Java 8 Date – LocalDate, LocalDateTime, Instant

PANKAJ — 16 COMMENTS

Java 8 Date Time API is one of the most sought after change for developers. Java has been missing a consistent approach for Date and Time from start and Java 8 Date Time API is a welcome addition to the core Java APIs.

Table of Contents [hide]

Why do we need new Java Date Time API?

Before we start looking at the Java 8 Date Time API, let's see why do we need a new API for this. There have been several problems with the existing date and time related classes in java, some of them are:

1. Java Date Time classes are not defined consistently, we have Date Class in both `java.util` as well as `java.sql` packages. Again formatting and parsing classes are defined in `java.text` package.
2. `java.util.Date` contains both date and time, whereas `java.sql.Date` contains only date. Having this in `java.sql` package doesn't make sense. Also both the classes have same name, that is a very bad design itself.
3. There are no clearly defined classes for time, timestamp, formatting and parsing. We have `java.text.DateFormat` abstract class for parsing and formatting need. Usually `SimpleDateFormat` class is used for parsing and formatting.
4. All the Date classes are mutable, so they are **not thread safe**. It's one of the biggest problem with Java Date and Calendar classes.
5. Date class doesn't provide internationalization, there is no timezone support. So `java.util.Calendar` and `java.util.TimeZone` classes were introduced, but they also have all the problems listed above.

There are some other issues with the methods defined in Date and Calendar classes but above problems make it clear that a robust Date Time API was needed

in Java. That's why Joda Time played a key role as a quality replacement for Java Date Time requirements.

Java 8 Date

Java 8 Date Time API is JSR-310 implementation. It is designed to overcome all the flaws in the legacy date time implementations. Some of the design principles of new Date Time API are:

1. **Immutability**: All the classes in the new Date Time API are immutable and good for multithreaded environments.
2. **Separation of Concerns**: The new API separates clearly between human readable date time and machine time (unix timestamp). It defines separate classes for Date, Time, DateTime, Timestamp, Timezone etc.
3. **Clarity**: The methods are clearly defined and perform the same action in all the classes. For example, to get the current instance we have `now()` method. There are format() and parse() methods defined in all these classes rather than having a separate class for them.
   All the classes use Factory Pattern and Strategy Pattern for better handling. Once you have used the methods in one of the class, working with other classes won't be hard.

4. **Utility operations**: All the new Date Time API classes comes with methods to perform common

tasks, such as plus, minus, format, parsing, getting separate part in date/time etc.
5. **Extendable**: The new Date Time API works on ISO-8601 calendar system but we can use it with other non ISO calendars as well.

Java 8 Date Time API Packages

Java 8 Date Time API consists of following packages.

1. **java.time Package**: This is the base package of new Java Date Time API. All the major base classes are part of this package, such as `LocalDate`, `LocalTime`, `LocalDateTime`, `Instant`, `Period`, `Duration` etc. All of these classes are immutable and thread safe. Most of the times, these classes will be sufficient for handling common requirements.
2. **java.time.chrono Package**: This package defines generic APIs for non ISO calendar systems. We can extend `AbstractChronology` class to create our own calendar system.
3. **java.time.format Package**: This package contains classes used for formatting and parsing date time objects. Most of the times, we would not be directly using them because principle classes in java.time package provide formatting and parsing methods.
4. **java.time.temporal Package**: This package contains temporal objects and we can use it for find out specific date or time related to date/time object. For example, we can use these to find out the first or last day of the month. You can identify these methods easily because they always have format “withXXX”.
5. **java.time.zone Package**: This package contains classes for supporting different time zones and their rules.

Java 8 Date Time API Examples

We have looked into most of the important parts of Java Date Time API. It's time now to look into most important classes of Date Time API with examples.

1. LocalDate

`LocalDate` is an immutable class that represents Date with default format of yyyy-MM-dd. We can use `now()` method to get the current date. We can also provide input arguments for year, month and date to create LocalDate instance. This class provides overloaded method for now() where we can pass ZoneId for getting date in specific time zone. This class provides the same functionality as `java.sql.Date`. Let's look at a simple example for it's usage.

```
package com.journaldev.java8.time;

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

/**
 * LocalDate Examples
 * @author pankaj
 *
 */
public class LocalDateExample {

	public static void
main(String[] args) {

		//Current Date
		LocalDate today =
LocalDate.now();

System.out.println("Current
```

LocalDate methods explanation is provided in comments, when we run this program, we get following output.

```
Current Date=2014-04-28
Specific Date=2014-01-01
Current Date in IST=2014-04-29
365th day from base date= 1971-01-01
100th day of 2014=2014-04-10
```

2. LocalTime

LocalTime is an immutable class whose instance represents a time in the human readable format. It's default format is hh:mm:ss.zzz. Just like LocalDate, this class provides time zone support and creating instance by passing hour, minute and second as input arguments. Let's look at it's usage with a simple program.

```
package com.journaldev.java8.time;

import java.time.LocalTime;
import java.time.ZoneId;

/**
 * LocalTime Examples
 * @author pankaj
 *
 */
public class LocalTimeExample {

        public static void
main(String[] args) {

                //Current Time
                LocalTime time =
LocalTime.now();
```

```
System.out.println("Current
Time="+time);
```

When we run above program for LocalTime examples, we get following output.

```
Current Time=15:51:45.240
Specific Time of Day=12:20:25.000000040
Current Time in IST=04:21:45.276
10000th second time= 02:46:40
```

