| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Date.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/Currency.html)   [**NEXT CLASS**](http://docs.google.com/java/util/Deque.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Date.html)    [**NO FRAMES**](http://docs.google.com/Date.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#lnxbz9) |

## **java.util**

Class Date

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 **java.util.Date**

**All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html), [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Date](http://docs.google.com/java/util/Date.html)> **Direct Known Subclasses:** [Date](http://docs.google.com/java/sql/Date.html), [Time](http://docs.google.com/java/sql/Time.html), [Timestamp](http://docs.google.com/java/sql/Timestamp.html)

public class **Date**extends [Object](http://docs.google.com/java/lang/Object.html)implements [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html), [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Date](http://docs.google.com/java/util/Date.html)>

The class Date represents a specific instant in time, with millisecond precision.

Prior to JDK 1.1, the class Date had two additional functions. It allowed the interpretation of dates as year, month, day, hour, minute, and second values. It also allowed the formatting and parsing of date strings. Unfortunately, the API for these functions was not amenable to internationalization. As of JDK 1.1, the Calendar class should be used to convert between dates and time fields and the DateFormat class should be used to format and parse date strings. The corresponding methods in Date are deprecated.

Although the Date class is intended to reflect coordinated universal time (UTC), it may not do so exactly, depending on the host environment of the Java Virtual Machine. Nearly all modern operating systems assume that 1 day = 24 × 60 × 60 = 86400 seconds in all cases. In UTC, however, about once every year or two there is an extra second, called a "leap second." The leap second is always added as the last second of the day, and always on December 31 or June 30. For example, the last minute of the year 1995 was 61 seconds long, thanks to an added leap second. Most computer clocks are not accurate enough to be able to reflect the leap-second distinction.

Some computer standards are defined in terms of Greenwich mean time (GMT), which is equivalent to universal time (UT). GMT is the "civil" name for the standard; UT is the "scientific" name for the same standard. The distinction between UTC and UT is that UTC is based on an atomic clock and UT is based on astronomical observations, which for all practical purposes is an invisibly fine hair to split. Because the earth's rotation is not uniform (it slows down and speeds up in complicated ways), UT does not always flow uniformly. Leap seconds are introduced as needed into UTC so as to keep UTC within 0.9 seconds of UT1, which is a version of UT with certain corrections applied. There are other time and date systems as well; for example, the time scale used by the satellite-based global positioning system (GPS) is synchronized to UTC but is *not* adjusted for leap seconds. An interesting source of further information is the U.S. Naval Observatory, particularly the Directorate of Time at:

<http://tycho.usno.navy.mil>

and their definitions of "Systems of Time" at:

<http://tycho.usno.navy.mil/systime.html>

In all methods of class Date that accept or return year, month, date, hours, minutes, and seconds values, the following representations are used:

* A year *y* is represented by the integer *y* - 1900.
* A month is represented by an integer from 0 to 11; 0 is January, 1 is February, and so forth; thus 11 is December.
* A date (day of month) is represented by an integer from 1 to 31 in the usual manner.
* An hour is represented by an integer from 0 to 23. Thus, the hour from midnight to 1 a.m. is hour 0, and the hour from noon to 1 p.m. is hour 12.
* A minute is represented by an integer from 0 to 59 in the usual manner.
* A second is represented by an integer from 0 to 61; the values 60 and 61 occur only for leap seconds and even then only in Java implementations that actually track leap seconds correctly. Because of the manner in which leap seconds are currently introduced, it is extremely unlikely that two leap seconds will occur in the same minute, but this specification follows the date and time conventions for ISO C.

In all cases, arguments given to methods for these purposes need not fall within the indicated ranges; for example, a date may be specified as January 32 and is interpreted as meaning February 1.

