

0101010
0100101
1101010

UD9.3- GESTIÓ DE DATES I TEMPS

Programació –1er DAW/DAM

0. CONTINGUTS

- ❖ Paquet `Java.util.date`
- ❖ Zones i fusos horaris
- ❖ Paquet `java.time`
 - La classe `LocalDate`
 - La classe `LocalTime`
 - La classe `LocalDateTime`
- ❖ Operacions
- ❖ Formats d'entrada i eixida
- ❖ Gestió de dates i temps amb zones horàries
 - La classe `ZonedDateTime`
 - La classe `ZoneDateTime`

1. Introducció

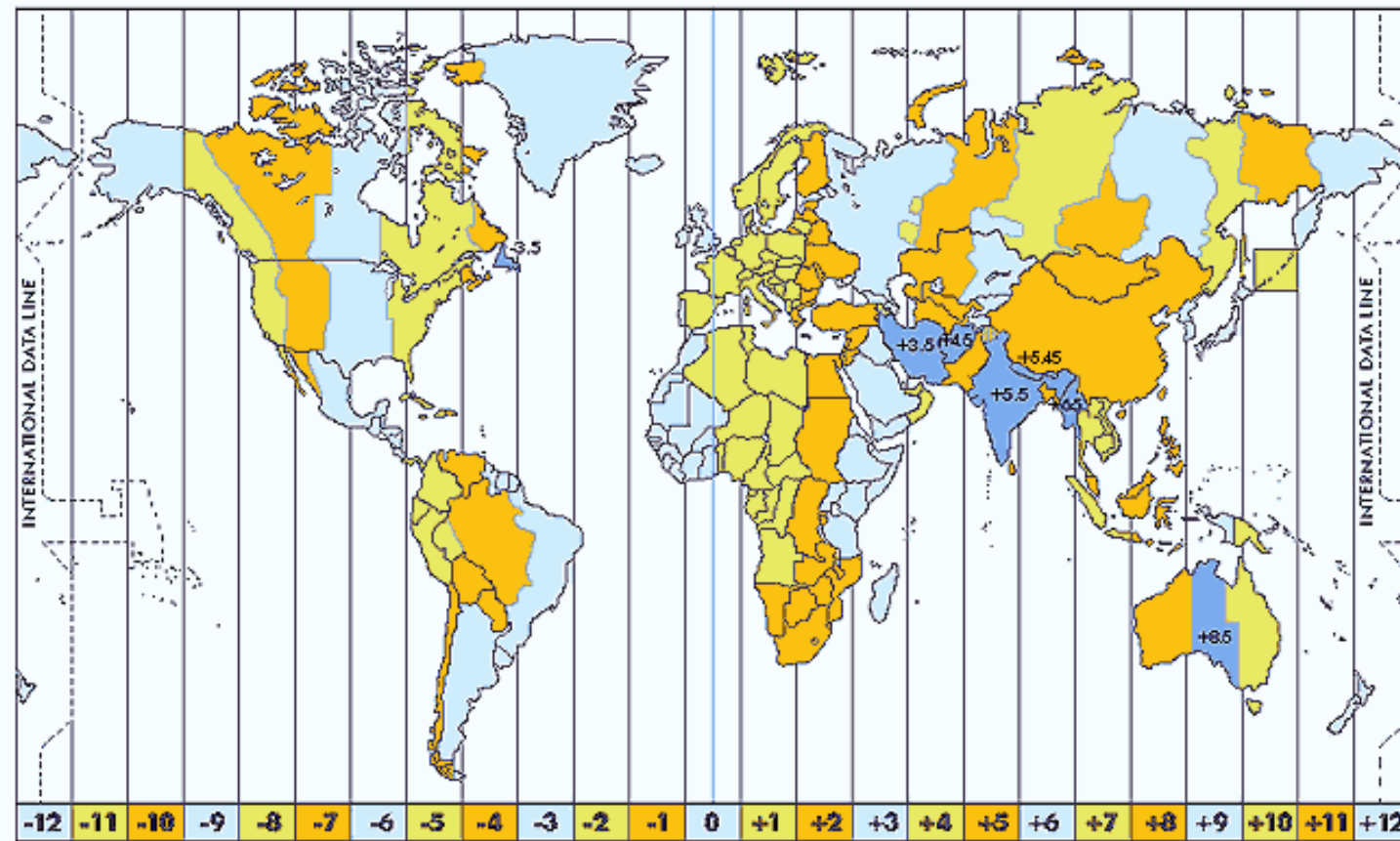
- Les gestió de **dates i temps** en una aplicació, és una tasca **complexa** per aspectes com:
 - Hem de mostrar la mateixa **hora** d'un registre a un usuari que es connecta des de **Londres i Madrid**?
 - **Horaris d'estiu i hivern.**
 - **Zones horàries.**
 - **Formats utilitzats** (dd/mm/yyyy , yyyy/mm/dd)



