Date-Time api is introduced in java 8

1. Overview

In this java 8 programming tutorial, we will be learning today **Date-Time api is introduced in java 8**. Before that first, we will see what are the drawbacks of ***java.util.Date api*** before java 8. And next, features of new Date Time api. What are the new classes introduced? Working example programs on each class and how the dates are simplified using with timezones.

2. Drawbacks of java.util.Date API

API ***java.util.Date*** drawbacks before java 8.

2.1 Not Thread Safe

Older version of Date api does not support for handling concurrent scenarios. So, It is not thread-safe. That means, developers need to handle multithreaded issues and have to write many lines of code in a consistent manner. The new api has introduced the support for thread-safe and more easy now to deal with the real-time concurrent scenarios.

2.2 Poor Desing and Lesser Operations

Old api has default values as Date starts from 1990, the month starts from 1 and day starts from 0. There is no uniformity here. And also old api provides fewer methods to do some operations whereas new api has come up with the various utility methods for such operations.

2.3 Difficulty in dealing with Timezone dates

I have a scenario in my career when I started my carrier. We were working with multiple clients, we need to show the times in their local timezones and found that this is not an easy job to do. But, we implemented somehow. Later some time, started seeing the differences when the daylight saving started. For this, we had to do some tweaks.
But, You are good to deal with timezones using Java 8 api and it made very simple. We will see the example programs on timezones.

3. New Java 8 Date API changes

As of now, We have seen the problems that we were facing before java. All are now addressed in new ***DateTime*** api. All new classes are placed under package ***java.time***.

Below are the two base class types and this is a category type that classes start with their name. But, many popular websites are describing these two are classes in java and I have searched but could not find these Local and Zoned as a class or interface in Java 8 api.

A) Local
B) Zoned

3.1 Local classes

Local − Simplified date-time API classes with no complexity of timezone handling.

Local API classes.

LocalDate
LocalTime
LocalDateTime

All these classes are Immutable classes and have private constructors. So, directly objects can not be created and should use the now() or of() methods to create its instances.

3.2 Zoned classes

Zoned - Specialized date-time API to deal with various timezones.

The zoned type has the only implementation class.

ZonedDateTime

In addition to this, there is a new concept ***TemporalAdjuster, ChronoUnits, Period and Duration*** added in java 8.

4. New Date Time API Examples

Let us write a simple example program for these classes.

How to compare two dates in java 8 ?

4.1 LocalDate Example

A date without a time-zone in the ISO-8601 calendar system, such as 2007-12-03 and it is in package ***java.time.LocalDate***.

This handles only date but not time.

```
package com.javaprogramto.w3schools.programs.java8.dateapi;

import java.time.LocalDate;
import java.time.Month;

public class LocalDateExample {

    public static void main(String[] args) {

        // Getting the current LocalDate object.
        LocalDate now = LocalDate.now();
```

```
        System.out.println("Present date : " + now);

        // getting year
        int year = now.getYear();

        // getting month
        Month month = now.getMonth();

        // getting day
        int day = now.getDayOfMonth();

        // printing all these three values.
        System.out.println("year : " + year + " , month : " + month + ",
day : " + day);

        // getting month in int value
        int monthInt = now.getMonthValue();

        // printing in format YYYY-MM-DD
        System.out.println("today date : " + year + "-" + monthInt + "-" +
day);

        LocalDate oldDate = LocalDate.of(2015, 2, 2);
        System.out.println("Old date : " + oldDate);
```

```
    }

}
```

Output:

```
Present date : 2019-12-26
year : 2019 , month : DECEMBER, day : 26
today date : 2019-12-26
Old date : 2015-02-02
```