**Since:** JDK1.0 **See Also:**[DateFormat](http://docs.google.com/java/text/DateFormat.html), [Calendar](http://docs.google.com/java/util/Calendar.html), [TimeZone](http://docs.google.com/java/util/TimeZone.html), [Serialized Form](http://docs.google.com/serialized-form.html#java.util.Date)

| **Constructor Summary** | |
| --- | --- |
| [**Date**](http://docs.google.com/java/util/Date.html#Date())()            Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond. |
| [**Date**](http://docs.google.com/java/util/Date.html#Date(int,%20int,%20int))(int year, int month, int date)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date) or GregorianCalendar(year + 1900, month, date).* |
| [**Date**](http://docs.google.com/java/util/Date.html#Date(int,%20int,%20int,%20int,%20int))(int year, int month, int date, int hrs, int min)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min) or GregorianCalendar(year + 1900, month, date, hrs, min).* |
| [**Date**](http://docs.google.com/java/util/Date.html#Date(int,%20int,%20int,%20int,%20int,%20int))(int year, int month, int date, int hrs, int min, int sec)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min, sec) or GregorianCalendar(year + 1900, month, date, hrs, min, sec).* |
| [**Date**](http://docs.google.com/java/util/Date.html#Date(long))(long date)            Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT. |
| [**Date**](http://docs.google.com/java/util/Date.html#Date(java.lang.String))([String](http://docs.google.com/java/lang/String.html) s)  **Deprecated.** *As of JDK version 1.1, replaced by DateFormat.parse(String s).* |

| **Method Summary** | |
| --- | --- |
| boolean | [**after**](http://docs.google.com/java/util/Date.html#after(java.util.Date))([Date](http://docs.google.com/java/util/Date.html) when)            Tests if this date is after the specified date. |
| boolean | [**before**](http://docs.google.com/java/util/Date.html#before(java.util.Date))([Date](http://docs.google.com/java/util/Date.html) when)            Tests if this date is before the specified date. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**clone**](http://docs.google.com/java/util/Date.html#clone())()            Return a copy of this object. |
| int | [**compareTo**](http://docs.google.com/java/util/Date.html#compareTo(java.util.Date))([Date](http://docs.google.com/java/util/Date.html) anotherDate)            Compares two Dates for ordering. |
| boolean | [**equals**](http://docs.google.com/java/util/Date.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) obj)            Compares two dates for equality. |
| int | [**getDate**](http://docs.google.com/java/util/Date.html#getDate())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY\_OF\_MONTH).* |
| int | [**getDay**](http://docs.google.com/java/util/Date.html#getDay())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY\_OF\_WEEK).* |
| int | [**getHours**](http://docs.google.com/java/util/Date.html#getHours())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.HOUR\_OF\_DAY).* |
| int | [**getMinutes**](http://docs.google.com/java/util/Date.html#getMinutes())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.MINUTE).* |
| int | [**getMonth**](http://docs.google.com/java/util/Date.html#getMonth())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.MONTH).* |
| int | [**getSeconds**](http://docs.google.com/java/util/Date.html#getSeconds())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.SECOND).* |
| long | [**getTime**](http://docs.google.com/java/util/Date.html#getTime())()            Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object. |
| int | [**getTimezoneOffset**](http://docs.google.com/java/util/Date.html#getTimezoneOffset())()  **Deprecated.** *As of JDK version 1.1, replaced by -(Calendar.get(Calendar.ZONE\_OFFSET) + Calendar.get(Calendar.DST\_OFFSET)) / (60 \* 1000).* |
| int | [**getYear**](http://docs.google.com/java/util/Date.html#getYear())()  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.YEAR) - 1900.* |
| int | [**hashCode**](http://docs.google.com/java/util/Date.html#hashCode())()            Returns a hash code value for this object. |
| static long | [**parse**](http://docs.google.com/java/util/Date.html#parse(java.lang.String))([String](http://docs.google.com/java/lang/String.html) s)  **Deprecated.** *As of JDK version 1.1, replaced by DateFormat.parse(String s).* |
| void | [**setDate**](http://docs.google.com/java/util/Date.html#setDate(int))(int date)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.DAY\_OF\_MONTH, int date).* |
| void | [**setHours**](http://docs.google.com/java/util/Date.html#setHours(int))(int hours)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.HOUR\_OF\_DAY, int hours).* |
| void | [**setMinutes**](http://docs.google.com/java/util/Date.html#setMinutes(int))(int minutes)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.MINUTE, int minutes).* |
| void | [**setMonth**](http://docs.google.com/java/util/Date.html#setMonth(int))(int month)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.MONTH, int month).* |
| void | [**setSeconds**](http://docs.google.com/java/util/Date.html#setSeconds(int))(int seconds)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.SECOND, int seconds).* |
| void | [**setTime**](http://docs.google.com/java/util/Date.html#setTime(long))(long time)            Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT. |
| void | [**setYear**](http://docs.google.com/java/util/Date.html#setYear(int))(int year)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.YEAR, year + 1900).* |
| [String](http://docs.google.com/java/lang/String.html) | [**toGMTString**](http://docs.google.com/java/util/Date.html#toGMTString())()  **Deprecated.** *As of JDK version 1.1, replaced by DateFormat.format(Date date), using a GMT TimeZone.* |
| [String](http://docs.google.com/java/lang/String.html) | [**toLocaleString**](http://docs.google.com/java/util/Date.html#toLocaleString())()  **Deprecated.** *As of JDK version 1.1, replaced by DateFormat.format(Date date).* |
| [String](http://docs.google.com/java/lang/String.html) | [**toString**](http://docs.google.com/java/util/Date.html#toString())()            Converts this Date object to a String of the form: |
| static long | [**UTC**](http://docs.google.com/java/util/Date.html#UTC(int,%20int,%20int,%20int,%20int,%20int))(int year, int month, int date, int hrs, int min, int sec)  **Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min, sec) or GregorianCalendar(year + 1900, month, date, hrs, min, sec), using a UTC TimeZone, followed by Calendar.getTime().getTime().* |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Constructor Detail** |
| --- |