1. Introducció

1.1 Fusos horaris

- Representen cadascuna de les **24 àrees** en què **es divideix la terra** per meridians.



1. Introducció

1.1 Fus horari

- Cada **meridià** delimita una àrea de la terra on, per convenció, **hauria de regir el mateix horari**.
 - Cada **meridià** es diferencia 1 **hora** del seu adjacent.
 - Tots els fusos es defineixen a partir del meridià de **Greenwich** i difereixen 15° .

El meridià de Greenwich (meridià 0) representa el temps universal coordinat (UTC).

1. Introducció

1.2 Zona horària

- A la pràctica, per raons històriques o d'afinitat, **no es segueixen estrictament les definicions de temps** de cada fus horari.
- D'aquesta manera, sorgeixen les **zones horàries** que són **zones de la terra** que segueixen la **mateixa definició** de temps.



Exemple:

Espanya == Alemanya

Espanya != Londres

2. Maneig de Dates i temps en Java

Com representar una data o un instant de temps?

- Fins **Java 8** es feia mitjançant la classe `java.util.Date`
- Emmagatzema els **mil·lisegons** que han passat **des de l'1 de gener de 1970 fins** el moment **actual (temps epoch)** prenent com a referència una zona horària.

Unix Epoch	Date and Time (GMT)
111111111	Tue, 10 Jul 1973 00:11:51 +0000
1111111111	Fri, 18 Mar 2005 01:58:31 +0000
1222222222	Wed, 24 Sep 2008 02:10:22 +0000
1333333333	Mon, 02 Apr 2012 02:22:13 +0000
1444444444	Sat, 10 Oct 2015 02:34:04 +0000
1555555555	Thu, 18 Apr 2019 02:45:55 +0000

Actualment ha estat **reemplaçada** pel paquet **java.time**

2. Maneig de Dates i temps en Java

- A partir de **Java 8** disposem el paquet `java.time` que **simplifica** enormement el tractament. Podem destacar les classes:
 - `java.time.LocalDate`: Representa una data sense tenir en compte la zona horaria ni el fus horari (Ex. 11/02/2020).
 - `java.time.LocalDateTime`: Representa un temps en una **precisió de nanosegons** sense tenir en compte la zona horària ni el fus horari (Ex. 10:15:30).
 - `java.time.LocalDateTime`: Representa un temps i una data combinats. (11/02/202010:15:30)

Les 3 classes representen objectes immutables

<https://docs.oracle.com/javase/8/docs/api/index.html>

3. java.time.LocalDate

3.1 Constructors

Els constructors de la classe `LocalDate` són **estàtics**:

- `LocalDate.now()`: Instància un nou objecte que representa la data actual del sistema.

```
public static void main(String[] args) {  
    LocalDate localDate = LocalDate.now();  
}
```

- `LocalDate.of(int year, int month, int day)`

```
public static void main(String[] args) {  
    LocalDate date = LocalDate.of(2020,11,11); //11/11/2020  
}
```

