The java.text.SimpleDateFormat class is used to both parse and format dates according to a formatting pattern you specify yourself. When parsing dates, the Java SimpleDateFormat typically parses the date from a [**Java String**](https://jenkov.com/java/strings.html). When formatting dates, the SimpleDateFormat typically formats a Date object into a String, although it can also format the date into a StringBuffer.

This text explains how to use the SimpleDateFormat class to format dates.

**Creating a SimpleDateFormat**

You create a SimpleDateFormat instance like this:

String pattern = "yyyy-MM-dd";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

The pattern parameter passed to the SimpleDateFormat constructor is the pattern to use for parsing and formatting of dates. The pattern syntax is covered later in this text. The pattern is just a regular [**Java String**](https://jenkov.com/java/strings.html).

**Formatting Dates**

Once you have created a SimpleDateFormat instance you can format dates using its format() method. Here is an example:

String pattern = "yyyy-MM-dd";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

String date = simpleDateFormat.format(new Date());

System.out.println(date);

The Date instance passed to the format() method is a **[java.util.Date](https://jenkov.com/java-date-time/java-util-date.html)** instance.

The output printed from the above SimpleDateFormat example would be:

2018-09-09

Notice how the formatted date string starts with the year, then month, then day. The sequence of the date fields are determined by the date pattern passed to the SimpleDateFormat constructor. As mentioned earlier, this format will be explained a bit later in this Java SimpleDateFormat tutorial.

**Format Date Into StringBuffer**

The Java SimpleDateFormat class is also capable of formatting a Date object into a StringBuffer, instead of returning an individual String with the date formatted. The SimpleDateFormat class does this via a version of the format() method that takes the Date, StringBuffer and a FieldPosition instance as parameters.

Here is an example of formatting a date into a StringBuffer using Java SimpleDateFormat :

StringBuffer stringBuffer = new StringBuffer();

Date now = new Date();

SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ssZ");

simpleDateFormat.format(now, stringBuffer, new FieldPosition(0));

It is not exactly clear how the FieldPosition instance is used. It seems the format() method appends the formatted String to the end of the StringBuffer no matter what the int value passed to the FieldPosition constructor is.

**Parsing Dates**

You can parse a String into a java.util.Date instance using the parse() method of the SimpleDateFormat instance. Here is an example:

String pattern = "yyyy-MM-dd";

SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern);

Date date = simpleDateFormat.parse("2018-09-09");

Once this code is executed, the date variable points to a Date instance representing september 9th, 2018.

**Pattern Syntax**

You can use the following symbols in your formatting pattern:

|  |  |
| --- | --- |
| y | Year (e.g. 12 or 2012). Use either yy or yyyy. |
| M | Month in year. Number of M's determine length of format (e.g. MM, MMM or MMMMM) |
| d | Day in month. Number of d's determine length of format (e.g. d or dd) |
| h | Hour of day, 1-12 (AM / PM) (normally hh) |
| H | Hour of day, 0-23 (normally HH) |
| m | Minute in hour, 0-59 (normally mm) |
| s | Second in minute, 0-59 (normally ss) |
| S | Millisecond in second, 0-999 (normally SSS) |
| E | Day in week (e.g Monday, Tuesday etc.) |
| D | Day in year (1-366) |
| F | Day of week in month (e.g. 1st Thursday of December) |
| w | Week in year (1-53) |
| W | Week in month (0-5) |
| a | AM / PM marker |
| k | Hour in day (1-24, unlike HH's 0-23) |
| K | Hour in day, AM / PM (0-11) |
| z | Time Zone |
| ' | Escape for text delimiter |
| ' | Single quote |

Characters other than these will be treated as normal text to insert into the pattern, and thus into the formatted dates.

Some characters can be used in different numbers. For instance, you can write either yy for a 2-character version of the year (e.g. 12), or you can write yyyy for a 4-character version of the year (e.g. 2012). For more information about the patterns accepted, see the JavaDoc for the SimpleDateFormat class.