4.2 LocalTime Example

A time without a time-zone in the ISO-8601 calendar system, such as 10:15:30. This handles only with time and not with a date.

```
package com.javaprogramto.w3schools.programs.java8.dateapi;
/**
 * Java program to LocalTime class
 *
```

```
 */

import java.time.LocalTime;

public class LocalTimeExample {

    public static void main(String[] args) {

        // Getting the curent LocalDate object.
        LocalTime now = LocalTime.now();
        System.out.println("Present time : " + now);

        // getting hour
        int hour = now.getHour();

        // getting minutes
        int minute = now.getMinute();

        // getting second
        int second = now.getSecond();

        // printing all these three values.
        System.out.println("Hour : " + hour + " , Minute : " + minute + ",
Second : " + second);
```

```
        // getting nano seconds
        int nanoSec = now.getNano();

        // printing in format HH:24MM:ss:SSS
        System.out.println("time now in format : " + hour + ":" + minute
+ ":0" + second + ":" + nanoSec);

        LocalTime oldTime = LocalTime.of(13, 20, 25);
        System.out.println("Old time : " + oldTime);

    }

}
```

Output:

```
Present time : 15:12:15.181
Hour : 15 , Minute : 12, Second : 15
time now in format : 15:12:015:181000000
Old time : 13:20:25
```

4.3 LocalDateTime Example

A date-time without a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30. This works with both date and time at a time.

```
package com.javaprogramto.w3schools.programs.java8.dateapi;

/**
 *
 * Java program to LocalDateTime class and examples
 *
 */
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;

public class LocalDateTimeExample {

   public static void main(String[] args) {

      // Getting the curent LocalDate object.
      LocalDateTime now = LocalDateTime.now();
      System.out.println("Present date : " + now);
```

```
        // getting year
        int year = now.getYear();

        // getting month
        Month month = now.getMonth();

        // getting day
        int day = now.getDayOfMonth();

        // getting hour
        int hour = now.getHour();

        // getting minutes
        int minute = now.getMinute();

        // getting second
        int second = now.getSecond();

        // printing all these three values.
        System.out.println("year : " + year + " , month : " + month + ",
day : " + day + "Hour : " + hour
                + " , Minute : " + minute + ", Second : " + second);
```

```
        // creating the older date using of() method.
        LocalDateTime oldDate = LocalDateTime.of(1990, 12, 10, 10, 10,
10);
        System.out.println("Old date : " + oldDate);


        // Creating a new LocalDate instance using toLocalDate()
method.
        LocalDate newlocalDate = now.toLocalDate();
        System.out.println("newlocalDate : " + newlocalDate);


        // Creating a new LocalTime instance using toLocalDate()
method.
        LocalTime newlocalTime = now.toLocalTime();
        System.out.println("newlocalTime : " + newlocalTime);


        // Date parsing from string to date
        LocalDateTime strToDate = LocalDateTime.parse("2019-10-
10T10:10:10");
        System.out.println("String to Date : " + strToDate);


    }


}
```

Output:

Present date : 2019-12-26T15:42:21.803

year : 2019 , month : DECEMBER, day : 26Hour : 15 , Minute : 42, Second : 21

Old date : 1990-12-10T10:10:10

newlocalDate : 2019-12-26

newlocalTime : 15:42:21.803

String to Date : 2019-10-10T10:10:10

4.4 ZonedDateTime Example

ZonedDateTime is used to work with timezones. If you are not working or not important to the timezones then you should use the LocalDateTime class.

A date-time with a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30+01:00 Europe/Paris. ZonedDateTime is an **immutable** object. Once it is created then its values can not be changed.
The ZoneId is an identifier used to represent different zones.

