

ANEXO:**Contenido**

1. Clases para manejo de Fechas	I
1.1 DATE y formateo de Fechas: DateFormat y SimpleDateFormat	1
1.1.1 La clase Date	1
1.1.2 Clase DateFormat	2
1.1.3 Parsear un String	3
1.1.4 SimpleDateFormat	3
1.2 CALENDAR y formateo de Fechas: DateFormat y SimpleDateFormat	4
1.3 Java.time y formateo de Fechas: DateTimeFormatter	6
1.3.1 Formateo de Fechas con java.time y DateTimeFormatter	6
1.3.2 Convertir texto a LocalDateTime	7
1.3.3 Convertir LocalDateTime a texto	7
2. Java: Manejo y formateo de Números : NumberFormat y DecimalFormat	8

1. Clases para manejo de Fechas

Las diferentes formas de manejar fechas en Java: **Date** (más antigua), **Calendar** y la más moderna API **java.time**.

1.1 DATE y formateo de Fechas: DateFormat y SimpleDateFormat

Java cuenta el número de milisegundos desde el **primero de enero de 1970**. Esto significa, que, por ejemplo, el 2 de enero de 1970 empezó 86400000 milisegundos después. Asimismo, el 31 de diciembre de 1969 empezo 86400000 milisegundos antes que el 1 de enero de 1970.

La clase **Date** de Java toma nota de **esos milisegundos como un valor Long** y como este un número con signo, las fechas pueden ser expresadas antes y después del comienzo del 1 de enero de 1970.

1.1.1 La clase Date

La clase Date, encontrada en el paquete **java.util**, encapsula un valor Long representando un momento específico en el tiempo.

Constructores:

- **Date()**, el cual crea un objeto Date teniendo por valor el día, mes, años y hora del momento en el cual fue creado.
- **Date(long milseg)** crea un objeto Date teniendo por valor el día, mes, años y hora tomando el momento *milseg* transcurrido desde el 1 de Enero de 1970 (Epoch).

El método:

- **getTime()** devuelve un valor long de un objeto Date, y son los milisegundos transcurrido desde el 1 de Enero de 1970.

- **toString():** Devuelve una representación en cadena de la fecha

En el ejemplo siguiente, usamos el constructor Date() para crear una fecha basada en su ejecución y usamos el método getTime() para ver el número de milisegundos que ese objeto Date representa.

Ejemplo:

```
import java.util.*;
public class Now {    public static void main(String[] args)
{    Date now = new Date();
long nowLong = now.getTime();
System.out.println("El valor es: " + nowLong);    }    }
```

Obtendremos:

El valor es: 1349129892668

Ahora ¿cómo podemos formatear ese valor para que sea un poco más entendible a simple vista y sin tener que tener una calculadora a mano?

1.1.2 Clase DateFormat

Un propósito de la clase DateFormat es crear Strings de un objeto Date de tal forma que la fecha se pueda manejar y entender más fácilmente. Se encuentra en el paquete java.text.

Dependiendo del país, la fecha tiene diferentes formatos. En España preferimos verla como “25 de diciembre 2012”, mientras que en Estados Unidos prefieren “Diciembre 25, 2012”.

DateFormat es una clase abstracta no podemos hacer crear objetos (no podemos hacer new DateFormat().)

Para instanciarla se utiliza el método **getDateInstance()**:

```
DateFormat df = DateFormat.getDateInstance();
```

Cuando **una instancia de DateFormat es creada sin ningún parámetro** se toma el lenguaje y el formato predeterminado.

DateFormat provee **algunas constantes** de solo lectura para que podamos usar de argumentos en el método getInstance() como por ejemplo: **SHORT, MEDIUM, LONG** y **FULL**.

