9 Fechas

1. Introducción

Java 8 introduce una nueva API para fechas y horas que es más fácil de leer, más amplia que la API anterior y thread-safe, es decir, los objetos se pueden utilizar con seguridad desde varios hilos de ejecución.

Esta API se introduce para cubrir los siguientes inconvenientes de la anterior:

1. *java.util.Date* no es thread-safe por lo que los desarrolladores tienen que hacer frente a problemas de concurrencia durante el uso de la fecha. La nueva API de fecha y hora es inmutable y no tiene métodos setter. Al ser inmutables, no necesitan ser clonadas.
2. Dificultad para manejar zona-horaria.

En java 8 se crea un nuevo paquete para el manejo de fechas, se trata del paquete `java.time`. Este paquete es una extensión a las clases *java.util.Date* y *java.util.Calendar* que vemos un poco limitado para manejo de fechas, horas y localización.

Las clases definidas en este paquete representan los principales conceptos de fecha - hora, incluyendo instantes, fechas, horas, períodos, zonas de tiempo, etc. Están basados en el sistema de calendario ISO, el cual es el calendario mundial *de-facto* que sigue las reglas del calendario Gregoriano.

2. Clase Locale

Hay que destacar la importancia y el valor de respetar el idioma y la región geográfica del usuario al desarrollar aplicaciones de software. Permitir que el usuario se comunique con el software en su propio idioma es un gran impulso para las ventas del software. Cuando se trata de Java, es el concepto de configuración regional de Java lo que explica el proceso de internacionalización.

La configuración regional de Java consta de tres elementos principales: idioma, país y variantes. Las variantes la utilizan los proveedores de software, ya sean sistemas operativos o navegadores, para funcionalidades adicionales.

En Java se utiliza la clase `Locale` del paquete `java.util` como una localización, es decir, representa una región geográfica, política o cultural específica. Una operación que requiere una localización para realizar su tarea se denomina sensible a la localización y utiliza la localización para adaptar la información al usuario. Por ejemplo, mostrar un número es una operación sensible a la configuración regional: el número debe formatearse de acuerdo con las costumbres y convenciones del país, región o cultura del usuario. Estas clases utilizan el objeto `Locale` para conocer qué configuración regional se está utilizando en la aplicación y se personalizan en cuanto a cómo formatear y presentar los datos al usuario. Por ejemplo, la clase `NumberFormat`, que puede devolver un número como 302 400 para Francia, 302 .400 para Alemania y 302, 400 para los Estados Unidos. También son clases sensibles a la configuración regional `DecimalFormat` y `DateTimeFormatter`.

Hay tres constructores disponibles para la creación de objetos `Locale`:

- `Locale(String language)`: utiliza solamente el idioma. Ejemplo: `Locale locale = new Locale("en");` En este caso, al no indicar un país, la aplicación no podrá adaptarse a las diferencias regionales del idioma.
- `Locale(String language, String country)`: utiliza tanto el idioma como el país. Ejemplo: `Locale locale = new Locale("en", "US");`
- `Locale(String language, String country, String variant)`: utiliza el idioma, el país y la variante. Ejemplo: `Locale locale = new Locale("en", "US", "SiliconValley");`

Los códigos se pueden encontrar en https://www.iana.org/assignments/language-subtag-registry/language-subtag-registry:

- *Type: language*: son los códigos de idioma.

- *Type: region*: son los códigos de país.
- *Type: variant*: son los códigos de la variante.

Los códigos de variante no conforman un estándar sino que son arbitrarios y específicos de la aplicación. Si existen variaciones en el idioma utilizado dentro del mismo país, se puede especificar una variante. Por ejemplo, en el Sur de Estados Unidos, la gente suele decir "y'all," cuando en el Norte dicen "you all." Se podría crear diferentes objetos Locale de esta forma:

```
Locale localeNorth = new Locale("en", "US" ,"NORTH");
Locale localeSouth = new Locale("en", "US", "SOUTH");
```

Si se crean objetos *Locale* con los códigos de variante NORTH y SOUTH, como en el ejemplo anterior, solo nuestra aplicación sabrá como tratarlos.

