JAVA @17

DateTime API

Objective

After completing this session, you will be able to understand,

- Java LocalDate Class and methods
- Java DateTimeFormatter Class
- Java LocalTime Class and methods
- Java LocalDateTime class and methods

LocalDate

LocalDate is an immutable class (present in java.time package) that represents a date with default format "**yyyy-MM-dd"** (year-month-day).

We can access other date fields, such as day-of-year, day-of-week and week-of-year from this class .

For example, the value "4th Jan 2007" can be stored in a LocalDate in the default format (2007-01-04).

This class does not store or represent a time or time-zone.

This class consists of default methods such as minusDays, minusMonths, minusYears, now, plusDays etc which help us in various scenarios where date is concerned.

Now, we will see the implementation of some most used methods of LocalDate Class.

LocalDate

This class consists of default methods such as minusDays, minusMonths, minusYears, now, plusDays etc which help us in various scenarios where date is concerned.

Now, we will see the implementation of some most used methods of LocalDate Class.

```
1. public class TestMain {
2.     public static void main(String[] args) {
3.         LocalDate today = LocalDate.now();
4.         // now() returns the current date from system clock
5.         System.out.println(today); // output will be today's date
6.         LocalDate dateObj = LocalDate.of(1997, 8, 29);
7.         // of() returns the instance of LocalDate
8.         //with specified arguements
9.         System.out.println(dateObj); // output : 1997-08-29
10.    }
11.}
```

DateTimeFormatter

Consider a scenario where we want to display the date (given above) i.e. 1997-08-29 in format of dd/MM/yy (i.e. 29/08/97).

In that case what can we do ?

To solve this problem Java provides us with a different class which is used for parsing dates in different formats. This class is called as DateTimeFormatter (present in java.time package). Implementation of the same is given below.

```
1.  public class TestMain {
2.      public static void main(String[] args) {
3.          LocalDate today = LocalDate.now();
4.          // now() returns the current date from system clock
5.          System.out.println(today); // output will be today's date
6.          LocalDate dateObj = LocalDate.of(1997, 8, 29);
7.          // of() returns the instance of LocalDate
8.          //with specified arguments
9.          DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yy");
10.         System.out.println(df.format(dateObj));
11.     }
12. }
```

LocalDate Implemetation

LocalDate provides methods with which we can manipulate the date,

for example plusYears() - which returns a copy of LocalDate with the specified number of years added, similarly minusDays(), minusWeeks(), etc.

```
1. public class TestMain {
2.     public static void main(String[] args) {
3.         LocalDate newDate = LocalDate.now();
4.         newDate = newDate.plusDays(34);
5.         System.out.println(newDate);   // TodayDate + 34 days
6.         newDate = newDate.plusYears(45);
7.         System.out.println(newDate);   // newDate + 45 Years
8.         newDate = newDate.minusWeeks(52);
9.         System.out.println(newDate);   // newDate - 52 weeks
10.    }
11.}
```

LocalDate Implementation

Now we will see some of the methods which will allow us to get the difference between two dates.

Java8 provides us ChronoUnit which is an enum, it is standard set of date periods units.

This set of units provide unit-based access to manipulate a date, time or date-time.

In example given below we are calculating difference between two date objects in Days and Months.

```
1.  public class TestMain {
2.     public static void main(String[] args) {
3.        LocalDate today = LocalDate.now();
4.        LocalDate newDateObj = today.minusWeeks(12);
5.        System.out.println(ChronoUnit.DAYS.between(newDateObj,today));  //84
6.        //between() calculates the amount of time between specified date objects
7.        System.out.println(ChronoUnit.MONTHS.between(newDateObj,today)); //3
8.        //compareTo() compares this date with specified date
9.        System.out.println(newDateObj.compareTo(today)); //-3
10. }
11. }
```

LocalDate - Methods

Method	Description
LocalDate now()	The current system clock date is returned.
LocalDate plusDays(long daysToAdd)	A LocalDate with the number of days added as specified is returned.
LocalDate plusMonths(long monthsToAdd)	A LocalDate with the number of months added as specified is returned.
LocalDate minusMonths(long monthsToSubtract)	A LocalDate with the number of months subtracted as specified is returned.
LocalDate minusDays(long daysToSubtract)	A LocalDate with the number of days subtracted as specified is returned.
boolean equals(Object obj)	Checks equality of two dates.
int compareTo(ChronoLocalDate other)	Compares the given date to another date. Returns 0, 1, -1 based on the equality of the specified dates.
int get(TemporalField field)	Fetches the value of the specified field from the given date as an integer.
String format(DateTimeFormatter formatter)	Formats the given date according to the given formatter.

LocalTime

LocalTime is a class that represents time with a default format of "hour-minute-second" (hh-mm-ss.zzz).

It is immutable by nature.

Time is represented to the precision of nanosecond. For example, the value "09:11:44.140" can be stored in a LocalTime in the default format.

LocalTime Class is present in java.time package.

LocalTime Methods

This class consists of default methods such as minusHours, minusMinutes, of, now, plusHours etc which help us in various scenarios where time is concerned.

Now we will see the implementation of some most used methods.

```
1. public class TestMain {
2.     public static void main(String[] args) {
3.         LocalTime currentTime = LocalTime.now();
4.         System.out.println(currentTime);
5.         // will display current time eg. 15:44:44.932359900
6.         LocalTime newTimeObj = LocalTime.of(9, 45, 59);
7.         System.out.println(newTimeObj); //09:45:59
8.         DateTimeFormatter df = DateTimeFormatter.ofPattern("h:mm:ss");
9.         System.out.println(df.format(currentTime));
10.        //will give output in specified format eg.3:44:44
11.    }
12.}
```

LocalTime Methods

As we have seen ChronoUnit and DateTimeFormatter in LocalDate, we will try methods related to those classes in tryout.