### Date

public **Date**()

Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.

**See Also:**[System.currentTimeMillis()](http://docs.google.com/java/lang/System.html#currentTimeMillis())

### Date

public **Date**(long date)

Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

**Parameters:**date - the milliseconds since January 1, 1970, 00:00:00 GMT.**See Also:**[System.currentTimeMillis()](http://docs.google.com/java/lang/System.html#currentTimeMillis())

### Date

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public **Date**(int year,  
 int month,  
 int date)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date) or GregorianCalendar(year + 1900, month, date).*

Allocates a Date object and initializes it so that it represents midnight, local time, at the beginning of the day specified by the year, month, and date arguments.

**Parameters:**year - the year minus 1900.month - the month between 0-11.date - the day of the month between 1-31.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### Date

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public **Date**(int year,  
 int month,  
 int date,  
 int hrs,  
 int min)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min) or GregorianCalendar(year + 1900, month, date, hrs, min).*

Allocates a Date object and initializes it so that it represents the instant at the start of the minute specified by the year, month, date, hrs, and min arguments, in the local time zone.

**Parameters:**year - the year minus 1900.month - the month between 0-11.date - the day of the month between 1-31.hrs - the hours between 0-23.min - the minutes between 0-59.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### Date

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public **Date**(int year,  
 int month,  
 int date,  
 int hrs,  
 int min,  
 int sec)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min, sec) or GregorianCalendar(year + 1900, month, date, hrs, min, sec).*

Allocates a Date object and initializes it so that it represents the instant at the start of the second specified by the year, month, date, hrs, min, and sec arguments, in the local time zone.

**Parameters:**year - the year minus 1900.month - the month between 0-11.date - the day of the month between 1-31.hrs - the hours between 0-23.min - the minutes between 0-59.sec - the seconds between 0-59.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### Date

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public **Date**([String](http://docs.google.com/java/lang/String.html) s)

**Deprecated.** *As of JDK version 1.1, replaced by DateFormat.parse(String s).*

Allocates a Date object and initializes it so that it represents the date and time indicated by the string s, which is interpreted as if by the [parse(java.lang.String)](http://docs.google.com/java/util/Date.html#parse(java.lang.String)) method.

**Parameters:**s - a string representation of the date.**See Also:**[DateFormat](http://docs.google.com/java/text/DateFormat.html), [parse(java.lang.String)](http://docs.google.com/java/util/Date.html#parse(java.lang.String))

| **Method Detail** |
| --- |

### clone

public [Object](http://docs.google.com/java/lang/Object.html) **clone**()

Return a copy of this object.