Normalmente, se específican códigos de variante para identificar diferencias causadas por la plataforma de ordenador. Por ejemplo, las diferencias de fuentes podría forzar a utilizar caracteres diferentes en Windows y en UNIX. Se podría definir los objetos **Locale** con estos códigos de variante:

```
Locale localeUnix = new Locale("de", "DE" ,"UNIX");
Locale localeWindows = new Locale("de", "DE", "WINDOWS");
```

Tal y como podemos observar en la API, la clase *Locale* tiene constantes predefinidas para algunos idiomas y países. Por ejemplo, **JAPAN** es la constante del país Japón mientras que **JAPANESE** es la constante del idioma japonés. Se puede crear un objeto *Locale* utilizando una constante:

```
Locale l1 = Locale.JAPAN;//Se crea un objeto Locale con el idioma
japonés y país Japón
Locale l2 = Locale.JAPANESE;// Se crea un objeto Locale con el idioma
japonés
```

Si no se asigna un objeto *Locale* a un objeto sensible a la localidad, entonces se utiliza la localización por defecto, la cual se puede conocer con el método `getDefault()`, que obtiene el valor actual de la configuración regional por defecto para esa instancia de la Máquina Virtual de Java. La máquina virtual de Java establece la configuración regional por defecto durante el inicio basándose en el entorno del host. Se puede cambiar utilizando el método *setDefault*.

```
package tema9_Fechas;

import java.util.Locale;

public class ClassLocale {
```

```
    public void show() {

        Locale l = Locale.getDefault();
        System.out.println(l.toString());

    }

    public static void main(String[] args) {

        new ClassLocale().show();

    }

}
```

Salida por consola:

```
es_ES
```

3. Enum DayOfWeek

En el enum `DayOfWeek` del paquete `java.time` se definen los días de la semana con el siguiente conjunto de constantes:

```
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY
```

Podemos crear variables de tipo *DayOfWeek* y asignarle una instancia del enum a través de dichas constantes:

```
DayOfWeek day1;//day1 es una variable del tipo de enumeración
DayOfWeek
day1 = DayOfWeek.MONDAY;//Se le asigna la instancia del lunes a través
de la constante MONDAY
```

También podemos crear una instancia del enum con el método estático `static DayOfWeek of(int dayOfWeek)`, que obtiene una instancia de *DayOfWeek* a partir de un valor *int* del 1 (lunes) al 7 (domingo).

```
DayOfWeek day2;//day2 es una variable del tipo de enumeración
DayOfWeek
day2 = DayOfWeek.of(1);//Se le asigna la instancia del lunes a través
del método of
```

Este enum contiene métodos que permiten manipular días hacia adelante y hacia atrás:

- `DayOfWeek plus(long days)`: devuelve el día de la semana que es el número especificado de días después de éste.
- `DayOfWeek minus(long days)`: devuelve el día de la semana que es el número especificado de días antes de éste.

```
DayOfWeek day3, day4;
day3 = day1.plus(4);//Viernes
day4 = day2.minus(2);//Sábado
```

Este enum contiene también el método `String getDisplayName(TextStyle style,Locale locale)` que devuelve la representación textual utilizada para identificar el día de la semana adecuada para su presentación al usuario. Los parámetros controlan el estilo del texto devuelto y la configuración regional. El estilo del texto se representa con el enum `TextStyle` donde se definen tres tamaños para el texto formateado: `full`, `short` y `narrow`.

```
package tema9_Fechas;

import java.time.DayOfWeek;
import java.time.format.TextStyle;
import java.util.Locale;

public class EnumDayOfWeek {

    public void show() {

        DayOfWeek day1, day2, day3, day4;//Variables del tipo de
enumeración DayOfWeek
        day1 = DayOfWeek.MONDAY;//Se le asigna la instancia del lunes
a través de la constante MONDAY
        day2 = DayOfWeek.of(1);//Se le asigna la instancia del lunes a
través del método of
        day3 = day1.plus(4);//Viernes
        day4 = day2.minus(2);//Sábado
        System.out.printf("day1: %s, day2: %s, day3: %s, day4:
%s",day1,day2,day3,day4);
        Locale l = new Locale("es","ES");//idioma español castellano,
país España
        System.out.printf("\nTextStyle.FULL:%s%n",
day1.getDisplayName(TextStyle.FULL, l));
        System.out.printf("TextStyle.NARROW:%s%n",
day1.getDisplayName(TextStyle.NARROW, l));
        System.out.printf("TextStyle.SHORT:%s%n",
day1.getDisplayName(TextStyle.SHORT, l));

    }

    public static void main(String[] args) {
```

```
        new EnumDayOfWeek().show();

    }

}
```

