java.util.Date

The java.util.Date class represents date and time in java. It provides constructors and methods to deal with date and time in java.

The java.util.Date class implements Serializable, Cloneable and Comparable<Date> interface. It is inherited by java.sql.Date, java.sql.Time and java.sql.Timestamp interfaces.

After Calendar class, most of the constructors and methods of java.util.Date class has been deprecated. Here, we are not giving list of any deprecated constructor and method.

java.util.Date Constructors

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor** | **Description** |
| 1) | Date() | Creates a date object representing current date and time. |
| 2) | Date(long milliseconds) | Creates a date object for the given milliseconds since January 1, 1970, 00:00:00 GMT. |

java.util.Date Methods

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | [boolean after(Date date)](https://www.javatpoint.com/java-date-after-method) | tests if current date is after the given date. |
| 2) | [boolean before(Date date)](https://www.javatpoint.com/java-date-before-method) | tests if current date is before the given date. |
| 3) | [Object clone()](https://www.javatpoint.com/java-date-clone-method) | returns the clone object of current date. |
| 4) | [int compareTo(Date date)](https://www.javatpoint.com/java-date-compareto-method) | compares current date with given date. |
| 5) | [boolean equals(Date date)](https://www.javatpoint.com/java-date-equals-method) | compares current date with given date for equality. |
| 6) | [static Date from(Instant instant)](https://www.javatpoint.com/java-date-from-method) | returns an instance of Date object from Instant date. |
| 7) | [long getTime()](https://www.javatpoint.com/java-date-gettime-method) | returns the time represented by this date object. |
| 8) | [int hashCode()](https://www.javatpoint.com/java-date-hashcode-method) | returns the hash code value for this date object. |
| 9) | [void setTime(long time)](https://www.javatpoint.com/java-date-settime-method) | changes the current date and time to given time. |
| 10) | [Instant toInstant()](https://www.javatpoint.com/java-date-toinstant-method) | converts current date into Instant object. |
| 11) | [String toString()](https://www.javatpoint.com/java-date-tostring-method) | converts this date into Instant object. |

java.util.Date Example

Let's see the example to print date in java using java.util.Date class.

**1st way:**

1. java.util.Date date=**new** java.util.Date();
2. System.out.println(date);

Output:

Wed Mar 27 08:22:02 IST 2015

**2nd way:**

1. **long** millis=System.currentTimeMillis();
2. java.util.Date date=**new** java.util.Date(millis);
3. System.out.println(date);

Output:

Wed Mar 27 08:22:02 IST 2015

java.sql.Date

The java.sql.Date class represents only date in java. It inherits java.util.Date class.

The java.sql.Date instance is widely used in JDBC because it represents the date that can be stored in database.

Some constructors and methods of java.sql.Date class has been deprecated. Here, we are not giving list of any deprecated constructor and method.

java.sql.Date Constructor

|  |  |  |
| --- | --- | --- |
| **No.** | **Constructor** | **Description** |
| 1) | Date(long milliseconds) | Creates a sql date object for the given milliseconds since January 1, 1970, 00:00:00 GMT. |

java.sql.Date Methods

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | void setTime(long time) | changes the current sql date to given time. |
| 2) | Instant toInstant() | converts current sql date into Instant object. |
| 3) | LocalDate toLocalDate() | converts current sql date into LocalDate object. |
| 4) | String toString() | converts this sql date object to a string. |
| 5) | static Date valueOf(LocalDate date) | returns sql date object for the given LocalDate. |
| 6) | static Date valueOf(String date) | returns sql date object for the given String. |

java.sql.Date Example: get current date

Let's see the example to **print date in java** using java.sql.Date class.

1. **public** **class** SQLDateExample {
2. **public** **static** **void** main(String[] args) {
3. **long** millis=System.currentTimeMillis();
4. java.sql.Date date=**new** java.sql.Date(millis);
5. System.out.println(date);
6. }
7. }

Output:

2015-03-30

Java String to java.sql.Date Example

Let's see the example to **convert string into java.sql.Date** using valueOf() method.

1. **import** java.sql.Date;
2. **public** **class** StringToSQLDateExample {
3. **public** **static** **void** main(String[] args) {
4. String str="2015-03-31";
5. Date date=Date.valueOf(str);//converting string into sql date
6. System.out.println(date);
7. }
8. }

Output:

2015-03-31

Java Calendar Class

Java Calendar class 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 interface.

Java Calendar class declaration

Let's see the declaration of java.util.Calendar class.

1. **public** **abstract** **class** Calendar **extends** Object
2. **implements** Serializable, Cloneable, Comparable<Calendar>

List of Calendar Methods

|  |  |  |
| --- | --- | --- |
| **No** | **Method** | **Description** |
| 1. | [public void add(int field, int amount)](https://www.javatpoint.com/post/java-calendar-add-method) | Adds the specified (signed) amount of time to the given calendar field. |
| 2. | [public boolean after (Object when)](https://www.javatpoint.com/post/java-calendar-after-method) | The method Returns true if the time represented by this Calendar is after the time represented by when Object. |
| 3. | [public boolean before(Object when)](https://www.javatpoint.com/post/java-calendar-before-method) | The method Returns true if the time represented by this Calendar is before the time represented by when Object. |
| 4. | [public final void clear(int field)](https://www.javatpoint.com/post/java-calendar-clear-method) | Set the given calendar field value and the time value of this Calendar undefined. |
| 5. | [public Object clone()](https://www.javatpoint.com/post/java-calendar-clone-method) | Clone method provides the copy of the current object. |
| 6. | [public int compareTo(Calendar anotherCalendar)](https://www.javatpoint.com/post/java-calendar-compareto-method) | The compareTo() method of Calendar class compares the time values (millisecond offsets) between two calendar object. |
| 7. | [protected void complete()](https://www.javatpoint.com/post/java-calendar-complete-method) | It fills any unset fields in the calendar fields. |
| 8. | [protected abstract void computeFields()](https://www.javatpoint.com/post/java-calendar-computefields-method) | It converts the current millisecond time value time to calendar field values in fields[]. |
| 9. | [protected abstract void computeTime()](https://www.javatpoint.com/post/java-calendar-computetime-method) | It converts the current calendar field values in fields[] to the millisecond time value time. |
| 10. | [public boolean equals(Object object)](https://www.javatpoint.com/post/java-calendar-equals-method) | The equals() method compares two objects for equality and Returns true if they are equal. |
| 11. | [public int get(int field)](https://www.javatpoint.com/post/java-calendar-get-method) | In get() method fields of the calendar are passed as the parameter, and this method Returns the value of fields passed as the parameter. |
| 12. | [public int getActualMaximum(int field)](https://www.javatpoint.com/post/java-calendar-getactualmaximum-method) | Returns the Maximum possible value of the calendar field passed as the parameter to getActualMaximum() method. |
| 13. | [public int getActualMinimum(int field)](https://www.javatpoint.com/post/java-calendar-getactualminimum-method) | Returns the Minimum possible value of the calendar field passed as parameter to getActualMinimum() methot. |
| 14. | [public static Set<String> getAvailableCalendarTypes()](https://www.javatpoint.com/post/java-calendar-getavailablecalendartypes-method) | Returns a set which contains string set of all available calendar type supported by Java Runtime Environment. |
| 15. | [public static Locale[] getAvailableLocales()](https://www.javatpoint.com/post/java-calendar-getavailablelocales-method) | Returns an array of all locales available in java runtime environment. |
| 16. | [public String getCalendarType()](https://www.javatpoint.com/post/java-calendar-getcalendartype-method) | Returns in string all available calendar type supported by Java Runtime Environment. |
| 17. | [public String getDisplayName(int field, int style, Locale locale)](https://www.javatpoint.com/post/java-calendar-getdisplayname-method) | Returns the String representation of the calendar field value passed as the parameter in a given style and local. |
| 18. | [public Map<String,Integer> getDisplayNames(int field, int style, Locale locale)](https://www.javatpoint.com/post/java-calendar-getdisplayname-method) | Returns Map representation of the calendar field value passed as the parameter in a given style and local. |
| 19. | [public int getFirstDayOfWeek()](https://www.javatpoint.com/post/java-calendar-getfirstdayofweek-method) | Returns the first day of the week in integer form. |
| 20. | [public abstract int getGreatestMinimum(int field)](https://www.javatpoint.com/post/java-calendar-getgreatestminimum-method) | This method returns the highest minimum value of Calendar field passed as the parameter. |
| 21. | [public static Calendar getInstance()](https://www.javatpoint.com/post/java-calendar-getinstance-method) | This method is used with calendar object to get the instance of calendar according to current time zone set by java runtime environment |
| 22. | [public abstract int getLeastMaximum(int field)](https://www.javatpoint.com/post/java-calendar-getleastmaximum-method) | Returns smallest value from all maximum value for the field specified as the parameter to the method. |
| 23. | [public abstract int getMaximum(int field)](https://www.javatpoint.com/post/java-calendar-getmaximum-method) | This method is used with calendar object to get the maximum value of the specified calendar field as the parameter. |
| 24. | [public int getMinimalDaysInFirstWeek()](https://www.javatpoint.com/post/java-calendar-getminimaldaysinfirstweek-method) | Returns required minimum days in integer form. |
| 25. | [public abstract int getMinimum(int field)](https://www.javatpoint.com/post/java-calendar-getminimum-method) | This method is used with calendar object to get the minimum value of specified calendar field as the parameter. |
| 26. | [public final Date getTime()](https://www.javatpoint.com/post/java-calendar-gettime-method) | This method gets the time value of calendar object and Returns date. |
| 27. | [public long getTimeInMillis()](https://www.javatpoint.com/post/java-calendar-gettimeinmillis-method) | Returns the current time in millisecond. This method has long as return type. |
| 28. | [public TimeZone getTimeZone()](https://www.javatpoint.com/post/java-calendar-gettimezone-method) | This method gets the TimeZone of calendar object and Returns a TimeZone object. |
| 29. | [public int getWeeksInWeekYear()](https://www.javatpoint.com/post/java-calendar-getweeksinweekyear-method) | Return total weeks in week year. Weeks in week year is returned in integer form. |
| 30. | [public int getWeekYear()](https://www.javatpoint.com/post/java-calendar-getweekyear-method) | This method gets the week year represented by current Calendar. |
| 31. | [public int hashCode()](https://www.javatpoint.com/post/java-calendar-hashcode-method) | All other classes in Java overload hasCode() method. This method Returns the hash code for calendar object. |
| 32. | [protected final int internalGet(int field)](https://www.javatpoint.com/post/java-calendar-internalget-method) | This method returns the value of the calendar field passed as the parameter. |
| 33. | [Public boolean isLenient()](https://www.javatpoint.com/post/java-calendar-islenient-method) | Return Boolean value. True if the interpretation mode of this calendar is lenient; false otherwise. |
| 34. | [public final boolean isSet(int field)](https://www.javatpoint.com/post/java-calendar-isset-method) | This method checks if specified field as the parameter has been set or not. If not set then it returns false otherwise true. |
| 35. | [public boolean isWeekDateSupported()](https://www.javatpoint.com/post/java-calendar-isweekdatesupported-method) | Checks if this calendar supports week date. The default value is false. |
| 36. | [public abstract void roll(int field, boolean up)](https://www.javatpoint.com/post/java-calendar-roll-method) | This method increase or decrease the specified calendar field by one unit without affecting the other field |
| 37. | [public void set(int field, int value)](https://www.javatpoint.com/post/java-calendar-set-method) | Sets the specified calendar field by the specified value. |
| 38. | [public void setFirstDayOfWeek(int value)](https://www.javatpoint.com/post/java-calendar-setfirstdayofweek-method) | Sets the first day of the week. The value which is to be set as the first day of the week is passed as parameter. |
| 39. | [public void setMinimalDaysInFirstWeek(int value)](https://www.javatpoint.com/post/java-calendar-setminimaldaysinfirstweek-method) | Sets the minimal days required in the first week. The value which is to be set as minimal days in first week is passed as parameter. |
| 40. | public final void setTime(Date date) | Sets the Time of current calendar object. A Date object id passed as the parameter. |
| 41. | public void setTimeInMillis(long millis) | Sets the current time in millisecond. |
| 42. | [public void setTimeZone(TimeZone value)](https://www.javatpoint.com/post/java-calendar-settimzone-method) | Sets the TimeZone with passed TimeZone value (object) as the parameter. |
| 43. | [public void setWeekDate(int weekYear, int weekOfYear, int dayOfWeek)](https://www.javatpoint.com/post/java-calendar-setweekdate-method) | Sets the current date with specified integer value as the parameter. These values are weekYear, weekOfYear and dayOfWeek. |
| 44. | public final Instant toInstant() | The toInstant() method convert the current object to an instant. |
| 45. | public String toString() | Returns string representation of the current object. |

