1) LocalDate

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa una fecha y tiene métodos para trabajar con ella

LocalDate
+ static LocalDate of(int año, int mes, int día) + static LocalDate of(int año, Month mes, int día) + static LocalDate now() + static LocalDate parse(String texto, DateTimeFormatter formato) + int getDayOfMonth() + DayOfWeek getDayOfWeek() + int getMonthValue() + int getYear() + boolean isLeapYear() + boolean isAfter(LocalDate d) + boolean isBefore(LocalDate d) + String format(DateTimeFormatter formato)

- El primer método **of** crea una fecha a partir de su año, mes y día.
- El segundo método **of** funciona igual que el anterior, pero el mes está expresado como una de las constantes definidas en la clase Month.
- El método **now** devuelve la fecha actual.
- El método **parse** permite obtener un LocalDate a partir de un String que contiene una fecha en el formato indicado por el objeto DateTimeFormatter
- El método **getDayOfMonth** devuelve el día del mes de la fecha que hay representada en el objeto LocalDate
- El método **getDayOfWeek** devuelve un objeto con información sobre el día de la semana de la fecha que hay representada en el objeto LocalDate
- El método **getMonthValue** devuelve el número del día del mes de la fecha que hay representada en el objeto LocalDate.
- El método **getYear** devuelve el año de la fecha que hay representada en el objeto LocalDate
- El método **isLeapYear** devuelve true si el año de la fecha que hay representada en el objeto LocalDate es bisiesto
- El método **isAfter** devuelve true si el objeto LocalDate es posterior al objeto LocalDate que se recibe como parámetro
- El método **isBefore** devuelve true si el objeto LocalDate es anterior al objeto LocalDate que se recibe como parámetro
- El método **format** devuelve un String en el que la fecha aparece con el formato indicado en el objeto DateTimeFormatter.

2) LocalTime

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa un momento del tiempo formado por una hora, minutos, segundos y nanosegundos.

LocalTime
+ static LocalTime of(int horas, int minutos) + static LocalTime of(int horas, int minutos, int segundos) + static LocalTime of(int horas, int minutos, int segundos, int nanosegundos) + static LocalTime now() + int getHour() + int getMinute() + int getSecond() + int getNano() + boolean isAfter(LocalTime t) + boolean isBefore(LocalTime t)

- El primer método **of** crea un LocalTime con una hora y unos minutos
- El segundo método **of** crea un LocalTime con una hora, unos minutos y unos segundos
- El tercer método **of** crea un LocalTime con una hora, unos minutos, unos segundos y unos nanosegundos
- El método **now** crea un LocalTime que tiene la hora actual del sistema
- El método **getHour** devuelve la hora asociada al momento representado en el objeto LocalTime
- El método **getMinute** devuelve los minutos asociados al momento representado en el objeto LocalTime
- El método **getSecond** devuelve los segudnos asociados al momento representado en el objeto LocalTime
- El método **getNano** devuelve los nanosegundos asociados al momento representado en el objeto LocalTime
- El método **isAfter** devuelve true si la hora representada por el objeto LocalTime es posterior a la del objeto LocalTime recibido como parámetro
- El método **isBefore** devuelve true si la hora representada por el objeto LocalTime es anterior a la del objeto LocalTime recibido como parámetro

3) LocalDateTime

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa un momento del tiempo formado por una combinación de un LocalDate y un LocalTime. Por tanto, es un momento situado en una fecha y hora concretos.

LocalTimeTime
+ static LocalDateTime of(int año, int mes, int día, int horas, int minutos, int segundos, int nanos) + static LocalDateTime of(LocalDate d, LocalTime t) + static LocalDateTime now()

- El primer método **of** crea un LocalDateTime con una hora y unos minutos
- El segundo método **of** crea un LocalDateTime con el momento indicado por los objetos LocalDate y LocalTime pasados como parámetros
- El método **now** obtiene un LocalDateTime con la fecha y hora actual del sistema

Importante: Por ser un LocalDateTime un objeto compuesto por un LocalDate y un LocalTime, la clase tiene **todos** los métodos de instancia que se han visto en ellas.

4) DateTimeFormatter

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase es usada por los métodos **parse** y **format** de la clase LocalDate para convertir fechas en String aplicando un formato, y también para hacer el proceso inverso. Por ejemplo, los LocalDate usan por defecto el formato año-mes-día. Gracias a esta clase podemos pasar un LocalDate a un String en el que se use otro formato, como día/mes/año.

DateTimeFormatter
+ String ofPattern(String patrón)

- El método **ofPattern** obtiene un DateTimeFormatter que trabaja con el patrón indicado como parámetro. El patrón es un texto que describe el formato usado para las fechas y horas. En él pueden usarse los siguientes símbolos con el significado que se indica:

Símbolo	Significado	Ejemplo
yy	año, con 2 dígitos	18
yyyy	Año, con 4 dígitos	2018
d	Día del mes	5
dd	Día del mes, con 2 dígitos	05
M	Número del mes	8
MM	Número del mes, con 2 dígitos	08
MMM	Abreviatura del mes	ago
MMMM	Nombre completo del mes	agosto
H	Hora	5
HH	Hora, con 2 dígitos	05
m	Minutos	7
mm	Minutos, con 2 dígitos	07
s	Segundos	3
ss	Segundos, con 2 dígitos	03
e	Número del día de la semana	1
ee	Número del día de la semana, con 2 dígitos	01
eee	Nombre abreviado del día de la semana	lun
eeee	Nombre completo del día de la semana	lunes

5) DayOfWeek

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa un día de la semana. Posee constantes que tienen predefinidos los objetos DayOfWeek correspondientes a todos los días de la semana:

- DayOfWeek.MONDAY
- DayOfWeek.TUESDAY
- DayOfWeek.WEDNESDAY
- DayOfWeek.THURSDAY
- DayOfWeek.FRIDAY
- DayOfWeek.SATURDAY
- DayOfWeek.SUNDAY

DayOfWeek
+ String name() + int getValue()

- El método **name** devuelve el nombre del día de la semana
- El método **getValue** devuelve el número del día de la semana, teniendo en cuenta que 1 es el lunes y 7 el domingo.