Salida por consola:

```
day1: MONDAY, day2: MONDAY, day3: FRIDAY, day4: SATURDAY
TextStyle.FULL: lunes
TextStyle.NARROW: L
TextStyle.SHORT: lun
```

4. Enum Month

En el enum `Month` del paquete `java.time` se definen los meses con el siguiente conjunto de constantes:

```
JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER,
OCTOBER, NOVEMBER, DECEMBER
```

También tiene los métodos `of`, `plus`, `minus` y `getDisplayName` como el enumerado de días de la semana. Además, contiene métodos para conocer el número de días máximos y mínimos de cada mes:

- `int minLength()`: obtiene la duración mínima de este mes en días. Febrero tiene una duración mínima de 28 días. Abril, junio, septiembre y noviembre tienen 30 días. Todos los demás meses tienen 31 días.
- `int maxLength()`: obtiene la duración máxima de un mes en días. Febrero tiene una duración máxima de 29 días. Abril, junio, septiembre y noviembre tienen 30 días. Todos los demás meses tienen 31 días.

```
package tema9_Fechas;

import java.time.Month;
import java.time.format.TextStyle;
import java.util.Locale;

public class EnumMonth {

    public void show() {

        Month month1, month2, month3, month4;//Variables del tipo de
enumeración Month
        month1 = Month.APRIL;//Se le asigna la instancia del mes de
abril a través de la constante APRIL
        month2 = Month.of(2);//Se le asigna la instancia del mes de
febrero a través del método of
```

```
        month3 = month1.plus(4);//Agosto
        month4 = month2.minus(3);//Noviembre
        System.out.printf("month1: %s, month2: %s, month3: %s, month4:
%s",month1,month2,month3,month4);
        System.out.printf("\n%s tiene como mínimo %d días y como
máximo %d días",month2,month2.minLength(),month2.maxLength());
        System.out.printf("\n%s tiene como mínimo %d días y como
máximo %d días",month1,month1.minLength(),month1.maxLength());
        System.out.printf("\n%s tiene como mínimo %d días y como
máximo %d días",month3,month3.minLength(),month3.maxLength());
        Locale l = new Locale("es","ES");//idioma español castellano,
país España
        System.out.printf("\nTextStyle.FULL: %s%n",
month1.getDisplayName(TextStyle.FULL, l));
        System.out.printf("TextStyle.NARROW: %s%n",
month1.getDisplayName(TextStyle.NARROW, l));
        System.out.printf("TextStyle.SHORT: %s%n",
month1.getDisplayName(TextStyle.SHORT, l));

    }

    public static void main(String[] args) {

        new EnumMonth().show();

    }

}
```

Salida por consola:

```
month1: APRIL, month2: FEBRUARY, month3: AUGUST, month4: NOVEMBER
FEBRUARY tiene como mínimo 28 días y como máximo 29 días
APRIL tiene como mínimo 30 días y como máximo 30 días
AUGUST tiene como mínimo 31 días y como máximo 31 días
TextStyle.FULL: abril
TextStyle.NARROW: A
TextStyle.SHORT: abr
```