Java Calendar Class Example

1. **import** java.util.Calendar;
2. **public** **class** CalendarExample1 {
3. **public** **static** **void** main(String[] args) {
4. Calendar calendar = Calendar.getInstance();
5. System.out.println("The current date is : " + calendar.getTime());
6. calendar.add(Calendar.DATE, -15);
7. System.out.println("15 days ago: " + calendar.getTime());
8. calendar.add(Calendar.MONTH, 4);
9. System.out.println("4 months later: " + calendar.getTime());
10. calendar.add(Calendar.YEAR, 2);
11. System.out.println("2 years later: " + calendar.getTime());
12. }
13. }

Output:

The current date is : Thu Jan 19 18:47:02 IST 2017

15 days ago: Wed Jan 04 18:47:02 IST 2017

4 months later: Thu May 04 18:47:02 IST 2017

2 years later: Sat May 04 18:47:02 IST 2019

Java Calendar Class Example: get()

1. **import** java.util.\*;
2. **public** **class** CalendarExample2{
3. **public** **static** **void** main(String[] args) {
4. Calendar calendar = Calendar.getInstance();
5. System.out.println("At present Calendar's Year: " + calendar.get(Calendar.YEAR));
6. System.out.println("At present Calendar's Day: " + calendar.get(Calendar.DATE));
7. }
8. }

Output:

At present Calendar's Year: 2017

At present Calendar's Day: 20

Java Calendar Class Example: getInstance()

1. **import** java.util.\*;
2. **public** **class** CalendarExample3{
3. **public** **static** **void** main(String[] args) {
4. Calendar calendar = Calendar.getInstance();
5. System.out.print("At present Date And Time Is: " + calendar.getTime());
6. }
7. }

Output:

At present Date And Time Is: Fri Jan 20 14:26:19 IST 2017

Java Calendar Class Example: getMaximum()

1. **import** java.util.\*;
2. **public** **class** CalendarExample4 {
3. **public** **static** **void** main(String[] args) {
4. Calendar calendar = Calendar.getInstance();
5. **int** maximum = calendar.getMaximum(Calendar.DAY\_OF\_WEEK);
6. System.out.println("Maximum number of days in week: " + maximum);
7. maximum = calendar.getMaximum(Calendar.WEEK\_OF\_YEAR);
8. System.out.println("Maximum number of weeks in year: " + maximum);
9. }
10. }

Output:

Maximum number of days in week: 7

Maximum number of weeks in year: 53

Java Calendar Class Example: getMinimum()

1. **import** java.util.\*;
2. **public** **class** CalendarExample5 {
3. **public** **static** **void** main(String[] args) {
4. Calendar cal = Calendar.getInstance();
5. **int** maximum = cal.getMinimum(Calendar.DAY\_OF\_WEEK);
6. System.out.println("Minimum number of days in week: " + maximum);
7. maximum = cal.getMinimum(Calendar.WEEK\_OF\_YEAR);
8. System.out.println("Minimum number of weeks in year: " + maximum);
9. }
10. }

Output:

Minimum number of days in week: 1

Minimum number of weeks in year: 1

# Java TimeZone class

Java TimeZone class represents a time zone offset, and also figures out daylight savings. It inherits the Object class.

## Java TimeZone class declaration

Let's see the declaration of java.util.TimeZone class.

1. **public** **abstract** **class** TimeZone **extends** Object  **implements** Serializable, Cloneable

### Methods of Java TimeZone

|  |  |
| --- | --- |
| **Method** | **Description** |
| static String[] getAvailableIDs() | It is used to get all the available IDs supported. |
| static TimeZone getDefault() | It is used to get the default TimeZone for this host. |
| String getDisplayName() | It is used to return a name of this time zone suitable for presentation to the user in the default locale. |
| String getID() | It is used to get the ID of this time zone |
| int getOffset(long date) | It is used to return the offset of this time zone from UTC at the specified date. |
| void setID(String ID) | It is used to set the time zone ID |

## Java TimeZone class Example: getAvailableIDs()

1. **import** java.util.\*;
2. **public** **class** TimeZoneExample1 {
3. **public** **static** **void** main( String args[] ){
4. String[] id = TimeZone.getAvailableIDs();
5. System.out.println("In TimeZone class available Ids are: ");
6. **for** (**int** i=0; i<id.length; i++){
7. System.out.println(id[i]);
8. }
9. }
10. }

Output:

In TimeZone class available Ids are:

Africa/Abidjan

Africa/Accra

Africa/Addis\_Ababa

Africa/Algiers

Africa/Asmara

Africa/Asmera

Africa/Bamako

Africa/Bangui

Africa/Banjul

Africa/Bissau and so on ....

## Java TimeZone class Example: getOffset()

1. **import** java.util.\*;
2. **public** **class** TimeZoneExample2 {
3. **public** **static** **void** main( String args[] ){
4. TimeZone zone = TimeZone.getTimeZone("Asia/Kolkata");
5. System.out.println("The Offset value of TimeZone: " +
6. zone.getOffset(Calendar.ZONE\_OFFSET));
7. }
8. }

Output:

The Offset value of TimeZone: 19800000

## Java TimeZone class Example: getID()

1. **import** java.util.\*;
2. **public** **class** TimeZoneExample3 {
3. **public** **static** **void** main( String args[] ){
4. TimeZone timezone = TimeZone.getTimeZone("Asia/Kolkata");
5. System.out.println("Value of ID is: " + timezone.getID());
6. }
7. }

Output:

Value of ID is: Asia/Kolkata

## Java TimeZone class Example: getDisplayName()

1. **import** java.util.\*;
2. **public** **class** TimeZoneExample4 {
3. **public** **static** **void** main( String args[] ){
4. TimeZone zone = TimeZone.getDefault();
5. String name = zone.getDisplayName();
6. System.out.println("Display name for default time zone: "+ name);
7. }
8. }

Output:

Display name for default time zone: India Standard Time

Java Date Format

There are two classes for formatting date in java: DateFormat and SimpleDateFormat.

The java.text.DateFormat class provides various methods to format and parse date and time in java in language independent manner. The DateFormat class is an abstract class. java.text.Format is the parent class and java.text.SimpleDateFormat is the subclass of java.text.DateFormat class.

In java, converting date into string is called formatting and vice-versa parsing. In other words, *formatting means date to string* and *parsing means string to date*.

java.text.DateFormat Fields

1. **protected** Calendar calendar
2. **protected** NumberFormat numberFormat
3. **public** **static** **final** **int** ERA\_FIELD
4. **public** **static** **final** **int** YEAR\_FIELD
5. **public** **static** **final** **int** MONTH\_FIELD
6. **public** **static** **final** **int** DATE\_FIELD
7. **public** **static** **final** **int** HOUR\_OF\_DAY1\_FIELD
8. **public** **static** **final** **int** HOUR\_OF\_DAY0\_FIELD
9. **public** **static** **final** **int** MINUTE\_FIELD
10. **public** **static** **final** **int** SECOND\_FIELD
11. **public** **static** **final** **int** MILLISECOND\_FIELD
12. **public** **static** **final** **int** DAY\_OF\_WEEK\_FIELD
13. **public** **static** **final** **int** DAY\_OF\_YEAR\_FIELD
14. **public** **static** **final** **int** DAY\_OF\_WEEK\_IN\_MONTH\_FIELD
15. **public** **static** **final** **int** WEEK\_OF\_YEAR\_FIELD
16. **public** **static** **final** **int** WEEK\_OF\_MONTH\_FIELD
17. **public** **static** **final** **int** AM\_PM\_FIELD
18. **public** **static** **final** **int** HOUR1\_FIELD
19. **public** **static** **final** **int** HOUR0\_FIELD
20. **public** **static** **final** **int** TIMEZONE\_FIELD
21. **public** **static** **final** **int** FULL
22. **public** **static** **final** **int** LONG
23. **public** **static** **final** **int** MEDIUM
24. **public** **static** **final** **int** SHORT
25. **public** **static** **final** **int** DEFAULT

java.text.DateFormat Methods

|  |  |  |
| --- | --- | --- |
| **No.** | **Public Method** | **Description** |
| 1) | final String format(Date date) | converts given Date object into string. |
| 2) | Date parse(String source)throws ParseException | converts string into Date object. |
| 3) | static final DateFormat getTimeInstance() | returns time formatter with default formatting style for the default locale. |
| 4) | static final DateFormat getTimeInstance(int style) | returns time formatter with the given formatting style for the default locale. |
| 5) | static final DateFormat getTimeInstance(int style, Locale locale) | returns time formatter with the given formatting style for the given locale. |
| 6) | static final DateFormat getDateInstance() | returns date formatter with default formatting style for the default locale. |
| 7) | static final DateFormat getDateInstance(int style) | returns date formatter with the given formatting style for the default locale. |
| 8) | static final DateFormat getDateInstance(int style, Locale locale) | returns date formatter with the given formatting style for the given locale. |
| 9) | static final DateFormat getDateTimeInstance() | returns date/time formatter with default formatting style for the default locale. |
| 10) | static final DateFormat getDateTimeInstance(int dateStyle,int timeStyle) | returns date/time formatter with the given date formatting style and time formatting style for the default locale. |
| 11) | static final DateFormat getDateTimeInstance(int dateStyle, int timeStyle, Locale locale) | returns date/time formatter with the given date formatting style and time formatting style for the given locale. |
| 12) | static final DateFormat getInstance() | returns date/time formatter with short formatting style for date and time. |
| 13) | static Locale[] getAvailableLocales() | returns an array of available locales. |
| 14) | Calendar getCalendar() | returns an instance of Calendar for this DateFormat instance. |
| 15) | NumberFormat getNumberFormat() | returns an instance of NumberFormat for this DateFormat instance. |
| 16) | TimeZone getTimeZone() | returns an instance of TimeZone for this DateFormat instance. |

Java DateFormat Example: Date to String

Let's see the simple example to **format date and time in java** using java.text.DateFormat class.

1. **import** java.text.DateFormat;
2. **import** java.util.Date;
3. **public** **class** DateFormatExample {
4. **public** **static** **void** main(String[] args) {
5. Date currentDate = **new** Date();
6. System.out.println("Current Date: "+currentDate);
7. String dateToStr = DateFormat.getInstance().format(currentDate);
8. System.out.println("Date Format using getInstance(): "+dateToStr);
9. }
10. }

Output:

Current Date: Tue Mar 31 14:37:23 IST 2015

Date Format using getInstance(): 31/3/15 2:37 PM

Let's see the full example to **format date and time in java** using java.text.DateFormat class.

1. **import** java.text.DateFormat;
2. **import** java.util.Date;
3. **public** **class** DateFormatExample2 {
4. **public** **static** **void** main(String[] args) {
5. Date currentDate = **new** Date();
6. System.out.println("Current Date: "+currentDate);
8. String dateToStr = DateFormat.getInstance().format(currentDate);
9. System.out.println("Date Format using getInstance(): "+dateToStr);
11. dateToStr = DateFormat.getDateInstance().format(currentDate);
12. System.out.println("Date Format using getDateInstance(): "+dateToStr);
14. dateToStr = DateFormat.getTimeInstance().format(currentDate);
15. System.out.println("Date Format using getTimeInstance(): "+dateToStr);
17. dateToStr = DateFormat.getDateTimeInstance().format(currentDate);
18. System.out.println("Date Format using getDateTimeInstance(): "+dateToStr);
20. dateToStr = DateFormat.getTimeInstance(DateFormat.SHORT).format(currentDate);
21. System.out.println("Date Format using getTimeInstance(DateFormat.SHORT): "+dateToStr);
23. dateToStr = DateFormat.getTimeInstance(DateFormat.MEDIUM).format(currentDate);
24. System.out.println("Date Format using getTimeInstance(DateFormat.MEDIUM): "+dateToStr);
26. dateToStr = DateFormat.getTimeInstance(DateFormat.LONG).format(currentDate);
27. System.out.println("Date Format using getTimeInstance(DateFormat.LONG): "+dateToStr);
29. dateToStr = DateFormat.getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT).format(currentDate);
30. System.out.println("Date Format using getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT): "+dateToStr);
32. }
33. }

Output:

Current Date: Tue Mar 31 14:37:23 IST 2015

Date Format using getInstance(): 31/3/15 2:37 PM

Date Format using getDateInstance(): 31 Mar, 2015

Date Format using getTimeInstance(): 2:37:23 PM

Date Format using getDateTimeInstance(): 31 Mar, 2015 2:37:23 PM

Date Format using getTimeInstance(DateFormat.SHORT): 2:37 PM

Date Format using getTimeInstance(DateFormat.MEDIUM): 2:37:23 PM

Date Format using getTimeInstance(DateFormat.LONG): 2:37:23 PM IST

Date Format using getDateTimeInstance(DateFormat.LONG,DateFormat.SHORT): 31 March, 2015 2:37 PM

Java DateFormat Example: String to Date

Let's see the simple example to **convert string into date** using java.text.DateFormat class.

