**Date and Time in Java**

In this article, we will go over how to work with dates and times in Java by using local date & time classes. These classes are available from Java 8.

We are going to discuss:

1. **LocalDate** represents just a date(year, month, day) — not time.
2. **LocalTime**represents only a time(hours, minutes, seconds, nanoseconds).
3. **LocalDateTime**represents date and time.

# LocalDate

We can use java.time.LocalDate.java to represent a date in our program as an object. In addition, it has many useful methods to manipulate its value.

import java.time.LocalDate;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDate date = LocalDate.now();  
 System.out.println(date); // 2021-08-16(current date)

}  
}

In the above example, we are getting our current date and just printing it out.

Also, we can construct our date object manually for any specific date.

import java.time.LocalDate;  
import java.time.Month;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDate date = LocalDate.of(2018, 5, 26);  
 System.out.println(date); // 2018-05-26  
  
 // or  
  
 date = LocalDate.of(2018, Month.MAY, 26);  
 System.out.println(date); // 2018-05-26  
 }  
}

We can get each value separately

import java.time.LocalDate;  
import java.time.Month;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDate date = LocalDate.of(2018, Month.MAY, 26);  
 System.out.println(date); // 2018-05-26  
  
 System.out.println("Year: " + date.getYear()); // Year: 2018  
 System.out.println("Month: " + date.getMonth()); // Month: MAY  
 System.out.println("Day: " + date.getDayOfMonth()); // Day: 26  
 System.out.println("Day of week: " + date.getDayOfWeek()); // Day of week: SATURDAY  
 }  
}

even day of the week.

We can add and subtract specific values from the date

import java.time.LocalDate;  
import java.time.Month;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDate date = LocalDate.of(2018, Month.MAY, 26);  
 System.out.println(date); // 2018-05-26  
  
 date = date.plusYears(2);  
 System.out.println(date); // 2020-05-26  
  
 date = date.plusMonths(1);  
 System.out.println(date); // 2020-06-26  
 System.out.println(date.getMonth()); // JUNE  
  
 date = date.plusDays(10);  
 System.out.println(date); // 2020-07-06  
  
 date = date.minusYears(2);  
 System.out.println(date); // 2018-07-06  
 }  
}

* We can add and subtract years, months, days, and weeks from our date object.
* Notice, LocalDate is an immutable object so, in order to change it, we need to reassign it and most of the methods return an instance of it.

There are more useful methods in LocalDate. They are pretty intuitive by their names.

Let’s talk about formating our local dates.

import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.Month;  
import java.time.format.DateTimeFormatter;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDate date = LocalDate.of(2018, Month.MAY, 26);  
 System.out.println(date); // 2018-05-26  
  
 DateTimeFormatter f = DateTimeFormatter.ofPattern("dd MMM, YYYY");  
 String strDate = f.format(date);  
 System.out.println(strDate); // 26 May, 2018  
  
 f = DateTimeFormatter.ofPattern("MM/dd/YYYY");  
 strDate = f.format(date);  
 System.out.println(strDate); // 05/26/2018  
 }  
}

* We can get a local date in different string formats by using java.time.format.DateTimeFormatter.java class. The pattern we provide follows [regular expression](https://en.wikipedia.org/wiki/Regular_expression).

Now let’s see how to get a date from a string.

import java.time.LocalDate;  
public class Main {  
 public static void main(String[] args) {  
 String str = "2016-03-26";  
 LocalDate date = LocalDate.parse(str);  
 // if format of string is default date format,  
 // we can convert in one step.  
 System.out.println(date); // 2016-03-26  
 System.out.println(date.getYear()); // 2016  
 System.out.println(date.getMonth()); // MARCH  
 System.out.println(date.getDayOfMonth()); // 26  
 }  
}

* if our string is in default format (YYYY-MM-dd), we can convert to date object just in one step.

import java.time.LocalDate;  
import java.time.format.DateTimeFormatter;  
  
public class Main {  
 public static void main(String[] args) {  
 String str = "August 26, 2021";  
  
 DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM dd, uuuu");  
 LocalDate date = LocalDate.parse(str, f);  
  
 System.out.println(date); // 2021-08-26  
 }  
}

* if you have your dates in a different format than a default one, you have to use DateTimeFormatter to provide your custom pattern.
* Somehow, YYYY pattern is not working so we need to use uuuu for the year.

# LocalTime

LocalTime is a class to work with time in java.

import java.time.LocalTime;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalTime time = LocalTime.now();  
 System.out.println(time); // 09:26:18.922276  
 }  
}

* It’s similar to LocalDate, but it’s used only for a time(hours, minutes, seconds, and nanoseconds).
* It is immutable as well.
* LocalTime.now() that’s how we get the current time.

import java.time.LocalTime;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalTime time = LocalTime.now();  
 System.out.println(time); // 09:26:18.922276  
  
 // build LocalTime object manually  
 time = LocalTime.of(9, 26, 18);  
  
 // also with nanoseconds  
 time = LocalTime.of(9, 26, 18, 20);  
 }  
}

There are a lot more useful methods where we can add, substruct, and compare.