**Pattern Examples**

Here are a few Java SimpleDateFormat date pattern examples:

|  |  |
| --- | --- |
| **Pattern** | **Example** |
| dd-MM-yy | 31-01-12 |
| dd-MM-yyyy | 31-01-2012 |
| MM-dd-yyyy | 01-31-2012 |
| yyyy-MM-dd | 2012-01-31 |
| yyyy-MM-dd HH:mm:ss | 2012-01-31 23:59:59 |

**Java Program illustrating SimpleDateFormat class**

// Java Program illustrating SimpleDateFormat class

import java.text.\*;

import java.util.\*;

public class NewClass

{

public static void main(String[] args)

{

SimpleDateFormat obj = new SimpleDateFormat("dd / MM / yy");

// Creating instance of the System date

Calendar c = Calendar.getInstance();

System.out.println("Present Date : " + c.getTime());

// Formatting Date according "dd / MM / yy"

String formattedDate = obj.format(c.getTime());

System.out.println("Date formatted : "+formattedDate);

}

}

// Java Program illustrating use of toPattern() method

import java.text.\*;

import java.util.Calendar;

public class NewClass

{

public static void main(String[] args) throws InterruptedException

{

SimpleDateFormat obj = new SimpleDateFormat();

// Initializing Calendar object

Calendar c = Calendar.getInstance();

// getting Current Date

String dateToday = obj.format(c.getTime());

System.out.println("Current Date : "+dateToday);

// Use of toPattern() method

// Printing Date Pattern

System.out.println("Date Pattern : "+obj.toPattern());

}

}

// Java Program illustrating

// use of parse() method

import java.text.\*;

import java.util.Calendar;

public class NewClass

{

public static void main(String[] args) throws InterruptedException

{

SimpleDateFormat obj = new SimpleDateFormat("MM / dd / yy");

try

{

Calendar c = Calendar.getInstance();

// Use of .parse() method to parse Date From String 's'

String s = "10 / 27 / 16" ;

c.setTime(obj.parse(s));

System.out.println("Time parsed from String : "+c.getTime());

}

catch (ParseException except)

{

except.printStackTrace();

}

}

}

// Java Program illustrating

// use of applyPattern() method

import java.text.\*;

import java.util.Calendar;

public class NewClass

{

public static void main(String[] args) throws InterruptedException

{

SimpleDateFormat obj = new SimpleDateFormat();

// Initializing calendar Object

Calendar c = Calendar.getInstance();

// Using 'arg' pattern

String arg = "dd / MM / yyyy HH:mm Z";

// Use of applyPattern() method to set date to 'arg' format

obj.applyPattern(arg);

// current date and time

String currentdate = obj.format(c.getTime());

System.out.println("Current Date : "+currentdate);

// Print the pattern being used

System.out.println("Pattern applied : "+obj.toPattern());

}

}

// Java Program illustrating

// use of format() method

import java.text.\*;

import java.util.Calendar;

public class NewClass

{

public static void main(String[] args) throws InterruptedException

{

SimpleDateFormat obj = new SimpleDateFormat();

// Initializing calendar Object

Calendar c = Calendar.getInstance();

System.out.println("Actual Date : "+c.getTime());

// Use of format() method to format Date to String

String currentdate = obj.format(c.getTime());

System.out.println("Formatted Date to String : "+currentdate);

}

}

**Calendar Class in Java with examples**

Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc. It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.

As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method Calendar.getInstance() to instantiate and implement a sub-class.

