DESARROLLO DE APLICACIONES WEB

Anexo II - Unidad 12

Intervalos de tiempo / Fechas y horas

1r DAW

IES La Mola de Novelda

Departament d'informàtica

Professor: Joan Carbonell Picó

Índice

Anexo II / Unidad 12: Intervalos de tiempo

1.- INTERVALOS DE TIEMPO

Cuando queramos medir un periodo de tiempo corto durante la ejecución de un programa, podemos usar el método ***System.currentTimeMillis()*** que, al leer del reloj del procesador, devuelve la hora actual como valor ***long***, con el número de milisegundos transcurridos entre las 0 horas del 1 de enero de 1970 y el instante actual. El interés de esta función está en llamarla en dos instantes diferentes y calcular el tiempo transcurrido restando los dos valores devueltos.

Cuando el intervalo de tiempo que queremos medir es demasiado pequeño para obtenerlo en milisegundos, podemos usar el método ***System.nanoTime()***, que devuelve el tiempo transcurrido desde un instante de referencia arbitrario hasta el presente en nanosegundos. La exactitud del valor devuelto va a depender de la arquitectura de nuestro ordenador

- Ejemplo: Implementa el código para medir el tiempo que se emplea en generar 100 números aleatorios.

```
long principio = System.nanoTime();
for(int i=0; i<100;i++){
   System.out.println(Math.random());
}
double fin = System.nanoTime();
System.out.println("Tiempo: "+(fin-principio)+" nanosegundos");
System.out.println("Tiempo: "+(fin-principio)/1000+" microsegundos");
System.out.println("Tiempo: "+(fin-principio)/1000000+" milisegundos");
System.out.println("Tiempo: "+(fin-principio)/1000000000+" segundos");
```

A veces nos interesa detener la ejecución del programa durante un intervalo de tiempo preciso. Para eso se usa el método ***sleep()*** de la clase ***Thread***. Este método tiene dos implementaciones sobrecargadas:

static void sleep (long milisegundos) → Detiene la ejecución durante los milisegundos que se les pasan como argumento.

static void sleep (long milisegundos, int nanosegundos) → Añade a la pausa los nanosegundos que se le pasan como segundo argumento

- Ejemplo: Si queremos que la ejecución se detenga 4000 milisegundos (4 segundos) y 100 nanosegundos pondremos:

```
Thread.sleep(4000, 100);
```

2.- FECHAS Y HORAS

2.1.-INTRODUCCIÓN

La implementación de las horas y las fechas en Java se basa en el calendario Gregoriano. Para disponer de las clases que vamos a ver deberemos importarlas con el paquete ***java.time*** y diversos subpaquetes, así como ***java.util.Locale***.

2.2.-LOCALDATE

Para fechas donde no tenemos en cuenta zonas horarias, usaremos ***LocalDate***. Los objetos ***LocalDate*** se crean con la función estática ***of()*** de la clase ***LocalDate***.

- Ejemplo: Creación de un objeto con fecha 12 de febrero de 2016

```
LocalDate f1 = LocalDate.of(2016,2,12);
System.out.println(f1);
```

Para el mes podemos utilizar el tipo enumerado ***Month***. En vez del mes 2 podemos poner ***Month.FEBRUARY***. Los posibles valores son los meses en inglés (***JANUARY***, ***FEBRUARY***, …). Como vemos en el ejemplo, un objeto ***LocalDate*** se puede mostrar por consola directamente , ya que tiene implementado el método ***toString()***.

Si necesitamos un ***LocalDate*** con la fecha actual, leída del sistema, llamaremos al método ***now()***.

```
LocalDate fechaActual = LocalDate.now();
```

A partir de ella, podemos obtener el día de la semana, que es un objeto de la clase ***DayOfWeek***:

```
DayOfWeek diaSemana = fechaActual.getDayOfWeek();
System.out.println(diaSemana);
```

Aparecerá por pantalla el día de la semana en inglés. ***DayOfWeek*** es otro tipo enumerado con los valores ***MONDAY***, ***TUESDAY***, … de los días de la semana en inglés.

