Uno de los problemas más comunes a los que se enfrentan los desarrolladores es el manejo de fechas, tiempo y zonas horarias, para resolver este problema se solían utilizar api's como Joda Time, Java 8 incluyó un conjunto de apis que nos ayudan a resolverlo sin incluir dependencias adicionales.

LocalDate Representa una fecha (día, mes, año) sin hora.

LocalDateTime Representa una fecha y una hora sin tener en cuenta el huso horario.

LocalTime Representa una hora sin tener en cuenta el huso horario.

OffsetDateTime Representa una fecha y una hora en formato UTC.

OffsetTime Representa una hora en formato UTC.

ZonedDateTime Representa una fecha y una hora con el uso horario correspondiente.

Duration Representa una duración expresada en horas, minutos y segundos.

Period Representa una duración expresada en días, meses y años.

MonthDay Representa un día y un mes, sin año.

YearMonth Representa un mes y un año, sin día.

of: devuelve una instancia de la clase inicializada con los distintos valores que se pasan como parámetro.

```
LocalDate navidad;
navidad=LocalDate.of(2014, 12,25);
```

from: conversión entre los distintos tipos. En caso de conversión hacia un tipo menos complejo, se pierde información.

```
LocalDateTime ahora;
ahora=LocalDateTime.now();

// transformación en LocalDate con pérdida de la hora

LocalDate hoy;
hoy=LocalDate.from(ahora);
```

parse: transforma la cadena de caracteres que se pasa como parámetro al tipo correspondiente.

```
LocalTime reloj;
reloj=LocalTime.parse("22:45:03");
```

withxxxxxx: devuelve una nueva instancia modificando la componente indicada por xxxxxx por el valor que se pasa como parámetro.

```
LocalTime reloj;
reloj=LocalTime.parse("22:45:03");
LocalTime nuevaHora;
nuevaHora=reloj.withHour(9);
```

plusxxxxx y minusxxxx: devuelve una nueva instancia de la clase tras agregar o sustraer el número de unidades indicado por el parámetro.

xxxxxx indica qué se agrega o se sustrae.

```
LocalDate pascua;
pascua=LocalDate.of(2014,4,20);
LocalDate ascension;
ascension=pascua.plusDays(39);
```

MonthDay Representa un día y un mes, sin año.

YearMonth Representa un mes y un año, sin día.

atxxxxxx: combina el objeto recibido como parámetro con el objeto en curso y devuelve el resultado de dicha asociación. Es posible, por ejemplo, combinar un objeto LocalDate y un objeto LocalTime para obtener un objeto LocalDateTime.

```
LocalDate diaPartido;
diaPartido=LocalDate.of(2014,7,13);
LocalTime horaPartido;
horaPartido=LocalTime.of(21,00);
LocalDateTime fin;
fin=diaPartido.atTime(horaPartido);
```

El pequeño ejemplo de código que se muestra a continuación muestra algunas operaciones sobre las fechas teniendo en cuenta que los días festivos son los sábados y domingos.

```
 MonthDay[] fiestas;
fiestas=new MonthDay[8];
fiestas[0]=MonthDay.of(1,1);
fiestas[1]=MonthDay.of(5,1);
fiestas[2]=MonthDay.of(8,15);
fiestas[3]=MonthDay.of(10,12);
fiestas[4]=MonthDay.of(11,1);
fiestas[5]=MonthDay.of(12,6);
fiestas[6]=MonthDay.of(12,8);
fiestas[7]=MonthDay.of(12,25);

int numDias;
int año;
LocalDate diaTest;
for (año=2014;año<2030;año++) {
            numDias=0;
            for(MonthDay test:fiestas) {
                diaTest=test.atYear(año);
                            if(diaTest.getDayOfWeek()==DayOfWeek.SATURDAY
                           ||diaTest.getDayOfWeek()==DayOfWeek.SUNDAY)
                            {
                                numDias++;
                            }
            }
System.out.println("En " + año + " hay " + numDias + " dia(s) festivo(s) sábado o domingo");
}
```

LocalDate, LocalTime y LocalDateTime

Las clases más comunes para el manejo de fechas con java 8 son LocalDate, LocalTime y LocalDateTime, se utilizan cuando la zona horaria no es requerida.