1. **import** java.text.DateFormat;
2. **import** java.util.Date;
3. **public** **class** DateFormatExample3 {
4. **public** **static** **void** main(String[] args)**throws** Exception {
5. Date d = DateFormat.getDateInstance().parse("31 Mar, 2015");
6. System.out.println("Date is: "+d);
7. }
8. }

Output:

Date is: Tue Mar 31 00:00:00 IST 2015

# Java SimpleDateFormat

The java.text.SimpleDateFormat class provides methods to format and parse date and time in java. The SimpleDateFormat is a concrete class for formatting and parsing date which inherits java.text.DateFormat class.

Notice that formatting means converting date to string and parsing means converting string to date.

## Java SimpleDateFormat Example: Date to String

Let's see the simple example to **format date in java** using java.text.SimpleDateFormat class.

1. **import** java.text.SimpleDateFormat;
2. **import** java.util.Date;
3. **public** **class** SimpleDateFormatExample {
4. **public** **static** **void** main(String[] args) {
5. Date date = **new** Date();
6. SimpleDateFormat formatter = **new** SimpleDateFormat("dd/MM/yyyy");
7. String strDate= formatter.format(date);
8. System.out.println(strDate);
9. }
10. }

Output:

13/04/2015

#### Note: M (capital M) represents month and m (small m) represents minute in java.

Let's see the full example to **format date and time in java** using java.text.SimpleDateFormat class.

1. **import** java.text.ParseException;
2. **import** java.text.SimpleDateFormat;
3. **import** java.util.Date;
4. **import** java.util.Locale;
5. **public** **class** SimpleDateFormatExample2 {
6. **public** **static** **void** main(String[] args) {
7. Date date = **new** Date();
8. SimpleDateFormat formatter = **new** SimpleDateFormat("MM/dd/yyyy");
9. String strDate = formatter.format(date);
10. System.out.println("Date Format with MM/dd/yyyy : "+strDate);
12. formatter = **new** SimpleDateFormat("dd-M-yyyy hh:mm:ss");
13. strDate = formatter.format(date);
14. System.out.println("Date Format with dd-M-yyyy hh:mm:ss : "+strDate);
16. formatter = **new** SimpleDateFormat("dd MMMM yyyy");
17. strDate = formatter.format(date);
18. System.out.println("Date Format with dd MMMM yyyy : "+strDate);
20. formatter = **new** SimpleDateFormat("dd MMMM yyyy zzzz");
21. strDate = formatter.format(date);
22. System.out.println("Date Format with dd MMMM yyyy zzzz : "+strDate);
24. formatter = **new** SimpleDateFormat("E, dd MMM yyyy HH:mm:ss z");
25. strDate = formatter.format(date);
26. System.out.println("Date Format with E, dd MMM yyyy HH:mm:ss z : "+strDate);
27. }
28. }

Output:

Date Format with MM/dd/yyyy : 04/13/2015

Date Format with dd-M-yyyy hh:mm:ss : 13-4-2015 10:59:26

Date Format with dd MMMM yyyy : 13 April 2015

Date Format with dd MMMM yyyy zzzz : 13 April 2015 India Standard Time

Date Format with E, dd MMM yyyy HH:mm:ss z : Mon, 13 Apr 2015 22:59:26 IST

## Java SimpleDateFormat Example: String to Date

Let's see the simple example to **convert string into date** using java.text.SimpleDateFormat class.

1. **import** java.text.ParseException;
2. **import** java.text.SimpleDateFormat;
3. **import** java.util.Date;
4. **public** **class** SimpleDateFormatExample3 {
5. **public** **static** **void** main(String[] args) {
6. SimpleDateFormat formatter = **new** SimpleDateFormat("dd/MM/yyyy");
7. **try** {
8. Date date = formatter.parse("31/03/2015");
9. System.out.println("Date is: "+date);
10. } **catch** (ParseException e) {e.printStackTrace();}
11. }
12. }

Output:

Date is: Tue Mar 31 00:00:00 IST 2015

# Get Current Date and Time in Java

There are many ways to get current date and time in java. There are many classes that can be used to get current date and time in java.

1. java.time.format.DateTimeFormatter class
2. java.text.SimpleDateFormat class
3. java.time.LocalDate class
4. java.time.LocalTime class
5. java.time.LocalDateTime class
6. java.time.Clock class
7. java.util.Date class
8. java.sql.Date class
9. java.util.Calendar class

## Get Current Date and Time: java.time.format.DateTimeFormatter

The LocalDateTime.now() method returns the instance of LocalDateTime class. If we print the instance of LocalDateTime class, it prints current date and time. To format the current date, you can use DateTimeFormatter class which is included in JDK 1.8.

1. **import** java.time.format.DateTimeFormatter;
2. **import** java.time.LocalDateTime;
3. **public** **class** CurrentDateTimeExample1 {
4. **public** **static** **void** main(String[] args) {
5. DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");
6. LocalDateTime now = LocalDateTime.now();
7. System.out.println(dtf.format(now));
8. }
9. }

Output:

2017/11/06 12:11:58

## Get Current Date and Time: java.text.SimpleDateFormat

The SimpleDateFormat class is also used for formatting date and time. But it is old approach.

1. **import** java.text.SimpleDateFormat;
2. **import** java.util.Date;
3. **public** **class** CurrentDateTimeExample2 {
4. **public** **static** **void** main(String[] args) {
5. SimpleDateFormat formatter = **new** SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
6. Date date = **new** Date();
7. System.out.println(formatter.format(date));
8. }
9. }

Output:

06/11/2017 12:26:18

## Get Current Date: java.time.LocalDate

The LocalDate.now() method returns the instance of LocalDate class. If we print the instance of LocalDate class, it prints current date.

1. System.out.println(java.time.LocalDate.now());

Output:

2017-01-23

## Get Current Time: java.time.LocalTime

The LocalTime.now() method returns the instance of LocalTime class. If we print the instance of LocalTime class, it prints current time.

1. System.out.println(java.time.LocalTime.now());

Output:

00:01:14.341

## Get Current Date & Time: java.time.LocalDateTime

The LocalDateTime.now() method returns the instance of LocalDateTime class. If we print the instance of LocalDateTime class, it prints current date and time both.

1. System.out.println(java.time.LocalDateTime.now());

Output:

2017-01-24T00:03:31.593

## Get Current Date & Time: java.time.Clock

The Clock.systemUTC().instant() method returns current date and time both.

1. System.out.println(java.time.Clock.systemUTC().instant());

Output:

2017-01-23T18:35:23.669Z

## Get Current Date & Time: java.util.Date

By printing the instance of java.util.Date class, you can print current date and time in java. There are two ways to do so.

**1st way:**

1. java.util.Date date=**new** java.util.Date();
2. System.out.println(date);

**2nd way:**

1. **long** millis=System.currentTimeMillis();
2. java.util.Date date=**new** java.util.Date(millis);
3. System.out.println(date);

Output:

Thu Mar 26 08:22:02 IST 2015

## Get Current Date: java.sql.Date

By printing the instance of java.sql.Date class, you can print current date in java. It doesn't print time. This date instance is generally used to save current date in database.

1. **long** millis=System.currentTimeMillis();
2. java.sql.Date date=**new** java.sql.Date(millis);
3. System.out.println(date);

Output:

2015-03-26

## Get Current Date & Time: java.util.Calendar

Calendar class can be used to get the instance of Date class. The getTime() method of Calendar class returns the instance of java.util.Date. The Calendar.getInstance() method returns the instance of Calendar class.

1. Date date=java.util.Calendar.getInstance().getTime();
2. System.out.println(date);

Output:

Thu Mar 26 08:22:02 IST 2015

#### Note: It is recommended to use Calendar class for getting current date and time in classical Date API. Since Java 8, you can use LocalDate, LocalTime or LocalDateTime classes.

## Date Formatting Using SimpleDateFormat

SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting.

### Example

import java.util.\*;

import java.text.\*;

public class DateDemo {

public static void main(String args[]) {

Date dNow = new Date( );

SimpleDateFormat ft =

new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

System.out.println("Current Date: " + ft.format(dNow));

}

}

This will produce the following result −

### Output

Current Date: Sun 2004.07.18 at 04:14:09 PM PDT

## Simple DateFormat Format Codes

To specify the time format, use a time pattern string. In this pattern, all ASCII letters are reserved as pattern letters, which are defined as the following −

|  |  |  |
| --- | --- | --- |
| **Character** | **Description** | **Example** |
| G | Era designator | AD |
| y | Year in four digits | 2001 |
| M | Month in year | July or 07 |
| d | Day in month | 10 |
| h | Hour in A.M./P.M. (1~12) | 12 |
| H | Hour in day (0~23) | 22 |
| m | Minute in hour | 30 |
| s | Second in minute | 55 |
| S | Millisecond | 234 |
| E | Day in week | Tuesday |
| D | Day in year | 360 |
| F | Day of week in month | 2 (second Wed. in July) |
| w | Week in year | 40 |
| W | Week in month | 1 |
| a | A.M./P.M. marker | PM |
| k | Hour in day (1~24) | 24 |
| K | Hour in A.M./P.M. (0~11) | 10 |
| z | Time zone | Eastern Standard Time |
| ' | Escape for text | Delimiter |
| " | Single quote | ` |

## Date Formatting Using printf

Date and time formatting can be done very easily using **printf** method. You use a two-letter format, starting with **t** and ending in one of the letters of the table as shown in the following code.

### Example

import java.util.Date;

public class DateDemo {

public static void main(String args[]) {

// Instantiate a Date object

Date date = new Date();

// display time and date

String str = String.format("Current Date/Time : %tc", date );

System.out.printf(str);

}

}

This will produce the following result −

### Output

Current Date/Time : Sat Dec 15 16:37:57 MST 2012

It would be a bit silly if you had to supply the date multiple times to format each part. For that reason, a format string can indicate the index of the argument to be formatted.

The index must immediately follow the % and it must be terminated by a $.

### Example

import java.util.Date;

public class DateDemo {

public static void main(String args[]) {

// Instantiate a Date object

Date date = new Date();

// display time and date

System.out.printf("%1$s %2$tB %2$td, %2$tY", "Due date:", date);

}

}

This will produce the following result −

### Output

Due date: February 09, 2004

Alternatively, you can use the < flag. It indicates that the same argument as in the preceding format specification should be used again.

### Example

import java.util.Date;

public class DateDemo {

public static void main(String args[]) {

// Instantiate a Date object

Date date = new Date();

// display formatted date

System.out.printf("%s %tB %<te, %<tY", "Due date:", date);

}

}

This will produce the following result −

### Output

Due date: February 09, 2004

## Date and Time Conversion Characters

|  |  |  |
| --- | --- | --- |
| **Character** | **Description** | **Example** |
| c | Complete date and time | Mon May 04 09:51:52 CDT 2009 |
| F | ISO 8601 date | 2004-02-09 |
| D | U.S. formatted date (month/day/year) | 02/09/2004 |
| T | 24-hour time | 18:05:19 |
| r | 12-hour time | 06:05:19 pm |
| R | 24-hour time, no seconds | 18:05 |
| Y | Four-digit year (with leading zeroes) | 2004 |
| y | Last two digits of the year (with leading zeroes) | 04 |
| C | First two digits of the year (with leading zeroes) | 20 |
| B | Full month name | February |
| b | Abbreviated month name | Feb |
| m | Two-digit month (with leading zeroes) | 02 |
| d | Two-digit day (with leading zeroes) | 03 |
| e | Two-digit day (without leading zeroes) | 9 |
| A | Full weekday name | Monday |
| a | Abbreviated weekday name | Mon |
| j | Three-digit day of year (with leading zeroes) | 069 |
| H | Two-digit hour (with leading zeroes), between 00 and 23 | 18 |
| k | Two-digit hour (without leading zeroes), between 0 and 23 | 18 |
| I | Two-digit hour (with leading zeroes), between 01 and 12 | 06 |
| l | Two-digit hour (without leading zeroes), between 1 and 12 | 6 |
| M | Two-digit minutes (with leading zeroes) | 05 |
| S | Two-digit seconds (with leading zeroes) | 19 |
| L | Three-digit milliseconds (with leading zeroes) | 047 |
| N | Nine-digit nanoseconds (with leading zeroes) | 047000000 |
| P | Uppercase morning or afternoon marker | PM |
| p | Lowercase morning or afternoon marker | pm |
| z | RFC 822 numeric offset from GMT | -0800 |
| Z | Time zone | PST |
| s | Seconds since 1970-01-01 00:00:00 GMT | 1078884319 |
| Q | Milliseconds since 1970-01-01 00:00:00 GMT | 1078884319047 |

There are other useful classes related to Date and time. For more details, you can refer to Java Standard documentation.