3. java.time.LocalDate

3.1 Constructors

- `LocalDate.parse(CharSequence textDate)` : Crea un objecte a partir de la data que representa la cadena **@textDate**.
 - El format de la cadena entrant ha de ser **ISO-8601** (yyyy-mm-dd).

```
public static void main (String[] args){  
  
    LocalDate date = LocalDate.parse("2020-11-11"); //11-11-2020  
  
}
```

3. java.time.LocalDate

3.2 Mètodes

<code>int getYear()</code>	Retorna l' any de la data representada per l'objecte
<code>int getDayOfYear()</code> <code>int getDayOfWeek().getValue()</code> <code>int getDayOfMonth()</code>	Retorna el dia de la data representada per l'objecte
<code>int getMonthValue()</code>	Retorna el mes de la data representada per l'objecte
<code>LocalDate plusDays(int days)</code> <code>LocalDate plusMonths(int months)</code> <code>LocalDate plusWeeks(int weeks)</code>	Retorna un nou objecte <code>LocalDate</code> amb la suma dels dies, mesos o anys indicats
<code>LocalDate minusDays(int days)</code> <code>LocalDate MinusMonths(int months)</code> <code>LocalDate minusWeeks(int weeks)</code>	Retorna un nou objecte <code>LocalDate</code> amb la resta dels dies, mesos o anys indicats
<code>boolean isAfter(ChronoLocalDate other)</code> <code>boolean isBefore(ChronoLocalDate other)</code>	Compara 2 objectes de tipus <code>LocalDate</code> i torna un booleà indicant si és més gran o més petit



3. java.time.LocalDate

3.3 Exemple

```
public class Example1 {  
  
    public static void main(String[] args){  
  
        LocalDate date = LocalDate.parse("2020-11-20");  
        System.out.println(date.getYear()); // 2020  
        System.out.println(date.getMonthValue()); // 11  
        System.out.println(date.getDayOfMonth()); // 20  
        System.out.println(date.getDayOfYear()); // 325  
  
        LocalDate dateAfter = date.plusDays(5);  
        System.out.println(dateAfter.isBefore(date)); // no  
        System.out.println(dateAfter.isAfter(date)); // yes  
        System.out.println(dateAfter.getDayOfYear()); // 330  
    }  
}
```

4. java.time.LocalDateTime

4.1 Constructors

Els constructors de la classe `LocalTime` són **estàtics**:

- `LocalTime.now()`: Instància un nou objecte que representa l'hora actual del sistema.

```
public static void main(String[]args) {  
    LocalTime time = LocalTime.now();  
}
```

- `LocalTime.of(int hour, int minute, int second)`

```
public static void main(String[]args) {  
    LocalTime time = LocalTime.of(10,30,00); //10:30:00  
}
```

4. java.time.LocalDateTime

4.1 Constructors

- `LocalTime.parse(CharSequence textDate)`: Crea un objecte `LocalTime` a partir de la data que representa la cadena `textDate`.
 - Per defecte, el format de la cadena entrant ha de ser **ISO-8601** (*hh:mm:ss*).

```
public static void main(String[] args) {  
    LocalDateTime time = LocalDateTime.parse("20:11:00");  
}
```

4. java.time.LocalDateTime

4.2 Mètodes

<code>int getHour()</code>	Retorna l'hora del temps representat per l'objecte
<code>int getMinutes()</code>	Retorna els minuts del temps representat per l'objecte
<code>int getSeconds()</code>	Retorna els segons del temps representat per l'objecte
<code>LocalTime plusHours(int hours)</code> <code>LocalTime plusSeconds(int seconds)</code> <code>LocalTime plusMinutes(int minutes)</code>	Retorna un nou objecte LocalDateTime amb la suma de les hores, segons o minuts indicats
<code>LocalTime minusHours(int hours)</code> <code>LocalTime minusSeconds(int seconds)</code> <code>LocalTime minusMinutes(int minutes)</code>	Retorna un nou objecte LocalDateTime amb la resta de les hores, segons o minuts indicats
<code>boolean isAfter(ChronoLocalTime other)</code> <code>boolean isBefore(ChronoLocalTime other)</code>	Compara 2 objectes de tipus LocalDateTime i torna un booleà indicant si és més gran o més petit