**Overrides:**[clone](http://docs.google.com/java/lang/Object.html#clone()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a clone of this instance.**See Also:**[Cloneable](http://docs.google.com/java/lang/Cloneable.html)

### UTC

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public static long **UTC**(int year,  
 int month,  
 int date,  
 int hrs,  
 int min,  
 int sec)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(year + 1900, month, date, hrs, min, sec) or GregorianCalendar(year + 1900, month, date, hrs, min, sec), using a UTC TimeZone, followed by Calendar.getTime().getTime().*

Determines the date and time based on the arguments. The arguments are interpreted as a year, month, day of the month, hour of the day, minute within the hour, and second within the minute, exactly as for the Date constructor with six arguments, except that the arguments are interpreted relative to UTC rather than to the local time zone. The time indicated is returned represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970).

**Parameters:**year - the year minus 1900.month - the month between 0-11.date - the day of the month between 1-31.hrs - the hours between 0-23.min - the minutes between 0-59.sec - the seconds between 0-59. **Returns:**the number of milliseconds since January 1, 1970, 00:00:00 GMT for the date and time specified by the arguments.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### parse

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public static long **parse**([String](http://docs.google.com/java/lang/String.html) s)

**Deprecated.** *As of JDK version 1.1, replaced by DateFormat.parse(String s).*

Attempts to interpret the string s as a representation of a date and time. If the attempt is successful, the time indicated is returned represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970). If the attempt fails, an IllegalArgumentException is thrown.

It accepts many syntaxes; in particular, it recognizes the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT". It also understands the continental U.S. time-zone abbreviations, but for general use, a time-zone offset should be used: "Sat, 12 Aug 1995 13:30:00 GMT+0430" (4 hours, 30 minutes west of the Greenwich meridian). If no time zone is specified, the local time zone is assumed. GMT and UTC are considered equivalent.

The string s is processed from left to right, looking for data of interest. Any material in s that is within the ASCII parenthesis characters ( and ) is ignored. Parentheses may be nested. Otherwise, the only characters permitted within s are these ASCII characters:

abcdefghijklmnopqrstuvwxyz  
 ABCDEFGHIJKLMNOPQRSTUVWXYZ  
 0123456789,+-:/

and whitespace characters.

A consecutive sequence of decimal digits is treated as a decimal number:

* If a number is preceded by + or - and a year has already been recognized, then the number is a time-zone offset. If the number is less than 24, it is an offset measured in hours. Otherwise, it is regarded as an offset in minutes, expressed in 24-hour time format without punctuation. A preceding - means a westward offset. Time zone offsets are always relative to UTC (Greenwich). Thus, for example, -5 occurring in the string would mean "five hours west of Greenwich" and +0430 would mean "four hours and thirty minutes east of Greenwich." It is permitted for the string to specify GMT, UT, or UTC redundantly-for example, GMT-5 or utc+0430.
* The number is regarded as a year number if one of the following conditions is true:
  + The number is equal to or greater than 70 and followed by a space, comma, slash, or end of string
  + The number is less than 70, and both a month and a day of the month have already been recognized

If the recognized year number is less than 100, it is interpreted as an abbreviated year relative to a century of which dates are within 80 years before and 19 years after the time when the Date class is initialized. After adjusting the year number, 1900 is subtracted from it. For example, if the current year is 1999 then years in the range 19 to 99 are assumed to mean 1919 to 1999, while years from 0 to 18 are assumed to mean 2000 to 2018. Note that this is slightly different from the interpretation of years less than 100 that is used in [SimpleDateFormat](http://docs.google.com/java/text/SimpleDateFormat.html).

* If the number is followed by a colon, it is regarded as an hour, unless an hour has already been recognized, in which case it is regarded as a minute.
* If the number is followed by a slash, it is regarded as a month (it is decreased by 1 to produce a number in the range 0 to 11), unless a month has already been recognized, in which case it is regarded as a day of the month.
* If the number is followed by whitespace, a comma, a hyphen, or end of string, then if an hour has been recognized but not a minute, it is regarded as a minute; otherwise, if a minute has been recognized but not a second, it is regarded as a second; otherwise, it is regarded as a day of the month.