To access the fields (hour, minute, etc.) we use get method. Implementation of the same is given below.

```
1. public class TestMain {
2.     public static void main(String[] args) {
3.         LocalTime newTimeObj = LocalTime.of(9, 45, 59);
4.         int hour = newTimeObj.getHour();
5.         int second = newTimeObj.getSecond();
6.         int minute = newTimeObj.getMinute();
7.         System.out.println(hour+":"+second+":"+minute); //9:45:59
8.     }
9. }
```

LocalTime Methods

Rest of the methods we will learn in tryout.

Description of some methods, of the LocalTime class, is given below.

Method	Description
LocalTime now()	The current system clock time is returned.
LocalTime of(int hour, int minute, int second)	According to the values of hours, minutes and seconds a LocalTime is returned.
LocalTime plusHours(long hoursToAdd)	A LocalTime with number of hours added as specified is returned.
LocalTime plusMinutes(long minutesToAdd)	A LocalTime with number of minutes added as specified is returned.
LocalTime minusHours(long hoursToSubtract)	A LocalTime with number of hours subtracted as specified is returned.
LocalTime minusMinutes(long minutesToSubtract)	A LocalTime with number of minutes subtracted as specified is returned.
int get(TemporalField field)	Fetches the value of the specified field from the given time as an integer.
String format(DateTimeFormatter formatter)	Formats the given time using the specified formatter.

LocalDateTime

LocalDateTime is an immutable class that represents a date-time, with the default format as "yyyy-MM-dd-HH-mm-ss.zzz" year-month-day-hour-minute-second.

Other date and time fields, such as day-of-year, day-of-week and week-of-year, can also be accessed.

Time is represented to nanosecond precision.

For example, the value "2nd October 2007 at 13:45.30.123456789" can be stored in a LocalDateTime.

This class does not store or represent a date or time-zone.

LocalDateTime Methods

This class consists of default methods such as minusDays, minusMonths, minusYears, now, plusDays etc which help us in various scenarios where date is concerned.

We will see the implementation of some methods.

```
1. public class TestMain {
2.     public static void main(String[] args) {
3.         LocalDateTime currentDate = LocalDateTime.now();
4.         System.out.println(currentDate);
5.         // output in default format eg. 2022-10-22T23:57:46.041224900
6.         System.out.println(currentDate.getDayOfWeek());
7.         // will give today's day eg. SATURDAY
8.     }
9. }
```

LocalDateTime Methods

If we want to display DateTime in specific format we use DateTimeFormatter and format method, as shown.

```
1.  public class TestMain {
2.      public static void main(String[] args) {
3.          LocalDateTime currentDate = LocalDateTime.now();
4.          System.out.println(currentDate);
5.          DateTimeFormatter df = DateTimeFormatter.ofPattern("dd/MM/yyyy/ hh:mm:ss a");
6.          // a is used to display 12 hr clock
7.          System.out.println(df.format(currentDate));
8.          //output in specified format eg. 23/10/2022/ 12:00:27 AM
9.      }
10. }
```

LocalDateTime Methods

Now, consider a situation where it is required to display the day,

can we do it using DateTimeFormatter class?

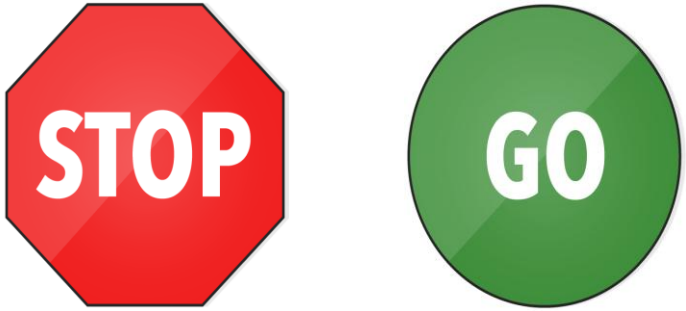
```
1.  public class TestMain {
2.     public static void main(String[] args) {
3.        LocalDateTime currentDate = LocalDateTime.now();
4.   DateTimeFormatter df = DateTimeFormatter.ofPattern("eeee dd/MM/yyyy/ hh:mm:ss a");
5.        System.out.println(df.format(currentDate));
6.        //output :- Sunday 23/10/2022/ 12:07:23 AM
7.   DateTimeFormatter dfNew = DateTimeFormatter.ofPattern("eee dd/MM/yyyy/ hh:mm:ss a");
8.        System.out.println(dfNew.format(currentDate));
9.        //output :- Sun 23/10/2022/ 12:07:23 AM
10.    }
11. }
```

As we can see we used modified the pattern with eeee or eee which displays the day of the date.

"eee” – Displays starting three letters of Day of the week such as “Mon”, “Tue”, etc.
“eeee” – Displays the full name of the Day of the week such as “Monday”, “Tuesday” etc.

Can we try "ee" ?

Time To Reflect



Trainees to reflect the following topics before proceeding.

- What is the class you use for processing dates?
- How to create customized format?
- What class can we use for formatting dates based on locales?

Thank you

You have successfully completed

Java DateTime API