- **Podemos convertir un objeto Date a String con el método format().**

Ejemplo:

```
Date now = new Date();
DateFormat df = DateFormat.getDateInstance();
DateFormat df1 = DateFormat.getDateInstance(DateFormat.SHORT);
DateFormat df2 = DateFormat.getDateInstance(DateFormat.MEDIUM);
DateFormat df3 = DateFormat.getDateInstance(DateFormat.LONG);
DateFormat df4 = DateFormat.getDateInstance(DateFormat.FULL);
String s = df.format(now);
String s1 = df1.format(now);
String s2 = df2.format(now);
String s3 = df3.format(now);
String s4 = df4.format(now);
System.out.println("(Default) Hoy es:" + s);
System.out.println("(SHORT) Hoy es:" + s1);
System.out.println("(MEDIUM) Hoy es:" + s2);
System.out.println("(LONG) Hoy es:" + s3);
System.out.println("(FULL) Hoy es:" + s4);
```

```
(Default) Hoy es:02-oct-2012
(SHORT) Hoy es:2/10/12
(MEDIUM) Hoy es:02-oct-2012
(LONG) Hoy es:2 de octubre de 2012
(FULL) Hoy es:martes 2 de octubre de 2012
```

Podemos utilizar otros formatos de fecha utilizando otro idioma y país. Se crea una instancia de la clase **Locale**. En el constructor se le pasa 2 parámetros que especifican el lenguaje y país respectivamente.

Por ejemplo:

```
Locale locIT = new Locale("it", "IT"); // Italy
Locale locPT = new Locale("pt"); // Portugal
Locale locBR = new Locale("pt", "BR"); // Brazil
Locale locIN = new Locale("hi", "IN"); // India
Locale locMEX = new Locale("es", "MX"); // México
```

Ejemplo:

```
Locale localBrasil = new Locale("pt", "BR");
DateFormat df =
    DateFormat.getDateInstance(DateFormat.LONG, localBrasil);
Date fecha = new Date();
System.out.println(df.format(fecha));
```

1.1.3 Parsear un String

Podemos usar la clase **DateFormat** para crear objetos Date desde un String, utilizando el **método parse()**.

Este método en particular puede lanzar una excepción **ParseException**, por lo tanto tenemos que capturarla.

Un ejemplo:

```
import java.util.*;
import java.text.*;
public class ParseExample
{ public static void main(String[] args) {
String ds = "November 1, 2012";
DateFormat df = DateFormat.getDateInstance();
try {
    Date d = df.parse(ds); }
catch(ParseException e)
{ System.out.println("Error al pasar de String a Fecha"); }}
```

1.1.4 SimpleDateFormat

Las instancias default devueltas por **DateFormat** muchas veces puede ser suficiente para algunos propósitos, pero no cubre todo el aspecto de posibilidades

Por ello **simpleDateFormat, nos permite construir nuestros propios formatos**.

Las fechas van a ser construidas tomando como referencia un String que especificará un patrón para su construcción.

Veamos un ejemplo sencillo con la ayuda del objeto **SimpleDateFormat** y usando los patrones dd, MM y yyyy para representar el día, el mes y el año.

```
Date fecha = new Date();
SimpleDateFormat formato = new SimpleDateFormat("dd/MM/yyyy");
```

```
String resultado = formato.format(fecha);
System.out.println("formato de la fecha:" + resultado);
```

El patrón usado dd/MM/yyyy retorna una fecha de la manera: 02/10/2012

Letras que se pueden utilizar en los patrones:

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
Y	Year	Year	1996, 96
Y	Week year	Year	2009, 09
M	Month in year	Month	July, Jul, 07
W	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day name in week	Text	Tuesday, Tue
u	Day number of week (1 = Monday, ..., 7 = Sunday)	Number	1
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978

1.2 CALENDAR y formateo de Fechas: DateFormat y SimpleDateFormat

El uso de Java Calendar se sigue utilizando aún en día. Aunque la API java.time introducida en Java 8 es más moderna y recomendada para manejar fechas y horas, muchas aplicaciones clásicas de Java aún dependen de las clases Date y Calendar. Estas clases dividen las responsabilidades de la siguiente manera:

1. Almacenar el valor de la fecha: `java.util.Date`
2. Aplicar diferentes formatos a la fecha: `java.text.DateFormat` y `java.text.SimpleDateFormat`
3. Hacer cálculos de fechas: `java.util.Calendar`

Una vez creada una fecha con Date, esta simplemente almacena la información. Si queremos realizar operaciones sobre ella lo que tenemos que construir es un Java Calendar