A consecutive sequence of letters is regarded as a word and treated as follows:

* A word that matches AM, ignoring case, is ignored (but the parse fails if an hour has not been recognized or is less than 1 or greater than 12).
* A word that matches PM, ignoring case, adds 12 to the hour (but the parse fails if an hour has not been recognized or is less than 1 or greater than 12).
* Any word that matches any prefix of SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, or SATURDAY, ignoring case, is ignored. For example, sat, Friday, TUE, and Thurs are ignored.
* Otherwise, any word that matches any prefix of JANUARY, FEBRUARY, MARCH, APRIL, MAY, JUNE, JULY, AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, or DECEMBER, ignoring case, and considering them in the order given here, is recognized as specifying a month and is converted to a number (0 to 11). For example, aug, Sept, april, and NOV are recognized as months. So is Ma, which is recognized as MARCH, not MAY.
* Any word that matches GMT, UT, or UTC, ignoring case, is treated as referring to UTC.
* Any word that matches EST, CST, MST, or PST, ignoring case, is recognized as referring to the time zone in North America that is five, six, seven, or eight hours west of Greenwich, respectively. Any word that matches EDT, CDT, MDT, or PDT, ignoring case, is recognized as referring to the same time zone, respectively, during daylight saving time.

Once the entire string s has been scanned, it is converted to a time result in one of two ways. If a time zone or time-zone offset has been recognized, then the year, month, day of month, hour, minute, and second are interpreted in UTC and then the time-zone offset is applied. Otherwise, the year, month, day of month, hour, minute, and second are interpreted in the local time zone.

**Parameters:**s - a string to be parsed as a date. **Returns:**the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by the string argument.**See Also:**[DateFormat](http://docs.google.com/java/text/DateFormat.html)

### getYear

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getYear**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.YEAR) - 1900.*

Returns a value that is the result of subtracting 1900 from the year that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

**Returns:**the year represented by this date, minus 1900.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setYear

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setYear**(int year)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.YEAR, year + 1900).*

Sets the year of this Date object to be the specified value plus 1900. This Date object is modified so that it represents a point in time within the specified year, with the month, date, hour, minute, and second the same as before, as interpreted in the local time zone. (Of course, if the date was February 29, for example, and the year is set to a non-leap year, then the new date will be treated as if it were on March 1.)

**Parameters:**year - the year value.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getMonth

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getMonth**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.MONTH).*

Returns a number representing the month that contains or begins with the instant in time represented by this Date object. The value returned is between 0 and 11, with the value 0 representing January.

**Returns:**the month represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setMonth

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setMonth**(int month)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.MONTH, int month).*

Sets the month of this date to the specified value. This Date object is modified so that it represents a point in time within the specified month, with the year, date, hour, minute, and second the same as before, as interpreted in the local time zone. If the date was October 31, for example, and the month is set to June, then the new date will be treated as if it were on July 1, because June has only 30 days.

**Parameters:**month - the month value between 0-11.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getDate

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getDate**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY\_OF\_MONTH).*

Returns the day of the month represented by this Date object. The value returned is between 1 and 31 representing the day of the month that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

**Returns:**the day of the month represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setDate

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setDate**(int date)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.DAY\_OF\_MONTH, int date).*

Sets the day of the month of this Date object to the specified value. This Date object is modified so that it represents a point in time within the specified day of the month, with the year, month, hour, minute, and second the same as before, as interpreted in the local time zone. If the date was April 30, for example, and the date is set to 31, then it will be treated as if it were on May 1, because April has only 30 days.

**Parameters:**date - the day of the month value between 1-31.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getDay

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getDay**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.DAY\_OF\_WEEK).*

Returns the day of the week represented by this date. The returned value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

**Returns:**the day of the week represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getHours

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getHours**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.HOUR\_OF\_DAY).*

Returns the hour represented by this Date object. The returned value is a number (0 through 23) representing the hour within the day that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

**Returns:**the hour represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setHours

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setHours**(int hours)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.HOUR\_OF\_DAY, int hours).*

Sets the hour of this Date object to the specified value. This Date object is modified so that it represents a point in time within the specified hour of the day, with the year, month, date, minute, and second the same as before, as interpreted in the local time zone.

**Parameters:**hours - the hour value.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getMinutes

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getMinutes**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.MINUTE).*

Returns the number of minutes past the hour represented by this date, as interpreted in the local time zone. The value returned is between 0 and 59.

