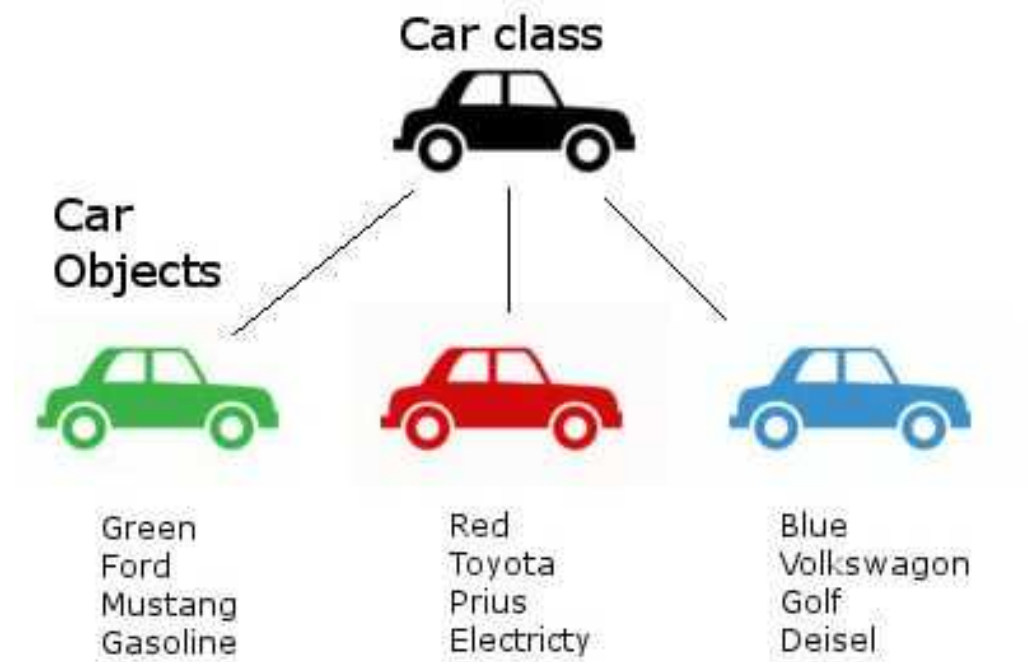


Anexo UD05: Utilización avanzada de clases



1. Wrappers (Envoltorios)

- 1. 1. Métodos `valueOf()`
- 1. 2. Métodos `parseXxx()`
- 1. 3. Métodos `toString()`
- 1. 4. Métodos `toXxxxString()` (Binario, Hexadecimal y Octal)

2. Clase `Date`

- 2. 1. Clase `GregorianCalendar`
- 2. 2. Paquete `java.time`
 - 2. 2. 1. `LocalDate`
 - 2. 2. 2. `LocalTime`
 - 2. 2. 3. `LocalDateTime`
 - 2. 2. 4. `Duration`
 - 2. 2. 5. `Period`
- 2. 3. `ChronoUnit`
- 2. 4. Introducir fecha como Cadena
- 2. 5. Manipulación
- 2. 6. Operaciones
 - 2. 6. 1. Operaciones con `LocalDate`
 - 2. 6. 2. Operaciones con `LocalTime`
 - 2. 6. 3. Operaciones con `LocalDateTime`
- 2. 7. Formatos
 - 2. 7. 1. Día de la Semana

3. Ejemplo Anexo UD05

- 3. 1. `Anexo1Wrappers`
- 3. 2. `Anexo2Date`

4. Fuentes de información

1. Wrappers (Envoltorios)

Los wrappers permiten "envolver" datos primitivos en objetos, también se llaman clases contenedoras. La diferencia entre un tipo primitivo y un wrapper es que este último es una clase y por tanto, cuando trabajamos con wrappers estamos trabajando con objetos.

Como son objetos debemos tener cuidado en el paso como parámetro en métodos ya que en el wrapper se realiza por referencia.

Una de las principales ventajas del uso de wrappers son la facilidad de conversión entre tipos primitivos y cadenas.

Hay una clase contenedora por cada uno de los tipos primitivos de Java. Los datos primitivos se escriben en minúsculas y los wrappers se escriben con la primera letra en mayúsculas.

Tipo primitivo	Wrapper asociado
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Char
boolean	Boolean

Cada clase wrapper tiene dos constructores, uno se le pasa por parámetro el dato de tipo primitivo y otro se le pasa un `String`.

Para wrapper `Integer`:

```
1 Integer(int)
2 Integer(String)
```

Ejemplo:

```
1 Integer i1 = new Integer(42);
2 Integer i2 = new Integer ("42");
3 Float f1 = new Float(3.14f);
4 Float f2 = new Float ("3.14f");
```

Antiguamente, una vez asignado un valor a un objeto o wrapper `Integer`, este no podía cambiarse. Actualmente e internamente realizar un apoyo en variables y wrappers internos para poder variar el valor de un wrapper.

Ejemplo:

```
1 Integer y = new Integer(567);           //Crea el objeto
2 y++;                                   //Lo desenvuelve, incrementa y lo vuelve a
   envolver
3 System.out.println("Valor: " + y);     //Imprime el valor del Objeto y
```

Los wrapper disponen de una serie de métodos que permiten realizar funciones de conversión de datos. Por ejemplo, el wrapper `Integer` dispone de los siguientes métodos:

Método	Descripción
<code>Integer(int)</code> <code>Integer(String)</code>	Constructores
<code>byteValue()</code> <code>shortValue()</code> <code>intValue()</code> <code>longValue()</code> <code>doubleValue()</code> <code>floatValue()</code>	Funciones de conversión con datos primitivos
<code>Integer decode(String)</code> <code>Integer parseInt(String)</code> <code>Integer parseInt(String, int)</code> <code>Integer valueOf(String)</code> <code>String toString()</code>	Conversión a String
<code>String toBinaryString(int)</code> <code>String toHexString(int)</code> <code>String toOctalString(int)</code>	Conversión a otros sistemas de numeración
<code>MAX_VALUE</code> , <code>MIN_VALUE</code> , <code>TYPE</code>	Constantes

1.1. Métodos `valueOf()`

El método `valueOf()` permite crear objetos wrapper y se le pasa un parámetro `String` y opcionalmente otro parámetro que indica la base en la que será representado el primer parámetro.