## Parsing Strings into Dates

The SimpleDateFormat class has some additional methods, notably parse( ), which tries to parse a string according to the format stored in the given SimpleDateFormat object.

### Example

import java.util.\*;

import java.text.\*;

public class DateDemo {

public static void main(String args[]) {

SimpleDateFormat ft = new SimpleDateFormat ("yyyy-MM-dd");

String input = args.length == 0 ? "1818-11-11" : args[0];

System.out.print(input + " Parses as ");

Date t;

try {

t = ft.parse(input);

System.out.println(t);

} catch (ParseException e) {

System.out.println("Unparseable using " + ft);

}

}

}

A sample run of the above program would produce the following result −

### Output

1818-11-11 Parses as Wed Nov 11 00:00:00 EST 1818

## Sleeping for a While

You can sleep for any period of time from one millisecond up to the lifetime of your computer. For example, the following program would sleep for 3 seconds −

### Example

import java.util.\*;

public class SleepDemo {

public static void main(String args[]) {

try {

System.out.println(new Date( ) + "\n");

Thread.sleep(5\*60\*10);

System.out.println(new Date( ) + "\n");

} catch (Exception e) {

System.out.println("Got an exception!");

}

}

}

This will produce the following result −

### Output

Sun May 03 18:04:41 GMT 2009

Sun May 03 18:04:51 GMT 2009

## Measuring Elapsed Time

Sometimes, you may need to measure point in time in milliseconds. So let's re-write the above example once again −

### Example

import java.util.\*;

public class DiffDemo {

public static void main(String args[]) {

try {

long start = System.currentTimeMillis( );

System.out.println(new Date( ) + "\n");

Thread.sleep(5\*60\*10);

System.out.println(new Date( ) + "\n");

long end = System.currentTimeMillis( );

long diff = end - start;

System.out.println("Difference is : " + diff);

} catch (Exception e) {

System.out.println("Got an exception!");

}

}

}

This will produce the following result −

### Output

Sun May 03 18:16:51 GMT 2009

Sun May 03 18:16:57 GMT 2009

Difference is : 5993

## GregorianCalendar Class

GregorianCalendar is a concrete implementation of a Calendar class that implements the normal Gregorian calendar with which you are familiar. We did not discuss Calendar class in this tutorial, you can look up standard Java documentation for this.

The **getInstance( )** method of Calendar returns a GregorianCalendar initialized with the current date and time in the default locale and time zone. GregorianCalendar defines two fields: AD and BC. These represent the two eras defined by the Gregorian calendar.

There are also several constructors for GregorianCalendar objects −

|  |  |
| --- | --- |
| **Sr.No.** | **Constructor & Description** |
| 1 | **GregorianCalendar()**  Constructs a default GregorianCalendar using the current time in the default time zone with the default locale. |
| 2 | **GregorianCalendar(int year, int month, int date)**  Constructs a GregorianCalendar with the given date set in the default time zone with the default locale. |
| 3 | **GregorianCalendar(int year, int month, int date, int hour, int minute)**  Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale. |
| 4 | **GregorianCalendar(int year, int month, int date, int hour, int minute, int second)**  Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale. |
| 5 | **GregorianCalendar(Locale aLocale)**  Constructs a GregorianCalendar based on the current time in the default time zone with the given locale. |
| 6 | **GregorianCalendar(TimeZone zone)**  Constructs a GregorianCalendar based on the current time in the given time zone with the default locale. |
| 7 | **GregorianCalendar(TimeZone zone, Locale aLocale)**  Constructs a GregorianCalendar based on the current time in the given time zone with the given locale. |

Here is the list of few useful support methods provided by GregorianCalendar class −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **void add(int field, int amount)**  Adds the specified (signed) amount of time to the given time field, based on the calendar's rules. |
| 2 | **protected void computeFields()**  Converts UTC as milliseconds to time field values. |
| 3 | **protected void computeTime()**  Overrides Calendar Converts time field values to UTC as milliseconds. |
| 4 | **boolean equals(Object obj)**  Compares this GregorianCalendar to an object reference. |
| 5 | **int get(int field)**  Gets the value for a given time field. |
| 6 | **int getActualMaximum(int field)**  Returns the maximum value that this field could have, given the current date. |
| 7 | **int getActualMinimum(int field)**  Returns the minimum value that this field could have, given the current date. |
| 8 | **int getGreatestMinimum(int field)**  Returns highest minimum value for the given field if varies. |
| 9 | **Date getGregorianChange()**  Gets the Gregorian Calendar change date. |
| 10 | **int getLeastMaximum(int field)**  Returns lowest maximum value for the given field if varies. |
| 11 | **int getMaximum(int field)**  Returns maximum value for the given field. |
| 12 | **Date getTime()**  Gets this Calendar's current time. |
| 13 | **long getTimeInMillis()**  Gets this Calendar's current time as a long. |
| 14 | **TimeZone getTimeZone()**  Gets the time zone. |
| 15 | **int getMinimum(int field)**  Returns minimum value for the given field. |
| 16 | **int hashCode()**  Overrides hashCode. |
| 17 | **boolean isLeapYear(int year)**  Determines if the given year is a leap year. |
| 18 | **void roll(int field, boolean up)**  Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields. |
| 19 | **void set(int field, int value)**  Sets the time field with the given value. |
| 20 | **void set(int year, int month, int date)**  Sets the values for the fields year, month, and date. |
| 21 | **void set(int year, int month, int date, int hour, int minute)**  Sets the values for the fields year, month, date, hour, and minute. |
| 22 | **void set(int year, int month, int date, int hour, int minute, int second)**  Sets the values for the fields year, month, date, hour, minute, and second. |
| 23 | **void setGregorianChange(Date date)**  Sets the GregorianCalendar change date. |
| 24 | **void setTime(Date date)**  Sets this Calendar's current time with the given Date. |
| 25 | **void setTimeInMillis(long millis)**  Sets this Calendar's current time from the given long value. |
| 26 | **void setTimeZone(TimeZone value)**  Sets the time zone with the given time zone value. |
| 27 | **String toString()**  Returns a string representation of this calendar. |

### Example

import java.util.\*;

public class GregorianCalendarDemo {

public static void main(String args[]) {

String months[] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",

"Oct", "Nov", "Dec"};

int year;

// Create a Gregorian calendar initialized

// with the current date and time in the

// default locale and timezone.

GregorianCalendar gcalendar = new GregorianCalendar();

// Display current time and date information.

System.out.print("Date: ");

System.out.print(months[gcalendar.get(Calendar.MONTH)]);

System.out.print(" " + gcalendar.get(Calendar.DATE) + " ");

System.out.println(year = gcalendar.get(Calendar.YEAR));

System.out.print("Time: ");

System.out.print(gcalendar.get(Calendar.HOUR) + ":");

System.out.print(gcalendar.get(Calendar.MINUTE) + ":");

System.out.println(gcalendar.get(Calendar.SECOND));

// Test if the current year is a leap year

if(gcalendar.isLeapYear(year)) {

System.out.println("The current year is a leap year");

}else {

System.out.println("The current year is not a leap year");

}

}

}

This will produce the following result −

### Output

Date: Apr 22 2009

Time: 11:25:27

The current year is not a leap year

Java Date and Time

The java.time, java.util, java.sql and java.text packages contains classes for representing date and time. Following classes are important for dealing with date in java.

Java 8 Date/Time API

Java has introduced a new Date and Time API since Java 8. The java.time package contains Java 8 Date and Time classes.

* [java.time.LocalDate class](https://www.javatpoint.com/java-localdate)
* [java.time.LocalTime class](https://www.javatpoint.com/java-localtime)
* [java.time.LocalDateTime class](https://www.javatpoint.com/java-localdatetime)
* [java.time.MonthDay class](https://www.javatpoint.com/java-monthday)
* [java.time.OffsetTime class](https://www.javatpoint.com/java-offsettime)
* [java.time.OffsetDateTime class](https://www.javatpoint.com/java-offsetdatetime)
* [java.time.Clock class](https://www.javatpoint.com/java-clock)
* [java.time.ZonedDateTime class](https://www.javatpoint.com/java-zoneddatetime)
* [java.time.ZoneId class](https://www.javatpoint.com/java-zoneid)
* [java.time.ZoneOffset class](https://www.javatpoint.com/java-zoneoffset)
* [java.time.Year class](https://www.javatpoint.com/java-year)
* [java.time.YearMonth class](https://www.javatpoint.com/java-yearmonth)
* [java.time.Period class](https://www.javatpoint.com/java-period)
* [java.time.Duration class](https://www.javatpoint.com/java-duration)
* [java.time.Instant class](https://www.javatpoint.com/java-instant)
* [java.time.DayOfWeek enum](https://www.javatpoint.com/java-dayofweek-enum)
* [java.time.Month enum](https://www.javatpoint.com/java-month-enum)

Classical Date/Time API

But classical or old Java Date API is also useful. Let's see the list of classical Date and Time classes.

* [java.util.Date class](https://www.javatpoint.com/java-util-date)
* [java.sql.Date class](https://www.javatpoint.com/java-sql-date)
* [java.util.Calendar class](https://www.javatpoint.com/java-util-calendar)
* java.util.GregorianCalendar class
* [java.util.TimeZone class](https://www.javatpoint.com/java-util-timezone)
* java.sql.Time class
* java.sql.Timestamp class

Formatting Date and Time

We can format date and time in java by the use of following classes:

* [java.text.DateFormat class](https://www.javatpoint.com/java-date-format)
* [java.text.SimpleDateFormat class](https://www.javatpoint.com/java-simpledateformat)

Get Current Date and Time

We can get current date and time in java by many ways.

* [How to get current Date and Time](https://www.javatpoint.com/java-get-current-date)

# Java LocalDate class

Java LocalDate class is an immutable class that represents Date with a default format of yyyy-MM-dd. It inherits Object class and implements the ChronoLocalDate interface

## Java LocalDate class declaration

Let's see the declaration of java.time.LocalDate class.

1. **public** **final** **class** LocalDate **extends** Object
2. **implements** Temporal, TemporalAdjuster, ChronoLocalDate, Serializable

### Methods of Java LocalDate

|  |  |
| --- | --- |
| **Method** | **Description** |
| LocalDateTime atTime(int hour, int minute) | It is used to combine this date with a time to create a LocalDateTime. |
| int compareTo(ChronoLocalDate other) | It is used to compares this date to another date. |
| boolean equals(Object obj) | It is used to check if this date is equal to another date. |
| String format(DateTimeFormatter formatter) | It is used to format this date using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this date as an int. |
| boolean isLeapYear() | It is used to check if the year is a leap year, according to the ISO proleptic calendar system rules. |
| LocalDate minusDays(long daysToSubtract) | It is used to return a copy of this LocalDate with the specified number of days subtracted. |
| LocalDate minusMonths(long monthsToSubtract) | It is used to return a copy of this LocalDate with the specified number of months subtracted. |
| static LocalDate now() | It is used to obtain the current date from the system clock in the default time-zone. |
| LocalDate plusDays(long daysToAdd) | It is used to return a copy of this LocalDate with the specified number of days added. |
| LocalDate plusMonths(long monthsToAdd) | It is used to return a copy of this LocalDate with the specified number of months added. |

## Java LocalDate Example

1. **import** java.time.LocalDate;
2. **public** **class** LocalDateExample1 {
3. **public** **static** **void** main(String[] args) {
4. LocalDate date = LocalDate.now();
5. LocalDate yesterday = date.minusDays(1);
6. LocalDate tomorrow = yesterday.plusDays(2);
7. System.out.println("Today date: "+date);
8. System.out.println("Yesterday date: "+yesterday);
9. System.out.println("Tommorow date: "+tomorrow);
10. }
11. }

Output:

Today date: 2017-01-13

Yesterday date: 2017-01-12

Tommorow date: 2017-01-14

## Java LocalDate Example: isLeapYear()

1. **import** java.time.LocalDate;
2. **public** **class** LocalDateExample2 {
3. **public** **static** **void** main(String[] args) {
4. LocalDate date1 = LocalDate.of(2017, 1, 13);
5. System.out.println(date1.isLeapYear());
6. LocalDate date2 = LocalDate.of(2016, 9, 23);
7. System.out.println(date2.isLeapYear());
8. }
9. }

Output:

false

true

## Java LocalDate Example: atTime()

1. **import** java.time.\*;
2. **public** **class** LocalDateExample3 {
3. **public** **static** **void** main(String[] args) {
4. LocalDate date = LocalDate.of(2017, 1, 13);
5. LocalDateTime datetime = date.atTime(1,50,9);
6. System.out.println(datetime);
7. }
8. }

Output:

2017-01-13T01:50:09

# Java LocalTime class

Java LocalTime class is an immutable class that represents time with a default format of hour-minute-second. It inherits Object class and implements the Comparable interface.