6) Instant

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa un instante de tiempo concreto.

Instant
+ static Instant now() + static Instant ofEpochMillis(long n)

- **now:** obtiene un objeto Instant con el momento actual
- **ofEpochMillis:** Obtiene un objeto Instant a partir del tiempo Unix pasado como parámetro.

7) Duration

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa el intervalo de tiempo comprendido entre dos instantes

Duration
+ static Duration between (Instant t1, Instant t2) + long getSeconds() + long toDays() + long toHours() + long toMillis() + long toMinutes()

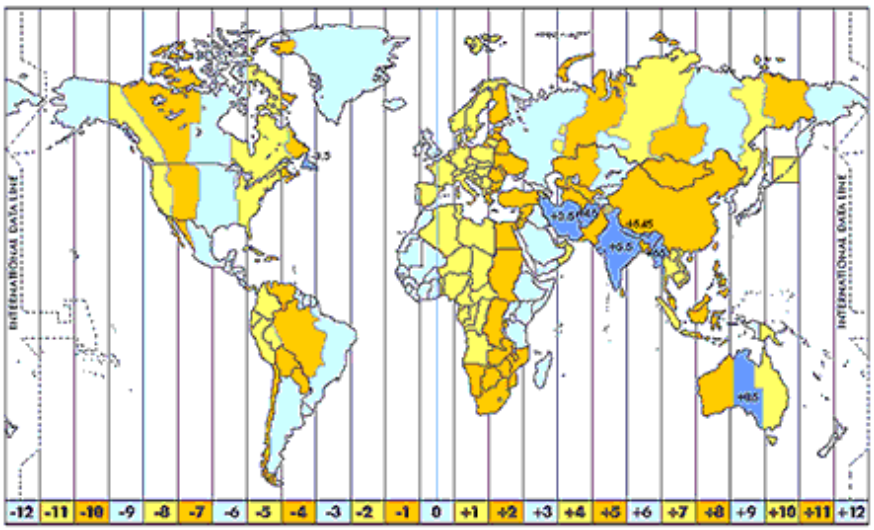
- El método **between** obtiene un objeto Duration con la duración del intervalo de tiempo comprendido entre los instantes definidos en t1 y t2
- Los demás métodos obtienen la duración expresada en las correspondientes unidades.

8) ZoneId

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa una zona horaria del planeta.

☞ ***¿Qué son las zonas horarias?*** *Como ya sabes, el planeta rota sobre sí mismo y eso hace que en unos lugares sea de día, en otros de noche, etc. Para que la sensación de una hora sea reconocida por igual en todo el mundo (por ejemplo, las 0:00 sea interpretado como "medianoche" en todo el planeta), es necesario variar las horas conforme nos movemos por el planeta. La división GMT divide el planeta en 24 zonas, siendo la zona de referencia la zona 0 (GMT+0), que es la hora de la ciudad inglesa de Greenwich. Nosotros en invierno estamos en la zona GMT+1 (nuestra hora se obtiene sumando 1 hora a la hora de Greenwich) y en verano adoptamos la zona GMT+2 (sumamos 2 horas). En general, la hora en la zona +X se obtiene sumando X a la hora de Greenwich, y la de la zona –X se obtiene restando X a la hora de Greenwich.*



ZoneId
+ static Set<String> getAvailableZoneIds() + static ZoneId of(String nombre)

- **getAvailableZoneIds:** Devuelve un conjunto con los nombres de todas las zonas horarias del planeta
- **of:** Devuelve un objeto ZoneId con la zona horaria cuyo nombre se pasa como parámetro. Dicho nombre debe ser algún valor válido, de los que devuelve el método getAvailableZoneIds.

9) ZonedDateTime

- **Librería:** *Librería estándar de Java*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.time*

Esta clase representa una fecha y hora en una zona horaria concreta del planeta. De hecho, esta clase incluso tiene en cuenta los cambios de hora que se producen en verano y en invierno.

Como esta clase incluye una fecha y una hora, es posible usar sobre ella todos los métodos de LocalDate y LocalTime: métodos get, métodos para darle formato, etc.

ZonedDateTime
+ static ZonedDateTime now() + static ZonedDateTime now(ZoneId z) + static ZonedDateTime of(LocalDateTime d, ZoneId z) + static ZonedDateTime ofInstant(Instant t)

- **primer método now:** Obtiene un ZonedDateTime con la fecha y hora actual de la zona horaria por defecto del equipo.
- **segundo método now:** Obtiene un ZonedDateTime con la fecha y hora actual de la zona horaria que se pasa como parámetro.
- **segundo método now:** Obtiene un ZonedDateTime cuya fecha y hora es la del LocalDateTime pasado como primer parámetro, y en la zona horaria que se pasa como segundo parámetro.
- **ofInstant:** Crea un ZonedDateTime a partir de un Instant