Ejemplo:

```

1 // Convierte el 101011 (base 2) a 43 y le asigna el valor al objeto Integer i3
2 Integer i3 = Integer.valueOf("101011", 2);
3 System.out.println(i3);
4
5 // Asigna 3.14 al objeto Float f3
6 Float f3 = Float.valueOf("3.14f");
7 System.out.println(f3);

```

Métodos `xxxValue()`.

Los métodos `xxxValue()` permiten convertir un wrapper en un dato de tipo primitivo y no necesitan argumentos.

Ejemplo:

```

1 Integer i4 = 120; // Crea un nuevo objeto wrapper
2 byte b = i4.byteValue(); // Convierte el valor de i4 a un primitivo byte
3 short s1 = i4.shortValue(); // Otro de los métodos de Integer
4 double d = i4.doubleValue(); // Otro de los métodos xxxValue de Integer
5 System.out.println(s1); // Muestra 120 como resultado
6
7 Float f4 = 3.14f; // Crea un nuevo objeto wrapper
8 short s2 = f4.shortValue(); // Convierte el valor de f4 en un primitivo short
9 System.out.println(s2); // El resultado es 3 (truncado, no redondeado)

```

1.2. Métodos `parseXxxx()`

Los métodos `parseXxxx()` permiten convertir un wrapper en un dato de tipo primitivo y le pasamos como parámetro el `String` con el valor que deseamos convertir y opcionalmente la base a la que convertiremos el valor (2, 8, 10 o 16).

Ejemplo:

```

1 double d4 = Double.parseDouble("3.14"); // Convierte un String a primitivo
2 System.out.println("d4 = " + d4); // El resultado será d4 = 3.14
3 long l2 = Long.parseLong("101010", 2); // un String binario a primitivo
4 System.out.println("l2 = " + l2); // El resultado es l2 = 42

```

1.3. Métodos `toString()`

El método `toString()` permite retornar un `String` con el valor primitivo que se encuentra en el objeto contenedor. Se le pasa un parámetro que es el wrapper y opcionalmente para `Integer` y `Long` un parámetro con la base a la que convertiremos el valor (2, 8, 10 o 16).

Ejemplo:

```

1 Double d1 = new Double("3.14");
2 System.out.println("d1 = " + d1.toString() ); // El resultado es d = 3.14
3 String d2 = Double.toString(3.14); // d2 = "3.14"
4 System.out.println("d2 = " + d2); // El resultado es d = 3.14
5 String s3 = "hex = " + Long.toString(254, 16); // s3 = "hex = fe"
6 System.out.println("s3 = " + s3); // El resultado es s3 = 3.14

```

1.4. Métodos `toXXXXString()` (Binario, Hexadecimal y Octal)

Los métodos `toXXXXString()` permiten a las clases contenedoras `Integer` y `Long` convertir números en base 10 a otras bases, retornando un `String` con el valor primitivo que se encuentra en el objeto contenedor.

Ejemplo:

```

1 String s4 = Integer.toHexString(254); // Convierte 254 a hex
2 System.out.println("254 es " + s4); // Resultado: "254 es fe"
3 String s5 = Long.toOctalString(254); // Convierte 254 a octal
4 System.out.println("254(oct) = " + s5); // Resultado: "254(oct) = 376"

```

Para resumir, los métodos esenciales para las conversiones son:

- `primitive xxxValue()` – Para convertir de `Wrapper` a `primitive`
- `primitive parseXxx(String)` – Para convertir un `String` en `primitive`
- `Wrapper valueOf(String)` – Para convertir `String` en `Wrapper`

2. Clase Date

La clase `Date` es una utilidad contenida en el paquete `java.util` y permiten trabajar con fechas y horas. La fechas y hora se almacenan en un entero de tipo `Long` que almacena los milisegundos transcurridos desde el 1 de Enero de de 1970 que se obtienen con `getTime()`. (Importamos `java.util.Date`).

Ejemplo:

```
1 Date fecha = new Date(2021, 9, 19);
2 System.out.println(fecha);           //Mon Sep 19 00:00:00 CEST 3921
3 System.out.println(fecha.getTime()); //615901464000000
```

2.1. Clase `GregorianCalendar`

Para utilizar fechas y horas se utiliza la clase `GregorianCalendar` que dispone de variable enteras como: `DAY_OF_WEEK`, `DAY_OF_MONTH`, `YEAR`, `MONTH`, `HOUR`, `MINUTE`, `SECOND`, `MILLISECOND`, `WEEK_OF_MONTH`, `WEEK_OF_YEAR`, ... (Importamos Clase `java.util.Calendar` y `java.util.GregorianCalendar`)

Ejemplo 1:

```
1 Calendar calendar = new GregorianCalendar(2021, 9, 19);
2 System.out.println(calendar.getTime()); //Sun Sep 19 00:00:00 CEST 2021
```

Ejemplo 2:

```
1 Date d = new Date();
2 GregorianCalendar c = new GregorianCalendar();
3 System.out.println("Fecha: "+d); //Fecha: Thu Aug 19 20:06:14 CEST 2021
4 System.out.println("Info: "+c); //Info:
5 //java.util.GregorianCalendar[time=1629396374723,areFieldsSet=true
6 //,areAllFieldsSet=true
7 //,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Europe/Madrid",offset=3600000
8 //,dstSavings=3600000,useDaylight=true,transitions=163
9 //,lastRule=java.util.SimpleTimeZone[id=Europe/Madrid,offset=3600000
10 //,dstSavings=3600000,useDaylight=true,startYear=0,startMode=2,startMonth=2
11 //,startDay=-1,startDayOfWeek=1,startTime=3600000,startTimeMod2.1e=2,endMode=2
12 //,endMonth=9,endDay=-1,endDayOfWeek=1,endTime=3600000,endTimeMode=2]]
13 //,firstDayOfWeek=2,minimalDaysInFirstWeek=4,ERA=1,YEAR=2021,MONTH=7,WEEK_OF_YEA
14 //,WEEK_OF_MONTH=3,DAY_OF_MONTH=19,DAY_OF_YEAR=231,DAY_OF_WEEK=5
15 //,DAY_OF_WEEK_IN_MONTH=3,AM_PM=1,HOUR=8,HOUR_OF_DAY=20,MINUTE=6,SECOND=14
16 //,MILLISECOND=723,ZONE_OFFSET=3600000,DST_OFFSET=3600000]
17 c.setTime(d);
18 System.out.print(c.get(Calendar.DAY_OF_MONTH));
19 System.out.print("/");
```

```

20 System.out.print(c.get(Calendar.MONTH)+1);
21 System.out.print("/");
22 System.out.println(c.get(Calendar.YEAR)+1); //19/9/2022