4. java.time.LocalDateTime

4.3 Exemple

```
public class Example2 {  
  
    public static void main(String[] args){  
  
        LocalDateTime time = LocalDateTime.parse("20:11:00");  
  
        System.out.println(time.getHour());    //20  
        System.out.println(time.getMinute()); //11  
        System.out.println(time.getSecond()); //00  
  
        LocalDateTime timeAfter = time.plusHours(5);  
        System.out.println(timeAfter.isBefore(time)); //false  
        System.out.println(timeAfter.isAfter(time));  //true  
  
    }  
  
}
```

5. java.time.LocalDateTime

- **Combina** les 2 **classes** anteriors i representa un instant de temps determinat (**data i hora**).
- Igual que les anteriors **no té una zona horària específica**. (Serà la del **sistema en què s'execute**).
- Presenta mètodes similars als **analitzats anteriorment**.

```
public static void main(String[] args){  
    /** Creació objecte a partir de (any, mes, dia, hora, minut, segon,  
    * nanosegon) */  
    LocalDateTime dateTime = LocalDateTime.of(1989, 11, 11, 5, 30, 45, 35);  
    //1989-11-11T05:30:45.000000035  
  
    //creant un objecte del moment actual  
    LocalDateTime dateTimeNow = LocalDateTime.now();  
}
```



5. java.time.LocalDateTime

5.3 Exemple

```
public class Example3 {

    public static void main(String[] args){

        LocalDateTime date = LocalDateTime.parse("2023-03-21T08:30:35");
        showDateDetails(date);

        LocalDateTime dateAfter = date.plusDays(5).plusHours(5).plusMinutes(20);
        showDateDetails(dateAfter);

        System.out.println(dateAfter.isBefore(date)); //false
        System.out.println(dateAfter.isAfter(date)); //true

    }

    private static void showDateDetails(LocalDateTime date) {
        System.out.println(date.getYear());
        System.out.println(date.getMonth().getValue());
        System.out.println(date.getDayOfMonth());
        System.out.println(date.getDayOfYear());
        System.out.println(date.getHour());
        System.out.println(date.getMinute());
    }
}
```

2023
3
21
80
8
30

2023
3
26
85
13
50

5.1 Activitat

Activitat 12.- Crea un programa que mostre la data i l'hora actual del sistema. A més, haureu de donar l'opció a l'usuari de mostrar només l'any, el mes o el dia.

Exemple execució

```
La data i l'hora actual del sistema és: 2021-04-
```

```
14T19:42:21.479366
```

```
Què vols visualitzar?:
```

```
1) Dia
```

```
2) Mes
```