If we want to have time in a different format, we can use DateTimeFormatter as well.

LocalTime time = LocalTime.of(9, 00);  
  
// Formatting time  
DateTimeFormatter f = DateTimeFormatter.ofPattern("hh:mm a");  
// 'a' will add time extension for 12 hours format  
String myTime = f.format(time);  
  
System.out.println("My time: " + myTime); // My time: 09:00 AM

# LocalDateTime

The LocalDateTime is used if we need to work with dates and times at the same time.

import java.time.LocalDateTime;  
  
public class Main {  
 public static void main(String[] args) {  
 // get current date & time  
 LocalDateTime dateTime = LocalDateTime.now();  
 System.out.println(dateTime); // 2021-08-17T09:56:26.928961  
 }  
}

* LocalDateTime can work dates and times as well.
* The methods are really similar to LocalDate and LocalTime
* After T time goes

Let’s see more examples

import java.time.LocalDate;  
import java.time.LocalDateTime;  
import java.time.LocalTime;  
  
public class Main {  
 public static void main(String[] args) {  
 // get current date & time  
 LocalDateTime dateTime = LocalDateTime.now();  
 System.out.println(dateTime); // 2021-08-17T09:56:26.928961  
  
 // build LocalDateTime object manually  
 LocalDateTime dateTime1 = LocalDateTime.of(2015, 03, 26, 6, 00);  
 System.out.println(dateTime1); // 2015-03-26T06:00  
  
 // build LocalDateTime object from date and time objects  
 LocalDate date = LocalDate.of(1991, 05, 05);  
 LocalTime time = LocalTime.of(7, 00, 00);  
 LocalDateTime dateTime2 = LocalDateTime.of(date, time);  
 System.out.println(dateTime2); // 1991-05-05T07:00  
 }  
}

Few more examples of formatting the LocalDateTime object.

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
public class Main {  
 public static void main(String[] args) {  
 LocalDateTime dateTime = LocalDateTime.of(1996, 03, 26, 19, 0);  
 System.out.println(dateTime); // 1996-03-26T19:00  
  
 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MMM dd, YYYY - hh:mm a");  
 String dateTimeStr = formatter.format(dateTime);  
 System.out.println(dateTimeStr); // Mar 26, 1996 - 07:00 PM  
 }  
}

Let’s see String to LocalDateTime

import java.time.LocalDateTime;  
import java.time.format.DateTimeFormatter;  
  
public class Main {  
 public static void main(String[] args) {  
 String str = "Welcome, today date & time: July 15, 2021 - 10:05 AM";  
 // get date & time part from String  
 String dateStr = str.split("time:")[1].trim();  
 // Print String date & time  
 System.out.println(dateStr); // July 15, 2021 - 10:05 AM  
  
 // give format of date String to DateTimeFormatter  
 DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM dd, uuuu - hh:mm a");  
  
 // convert to LocalDateTime object  
 LocalDateTime actualDate = LocalDateTime.parse(dateStr, f);  
 // print LocalDateTime object  
 System.out.println(actualDate); // 2021-07-15T10:05  
 }  
}

# Summary

To work with dates only, you can use LocalDate, and to work with time only, you can use LocalTime. LocalDateTime can be used to represent a date and time with one object. You can get the current date & time by using LocalDateTime.now() method. DateTimeFormatter can be used to convert date & time objects to String in different formats. Similarly, it can be used to convert String to date & time objects by providing regular expression patterns.

# Q&A

**Q:** How about time zones? Do local date & time classes handle it automatically?  
**A:**No, they don’t. Java suggests to think we are all in the same time zone, but if you really want to have a time zone option, you will need to use ZonedDateTime class.

**Q:** Where does Java get current date & time information?  
**A:**Simple, from your Operating System where code is running.

**Q:**What is Period in local date & time framework?  
**A:** Period represents some period of time. It can have years, months, and days.

**Q:**What is Duration in the local date & time framework?  
**A:** Duration is also some period of time, but can have smaller units. For example the minimum period for Period is 1 day, in contrast Duration can have seconds, milliseconds, and even nanoseconds.

**Q:**What’s Instant in the local date & time framework?  
**A:**It represents a specific moment in time. For example, by using Instant object you could calculate how much it took to run your program:

// get specific moment time in the beginning   
Instant start = Instant.now();

for (int i = 0; i < 1000; i++) {   
 System.out.println("Hello, World!");   
}

// get specific moment time in the end  
Instant finish = Instant.now();

Page **10** of **10Page 10 of 10**// find out the difference   
long timeElapsed = Duration.between(start, finish).toMillis(); System.out.println("Milliseconds: " + timeElapsed);

**Q:** What is epoch time?  
**A:** It’s the number of seconds elapsed from 00:00:00 UTC January 1, 1970. It’s a Unix system to describe points in time. LocalDateTime has build-in toEpochSeconds() method to get epoch time.