```

2.2. Paquete `java.time`

El paquete `java.time` dispone de las clases `LocalDate`, `LocalTime`, `LocalDateTime`, `Duration` y `Period` para trabajar con fechas y horas.

Estas clases no tienen constructores públicos, y por tanto, no se puede usar `new` para crear objetos de estas clases. Necesitas usar sus métodos `static` para instanciarlas.

No es válido llamar directamente al constructor usando `new`, ya que no tienen un constructor público.

Ejemplo:

```

1 LocalDate d = new LocalDate(); //NO compila

```

2.2.1. `LocalDate`

`LocalDate` representa una fecha determinada. Haciendo uso del método `of()`, esta clase puede crear un `LocalDate` teniendo en cuenta el año, mes y día. Finalmente, para capturar el `LocalDate` actual se puede usar el método `now()`:

Ejemplo:

```

1 LocalDate date = LocalDate.of(1989, 11, 11); //1989-11-11
2 System.out.println(date.getYear()); //1989
3 System.out.println(date.getMonth()); //NOVEMBER
4 System.out.println(date.getDayOfMonth()); //11
5 date = LocalDate.now();
6 System.out.println(date); //2021-08-19

```

2.2.2. `LocalTime`

`LocalTime`, representa un tiempo determinado. Haciendo uso del método `of()`, esta clase puede crear un `LocalTime` teniendo en cuenta la hora, minuto, segundo y nanosegundo. Finalmente, para capturar el `LocalTime` actual se puede usar el método `now()`.

```

1 LocalTime time = LocalTime.of(5, 30, 45, 35); //05:30:45:35
2 System.out.println(time.getHour()); //5
3 System.out.println(time.getMinute()); //30
4 System.out.println(time.getSecond()); //45
5 System.out.println(time.getNano()); //35
6 time = LocalTime.now();
7 System.out.println(time); //20:13:53.118044

```


2.2.3. LocalDateTime

`LocalDateTime`, es una clase compuesta, la cual combina las clases anteriormente mencionadas `LocalDate` y `LocalTime`. Podemos construir un `LocalDateTime` haciendo uso de todos los campos (año, mes, día, hora, minuto, segundo, nanosegundo).

Ejemplo:

```
1 | LocalDateTime dateTime = LocalDateTime.of(1989, 11, 11, 5, 30, 45, 35);
```

También, se puede crear un objeto `LocalDateTime` basado en los tipos `LocalDate` y `LocalTime`, haciendo uso del método `of()` (`LocalDate date`, `LocalTime time`):

Ejemplo:

```
1 | LocalDate date = LocalDate.of(1989, 11, 11);
2 | LocalTime time = LocalTime.of(5, 30, 45, 35);
3 | LocalDateTime dateTime = LocalDateTime.of(date, time);
4 | LocalDateTime dateTime = LocalDateTime.now();
```

2.2.4. Duration

`Duration`, hace referencia a la diferencia que existe entre dos objetos de tiempo. La duración denota la cantidad de tiempo en horas, minutos y segundos.

Ejemplo:

```
1 | LocalTime localTime1 = LocalTime.of(12, 25);
2 | LocalTime localTime2 = LocalTime.of(17, 35);
3 | Duration duration1 = Duration.between(localTime1, localTime2);
4 | System.out.println(duration1); //PT5H10M
5 | System.out.println(duration1.toDays()); //0
6 |
7 | LocalDateTime localDateTime1 = LocalDateTime.of(2016, Month.JULY, 18, 14, 13);
8 | LocalDateTime localDateTime2 = LocalDateTime.of(2016, Month.JULY, 20, 12, 25);
9 | Duration duration2 = Duration.between(localDateTime1, localDateTime2);
10 | System.out.println(duration2); //PT46H12M
11 | System.out.println(duration2.toDays()); //1
```

También, se puede crear `Duration` basado en los métodos `ofDays(long days)`, `ofHours(long hours)`, `ofMillis(long millis)`, `ofMinutes(long minutes)`, `ofNanos(long nanos)`, `ofSeconds(long seconds)`.

Ejemplo:

```
1 | Duration duracion3 = Duration.ofDays(1);
2 | System.out.println(duracion3); //PT24H
3 | System.out.println(duracion3.toDays()); //1
```

2.2.5. Period

`Period`, hace referencia a la diferencia que existe entre dos fechas. Esta clase denota la cantidad de tiempo en años, meses y días.

```
1 LocalDate localDate1 = LocalDate.of(2016, 7, 18);
2 LocalDate localDate2 = LocalDate.of(2016, 7, 20);
3 Period periodo1 = Period.between(localDate1, localDate2);
4 System.out.println(periodo1); //P2D
```

Se puede crear `Period` basado en el método `of(int years, int months, int days)`. En el siguiente ejemplo, se crea un período de 1 año 2 meses y 3 días:

```
1 Period periodo2 = Period.of(1, 2, 3);
2 System.out.println(periodo2); //P1Y2M3D
```

Se puede crear `Period` basado en los métodos `ofDays(int days)`, `ofMonths(int months)`, `ofWeeks(int weeks)`, `ofYears(int years)`.

