# Parsing and Formatting

The temporal-based classes in the Date-Time API provide parse methods for parsing a string that contains date and time information. These classes also provide format methods for formatting temporal-based objects for display. In both cases, the process is similar: you provide a pattern to the DateTimeFormatter to create a formatter object. This formatter is then passed to the parse or format method.

The DateTimeFormatter class provides numerous [predefined formatters](https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html" \l "predefined" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank), or you can define your own.

The parse and the format methods throw an exception if a problem occurs during the conversion process. Therefore, your parse code should catch the DateTimeParseException error and your format code should catch the DateTimeException error. For more information on exception handing, see [Catching and Handling Exceptions](https://docs.oracle.com/javase/tutorial/essential/exceptions/handling.html" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_top).

The DateTimeFormatter class is both immutable and thread-safe; it can (and should) be assigned to a static constant where appropriate.

**Version Note:** The java.time date-time objects can be used directly with java.util.Formatter and String.format by using the familiar pattern-based formatting that was used with the legacy java.util.Date and java.util.Calendar classes.

## Parsing

The one-argument [parse(CharSequence)](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html" \l "parse-java.lang.CharSequence-" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) method in the LocalDate class uses the ISO\_LOCAL\_DATE formatter. To specify a different formatter, you can use the two-argument [parse(CharSequence, DateTimeFormatter)](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html" \l "parse-java.lang.CharSequence-java.time.format.DateTimeFormatter-" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) method. The following example uses the predefined BASIC\_ISO\_DATE formatter, which uses the format 19590709 for July 9, 1959.

String in = ...;

LocalDate date = LocalDate.parse(in, DateTimeFormatter.BASIC\_ISO\_DATE);

You can also define a formatter using your own pattern. The following code, from the [Parse](https://docs.oracle.com/javase/tutorial/datetime/iso/examples/Parse.java" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) example, creates a formatter that applies a format of "MMM d yyyy". This format specifies three characters to represent the month, one digit to represent day of the month, and four digits to represent the year. A formatter created using this pattern would recognize strings such as "Jan 3 2003" or "Mar 23 1994". However, to specify the format as "MMM dd yyyy", with two characters for day of the month, then you would have to always use two characters, padding with a zero for a one-digit date: "Jun 03 2003".

String input = ...;

try {

DateTimeFormatter formatter =

DateTimeFormatter.ofPattern("MMM d yyyy");

LocalDate date = LocalDate.parse(input, formatter);

System.out.printf("%s%n", date);

}

catch (DateTimeParseException exc) {

System.out.printf("%s is not parsable!%n", input);

throw exc; // Rethrow the exception.

}

// 'date' has been successfully parsed

The documentation for the DateTimeFormatter class specifies the [full list of symbols](https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html" \l "patterns" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) that you can use to specify a pattern for formatting or parsing.

The StringConverter example on the [Non-ISO Date Conversion](https://docs.oracle.com/javase/tutorial/datetime/iso/nonIso.html" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_top) page provides another example of a date formatter.

## Formatting

The [format(DateTimeFormatter)](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html" \l "format-java.time.format.DateTimeFormatter-" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) method converts a temporal-based object to a string representation using the specified format. The following code, from the [Flight](https://docs.oracle.com/javase/tutorial/datetime/iso/examples/Flight.java" \t "https://docs.oracle.com/javase/tutorial/datetime/iso/_blank) example, converts an instance of ZonedDateTime using the format "MMM d yyy hh:mm a". The date is defined in the same manner as was used for the previous parsing example, but this pattern also includes the hour, minutes, and a.m. and p.m. components.

ZoneId leavingZone = ...;

ZonedDateTime departure = ...;

try {

DateTimeFormatter format = DateTimeFormatter.ofPattern("MMM d yyyy hh:mm a");

String out = departure.format(format);

System.out.printf("LEAVING: %s (%s)%n", out, leavingZone);

}

catch (DateTimeException exc) {

System.out.printf("%s can't be formatted!%n", departure);

throw exc;

}

The output for this example, which prints both the arrival and departure time, is as follows:

LEAVING: Jul 20 2013 07:30 PM (America/Los\_Angeles)

ARRIVING: Jul 21 2013 10:20 PM (Asia/Tokyo)