Métodos para obtener una parte de la fecha

Método	Descripción	Ejemplo
<code>get(Calendar.YEAR)</code>	Obtiene el año.	<code>calendar.get(Calendar.YEAR)</code>
<code>get(Calendar.MONTH)</code>	Obtiene el mes (0-11, donde 0 es enero y 11 es diciembre).	<code>calendar.get(Calendar.MONTH)</code>
<code>get(Calendar.DAY_OF_MONTH)</code>	Obtiene el día del mes.	<code>calendar.get(Calendar.DAY_OF_MONTH)</code>
<code>get(Calendar.HOUR_OF_DAY)</code>	Obtiene la hora del día (0-23).	<code>calendar.get(Calendar.HOUR_OF_DAY)</code>
<code>get(Calendar.MINUTE)</code>	Obtiene los minutos.	<code>calendar.get(Calendar.MINUTE)</code>
<code>get(Calendar.SECOND)</code>	Obtiene los segundos.	<code>calendar.get(Calendar.SECOND)</code>

get(Calendar.DAY_OF_WEEK)	Obtiene el día de la semana (1-7, donde 1 es domingo y 7 es sábado).	calendar.get(Calendar.DAY_OF_WEEK)
get(Calendar.WEEK_OF_YEAR)	Obtiene la semana del año.	calendar.get(Calendar.WEEK_OF_YEAR)
get(Calendar.WEEK_OF_MONTH)	Obtiene la semana del mes.	calendar.get(Calendar.WEEK_OF_MONTH)

Otros métodos

Método	Descripción	Ejemplo
set(int field, int value)	Establece el valor de un campo específico.	calendar.set(Calendar.MONTH, Calendar.JANUARY)
add(int field, int amount)	Suma o resta una cantidad de tiempo a un campo.	calendar.add(Calendar.DAY_OF_MONTH, 5)
getTime()	Devuelve un objeto Date que representa el tiempo.	calendar.getTime()
getTimeInMillis()	Devuelve el tiempo en milisegundos desde el Epoch (1 de Enero de 1970).	calendar.getTimeInMillis()
setTime(Date date)	Establece el tiempo del calendario con un objeto Date.	calendar.setTime(new Date())
roll(int field, boolean up)	Incrementa el valor del campo hacia arriba o abajo.	calendar.roll(Calendar.DAY_OF_MONTH, true)
getActualMaximum(int field)	Devuelve el valor máximo que el campo puede tener.	calendar.getActualMaximum(Calendar.DAY_OF_MONTH)

```
Calendar calendar = Calendar.getInstance();

    // Obtener el año actual
int year = calendar.get(Calendar.YEAR);
System.out.println("Año actual: " + year);

    // Establecer el mes a enero
calendar.set(Calendar.MONTH, Calendar.JANUARY);
System.out.println("Fecha con mes cambiado: " + calendar.getTime());

    // Añadir 5 días al día actual
calendar.add(Calendar.DAY_OF_MONTH, 5);
System.out.println("Fecha después de añadir 5 días: " + calendar.getTime());

    // Obtener el tiempo en milisegundos desde el 1 de Enero de 1970
long millis = calendar.getTimeInMillis();
System.out.println("Milisegundos desde el 1 de Enero de 1970: " + millis);

    // Rodar el día del mes hacia arriba
calendar.roll(Calendar.DAY_OF_MONTH, true);
System.out.println("Fecha después de rodar el día hacia arriba: "
+ calendar.getTime());

    // Obtener el máximo día del mes
int maxDay = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
System.out.println("Máximo día del mes: " + maxDay);
```

Ejemplo de formatear la fecha

```
// Formatear la fecha con SimpleDateFormat
//la fecha y hora representadas serían: 20 de octubre de 2024, 18:31:06
    calendar.set(2024, Calendar.OCTOBER, 20, 18, 31, 6);
    SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
    String formattedDate = formatter.format(calendar.getTime());
    System.out.println("Fecha formateada: " + formattedDate);
```

1.3 Java.time y formateo de Fechas: DateTimeFormatter

La API **java.time** introducida en Java 8 es una mejora significativa sobre las clases Date y Calendar de java.util.