3. LocalDateTime

`LocalDateTime` is an immutable date-time object that represents a date-time, with default format as yyyy-MM-dd-HH-mm-ss.zzz. It provides a factory method that takes `LocalDate` and `LocalTime` input arguments to create `LocalDateTime` instance. Let's look it's usage with a simple example.

```
package com.journaldev.java8.time;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Month;
import java.time.ZoneId;
import java.time.ZoneOffset;

public class LocalDateTimeExample {

        public static void
main(String[] args) {

                //Current Date
                LocalDateTime today
= LocalDateTime.now();
```

```
System.out.println("Current
DateTime="+today);
```

In all the three examples, we have seen that if we provide invalid arguments for creating Date/Time, then it throws `java.time.DateTimeException` that is a RuntimeException, so we don't need to explicitly catch it.

We have also seen that we can get Date/Time data by passing `ZoneId`, you can get the list of supported ZoneId values from it's javadoc. When we run above class, we get following output.

```
Current DateTime=2014-04-
28T16:00:49.455
Current DateTime=2014-04-
28T16:00:49.493
Specific Date=2014-01-01T10:10:30
Current Date in IST=2014-04-
29T04:30:49.493
10000th second time from 01/01/1970=
1970-01-01T02:46:40
```

4. Instant

Instant class is used to work with machine readable time format, it stores date time in unix timestamp. Let's see it's usage with a simple program.

```
package com.journaldev.java8.time;

import java.time.Duration;
import java.time.Instant;

public class InstantExample {
```

```
        public static void
main(String[] args) {
                //Current timestamp
                Instant timestamp =
Instant.now();

System.out.println("Current
Timestamp = "+timestamp);

                //Instant from
timestamp
                Instant specificTime
=
Instant.ofEpochMilli(timestamp.toEpoc
```

Output of above program is:

```
Current Timestamp = 2014-04-
28T23:20:08.489Z
Specific Time = 2014-04-
28T23:20:08.489Z
PT720H
```

5. Java 8 Date API Utilities

As mentioned earlier, most of the Date Time principle classes provide various utility methods such as plus/minus days, weeks, months etc. There are some other utility methods for adjusting the date using `TemporalAdjuster` and to calculate the period between two dates.

```
package com.journaldev.java8.time;

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Period;
import
java.time.temporal.TemporalAdjusters;
```

```
public class DateAPIUtilities {

        public static void
main(String[] args) {

                LocalDate today =
LocalDate.now();

                //Get the Year,
check if it's leap year

System.out.println("Year
```

Output of above program is:

```
Year 2014 is Leap Year? false
Today is before 01/01/2015? true
Current Time=2014-04-28T16:23:53.154
10 days after today will be 2014-05-08
3 weeks after today will be 2014-05-19
20 months after today will be 2015-12-
28
10 days before today will be 2014-04-18
3 weeks before today will be 2014-04-07
20 months before today will be 2012-08-
28
First date of this month= 2014-04-01
Last date of this year= 2014-12-31
Period Format= P8M3D
Months remaining in the year= 8
```

6. Java 8 Date Parsing and Formatting

It’s very common to format date into different formats and then parse a String to get the Date Time objects. Let’s see it with simple examples.

```
package com.journaldev.java8.time;

import java.time.Instant;
import java.time.LocalDate;
import java.time.LocalDateTime;
import
java.time.format.DateTimeFormatter;

public class DateParseFormatExample
{

        public static void
main(String[] args) {

                //Format examples
                LocalDate date =
LocalDate.now();
                //default format

System.out.println("Default format
of LocalDate="+date);
```

When we run above program, we get following output.

```
Default format of LocalDate=2014-04-28
28::Apr::2014
20140428
Default format of LocalDateTime=2014-
04-28T16:25:49.341
28::Apr::2014 16::25::49
20140428
Default format of Instant=2014-04-
28T23:25:49.342Z
Default format after parsing = 2014-04-
27T21:39:48
```

7. Java 8 Date API Legacy Date Time Support

Legacy Date/Time classes are used in almost all the applications, so having backward compatibility is a must. That's why there are several utility methods through which we can convert Legacy classes to new classes and vice versa. Let's see this with a simple example.

```
package com.journaldev.java8.time;

import java.time.Instant;
import java.time.LocalDateTime;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;

public class DateAPILegacySupport {

        public static void
main(String[] args) {

                //Date to Instant
                Instant timestamp =
new Date().toInstant();
                //Now we can convert
Instant to LocalDateTime or other
```

When we run above application, we get following output.

```
Date = 2014-04-28T16:28:54.340
2014-04-28T23:28:54.395Z
America/Los_Angeles
2014-04-28T16:28:54.404-
```

```
07:00[America/Los_Angeles]
Mon Apr 28 16:28:54 PDT 2014
sun.util.calendar.ZoneInfo[id="America/L

java.util.GregorianCalendar[time=1398727
```

As you can see that legacy `TimeZone` and `GregorianCalendar` classes toString() methods are too verbose and not user friendly.

That's all for Java 8 Date Time API, I like this new API a lot. Some of the most used classes will be LocalDate and LocalDateTime for this new API. It's very easy to work with and having similar methods that does a particular job makes it easy to find. It will take some time from moving legacy classes to new Date Time classes, but I believe it will be worthy of the time.

You can download all the example code from my GitHub Repository.