Code & Learn (DATE)

`Date` en Java

La clase `Date` pertenece al paquete `java.util` y representa un instante específico en el tiempo, con precisión hasta los milisegundos. Originalmente, `Date` era la clase principal para representar fechas en Java, pero tiene varios inconvenientes, como el manejo implícito de horas y la falta de claridad en la API para trabajar solo con fechas.

Aunque sigue siendo válida, la clase `Date` ha sido reemplazada en gran medida por la nueva API de fechas y horas de Java 8 (`java.time`), que es más flexible y segura para trabajar con fechas y horas.

Características de

`Date`:

- Representa un **instante específico en el tiempo** (fecha y hora), almacenado en milisegundos desde la **época Unix** (1 de enero de 1970).
- La clase es **mutable**, lo que significa que su estado puede cambiar después de ser creada.
- Utiliza **milisegundos** como unidad de tiempo.
- Es **no segura para hilos**, lo que significa que no puede ser usada de manera concurrente sin protección adicional.
- **Métodos obsoletos** como `getYear()`, `getMonth()`, `getDate()`, etc., que no son recomendados para su uso.

Ejemplo básico

```
    fechaActual = new Date();
    System.out.println("Fecha actual: " + fechaActual);
```

En Java, el manejo de fechas y horas es un aspecto esencial para muchas aplicaciones. A partir de **Java 8**, se introdujo el paquete `java.time`, el cual proporciona una nueva API para trabajar con fechas y tiempos de manera más eficiente y sencilla. Esta documentación cubre las principales clases y métodos para trabajar con fechas en Java, desde la manipulación básica hasta las operaciones avanzadas.

Importación de Clases

Para utilizar las funcionalidades de fechas y tiempos, es necesario importar las clases adecuadas del paquete `java.time`.

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.Month;
import java.time.format.DateTimeFormatter;
```

Clases más comunes de `java.time`:

- **`LocalDate`**: Representa una fecha (año, mes, día) sin la hora.
- **`LocalTime`**: Representa una hora del día sin la fecha.
- **`LocalDateTime`**: Combina `LocalDate` y `LocalTime` para representar una fecha y hora.
- **`Instant`**: Representa un punto en el tiempo (por ejemplo, con la precisión de milisegundos).
- **`Duration`**: Para medir intervalos de tiempo entre instantes.
- **`Period`**: Para medir intervalos entre fechas.
- **`ZoneId`**: Representa una zona horaria.
- **`ZonedDateTime`**: Fecha y hora con zona horaria.

Creación de Fechas y Horas

`LocalDate`

Permite crear una fecha con año, mes y día.

```
LocalDate fechaActual = LocalDate.now();
        System.out.println("Fecha actual: " + fechaActual);
```

Diferencias entre `Date` y `LocalDate`

A continuación se presentan las principales diferencias entre la clase `Date` (de `java.util`) y la clase `LocalDate` (de `java.time`):

Característica	`Date`	`LocalDate`
Paquete	`java.util`	`java.time`
Tipo de dato	Representa una **fecha y hora** (con precisión en milisegundos).	Representa solo una **fecha** (sin hora).
Precisión	Precisión de **milisegundos**, incluye fecha y hora.	Precisión de **días** (solo la parte de la fecha).
Mutabilidad	**Mutable**, su estado puede modificarse después de ser creado.	**Inmutable**, su valor no puede cambiar después de ser creado.
Operaciones de fecha	Necesita clases adicionales como `Calendar` o `SimpleDateFormat` para manipular fechas.	Tiene métodos integrados como `plusDays()`, `minusDays()`, `isBefore()`, `isAfter()`, etc., para manipular fechas.

Característica	`Date`	`LocalDate`
Formato y conversión	Requiere el uso de `SimpleDateFormat` o `DateFormat` para convertir entre cadenas y objetos `Date`.	Usa `DateTimeFormatter` para formatear y convertir fechas fácilmente.
Soporte de zona horaria	No maneja zonas horarias directamente, utiliza `TimeZone` o `Calendar` para manejar la zona horaria.	No tiene concepto de zona horaria, solo maneja la fecha (año, mes, día).
Propósito principal	Representa un punto en el tiempo, incluyendo la fecha y la hora.	Representa solo una fecha sin hora, útil para aplicaciones que solo necesitan la fecha (por ejemplo, cumpleaños, días de eventos).
Compatibilidad	Antigua y parte de la API clásica, en desuso para algunas tareas debido a su diseño.	Introducida en Java 8, es parte de la nueva API de fechas y horas (`java.time`), recomendada para nuevas aplicaciones.
Métodos obsoletos	Contiene métodos obsoletos como `getYear()`, `getMonth()`, `getDate()`, etc.	No tiene métodos obsoletos y proporciona una API moderna y segura para trabajar con fechas.

LocalTime

Permite crear una hora con horas y minutos (opcionalmente con segundos y nanosegundos).

