# Introducción

Para gestionar las fechas, usaremos la clase *LocalDate*; para la hora, *LocalTime* y para la combinación de fecha y hora, *LocalDateTime*. Las tres clases tienen implementado el método *toString()*, aunque éste los muestra con formato anglosajón. Para representar fechas y/u horas formateadas a nuestro gusto, usaremos la clase *DateTimeFormatter*.

## Fechas

Podemos crear un objeto *Localdate* de varias formas. Por ejemplo, el 12 de marzo de 2021, sería,

```
LocalDate d1 = LocalDate.of(2021, 3, 12);
```

o bien,

```
LocalDate d1 = LocalDate.parse("2021-03-12");
```

Si queremos crear la fecha actual en el contexto (país, zona horaria, etc.) de la máquina que ejecute el programa escribiremos,

```
LocalDate d2 = LocalDate.now();
```

Para incrementar *d1* en cuatro días,

```
LocalDate d3 = d1.plus(4, ChronoUnit.DAYS);
```

De forma análoga se puede incrementar en un número de semanas, meses o años.

O bien, para decrementar, disponemos de *minus()*.

Los métodos de las *API* de fechas y horas devuelven objetos inmutables y nunca *null* para que puedan ser encadenados de forma segura. Así el valor devuelto por un método se puede usar como parámetro de entrada del siguiente sin tener que introducir una variable intermedia. Por ejemplo,

```
LocalDate d4 = LocalDate.now().minus(2, ChronoUnit.DAYS);
```

asigna a *d4* la fecha de anteayer.

La clase *LocalDate* también proporciona una serie de métodos que permiten extraer campos de las fechas, como el día, el mes o el año, o averiguar el día de la semana, o bien hacer comprobaciones y comparaciones. La sentencia,

```
int dia = d1.getDay()//el 12
```

nos devuelve el día del mes. En cambio,

```
int mes = d1.getMonth();// el 3 (marzo)
```

es el mes del año.

Para saber si el año de una fecha es bisiesto,

```
boolean bisiesto = d1.isLeapYear();
```

Para comparar dos fechas,

```
boolean despues = d3.isAfter(d1);//true: d3 es después que d1
```

o bien,

```
boolean antes = d3.isBefore(d1);//false: d3 no es antes que d1
```

De todas formas, la clase *LocalDate* tiene implementada la interfaz *Comparable*, de modo que,

```
d3.compareTo(d1)
```

devuelve un número positivo, el número de años completos entre *d1* y *d3*; a igualdad de años, la diferencia de meses y, a igualdad de años y meses, la diferencia de días. Esto supone que una tabla de fechas se puede ordenar de menor (más antigua) a mayor con *Arrays.sort()*.

El periodo de tiempo entre dos fechas se obtiene con el método *until()*, que devuelve un objeto de la clase Period,

```
Period p = LocalDate.parse("2015-05-12").
                    until(LocalDate.parse("2021-08-05"));
```

o bien,

```
Period p = Period.between(LocalDate.parse("2015-05-12"),
                                  LocalDate.parse("2021-08-05"));
System.out.println(p);
```

Muestra por pantalla,

```
P6Y2M24D
```

que significa un periodo de 6 años, 2 meses y 24 días. Estos tres valores, se pueden extraer de *p* con los métodos *getDays(), getMonths y getYears()*.

Si queremos obtener el número de meses (u otra unidad de tiempo) completos comprendidos entre dos fechas,

```
long totalDias = ChronoUnit.MONTHS.between(LocalDate.of(2015, 11, 21),
                LocalDate.of(2021, 9, 25));//70 meses
```

Hasta ahora, todas las fechas se han mostrado con formatos donde primero aparece el año, después el mes y luego el día del mes. Si se ejecuta la sentencia,

```
System.out.println(d1);
```

obtenemos por pantalla,

```
2021-03-12
```

Para mostrar una fecha con formato español, tendremos que definirlo creando un objeto del tipo *DateTimeFormatter* con el patrón deseado,

```
DateTimeFormatter formato1 = DateTimeFormatter.ofPattern("dd-MM-yyyy");
```

con dos cifras para el día, otras dos para el mes y cuatro para el año, en ese orden. Obsérvese que, en la definición del patrón, las letras del mes van en mayúsculas. Cuando se ponen en minúsculas se refieren a los minutos, como veremos más adelante.

También podemos establecer el formato de un contexto determinado por medio de,

```
DateTimeFormatter formato2=DateTimeFormatter.
                ofLocalizedDate(FormatStyle.SHORT).
                    withLocale(Locale.getDefault());
```

donde *FormatStyle* es un tipo enumerado con los valores *FULL, LONG, MEDIUM y SHORT*, que muestra las fechas con distinto grado de detalle. El método *withLocale()* establece la localización del estilo. El valor *Locale.getDefault()* detecta el contexto de la máquina donde se ejecuta el programa. La sentencia,

```
System.out.println(d1.format(formato2));
```

muestra por pantalla,

```
12/03/21
```

La clase *DateTimeFormatter* también nos permite hacer la operación contraria, es decir, interpretar una cadena que representa una fecha y convertirla en un objeto *LocalDate*,

```
LocalDate d5 = LocalDate.parse("04/09/15", formato2);
System.out.println(d5);
```

obteniéndose por pantalla,

```
2015-09-04
```

Si en el método *parse()* se omite el formato de usuario, *Java* emplea el formato por defecto *"yyyy-MM-dd"*,

```
LocalDate d5 = LocalDate.parse("2015-09-04");
```

## Horas

Para las horas (sin fecha) usaremos la clase *LocalTime*. Si queremos crear un objeto *LocalTime* con la hora 14:32,

```
LocalTime t1 = LocalTime.of(14, 32);
```

o bien,

```
LocalTime t1 = LocalTime.parse("14:32");
```

Con segundos,

```
LocalTime t2 = LocalTime. of(14, 32, 15);
```

La hora actual a partir del reloj del sistema,

```
LocalTime t3 = LocalTime.now();
System.out.println(t3);
```

Obtendríamos por pantalla algo así como,

```
11:14:51.302300900
```

donde la parte decimal de los segundos son nanosegundos.

