# LOCALTIME



## 1.Introducción

LocalTime es una clase que representa una hora específica sin información de zona horaria, por eso es tan buena para trabajar con tiempos locales como horarios de apertura de una tienda o el inicio de una reunión. Además, es inmutable, lo que significa que puedes usarla sin preocuparte por problemas de concurrencia o cambios inesperados en su valor.

Anterior a esta clase teníamos que usar clases como Date o Calendar, haciendo que resultase más complejo trabajar con horas o fechas debido a que el código no era muy legible, métodos confusos y aparecían errores frecuentemente.

Oracle junto a otros desarrolladores participaron en la definición de las nuevas clases oficiales para el manejo tanto de horas como fechas en Java: las incluidas en el paquete java.time. Este paquete incluye muchas clases, pero las básicas son:

- LocalDate: representa a fechas sin la hora y nos facilita su manejo para declararlas, sumar y restar fechas y compararlas.
- LocalTime: es idéntica a la anterior pero para el manejo de horas, sin ninguna fecha asociada, pudiendo así compararlas, sumar o restar tiempo a las mismas...
- LocalDateTime: como puedes suponer, es una combinación de las dos anteriores, que permite hacer lo mismo con fechas y horas simultáneamente.
- Instant: es muy parecida a la anterior pero a la vez muy diferente. Se usa para almacenar un punto determinado en el tiempo, o sea con fecha y hora, pero guarda su valor como un timestamp de UNIX, es decir, en nanosegundos desde el epoch de UNIX (1/1/1970 a las 00:00) y usando la zona horaria UTC. Es muy útil para manejar momentos en el tiempo de

manera neutra e intercambiarlo entre aplicaciones y sistemas, por lo que lo verás utilizado muy a menudo.
- ZonedDateTime: esta clase es como la LocalDateTime pero teniendo en cuenta una zona horaria concreta, ya que las anteriores no la tienen en cuenta.
- Period: esta clase auxiliar nos ayuda a obtener diferencias entre fechas en distintos periodos (segundos, minutos, días...) y también a añadir esas diferencias a las fechas.
- Duration: esta es muy parecida a la anterior pero para manejo de horas exclusivamente.

### 1.1.Características principales:

1. Representa el tiempo como hora:minuto:segundo:nanosegundo.
2. Te da métodos para ajustar, comparar y formatear tiempos.
3. Es fácil de usar y leer gracias a su diseño claro.

## 2.Cómo crear instancias:

### 2.1.Obtener la hora actual

Para obtener la hora actual del sistema, lo mejor es usar el método LocalTime.now()

```
import java.time.LocalTime;

public class Main {
   public static void main(String[] args) {
      LocalTime now = LocalTime.now();  //Instanciamos el objeto de tipo LocalTime
      System.out.println("Hora actual: " + now);
   }
}
```

Respuesta en consola: **Hora actual: 18:52:09.914704**

### 2.2.Crear un tiempo específico

Si lo que necesitamos es un tiempo en específico, usaremos el método LocalTime.of;

```
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalTime fixedTime = LocalTime.of(14, 30); // 14:30
        System.out.println("Hora específica: " + fixedTime);
    }
}
```

Respuesta en consola: **Hora actual: 14:30**

Observa cómo con ofNanoOfDay() o ofSecondOfDay() puedes construir tiempos usando valores numéricos.

```
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalTime fromNanos = LocalTime.ofNanoOfDay(1_000_000_000L); // 00:00:01
        LocalTime fromSeconds = LocalTime.ofSecondOfDay(3600); // 01:00:00
        System.out.println("Desde nanos: " + fromNanos);
        System.out.println("Desde segundos: " + fromSeconds);
    }
}
```

Respuesta en consola:

**Desde nanos: 00:00:01**
**Desde segundos: 01:00**

Antes de continuar con la manipulación de tiempos, comentar que es posible extraer cualquier parte de de una hora a través de los métodos get() que ofrecen estas clases. Por ejemplo, getHour(), getMinute(), getSecond()...

```
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalTime time = LocalTime.of(14, 30, 45); // 14:30:45

        int hour = time.getHour();
        int minute = time.getMinute();
        int second = time.getSecond();

        System.out.println("Hora: " + hour);
```

```
        System.out.println("Minuto: " + minute);
        System.out.println("Segundo: " + second);
    }
}
```

Respuesta en consola:

**Hora: 14**
**Minuto: 30**
**Segundo: 45**

# 3.Manipulación de tiempos

## 3.1.Sumar o restar tiempo

Podemos sumar o restar horas,minutos,segundos o nanosegundos. Aquí te demuestro como:

```
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalTime now = LocalTime.now();
        LocalTime plusHours = now.plusHours(2); // Sumar 2 horas
        LocalTime minusMinutes = now.minusMinutes(30); // Restar 30 minutos
        System.out.println("Hora actual: " + now);
        System.out.println("Más 2 horas: " + plusHours);
        System.out.println("Menos 30 minutos: " + minusMinutes);
    }
}
```

