11 December 2020 118343 0

11 parse() methods in Java with Examples

Published in the Strings in Java group

Parsing in its most general sense is the extraction of the necessary information from some piece of data, most often textual data. What is parse in Java? There are many Java classes that have the `parse()` method. Usually the `parse()` method receives some string as input, "extracts" the necessary information from it and converts it into an object of the calling class. For example, it received a string and returned the date that was "hiding" in this string.

In this post, we are going to take a look at the 10 useful variations of `parse()`.

0. parseInt()

Let's start from one of the most popular `parse()` methods, that is not exactly `parse()`, but `parseInt()`.

Java `parseInt ()` method is used to get the primitive data type from a specific string. In other words it converts a string to a number. `parseInt ()` can have one or two arguments.

Here's the syntax of `parseInt()`:

```
1  static int parseInt(String s)
2  static int parseInt(String s, int radix)
```

Where `s` is the string representing a signed decimal value and radix the base of a numerical system. Keep in mind that there's no default base value — you need to enter one within the range of 2 and 36.

Here's an example. How to parse with ParseInt():

```
1  public class ParseInt0 {
2
3        public static void main(String args[]){
4            int x = Integer.parseInt("12");
5            double c = Double.parseDouble("12");
6            int b = Integer.parseInt("100",2);
7
8            System.out.println(Integer.parseInt("12"));
9            System.out.println(Double.parseDouble("12"));
10           System.out.println(Integer.parseInt("100",2));
11           System.out.println(Integer.parseInt("101", 8));
12
13       }
14   }
```

The output is:

```
12
12.0
```

```
4
65
```

1. Period parse() method

`Period` is a Java class to model a quantity of time in terms of years, months and days such as “3 years, 5 months and 2 days”. It has a `parse()` method to obtain a period from a text.

Here’s the syntax of `period parse()`

```
1  public static Period parse(CharSequence text)
```

CharSequence is an Interface, implemented by Strings. So you can use Strings as a text element in `parse()` method. Sure the string should be in proper format to return an object of Period class.

This format is **PnYnMnD**.

Where Y stands for “year”, M — for “month”, D — for “day”. N is a number corresponding to each period value.

- The method has one parameter — a text value.
- `Parse()` returns a Period value where the value of a string becomes a parameter.
- As an exception, `period parse()` can return the DateTimeParseException if the string value doesn’t meet the structure of a period.

Here’s an example of using `Period parse()` in the real-world context:

```java
import java.time.Period;
public class ParseDemo1 {

   public static void main(String[] args)
   {
       //Here is the age String in format to  parse
       String age = "P17Y9M5D";

       // Converting strings into period value
       // using parse() method
       Period p = Period.parse(age);
       System.out.println("the age is: ");
       System.out.println(p.getYears() + " Years\n"
                          + p.getMonths() + " Months\n"
                          + p.getDays() + " Days\n");
   }
}
    }
}
```

The output is:

```
the age is:
17 Years
9 Months
5 Days
```

2.SimpleDateFormat Parse() method

`SimpleDateFormat` is a class that is used for formatting and parsing dates in a locale-sensitive manner. `SimpleDateFormat parse()` method breaks a string down into date tokens and returns a Data value in the corresponding format. The method starts parsing the string at an index, defined by a developer.

This is the syntax of the `SimpleDateFormat parse()`:

```
1  public Date parse(String the_text, ParsePosition position)
```

The method has two parameters:

- Position: the data at the starting index which is always the ParsePosition object type.
- the_text: defines the string that the method will parse and is a String type value.

You can use this method without position declaration. In this case the data starts from zero index.

`SimpleDateFormat parse()` returns a date or a null value (in case the string wasn't processed due to an error).

Here's an example of `SimpleDateFormat parse()` implementation:

```java
// Parsing strings into the Date format with two different patterns import java.text.ParseExce
import java.text.ParsePosition;
import java.text.SimpleDateFormat;
import java.util.Date;

public class ParseDemo2 {
   public static void main(String[] args) throws ParseException {
       SimpleDateFormat simpleDateFormat1 = new SimpleDateFormat("MM/dd/yyyy");
       SimpleDateFormat simpleDateFormat2 = new SimpleDateFormat("dd/MM/yyyy");
       //simpleDateFormat1.setLenient(false);
       Date date1 = simpleDateFormat1.parse("010/14/2020");
       System.out.println(date1);
       Date date2 = simpleDateFormat2.parse("14/10/2020");
       System.out.println(date2);
       ParsePosition p1 = new ParsePosition(18);
       ParsePosition p2 = new ParsePosition(19);
```

```
        ParsePosition p3 = new ParsePosition(5);

        String myString = "here is the date: 14/010/2020";
        Date date3 = simpleDateFormat2.parse(myString,p1);
        Date date4 = simpleDateFormat2.parse(myString,p2);
        Date date5 = simpleDateFormat2.parse(myString,p3);

        System.out.println(date3);
        System.out.println(date4);
        System.out.println(date5);
    }
}
```