5. Clase LocalDate

La clase `LocalDate` del paquete `java.time` nos permite registrar fechas específicas inmutables.

```java
package tema9_Fechas;

import java.time.LocalDate;

public class ClassLocalDate {
```

```
    public void show() {

        LocalDate currentDate, concertDate, parseDate;
        currentDate = LocalDate.now();//Se obtiene la fecha actual
        System.out.printf("Fecha actual: %s", currentDate);
        System.out.printf(", es decir, día %s del mes %s del año %s.",
currentDate.getDayOfMonth(),
                currentDate.getMonthValue(), currentDate.getYear());
        concertDate = LocalDate.of(2022, 8, 13);//Obtiene una
instancia de LocalDate a partir de un año, mes y día
        System.out.printf("\n\nEl concierto es el %s.", concertDate);
        System.out.println(" Las entradas se podrán comprar como
máximo hasta");
        System.out.printf("5 días antes, es decir, hasta el %s.",
concertDate.minusDays(5));
        parseDate = LocalDate.parse("2022-02-22");//Obtiene un
LocalDate a partir de una cadena
        System.out.printf("\n\nFecha obtenida a partir de una cadena:
%s", parseDate);

    }

    public static void main(String[] args) {

        new ClassLocalDate().show();

    }

}
```

Salida por consola:

```
Fecha actual: 2022-02-03, es decir, día 3 del mes 2 del año 2022.

El concierto es el 2022-08-13. Las entradas se podrán comprar como
máximo hasta
5 días antes, es decir, hasta el 2022-08-08.

Fecha obtenida a partir de una cadena: 2022-02-22
```

6. Clase YearMonth

Con la clase `YearMonth` del paquete `java.time` podemos generar objetos inmutables que representan la combinación de un año y un mes.

```
package tema9_Fechas;

import java.time.Month;
```

```
import java.time.YearMonth;

public class ClassYearMonth {

    public void show() {

        YearMonth currentYearMonth, ofYearMonth, parseYearMonth;
        currentYearMonth = YearMonth.now();//Se obtiene el YearMonth
actual
        System.out.printf("Este año-mes es %s y tiene %d días",
currentYearMonth, currentYearMonth.lengthOfMonth());
        ofYearMonth = YearMonth.of(2004, Month.FEBRUARY);//Obtiene una
instancia YearMonth a partir de un año y un mes
        System.out.printf("\n\nEl año-mes %s tuvo %d días",
ofYearMonth, ofYearMonth.lengthOfMonth());
        ofYearMonth = YearMonth.of(2010, Month.FEBRUARY);
        System.out.printf("\nEl año-mes %s tuvo %d días", ofYearMonth,
ofYearMonth.lengthOfMonth());
        ofYearMonth = YearMonth.of(2000, Month.FEBRUARY);
        System.out.printf("\nEl año-mes %s tuvo %d días", ofYearMonth,
ofYearMonth.lengthOfMonth());
        parseYearMonth = YearMonth.parse("2022-12");//Obtiene un
YearMonth a partir de una cadena
        System.out.printf("\n\nAño-mes obtenido a partir de una
cadena: %s", parseYearMonth);
    }

    public static void main(String[] args) {

        new ClassYearMonth().show();

    }

}
```

Salida por consola:

```
Este año-mes es 2022-02 y tiene 28 días

El año-mes 2004-02 tuvo 29 días
El año-mes 2010-02 tuvo 28 días
El año-mes 2000-02 tuvo 29 días

Año-mes obtenido a partir de una cadena: 2022-12
```

7. Clase MonthDay

Con la clase `MonthDay` del paquete `java.time` podemos generar objetos inmutables que representan la combinación de un día y un mes.

```
package tema9_Fechas;

import java.time.Month;
import java.time.MonthDay;

public class ClassMonthDay {

    public void show() {

        MonthDay currentMonthDay, ofMonthDay, parseMonthDay;
        currentMonthDay = MonthDay.now();//Se obtiene el MonthDay
actual
        System.out.printf("Este día es %s, es decir, día %d del mes
%d", currentMonthDay,
                currentMonthDay.getDayOfMonth(),
currentMonthDay.getMonthValue());
        ofMonthDay = MonthDay.of(Month.FEBRUARY, 29);//Obtiene una
instancia MonthDay a partir de un mes y un día
        System.out.printf("\n\nEl día %s del mes %s%s es válido para
el año 2004", ofMonthDay.getDayOfMonth(),
                ofMonthDay.getMonthValue(),
ofMonthDay.isValidYear(2004) ? "" : " no");
        System.out.printf("\nEl día %s del mes %s%s es válido para el
año 2010", ofMonthDay.getDayOfMonth(),
                ofMonthDay.getMonthValue(),
ofMonthDay.isValidYear(2010) ? "" : " no");
        parseMonthDay = MonthDay.parse("--12-03");//Obtiene un
MonthDay a partir de una cadena
        System.out.printf("\n\nMes-día obtenido a partir de una
cadena: %s", parseMonthDay);

    }

    public static void main(String[] args) {

        new ClassMonthDay().show();

    }

}
```