LocalDate

Un LocalDate representa una fecha en formato ISO (yyyy-MM-dd) **sin tiempo.**

```
LocalDate date = LocalDate.now();
System.out.println(date);
```

El código anterior tendrá la siguiente salida(Entendiendo que el post se escribió el 30-10-2018):

```
2018-10-30
```

Como vemos es una fecha **sin tiempo y sin zona horaria.** Veamos algunas otras formas de crear un LocalDate:

```
LocalDate date2 = LocalDate.of(2018, 10, 30);
LocalDate date3 = LocalDate.parse("2018-10-30");
```

Las 2 expresiones crearán objetos de tipo LocalDate con la fecha del 30-10-2018.

Operaciones que se pueden realizar con LocalDate

- Manipulación de fechas (Sumar o restar días, meses, años, etc):

```
LocalDate date = LocalDate.parse("2018-10-30");
LocalDate newDate = date.plusDays(10);
System.out.println(date);
System.out.println(newDate);
```

Salida:

```
2018-10-30
2018-11-09
```

Como vemos a la fecha inicial se le sumaron 10 días y el mes se actualizó de forma automática, esto nos permite evitar considerar el número de días en un mes, el horario de verano, etc.

```
LocalDate date = LocalDate.parse("2018-10-30");
LocalDate newDate = date.plusMonths(3);
System.out.println(date);
System.out.println(newDate);
```

Salida:

```
2018-10-30
2019-01-30
```

De igual forma podemos hacerlo con los meses y LocalDate resolverá si es necesario cambiar de año.

Recordemos que cuando hacemos operaciones sobre las fechas debemos asignar la respuesta a una nueva referencia ya que el objeto original no se modificará puesto que los objetos LocalDate son immutables.

- Comparación entre fechas

  Así como podemos realizar operaciones entre las fechas podemos hacer comparaciones entre ellas, veamos algunos ejemplos:

  Valida si una fecha es antes que otra:

```
System.out.println(LocalDate.parse("2018-10-
30").isBefore(LocalDate.parse("2018-10-31")));
```

  Salida

```
true
```

  Valida si un año es bisiesto:

```
System.out.println(LocalDate.parse("2018-10-30").isLeapYear());
```

  Salida:

```
false
```

- Obtener información sobre alguna fecha

  El siguiente paso será obtener información sobre alguna fecha en específico veamos algunos ejemplos:

  Obtener el día de la semana de mi cumpleaños:

```
System.out.println(LocalDate.parse("2019-08-19").getDayOfWeek());
```

  Salida:

```
MONDAY
```

  Extraer información de una fecha:

```
System.out.println(LocalDate.parse("2019-08-19").getDayOfWeek());
```

  Salida:

  1

  Podemos ver el detalle de todos los métodos disponibles de la clase LocalDate en el siguiente *link*.

  LocalTime

  LocalTime representa una hora sin la fecha, del mismo modo que con LocalDate podemos crearlo haciendo uso de los métodos now(), parse(..) y of(..), veamos algunos ejemplos:

```
LocalTime time = LocalTime.now();
LocalTime time2 = LocalTime.parse("11:00:59.759");
LocalTime time3 = LocalTime.of(11, 00, 59);
System.out.println(time);
System.out.println(time2);
System.out.println(time3);
```

  Salida:

```
11:02:06.198
```

```
11:00:59.759
11:00:59
```

Las 3 anteriores son formas válidas de crear un objeto LocalTime. Veamos algunas de las operaciones que podemos realizar.

- Modificar un LocalTime

  La primera operación que veremos es como modificar un LocalTime:

```
LocalTime time = LocalTime.parse("11:00:59.759");
LocalTime time2 = time.plusHours(1);
System.out.println(time2);
```

  Salida:

```
12:00:59.759
```

  Con el método plusHours crearemos un nuevo LocalTime con la nueva hora calculada.

