las clases `Duration` o `Period`.

Si queremos representar una cantidad temporal que se mueva en el rango de días, horas, minutos, segundos milisegundos o nanosegundos usaremos el tipo `Duration`. El rango de valores que puede tener una duración es demasiado grande para almacenarlo en un número, así que, internamente, los objetos de tipo `Duration` usan dos atributos, uno de tipo `long`, que representa los segundos y otro de tipo `int` para los nanosegundos. El tipo `Duration` proporciona métodos específicos para crear duraciones a partir de una cantidad que representa un número de días (`ofDays`), o a partir de un número de horas (`ofHours`), o de un número de minutos (`ofMinutes`), o de un número de segundos (`ofSeconds`), o de un número de milisegundos (`ofMillis`), o de nanosegundos (`ofNanos`). Finalmente, el tipo también proporciona un método que permite crear una duración a partir de una cantidad, y un valor de tipo `TemporalUnit` (un tipo que representa la magnitud en la que está expresada la cantidad).

Si queremos representar una cantidad temporal que se mueva en el rango de días, semanas, meses o años usaremos el tipo `Period`. Internamente, los objetos de tipo `Period` se almacenan con años, meses y días. El tipo `Period` proporciona métodos específicos para crear periodos a partir de una cantidad que representa un número de días (`ofDays`), o a partir de un número de semanas (`ofWeeks`), o de un número de meses (`ofMonths`), o de un número de años (`ofYears`). Finalmente, el tipo también proporciona un método que permite crear una duración a partir de un número de años, meses y días, y otro (`between`) que permite crear un periodo como el número de años meses y días transcurridos entre dos fechas. En este caso, se incluye la fecha inicial y se excluye la final.

TIPO	MÉTODOS DE FACTORÍA
Duration	static Duration ofDays (long days) static Duration ofHours (long hours) static Duration ofMinuts (long minutes) static Duration ofSeconds (long seconds) static Duration ofMillis (long millis) static Duration ofNanos (long nanos) static Duration of (long amount, TemporalUnit unit) static Duration between(Temporal startInclusive, Temporal endExclusive)
Period	static Period ofDays(int days) static Period ofWeeks(int weeks) static Period ofMonths(int months)

	`static Period ofYears(int years)`
	`static Period of(int years, int months, int days)`
	`static Period between (LocalDate startDateInclusive,`
	`LocalDate endDateExclusive)`

El siguiente trozo de código crea siete objetos de tipo duración usando los distintos métodos de factoría proporcionados por el tipo `Duration`.

```
Duration d1 = Duration.ofDays(6);
System.out.println("Duración 1 - " + d1);

Duration d2 = Duration.ofHours(6);
System.out.println("Duración 2 - " + d2);

Duration d3 = Duration.ofMinutes(6);
System.out.println("Duración 3 - " + d3);

Duration d4 = Duration.ofSeconds(6);
System.out.println("Duración 4 - " + d4);

Duration d5 = Duration.ofMillis(6);
System.out.println("Duración 5 - " + d5);

Duration d6 = Duration.ofNanos(6);
System.out.println("Duración 6 - " + d6);

Duration d7 = Duration.of(66, ChronoUnit.MINUTES);
System.out.println("Duración 7 - " + d7);

LocalTime hora1 = LocalTime.of(15, 30);
LocalTime hora2 = LocalTime.of(15, 45);
Duration d8 = Duration.between(hora1, hora2);
System.out.println("Duración 8 - " + d8);
```

El código anterior produce por consola la siguiente salida:

```
Duración 1 - PT144H
Duración 2 - PT6H
Duración 3 - PT6M
Duración 4 - PT6S
Duración 5 - PT0.006S
Duración 6 - PT0.000000006S
Duración 7 - PT1H6M
Duración 8 - PT15M
```

Note que el formato de la representación como cadena será `PTnHnMnS`, donde n es el número relevante de horas, minutos o segundos de la duración. Cualquier fracción de segundos se representa con el punto decimal. Si una sección tiene un valor cero, se omite. Así, aunque d1 se creara dando como unidad el número de días, representa una duración de 7 días, su representación como cadena es PT144H, que se muestra en horas. Fíjese también en que aunque d7 se crea dando la información en minutos, su representación como cadena es

PT1H6M (1 hora y 6 minutos). Por último, también se puede obtener una duración a partir de dos objetos temporales con el método `between`. En el ejemplo se calcula la diferencia entre dos objetos de tipo `LocalTime`, dando como resultado una diferencia de 15 minutos.