Respuesta en consola:

**Hora actual: 18:55:36.415324**
**Más 2 horas: 20:55:36.415324**
**Menos 30 minutos: 18:25:36.415324**

## 3.2.Comparar tiempos

Cuando vayamos a comparar tiempos, utilizaremos los métodos isBefore() y is After(), que devuelven un booleano, como te muestro a continuación:

```
import java.time.LocalTime;

public class Main {
   public static void main(String[] args) {
      LocalTime time1 = LocalTime.of(10, 0);
      LocalTime time2 = LocalTime.of(12, 30);
      System.out.println("¿time1 es antes de time2? " + time1.isBefore(time2));
      System.out.println("¿time1 es después de time2? " + time1.isAfter(time2));
   }
}
```

Respuesta en consola:

```
¿time1 es antes de time2? true
¿time1 es después de time2? false
```

También podríamos ordenar tiempos utilizando el método **compareTo()**, que devuelve un entero:

```
import java.time.LocalTime;

public class Main {
   public static void main(String[] args) {
      LocalTime time1 = LocalTime.of(10, 0);
      LocalTime time2 = LocalTime.of(12, 30);

      int resultado = time1.compareTo(time2);

      if (resultado < 0) {
         System.out.println("time1 es antes de time2");
      } else if (resultado > 0) {
         System.out.println("time1 es después de time2");
      } else {
         System.out.println("time1 y time2 son iguales");
      }
   }
}
```

# 4.Ajustar valores

Algo sorprendente es que, con métodos como withHour() o withMinute() puedes cambiar partes específicas del tiempo. Te explico cómo:

```
import java.time.LocalTime;

public class Main {
    public static void main(String[] args) {
        LocalTime time = LocalTime.of(10, 0);
        LocalTime adjustedTime = time.withHour(15).withMinute(45); // Cambiar a 15:45
        System.out.println("Hora ajustada: " + adjustedTime);
    }
}
```

Respuesta en consola: **Hora ajustada: 15:45**

Para convertir un LocalTime en un formato legible, usa DateTimeFormatter. Recuerda que también puedes parsear cadenas con el método parse().

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

public class Main {
    public static void main(String[] args) {
        LocalTime time = LocalTime.of(16, 30);
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
        String formattedTime = time.format(formatter);
        System.out.println("Hora formateada: " + formattedTime);
            LocalTime                                    parsedTime                                    =
            LocalTime.parse("18:45",DateTimeFormatter.ofPattern("HH:mm"));
        System.out.println("Hora parseada: " + parsedTime);
    }
}
```

Respuesta en consola:

**Hora formateada: 16:30:00**
**Hora parseada: 18:45**

Vamos a analizar esta función.

Aquí se importan dos clases:

- LocalTime: Para representar una hora específica del día.
- DateTimeFormatter: Para dar formato y analizar (parse) tiempos.

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

A continuación creamos una variable de tipo LocalTime llamada **time**. La instanciaremos con la hora 16:30 y para ello utilizaremos la función LocalTime.of (hora, minuto).

**LocalTime time = LocalTime.of(16, 30);**

Luego se crea un formateador de fecha/hora (DateTimeFormatter) con el patrón "HH:mm:ss", que significa:

- HH: Hora en formato de 24 horas.
- mm: Minutos.
- ss: Segundos.

**DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");**

Creamos un variable de tipo String llamada formattedTime, que será igual a la hora formateada. Un objeto de tipo LocalTime, tiene un método llamado **format** que recibe como parámetro un objeto de tipo **DateTimeFormatter** que se encarga de transformar, en una cadena de caracteres, la hora con el patrón asignado previamente.

**String formattedTime = time.format(formatter);**
**System.out.println("Hora formateada: " + formattedTime)**

Consola: **Hora formateada: 16:30:00**

(Se añaden los segundos como 00 porque LocalTime.of(16,30) no especifica segundos, por lo que se asume 00).

Para hacer el efecto contrario, es decir, convertir un String en un LocalTime, usamos el método parse () perteneciente a la clase LocalTime.

**LocalTime parsedTime = LocalTime.parse("18:45", DateTimeFormatter.ofPattern("HH:mm"));**

Aquí se toma un String "18:45" y se convierte en un objeto LocalTime usando parse(). El formato HH:mm permite interpretar correctamente "18:45" como 6:45 PM.

# 5.Bibliografía


- https://keepcoding.io/blog/como-funciona-java-time-localtime-en-java/
- https://www.w3schools.com/java/java_date.asp
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/time/LocalTime.html
- https://www.campusmvp.es/recursos/post/como-manejar-correctamente-fechas-en-java-el-paquete-java-time.aspx