**Returns:**the number of minutes past the hour represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setMinutes

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setMinutes**(int minutes)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.MINUTE, int minutes).*

Sets the minutes of this Date object to the specified value. This Date object is modified so that it represents a point in time within the specified minute of the hour, with the year, month, date, hour, and second the same as before, as interpreted in the local time zone.

**Parameters:**minutes - the value of the minutes.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getSeconds

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getSeconds**()

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.get(Calendar.SECOND).*

Returns the number of seconds past the minute represented by this date. The value returned is between 0 and 61. The values 60 and 61 can only occur on those Java Virtual Machines that take leap seconds into account.

**Returns:**the number of seconds past the minute represented by this date.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### setSeconds

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public void **setSeconds**(int seconds)

**Deprecated.** *As of JDK version 1.1, replaced by Calendar.set(Calendar.SECOND, int seconds).*

Sets the seconds of this Date to the specified value. This Date object is modified so that it represents a point in time within the specified second of the minute, with the year, month, date, hour, and minute the same as before, as interpreted in the local time zone.

**Parameters:**seconds - the seconds value.**See Also:**[Calendar](http://docs.google.com/java/util/Calendar.html)

### getTime

public long **getTime**()

Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.

**Returns:**the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this date.

### setTime

public void **setTime**(long time)

Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT.

**Parameters:**time - the number of milliseconds.

### before

public boolean **before**([Date](http://docs.google.com/java/util/Date.html) when)

Tests if this date is before the specified date.

**Parameters:**when - a date. **Returns:**true if and only if the instant of time represented by this Date object is strictly earlier than the instant represented by when; false otherwise. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if when is null.

### after

public boolean **after**([Date](http://docs.google.com/java/util/Date.html) when)

Tests if this date is after the specified date.

**Parameters:**when - a date. **Returns:**true if and only if the instant represented by this Date object is strictly later than the instant represented by when; false otherwise. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if when is null.

### equals

public boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) obj)

Compares two dates for equality. The result is true if and only if the argument is not null and is a Date object that represents the same point in time, to the millisecond, as this object.

Thus, two Date objects are equal if and only if the getTime method returns the same long value for both.

**Overrides:**[equals](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)) in class [Object](http://docs.google.com/java/lang/Object.html) **Parameters:**obj - the object to compare with. **Returns:**true if the objects are the same; false otherwise.**See Also:**[getTime()](http://docs.google.com/java/util/Date.html#getTime())

### compareTo

public int **compareTo**([Date](http://docs.google.com/java/util/Date.html) anotherDate)

Compares two Dates for ordering.

**Specified by:**[compareTo](http://docs.google.com/java/lang/Comparable.html#compareTo(T)) in interface [Comparable](http://docs.google.com/java/lang/Comparable.html)<[Date](http://docs.google.com/java/util/Date.html)> **Parameters:**anotherDate - the Date to be compared. **Returns:**the value 0 if the argument Date is equal to this Date; a value less than 0 if this Date is before the Date argument; and a value greater than 0 if this Date is after the Date argument. **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if anotherDate is null.**Since:** 1.2

### hashCode

public int **hashCode**()

Returns a hash code value for this object. The result is the exclusive OR of the two halves of the primitive long value returned by the [getTime()](http://docs.google.com/java/util/Date.html#getTime()) method. That is, the hash code is the value of the expression:

(int)(this.getTime()^(this.getTime() >>> 32))

**Overrides:**[hashCode](http://docs.google.com/java/lang/Object.html#hashCode()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a hash code value for this object.**See Also:**[Object.equals(java.lang.Object)](http://docs.google.com/java/lang/Object.html#equals(java.lang.Object)), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### toString

public [String](http://docs.google.com/java/lang/String.html) **toString**()

Converts this Date object to a String of the form:

dow mon dd hh:mm:ss zzz yyyy

where:

* dow is the day of the week (Sun, Mon, Tue, Wed, Thu, Fri, Sat).
* mon is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).
* dd is the day of the month (01 through 31), as two decimal digits.
* hh is the hour of the day (00 through 23), as two decimal digits.
* mm is the minute within the hour (00 through 59), as two decimal digits.
* ss is the second within the minute (00 through 61, as two decimal digits.
* zzz is the time zone (and may reflect daylight saving time). Standard time zone abbreviations include those recognized by the method parse. If time zone information is not available, then zzz is empty - that is, it consists of no characters at all.
* yyyy is the year, as four decimal digits.