The output is:

```
Wed Oct 14 00:00:00 EEST 2020
Wed Oct 14 00:00:00 EEST 2020
Wed Oct 14 00:00:00 EEST 2020
Sun Oct 04 00:00:00 EEST 2020
null
```

The last one is null because there is no date pattern starting from the 5th position. By the way if you try to parse the date5 without position such as `Date date5 = simpleDateFormat2.parse(myString)`, you'll get an exception:

```
Exception in thread "main" java.text.ParseException: Unparseable date: "here is the date:
14/010/2020"
```

```
        at java.base/java.text.DateFormat.parse(DateFormat.java:396)
        at ParseDemo2.main(ParseDemo2.java:22)
```

3. LocalDate parse() method

`LocalDate` is a class that appeared in Java 8 to represent a date such as year-month-day (day-of-year, day-of-week and week-of-year, can also be accessed).

LocalDate doesn't represent a time or time-zone.

`LocalDate parse()` method has two variants. Both of them help to convert a string into a new Java 8 date API — `java.time.LocalDate`.

parse(CharSequence text, DateTimeFormatter, formatter)

This method parses a string using a specific formatter to obtain an instance of LocalDate.

Here's the syntax of the method:

```
1  public static LocalTime parse(CharSequence text,
2                                DateTimeFormatter formatter)
```

There are two parameters for the method — the text that will be parsed and the formatter a developer will apply. As a return value, the method returns a LocalTime object that will be recognized as the local day-time.

If the text value couldn't go through parsing, the system throws the DayTimeParseException.

Let's have a code example of using `LocalDate parse()` with two parameters:

```
1  import java.time.LocalDate;
2  import java.time.format.DateTimeFormatter;
3  public class ParserDemo3 {
4
5     public static void main(String[]args) {
6
7         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
8         String date = "14/10/2020";
9         LocalDate localDate = LocalDate.parse(date, formatter);
10        System.out.println("parsed local date: " + localDate);
11        System.out.println("formatted local date: " + formatter.format(localDate));
12    }
13 }
```

The output is:

```
parsed local date: 2020-10-14
formatted local date: 14/10/2020
```

`LocalDate parse()` method with one parameter has the next syntax:

```
1  public static LocalTime parse(CharSequence text)
```

This method doesn't require specifying a formatter. After a developer enters the string values into the brackets, the system will automatically use the DateTimeFormatter.ISO_LOCAL_DATE.

This method has a single parameter — a CharSequence text. You may use here the string value. Make sure it's not null and respects the structure of the formatter. If there's no way to parse a string, a developer gets the DateTimeExceptionAlert.

Here's an example of `LocalDate parse()` application:

```
1  import java.time.*;
2  public class ParseDemo3 {
3        public static void main(String[] args)
4        {
5            // let's make a new LocalDate object
6            LocalDate localDate = LocalDate.parse("2020-10-14");
7            System.out.println("LocalDate : " + localDate);
8        }
9    }
```

The output is:

```
LocalDate : 2020-10-14
```

4. LocalDateTime parse() method

`LocalDateTime` a date-time object that represents a date-time viewed pretty often as year-month-day-hour-minute-second. Also developers can use other date and time fields (day-of-year, day-of-week and week-of-year). This class is immutable. Time is represented to nanosecond precision. For example, you can store the value "17nd November 2020 at 13:30.30.123456789" in a LocalDateTime.

This class is not about representing a time-zone. It is rather a standard date representation plus local time.

`LocalDateTime parse()` method represented in two variants:

- `static LocalDateTime parse(CharSequence text)` returns an instance of LocalDateTime from a text string such as 2007-12-03T10:15:30.
- `static LocalDateTime parse(CharSequence text, DateTimeFormatter formatter)` returns an instance of LocalDateTime from a text string using a specific formatter.

Here is an example of `LocalDateTime parse()` method:

```
1  import java.time.*;
2  public class ParseDemo11 {
3        public static void main(String[] args) {
4            LocalDateTime localDateTime = LocalDateTime.parse("2020-11-17T19:34:50.63");
5            System.out.println("LocalDateTime is: " + localDateTime);
6        }
7    }
```

The output is:

```
LocalDateTime is: 2020-11-17T19:34:50.630
```

5. ZonedDateTime parse() method

Class `ZonedDateTime` represents a date-time with a time-zone. This class is immutable. It stores date and time fields to a precision of nanoseconds, and a time zone, with a zone offset used to handle ambiguous local date-times. So if you need to keep a value such as "14nd October 2020 at 17:50.30.123456789 +02:00 in the Europe/Paris time-zone" you can use ZonedDateTime. The class is often used to manipulate local time-based data.