Si intentáramos crear una duración con el método of y una unidad temporal no soportada por el tipo, se elevaría la excepción `UnsupportedTemporalTypeException`. Así, por ejemplo, la duración `d8` del siguiente trozo de código no se llegaría a crear, elevándose la excepción correspondiente.

```
Duration d8 = Duration.of(66, ChronoUnit.YEARS);
System.out.println("Duración 8 - " + d8);
```

En el siguiente trozo de código se crean cinco objetos de tipo `Period` usando cada uno de los métodos de factoría comentados anteriormente.

```
Period p1 = Period.ofDays(36);
System.out.println("Periodo 1 - " + p1);

Period p2 = Period.ofWeeks(6);
System.out.println("Periodo 2 - " + p2);

Period p3 = Period.ofMonths(16);
System.out.println("Periodo 3 - " + p3);

Period p4 = Period.ofYears(6);
System.out.println("Periodo 4 - " + p4);

Period p5 = Period.of(1, 13, 4);
System.out.println("Periodo 5 - " + p5);

LocalDate fecha1= LocalDate.of (2016, Month.SEPTEMBER, 19);
LocalDate fecha2= LocalDate.of (2017, Month.JANUARY, 13);
Period p6 = Period.between(fecha2, fecha1);
System.out.println("Periodo 6 - " + p6);
```

La salida por consola que produce el código anterior es:

```
Periodo 1 - P36D
Periodo 2 - P42D
Periodo 3 - P16M
Periodo 4 - P6Y
Periodo 5 - P1Y13M4D
Periodo 6 - P-3M-24D
```

En este caso el formato es `PnYnMnD`, donde n es el número días, meses o años. Note que en este caso, a diferencia de lo que ocurría con el tipo `Duration`, no se muestra el número relevante, sino que la información se muestra de acuerdo a la forma de crear el objeto. Por ejemplo, p5 representa un periodo de 1 año, 13 meses y 4 días, y se muestra tal cual (`P1Y13M4D`), en lugar de mostrarse como 2 años, 1 mes y 4 días.

Finalmente, fíjese en que el último periodo es negativo, ya que la fecha inicial del periodo (`fecha2`) es posterior a la fecha final del mismo (`fecha1`).

4. Métodos de consulta (get) sobre fechas

Los métodos consultores más interesantes, desde el punto de vista de la asignatura, para consultar fechas y horas son los que se muestran en la siguiente tabla.

TIPOS	MÉTODOS DE CONSULTA

`LocalDate`
`LocalDateTime`
	`int getDayOfMonth()`
	`DayOfWeek getDayOfWeek()`
	`int getDayOfYear()`
	`Month getMonth()`
	`int getMonthValue()`
	`int getYear()`

`LocalTime`
`LocalDateTime`
	`int getHour()`
	`int getMinute()`
	`int getNano()`
	`int getSecond()`

El primer bloque de métodos se puede invocar tanto desde objetos de tipo `LocalDate` como desde objetos de tipo `LocalDateTime` y permite:

- Consultar el día del mes (`getDayOfMonth`) de una fecha, obteniendo un número entero de 1 a 31.
- Consultar el día de la semana (`getDayOfWeek`) de una fecha, obteniendo un valor del enumerado `DayOfWeek`[2]. `DayOfWeek` define las etiquetas `SUNDAY`, `MONDAY`, `TUESDAY`, `WENEDSDAY`, `THURSDAY`, `FRIDAY` y `SATURDAY`, para cada uno delos días de la semana.
- Consultar el día del año (`getDayOfYear`) de una fecha, obteniendo un número entero de 1 a 365.
- Consultar el mes (`getMonth`) de una fecha, obteniendo un valor del enumerado `Month`. `Month` define las etiquetas `JANUARY`, `FEBRUARY`, `MARCH`, `APRIL`, `JUNE`, `JULY`, `AUGUST`, `SEPTEMBER`, `OCTOBER`, `NOVEMBER` y `DECEMBER`, para cada uno de los meses del año.
- Consultar el mes, pero como valor numérico (`getMonthValue`), obteniendo un número entero entre 1 y 12.
- Consultar el año (`getYear`), obteniendo un número entero que representa el año.

El segundo bloque de métodos se puede invocar tanto desde objetos de tipo `LocalTime` como desde objetos de tipo `LocalDateTime` y permiten:

- Consultar la hora (`getHour`), obteniendo un número entero de 0 a 23.
- Consultar los minutos de una hora (`getMinute`), obteniendo un número entero de 0 a 59.
- Consultar los nanosegundos de una hora (`getNano`), obteniendo un número entero de 0 a 999.999.999.
- Consultar los segundos de una hora (`getSecond`), obteniendo un número entero de 0 a 59.

[2] https://docs.oracle.com/javase/8/docs/api/java/time/DayOfWeek.html

En el siguiente trozo de código se ve cómo se usan estos métodos con un objeto de tipo LocalDateTime:

```
LocalDateTime ahora = LocalDateTime.now();
System.out.println("Día del mes: " + ahora.getDayOfMonth());
System.out.println("Día de la semana: " + ahora.getDayOfWeek());
System.out.println("Día del año: " + ahora.getDayOfYear());
System.out.println("Mes: " + ahora.getMonth());
System.out.println("Mes (como número): " + ahora.getMonthValue());
System.out.println("Hora: " + ahora.getHour());
System.out.println("Minuto: " + ahora.getMinute());
System.out.println("Segundo: " + ahora.getSecond());
System.out.println("Nanosegundo: " + ahora.getNano());
```

El resultado que se muestra por consola al ejecutar el código anterior es:

```
Día del mes: 7
Día de la semana: FRIDAY
Día del año: 281
Mes: OCTOBER
Mes (como número): 10
Hora: 14
Minuto: 0
Segundo: 40
Nanosegundo: 989000000
```

5. Métodos de consulta (get) sobre intervalos temporales

En relación a los intervalos temporales (Duration y Period), algunos de los métodos más interesantes de consulta son los que se muestran en la tabla siguiente:

TIPOS	MÉTODOS DE CONSULTA
Duration	int getNano() long getSeconds() long get (TemporalUnit unit)
Period	int getDays() int getMonths() int getYears() long get (TemporalUnit unit)

El tipo `Duration` proporciona métodos para:

- Consultar el número de nano segundos de la duración (`getNano`), obteniendo un número entero cuyo rango es de 0 999.999.999.
- Consultar el número de segundos de la duración (`getSeconds`), obteniendo un número entero, que puede tomar valores negativos para expresar duraciones negativas.