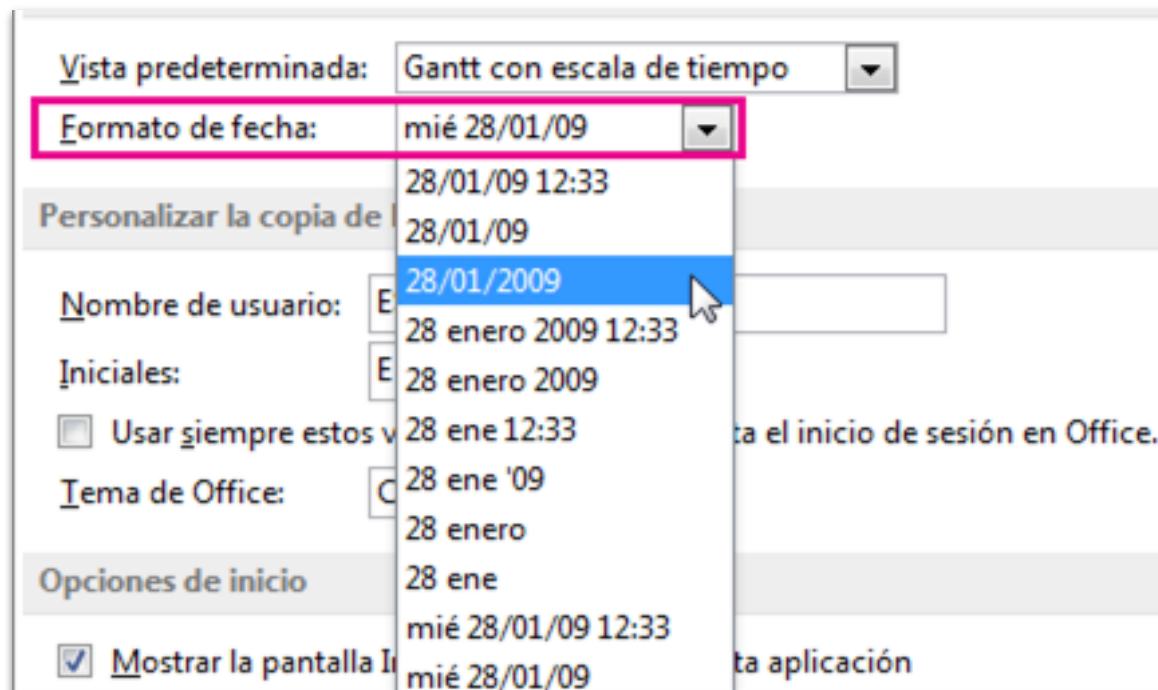
```
3) Any
```

```
1
```

```
Dia = 14
```

6. Format de dates.

- Les classes anteriorment analitzades **representen** una **data**, **hora** o **ambdues** al sistema.
- Quan necessitem mostrar-la a l'usuari haurem de convertir-la a una representació en **format String**.
- **Cada país o sistema d'informació** fa ús d'un **format** determinat.



6. Format de dates.

6.1 La classe `java.time.format.DateTimeFormatter`

- Ens permet especificar el format amb què volem que es visualitzi una **data / hora concreta** d'un objecte `LocalDate`, `LocalTime` o `LocalDateTime`.
- Per això s'utilitzen una sèrie de **meta-caràcters** que podem consultar a la [api de java](#).

```
public class Example3 {  
    public static void main(String[] args){  
        LocalDate nowDate = LocalDate.of(2020,2,20);  
        //Creem el formatador  
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy/MM/dd");  
        // Ho apliquem per formatar la data  
        System.out.println(nowDate.format(formatter)); //2020/02/20  
    }  
}
```

meta-
caràcters

6. Format de dates.

6.1.1 Exemple de patrons

Patró	Exemple
dd-MMM-yyyy d dia del mes (dd → 2 digits) M mes de l'any (MMM → 3 caràcters) y any de l'era (yyyy → 4 digits)	14-Jul-2018 12-Apr-2020
yyyy-MM-dd d dia del mes (dd → 2 digits) M mes de l'any (MM → 2 digits) y any de l'era (yyyy → 4 digits)	2018-07-14 2020-04-12
dd/MM/yyyy d dia del mes (dd → 2 digits) M mes de l'any (MM → 2 digits) y any de l'era (yyyy → 4 digits)	14/07/2018 12/04/2020
hh:mm:ss a h hora del dia 1-12 (hh → 2 digits) m minuts de l'hora (mm → 2 digits) s segons de l'hora (ss → 2 digits) a am-pm-of-day	12:08 PM 02:08 AM
dd/MM/yyyy HH:mm:ss HH hora del dia 00-24 (hh → 2 digits)	11/02/2020 23:30:00 10/01/2020 12:30:00

6. Format de dates.

6.1.2 Formats de sortida

- D'aquesta manera podem donar **qualsevol format** de sortida a un objecte de tipus `LocalDate`, `LocalTime` o `LocalDateTime`.

```
public class Example4 {  
  
    public static void main(String[] args) {  
  
        LocalTime hora = LocalTime.now();  
        DateTimeFormatter f = DateTimeFormatter.ofPattern("'Són les ' h ' i ' mm");  
        System.out.println(hora.format(f)); // Són les 4 i 40  
  
        LocalDate date = LocalDate.now();  
        DateTimeFormatter f = DateTimeFormatter.ofPattern("dd-MMM-yyyy");  
        System.out.println(date.format(f)); // 21-Apr-2020  
  
    }  
}
```