## Java LocalTime class declaration

Let's see the declaration of java.time.LocalTime class.

1. **public** **final** **class** LocalTime **extends** Object
2. **implements** Temporal, TemporalAdjuster, Comparable<LocalTime>, Serializable

### Methods of Java LocalTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| LocalDateTime atDate(LocalDate date) | It is used to combine this time with a date to create a LocalDateTime. |
| int compareTo(LocalTime other) | It is used to compare this time to another time. |
| String format(DateTimeFormatter formatter) | It is used to format this time using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this time as an int. |
| LocalTime minusHours(long hoursToSubtract) | It is used to return a copy of this LocalTime with the specified number of hours subtracted. |
| LocalTime minusMinutes(long minutesToSubtract) | It is used to return a copy of this LocalTime with the specified number of minutes subtracted. |
| static LocalTime now() | It is used to obtain the current time from the system clock in the default time-zone. |
| static LocalTime of(int hour, int minute, int second) | It is used to obtain an instance of LocalTime from an hour, minute and second. |
| LocalTime plusHours(long hoursToAdd) | It is used to return a copy of this LocalTime with the specified number of hours added. |
| LocalTime plusMinutes(long minutesToAdd) | It is used to return a copy of this LocalTime with the specified number of minutes added. |

## Java LocalTime Example: now()

1. **import** java.time.LocalTime;
2. **public** **class** LocalTimeExample1 {
3. **public** **static** **void** main(String[] args) {
4. LocalTime time = LocalTime.now();
5. System.out.println(time);
6. }
7. }

Output:

15:19:47.459

## Java LocalTime Example: of()

1. **import** java.time.LocalTime;
2. **public** **class** LocalTimeExample2 {
3. **public** **static** **void** main(String[] args) {
4. LocalTime time = LocalTime.of(10,43,12);
5. System.out.println(time);
6. }
7. }

Output:

10:43:12

## Java LocalTime Example: minusHours() and minusMinutes()

1. **import** java.time.LocalTime;
2. **public** **class** LocalTimeExample3 {
3. **public** **static** **void** main(String[] args) {
4. LocalTime time1 = LocalTime.of(10,43,12);
5. System.out.println(time1);
6. LocalTime time2=time1.minusHours(2);
7. LocalTime time3=time2.minusMinutes(34);
8. System.out.println(time3);
9. }
10. }

Output:

10:43:12

08:09:12

## Java LocalTime Example: plusHours() and plusMinutes()

1. **import** java.time.LocalTime;
2. **public** **class** LocalTimeExample4 {
3. **public** **static** **void** main(String[] args) {
4. LocalTime time1 = LocalTime.of(10,43,12);
5. System.out.println(time1);
6. LocalTime time2=time1.plusHours(4);
7. LocalTime time3=time2.plusMinutes(18);
8. System.out.println(time3);
9. }
10. }

Output:

10:43:12

15:01:12

## Java LocalTime Example

1. **import** java.time.\*;
2. **import** java.time.temporal.ChronoUnit;
3. **public** **class** LocalTimeExample5 {
4. **public** **static** **void** main(String... args) {
5. ZoneId zone1 = ZoneId.of("Asia/Kolkata");
6. ZoneId zone2 = ZoneId.of("Asia/Tokyo");
7. LocalTime time1 = LocalTime.now(zone1);
8. System.out.println("India Time Zone: "+time1);
9. LocalTime time2 = LocalTime.now(zone2);
10. System.out.println("Japan Time Zone: "+time2);
11. **long** hours = ChronoUnit.HOURS.between(time1, time2);
12. System.out.println("Hours between two Time Zone: "+hours);
13. **long** minutes = ChronoUnit.MINUTES.between(time1, time2);
14. System.out.println("Minutes between two time zone: "+minutes);
15. }
16. }

Output:

India Time Zone: 14:56:43.087

Japan Time Zone: 18:26:43.103

Hours between two Time Zone: 3

Minutes between two time zone: 210

# Java LocalDateTime class

Java LocalDateTime class is an immutable date-time object that represents a date-time, with the default format as yyyy-MM-dd-HH-mm-ss.zzz. It inherits object class and implements the ChronoLocalDateTime interface.

## Java LocalDateTime class declaration

Let's see the declaration of java.time.LocalDateTime class.

1. **public** **final** **class** LocalDateTime **extends** Object
2. **implements** Temporal, TemporalAdjuster, ChronoLocalDateTime<LocalDate>, Serializable

### Methods of Java LocalDateTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| String format(DateTimeFormatter formatter) | It is used to format this date-time using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this date-time as an int. |
| LocalDateTime minusDays(long days) | It is used to return a copy of this LocalDateTime with the specified number of days subtracted. |
| static LocalDateTime now() | It is used to obtain the current date-time from the system clock in the default time-zone. |
| static LocalDateTime of(LocalDate date, LocalTime time) | It is used to obtain an instance of LocalDateTime from a date and time. |
| LocalDateTime plusDays(long days) | It is used to return a copy of this LocalDateTime with the specified number of days added. |
| boolean equals(Object obj) | It is used to check if this date-time is equal to another date-time. |

## Java LocalDateTime Example

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample1 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime now = LocalDateTime.now();
6. System.out.println("Before Formatting: " + now);
7. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
8. String formatDateTime = now.format(format);
9. System.out.println("After Formatting: " + formatDateTime);
10. }
11. }

Output:

Before Formatting: 2017-01-13T17:09:42.411

After Formatting: 13-01-2017 17:09:42

## Java LocalDateTime Example: now()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample2 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.now();
6. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
7. String formatDateTime = datetime1.format(format);
8. System.out.println(formatDateTime);
9. }
10. }

Output:

14-01-2017 11:42:32

## Java LocalDateTime Example: get()

1. **import** java.time.LocalDateTime;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** LocalDateTimeExample3 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime a = LocalDateTime.of(2017, 2, 13, 15, 56);
6. System.out.println(a.get(ChronoField.DAY\_OF\_WEEK));
7. System.out.println(a.get(ChronoField.DAY\_OF\_YEAR));
8. System.out.println(a.get(ChronoField.DAY\_OF\_MONTH));
9. System.out.println(a.get(ChronoField.HOUR\_OF\_DAY));
10. System.out.println(a.get(ChronoField.MINUTE\_OF\_DAY));
11. }
12. }

Output:

1

44

13

15

956

## Java LocalDateTime Example: minusDays()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample4 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.of(2017, 1, 14, 10, 34);
6. LocalDateTime datetime2 = datetime1.minusDays(100);
7. System.out.println("Before Formatting: " + datetime2);
8. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
9. String formatDateTime = datetime2.format(format);
10. System.out.println("After Formatting: " + formatDateTime );
11. }
12. }

Output:

Before Formatting: 2016-10-06T10:34

After Formatting: 06-10-2016 10:34

## Java LocalDateTime Example: plusDays()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample5 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.of(2017, 1, 14, 10, 34);
6. LocalDateTime datetime2 = datetime1.plusDays(120);
7. System.out.println("Before Formatting: " + datetime2);
8. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
9. String formatDateTime = datetime2.format(format);
10. System.out.println("After Formatting: " + formatDateTime );
11. }
12. }

Output:

Before Formatting: 2017-05-14T10:34

After Formatting: 14-05-2017 10:34

# Java LocalDateTime class

Java LocalDateTime class is an immutable date-time object that represents a date-time, with the default format as yyyy-MM-dd-HH-mm-ss.zzz. It inherits object class and implements the ChronoLocalDateTime interface.

## Java LocalDateTime class declaration

Let's see the declaration of java.time.LocalDateTime class.

1. **public** **final** **class** LocalDateTime **extends** Object
2. **implements** Temporal, TemporalAdjuster, ChronoLocalDateTime<LocalDate>, Serializable

### Methods of Java LocalDateTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| String format(DateTimeFormatter formatter) | It is used to format this date-time using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this date-time as an int. |
| LocalDateTime minusDays(long days) | It is used to return a copy of this LocalDateTime with the specified number of days subtracted. |
| static LocalDateTime now() | It is used to obtain the current date-time from the system clock in the default time-zone. |
| static LocalDateTime of(LocalDate date, LocalTime time) | It is used to obtain an instance of LocalDateTime from a date and time. |
| LocalDateTime plusDays(long days) | It is used to return a copy of this LocalDateTime with the specified number of days added. |
| boolean equals(Object obj) | It is used to check if this date-time is equal to another date-time. |

## Java LocalDateTime Example

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample1 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime now = LocalDateTime.now();
6. System.out.println("Before Formatting: " + now);
7. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
8. String formatDateTime = now.format(format);
9. System.out.println("After Formatting: " + formatDateTime);
10. }
11. }

Output:

Before Formatting: 2017-01-13T17:09:42.411

After Formatting: 13-01-2017 17:09:42

## Java LocalDateTime Example: now()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample2 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.now();
6. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");
7. String formatDateTime = datetime1.format(format);
8. System.out.println(formatDateTime);
9. }
10. }

Output:

14-01-2017 11:42:32

## Java LocalDateTime Example: get()

1. **import** java.time.LocalDateTime;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** LocalDateTimeExample3 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime a = LocalDateTime.of(2017, 2, 13, 15, 56);
6. System.out.println(a.get(ChronoField.DAY\_OF\_WEEK));
7. System.out.println(a.get(ChronoField.DAY\_OF\_YEAR));
8. System.out.println(a.get(ChronoField.DAY\_OF\_MONTH));
9. System.out.println(a.get(ChronoField.HOUR\_OF\_DAY));
10. System.out.println(a.get(ChronoField.MINUTE\_OF\_DAY));
11. }
12. }

Output:

1

44

13

15

956

## Java LocalDateTime Example: minusDays()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample4 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.of(2017, 1, 14, 10, 34);
6. LocalDateTime datetime2 = datetime1.minusDays(100);
7. System.out.println("Before Formatting: " + datetime2);
8. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
9. String formatDateTime = datetime2.format(format);
10. System.out.println("After Formatting: " + formatDateTime );
11. }
12. }

Output:

Before Formatting: 2016-10-06T10:34

After Formatting: 06-10-2016 10:34

## Java LocalDateTime Example: plusDays()

1. **import** java.time.LocalDateTime;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** LocalDateTimeExample5 {
4. **public** **static** **void** main(String[] args) {
5. LocalDateTime datetime1 = LocalDateTime.of(2017, 1, 14, 10, 34);
6. LocalDateTime datetime2 = datetime1.plusDays(120);
7. System.out.println("Before Formatting: " + datetime2);
8. DateTimeFormatter format = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
9. String formatDateTime = datetime2.format(format);
10. System.out.println("After Formatting: " + formatDateTime );
11. }
12. }

Output:

Before Formatting: 2017-05-14T10:34

After Formatting: 14-05-2017 10:34

# Java OffsetTime class

Java OffsetTime class is an immutable date-time object that represents a time, often viewed as hour-minute-second offset. It inherits Object class and implements the Comparable interface.

## Java OffsetTime class declaration

Let's see the declaration of java.time.OffsetTime class.