- Consultar uno de los dos atributos (`get`), indicando la unidad temporal para hacer referencia al atributo que se quiere recuperar. Como unidad temporal solo se admiten los valores `NANO` y `SECONDS`. En caso de indicar una unidad distinta a estas dos, se elevará una excepción.

En el siguiente trozo de código se muestra cómo se usan estos métodos:

```
Duration d = Duration.of(66, ChronoUnit.MINUTES);

System.out.println("Segundos: "+ d.getSeconds());
System.out.println("Nanosegundos: "+ d.getNano());
System.out.println("Segundos: "+ d.get(ChronoUnit.SECONDS));
System.out.println("Nanosegundos: "+ d.get(ChronoUnit.NANOS));
System.out.println("Minutos: "+ d.get(ChronoUnit.MINUTES));
```

La salida que se muestra en consola al ejecutar el código anterior es

```
Segundos: 3960
Nanosegundos: 0
Segundos: 3960
Nanosegundos: 0
Exception in thread "main" java.time.temporal.UnsupportedTemporalTypeException:
Unsupported unit: Minutes
    at java.time.Duration.get(Unknown Source)
```

Note que los métodos `getSeconds` y `getNanos` son un reflejo de cómo se almacena la información en los atributos del tipo. Aunque la duración `d` se haya creado a partir de un número de minutos, internamente ese dato se almacena en segundos y nanosegundos. Así, cuando consultamos los segundos vemos que nos muestra 3960.
Como puede comprobar, la llamada `get(ChronoUnit.SECONDS)` es equivalente a una llamada a `getSeconds`, y la llamada a `get(ChronoUnit.NANOS)` es equivalente a una llamada a `getNanos`. Finalmente, es interesante remarcar que cuando intentamos hacer una consulta con el método `get` y `MINUTES` como unidad temporal se eleva una excepción de tipo `UnsupportedTemporalTypeException`.

El tipo `Period` proporciona métodos para:

- Consultar el número de días del periodo (`getDays`), obteniendo un número entero que puede ser negativo.
- Consultar el número de meses del periodo (`getMonths`), obteniendo un número entero, que puede tomar valores negativos.
- Consultar el número de años del periodo (`getYears`), obteniendo un número entero, que puede tomar valores negativos.
- Consultar uno de los tres atributos a los que hacen referencia los métodos consultores anteriores (`get`), indicando la unidad temporal para hacer referencia al atributo que se quiere recuperar. Como unidad temporal solo se admiten los valores `DAYS`, `MONTHS` y `YEARS`. En caso de indicar una unidad distinta a estas dos, se elevará una excepción.

En el siguiente trozo de código se muestra el uso de estos métodos:

```
Period p = Period.of(1, 13, 4);

System.out.println("Años: " + p.getYears());
System.out.println("Meses: " + p.getMonths());
System.out.println("Días: " + p.getDays());
System.out.println("Años: " + p.get(ChronoUnit.YEARS));
System.out.println("Meses: " + p.get(ChronoUnit.MONTHS));
System.out.println("Días: " + p.get(ChronoUnit.DAYS));
System.out.println("Semanas: " + p.get(ChronoUnit.WEEKS));
```

La salida que se muestra por la consola al ejecutar el código anterior sería:

```
Años: 1
Meses: 13
Días: 4
Años: 1
Meses: 13
Días: 4
Exception in thread "main" java.time.temporal.UnsupportedTemporalTypeException:
Unsupported unit: Weeks
    at java.time.Period.get(Unknown Source)
```

Al igual que ocurría con el tipo `Duration`, los métodos `getYears`, `getMonths` y `getDays` son un reflejo de cómo está almacenada la información en los atributos del tipo. Como se puede comprobar a la hora de crear los objetos de tipo `Period`, la información no se normaliza, es decir, no se guarda como 2 años, 1 mes y 4 días, lo que se puede comprobar al recuperar la información.
También podemos ver que la llamada `get(ChronoUnit.YEARS)` es equivalente a `getYears()`; la llamada `get(ChronoUnit.MONTHS)` es equivalente a `getMonths()`; y la llamada `get(ChronoUnit.DAYS)` es equivalente a `getDays()`. Por último, al usar `get` con la unidad `WEEKS`, se eleva una excepción de tipo `UnsupportedTemporalTypeException`.