Ejemplo:

```
1 Period periodo3 = Period.ofYears(1);
2 System.out.println(periodo3); //P1Y
```

2.3. ChronoUnit

Permite devolver el tiempo transcurrido entre dos fechas en diferentes formatos (`DAYS`, `MONTHS`, `YEARS`, `HOURS`, `MINUTES`, `SECONDS`, ...). Debemos importar la clase `time.temporal.ChronoUnit`;

Ejemplo:

```
1 LocalDate fechaInicio = LocalDate.of(2016, 7, 18);
2 LocalDate fechaFin = LocalDate.of(2016, 7, 20);
3 // Calculamos el tiempo transcurrido entre las dos fechas
4 // con la clase ChronoUnit y la unidad temporal en la que
5 // queremos que nos lo devuelva, en este caso DAYS.
6 long tiempo = ChronoUnit.DAYS.between(fechaInicio, fechaFin);
7 System.out.println(tiempo); //2
```

2.4. Introducir fecha como Cadena

Podemos introducir la fecha como una cadena con el formato que deseemos y posteriormente convertir a fecha con la sentencia `parse`. Debemos importar las clases `time` y `time.format`.

Ejemplo:

```

1 | DateTimeFormatter formato = DateTimeFormatter.ofPattern("d/MM/u");
2 | String fechaCadena = "16/08/2016";
3 | LocalDate mifecha = LocalDate.parse(fechaCadena, formato);
4 | System.out.println(formato.format(mifecha)); //16/08/2016

```

Ojo! a partir de Java 8 `y` es para el año de la era (BC AD), y para el año debemos usar `u`

Más detalles sobre los formatos: <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

2.5. Manipulación

1. Manipulando `LocalDate`

Haciendo uso de los métodos `withYear(int year)`, `withMonth(int month)`, `withDayOfMonth(int dayOfMonth)`, `with(TemporalField field, long newValue)` se puede modificar el `LocalDate`.

Ejemplo:

```

1 | LocalDate date = LocalDate.of(2016, 7, 25);
2 | LocalDate date1 = date.withYear(2017);
3 | LocalDate date2 = date.withMonth(8);
4 | LocalDate date3 = date.withDayOfMonth(27);
5 | System.out.println(date); //2016-07-25
6 | System.out.println(date1); //2017-07-25
7 | System.out.println(date2); //2016-08-25
8 | System.out.println(date3); //2016-07-27

```

2. Manipulando `LocalTime`

Haciendo uso de los métodos `withHour(int hour)`, `withMinute(int minute)`, `withSecond(int second)`, `withNano(int nanoOfSecond)` se puede modificar el `LocalTime`.

Ejemplo:

```

1 | LocalTime time = LocalTime.of(14, 30, 35);
2 | LocalTime time1 = time.withHour(20);
3 | LocalTime time2 = time.withMinute(25);
4 | LocalTime time3 = time.withSecond(23);
5 | LocalTime time4 = time.withNano(24);
6 | System.out.println(time); //14:30:35
7 | System.out.println(time1); //20:30:35
8 | System.out.println(time2); //14:25:35
9 | System.out.println(time3); //14:30:23
10 | System.out.println(time4); //14:30:35.000000024

```

3. Manipulando `LocalDateTime`

`LocalDateTime` provee los mismo métodos mencionados en las clases `LocalDate` y `LocalTime`.

Ejemplo:

```

1 LocalDateTime dateTime = LocalDateTime.of(2016, 7, 25, 22, 11, 30);
2 LocalDateTime dateTime1 = dateTime.withYear(2017);
3 LocalDateTime dateTime2 = dateTime.withMonth(8);
4 LocalDateTime dateTime3 = dateTime.withDayOfMonth(27);
5 LocalDateTime dateTime4 = dateTime.withHour(20);
6 LocalDateTime dateTime5 = dateTime.withMinute(25);
7 LocalDateTime dateTime6 = dateTime.withSecond(23);
8 LocalDateTime dateTime7 = dateTime.withNano(24);
9 System.out.println(dateTime); //2016-07-25T22:11:30
10 System.out.println(dateTime1); //2017-07-25T22:11:30
11 System.out.println(dateTime2); //2016-08-25T22:11:30
12 System.out.println(dateTime3); //2016-07-27T22:11:30
13 System.out.println(dateTime4); //2016-07-25T20:11:30
14 System.out.println(dateTime5); //2016-07-25T22:25:30
15 System.out.println(dateTime6); //2016-07-25T22:11:23
16 System.out.println(dateTime7); //2016-07-25T22:11:30.000000024

```

2.6. Operaciones

2.6.1. Operaciones con `LocalDate`

Realizar operaciones como suma o resta de días, meses, años, etc es muy fácil con la nueva `Date` API. Los siguientes métodos `plus(long amountToAdd, TemporalUnit unit)`, `minus(long amountToSubtract, TemporalUnit unit)` proveen una manera general de realizar estas operaciones. (Debemos importar la clase `java.time.temporal.ChronoUnit` para poder utilizar las unidades: `ChronoUnit.YEARS`, `ChronoUnit.MONTHS`, `ChronoUnit.DAYS`).

Ejemplo:

```

1 LocalDate date = LocalDate.of(2016, 7, 18);
2 LocalDate datePlusOneDay = date.plus(1, ChronoUnit.DAYS);
3 LocalDate dateMinusOneDay = date.minus(1, ChronoUnit.DAYS);
4 System.out.println(date); // 2016-07-18
5 System.out.println(datePlusOneDay); // 2016-07-19
6 System.out.println(dateMinusOneDay); // 2016-07-17

```

También se puede hacer cálculos basados en un `Period`. En el siguiente ejemplo, se crea un `Period` de 1 día para poder realizar los cálculos.

Ejemplo:

```

1 LocalDate date = LocalDate.of(2016, 7, 18);
2 LocalDate datePlusOneDay = date.plus(Period.ofDays(1));
3 LocalDate dateMinusOneDay = date.minus(Period.ofDays(1));
4 System.out.println(date); // 2016-07-18
5 System.out.println(datePlusOneDay); // 2016-07-19
6 System.out.println(dateMinusOneDay); // 2016-07-17

```

Finalmente, haciendo uso de métodos explícitos como `plusDays(long daysToAdd)` y `minusDays(long daysToSubtract)` se puede indicar el valor a incrementar o reducir.