1. **public** **final** **class** OffsetTime **extends** Object
2. **implements** Temporal, TemporalAdjuster, Comparable<OffsetTime>, Serializable

### Methods of Java OffsetTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| String format(DateTimeFormatter formatter) | It is used to format this time using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this time as an int. |
| int getHour() | It is used to get the hour-of-day field. |
| int getMinute() | It is used to get the minute-of-hour field. |
| int getSecond() | It is used to get the second-of-minute field. |
| static OffsetTime now() | It is used to obtain the current time from the system clock in the default time-zone. |
| static OffsetTime of(LocalTime time, ZoneOffset offset) | It is used to obtain an instance of OffsetTime from a local time and an offset. |
| ValueRange range(TemporalField field) | It is used to get the range of valid values for the specified field. |
| VLocalTime toLocalTime() | It is used to get the LocalTime part of this date-time. |

## Java OffsetTime class Example

1. **import** java.time.OffsetTime;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** OffsetTimeExample1 {
4. **public** **static** **void** main(String[] args) {
5. OffsetTime offset = OffsetTime.now();
6. **int** h = offset.get(ChronoField.HOUR\_OF\_DAY);
7. System.out.println(h);
8. **int** m = offset.get(ChronoField.MINUTE\_OF\_DAY);
9. System.out.println(m);
10. **int** s = offset.get(ChronoField.SECOND\_OF\_DAY);
11. System.out.println(s);
12. }
13. }

Output:

16

970

58224

## Java OffsetTime class Example: getHour()

1. **import** java.time.OffsetTime;
2. **public** **class** OffsetTimeExample2 {
3. **public** **static** **void** main(String[] args) {
4. OffsetTime offset = OffsetTime.now();
5. **int** h = offset.getHour();
6. System.out.println(h+ " hour");
7. }
8. }

Output:

15 hour

## Java OffsetTime class Example: getMinute()

1. **import** java.time.OffsetTime;
2. **public** **class** OffsetTimeExample3 {
3. **public** **static** **void** main(String[] args) {
4. OffsetTime offset = OffsetTime.now();
5. **int** m = offset.getMinute();
6. System.out.println(m+ " minute");
7. }
8. }

Output:

24 minute

## Java OffsetTime class Example: getSecond()

1. **import** java.time.OffsetTime;
2. **public** **class** OffsetTimeExample4 {
3. **public** **static** **void** main(String[] args) {
4. OffsetTime offset = OffsetTime.now();
5. **int** s = offset.getSecond();
6. System.out.println(s+ " second");
7. }
8. }

Output:

8 second

# Java OffsetDateTime class

Java OffsetDateTime class is an immutable representation of a date-time with an offset. It inherits Object class and implements the Comparable interface.

OffsetDateTime class is used to store the date and time fields, to a precision of nanoseconds.

## Java OffsetDateTime class declaration

Let's see the declaration of java.time.OffsetDateTime class.

1. **public** **final** **class** OffsetDateTime **extends** Object
2. **implements** Temporal, TemporalAdjuster, Comparable<OffsetDateTime>, Serializable

### Methods of Java OffsetDateTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| int get(TemporalField field) | It is used to get the value of the specified field from this date-time as an int. |
| int getDayOfMonth() | It is used to get the day-of-month field. |
| iint getDayOfYear() | It is used to get the day-of-year field. |
| DayOfWeek getDayOfWeek() | It is used to get the day-of-week field, which is an enum DayOfWeek. |
| OffsetDateTime minusDays(long days) | It is used to return a copy of this OffsetDateTime with the specified number of days subtracted. |
| static OffsetDateTime now() | It is used to obtain the current date-time from the system clock in the default time-zone. |
| OffsetDateTime plusDays(long days) | It is used to return a copy of this OffsetDateTime with the specified number of days added. |
| LocalDate toLocalDate() | It is used to get the LocalDate part of this date-time. |

## Java OffsetDateTime class Example: getDayOfMonth()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample1 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offsetDT = OffsetDateTime.now();
5. System.out.println(offsetDT.getDayOfMonth());
6. }
7. }

Output:

18

## Java OffsetDateTime class Example: getDayOfYear()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample2 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offsetDT = OffsetDateTime.now();
5. System.out.println(offsetDT.getDayOfYear());
6. }
7. }

Output:

18

## Java OffsetDateTime class Example: getDayOfWeek()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample3 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offsetDT = OffsetDateTime.now();
5. System.out.println(offsetDT.getDayOfWeek());
6. }
7. }

Output:

WEDNESDAY

## Java OffsetDateTime class Example: toLocalDate()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample4 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offsetDT = OffsetDateTime.now();
5. System.out.println(offsetDT.toLocalDate());
6. }
7. }

Output:

2017-01-18

## Java OffsetDateTime class Example: minusDays()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample5 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offset = OffsetDateTime.now();
5. OffsetDateTime value =  offset.minusDays(240);
6. System.out.println(value);
7. }
8. }

Output:

2016-05-23T12:12:31.642+05:30

## Java OffsetDateTime class Example: plusDays()

1. **import** java.time.OffsetDateTime;
2. **public** **class** OffsetDateTimeExample6 {
3. **public** **static** **void** main(String[] args) {
4. OffsetDateTime offset = OffsetDateTime.now();
5. OffsetDateTime value =  offset.plusDays(240);
6. System.out.println(value);
7. }
8. }

Output:

2017-09-15T13:50:30.526+05:30

# Java Clock class

Java Clock class is used to provide an access to the current instant, date and time using a time zone. It inherits the Object class.

## Java Clock class declaration

Let's see the declaration of java.time.Clock class.

1. **public** **abstract** **class** Clock **extends** Object

### Methods of Java Clock

|  |  |
| --- | --- |
| **Method** | **Description** |
| abstract ZoneId getZone() | It is used to get the time-zone being used to create dates and times. |
| abstract Instant instant() | It is used to get the current instant of the clock. |
| static Clock offset(Clock baseClock, Duration offsetDuration) | It is used to obtain a clock that returns instants from the specified clock with the specified duration added |
| static Clock systemDefaultZone() | It is used to obtain a clock that returns the current instant using the best available system clock, converting to date and time using the default time-zone. |
| static Clock systemUTC() | It is used to obtain a clock that returns the current instant using the best available system clock, converting to date and time using the UTC time zone. |

## Java Clock class Example: getZone()

1. **import** java.time.Clock;
2. **public** **class** ClockExample1 {
3. **public** **static** **void** main(String[] args) {
4. Clock c = Clock.systemDefaultZone();
5. System.out.println(c.getZone());
6. }
7. }

Output:

Asia/Calcutta

## Java Clock class Example: instant()

1. **import** java.time.Clock;
2. **public** **class** ClockExample2 {
3. **public** **static** **void** main(String[] args) {
4. Clock c = Clock.systemUTC();
5. System.out.println(c.instant());
6. }
7. }

Output:

2017-01-14T07:11:07.748Z

## Java Clock class Example: systemUTC()

1. **import** java.time.Clock;
2. **public** **class** ClockExample3 {
3. **public** **static** **void** main(String[] args) {
4. Clock c = Clock.systemUTC();
5. System.out.println(c.instant());
6. }
7. }

Output:

2017-01-14T07:11:07.748Z

## Java Clock class Example: offset()

1. **import** java.time.Clock;
2. **import** java.time.Duration;
3. **public** **class** ClockExample4 {
4. **public** **static** **void** main(String[] args) {
5. Clock c = Clock.systemUTC();
6. Duration d = Duration.ofHours(5);
7. Clock clock = Clock.offset(c, d);
8. System.out.println(clock.instant());
9. }
10. }

Output:

2017-01-14T14:15:25.389Z

# Java ZonedDateTime class

Java ZonedDateTime class is an immutable representation of a date-time with a time-zone. It inherits Object class and implements the ChronoZonedDateTime interface.

ZonedDateTime class is used to store all date and time fields, to a precision of nanoseconds, and a time-zone with a zone offset used to handle ambiguous local date-times.

## Java ZonedDateTime class declaration

Let's see the declaration of java.time.ZonedDateTime class.

1. **public** **final** **class** ZonedDateTime **extends** Object
2. **implements** Temporal, ChronoZonedDateTime<LocalDate>, Serializable

### Methods of Java ZonedDateTime

|  |  |
| --- | --- |
| **Method** | **Description** |
| String format(DateTimeFormatter formatter) | It is used to format this date-time using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this date-time as an int. |
| ZoneId getZone() | It is used to get the time-zone, such as 'Asia/Kolkata'. |
| ZonedDateTime withZoneSameInstant(ZoneId zone) | It is used to return a copy of this date-time with a different time-zone, retaining the instant. |
| static ZonedDateTime now() | It is used to obtain the current date-time from the system clock in the default time-zone. |
| static ZonedDateTime of(LocalDate date, LocalTime time, ZoneId zone) | It is used to obtain an instance of ZonedDateTime from a local date and time. |
| ZonedDateTime minus(long amountToSubtract, TemporalUnit unit) | It is used to return a copy of this date-time with the specified amount subtracted. |
| ZonedDateTime plus(long amountToAdd, TemporalUnit unit) | It is used to return a copy of this date-time with the specified amount added. |

## Java ZonedDateTime class Example

1. **import** java.time.ZonedDateTime;
2. **public** **class** ZonedDateTimeExample1{
3. **public** **static** **void** main(String[] args) {
4. ZonedDateTime zone = ZonedDateTime.parse("2016-10-05T08:20:10+05:30[Asia/Kolkata]");
5. System.out.println(zone);
6. }
7. }

Output:

2016-10-05T08:20:10+05:30[Asia/Kolkata]

## Java ZonedDateTime class Example: of() and withZoneSameInstant()

1. **import** java.time.\*;
2. **public** **class** ZonedDateTimeExample2{
3. **public** **static** **void** main(String[] args) {
4. LocalDateTime  ldt = LocalDateTime.of(2017, Month.JANUARY,  19,   15,   26);
5. ZoneId  india = ZoneId.of("Asia/Kolkata");
6. ZonedDateTime zone1  = ZonedDateTime.of(ldt, india);
7. System.out.println("In India Central Time Zone: " + zone1);
8. ZoneId  tokyo = ZoneId.of("Asia/Tokyo");
9. ZonedDateTime zone2   = zone1.withZoneSameInstant(tokyo);
10. System.out.println("In Tokyo Central Time Zone:"  + zone2);
11. }
12. }

Output:

In India Central Time Zone: 2017-01-19T15:26+05:30[Asia/Kolkata]

In Tokyo Central Time Zone:2017-01-19T18:56+09:00[Asia/Tokyo]

## Java ZonedDateTime class Example: getZone()

1. **import** java.time.ZonedDateTime;
2. **public** **class** ZonedDateTimeExample3{
3. **public** **static** **void** main(String[] args) {
4. ZonedDateTime zone =ZonedDateTime.now();
5. System.out.println(zone.getZone());
6. }
7. }

Output:

Asia/Kolkata

## Java ZonedDateTime class Example: minus()

1. **import** java.time.Period;
2. **import** java.time.ZonedDateTime;
3. **public** **class** ZonedDateTimeExample4 {
4. **public** **static** **void** main(String[] args) {
5. ZonedDateTime zone= ZonedDateTime.now();
6. ZonedDateTime m = zone.minus(Period.ofDays(126));
7. System.out.println(m);
8. }
9. }

Output:

2016-09-15T12:54:01.354+05:30[Asia/Kolkata]

## Java ZonedDateTime class Example: plus()

1. **import** java.time.\*;
2. **public** **class** ZonedDateTimeExample5{
3. **public** **static** **void** main(String[] args) {
4. ZonedDateTime zone= ZonedDateTime.now();
5. ZonedDateTime p = zone.plus(Period.ofDays(126));
6. System.out.println(p);
7. }
8. }

Output:

2017-05-25T12:56:12.417+05:30[Asia/Kolkata]

# Java ZoneId class

Java ZoneId class specifies a time zone identifier and provides a rule for converting between an Instant and a LocalDateTime. It inherits Object class and implements the Serializable interface.

## Java ZoneId class declaration

Let's see the declaration of java.time.ZoneId class.