6. Operaciones aritméticas sobre fechas

Los operadores + y – en Java no están sobrecargados para poder ser usados con los tipos relacionados con las fechas. Así que para sumar o restar días, meses, años, horas, minutos, segundos o nanosegundos se utilizan un conjunto de métodos cuyo nombre empieza por `plus`, para realizar operaciones de suma, y `minus` para realizar operaciones de resta. Como `LocalDate`, `LocalTime` y `LocalDateTime` son tipos inmutables, todos los métodos de este grupo devuelven una copia del objeto sobre el que se invocan con la cantidad sumada o sustraída.
El conjunto de métodos disponible se muestra en la tabla siguiente. La notación usada en la tabla para hacer referencia al tipo de retorno de los métodos indica si el método devolverá un tipo u otro, dependiendo del tipo del objeto con el que se invoca. Es decir, si el método `minusDays` se invoca a través de un objeto de

tipo `LocalDate`, el resultado será `LocalDate`, mientras que si se invoca a través de un objeto de tipo `LocalDateTime`, el valor devuelto será de tipo `LocalDateTime`.

TIPOS	MÉTODOS RELACIONADOS CON OPERACIONES ARITMÉTICAS
LocalDate LocalDateTime	[LocalDate|LocalDateTime] minusDays(long daysToSubtract) [LocalDate|LocalDateTime] minusWeeks(long weeksToSubtract) [LocalDate|LocalDateTime] minusMonths(long monthsToSubtract) [LocalDate|LocalDateTime] minusYears(long yearsToSubtract) [LocalDate|LocalDateTime] minus (long amount, TemporalUnit unit) [LocalDate|LocalDateTime] minus (TemporalAmount amountToSubtract) [LocalDate|LocalDateTime] plusDays(long daysToAdd) [LocalDate|LocalDateTime] plusWeeks(long weeksToAdd) [LocalDate|LocalDateTime] plusMonths(long monthsToAdd) [LocalDate|LocalDateTime] plusYears(long yearsToAdd) [LocalDate|LocalDateTime] plus (long amount, TemporalUnit unit) [LocalDate|LocalDateTime] plus (TemporalAmount amountToAdd)
LocalTime LocalDateTime	[LocalTime|LocalDateTime] minusNanos(long nanosToSubtract) [LocalTime|LocalDateTime] minusSeconds(long secondsToSubtract) [LocalTime|LocalDateTime] minusMinutes(long minutesToSubtract) [LocalTime|LocalDateTime] minusHours(long hoursToSubtract) [LocalTime|LocalDateTime] minus (long amount, TemporalUnit unit) [LocalTime|LocalDateTime] minus (TemporalAmount amountToSubtract) [LocalTime|LocalDateTime] plusNanos(long nanosToAdd) [LocalTime|LocalDateTime] plusSeconds(long secondsToAdd) [LocalTime|LocalDateTime] plusMinutes(long minutesToAdd) [LocalTime|LocalDateTime] plusHours(long hoursToAdd) [LocalTime|LocalDateTime] plus (long amount, TemporalUnit unit) [LocalTime|LocalDateTime] plus (TemporalAmount amountToAdd)

Los métodos `minusDays`, `minusWeeks`, `minusMonths`, `minusYears`, `minusNanos`, `minusSeconds`, `minusMinutes` y `minusHours`, devuelven una fecha u hora, con la cantidad pasada como parámetro sustraída de la fecha u hora con la que se invoca. La magnitud de la cantidad viene dada por el propio nombre del método. Es decir, si invocamos al método `minusDays`, la cantidad sustraída representará el número de días a sustraer. De forma equivalente, hay versiones de estos métodos para añadir una cantidad (`plusDays`, `plusWeeks`, `plusMonths`, `plusYears`, `plusNanos`, `plusSeconds`, `plusMinutes` y `plusHours`).

Algunos ejemplos de uso de estos métodos son:

```
LocalDate f1 = LocalDate.of(2008,Month.FEBRUARY,29);
LocalDate f2 = f1.plusYears(1);
System.out.println("Un año después..."+ f2);

LocalDate f3 = LocalDate.of(2007,Month.MARCH,31);
LocalDate f4 = f3.plusMonths(1);
System.out.println("Un mes después..."+ f4);

LocalDate f5 = LocalDate.of(2008,Month.DECEMBER,31);
LocalDate f6 = f5.plusDays(1);
System.out.println("Un día después..."+ f6);
```

Al ejecutar este código la salida que se muestra por consola es:

```
Un año después...2009-02-28
Un mes después...2007-04-30
Un día después...2009-01-01
```

Note que, en este caso, tanto `plusYears` como `plusMonths` hacen los ajustes necesarios a las fechas para que el resultado de la operación sea una fecha válida. Fíjese en que si sumamos un año al 29 de febrero de 2008, el resultado sería el 29 de febrero de 2009 (fecha inválida ya que 2009 no es un año bisiesto) y el método ajusta la fecha al 28 de Febrero de 2009. Lo mismo ocurre con la operación `plusMonths`, ya que al sumar un mes al 31 de marzo de 2007, obtendríamos el 31 de abril de 2007 (fecha inválida porque abril es un mes que no tiene 31 días), que es ajustado al 30 de abril de 2007.