Salida por consola:

```
Este día es --02-07, es decir, día 7 del mes 2

El día 29 del mes 2 es válido para el año 2004
El día 29 del mes 2 no es válido para el año 2010

Mes-día obtenido a partir de una cadena: --12-03
```

8. Clase Year

Con la clase `Year` del paquete `java.time` podemos generar objetos inmutables que representan un año concreto.

```java
package tema9_Fechas;

import java.time.Year;

public class ClassYear {

    public void show() {

        Year currentYear, ofYear, parseYear;
        currentYear = Year.now();//Obtiene el año actual
        System.out.printf("Este año es %s y %s es bisiesto%n",
currentYear, currentYear.isLeap() ? "sí" : "no");
        ofYear = Year.of(2012);//Obtiene una instancia Year a partir
de un año
        System.out.printf("\nEl año %s es %s al año actual", ofYear,
                ofYear.isBefore(currentYear) ? "anterior" :
"posterior");
        parseYear = Year.parse("2021");//Obtiene un Year a partir de
una cadena
        System.out.printf("\n\nAño obtenido a partir de una cadena:
%s", parseYear);

    }

    public static void main(String[] args) {

        new ClassYear().show();

    }

}
```

Salida por consola:

```
Este año es 2022 y no es bisiesto

El año 2012 es anterior al año actual

Año obtenido a partir de una cadena: 2021
```

9. Clase LocalTime

Con la clase `LocalTime` del paquete `java.time` podemos generar objetos inmutables que representan una hora concreta.

```
package tema9_Fechas;

import java.time.LocalTime;

public class ClassLocalTime {

    public void show() {

        LocalTime currentTime, meetingTime, parseTime;
        currentTime = LocalTime.now();//Se obtiene la hora actual
        System.out.printf("Hora actual: %s", currentTime);
        System.out.printf("\nHora: %s Minutos: %s Segundos: %s
Nanosegundos: %s", currentTime.getHour(),
                currentTime.getMinute(), currentTime.getSecond(),
currentTime.getNano());
        meetingTime = LocalTime.of(10, 15);//Obtiene una instancia de
LocalTime indicando hora y minutos
        System.out.printf("\n\nHora de la reunión: %s", meetingTime);
        parseTime = LocalTime.parse("11:45:30");//Obtiene un LocalTime
a partir de una cadena
        System.out.printf("\n\nHora obtenida a partir de una cadena:
%s", parseTime);

    }

    public static void main(String[] args) {

        new ClassLocalTime().show();

    }

}
```

Salida por consola:

```
Hora actual: 08:56:53.868514995
Hora: 8 Minutos: 56 Segundos: 53 Nanosegundos: 868514995

Hora de la reunión: 10:15

Hora obtenida a partir de una cadena: 11:45:30
```