Esta API es más moderna, intuitiva y está diseñada para ser inmutable y segura para hilos. Vamos a profundizar en cómo manejar y formatear fechas utilizando java.time.

Inmutabilidad significa que una vez que **un objeto ha sido creado, su estado no puede ser modificado**. En otras palabras, los **objetos inmutables no pueden cambiar después de su creación**.

Ventajas de la Inmutabilidad:

1. **Seguridad en Hilos:** Los objetos inmutables son intrínsecamente seguros para su uso en múltiples hilos sin necesidad de sincronización. Esto se debe a que **no hay riesgo de que un hilo modifique el estado del objeto mientras otro hilo lo está utilizando**.
2. **Simplicidad:** Los objetos inmutables son más fáciles de razonar y depurar porque su estado no cambia. Esto reduce la complejidad y los errores en el código.
3. **Consistencia:** Los objetos inmutables garantizan que su estado es consistente y no puede ser alterado accidentalmente.

Las clases **Date y Calendar son mutables**, lo que significa que su estado puede cambiar después de su creación. Esto puede llevar a problemas de concurrencia si múltiples hilos intentan modificar el mismo objeto al mismo tiempo.

PRINCIPALES CLASES DE JAVA.TIME

1. **LocalDate:** Representa una fecha (año, mes, día) sin información de hora.
2. **LocalTime:** Representa una hora (hora, minuto, segundo, nanosegundo) sin información de fecha.
3. **LocalDateTime:** Combina LocalDate y LocalTime para representar una fecha y hora.
4. **ZonedDateTime:** Representa una fecha y hora con una zona horaria.
5. **Instant:** Representa un instante en el tiempo (similar a Date).

1.3.1 Formateo de Fechas con java.time y DateTimeFormatter

La clase **DateTimeFormatter**, introducida en Java 8 junto con la API java.time, permite formatear y parsear fechas y horas de manera segura, legible y flexible. Es la alternativa moderna a **SimpleDateFormat**.

Características clave de DateTimeFormatter:

- Es inmutable y segura para hilos.
- Permite formatear objetos como LocalDate, LocalTime, LocalDateTime, ZonedDateTime, etc.
- Puede usar formatos predefinidos o personalizados.

La clase DateTimeFormatter tiene **varios métodos** útiles. Aquí van los más importantes:

Método	Descripción
<code>ofPattern(String pattern)</code>	Crea un formateador con un patrón personalizado (ej. "dd/MM/yyyy")
<code>format(TemporalAccessor)</code>	Convierte una fecha/hora en texto según el patrón
<code>parse(CharSequence text)</code>	Convierte un texto en un objeto temporal (como <code>LocalDateTime</code>)
<code>withLocale(Locale)</code>	Aplica un idioma específico para nombres de meses, días, etc.
<code>withZone(ZoneId)</code>	Aplica una zona horaria al formateador (útil con <code>ZonedDateTime</code>)

Símbolos más usados para construir patrones de fecha y hora:

Símbolo	Significado	Ejemplo
<code>y</code>	Año	<code>2025</code> , <code>25</code>
<code>M</code>	Mes (número o nombre)	<code>9</code> , <code>09</code> , <code>Sep</code> , <code>septiembre</code>
<code>d</code>	Día del mes	<code>15</code>
<code>E</code>	Día de la semana	<code>Mon</code> , <code>lunes</code>
<code>H</code>	Hora (0-23)	<code>01</code> , <code>13</code>
<code>h</code>	Hora (1-12)	<code>01</code> , <code>11</code>
<code>m</code>	Minutos	<code>16</code>
<code>s</code>	Segundos	<code>45</code>
<code>a</code>	AM/PM	<code>AM</code> , <code>PM</code>
<code>'texto'</code>	Texto literal	<code>'de'</code> → <code>de</code>

1.3.2 Convertir texto a LocalDateTime

Usamos `DateTimeFormatter.parse()` junto con `LocalDateTime.parse()`.