Ejemplo:

```
1 LocalDate date = LocalDate.of(2016, 7, 18);
2 LocalDate datePlusOneDay = date.plusDays(1);
3 LocalDate dateMinusOneDay = date.minusDays(1);
4 System.out.println(date);           // 2016-07-18
5 System.out.println(datePlusOneDay); // 2016-07-19
6 System.out.println(dateMinusOneDay); // 2016-07-17
```

2.6.2. Operaciones con `LocalTime`

La nueva `Date` API permite realizar operaciones como suma y resta de horas, minutos, segundos, etc. Al igual que `LocalDate`, los siguientes métodos `plus(long amountToAdd, TemporalUnit unit)`, `minus(long amountToSubtract, TemporalUnit unit)` proveen una manera general de realizar estas operaciones.

(Debemos importar la clase `java.time.temporal.ChronoUnit` para poder utilizar las unidades: `ChronoUnit.HOURS`, `ChronoUnit.MINUTES`, `ChronoUnit.SECONDS`, `ChronoUnit.NANOS`).

Ejemplo:

```
1 LocalTime time = LocalTime.of(15, 30);
2 LocalTime timePlusOneHour = time.plus(1, ChronoUnit.HOURS);
3 LocalTime timeMinusOneHour = time.minus(1, ChronoUnit.HOURS);
4 System.out.println(time);           // 15:30
5 System.out.println(timePlusOneHour); // 16:30
6 System.out.println(timeMinusOneHour); // 14:30
```

También se puede hacer cálculos basados en un `Duration`. En el siguiente ejemplo, se crea un `Duration` de 1 hora para poder realizar los cálculos.

```
1 LocalTime time = LocalTime.of(15, 30);
2 LocalTime timePlusOneHour = time.plus(Duration.ofHours(1));
3 LocalTime timeMinusOneHour = time.minus(Duration.ofHours(1));
4 System.out.println(time);           // 15:30
5 System.out.println(timePlusOneHour); // 16:30
6 System.out.println(timeMinusOneHour); // 14:30
```

Finalmente, haciendo uso de métodos explícitos como `plusHours(long hoursToAdd)` y `minusHours(long hoursToSubtract)` se puede indicar el valor a incrementar o reducir.

Ejemplo:

```

1 LocalTime time = LocalTime.of(15, 30);
2 LocalTime timePlusOneHour = time.plusHours(1);
3 LocalTime timeMinusOneHour = time.minusHours(1);
4 System.out.println(time);           // 15:30
5 System.out.println(timePlusOneHour); // 16:30
6 System.out.println(timeMinusOneHour); // 14:30

```

2.6.3. Operaciones con `LocalDateTime`

`LocalDateTime`, al ser una clase compuesta por `LocalDate` y `LocalTime` ofrece los mismos métodos para realizar operaciones.

(Debemos importar la clase `java.time.temporal.ChronoUnit` para poder utilizar las unidades: `ChronoUnit.YEARS`, `ChronoUnit.MONTHS`, `ChronoUnit.DAYS`, `ChronoUnit.HOURS`, `ChronoUnit.MINUTES`, `ChronoUnit.SECONDS`, `ChronoUnit.NANOS`).

Ejemplo:

```

1 LocalDateTime dateTime = LocalDateTime.of(2016, 7, 28, 14, 30);
2 LocalDateTime dateTime1 = dateTime.plus(1, ChronoUnit.DAYS).plus(1,
   ChronoUnit.HOURS); LocalDateTime dateTime2 = dateTime.minus(1,
   ChronoUnit.DAYS).minus(1, ChronoUnit.HOURS);
3 System.out.println(dateTime); // 2016-07-28T14:30
4 System.out.println(dateTime1); // 2016-07-29T15:30
5 System.out.println(dateTime2); // 2016-07-27T13:30

```

En el siguiente ejemplo, se hace uso de `Period` y `Duration`:

```

1 LocalDateTime dateTime = LocalDateTime.of(2016, 7, 28, 14, 30);
2 LocalDateTime dateTime1 =
   dateTime.plus(Period.ofDays(1)).plus(Duration.ofHours(1));
3 LocalDateTime dateTime2 =
   dateTime.minus(Period.ofDays(1)).minus(Duration.ofHours(1));
4 System.out.println(dateTime); // 2016-07-28T14:30
5 System.out.println(dateTime1); // 2016-07-29T15:30
6 System.out.println(dateTime2); // 2016-07-27T13:30

```

Finalmente, haciendo uso de los métodos `plusX(long xToAdd)` o `minusX(long xToSubtract)`:

```

1 LocalDateTime dateTime = LocalDateTime.of(2016, 7, 28, 14, 30);
2 LocalDateTime dateTime1 = dateTime.plusDays(1).plusHours(1);
3 LocalDateTime dateTime2 = dateTime.minusDays(1).minusHours(1);
4 System.out.println(dateTime); // 2016-07-28T14:30
5 System.out.println(dateTime1); // 2016-07-29T15:30
6 System.out.println(dateTime2); // 2016-07-27T13:30

```

Además, métodos como `isBefore`, `isAfter`, `isEqual` están disponibles para comparar las siguientes clases `LocalDate`, `LocalTime` y `LocalDateTime`.

Ejemplo:

```
1 LocalDate date1 = LocalDate.of(2016, 7, 28);
2 LocalDate date2 = LocalDate.of(2016, 7, 29);
3 boolean isBefore = date1.isBefore(date2); //true
4 boolean isAfter = date2.isAfter(date1); //true
5 boolean isEqual = date1.isEqual(date2); //false
```

2.7. Formatos

Cuando se trabaja con fechas, en ocasiones se requiere de un formato personalizado. Podemos usar el método `ofPattern(String pattern)`, para definir un formato en particular.