`ZondeDateTime parse()` is a parser that breaks the string down into tokens in the ISO-8061 system. Here's an example of a value you'll get after parsing:

```
2020-04-05T13:30:25+01:00 Europe/Rome
```

It is used whenever high-precision data is needed (after all, the data you get is precise up to nanoseconds). The class is often used to manipulate local time-based data.

Let's take a look at the general syntax of the `ZonedDateTime parse()` method developers use to convert string values into ZonedDateTime class.

```
1  public static ZonedDateTime parse(CharSequence text)
```

The only parameter the method uses is a string text. As a return value, you'll get one or a series of objects in the ZonedDateTime format. If there's an error during parsing or it's impossible, to begin with, the method returns DateTimeParseException.

Also there is a `parse()` method with two variables.

```
public static ZonedDateTime parse(CharSequence text, DateFormatter formatter)
```

This method obtains an instance of ZonedDateTime from a text value using a specific formatter. The method with one parameter, the formatter `DateTimeFormatter.ISO_LOCAL_TIME` is used by default.

Let's take a look at the use case for ZonedDateTime parse():

```
// An example program that uses
// ZonedDateTime.parse() method

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class ParseDemo4 {
   public static void main(String[] args) {
       ZonedDateTime zonedDateTime = ZonedDateTime.parse("2020-10-15T10:15:30+01:00");
       System.out.println(zonedDateTime);

       DateTimeFormatter dateTimeFormatter = DateTimeFormatter.ISO_ZONED_DATE_TIME;
       String date = "2020-10-15T13:30:25+01:00";
       ZonedDateTime zoneDateTime1 = ZonedDateTime.parse(date, dateTimeFormatter);
       System.out.println(zoneDateTime1);
   }
}
```

The output is:

```
2020-10-15T10:15:30+01:00
2020-10-15T13:30:25+01:00
```


6. LocalTime parse() method

Class `LocalTime` represents a time, often viewed as hour-minute-second. This class is also immutable such as ZonedDateTime.Time is represented to nanosecond precision. For example, the value "13:45.30.123456789" can be stored in a LocalTime.

There are two `LocalTime parse()` methods, with one and two parameters. Let's take a look at both:

```
1  public static LocalTime parse(CharSequence text)
```

You can use `LocalTime parse()` with only one parameter, the string you want to parse. In this case, the formatter

DateTimeFormatter.ISO_LOCAL_TIME is used by default.

```
Method with two parameters has the next syntax:
public static LocalTime parse(CharSequence text,
                              DateTimeFormatter formatter)
```

It Obtains an instance of LocalTime from a text value using a specific formatter.

The both of the methods return LocalTime value in the hh/mm/ss format. Watch out for DateTimeParceException alerts. They mean that the format of the string text doesn't correspond to that of LocalTime objects.

Here's an example of using `LocalTime parse()` in production:

```
import java.time.*;
import java.time.format.*;
public class ParseDemo5 {

       public static void main(String[] args)
       {

           LocalTime localTime
                   = LocalTime.parse("10:25:30");

           // return the output value
           System.out.println("LocalTime : "
                              + localTime);

```

```
15            // create a formater
16            DateTimeFormatter formatter
17                    = DateTimeFormatter.ISO_LOCAL_TIME;
18
19            LocalTime localTime1
20                    = LocalTime.parse("12:30:50");
21            // parse a string to get a LocalTime object in return
22
23            LocalTime.parse("12:30:50",
24                formatter);
25            // print the output
26            System.out.println("LocalTime : "
27                               + localTime1);
28        }
29    }
```

7. MessageFormat Parse() method

`MessageFormat` extends the Format class. Format is an abstract base class for formatting locale-sensitive data (dates, messages, and numbers).

MessageFormat gets some objects and formats them. Then it inserts the formatted strings into the pattern at the appropriate places.

The `MessageFormat parse()` is used to get a string value if given the index beginning. Here's the general syntax of the method:

```
public Object[] parse(String source, ParsePosition position)
```

Where source is a string to parse and position is the starting index of parsing.