- Validar un LocalTime

  El siguiente punto será hacer validaciones sobre un LocalTime:

```
LocalTime time = LocalTime.parse("11:00:59.759");
LocalTime time2 = LocalTime.parse("12:00:59.759");
System.out.println(time.isBefore(time2));
```

  Salida:

```
true
```

  Usando métodos como isBefore podremos saber si una hora es mayor a otra.

- Extraer información de una hora:

  El siguiente paso será extraer solo una parte del objeto LocalTime:

```
LocalTime time = LocalTime.parse("11:00:59.759");
System.out.println(time.getHour());
```

  Salida:

  1

  Esto lo utilizaremos en caso de que nuestra aplicación utilice solo la hora para realizar algún proceso.

  LocalDateTime

  Representa una combinación entre LocalDate y LocalTime:

```
LocalDateTime dateTime = LocalDateTime.now();
LocalDateTime dateTime1=LocalDateTime.of(2018, 10, 10, 11, 25);
LocalDateTime dateTime2=LocalDateTime.parse("2018-10-10T11:25");

System.out.println(dateTime);
System.out.println(dateTime1);
System.out.println(dateTime2);
```

Salida:

```
2018-10-30T11:46:58.274
2018-10-10T11:25
2018-10-10T11:25
```

En la salida anterior podemos ver como la salida incluye fecha y hora por cada objeto. Veamos algunas de las operaciones que podemos realizar con estos objetos.

- Manipular un LocalDateTime:

  De igual forma que con los anteriores podemos realzar manipulaciones sobre el LocalDateTime:

  ```
  LocalDateTime dateTime=LocalDateTime.parse("2018-10-10T11:25");
  LocalDateTime newDateTime = dateTime.plusDays(1).plusHours(2);
  System.out.println(newDateTime);
  ```

  Lo anterior creará un objeto LocalDateTime le agregará 1 día y después 2 horas, recordemos que debemos asignar el resultado a una nueva referencia ya que el objeto original no se modificará sino que se devolverá uno nuevo.

- Realizar validaciones sobre un LocalDateTime

  Como lo vimos en los ejemplos anteriores podemos realizar validaciones sobre el LocalDateTime:

  ```
  LocalDateTime dateTime=LocalDateTime.parse("2018-10-10T11:25");
  LocalDateTime dateTime2=LocalDateTime.parse("2019-10-10T11:25");
  System.out.println(dateTime.isBefore(dateTime2));
  ```

  Salida:

  ```
  true
  ```

  Se realizará del mismo modo que en los casos anteriores solo que ahora considerará tanto la fecha como la hora.

ZonedDateTime

Si necesitas trabajar con zonas horarias puedes utilizar esta clase que te provee el manejo de fechas con hora para la zona que determines. La lista de zonas disponibles las puedes consultar desde la clase ZoneId

```
ZoneId.getAvailableZoneIds().forEach(z -> System.out.println(z)); // list of all zones
ZoneId zoneId = ZoneId.of("America/Panama");
```

Para 'moverse' a otra zona horaria, por ejemplo Tokyo, haces uso de de LocalDateTime en conjunto con ZoneDateTime pasando la zona "Asia/Tokyo"

```
LocalDateTime lo = LocalDateTime.of(2017, Month.AUGUST, 20, 8, 30);
ZonedDateTime zonedDateTime = ZonedDateTime.of(lo, zoneId);
System.out.println(zonedDateTime); // 2017-08-20T08:30-05:00[America/Panama]
ZonedDateTime tokyoDateTime = lo.atZone(ZoneId.of("Asia/Tokyo"));
System.out.println(tokyoDateTime); // 2017-08-20T08:30+09:00[Asia/Tokyo]
```

Period

Con la clase Period puedes obtener la diferencia entre dos fechas o utilizarlo para modificar valores de alguna fecha.

```
LocalDate startLocalDate = LocalDate.of(2017, 10, 10);
LocalDate endLocalDate = startLocalDate.plus(Period.ofDays(10));
// 2017-10-20
int diffDays = Period.between(startLocalDate, endLocalDate).getDays();
System.out.println(diffDays); // 10
```