Las llamadas a los métodos `plus` o `minus` pueden sustituir a las operaciones anteriores con los parámetros adecuados. Así, el siguiente código es equivalente al anterior, y su salida por la consola sería exactamente la misma.

```
LocalDate f1 = LocalDate.of(2008,Month.FEBRUARY,29);
LocalDate f2 = f1.plus(1, ChronoUnit.YEARS);
System.out.println("Un año después..."+ f2);

LocalDate f3 = LocalDate.of(2007,Month.MARCH,31);
LocalDate f4 = f3.plus(1, ChronoUnit.MONTHS);
System.out.println("Un mes después..."+ f4);

LocalDate f5 = LocalDate.of(2008,Month.DECEMBER,31);
LocalDate f6 = f5.plus(1, ChronoUnit.DAYS);
System.out.println("Un día después..."+ f6);
```

Los métodos `plus` o `minus` elevan la excepción `UnsupportedTemporalTypeExcepcion` si se le pasa como parámetro una unidad temporal a la que no le dan soporte. Por ejemplo, al ejecutar la siguiente línea de código se elevaría esta excepción, ya que `LocalDate` es un tipo que no trabaja con nanosegundos.

```
LocalDate f7 = f5.plus(1, ChronoUnit.NANOS);
```

Por último, también es interesante notar que el método `minus` es equivalente al método plus con la cantidad en negativo. Es decir, las dos siguientes líneas de código son equivalentes.

```
LocalDate f8 = f5.plus(-1, ChronoUnit.DAYS);
LocalDate f8 = f5.minus(1, ChronoUnit.DAYS);
```

`plus` y `minus` también tienen una versión con un solo parámetro que representa una cantidad temporal. De esta forma podemos sumar duraciones o periodos a una fecha o a una hora, como en el siguiente ejemplo.

```
LocalDate f1 = LocalDate.of(2008,Month.FEBRUARY,29);
Period p1 = Period.ofYears(1);
```

```
LocalDate f2 = f1.plus(p1);
System.out.println("Un año después..."+ f2);

LocalTime hora1 = LocalTime.of(12,12,12);
Duration d1 = Duration.ofMinutes(40);
LocalTime hora2 = hora1.plus(d1);
System.out.println("40 minutos después..."+ hora2);

LocalDateTime fh1 = LocalDateTime.of(2008,Month.DECEMBER,31, 12, 12, 12);
LocalDateTime fh2 = fh1.plus(p1).plus(d1);
System.out.println("Un año y 40 minutos después..."+ fh2);
```

cuya salida por la consola es

```
Un año después...2009-02-28
40 minutos después...12:52:12
Un año y 40 minutos después...2009-12-31T12:52:12
```

Estas versiones de `plus` y `minus` elevan una excepción de tipo `UnsupportedTemporalType` excepción si se usa como parámetro una cantidad temporal no soportada. Así, esta excepción se elevará si intentamos sumar o restar a un objeto de tipo `LocalDate` una duración, o a un objeto de tipo `LocalTime` un periodo.

El conjunto de métodos anteriores nos sirve para sumar o restar una cantidad temporal a una fecha. Para restar fechas, obteniendo como resultado una cantidad temporal, por ejemplo, ver el número de días transcurridos entre dos fechas se usa el método `until`. La declaración del método `until` para los tipos `LocalDate`, `LocalTime` y `LocalDateTime` se puede ver en la tabla siguiente:

TIPOS	MÉTODOS RELACIONADOS CON OPERACIONES ARITMÉTICAS
LocalDate	Period until (ChronoLocalDate endDateExclusive) long until (Temporal endExclusive, TemporalUnit unit)
LocalTime LocalDateTime	long until (Temporal endExclusive, TemporalUnit unit)

`LocalDate` tiene dos versiones del método `until`: una que toma una fecha (`endDateExclusive`), y devuelve el periodo resultado de restar la dos fechas; y otra que toman una fecha un una unidad temporal y devuelve un long que representa la diferencia entre las fechas, medidas en las unidades dadas por el parámetro `unit`. Por ejemplo, para calcular la diferencia entre dos fechas podemos ejecutar el siguiente código usando la primera versión del método `until`:

```
LocalDate fechaNacimiento = LocalDate.of(2013, 12,3);
LocalDate hoy = LocalDate.now();
Period p = fechaNacimiento.until(hoy);
System.out.println ("El periodo entre las dos fechas es " + p);
System.out.println ("Los años del periodo son " + p.getYears());
System.out.println ("Los meses del periodo son " + p.getMonths());
System.out.println ("Los días de periodo son " + p.getDays());
System.out.println ("El número total de meses entre las dos fechas es " +
                                                        p.toTotalMonths());
```

se muestra por la consola

```
Los años del periodo son 2
Los meses entre las dos fechas son 11
El número total de d entre las dos fechas es 0
El número total de meses entre las dos fechas es 35
```

note que para obtener el número total de meses del periodo hay que usar el método `toTotalMonths()`, ya que el método consultor `getMonths()` nos devuelve el número de meses del periodo (los 11 meses de los 2 años y 11 meses que representa el periodo). Una forma equivalente de obtener el periodo de tiempo transcurrido entre dos fechas es usar el método de factoría `between()` del tipo `Period` (ver sección 3). Así la llamada a `between` siguiente

```
Period p = Period.between (fechaNacimiento, hoy);
```

es equivalente a la llamada a `until` vista en el trozo de código anterior.