Here is an example of `MessageFormat parse()` method working:

```
import java.text.MessageFormat;
import java.text.ParsePosition;

public class ParseDemo7 {
   public static void main(String[] args) {
    try {
           MessageFormat messageFormat = new MessageFormat("{1, number, #}, {0, number, #.#},

           ParsePosition pos = new ParsePosition(3);
           Object[] hash = messageFormat.parse("12.101, 21.382, 35.121", pos);

           System.out.println("value after parsing: ");
           for (int i = 0; i < hash.length; i++)
               System.out.println(hash[i]);
       }
       catch (NullPointerException e) {
          System.out.println("\nNull");
          System.out.println("Exception thrown : " + e);
       } }
}
```

8. Level parse() method

When a programmer uses a Logger to log a message it is logged with a certain log level. There are seven built-in log levels:

- SEVERE
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST

Also there are additional levels OFF that can be used to turn off logging and ALL that can be used to enable logging of all messages.

The log level is represented by the class `java.util.logging.Level`. Level class contains a constant for every of these seven levels. So you use one of these constants, including All and OFF while logging a message to a Logger. Also all these levels initialized to some integers. For example FINE is initialized to 500.

The `Level parse()` method parses an information needed from a text value and returns a Level object.

Here's the syntax of `level parse()` method:

```
1  public static Level parse(String name)
```

The parameter of a method is the name of a string a developer wants to parse. It could be a name of the level, its initialising name or some other integer. In return, a programmer gets a Level name value, corresponding to that of the initial string.

In case the argument contains symbols that are impossible to parse, the system will throw the IllegalArgumentException. If a string contains no values, a developer gets a NullPointerException.

Here's a piece of code that shows the implementation of `Level parse()`.

```
1  import java.util.logging.Level;
2  public class ParseDemo6 {
3
4     public static void main(String[] args)
5     {
```

```
        Level level = Level.parse("500");
        System.out.println("Level = " + level.toString());

        Level level1 = Level.parse("FINE");
        System.out.println("Level = " + level1.toString());

        Level level2 = level.parse ("OFF");
        System.out.println(level2.toString());
    }
}
```

The output is:

```
Level = FINE
Level = FINE
OFF
```

9. Instant parse() method

`Instant` class models a single instantaneous point on the time-line. You can use it for recording event timestamps in your app.

`Instant parse()` obtains an Instant value from a text value. The string will later be stored as a UTC Time Zone Value. The system uses DateTimeFormatter.ISO_INSTANT such as 2020-10-14T11:28:15.00Z.

Here is a syntax of `Instant parse()` method:

```
public static Instant parse(CharSequence text)
```

To parse a string and get an instant, a developer needs to make sure the string contains some text. In case it's null, you'll get a DateTimeException.

Here's an example of using `Instant parse` in Java:

```
import java.time.Instant;

public class ParseDemo8 {
	public static void main(String[] args) {

		Instant instant = Instant.parse("2020-10-14T10:37:30.00Z");
		System.out.println(instant);
	}
  }
```

The output is:

```
2020-10-14T10:37:30Z
```

10. NumberFormat parse() method

The `java.text.NumberFormat` class is used to format numbers. `NumberFormat parse()` is a default method of this class.

The `parse ()` method of the NumberFormat class converts a string to a number.

Developers use it to break a string down to its component numbers. The parsing starts from the beginning of the string. If you call `setParseIntegerOnly (true)` before calling the `parse ()` method, as shown in the following example, then only the integer part of the number is converted.

Here's the syntax of `NumberFormat parse()`:

```
1  public Number parse(String str)
```

As a parameter, the function accepts strings. A return value of the parsing is a numeric value. If a string beginning cannot be parsed, you'll get a ParseException warning.

To see the application of the `NumberFormat parse()` method, take a look at the example below:

```
1  import java.text.NumberFormat;
2  import java.text.ParseException;
3
4  public class ParseDemo9 {
5
6        public static void main(String[] args) throws ParseException {
```

```
            NumberFormat numberFormat = NumberFormat.getInstance();
            System.out.println(numberFormat.parse("3,141592"));
            numberFormat.setParseIntegerOnly(true);
            System.out.println(numberFormat.parse("3,141592"));
        }
    }
```

The output is:

```
3.141592
3
```

Conclusion

There are a lot of parsing methods developers can use to convert strings into various data types. Although remembering them might seem tedious, such a variety of commands gives developers a lot of flexibility and precision. Be sure to practice using data parsing to make sure you remember the syntax and will not forget which parameters are essential for each method.

Alex Vypirailenko

Java Developer at Toshiba Global Commerce Solutions

Before IT, Alexandr managed to work in various fields and companies: at Guinness World Records, London Olympics 2021, and Nielsen, ... [Read full bio]