10. Clase LocalDateTime

Con la clase `LocalDateTime` del paquete `java.time` podemos generar objetos inmutables que representan una fecha y hora concretas.

```java
package tema9_Fechas;

import java.time.LocalDateTime;

public class ClassLocalDateTime {

    public void show() {

        LocalDateTime currentDateTime, meetingDateTime, parseDateTime;
        currentDateTime = LocalDateTime.now();//Se obtiene la fecha y
hora actual
        System.out.printf("Fecha y hora actual: %s", currentDateTime);
        System.out.printf("\nDía del año: %s",
currentDateTime.getDayOfYear());
        System.out.printf("\nDía de la semana: %s",
currentDateTime.getDayOfWeek());
        System.out.printf("\nHora: %s", currentDateTime.getHour());
        System.out.printf("\nNanosegundos: %s",
currentDateTime.getNano());
        meetingDateTime = LocalDateTime.of(2022, 11, 16, 9, 30,
45);//Obtiene una instancia de LocalDateTime a partir de un
año,mes,día,hora,minutos y segundos
        System.out.printf("\n\nLa reunión es %s", meetingDateTime);
        System.out.printf("\nFecha de la reunión: %s",
meetingDateTime.toLocalDate());
        System.out.printf("\nHora de la reunión: %s",
meetingDateTime.toLocalTime());
        parseDateTime = LocalDateTime.parse("2022-12-
03T10:15:30");//Obtiene un LocalDateTime a partir de una cadena
        System.out.printf("\n\nFecha y hora obtenida a partir de una
cadena: %s", parseDateTime);

    }

    public static void main(String[] args) {

        new ClassLocalDateTime().show();
```

```
    }

}
```

Salida por consola:

```
Fecha y hora actual: 2022-02-04T13:48:19.851351656
Día del año: 35
Día de la semana: FRIDAY
Hora: 13
Nanosegundos: 851351656

La reunión es 2022-11-16T09:30:45
Fecha de la reunión: 2022-11-16
Hora de la reunión: 09:30:45

Fecha y hora obtenida a partir de una cadena: 2022-12-03T10:15:30
```

11. Clase DateTimeFormatter

La clase `DateTimeFormatter` del paquete `java.time.format` se utiliza para dar formato a las fechas.

Se puede indicar el formato con:

- formatos predefinidos.
- creando patrones que se basan en una secuencia de letras y símbolos.

En la API podemos encontrar dichos formatos predefinidos y el significado de las letras y símbolos para crear un patrón.

Los métodos `parse` que hemos visto en clases anteriores están sobrecargados para utilizarlos con un formato concreto.

```
package tema9_Fechas;

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

public class ClassDateTimeFormatter {

    public void show() {

        LocalDateTime localDateTime;

        //Formato predefinido ISO_ORDINAL_DATE: año y día del año
```

```
        DateTimeFormatter formatter =
DateTimeFormatter.ISO_ORDINAL_DATE;
        System.out.printf("Formato predefinido ISO_ORDINAL_DATE: %s",
formatter.format(LocalDate.now()));

        //Creación de patrones basados en secuencia de letras y
símbolos
        formatter = DateTimeFormatter.ofPattern("d M y");
        System.out.printf("\n\nPatrón \"%s\": %s", "d M y",
formatter.format(LocalDate.now()));
        formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        System.out.printf("\nPatrón \"%s\": %s", "dd/MM/yyyy",
formatter.format(LocalDate.now()));
        formatter = DateTimeFormatter.ofPattern("d MMM yy");
        System.out.printf("\nPatrón \"%s\": %s", "d MMM yy",
formatter.format(LocalDate.now()));
        formatter = DateTimeFormatter.ofPattern("dd-MMMM-yyyy");
        System.out.printf("\nPatrón \"%s\": %s", "dd-MMMM-yyyy",
formatter.format(LocalDate.now()));

        //Uso de método parse con formato
        formatter = DateTimeFormatter.ofPattern("d MMMM uu HH:mm:ss");
        localDateTime = LocalDateTime.parse("7 febrero 22 10:01:30",
formatter);
        System.out.printf("\n\nFecha y hora obtenida a partir de una
cadena y un formato: %s", localDateTime);

    }

    public static void main(String[] args) {

        new ClassDateTimeFormatter().show();

    }

}
```

Salida por consola:

```
Formato predefinido ISO_ORDINAL_DATE: 2022-038

Patrón "d M y": 7 2 2022
Patrón "dd/MM/yyyy": 07/02/2022
Patrón "d MMM yy": 7 feb 22
Patrón "dd-MMMM-yyyy": 07-febrero-2022

Fecha y hora obtenida a partir de una cadena y un formato: 2022-02-
07T10:01:30
```