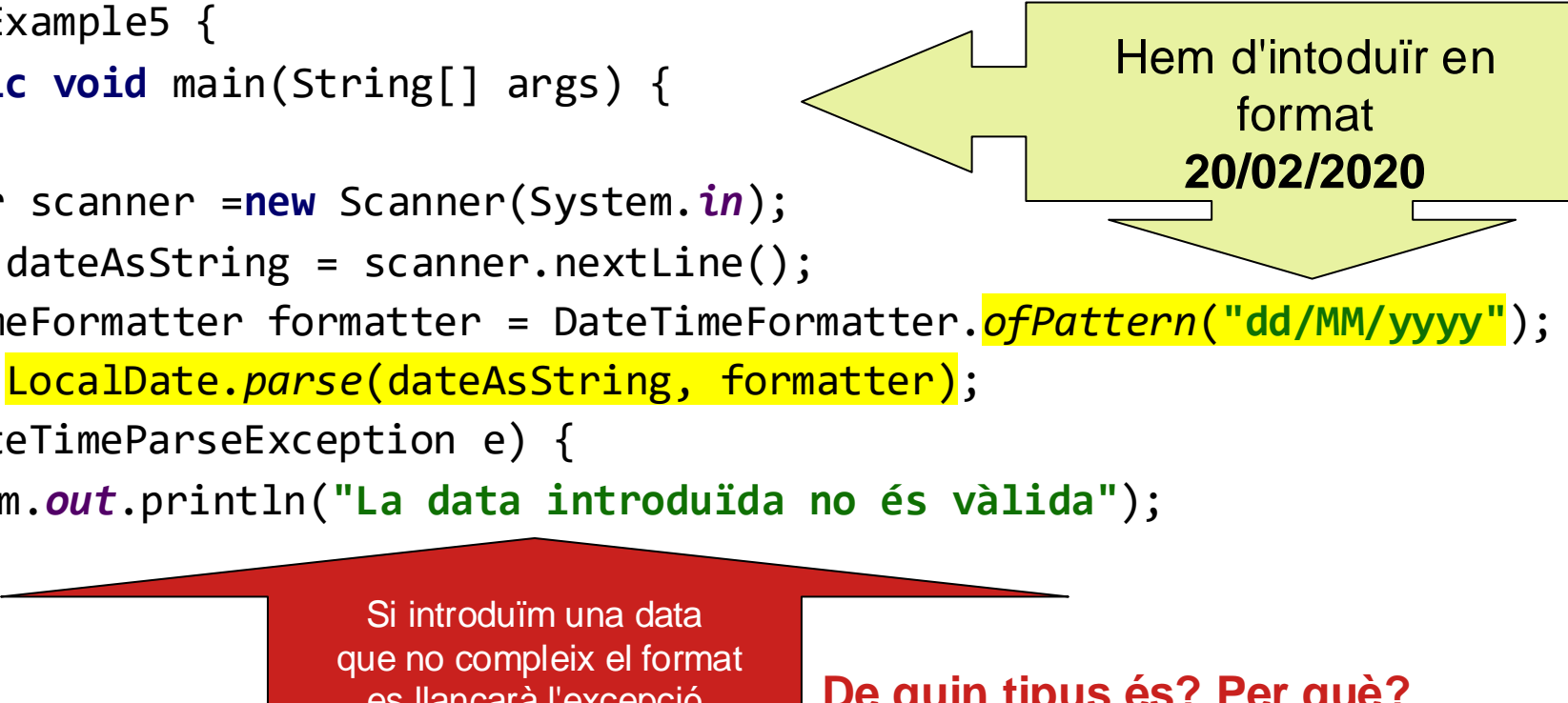


6. Format de dates.

6.1.2 Formats d'entrada

- De la mateixa manera, ens pot ser útil per especificar el format en què estem introduint la **data/hora** quan truquem al mètode estàtic `parse()` per construir un objecte de tipus `LocalDate`, `LocalTime` o `LocalDateTime`.

```
public class Example5 {  
    public static void main(String[] args) {  
        try {  
            Scanner scanner = new Scanner(System.in);  
            String dateAsString = scanner.nextLine();  
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");  
            return LocalDate.parse(dateAsString, formatter);  
        } catch (DateTimeParseException e) {  
            System.out.println("La data introduïda no és vàlida");  
        }  
    }  
}
```