**Overrides:**[toString](http://docs.google.com/java/lang/Object.html#toString()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a string representation of this date.**See Also:**[toLocaleString()](http://docs.google.com/java/util/Date.html#toLocaleString()), [toGMTString()](http://docs.google.com/java/util/Date.html#toGMTString())

### toLocaleString

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public [String](http://docs.google.com/java/lang/String.html) **toLocaleString**()

**Deprecated.** *As of JDK version 1.1, replaced by DateFormat.format(Date date).*

Creates a string representation of this Date object in an implementation-dependent form. The intent is that the form should be familiar to the user of the Java application, wherever it may happen to be running. The intent is comparable to that of the "%c" format supported by the strftime() function of ISO C.

**Returns:**a string representation of this date, using the locale conventions.**See Also:**[DateFormat](http://docs.google.com/java/text/DateFormat.html), [toString()](http://docs.google.com/java/util/Date.html#toString()), [toGMTString()](http://docs.google.com/java/util/Date.html#toGMTString())

### toGMTString

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public [String](http://docs.google.com/java/lang/String.html) **toGMTString**()

**Deprecated.** *As of JDK version 1.1, replaced by DateFormat.format(Date date), using a GMT TimeZone.*

Creates a string representation of this Date object of the form: d mon yyyy hh:mm:ss GMT where:

* *d* is the day of the month (1 through 31), as one or two decimal digits.
* *mon* is the month (Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec).
* *yyyy* is the year, as four decimal digits.
* *hh* is the hour of the day (00 through 23), as two decimal digits.
* *mm* is the minute within the hour (00 through 59), as two decimal digits.
* *ss* is the second within the minute (00 through 61), as two decimal digits.
* *GMT* is exactly the ASCII letters "GMT" to indicate Greenwich Mean Time.

The result does not depend on the local time zone.

**Returns:**a string representation of this date, using the Internet GMT conventions.**See Also:**[DateFormat](http://docs.google.com/java/text/DateFormat.html), [toString()](http://docs.google.com/java/util/Date.html#toString()), [toLocaleString()](http://docs.google.com/java/util/Date.html#toLocaleString())

### getTimezoneOffset

[@Deprecated](http://docs.google.com/java/lang/Deprecated.html)  
public int **getTimezoneOffset**()

**Deprecated.** *As of JDK version 1.1, replaced by -(Calendar.get(Calendar.ZONE\_OFFSET) + Calendar.get(Calendar.DST\_OFFSET)) / (60 \* 1000).*

Returns the offset, measured in minutes, for the local time zone relative to UTC that is appropriate for the time represented by this Date object.

For example, in Massachusetts, five time zones west of Greenwich:

new Date(96, 1, 14).getTimezoneOffset() returns 300

because on February 14, 1996, standard time (Eastern Standard Time) is in use, which is offset five hours from UTC; but:

new Date(96, 5, 1).getTimezoneOffset() returns 240

because on June 1, 1996, daylight saving time (Eastern Daylight Time) is in use, which is offset only four hours from UTC.

This method produces the same result as if it computed:

(this.getTime() - UTC(this.getYear(),   
 this.getMonth(),   
 this.getDate(),  
 this.getHours(),   
 this.getMinutes(),   
 this.getSeconds())) / (60 \* 1000)

**Returns:**the time-zone offset, in minutes, for the current time zone.**See Also:**[Calendar.ZONE\_OFFSET](http://docs.google.com/java/util/Calendar.html#ZONE_OFFSET), [Calendar.DST\_OFFSET](http://docs.google.com/java/util/Calendar.html#DST_OFFSET), [TimeZone.getDefault()](http://docs.google.com/java/util/TimeZone.html#getDefault())

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/Date.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/Currency.html)   [**NEXT CLASS**](http://docs.google.com/java/util/Deque.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/Date.html)    [**NO FRAMES**](http://docs.google.com/Date.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#3dy6vkm) | [METHOD](#lnxbz9) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).