1️⃣

Manejo de fechas

▼ 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.

```
class Prueba {
  public static void main(String arg[]) {
    LocalDate date=LocalDate.now();
    System.out.println(date);
  }
}
```

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

```
class Prueba {
  public static void main(String arg[]) {
    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):

```
public static void main(String args[]) {
  LocalDate date=LocalDate.parse("2018-10-30");
  LocalDate newDate=date.plusDays(10);
```

```
  System.out.prinln(date);
  System.out.prinln(newDate);
}
```

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.

```
public static void main(String args[]) {
  LocalDate date=LocalDate.parse("2018-10-30");
  LocalDate newDate=date.plusMonths(3);
  System.out.prinln(date);
  System.out.prinln(newDate);
}
```

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

Válida si una fecha es anterior que otra:

```
class Prueba {
  public static void main(String args[]) {
    System.out.prinln(LocalDate.parse("2018-10-30").isBefore(LocalDate.paa
rse("2018-10-31")));
  }
}
```

Válida si un año es bisiesto:

```
class Prueba {
  public static void main(String args[]) {
    System.out.prinln(LocalDate.parse("2018-10-30").isLeapYear());
  }
}
```

Podemos realizar validaciones muy simples sin necesidad de escribir código complejo.

Obtener el día de la semana de mi cumpleaños:

```
public static void main(String args[]) {
  System.out.prinln(LocalDate.parse("2019-08-19").getDayOfWeek());
}
```

Extraer información de una fecha:

```
public static void main(String args[]) {
  System.out.prinln(LocalDate.parse("2019-08-19").getMonth());
}
```

Si lo único que necesitamos de una fecha es el mes, podemos fácilmente extraerlo.

▼ 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()`.

```
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);
```

▼ Operaciones que podemos realizar con LocalTime

Modificar un LocalTime:

La primera operación que veremos es como modificar un LocalTime.

```
class Prueba {
  public static void main(String arg[]) {
    LocalTime time=LocalTime.parse("11:00:59.759");
    LocalTime time2=time.plusHours(1);
    System.out.println(time2);
```

```
  }
}
```

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.

```
public static void main(String arg[]) {
  LocalTime time=LocalTime.parse("11:00:59.759");
  LocalTime time2=LocalTime.parse("12:00:59.759");
  System.out.println(time.isBefore(time2));
}
```

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.

```
public static void main(String arg[]) {
  LocalTime time=LocalTime.parse("11:00:59.759");
  System.out.println(time.getHour());
}
```

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

▼ LocalDateTime

La siguiente clase a analizar será LocalDateTime la cual representa una combinación entre LocalDate y LocalTime, veamos como crearlo:

```
LocalDateTime dateTime=LocalDateTime.now();
LocalDateTime dateTime2=LocalDateTime.of(2018,10,10,11,25);
LocalDateTime dateTime3=LocalDateTime.parse("2018-10-10T11:25");
System.out.println(dateTime);
System.out.println(dateTime2);
System.out.println(dateTime3);
```

Podemos ver como la salida incluye fecha y hora por cada objeto.

▼ Operaciones que podemos realizar con estos objetos

Manipular un LocalDateTime:

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

```
public static void main(String arg[]) {
  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:

```
public static void main(String arg[]) {
  LocalDateTime dateTime=LocalDateTime.parse("2018-10-10T11:25");
  LocalDateTime dateTime2=LocalDateTime.parse("2019-10-10T11:25");
  System.out.println(dateTime.isBefore(dateTime2));
}
```

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

▼ Period

La clase Period se utiliza para modificar valores de una fecha u obtener la diferencia entre dos fechas.

En el siguiente ejemplo tomaré fecha y calcularé el periodo de tiempo que falta desde la fecha actual:

```
LocalDate currentDate=LocalDate.now();
LocalDate date=LocalDate.parse("2019-08-19");
Period period=Period.between(currentTime,date);
Sytem.out.println(String.format(DateTimeFormatter.ofPattern("Years %yyyy Months %MM Da
ys %dd",period.getYears(),period.getMonths(),period.getDays())));
```

▼ Duration

La clase Duration funciona de forma similar que Period la única diferencia es que en lugar de trabajar con fechas trabaja con tiempo, veamos el siguiente ejemplo:

Es la 1:35 pm y mi hora de salida del trabajo es a las 5:30 pm, calculemos cuantos minutos faltan para salir:

```
LocalTime currentTime=LocalTime.of(1,35);
LocalTime timeToLeave=LocalTime.of(5,30);
Duration duration=Duration.between(currentTime,timeToLeave);
System.out.println(String.format("Minutes %s",duration.toMinutes()));
```