Hem d'introduir en format
20/02/2020

Si introduïm una data que no compleix el format es llançarà l'excepció

De quin tipus és? Per què?

6. Format de dates.

6.1.3 Formatadors predefinits

- La **Api oficial** proveeix una sèrie de formatadors amb els formats **ISO** més utilitzats

- ✓ **ISO_LOCAL_DATE** puede utilizarse con *LocalDate* y *LocalDateTime* (contienen una fecha).
- ✓ **ISO_LOCAL_TIME** puede utilizarse con *LocalTime* y *LocalDateTime* (contienen una hora).
- ✓ **ISO_LOCAL_DATE_TIME** puede utilizarse solo con *LocalDateTime* (contiene una fecha y una hora).

```
public class Example6 {  
  
    public static void main(String[] args){  
        LocalDateTime dataHora = LocalDateTime.now();  
  
        DateTimeFormatter isoFecha = DateTimeFormatter.ISO_LOCAL_DATE;  
        System.out.println(dataHora.format(isoData));  
  
        DateTimeFormatter isoHora = DateTimeFormatter.ISO_LOCAL_TIME;  
        System.out.println(dataHora.format(isoHora));  
  
        DateTimeFormatter isoFechaHora = DateTimeFormatter.ISO_LOCAL_DATE_TIME;  
        System.out.println(dataHora.format(isoDataHora));  
    }  
}
```

6.1 Activitat Prèvia

Activitat 13.- Creeu un programa que a partir d'una data i hora introduïda per l'usuari en format **YYYY/MM/DD hh:mm:ss**, la mostre en format local **DD/MM/YYYY hh:mm:ss**

Exemple execució

Introdueix una data en format (YYYY/MM/DD hh:mm:ss)

11/02/2021 00:00:00

La data i hora és: 11/02/2021 00:00:00

7. Maneig de Zones Horàries

7.1 La classe `java.time.ZoneId`

- La classe `java.time.ZoneId` ens permet identificar les diferents zones horàries del planeta, així podem:
 - **Especificar la zona horària** en què volem obtenir la **data/hora** del sistema en què volem treballar:

```
// Creem un objecte LocalDateTime amb l'hora a la zona Europe/London  
timezone (UTC)
```

```
LocalDateTime now = LocalDateTime.now(ZoneId.of("Europe/London"));
```

7. Maneig de Zones Horàries

7.1 La classe `java.time.ZoneId`

- **Llistar** tots els possibles **identificadors** de totes les zones existents:

```
//list all timezones  
System.out.println(ZoneId.getAvailableZoneIds());
```

```
[Asia/Aden, America/Cuiaba, Etc/GMT+9, Etc/GMT+8, Africa/Nairobi,  
America/Marigot, Asia/Aqtau, Pacific/Kwajalein, America/El_Salvador,  
Asia/Pontianak, Africa/Cairo, Pacific/Pago_Pago, etc]
```

- Consultar la zona configurada al sistema de la màquina on s'executa l'aplicació:

```
//Obté la zona configurada del sistema  
System.out.println(ZoneId.systemDefault());
```

8. Activitat Prèvia

- **Activitat 14.** Crea un programa que mostre l'hora actual a les zones horàries següents.
 - Europe/Madrid
 - Africa/Cairo
 - Europe/London

Exemple execució

L'hora a Europe/Madrid és 09:19:29

L'hora a Africa/Cairo és 10:19:29

L'hora a Europe/London és 08:19:29

8. Activitat Prèvia

- Això és tot... de moment :-)