Para utilizar `DateTimeFormatter.ofPattern` debemos importar la clase con `import java.time.format.DateTimeFormatter;`

Ejemplo:

```
1 LocalDate mifecha = LocalDate.of(2016, 7, 25);
2 String fechaTexto=mifecha.format(DateTimeFormatter.ofPattern("eeee', ' dd 'de'
   MMMM 'del' uuuu"));
3 System.out.println("La fecha es: "+fechaTexto); // La fecha es: lunes, 25 de
   julio del 2016
```

El patrón del formato se realiza en función a la siguiente tabla de símbolos:

Símbolo	Descripción	Salida
y	Año	2004; 04
D	Día del Año	189
M	Mes del Año	7; 07; Jul; July; J
d	Día del Mes	10
w	Semana del Año	27
E	Día de la Semana	Tue; Tuesday; T
F	Semana del Mes	3
a	AM/PM	PM
K	Hora AM/PM (0-11)	0
H	Hora del día (0-23)	0
m	Minutos de la hora	30
s	Segundos del minuto	55
n	Nanosegundos del Segundo	987654321
"	Texto	'Día de la semana'

2.7.1. Día de la Semana

La función `getDayOfWeek()` devuelve un elemento del tipo `DayOfWeek` que corresponde el día de la semana de una fecha. Debemos importar la clase `java.time.DayOfWeek`.

Por ejemplo, el lunes será `DayOfWeek.MONDAY`.

Ejemplo:

```

1  LocalDate lafecha = LocalDate.of(2016, 7, 25);
2  if (lafecha.getDayOfWeek().equals(DayOfWeek.SATURDAY)) {
3      System.out.println("La fecha es Sábado");
4  } else {
5      System.out.println("La fecha NO es Sábado");
6  }
7  //La fecha NO es Sábado

```


3. Ejemplo Anexo UD05

3.1. Anexo1Wrappers

```

1 package UD05;
2
3 public class Anexo1Wrappers {
4
5     public static void main(String[] args) {
6
7         // WRAPPERS
8         Integer i1 = new Integer(42);
9         Integer i2 = new Integer("42");
10        Float f1 = new Float(3.14f);
11        Float f2 = new Float("3.14f");
12
13        Integer y = new Integer(567);          //Crea el objeto
14        y++;                                   //Lo desenvuelve, incrementa y lo vuelve a
envolver
15        System.out.println("Valor: " + y); //Imprime el valor del Objeto y
16
17        // VALUEOF
18        // Convierte el 101011 (base 2) a 43 y le asigna el valor al objeto
Integer i1
19        Integer i3 = Integer.valueOf("101011", 2);
20        System.out.println(i3);
21
22        // Asigna 3.14 al objeto Float f3
23        Float f3 = Float.valueOf("3.14f");
24        System.out.println(f3);
25
26        // XXXVALUE
27        Integer i4 = 120; // Crea un nuevo objeto wrapper
28        byte b = i4.byteValue(); // Convierte el valor de i4 a un primitivo byte
29        short s1 = i4.shortValue(); // Otro de los métodos de Integer
30        double d = i4.doubleValue(); // Otro de los métodos xxxValue de Integer
31        System.out.println(s1); // Muestra 120 como resultado
32
33        Float f4 = 3.14f; // Crea un nuevo objeto wrapper
34        short s2 = f4.shortValue(); // Convierte el valor de f4 en un primitivo
short
35        System.out.println(s2); // El resultado es 3 (truncado, no redondeado)
36
37        // PARSEXXX
38        double d4 = Double.parseDouble("3.14"); // Convierte un String a
primitivo
39        System.out.println("d4 = " + d4); // El resultado será d4 = 3.14
40        long l2 = Long.parseLong("101010", 2); // un String binario a primitivo
41        System.out.println("l2 = " + l2); // El resultado es l2 = 42
42

```

```

43 // ToString
44 Double d1 = new Double("3.14");
45 System.out.println("d1 = " + d1.toString()); // El resultado es d1 =
3.14
46 String d2 = Double.toString(3.14); // d2 = "3.14"
47 System.out.println("d2 = " + d2); // El resultado es d2 = 3.14
48 String s3 = "hex = " + Long.toString(254, 16); // s3 = "hex = fe"
49 System.out.println("s3 = " + s3); // El resultado es s3 = 3.14
50
51 // ToXXXString
52 String s4 = Integer.toHexString(254); // Convierte 254 a hex
53 System.out.println("254 es " + s4); // Resultado: "254 es fe"
54 String s5 = Long.toOctalString(254); // Convierte 254 a octal
55 System.out.println("254(oct) = " + s5); // Resultado: "254(oct) = 376"
56 }
57 }

```

3.2. Anexo2Date

```

1 package UD05;
2
3 import java.util.Calendar;
4 import java.util.Date;
5 import java.util.GregorianCalendar;
6 import java.time.*;
7 import java.time.format.DateTimeFormatter;
8 import java.time.temporal.ChronoUnit;
9
10 public class Anexo2Date {
11
12     public static void main(String[] args) {
13
14         //Clase Date (java.util.Date)
15         Date fecha = new Date(2021, 8, 19);
16         System.out.println(fecha); //Mon Sep 19 00:00:00 CEST 3921
17         System.out.println(fecha.getTime()); //61590146400000
18
19         //Clase GregorianCalendar (java.util.Calendar y
java.util.GregorianCalendar)
20         Calendar calendar = new GregorianCalendar(2021, 8, 19);
21         System.out.println(calendar.getTime()); //Sun Sep 19 00:00:00 CEST 2021
22
23         Date d = new Date();
24         GregorianCalendar c = new GregorianCalendar();
25         System.out.println("Fecha: " + d); //Fecha: Thu Aug 19 20:06:14 CEST
2021
26         System.out.println("Info: " + c); //Info:
java.util.GregorianCalendar[time=1629396374723,
27
//areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util.calendar.Z
oneInfo