La diferencia entre dos fechas también se puede calcular usando la segunda versión de until. Con esta segunda versión especificamos la unidad en la que queremos obtener la diferencia. Así, en el siguiente trozo de código podemos ver cómo se calcula la diferencia en meses, días, años o semanas entre dos fechas.

```
long meses = fechaNacimiento.until(hoy, ChronoUnit.MONTHS);
System.out.println ("El número de meses entre las dos fechas es "+ meses);
long dias = fechaNacimiento.until(hoy, ChronoUnit.DAYS);
System.out.println ("El número de días entre las dos fechas es "+ dias);
long anyos = fechaNacimiento.until(hoy, ChronoUnit.YEARS);
System.out.println ("El número de años entre las dos fechas es "+ anyos);
long semanas = fechaNacimiento.until(hoy, ChronoUnit.WEEKS);
System.out.println ("El número de semanas entre las dos fechas es "+ semanas);
```

7. Operaciones aritméticas con cantidades temporales

El conjunto de métodos para realizar operaciones aritméticas de suma y resta de cantidades temporales es similar al usado para los tipos relacionados con las fechas y horas. La diferencia más notable es que los tipos `Period` y `Duration` proporcionan un método para multiplicar la cantidad temporal que representan por un escalar (`multipliedBy`). La lista de métodos asociados a cada tipo se puede ver en la tabla siguiente:

TIPOS	MÉTODOS RELACIONADOS CON OPERACIONES ARITMÉTICAS
Period	Period minusDays(long daysToSubtract) Period minusMonths(long monthsToSubtract) Period minusYears(long yearsToSubtract) Period minus (TemporalAmount amountToSubtract) Period plusDays(long daysToAdd) Period plusMonths(long monthsToAdd) Period plusYears(long yearsToAdd) Period plus (TemporalAmount amountToAdd) Period multipliedBy (int scalar)

Duration	Duration minusNanos(long nanosToSubtract) Duration minusSeconds(long secondsToSubtract) Duration minusMinutes(long minutesToSubtract) Duration minusHours(long hoursToSubtract) Duration minusDays(long daysToSubtract) Duration minus(long amountToSubtract, TemporalUnit unit) Duration minus (Duration duration) Duration plusNanos(long nanosToAdd) Duration plusSeconds(long secondsToAdd) Duration plusMinutes(long minutesToAdd) Duration plusHours(long hoursToAdd) Duration plusDays(long daysToAdd) Duration plus(long amountToAdd, TemporalUnit unit) Duration plus(Duration duration)

Al igual que ocurría también con las fechas y horas, al ser `Period` y `Duration` tipos inmutables, todas las operaciones devuelven copias de los objetos originales con la cantidad añadida o sustraída. Algunos ejemplos de uso de estas operaciones son los siguientes.

```
Duration d1 = Duration.of(66, ChronoUnit.MINUTES);
Duration d2 = d1.plusDays(1);
System.out.println("Un día más..."+d2);
Duration d3 = d1.plus(d2);
System.out.println("d1 más d2..."+d3);
Duration d4 = d1.multipliedBy(2);
System.out.println("Duplicamos la duración..."+d4);

Period p1 = Period.ofDays(36);
Period p2 = p1.plusMonths(1);
System.out.println("Un mes más..."+p2);
Period p3 = p1.plus(p2);
System.out.println("p1 más p2..."+p3);
Period p4 = p1.multipliedBy(2);
System.out.println("Duplicamos el periodo..."+p4);
```

Al ejecutar el código anterior por la consola se obtiene lo siguiente:

```
Un día más...PT25H6M
d1 más d2...PT26H12M
Duplicamos la duración...PT2H12M
Un mes más...P1M36D
p1 más p2...P1M72D
Duplicamos el periodo...P72D
```

8. Comparación de fechas y horas

Además de los métodos `equals` y `compareTo`, los tipos `LocalDate`, `LocalDateTime` y `LocalTime`, proporcionan los métodos `isAfter`, `isBefore` e `isEquals` (este último solo para `LocalDate` y `LocalDateTime`), que nos permiten saber, dadas dos fechas si una es posterior, anterior o igual a otra, respectivamente. La cabecera de los métodos proporcionados por cada tipo se muestra en la siguiente tabla.

TIPOS	MÉTODOS RELACIONADOS CON OPERACIONES ARITMÉTICAS
LocalDate	boolean isAfter(ChronoLocalDate other) boolean isBefore(ChronoLocalDate other) boolean isEqual(ChronoLocalDate other)
LocalDateTime	boolean isAfter(ChronoLocalDateTime<?> other) boolean isBefore(ChronoLocalDateTime<?> other) boolean isEqual(ChronoLocalDateTime<?> other)
LocalTime	boolean isAfter(LocalTime other) boolean isBefore(LocalTime other)

En el siguiente trozo de código se muestra un ejemplo de uso de estos métodos con el tipo `LocalDate`.

```
LocalDate fecha1= LocalDate.of (2016, Month.SEPTEMBER, 19);
LocalDate fecha2= LocalDate.of (2017, Month.JANUARY, 13);
System.out.println("¿Es fecha1 anterior a fecha2? " +
                                              fecha1.isBefore(fecha2));
System.out.println("¿Es fecha1 posterior a fecha2? " +
                                               fecha1.isAfter(fecha2));
System.out.println("¿Es fecha1 igual a fecha2? "+fecha1.isEqual(fecha2));
```

9. Conversión de magnitudes en cantidades temporales

Los tipos `Period` y `Duration` proporcionan una serie de métodos para obtener la cantidad temporal que representan en una magnitud concreta. La tabla siguiente muestra una lista de estos métodos.