A partir de un *LocalTime*, se pueden obtener los distintos campos con *getHour(), getMinute() o getSecond()*,

```
int minuto=t1.getMinute();// 32
```

Podemos incrementar un *LocalTime* con *plus()*. Para añadir 35 minutos,

```
LocalTime t4 = t1.plus(35, ChronoUnit.MINUTES);// las 15:07
```

O decrementar con *minus()*,

```
LocalTime t5 = t1.minus(5, ChronoUnit.HOURS);// las 9:32
```

Para la clase *LocalTime* también disponemos de los métodos *isAfter()* e *isBefore()*. Así mismo, tiene implementada la interfaz *Comparable*, con lo cual sus objetos se pueden ordenar.

El intervalo de tiempo entre dos instancias *LocalTime* se obtiene con el método *between()*, como un objeto de la clase *Duration*

```
Duration d = Duration.between(t2, LocalTime.of(18, 21, 1));
System.out.println(d);
```

obteniéndose por pantalla,

```
PT3H48M46S
```

es decir, 3 horas, 48 minutos y 46 segundos.

También se puede obtener el intervalo en una unidad de tiempo determinada,

```
long intervalo = ChronoUnit.MINUTES.
                 between(t2, LocalTime.of(18, 21, 1));//228 min
```

El formato con que se representan las horas también se puede configurar con la clase *DateTimeFormatter*. Por ejemplo, si queremos mostrar los segundos con tres cifras decimales,

```
DateTimeFormatter formato3=DateTimeFormatter.ofPattern("hh:mm:ss.SSS");
```

Vemos que las cifras de los minutos se representan con ‘m’ minúscula y las cifras decimales de los segundos con ‘S’ mayúscula.

```
System.out.println(LocalTime.now().format(formato3));
```

nos mostraría algo como,

```
12:47:37.327
```

## Fechas - Horas

Cuando queremos combinar la fecha con la hora, usamos objetos de la clase *LocalDateTime*. Por ejemplo, para obtener la instancia correspondiente al día 31 de diciembre de 2020 a las 23h 59 min 59 s,

```
LocalDateTime dt1 = LocalDateTime.of(2020, 12, 31, 23, 59, 59);
System.out.println(dt1);// 2020-12-31T23:59:59
```

Podemos observar en el formato de salida que la letra ‘T’ aparece separando la fecha de la hora.

Para obtener una instancia con la fecha y hora actual, leídos del sistema, usaremos el método *now()*,

```
LocalDateTime dt2 = LocalDateTime.now();
```

Podemos obtener cualquiera de los campos de una instancia con *getYear()*, *getMonth()*, etc.,

```
int hora = dt1.getHour();// 23
```

Para incrementar el valor de una fecha-hora usamos *plus()*, y para decrementarlo, *minus()*. Por ejemplo, dentro de un año a esta misma hora,

```
LocalDateTime dt3 = LocalDateTime.now().plus(1, ChronoUnit.YEARS);
```

De una instancia *LocalDateTime* se pueden extraer la fecha y la hora por separado con *toLocalDate()* y *toLocalTime()*,

```
LocalDateTime dt4 = LocalDateTime.of(2021, 6, 14, 12, 30, 30);
LocalDate d4 = dt4.toLocalDate();// la fecha
```

También disponemos de los métodos *isAfter()* e *isBefore()*, así como de la interfaz *Comparable*. Por tanto, podemos usar los métodos *sort()* con elementos *LocalDateTime*.

Para calcular el intervalo de tiempo entre dos instancias, medido en una unidad determinada, usamos *until()*. Por ejemplo, para calcular el número de horas completas que hay entre *dt1* y *dt4*,

```
long n = dt1.until(dt4, ChronoUnit.HOURS);// 3948 horas
```

Si queremos usar un formato de salida de usuario, creamos una instancia de DateTimeFormatter,

```
DateTimeFormatter formato4 = DateTimeFormatter.
                ofPattern("dd-MM-yyyy'T'hh:mm:ss.SSSS");
```

Vemos que la 'T' que separa la fecha de la hora va entre comillas. Esto significa que debe aparecer tal cuál, y no va a ser sustituida por ningún valor. Igualmente, podemos hacer uso de los métodos *ofLocalizedDateTime()* y *withLocale()* para escoger formatos locales predefinidos con distinto grado de detalle para la fecha y para la hora,

```
DateTimeFormatter formato5 = DateTimeFormatter.
            ofLocalizedDateTime(FormatStyle.SHORT, FormatStyle.MEDIUM).
            withLocale(Locale.getDefault());
LocalDateTime dt5 = LocalDateTime.now();
System.out.println(dt5.format(formato5));// 12/30/20, 11:41:58 AM
```