Por lo tanto, si queremos a partir de un ***LocalDate***, obtener el día del mes, les mes del año o el año, se usan los métodos:

```
int getDayOfMonth ( ) // devuelve el día del mes.
Month getMonth ( ) // devuelve un valor del tipo enumerado Month
int getMonthValue ( ) // devuelve el mes como número del 1 al 12
```

int getYear ( ) // devuelve el año

Las fechas se pueden incrementar con el método ***plus()***.

```
LocalDate fechaActual = LocalDate.now();
System.out.println(fechaActual);                          2023-04-28
LocalDate f2 = fechaActual.plus(3, ChronoUnit.DAYS);      2023-05-01
System.out.println(f2);
```

El tipo enumerado ***ChronoUnit*** tiene como valores distintas unidades de medida del tiempo, como ***DAYS***, ***MONTHS***, ***YEARS*** y ***WEEKS*** entre otros. En el ejemplo vemos que el primer parámetro de ***plus()*** indica el número de unidades que queremos añadir al segundo parámetro. También podemos decrementar con el método ***minus()***.

```
LocalDate fechaActual = LocalDate.now();
System.out.println(fechaActual);                          2023-04-28
LocalDate f2 = fechaActual.plus(3, ChronoUnit.DAYS);      2023-05-01
LocalDate f3 = fechaActual.minus(2,ChronoUnit.YEARS);     2021-04-28
System.out.println(f2);
System.out.println(f3);
```

También podemos comparar dos fechas además de otras acciones con los métodos:

boolean equals ( ) // true si son iguales.
int compareTo ( ) // compara con orden cronológico
boolean isLeapYear() // true si el año de la fecha que lo invoca es bisiesto

Para calcular el periodo transcurrido entre dos fechas, usamos el método:

Period until (LocalDate otraFecha)

Donde ***Period*** es una clase cuyos objetos representan un periodo de tiempo, que consta de años, meses y días. Para obtener los días, meses o años de un periodo usaremos el método:

long get (TemporalUnit unidad)

Donde unidad puede ser ***ChronoUnit.DAYS***, ***ChronoUnit.MONTHS*** o ***ChronoUbnit.YEARS***.

- Ejemplo: El periodo devuelto consiste en 14 meses (1 año y 2 meses). También podemos obtener el periodo completo expresado en meses con ***periodo.toTotalMonths()***.

```
LocalDate fechaActual = LocalDate.now();                  2023-04-28
System.out.println(fechaActual);                          1
LocalDate f4 = fechaActual.plus(14,ChronoUnit.MONTHS);    2
Period periodo = fechaActual.until(f4);                   14
System.out.println(periodo.get(ChronoUnit.YEARS));
```

```
System.out.println(periodo.get(ChronoUnit.MONTHS));
System.out.println(periodo.toTotalMonths());
```

A la hora de mostrar una fecha, o de obtenerla a partir de una cadena, se usa la clase ***DateTimeFormatter***, cuyos objetos definen un formato de fecha y/o hora. El método o***fPattern()*** genera el formato a partir de un patrón que se le pasa como parámetro.

- Ejemplo: Si queremos mostrar por pantalla una fecha con un formato del tipo 12-02-2020 crearemos el siguiente formato (el formateador se aplica a una fecha para generar la cadena que la representa) :

```
DateTimeFormatter formato1 = DateTimeFormatter.ofPattern("dd-MM-yy");
LocalDate fecha1 = LocalDate.of(2016,2,12);
String cadenaFecha = fecha1.format(formato1);
System.out.println(cadenaFecha);
DateTimeFormatter formato2 = DateTimeFormatter.ofPattern("dd 'del mes' MM 'del año' yyyy");
cadenaFecha = fecha1.format(formato2);
System.out.println(cadenaFecha);
```

```
12-02-16
12 del mes 02 del año 2016
```

- Ejemplo: Si queremos un formato estándar adaptado a un país concreto. El método ***ofLocalizedDate()*** permite escoger un formato más o menos detallado (***FormatStyle.FULL***, ***FormatStyle.SHORT***, ***FormatStyle.LONG*** o ***FormatStyle.MEDIUM***), mientras que ***withLocale()*** nos adapta el formato al país que queramos de entre una lista (***Locale.US***, ***Locale.ITALY***, ...) o bien adopta el formato del país del sistema con ***Locale.getDefault()***.

```
DateTimeFormatter formato3 = DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL).withLocale(Locale.getDefault());
String cadenaFecha = fecha1.format(formato3);
System.out.println(cadenaFecha);FULL).withLocale(Locale.getDefault());
```

```
divendres, 12 de febrer de 2016
```

