Date

The Date-Time API provides four classes that deal exclusively with date information, without respect to time or time zone. The use of these classes are suggested by the class names: `LocalDate`, `YearMonth`, `MonthDay`, and `Year`.

The LocalDate Class

A `LocalDate` represents a year-month-day in the ISO calendar and is useful for representing a date without a time. You might use a `LocalDate` to track a significant event, such as a birth date or wedding date. The following examples use the of and with methods to create instances of `LocalDate`:

```
LocalDate date = LocalDate.of(2000, Month.NOVEMBER, 20);
LocalDate nextWed = date.with(TemporalAdjusters.next(DayOfWeek.WEDNESDAY));
```

For more information about the `TemporalAdjuster` interface, see the section on Temporal Adjuster.

In addition to the usual methods, the `LocalDate` class offers getter methods for obtaining information about a given date. The `getDayOfWeek()` method returns the day of the week that a particular date falls on. For example, the following line of code returns "MONDAY":

```
DayOfWeek dotw = LocalDate.of(2012, Month.JULY, 9).getDayOfWeek();
```

The following example uses a `TemporalAdjuster` to retrieve the first Wednesday after a specific date.

```
LocalDate date = LocalDate.of(2000, Month.NOVEMBER, 20);
TemporalAdjuster adj = TemporalAdjusters.next(DayOfWeek.WEDNESDAY);
LocalDate nextWed = date.with(adj);
System.out.printf("For the date of %s, the next Wednesday is %s.%n",
                  date, nextWed);
```

Running the code produces the following:

```
For the date of 2000-11-20, the next Wednesday is 2000-11-22.
```

The Period and Duration section also has examples using the `LocalDate` class.

The YearMonth Class

The `YearMonth` class represents the month of a specific year. The following example uses the `lengthOfMonth()` method to determine the number of days for several year and month combinations.

```
YearMonth date = YearMonth.now();
System.out.printf("%s: %d%n", date, date.lengthOfMonth());

YearMonth date2 = YearMonth.of(2010, Month.FEBRUARY);
System.out.printf("%s: %d%n", date2, date2.lengthOfMonth());

YearMonth date3 = YearMonth.of(2012, Month.FEBRUARY);
System.out.printf("%s: %d%n", date3, date3.lengthOfMonth());
```

The output from this code looks like the following:

```
2013-06: 30
2010-02: 28
2012-02: 29
```

The MonthDay Class

The `MonthDay` class represents the day of a particular month, such as New Year's Day on January 1.

The following example uses the `isValidYear()` method to determine if February 29 is valid for the year 2010. The call returns false, confirming that 2010 is not a leap year.

```
MonthDay date = MonthDay.of(Month.FEBRUARY, 29);
boolean validLeapYear = date.isValidYear(2010);
```

The Year Class

The `Year` class represents a year. The following example uses the `isLeap()` method to determine if the given year is a leap year. The call returns `true`, confirming that 2012 is a leap year.

```
boolean validLeapYear = Year.of(2012).isLeap();
```

Last update: *January 27, 2022*

DayOfWeek and Month Enums ← **Date** → Date and Time

Home > Tutorials > The Date Time API > Date