```

```

28      //[id="Europe/Madrid",offset=3600000,dstSavings=3600000,useDaylight=true,trans
29      itions=163,
30
31      //lastRule=java.util.SimpleTimeZone[id=Europe/Madrid,offset=3600000,dstSavings
32      =3600000,
33
34      //useDaylight=true,startYear=0,startMode=2,startMonth=2,startDay=-1,startDayOf
35      Week=1,
36
37      //startTime=3600000,startTimeMode=2,endMode=2,endMonth=9,endDay=-1,endDayOfWee
38      k=1,
39
40      //endTime=3600000,endTimeMode=2]],firstDayOfWeek=2,minimalDaysInFirstWeek=4,ER
41      A=1,
42
43      //YEAR=2021,MONTH=7,WEEK_OF_YEAR=33,WEEK_OF_MONTH=3,DAY_OF_MONTH=19,DAY_OF_YEA
44      R=231,
45
46      //DAY_OF_WEEK=5,DAY_OF_WEEK_IN_MONTH=3,AM_PM=1,HOUR=8,HOUR_OF_DAY=20,MINUTE=6,
47      SECOND=14,
48      //MILLISECOND=723,ZONE_OFFSET=3600000,DST_OFFSET=3600000]
49      c.setTime(d);
50      System.out.print(c.get(Calendar.DAY_OF_MONTH));
51      System.out.print("/");
52      System.out.print(c.get(Calendar.MONTH) + 1);
53      System.out.print("/");
54      System.out.println(c.get(Calendar.YEAR) + 1); //19/8/2022
55
56      //LocalDate, LocalTime, LocalDateTime, Duration y Period (java.time.*)
57      //LocalDate d = new LocalDate(); //NO compila
58      LocalDate date = LocalDate.of(1989, 11, 11); //1989-11-11
59      System.out.println(date.getYear()); //1989
60      System.out.println(date.getMonth()); //NOVEMBER
61      System.out.println(date.getDayOfMonth()); //11
62      date = LocalDate.now();
63      System.out.println(date); //2021-08-19
64
65      LocalTime time = LocalTime.of(5, 30, 45, 35); //05:30:45:35
66      System.out.println(time.getHour()); //5
67      System.out.println(time.getMinute()); //30
68      System.out.println(time.getSecond()); //45
69      System.out.println(time.getNano()); //35
70      time = LocalTime.now();
71      System.out.println(time); //20:13:53.118044
72
73      LocalDateTime dateTime = LocalDateTime.of(1989, 11, 11, 5, 30, 45, 35);
74
75      LocalDate date2 = LocalDate.of(1989, 11, 11);
76      LocalTime time2 = LocalTime.of(5, 30, 45, 35);
77      LocalDateTime dateTime1 = LocalDateTime.of(date, time);
78      LocalDateTime dateTime2 = LocalDateTime.now();

```

```

66
67     LocalDateTime localTime1 = LocalDateTime.of(12, 25);
68     LocalDateTime localTime2 = LocalDateTime.of(17, 35);
69     Duration duration1 = Duration.between(localTime1, localTime2);
70     System.out.println(duration1); //PT5H10M
71     System.out.println(duration1.toDays()); //0
72
73     LocalDateTime localDateTime1 = LocalDateTime.of(2016, Month.JULY, 18,
14, 13);
74     LocalDateTime localDateTime2 = LocalDateTime.of(2016, Month.JULY, 20,
12, 25);
75     Duration duration2 = Duration.between(localDateTime1, localDateTime2);
76     System.out.println(duration2); //PT46H12M
77     System.out.println(duration2.toDays()); //1
78
79     Duration duracion3 = Duration.ofDays(1);
80     System.out.println(duracion3); //PT24H
81     System.out.println(duracion3.toDays()); //1
82
83     LocalDate localDate1 = LocalDate.of(2016, 7, 18);
84     LocalDate localDate2 = LocalDate.of(2016, 7, 20);
85     Period periodo1 = Period.between(localDate1, localDate2);
86     System.out.println(periodo1); //P2D
87
88     Period periodo2 = Period.of(1, 2, 3);
89     System.out.println(periodo2); //P1Y2M3D
90
91     Period periodo3 = Period.ofYears(1);
92     System.out.println(periodo3); //P1Y
93
94     //CHRONOUNIT (java.time.temporal.ChronoUnit)
95     LocalDate fechaInicio = LocalDate.of(2016, 7, 18);
96     LocalDate fechaFin = LocalDate.of(2016, 7, 20);
97     // Calculamos el tiempo transcurrido entre las dos fechas
98     // con la clase ChronoUnit y la unidad temporal en la que
99     // queremos que nos lo devuelva, en este caso DAYS.
100    long tiempo = ChronoUnit.DAYS.between(fechaInicio, fechaFin);
101    System.out.println(tiempo); //2
102
103    //Introducir fecha por teclado (java.time.format.DateTimeFormatter)
104    DateTimeFormatter formato = DateTimeFormatter.ofPattern("d/MM/yyyy");
105    String fechaCadena = "16/08/2016";
106    LocalDate mifecha = LocalDate.parse(fechaCadena, formato);
107    System.out.println(formato.format(mifecha)); //16/08/2016
108
109    //Manipulación
110    LocalDate fec = LocalDate.of(2016, 7, 25);
111    LocalDate fec1 = fec.withYear(2017);
112    LocalDate fec2 = fec.withMonth(8);
113    LocalDate fec3 = fec.withDayOfMonth(27);
114    System.out.println(date); //2016-07-25
115    System.out.println(fec1); //2017-07-25