```
LocalDate fecha = LocalDate.of(2025, 1, 9);  // Anio, Mes, Dia
System.out.println("Fecha: " + fecha);
```

LocalDateTime

Permite crear una fecha y hora combinada.

```
    LocalDateTime ahora = LocalDateTime.now();
    System.out.println("Fecha y hora actual: " + ahora);

    // Crear una fecha y hora especifica
    LocalDateTime fechaEspecifica = LocalDateTime.of(2023, 12, 25, 15,
30);
    System.out.println("Fecha y hora específica: " + fechaEspecifica);
```

Instant

Representa un punto específico en el tiempo, generalmente utilizado para medición de tiempo exacto.

```
    Instant ahora = Instant.now();
    System.out.println("Instant actual: " + ahora);
```

Operaciones con Fechas y Horas

Sumar y Restar Fechas

- `plusDays()`: Suma días a una fecha.
- `plusMonths()`: Suma meses a una fecha.
- `plusYears()`: Suma años a una fecha.
- `minusDays()`: Resta días a una fecha.
- `minusMonths()`: Resta meses a una fecha.
- `minusYears()`: Resta años a una fecha.

Comparación de Fechas

- `isBefore()`: Verifica si una fecha es anterior a otra.
- `isAfter()`: Verifica si una fecha es posterior a otra.
- `isEqual()`: Verifica si dos fechas son iguales.

Diferencia entre Fechas

- `Period.between()`: Calcula la diferencia entre dos fechas en términos de años, meses y días.

Formateo y Análisis de Fechas

Formatear Fechas

- `format()`: Convierte una fecha u hora en una cadena de texto con un formato específico.

```
    // Formatear el ZonedDateTime
    DateTimeFormatter formato = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss z");
    String fechaFormateada = fechaHoraEnZona.format(formato);
    System.out.println("Fecha formateada: " + fechaFormateada);
```

Analizar Fechas

- `parse()`: Convierte una cadena de texto en una fecha utilizando un formato específico.

Uso de `Instant` y `Duration`

`Instant`

Representa un instante en el tiempo (usualmente en milisegundos o nanosegundos desde el 1 de enero de 1970).

`Duration`

Permite calcular la duración entre dos instantes. Puede expresar la duración en segundos o en fracciones de segundo.

Manejo de Zonas Horarias

`ZonedDateTime`

Representa una fecha y hora con información de zona horaria.

Conversiones de Zona Horaria

- **`withZoneSameInstant()`**: Convierte una fecha y hora a otra zona horaria, manteniendo el mismo instante.

Consideraciones Importantes

- **Inmutabilidad**: Las clases de `java.time` son inmutables. Esto significa que en lugar de modificar un objeto existente, se crea un nuevo objeto con el cambio deseado.
- **Manejo de fechas y tiempos locales**: Las clases `LocalDate`, `LocalTime` y `LocalDateTime` no manejan zonas horarias. Para gestionar zonas horarias, se debe usar `ZonedDateTime`.
- **Precisión**: Las clases `Duration` e `Instant` tienen una alta precisión para manejar intervalos de tiempo, incluyendo milisegundos y nanosegundos.

8. Resumen de Métodos Clave

Método	Descripción
`now()`	Obtiene la fecha y hora actual.
`of()`	Crea una fecha o hora a partir de valores específicos.
`plusDays()`	Suma días a una fecha.
`minusDays()`	Resta días a una fecha.
`plusMonths()`	Suma meses a una fecha.
`minusMonths()`	Resta meses a una fecha.
`plusYears()`	Suma años a una fecha.
`minusYears()`	Resta años a una fecha.
`isBefore()`	Compara si una fecha es anterior a otra.
`isAfter()`	Compara si una fecha es posterior a otra.
`isEqual()`	Compara si dos fechas son iguales.
`format()`	Formatea una fecha a una cadena de texto.
`parse()`	Analiza una cadena y la convierte en una fecha.

Método	Descripción
`Duration.between()`	Calcula la duración entre dos instantes.

9. Conclusión

La API `java.time` introducida en Java 8 proporciona una manera robusta, sencilla y eficiente de trabajar con fechas y horas. Permite realizar cálculos, comparaciones, análisis y formateos de manera muy flexible y con un enfoque orientado a objetos. Se recomienda siempre utilizar estas nuevas clases en lugar de las antiguas `Date` y `Calendar`, que tienen algunas limitaciones y comportamientos inesperados.

Licencia