Un formateador también sirve para analizar una cadena y convertirla en un objeto ***LocalDate***, haciendo el trabajo inverso al que acabamos de ver. Para ello se usa el método estático de ***LocalDate***:

static LocalDate parse (String cadEntrada, DateTimeFormatter formateador)

- Ejemplo: Si queremos generar la fecha 15 de junio de 2010, a partir de la cadena “15-06-2010” con el formato definido en formato1 (dd-MM-yyyy)

```
DateTimeFormatter formato1 = DateTimeFormatter.ofPattern("dd-MM-yyyy");
cadenaFecha = "15-06-2010";
```

```
LocalDate fecha5 = LocalDate.parse(cadenaFecha,formato1);
System.out.println(fecha5);
```

```
2010-06-15
```

De esta manera podremos leer fechas a partir de cadenas introducidas por teclado. Si la cadena no se ajusta al formato que le hemos pasado como segundo parámetro, saltará la excepción ***DateTimeParseExcepcion***:

```
try{
        cadenaFecha = "15-06-2010";
        LocalDate fecha5 = LocalDate.parse(cadenaFecha,formato1);
        System.out.println(fecha5);
}catch (DateTimeParseException dtpe){
        System.out.println("Cadena no se ajusta al formato");
}
```

```
Cadena no se ajusta al formato
```

3.- LOCALTIME

Para el tratamiento de las horas en Java se usa la clase ***LocalTime***, que tiene un conjunto de métodos similares a ***LocalDate***, pero con campos distintos. En vez de días, meses y años, trabaja con horas, minutos, segundos y nanosegundos.

Como ocurre con ***LocalDate***, los objetos ***LocalTime*** se pueden crear con los siguientes métodos estáticos sobrecargado de la clase ***LocalTime***:

static LocalTime of(int hora, int minuto, int segundo, int nanosegundo)
static LocalTime of(int hora, int minuto, int segundo)

- Ejemplo: Creación de un LocalTime a partir de una hora.

```
LocalTime time1 = LocalTime.of(12,30,45);
System.out.println(time1);
```

```
12:30:45
```

También podemos leer la hora del sistema con la función **static LocalTime now()**. A partir de un objeto ***LocalTime***, se pueden obtener sus distintos campos con **int getHour()**, **int getMinute()**, **int getSecond()**. Además, ***LocalTime*** dispone de los métodos de comparación **boolean equals(Object otraHora)**, **int compareTo(Object otraHora)**, así como los métodos de incremento **LocalTime plus(long cantidad, TemporalUnit unidad)** y decremento **LocalTime minus(long cantidad, TemporalUnit unidad).**

Los periodos comprendidos entre dos objetos ***LocalTime*** se calculan con el método:

Longs until(LocalTime otraHora, TemporalUnit unidad)

Por último, para imprimir una hora o generar un objeto ***LocalTime*** a partir de una cadena, se

usa la misma clase formateadora ***DateTimeFormatter***, así cono los métodos ***ofPattern()*** y ***parse()***.

- Ejemplo:

```
LocalTime time1 = LocalTime.of(12,30,45);
System.out.println(time1);
LocalTime time2 = LocalTime.now();
System.out.println(time2);
System.out.println(time2.getMinute());
LocalTime time3 = time1.plus(45,ChronoUnit.MINUTES);
System.out.println(time3);
long diferencia = time1.until(time3,ChronoUnit.MINUTES);
System.out.println(diferencia);
DateTimeFormatter formato4 = DateTimeFormatter.ofPattern("HH.mm.ss");
String cadenaTime = formato4.format(time3);
System.out.println(cadenaTime);
LocalTime time4 = LocalTime.parse(cadenaTime,formato4);
System.out.println(time4);
```

```
12:30:45
19:08:37.247532238
8
13:15:45
45
13.15.45
13:15:45
```

Para terminar, existe una tercera clase que combina las fechas y las horas, ***LocalDateTime***. Sus métodos son los mismos que los de ***LocalDate*** y ***LocalTime***, y emplea también ***DateTimeFormatter***.