In this example, converting India time to Australia and Los_Angeles time zones.

package com.java.w3schools.blog.java.program.to.java8.datetime;

```
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

/**
 *
 * Java program to ZonedDateTime
 *
 * @author JavaProgramTo.com
 *
 */
public class ZonedDateTimeExample {

 public static void main(String[] args) {

  LocalDateTime currentDateTime = LocalDateTime.now();

  System.out.println("India current DateTime : " + currentDateTime);

  DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd
MMM yyyy HH:mm:ss");
```

```
  ZonedDateTime zonedDateTime =
currentDateTime.atZone(ZoneId.of("Australia/Sydney"));


  System.out.println("Aus Zoned time : " +
formatter.format(zonedDateTime));


  ZoneId zoneId = zonedDateTime.getZone();
  System.out.println("zone id : " + zoneId);


  ZonedDateTime pstZoneTime =
zonedDateTime.withZoneSameInstant(ZoneId.of("America/Los_Ange
les"));
  System.out.println("PST time now : " +
formatter.format(pstZoneTime));
 }


}
```

Output:

```
India current DateTime : 2019-12-27T00:29:06.803908
Aus Zoned time : 27 Dec 2019 00:29:06
zone id : Australia/Sydney
PST time now : 26 Dec 2019 05:29:06
```

4.5 TemporalAdjusters Example

TemporalAdjusters is a utility class to provide the support for date-time api.

```
package com.java.w3schools.blog.java.program.to.java8.datetime;

import java.time.LocalDate;
import java.time.temporal.TemporalAdjusters;

/**
 *
 * TemporalAdjuster example
 *
 * @author JavaProgramTo.com
 *
 */
public class TemporalAdjusterExample {

 public static void main(String[] args) {

  LocalDate localDate = LocalDate.now();
```

```
  System.out.println("Current Date : " + localDate);

  // get first day of the month
  LocalDate firstDayOfMonth =
localDate.with(TemporalAdjusters.firstDayOfMonth());
  System.out.println("firstDayOfMonth : " + firstDayOfMonth);

  // get first day of next month
  LocalDate firstDayOfNextMonth =
localDate.with(TemporalAdjusters.firstDayOfNextMonth());
  System.out.println("firstDayOfNextMonth : " +
firstDayOfNextMonth);

  // first day of the year
  LocalDate firstDayOfYear =
localDate.with(TemporalAdjusters.firstDayOfYear());
  System.out.println("firstDayOfYear : " + firstDayOfYear);

 }

}
```

Output:

Current Date : 2019-12-27

firstDayOfMonth : 2019-12-01

firstDayOfNextMonth : 2020-01-01

firstDayOfYear : 2019-01-01

4.6 ChronoUnits, Period and Duration Examples

ChronoUnit is an enum under package java.time.temporal and used to represent the constants in the old api.
Period class is used to get the differences between the two dates. This deals with only day, year and month.
Duration class is used to deal with seconds and nanoseconds

```
import java.time.Duration;

import java.time.LocalDate;

import java.time.LocalTime;

import java.time.Period;

import java.time.temporal.ChronoUnit;


public class ChronoUnitsPeriodDurationExample {


 public static void main(String[] args) {
```

```
LocalDate now = LocalDate.now();
System.out.println("Present time : " + now);

// ChronoUnit examples

LocalDate plus10days = now.plus(10, ChronoUnit.DAYS);
System.out.println("adding 10 days : " + plus10days);

LocalDate plus10months = now.plus(10, ChronoUnit.MONTHS);
System.out.println("adding 10 months : " + plus10months);

// Period example
LocalDate octDate = LocalDate.of(2019, 12, 1);
System.out.println("Oct date : " + octDate);

int months = Period.between(octDate, now).getMonths();
System.out.println("Months difference : " + months);

int days = Period.between(octDate, now).getDays();
System.out.println("days difference : " + days);

// Duration example

LocalTime timenow = LocalTime.of(10, 10, 0);
```

```
  LocalTime yesterdayTime = LocalTime.of(10, 10, 10);


  long seconds = Duration.between(timenow,
yesterdayTime).getSeconds();
  System.out.println("seconds : " + seconds);
 }


}
```

Output:

```
Present time : 2019-12-27
adding 10 days : 2020-01-06
adding 10 months : 2020-10-27
Oct date : 2019-12-01
Months difference : 0
days difference : 26
seconds : 10
```

5. Conclusion

In this article, We have **seen all new java 8 date time package, new classes, along with the example programs.**