TIPOS	MÉTODOS RELACIONADOS CON OPERACIONES ARITMÉTICAS
Period	long toTotalMonths()
Duration	long toDays() long toHours() long toMinutes() long toMillis() long toNanos()

El tipo `Period` proporciona solamente el método `toTotalMonths()` que devuelve el número total de meses del periodo. El tipo `Duration` proporciona métodos para obtener el número de días que representa el periodo (`toDays`), el número de horas (`toHours`), el número de minutos (`toMinutes`), el número de milisegundos (`toMillis`), o el número de nanosegundos (`toNanos`).

Ejemplos de uso de estos métodos se pueden ver en el siguiente trozo de código:

```
Period p = Period.of(1, 13, 4);
System.out.println("Periodo en meses: " + p.toTotalMonths());

Duration d = Duration.of(66, ChronoUnit.MINUTES);
System.out.println("Duración 7 - " + d);
System.out.println("Duración en días: " + d.toDays());
System.out.println("Duración en horas: " + d.toHours());
System.out.println("Duración en minutos: " + d.toMinutes());
System.out.println("Duración en milisegundos: " + d.toMillis());
System.out.println("Duración en nanosegundos: " + d.toNanos());
```

El resultado de la ejecución del código anterior en la consola es el siguiente

```
Periodo en meses: 25
Duración en días: 0
Duración en horas: 1
Duración en minutos: 66
Duración en milisegundos: 3960000
Duración en nanosegundos: 3960000000000
```

10. Formateo de fechas y horas

En ocasiones es necesario mostrar las fechas o las horas en un formato distinto al que por defecto se ha definido en los métodos `toString` de los tipos `LocalDate`, `LocalTime` o `LocalDateTime`. Para obtener una cadena que represente la fecha o la hora en un formato distinto, los tres tipos anteriores proporcionan el método `format`, que tiene como argumento un objeto de tipo `DateTimeFormatter`[3], con el que se especifica el formato de la cadena que se obtendrá a partir de la fecha.

`LocalDate`
`LocalTime`
`LocalDateTime`
	`String format(DateTimeFormatter formatter)`

Para crear un objeto de tipo `DateTimeFormatter`, se puede usar el método de factoría `ofPattern`, que toma una cadena de caracteres como parámetro de entrada, con la que se especifica el formato. La cadena de caracteres está formada por una secuencia de letras y símbolos. Las letras del patrón tienen un significado especial. Por ejemplo, al ejecutar el siguiente trozo de código

```
LocalDateTime fechaHora = LocalDateTime.of(2012, Month.JULY, 7, 22, 36);
String fechaHoraFormateada = fechaHora.format(
                                DateTimeFormatter.ofPattern("dd-MM-yy HH:mm (EEEE)");
System.out.println("Fecha y hora: " + fechaHoraFormateada);
```

Obtenemos por la consola el siguiente mensaje

[3] https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

```
Fecha y hora: 07-07-12 22:36 (martes)
```

Note que en el patrón `dd` hace referencia al día de la fecha, escrito con dos dígitos; `MM` hace referencia al mes de la fecha, escrito en formato numérico y con dos dígitos; `yy`, al año escrito con dos dígitos; `hh` a la hora, escrita con dos dígitos; `mm`, a los minutos (con dos dígitos); y, finalmente, `EEEE` hace referencia al día de la semana, escrito en formato largo. Algunos de los caracteres que se pueden usar en el patrón y su significado los puede ver en la tabla siguiente. Para ver una lista completa de los mismos, acuda a la documentación de Java.

SÍMBOLO	SIGNIFICADO	PRESENTACIÓN	EJEMPLOS
y	Año de la era	Año	2004; 04
M/L	Mes de año	Número /Texto	7;07;Jul;Julio;J
d	Día del mes	Número	10
E	Día de la semana	Texto	Mar; Martes; M
H	Hora del día (de 0 a 23)	Número	0
m	Minuto de la hora	Número	30
S	Segundo del minuto	Número	55
n	Nanosegundos	Número	987654321

El número de veces que se repite una letra en el patrón determina el formato, con la siguiente interpretación:

- **Texto.** El estilo del texto, viene determinado por el número de veces que se repite la letra en el patrón. Si se repite menos de 4 veces, se usará el formato corto. Si se repite 4 veces se usa el formato largo. Si se repite 5 veces se usa el formato estrecho. Así, en el ejemplo anterior se repite cuatro veces la letra E, para que el día de la semana aparezca en formato largo (martes). Si quisiéramos que apareciera en formato corto (mar), repetiríamos la E tres veces. Finalmente, si quisiéramos que apareciera solamente una M, en el patrón tendríamos que poner la E cinco veces.
- **Número.** Si la letra se repite solamente una vez, el número se escribe con el mínimo número de dígitos y sin rellenar con ceros por la izquierda. Si se repite más de una vez, el número de veces que se repite indica el ancho del campo, y se rellenará con ceros por la izquierda hasta completar el ancho. En el patrón del ejemplo anterior, la d aparece repetidas dos veces, lo que indica que el día aparecerá escritos con dos dígitos, y completado con ceros por la izquierda, si fuera necesario.
- **Número /Texto.** Si el patrón tiene más de tres letras, se usan las reglas del texto, y si tiene menos de tres, la regla de los números. En el patrón del ejemplo anterior, la M aparece duplicada, por lo que el mes se muestra en formato número con dos dígitos completado con ceros por la izquierda. Si quisiéramos escribir el mes como texto en formato largo, tendríamos que repetir la M cuatro veces.
- **Año.** En este caso el número de veces que se repite la letra indica el ancho mínimo para que se complete el número con ceros. Si la letra se repite dos veces, el año se representa en formato reducido (con dos dígitos). Este es el caso del patrón del ejemplo anterior, en el que la letra y se repite dos veces, lo que significa que queremos que el año se muestre con dos dígitos.

