LocalDateTime

1.Características

La clase LocalDateTime pertenece al paquete java.time y representa una fecha y hora sin zona horaria. Es una combinación de LocalDate (fecha) y LocalTime (hora).

- Representa la fecha y hora en el formato yyyy-MM-dd-HH-mm-ss.zzz.
- Es parte del paquete java.time, diseñado para simplificar el manejo de fechas y horas en Java.
- Implementa la interfaz ChronoLocalDateTime y hereda de la clase Object.
- Inmutabilidad y seguridad en hilos: Las instancias de LocalDateTime no se pueden modificar después de ser creadas, por eso son buenísimas para entornos multihilo.
- Versatilidad: Es perfecta para operaciones donde la zona horaria no es relevante, como configurar recordatorios locales o registrar logs de eventos.

Métodos principales de java.time.LocalDateTime

Estos son los métodos más usados:

- now(): Obtiene la fecha y hora actual.
- plusDays(long days): Suma días a la fecha actual.
- minusHours(long hours): Resta horas a la fecha actual.
- format(DateTimeFormatter formatter): Da formato a la fecha.
- parse(CharSequence text, DateTimeFormatter formatter): Convierte texto en fecha.

2.Crear instancias

Ten en cuenta que puedes crear un objeto LocalDateTime de diferentes maneras:

```
import java.time.LocalDateTime;

public class Ejemplo {
    public static void main(String[] args) {
        // Fecha y hora actuales
        LocalDateTime ahora = LocalDateTime.now();
        System.out.println("Fecha y hora actuales: " + ahora);

        // Fecha y hora específicas
        LocalDateTime fechaEspecifica = LocalDateTime.of(2025, 1, 15, 14, 30);
        System.out.println("Fecha específica: " + fechaEspecifica);
    }
}
```

Otra forma de instanciar objetos de tipo LocalDateTime es utilizando el método .of() donde puedes introducir directamente la fecha y hora deseada.

3.Modificar fechas y horas

Puedes crear nuevas instancias basadas en operaciones sobre la original:

```
public class ModificarFechas {
    public static void main(String[] args) {
        LocalDateTime fechaBase = LocalDateTime.of(2025, 1, 15, 14, 30);

        // Sumar tiempo
        LocalDateTime suma = fechaBase.plusDays(5).plusHours(2);
        System.out.println("Fecha modificada (sumar): " + suma);

        // Restar tiempo
        LocalDateTime resta = fechaBase.minusWeeks(1).minusMinutes(30);
        System.out.println("Fecha modificada (restar): " + resta);
    }
}
```

Además, puedes obtener ambas partes del objeto (LocalDate, LocalTime) por separado:

```
// Obtener instancias de LocalDate
    LocalDate fecha = fechaBase.toLocalDate();
    LocalTime hora=fechaBase.toLocalTime();
```

4.Formatear fechas

En caso de que quieras o necesites mostrar la fecha en un formato legible, se aconseja usar DateTimeFormatter:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class FormatearFechas {
   public static void main(String[] args) {
      LocalDateTime ahora = LocalDateTime.now();
      DateTimeFormatter formato = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

      String fechaFormateada = ahora.format(formato);
      System.out.println("Fecha formateada: " + fechaFormateada);
   }
}
```

5.Parsear cadenas a fechas

Aquí te muestro que, si tienes una cadena y necesitas convertirla en un objeto LocalDateTime, usa el método parse:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class ParsearFechas {
   public static void main(String[] args) {
      String fechaTexto = "15/01/2025 14:30";
      DateTimeFormatter formato = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

      LocalDateTime fecha = LocalDateTime.parse(fechaTexto, formato);
```

```
 System.out.println("Fecha parseada: " + fecha);
    }
}
```

6.Clase Duration

Hay que mencionar la clase Duration perteneciente también al paquete java.time.

La clase Duration se utiliza para representar una cantidad de tiempo en términos de horas, minutos o segundos. Se usa comúnmente para calcular diferencias entre LocalDateTime o para sumar/restar periodos de tiempo.

6.1Métodos más utilizados:

Método	**Descripción**
between(Temporal startInclusive, Temporal endExclusive)	Obtiene la diferencia entre dos objetos temporales
ofDays(long days)	Obtiene un tipo Duration que representa una cantidad de días estándar de 24 horas.
ofHours(long hours)	Obtiene un tipo Duration que representa una cantidad de horas estándar.
ofMinutes(long minutes)	Obtiene un tipo Duration que representa una cantidad de minutos estándar.
get(TemporalUnit unit)	Obtiene el valor de la unidad solicitada
isZero()	Comprueba si la duración devuelve 0
plus(Duration duration)	Devuelve un copia con la duración especificada añadida
toDays()	Obtiene el número de días
toHours()	Obtiene el número de horas
toMinutes()	Obtiene el número de minutos

```
import java.time.Duration;
import java.time.LocalDateTime;

public class EjemploDuration {
    public static void main(String[] args) {
        LocalDateTime salida = LocalDateTime.of(2025, 3, 28, 10, 30);

        Duration tiempoVuelo = Duration.ofHours(2).plusMinutes(45);
        LocalDateTime llegada = salida.plus(tiempoVuelo);

        System.out.println("Hora de salida: " + salida);
            System.out.println("Tiempo de vuelo: " + tiempoVuelo.toHours() + " horas y " +
tiempoVuelo.toMinutesPart() + " minutos");
        System.out.println("Hora estimada de llegada: " + llegada);
    }
}
```

Respuesta en consola:

```
Hora de salida: 2025-03-28T10:30
Tiempo de vuelo: 2 horas y 45 minutos
Hora estimada de llegada: 2025-03-28T13:15
```

7.Bibliografía

https://keepcoding.io/blog/como-usar-java-time-localdate-en-java/
https://www.geeksforgeeks.org/java-time-duration-class-in-java/