```

```

116     System.out.println(fec2); //2016-08-25
117     System.out.println(fec3); //2016-07-27
118
119     LocalDateTime tim = LocalDateTime.of(14, 30, 35);
120     LocalDateTime tim1 = tim.withHour(20);
121     LocalDateTime tim2 = tim.withMinute(25);
122     LocalDateTime tim3 = tim.withSecond(23);
123     LocalDateTime tim4 = tim.withNano(24);
124     System.out.println(tim); //14:30:35
125     System.out.println(tim1); //20:30:35
126     System.out.println(tim2); //14:25:35
127     System.out.println(tim3); //14:30:23
128     System.out.println(tim4); //14:30:35.000000024
129
130     LocalDateTime dateTim = LocalDateTime.of(2016, 7, 25, 22, 11, 30);
131     LocalDateTime dateTim1 = dateTim.withYear(2017);
132     LocalDateTime dateTim2 = dateTim.withMonth(8);
133     LocalDateTime dateTim3 = dateTim.withDayOfMonth(27);
134     LocalDateTime dateTim4 = dateTim.withHour(20);
135     LocalDateTime dateTim5 = dateTim.withMinute(25);
136     LocalDateTime dateTim6 = dateTim.withSecond(23);
137     LocalDateTime dateTim7 = dateTim.withNano(24);
138     System.out.println(dateTim); //2016-07-25T22:11:30
139     System.out.println(dateTim1); //2017-07-25T22:11:30
140     System.out.println(dateTim2); //2016-08-25T22:11:30
141     System.out.println(dateTim3); //2016-07-27T22:11:30
142     System.out.println(dateTim4); //2016-07-25T20:11:30
143     System.out.println(dateTim5); //2016-07-25T22:25:30
144     System.out.println(dateTim6); //2016-07-25T22:11:23
145     System.out.println(dateTim7); //2016-07-25T22:11:30.000000024
146
147     //OPERACIONES
148     LocalDate date3 = LocalDate.of(2016, 7, 18);
149     LocalDate date3PlusOneDay = date3.plus(1, ChronoUnit.DAYS);
150     LocalDate date3MinusOneDay = date3.minus(1, ChronoUnit.DAYS);
151     System.out.println(date3); // 2016-07-18
152     System.out.println(date3PlusOneDay); // 2016-07-19
153     System.out.println(date3MinusOneDay); // 2016-07-17
154
155     LocalDate date4 = LocalDate.of(2016, 7, 18);
156     LocalDate date4PlusOneDay = date4.plus(Period.ofDays(1));
157     LocalDate date4MinusOneDay = date4.minus(Period.ofDays(1));
158     System.out.println(date4); // 2016-07-18
159     System.out.println(date4PlusOneDay); // 2016-07-19
160     System.out.println(date4MinusOneDay); // 2016-07-17
161
162     LocalDate date5 = LocalDate.of(2016, 7, 18);
163     LocalDate date5PlusOneDay = date5.plusDays(1);
164     LocalDate date5MinusOneDay = date5.minusDays(1);
165     System.out.println(date5); // 2016-07-18
166     System.out.println(date5PlusOneDay); // 2016-07-19
167     System.out.println(date5MinusOneDay); // 2016-07-17

```

```

168
169     LocalDateTime time3 = LocalDateTime.of(15, 30);
170     LocalDateTime time3PlusOneHour = time3.plus(1, ChronoUnit.HOURS);
171     LocalDateTime time3MinusOneHour = time3.minus(1, ChronoUnit.HOURS);
172     System.out.println(time3);           // 15:30
173     System.out.println(time3PlusOneHour); // 16:30
174     System.out.println(time3MinusOneHour); // 14:30
175
176     LocalDateTime time4 = LocalDateTime.of(15, 30);
177     LocalDateTime time4PlusOneHour = time4.plus(Duration.ofHours(1));
178     LocalDateTime time4MinusOneHour = time4.minus(Duration.ofHours(1));
179     System.out.println(time4);           // 15:30
180     System.out.println(time4PlusOneHour); // 16:30
181     System.out.println(time4MinusOneHour); // 14:30
182
183     LocalDateTime time5 = LocalDateTime.of(15, 30);
184     LocalDateTime time5PlusOneHour = time5.plusHours(1);
185     LocalDateTime time5MinusOneHour = time5.minusHours(1);
186     System.out.println(time5);           // 15:30
187     System.out.println(time5PlusOneHour); // 16:30
188     System.out.println(time5MinusOneHour); // 14:30
189
190     LocalDateTime dateTime3 = LocalDateTime.of(2016, 7, 28, 14, 30);
191     LocalDateTime dateTime4 = dateTime3.plus(1, ChronoUnit.DAYS).plus(1,
ChronoUnit.HOURS);
192     LocalDateTime dateTime5 = dateTime3.minus(1, ChronoUnit.DAYS).minus(1,
ChronoUnit.HOURS);
193     System.out.println(dateTime3); // 2016-07-28T14:30
194     System.out.println(dateTime4); // 2016-07-29T15:30
195     System.out.println(dateTime5); // 2016-07-27T13:30
196
197     LocalDateTime dateTime6 = LocalDateTime.of(2016, 7, 28, 14, 30);
198     LocalDateTime dateTime7 =
dateTime6.plus(Period.ofDays(1)).plus(Duration.ofHours(1));
199     LocalDateTime dateTime8 =
dateTime6.minus(Period.ofDays(1)).minus(Duration.ofHours(1));
200     System.out.println(dateTime6); // 2016-07-28T14:30
201     System.out.println(dateTime7); // 2016-07-29T15:30
202     System.out.println(dateTime8); // 2016-07-27T13:30
203
204     LocalDateTime dateTime9 = LocalDateTime.of(2016, 7, 28, 14, 30);
205     LocalDateTime dateTime10 = dateTime9.plusDays(1).plusHours(1);
206     LocalDateTime dateTime11 = dateTime9.minusDays(1).minusHours(1);
207     System.out.println(dateTime9); // 2016-07-28T14:30
208     System.out.println(dateTime10); // 2016-07-29T15:30
209     System.out.println(dateTime11); // 2016-07-27T13:30
210
211     LocalDate dat1 = LocalDate.of(2016, 7, 28);
212     LocalDate dat2 = LocalDate.of(2016, 7, 29);
213     boolean isBefore = dat1.isBefore(dat2); //true
214     boolean isAfter = date2.isAfter(dat1); //true
215     boolean isEqual = dat1.isEqual(dat2); //false

```

```
216
217     //Formatos (java.time.format.DateTimeFormatter)
218     LocalDate mifecha2 = LocalDate.of(2016, 7, 25);
219     String fechaTexto = mifecha2.format(DateTimeFormatter.
220         ofPattern("eeee", ' dd 'de' MMMM
'del' yyyy"));
221     System.out.println("La fecha es: " +
222         fechaTexto); // La fecha es: lunes, 25 de julio del
2016
223
224     //DAYOFWEEK
225     LocalDate lafecha = LocalDate.of(2016, 7, 25);
226     if (lafecha.getDayOfWeek().equals(DayOfWeek.SATURDAY)) {
227         System.out.println("La fecha es Sábado");
228     } else {
229         System.out.println("La fecha NO es Sábado");
230     }
231     //La fecha NO es Sábado
232 }
233 }
```

4. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Rama\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)