Algunos ejemplos más de formato son los siguientes:

```
LocalDate fecha = LocalDate.of(2016, 12,3);
String fechaFormateada = fecha.format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));
System.out.println("Fecha: " + fechaFormateada);
```

```
LocalTime hora = LocalTime.now();
String horaFormateada = hora.format(DateTimeFormatter.ofPattern("HH:mm:ss:n"));
System.out.println("Hora: " + horaFormateada);
```

La salida resultado de ejecutar el código es la siguiente:

```
Fecha: 03-12-2016
Hora: 19:00:19:56000000
```

11. Formateo de intervalos temporales

Al igual que ocurre con las fechas y horas, en ocasiones es necesario mostrar las duraciones en un formato distinto al que por defecto se ha definido en los métodos `toString` de los tipos `Duration` o `Period`. Sin embargo, estos tipos no disponen de un método `format`, al igual que tienen los tipos que representan fechas. Lo que haremos en este caso es usar el método format de String y los métodos consultores (disponibles a partir de Java 9) que permiten acceder a cada una de las unidades que componen la duración.

Algunos ejemplos de formateo de duraciones y periodos son:

```
Duration d = Duration.ofMillis(38114000);
long h = d.toHours();
long m = d.toMinutesPart();
long s = d.toSecondsPart();
String durStr = String.format("%02d:%02d:%02d", h, m, s);
System.out.println("Duración: " + d +"-"+ durStr);

Period p = Period.of(1, 13, 4);
long years = p.getYears();
long months = p.getMonths();
long days = p.getDays();
String perStr = String.format(
                    "%d año(s) %d mes(es) y %d día(s)", years, months, days);
System.out.println("Periodo: " + p + " - " + perStr);
```

La salida resultado de ejecutar el código es la siguiente:

```
Duración: PT10H35M14S - 10:35:14
Periodo: P1Y13M4D - 1 año(s) 13 mes(es) y 4 día(s)
```

12. Conversión de cadenas a fechas y horas

Para crear un objeto de tipo `LocalDate`, `LocalTime` o `LocalDateTime` a partir de una cadena de caracteres se usa el método `parse` que tienen estos tipos, que tiene como argumento la cadena que se quiere convertir a fecha u hora, y un objeto de tipo `DateTimeFormatter`[4], con el que se especifica el formato

[4] https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html

que se espera que tenga la cadena a partir de la que se obtendrá fecha u hora. El uso detallado del tipo `DateTimeFormatter` lo puede ver en la Sección 10. Algunos ejemplos de conversiones de cadena a `LocalDate`, `LocalTime` y `LocalDateTime` son:

```
String d = "07/07/12";
LocalDate ld = LocalDate.parse(d, DateTimeFormatter.ofPattern("d/M/yy"));

String t = "22:36";
LocalTime lt = LocalTime.parse(t, DateTimeFormatter.ofPattern("H:m"));

String fh = "07-07-12 22:36";
LocalDateTime ldt = LocalDateTime.parse(fh,
                                DateTimeFormatter.ofPattern("dd-MM-yy HH:mm"));

System.out.println(ld);
System.out.println(lt);
System.out.println(ldt);
```

La salida resultado de ejecutar el código es la siguiente:

```
2012-07-07
22:36
2012-07-07T22:36
```

13. Conversión de cadenas a intervalos temporales

Los tipos `Duration` y `Period` tienen un diseño similar a los tipos que trabajan con fechas y horas para convertir una cadena de caracteres a un objeto que representa un intervalo temporal: en ambos tipos se usa el método `parse`. Sin embargo, a diferencia de las fechas y horas, el método `parse` de los tipos `Duration` y `Period` solo tiene como argumento la cadena que se quiere convertir duración o periodo. Esto significa que el método `parse` solo puede convertir a duraciones y periodos aquellas cadenas que tengan el mismo formato que produce el `toString` de cada tipo. Por ejemplo, el siguiente trozo de código convierte las cadenas con el formato adecuado a objetos de tipo `Duration` y `Period`, respectivamente.

```
String ds = "PT10H35M14S";
Duration d = Duration.parse(ds);
System.out.println("Duración: " + d);

String ps = "P1Y13M4D";
Period p = Period.parse(ps);
System.out.println("Periodo: " + p);
```

Si queremos convertir cadenas con un formato distinto al que tiene la representación como cadena por defecto del tipo, tenemos de definir nuestro propio método, que ya depende de la fecha.