12. Enum ChronoUnit

En este enum se definen un conjunto estándar de unidades de períodos de fecha, como por ejemplo, *DAYS, MONTHS, WEEKS, YEARS, HOURS, MINUTES, SECONDS*, etc.

```
package tema9_Fechas;

import java.time.LocalDateTime;
import java.time.Month;
import java.time.temporal.ChronoUnit;

public class EnumChronoUnit {

    public void show() {

        LocalDateTime currentDateTime, concertDateTime;
        currentDateTime = LocalDateTime.now();
        concertDateTime = LocalDateTime.of(2022, Month.AUGUST, 13, 21,
30);
        System.out.printf("Quedan para el concierto: %d meses o %d
días o %d horas o %d minutos o %d segundos",
                ChronoUnit.MONTHS.between(currentDateTime,
concertDateTime),
                ChronoUnit.DAYS.between(currentDateTime,
concertDateTime),
                ChronoUnit.HOURS.between(currentDateTime,
concertDateTime),
                ChronoUnit.MINUTES.between(currentDateTime,
concertDateTime),
                ChronoUnit.SECONDS.between(currentDateTime,
concertDateTime));

    }

    public static void main(String[] args) {

        new EnumChronoUnit().show();

    }

}
```

Salida por consola:

```
Quedan para el concierto: 6 meses o 187 días o 4496 horas o 269792
minutos o 16187549 segundos
```

13. Clase Period

Esta clase modela un período de tiempo en términos de años, meses y días.

```
package tema9_Fechas;

import java.time.LocalDate;
import java.time.Period;

public class ClassPeriod {

    public void show() {

        Period period;
        LocalDate oldDate, currentDate;
        oldDate = LocalDate.of(1988, 9, 28);
        currentDate = LocalDate.now();
        period = Period.between(oldDate, currentDate);//Obtiene una
instancia de Period a partir de dos fechas
        System.out.printf("Período entre %s y %s: son %d años, %d
meses y %d días. ", oldDate, currentDate,
                period.getYears(), period.getMonths(),
period.getDays());
        System.out.printf("En total son %d meses.",
period.toTotalMonths());
        period = Period.ofWeeks(50);//Obtiene una instancia de Period
de 50 semanas
        System.out.printf("\n\n50 semanas son %d días.",
period.getDays());

    }

    public static void main(String[] args) {

        new ClassPeriod().show();

    }

}
```

Salida por consola:

```
Período entre 1988-09-28 y 2022-02-07: son 33 años, 4 meses y 10 días.
En total son 400 meses.

50 semanas son 350 días.
```

14. Clase Duration

Esta clase modela un período de tiempo en términos de segundos y nanosegundos. Se puede acceder a ella utilizando otras unidades basadas en la duración, como los minutos y las horas.

```java
package tema9_Fechas;

import java.time.Duration;

public class ClassDuration {

    public void show() {

        Duration duration;
        duration = Duration.ofMinutes(10);
        System.out.printf("Minutos: %d", duration.toMinutes());
        System.out.printf("\nSegundos: %d", duration.toSeconds());
        System.out.printf("\nNanosegundos: %d", duration.toNanos());

    }

    public static void main(String[] args) {

        new ClassDuration().show();

    }

}
```

Salida por consola:

```
Minutos: 10
Segundos: 600
Nanosegundos: 600000000000
```

15. Enum TimeUnit

Con este enum se representan duraciones de tiempo en una determinada unidad. Proporciona métodos de utilidad para convertir entre unidades y para realizar operaciones de temporización y retraso.

```java
package tema9_Fechas;

import java.util.concurrent.TimeUnit;

public class EnumTimeUnit {
```

```
    public void show() {

        try {
            for (int i = 0; i < 5; i++) {
                System.out.printf("Iteración %d\n", i);
                TimeUnit.SECONDS.sleep(2);//Se duerme durante 2
segundos
            }
        } catch (InterruptedException e) {
            System.out.println("Interrupción mientras se duerme");
        }

    }

    public static void main(String[] args) {

        new EnumTimeUnit().show();

    }

}
```