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

public class TextoADateTime {
    public static void main(String[] args) {
        String texto = "15/09/2025 01:16:45";
        DateTimeFormatter formato =
            DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");

        LocalDateTime fechaHora = LocalDateTime.parse(texto, formato);

        System.out.println("Fecha convertida: " + fechaHora);
    }
}
```

Salida:

Fecha convertida: 2025-09-15T01:16:45

1.3.3 Convertir LocalDateTime a texto

Usamos `DateTimeFormatter.format()`.

java

```

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

public class DateTimeATexto {
    public static void main(String[] args) {
        LocalDateTime fechaHora = LocalDateTime.now();
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("EEEE, dd 'de' "
        "MMMM 'de' yyyy HH:mm:ss");

        String texto = fechaHora.format(formato);

        System.out.println("Fecha formateada: " + texto);
    }
}

```

■ Salida

Fecha formateada: lunes, 15 de septiembre de 2025 01:17:45

2. Java: Manejo y formateo de Números : NumberFormat y DecimalFormat

NumberFormat es la clase más sencilla para dar formatos a los números teniendo en cuenta el país y el idioma. Principalmente presenta los siguientes métodos:

Al igual que DateFormat, es una clase abstracta, por lo que para instanciarla tenemos que utilizar, uno de estos dos métodos:

- **getInstance()** - Simplemente obtiene el formato del idioma actual.
- **getCurrencyInstance()** - Igual que el anterior, pero con formato de moneda.

Los anteriores métodos se le puede pasar un **parámetro de tipo Locale**, que sustituirá al idioma y país actual.

Para realizar el formato será necesario llamar al método **format()**.

Ejemplo:

```

NumberFormat nf = NumberFormat.getInstance();
System.out.println(nf.format(76543210.1234));
// Resultado: 76.543.210

// Ahora con un locale distinto
nf = NumberFormat.getInstance(Locale.ENGLISH);
System.out.println(nf.format(76543210.1234));
// Resultado: 76,543,210
/*
 * Método getIntegerInstance, sirve para redondear números decimales.
 * Ojo, redondear, ¡no truncar!
 */
nf = NumberFormat.getIntegerInstance();
System.out.println(nf.format(123456.789));
// Resultado: 123.457

// Ahora, en Francés
nf = NumberFormat.getIntegerInstance(Locale.FRENCH);
System.out.println(nf.format(123456.789));
// Resultado: 123 457

/* Formato moneda */
nf = NumberFormat.getCurrencyInstance();
System.out.println(nf.format(12345678));
// Resultado: 12.345.678,00 €

```

```
// Ahora en dólares:  
nf = NumberFormat.getCurrencyInstance(Locale.US);  
System.out.println(nf.format(12345678 * 1.5023));  
// Resultado: $18,546,912.06
```

DecimalFormat

La clase DecimalFormat se encuentra en el paquete java.text y nos permite mostrar los números con el formato que queramos, es decir, con cuántos decimales, si queremos punto o coma para los decimales, etc.

Un uso simple de DecimalFormat puede ser este

```
import java.text.DecimalFormat;  
DecimalFormat formateador = new DecimalFormat("####.#####");  
  
// Esto sale en pantalla con cuatro decimales, es decir, 3,4324  
System.out.println (formateador.format (3.43242383));
```

Símbolos para utilizar en los patrones

Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
E	Number	Yes	Separates mantissa and exponent in scientific notation. <i>Need not be quoted in prefix or suffix.</i>
;	Subpattern boundary	Yes	Separates positive and negative subpatterns
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage

Si usamos ceros en vez de #, los huecos se llenarán con ceros.

```
import java.text.DecimalFormat;  
DecimalFormat formateador = new DecimalFormat("0000.0000");  
// Esto sale en pantalla con cuatro cifras enteras y cuatro decimales, es decir,  
0003,4300  
System.out.println (formateador.format (3.43));
```

Porcentajes

Si usamos en el patrón el signo de porcentaje %, el número se multiplicará automáticamente por 100 al presentarlo en pantalla.

```
DecimalFormat formateador = new DecimalFormat("##.##%");  
// Esto saca en pantalla 34,44%  
System.out.println (formateador.format(0.3444));
```