1. public abstract class ZoneId extends Object implements Serializable

### Methods of Java ZoneId

|  |  |
| --- | --- |
| **Method** | **Description** |
| String getDisplayName(TextStyle style, Locale locale) | It is used to get the textual representation of the zone, such as 'India Time' or '+05:30'. |
| abstract String getId() | It is used to get the unique time-zone ID. |
| static ZoneId of(String zoneId) | It is used to obtain an instance of ZoneId from an ID ensuring that the ID is valid and available for use. |
| static ZoneId systemDefault() | It is used to get the system default time-zone. |
| boolean equals(Object obj) | It is used to check if this time-zone ID is equal to another time-zone ID. |

## Java ZoneId class Example

1. **import** java.time.\*;
2. **public** **class** ZoneIdExample1 {
3. **public** **static** **void** main(String... args) {
4. ZoneId zoneid1 = ZoneId.of("Asia/Kolkata");
5. ZoneId zoneid2 = ZoneId.of("Asia/Tokyo");
6. LocalTime id1 = LocalTime.now(zoneid1);
7. LocalTime id2 = LocalTime.now(zoneid2);
8. System.out.println(id1);
9. System.out.println(id2);
10. System.out.println(id1.isBefore(id2));
11. }
12. }

Output:

14:28:58.230

17:58:58.230

true

## Java ZoneId class Example: systemDefault()

1. **import** java.time.ZoneId;
2. **public** **class** ZoneIdExample2 {
3. **public** **static** **void** main(String[] args) {
4. ZoneId zone = ZoneId.systemDefault();
5. System.out.println(zone);
6. }
7. }

Output:

Asia/Kolkata

## Java ZoneId class Example: getId()

1. **import** java.time.ZoneId;
2. **public** **class** ZoneIdExample3 {
3. **public** **static** **void** main(String[] args) {
4. ZoneId z = ZoneId.systemDefault();
5. String s = z.getId();
6. System.out.println(s);
7. }
8. }

Output:

Asia/Kolkata

## Java ZoneId class Example: getDisplayName()

1. **import** java.util.Locale;
2. **import** java.time.ZoneId;
3. **import** java.time.format.TextStyle;
4. **public** **class** ZoneIdExample4 {
5. **public** **static** **void** main(String[] args) {
6. ZoneId z = ZoneId.systemDefault();
7. System.out.println(z.getDisplayName(TextStyle.FULL, Locale.ROOT));
8. }
9. }

Output:

India Time

# Java ZoneOffset class

Java ZoneOffset class is used to represent the fixed zone offset from UTC time zone. It inherits the ZoneId class and implements the Comparable interface.

The ZoneOffset class declares three constants:

* **MAX:** It is the maximum supported zone offsets.
* **MIN:** It is the minimum supported zone offsets.
* **UTC:** It is the time zone offset constant for UTC.

## Java ZoneOffset class declaration

Let's see the declaration of java.time.ZoneOffset class.

1. **public** **final** **class** ZoneOffset **extends** ZoneId
2. **implements** TemporalAccessor, TemporalAdjuster, Comparable<ZoneOffset>, Serializable

### Methods of Java ZoneOffset

|  |  |
| --- | --- |
| **Method** | **Description** |
| Temporal adjustInto(Temporal temporal) | It is used to adjust the specified temporal object to have the same offset as this object. |
| int get(TemporalField field) | It is used to get the value of the specified field from this offset as an int. |
| boolean isSupported(TemporalField field) | It is used to check if the specified field is supported. |
| static ZoneOffset of(String offsetId) | It is used to obtain an instance of ZoneOffset using the ID. |
| static ZoneOffset ofHours(int hours) | It is used to obtain an instance of ZoneOffset using an offset in hours. |
| static ZoneOffset ofHoursMinutes(int hours, int minutes) | It is used to obtain an instance of ZoneOffset using an offset in hours and minutes. |
| ValueRange range(TemporalField field) | It is used to get the range of valid values |

## Java ZoneOffset Example

1. **import** java.time.\*;
2. **import** java.time.temporal.Temporal;
3. **public** **class** ZoneOffsetExample1 {
4. **public** **static** **void** main(String[] args) {
5. ZoneOffset zone = ZoneOffset.UTC;
6. Temporal temp = zone.adjustInto(ZonedDateTime.now());
7. System.out.println(temp);
8. }
9. }

Output:

2017-01-29T12:43:00.702+05:30[Asia/Calcutta]

## Java ZoneOffset Example: ofHours()

1. **import** java.time.ZoneOffset;
2. **public** **class** ZoneOffsetExample2 {
3. **public** **static** **void** main(String[] args) {
4. ZoneOffset zone = ZoneOffset.ofHours(5);
5. System.out.println(zone);
6. }
7. }

Output:

+05:00

## Java ZoneOffset Example: ofHoursMinutes()

1. **import** java.time.ZoneOffset;
2. **public** **class** ZoneOffsetExample3 {
3. **public** **static** **void** main(String[] args) {
4. ZoneOffset zone = ZoneOffset.ofHoursMinutes(5,30);
5. System.out.println(zone);
6. }
7. }

Output:

+05:30

## Java ZoneOffset Example: isSupported()

1. **import** java.time.ZoneOffset;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** ZoneOffsetExample4 {
4. **public** **static** **void** main(String[] args) {
5. ZoneOffset zone = ZoneOffset.UTC;
6. **boolean** b1 = zone.isSupported(ChronoField.OFFSET\_SECONDS);
7. System.out.println(b1);
8. **boolean** b2 = zone.isSupported(ChronoField.SECOND\_OF\_MINUTE);
9. System.out.println(b2);
10. }
11. }

Output:

true

false

# Java Year class

Java Year class is an immutable date-time object that represents a year. It inherits the Object class and implements the Comparable interface.

## Java Year class declaration

Let's see the declaration of java.time.Year class.

1. **public** **final** **class** Year **extends** Object **implements** Temporal, TemporalAdjuster, Comparable<Year>, Serializable

### Methods of Java Year

|  |  |
| --- | --- |
| **Method** | **Description** |
| LocalDate atDay(int dayOfYear) | It is used to combine this year with a day-of-year to create a LocalDate. |
| String format(DateTimeFormatter formatter) | It is used to format this year using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this year as an int. |
| boolean isAfter(Year other) | It is used to check if this year is after the specified year. |
| boolean isBefore(Year other) | It is used to check if this year is before the specified year. |
| boolean isLeap() | It is used to check if the year is a leap year, according to the ISO proleptic calendar system rules. |
| int length() | It is used to get the length of this year in days. |
| static Year now() | It is used to obtain the current year from the system clock in the default time-zone. |

## Java Year Example: now()

1. **import** java.time.Year;
2. **public** **class** YearExample1 {
3. **public** **static** **void** main(String[] args) {
4. Year y = Year.now();
5. System.out.println(y);
6. }
7. }

Output:

2017

## Java Year Example: atDay()

1. **import** java.time.LocalDate;
2. **import** java.time.Year;
3. **public** **class** YearExample2{
4. **public** **static** **void** main(String[] args) {
5. Year y = Year.of(2017);
6. LocalDate l = y.atDay(123);
7. System.out.println(l);
8. }
9. }

Output:

2017-05-03

## Java Year Example: length()

1. **import** java.time.Year;
2. **public** **class** YearExample3 {
3. **public** **static** **void** main(String[] args) {
4. Year year = Year.of(2017);
5. System.out.println(year.length());
6. }
7. }

Output:

365

## Java Year Example: isLeap()

1. **import** java.time.Year;
2. **public** **class** YearExample4 {
3. **public** **static** **void** main(String[] args) {
4. Year year = Year.of(2016);
5. System.out.println(year.isLeap());
6. }
7. }

Output:

true

# Java YearMonth class

Java YearMonth class is an immutable date-time object that represents the combination of a year and month. It inherits the Object class and implements the Comparable interface.

## Java YearMonth class declaration

Let's see the declaration of java.time.YearMonth class.

1. **public** **final** **class** YearMonth **extends** Object
2. **implements** Temporal, TemporalAdjuster, Comparable<YearMonth>, Serializable

### Methods of Java YearMonth

|  |  |
| --- | --- |
| **Method** | **Description** |
| Temporal adjustInto(Temporal temporal) | It is used to adjust the specified temporal object to have this year-month. |
| String format(DateTimeFormatter formatter) | It is used to format this year-month using the specified formatter. |
| int get(TemporalField field) | It is used to get the value of the specified field from this year-month as an int. |
| boolean isLeapYear() | It is used to check if the year is a leap year, according to the ISO proleptic calendar system rules. |
| static YearMonth now() | It is used to obtain the current year-month from the system clock in the default time zone. |
| static YearMonth of(int year, int month) | It is used to obtain an instance of YearMonth from a year and month. |
| YearMonth plus(TemporalAmount amountToAdd) | It is used to return a copy of this year-month with the specified amount added. |
| YearMonth minus (TemporalAmount amountToSubtract) | It is used to return a copy of this year-month with the specified amount subtracted. |

### Java YearMonth Example: now()

1. **import** java.time.YearMonth;
2. **public** **class** YearMonthExample1 {
3. **public** **static** **void** main(String[] args) {
4. YearMonth ym = YearMonth.now();
5. System.out.println(ym);
6. }
7. }

Output:

2017-01

### Java YearMonth Example: format()

1. **import** java.time.YearMonth;
2. **import** java.time.format.DateTimeFormatter;
3. **public** **class** YearMonthExample2 {
4. **public** **static** **void** main(String[] args) {
5. YearMonth ym = YearMonth.now();
6. String s = ym.format(DateTimeFormatter.ofPattern("MM yyyy"));
7. System.out.println(s);
8. }
9. }

Output:

01 2017

### Java YearMonth Example: get()

1. **import** java.time.YearMonth;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** YearMonthExample3 {
4. **public** **static** **void** main(String[] args) {
5. YearMonth y = YearMonth.now();
6. **long** l1 = y.get(ChronoField.YEAR);
7. System.out.println(l1);
8. **long** l2 = y.get(ChronoField.MONTH\_OF\_YEAR);
9. System.out.println(l2);
10. }
11. }

Output:

2017

1

### Java YearMonth Example: plus()

1. **import** java.time.\*;
2. **public** **class** YearMonthExample4 {
3. **public** **static** **void** main(String[] args) {
4. YearMonth ym1 = YearMonth.now();
5. YearMonth ym2 = ym1.plus(Period.ofYears(2));
6. System.out.println(ym2);
7. }
8. }

Output:

2019-01

### Java YearMonth Example: minus()

1. **import** java.time.\*;
2. **public** **class** YearMonthExample5 {
3. **public** **static** **void** main(String[] args) {
4. YearMonth ym1 = YearMonth.now();
5. YearMonth ym2 = ym1.minus(Period.ofYears(2));
6. System.out.println(ym2);
7. }
8. }

Output:

2015-01

# Java Period class

Java Period class is used to measures time in years, months and days. It inherits the Object class and implements the ChronoPeriod interface.

## Java Period class declaration

Let's see the declaration of java.time.Period class.

1. **public** **final** **class** Period **extends** Object **implements** ChronoPeriod, Serializable

### Methods of Java Period

|  |  |
| --- | --- |
| **Method** | **Description** |
| Temporal addTo(Temporal temporal) | It is used to add this period to the specified temporal object. |
| long get(TemporalUnit unit) | It is used to get the value of the requested unit. |
| int getYears() | It is used to get the amount of years of this period. |
| boolean isZero() | It is used to check if all three units of this period are zero. |
| Period minus(TemporalAmount amountToSubtract) | It is used to return a copy of this period with the specified period subtracted. |
| static Period of(int years, int months, int days) | It is used to obtain a Period representing a number of years, months and days. |
| Period plus(TemporalAmount amountToAdd) | It is used to return a copy of this period with the specified period added. |

### Java Period Example: addTo()

1. **import** java.time.\*;
2. **import** java.time.temporal.Temporal;
3. **public** **class** PeriodExample1 {
4. **public** **static** **void** main(String[] args) {
5. Period period = Period.ofDays(24);
6. Temporal temp = period.addTo(LocalDate.now());
7. System.out.println(temp);
8. }
9. }

Output:

2017-02-24

### Java Period Example: of()

1. **import** java.time.Period;
2. **public** **class** PeriodExample2 {
3. **public** **static** **void** main(String[] args) {
4. Period period = Period.of(2017,02,16);
5. System.out.println(period.toString());
6. }
7. }

Output:

P2017Y2M16D

### Java Period Example: minus()

1. **import** java.time.Period;
2. **public** **class** PeriodExample3 {
3. **public** **static** **void** main(String[] args) {
4. Period period1 = Period.ofMonths(4);
5. Period period2 = period1.minus(Period.ofMonths(2));
6. System.out.println(period2);
7. }
8. }

Output:

P2M

### Java Period Example: plus()

1. **import** java.time.Period;
2. **public** **class** PeriodExample4 {
3. **public** **static** **void** main(String[] args) {
4. Period period1 = Period.ofMonths(4);
5. Period period2 = period1.plus(Period.ofMonths(2));
6. System.out.println(period2);
7. }
8. }

Output:

P6M

# Java Duration class

Java Duration class is used to measures time in seconds and nanoseconds. It inherits the Object class and implements the Comparable interface.