Duration

Duration es el equivalente a Period pero para las horas.

```
LocalTime inicio = LocalTime.of(8, 30);
LocalTime fin = inicio.plus(Duration.ofHours(3));
// 11:30
long diffSeconds = Duration.between(inicio,fin).getSeconds();
System.out.println(diffSeconds); // 10800 seconds
```

OffsetDateTime Representa una fecha y una hora en formato UTC.

```
//public static OffsetDateTime of(LocalDate date, LocalTime time, ZoneOffset offset)

LocalDate localDate = LocalDate.parse("2017-02-03");
LocalTime localTime = LocalTime.parse("12:30:30");
OffsetDateTime date = OffsetDateTime.of(localDate, localTime,ZoneOffset.UTC);
System.out.println(date);

OffsetDateTime offsetDT1 = OffsetDateTime.now();
System.out.println("OffsetDateTime1: " + offsetDT1);

OffsetDateTime offsetDT2 = OffsetDateTime.now(Clock.systemUTC());
System.out.println("OffsetDateTime2: " + offsetDT2);

OffsetDateTime offsetDT3 = OffsetDateTime.now(ZoneId.of("Asia/Jakarta"));
System.out.println("OffsetDateTime3: " + offsetDT3);

OffsetDateTime offsetDT4 = OffsetDateTime.of(1980, 4, 9, 20, 15, 45, 345875000,
ZoneOffset.of("+07:00"));
System.out.println("OffsetDateTime4: " + offsetDT4);

OffsetDateTime offsetDT5 = OffsetDateTime.of(LocalDate.now(), LocalTime.of(15, 50, 25),
ZoneOffset.of("+07:00"));
System.out.println("OffsetDateTime5: " + offsetDT5);

OffsetDateTime offsetDT6 = OffsetDateTime.of(LocalDateTime.now(),
ZoneOffset.of("+07:00"));
System.out.println("OffsetDateTime6: " + offsetDT6);

OffsetDateTime offsetDT7 = OffsetDateTime.ofInstant(Instant.now(),
ZoneId.systemDefault());
System.out.println("OffsetDateTime7: " + offsetDT7);

OffsetDateTime offsetDT8 = OffsetDateTime.parse("2019-08-31T15:20:30+08:00");
System.out.println("OffsetDateTime8: " + offsetDT8);

OffsetDateTime offsetDT9 = OffsetDateTime.parse("1980-04-09T08:20:45+07:00",
DateTimeFormatter.ISO_OFFSET_DATE_TIME);
System.out.println("OffsetDateTime9: " + offsetDT9);
```

OffsetTime Representa una hora en formato UTC.

```
OffsetTime offsetTime1 = OffsetTime.now();
 System.out.println("OffsetTime1: " + offsetTime1);
 OffsetTime offsetTime2 = OffsetTime.now(Clock.systemUTC());
 System.out.println("OffsetTime2: " + offsetTime2);

 OffsetTime offsetTime3 = OffsetTime.now(ZoneId.of("Asia/Jakarta"));
 System.out.println("OffsetTime3: " + offsetTime3);

 OffsetTime offsetTime4 = OffsetTime.of(20, 15, 45, 345875000, ZoneOffset.of("+07:00"));
 System.out.println("OffsetTime4: " + offsetTime4);

 OffsetTime offsetTime5 = OffsetTime.of(LocalTime.of(15, 50, 25),
ZoneOffset.of("+07:00"));
 System.out.println("OffsetTime5: " + offsetTime5);

 OffsetTime offsetTime6 = OffsetTime.ofInstant(Instant.now(), ZoneId.systemDefault());
 System.out.println("OffsetTime6: " + offsetTime6);

 OffsetTime offsetTime7 = OffsetTime.parse("10:15:30+07:00");
 System.out.println("OffsetTime7: " + offsetTime7);

 OffsetTime offsetTime8 = OffsetTime.parse("18:30:15+08:00",
DateTimeFormatter.ISO_OFFSET_TIME);
 System.out.println("OffsetTime8: " + offsetTime8);
```