* Calendar.getInstance(): return a Calendar instance based on the current time in the default time zone with the default locale.
* Calendar.getInstance(TimeZone zone)
* Calendar.getInstance(Locale aLocale)
* Calendar.getInstance(TimeZone zone, Locale aLocale)

**Java program to demonstrate getInstance() method**:

|  |
| --- |
| // Date getTime(): It is used to return a  // Date object representing this  // Calendar's time value.    **import** java.util.\*;  **public** **class** Calendar1 {  **public** **static** **void** main(String args[])      {          Calendar c = Calendar.getInstance();          System.out.println("The Current Date is:" + c.getTime());      }  } |

**Output:**

The Current Date is:Tue Aug 28 11:10:40 UTC 2018

**Important Methods and their usage**

| **METHOD** | **DESCRIPTION** |
| --- | --- |
| abstract void add(int field, int amount) | It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar’s rules. |
| int get(int field) | It is used to return the value of the given calendar field. |
| abstract int getMaximum(int field) | It is used to return the maximum value for the given calendar field of this Calendar instance. |
| abstract int getMinimum(int field) | It is used to return the minimum value for the given calendar field of this Calendar instance. |
| Date getTime() | It is used to return a Date object representing this Calendar’s time value.</td |

Below programs illustrate the above methods:

**Program 1**: Java program to demonstrate get() method.

|  |
| --- |
| // Program to demonstrate get() method  // of Calendar class    **import** java.util.\*;  **public** **class** Calendar2 {  **public** **static** **void** main(String[] args)      {          // creating Calendar object          Calendar calendar = Calendar.getInstance();            // Demonstrate Calendar's get()method          System.out.println("Current Calendar's Year: " + calendar.get(Calendar.YEAR));          System.out.println("Current Calendar's Day: " + calendar.get(Calendar.DATE));          System.out.println("Current MINUTE: " + calendar.get(Calendar.MINUTE));          System.out.println("Current SECOND: " + calendar.get(Calendar.SECOND));      }  } |

**Output:**

Current Calendar's Year: 2018

Current Calendar's Day: 28

Current MINUTE: 10

Current SECOND: 45

**Program 2**: Java program to demonstrate getMaximum() method.

|  |
| --- |
| // Program to demonstrate getMaximum() method  // of Calendar class    **import** java.util.\*;  **public** **class** Calendar3 {  **public** **static** **void** main(String[] args)      {          // creating calendar object          Calendar calendar = Calendar.getInstance();    **int** max = calendar.getMaximum(Calendar.DAY\_OF\_WEEK);          System.out.println("Maximum number of days in a week: " + max);            max = calendar.getMaximum(Calendar.WEEK\_OF\_YEAR);          System.out.println("Maximum number of weeks in a year: " + max);      }  } |

**Output:**

Maximum number of days in a week: 7

Maximum number of weeks in a year: 53

**Program 3**: Java program to demonstrate the getMinimum() method.

|  |
| --- |
| // Program to demonstrate getMinimum() method  // of Calendar class    **import** java.util.\*;  **public** **class** Calendar4 {  **public** **static** **void** main(String[] args)      {          // creating calendar object          Calendar calendar = Calendar.getInstance();    **int** min = calendar.getMinimum(Calendar.DAY\_OF\_WEEK);          System.out.println("Minimum number of days in week: " + min);            min = calendar.getMinimum(Calendar.WEEK\_OF\_YEAR);          System.out.println("Minimum number of weeks in year: " + min);      }  } |

**Output:**

Minimum number of days in week: 1

Minimum number of weeks in year: 1

**Program 4**: Java program to demonstrate add() method.

|  |
| --- |
| // Program to demonstrate add() method  // of Calendar class    **import** java.util.\*;  **public** **class** Calendar5 {  **public** **static** **void** main(String[] args)      {          // creating calendar object          Calendar calendar = Calendar.getInstance();          calendar.add(Calendar.DATE, -15);          System.out.println("15 days ago: " + calendar.getTime());          calendar.add(Calendar.MONTH, 4);          System.out.println("4 months later: " + calendar.getTime());          calendar.add(Calendar.YEAR, 2);          System.out.println("2 years later: " + calendar.getTime());      }  } |

**Output:**

15 days ago: Mon Aug 13 11:10:57 UTC 2018

4 months later: Thu Dec 13 11:10:57 UTC 2018

2 years later: Sun Dec 13 11:10:57 UTC 2020