## Java Duration class declaration

Let's see the declaration of java.time.Duration class.

1. **public** **final** **class** Duration **extends** Object
2. **implements** TemporalAmount, Comparable<Duration>, Serializable

### Methods of Java Duration

|  |  |
| --- | --- |
| **Method** | **Description** |
| Temporal addTo(Temporal temporal) | It is used to add this duration to the specified temporal object. |
| static Duration between(Temporal startInclusive, Temporal endExclusive) | It is used to obtain a Duration representing the duration between two temporal objects. |
| long get(TemporalUnit unit) | It is used to get the value of the requested unit. |
| boolean isNegative() | It is used to check if this duration is negative, excluding zero. |
| boolean isZero() | It is used to check if this duration is zero length. |
| Duration minus(Duration duration) | It is used to return a copy of this duration with the specified duration subtracted. |
| Duration plus(Duration duration) | It is used to return a copy of this duration with the specified duration added. |

### Java Duration Example: get()

1. **import** java.time.\*;
2. **import** java.time.temporal.ChronoUnit;
3. **public** **class** DurationExample1 {
4. **public** **static** **void** main(String[] args) {
5. Duration d = Duration.between(LocalTime.NOON,LocalTime.MAX);
6. System.out.println(d.get(ChronoUnit.SECONDS));
7. }
8. }

Output:

43199

### Java Duration Example: isNegative()

1. **import** java.time.\*;
2. **public** **class** DurationExample2 {
3. **public** **static** **void** main(String[] args) {
4. Duration d1 = Duration.between(LocalTime.MAX,LocalTime.NOON);
5. System.out.println(d1.isNegative());
6. Duration d2 = Duration.between(LocalTime.NOON,LocalTime.MAX);
7. System.out.println(d2.isNegative());
8. }
9. }

Output:

true

false

### Java Duration Example: between()

1. **import** java.time.\*;
2. **import** java.time.temporal.ChronoUnit;
3. **public** **class** DurationExample3 {
4. **public** **static** **void** main(String[] args) {
5. Duration d = Duration.between(LocalTime.NOON,LocalTime.MAX);
6. System.out.println(d.get(ChronoUnit.SECONDS));
7. }
8. }

Output:

43199

### Java Duration Example: minus()

1. **import** java.time.\*;
2. **public** **class** DurationExample4 {
3. **public** **static** **void** main(String[] args) {
4. Duration d1 = Duration.between(LocalTime.NOON,LocalTime.MAX);
5. System.out.println(d1.getSeconds());
6. Duration d2 = d1.minus(d1);
7. System.out.println(d2.getSeconds());
8. }
9. }

Output:

43199

0

### Java Duration Example: plus()

1. **import** java.time.\*;
2. **public** **class** DurationExample5 {
3. **public** **static** **void** main(String[] args) {
4. Duration d1 = Duration.between(LocalTime.NOON,LocalTime.MAX);
5. System.out.println(d1.getSeconds());
6. Duration d2 = d1.plus(d1);
7. System.out.println(d2.getSeconds());
8. }
9. }

Output:

43199

86399

# Java Instant class

Java Instant class is used to represent the specific moment on the time line. It inherits the Object class and implements the Comparable interface.

## Java Instant class declaration

Let's see the declaration of java.time.Instant class.

1. **public** **final** **class** Instant **extends** Object
2. **implements** Temporal, TemporalAdjuster, Comparable<Instant>, Serializable

### Methods of Java Instant

|  |  |
| --- | --- |
| **Method** | **Description** |
| Temporal adjustInto(Temporal temporal). | It is used to adjust the specified temporal object to have this instant. |
| int get(TemporalField field) | It is used to get the value of the specified field from this instant as an int. |
| boolean isSupported(TemporalField field) | It is used to check if the specified field is supported. |
| Instant minus(TemporalAmount amountToSubtract) | It is used to return a copy of this instant with the specified amount subtracted. |
| static Instant now() | It is used to obtain the current instant from the system clock. |
| static Instant parse(CharSequence text) | It is used to obtain an instance of Instant from a text string such as 2007-12-03T10:15:30.00Z. |
| Instant plus(TemporalAmount amountToAdd) | It is used to return a copy of this instant with the specified amount added. |
| Instant with(TemporalAdjuster adjuster) | It is used to return an adjusted copy of this instant. |

### Java Instant Example: parse()

1. **import** java.time.Instant;
2. **public** **class** InstantExample1 {
3. **public** **static** **void** main(String[] args) {
4. Instant inst = Instant.parse("2017-02-03T10:37:30.00Z");
5. System.out.println(inst);
6. }
7. }

Output:

2017-02-03T10:37:30Z

### Java Instant Example: now()

1. **import** java.time.Instant;
2. **public** **class** InstantExample2 {
3. **public** **static** **void** main(String[] args) {
4. Instant instant = Instant.now();
5. System.out.println(instant);
6. }
7. }

Output:

2017-02-03T06:11:01.194Z

### Java Instant Example: minus()

1. **import** java.time.\*;
2. **public** **class** InstantExample3 {
3. **public** **static** **void** main(String[] args) {
4. Instant instant = Instant.parse("2017-02-03T11:25:30.00Z");
5. instant = instant.minus(Duration.ofDays(125));
6. System.out.println(instant);
7. }
8. }

Output:

2016-10-01T11:25:30Z

### Java Instant Example: plus()

1. **import** java.time.\*;
2. **public** **class** InstantExample4 {
3. **public** **static** **void** main(String[] args) {
4. Instant inst1 = Instant.parse("2017-02-03T11:25:30.00Z");
5. Instant inst2 = inst1.plus(Duration.ofDays(125));
6. System.out.println(inst2);
7. }
8. }

Output:

2017-06-08T11:25:30Z

### Java Instant Example: isSupported()

1. **import** java.time.Instant;
2. **import** java.time.temporal.ChronoUnit;
3. **public** **class** InstantExample5 {
4. **public** **static** **void** main(String[] args) {
5. Instant inst = Instant.parse("2017-02-03T11:35:30.00Z");
6. System.out.println(inst.isSupported(ChronoUnit.DAYS));
7. System.out.println(inst.isSupported(ChronoUnit.YEARS));
8. }
9. }

Output:

true

false

# Java DayOfWeek enum

In Java the DayOfWeek is an enum representing the 7 days of the week. In addition with the textual enum name, every day-of-week has an int value.

## Java DayOfWeek enum declaration

Let's see the declaration of java.time.DayOfWeek.

1. **public** **enum** DayOfWeek **extends** Enum<DayOfWeek> **implements** TemporalAccessor, TemporalAdjuster

### Methods of Java DayOfWeek

|  |  |
| --- | --- |
| **Method** | **Description** |
| int get(TemporalField field) | It is used to get the value of the specified field from this day-of-week as an int. |
| boolean isSupported(TemporalField field) | It is used to check if the specified field is supported. |
| DayOfWeek minus(long days) | It is used to return the day-of-week that is the specified number of days before this one. |
| DayOfWeek plus(long days) | It is used to return the day-of-week that is the specified number of days after this one. |
| static DayOfWeek of(int dayOfWeek) | It is used to obtain an instance of DayOfWeek from an int value. |
| static DayOfWeek[] values() | It is used to return an array containing the constants of this enum type, in the order they are declared. |

### Java DayOfWeek Example: get()

1. **import** java.time.\*;
2. **import** java.time.temporal.ChronoField;
3. **public** **class** DayOfWeekExample1 {
4. **public** **static** **void** main(String[] args) {
5. LocalDate localDate = LocalDate.of(2017, Month.JANUARY, 25);
6. DayOfWeek dayOfWeek = DayOfWeek.from(localDate);
7. System.out.println(dayOfWeek.get(ChronoField.DAY\_OF\_WEEK));
8. }
9. }

Output:

3

### Java DayOfWeek Example: of()

1. **import** java.time.DayOfWeek;
2. **public** **class** DayOfWeekExample2 {
3. **public** **static** **void** main(String[] args) {
4. DayOfWeek day = DayOfWeek.of(5);
5. System.out.println(day.name());
6. System.out.println(day.ordinal());
7. System.out.println(day.getValue());
8. }
9. }

Output:

FRIDAY

4

5

### Java DayOfWeek Example: plus()

1. **import** java.time.\*;
2. **public** **class** DayOfWeekExample3 {
3. **public** **static** **void** main(String[] args) {
4. LocalDate date = LocalDate.of(2017, Month.JANUARY, 31);
5. DayOfWeek day = DayOfWeek.from(date);
6. System.out.println(day.getValue());
7. day = day.plus(3);
8. System.out.println(day.getValue());
9. }
10. }

Output:

2

5

### Java DayOfWeek Example: minus()

1. **import** java.time.\*;
2. **public** **class** DayOfWeekExample4 {
3. **public** **static** **void** main(String[] args) {
4. LocalDate date = LocalDate.of(2017, Month.JANUARY, 31);
5. DayOfWeek day = DayOfWeek.from(date);
6. System.out.println(day.getValue());
7. day = day.minus(3);
8. System.out.println(day.getValue());
9. }
10. }

Output:

2

6

# Java Month enum

In Java the Month is an enum representing the 12 months of the year. In addition with the textual enum name, every month-of-year has an int value.

## Java Month enum declaration

Let's see the declaration of java.time.Month.

1. **public** **enum** Month **extends** Enum<Month> **implements** TemporalAccessor, TemporalAdjuster

### Methods of Java Month

|  |  |
| --- | --- |
| **Method** | **Description** |
| int getValue() | It is used to get the month-of-year int value |
| int get(TemporalField field) | It is used to get the value of the specified field from this month-of-year as an int. |
| int length(boolean leapYear) | It is used to get the length of this month in days. |
| int maxLength() | It is used to get the maximum length of this month in days. |
| int minLength() | It is used to get the minimum length of this month in days. |
| Month minus(long months) | It is used to return the month-of-year that is the specified number of months before this one. |
| Month plus(long months) | It is used to return the month-of-year that is the specified number of quarters after this one. |
| static Month of(int month) | It is used to obtain an instance of Month from an int value. |

## Java Month enum Example

1. **import** java.time.\*;
2. **import** java.time.temporal.\*;
3. **public** **class** MonthEnumExample1 {
4. **public** **static** **void** main(String[] args) {
5. Month month = Month.valueOf("January".toUpperCase());
6. System.out.printf("For the month of %s all Sunday are:%n", month);
7. LocalDate localdate = Year.now().atMonth(month).atDay(1).
8. with(TemporalAdjusters.firstInMonth(DayOfWeek.SUNDAY));
9. Month mi = localdate.getMonth();
10. **while** (mi == month) {
11. System.out.printf("%s%n", localdate);
12. localdate = localdate.with(TemporalAdjusters.next(DayOfWeek.SUNDAY));
13. mi = localdate.getMonth();
14. }
15. }
16. }

Output:

For the month of JANUARY all Sunday are:

2017-01-01

2017-01-08

2017-01-15

2017-01-22

2017-01-29

## Java Month enum Example: getValue()

1. **import** java.time.\*;
2. **public** **class** MonthEnumExample2 {
3. **public** **static** **void** main(String[] args) {
4. Month month = Month.from(LocalDate.now());
5. System.out.println(month.getValue());
6. System.out.println(month.name());
7. }
8. }

Output:

1

JANUARY

## Java Month enum Example: minus()

1. **import** java.time.\*;
2. **public** **class** MonthEnumExample3 {
3. **public** **static** **void** main(String[] args) {
4. Month month = Month.from(LocalDate.now());
5. System.out.println(month.minus(2));
6. }
7. }

Output:

NOVEMBER

## Java Month enum Example: plus()

1. **import** java.time.\*;
2. **public** **class** MonthEnumExample4 {
3. **public** **static** **void** main(String[] args) {
4. Month month = Month.from(LocalDate.now());
5. System.out.println(month.plus(2));
6. }
7. }

Output:

MARCH

## Java Month enum Example: length()

1. **import** java.time.\*;
2. **public** **class** MonthEnumExample5 {
3. **public** **static** **void** main(String[] args) {
4. Month month = Month.from(LocalDate.now());
5. System.out.println("Total Number of days: "+month.length(**true**));
6. }
7. }

Output:

Total Number of days: 31