Class java.util.Date

```
java.lang.Object
   |
   +----java.util.Date
```

public class **Date**
extends Object

A wrapper for a date. This class lets you manipulate dates in a system independent way. To print today's date use:

```
        Date d = new Date();
        System.out.println("today = " + d);
```

To find out what day corresponds to a particular date:

```
        Date d = new Date(63, 0, 16);   // January 16, 1963
        System.out.println("Day of the week: " + d.getDay());
```

The date can be set and examined according to the local time zone into the year, month, day, hour, minute and second.

While the API is intended to reflect UTC, Coordinated Universal Time, it doesn't do so exactly. This inexact behavior is inherited from the time system of the underlying OS. All modern OS's that I (jag) am aware of assume that 1 day = 24*60*60 seconds. In UTC, about once a year there is an extra second, called a "leap second" added to a day to account for the wobble of the earth. Most computer clocks are not accurate enough to be able to reflect this distinction. Some computer standards are defined in GMT, which is equivalent to UT, Universal Time. 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 the first is based on an atomic clock and the second is based on astronomical observations, which for all practical purposes is an invisibly fine hair to split. An interesting source of further information is the US Naval Observatory, particularly the Directorate of Time and their definitions of Systems of Time.

Version:
    1.14, 28 Jul 1995
Author:
    James Gosling, Arthur van Hoff

Constructor Index

o **Date**()
    Creates today's date/time.
o **Date**(long)
    Creates a date.
o **Date**(int, int, int)
    Creates a date.
o **Date**(int, int, int, int, int)
    Creates a date.
o **Date**(int, int, int, int, int, int)
    Creates a date.
o **Date**(String)
    Creates a date from a string according to the syntax accepted by parse().

Method Index

o **UTC**(int, int, int, int, int, int)
    Calculates a UTC value from YMDHMS.
o **after**(Date)
    Checks whether this date comes after the specified date.
o **before**(Date)
    Checks whether this date comes before the specified date.
o **equals**(Object)
    Compares this object against the specified object.
o **getDate**()
    Returns the day of the month.
o **getDay**()
    Returns the day of the week.
o **getHours**()
    Returns the hour.
o **getMinutes**()
    Returns the minute.
o **getMonth**()
    Returns the month.
o **getSeconds**()
    Returns the second.
o **getTime**()
    Returns the time in milliseconds since the epoch.
o **getTimezoneOffset**()
    Return the time zone offset in minutes for the current locale that is appropriate for this time.
o **getYear**()
    Returns the year after 1900.
o **hashCode**()

Computes a hashCode.

o **parse**(String)
Given a string representing a time, parse it and return the time value.

o **setDate**(int)
Sets the date.

o **setDay**(int)
Sets the day of the week.

o **setHours**(int)
Sets the hours.

o **setMinutes**(int)
Sets the minutes.

o **setMonth**(int)
Sets the month.

o **setSeconds**(int)
Sets the seconds.

o **setTime**(long)
Sets the time.

o **setYear**(int)
Sets the year.

o **toGMTString**()
Converts a date to a String, using the Internet GMT conventions.

o **toLocaleString**()
Converts a date to a String, using the locale conventions.

o **toString**()
Converts a date to a String, using the UNIX ctime conventions.

Constructors

o **Date**

```
public Date()
```

Creates today’s date/time.

o **Date**

```
public Date(long date)
```

Creates a date. The fields are normalized before the Date object is created. The argument does not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

date – the value of the argument to be created

o **Date**

```
public Date(int year,
                 int month,
                 int date)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31

o **Date**

```
public Date(int year,
            int month,
            int date,
            int hrs,
            int min)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31
hrs – hours between 0–23
min – minutes between 0–59

o **Date**

```
public Date(int year,
            int month,
            int date,
            int hrs,
            int min,
            int sec)
```

Creates a date. The fields are normalized before the Date object is created. The arguments do not have to be in the correct range. For example, the 32nd of January is correctly interpreted as the 1st of February. You can use this to figure out what day a particular date falls on.

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31
hrs – hours between 0–23
min – minutes between 0–59
sec – seconds between 0–59

o Date

```
public Date(String s)
```

Creates a date from a string according to the syntax accepted by parse().

Methods

o UTC

```
public static long UTC(int year,
                       int month,
                       int date,
                       int hrs,
                       int min,
                       int sec)
```

Calculates a UTC value from YMDHMS. Interpretes the parameters in UTC, *not in the local time zone.*

Parameters:

year – a year after 1900
month – a month between 0–11
date – day of the month between 1–31
hrs – hours between 0–23
min – minutes between 0–59
sec – seconds between 0–59

o ***parse***

```
public static long parse(String s)
```

Given a string representing a time, parse it and return the time value. It accepts many syntaxes, but most importantly, in accepts the IETF standard date syntax: "Sat, 12 Aug 1995 13:30:00 GMT". It understands the continental US time zone abbreviations, but for general use, a timezone 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.

o getYear

```
public int getYear()
```

Returns the year after 1900.

o setYear

```
public void setYear(int year)
```

Sets the year.
Parameters:
year – the year value

o **getMonth**

```
public int getMonth()
```

Returns the month. This method assigns months with the values 0–11, with January beginning at value 0.

o **setMonth**

```
public void setMonth(int month)
```

Sets the month.
Parameters:
month – the month value (0–11)

o **getDate**

```
public int getDate()
```

Returns the day of the month. This method assigns days with the values of 1 to 31.

o **setDate**

```
public void setDate(int date)
```

Sets the date.
Parameters:
date – the day value

o **getDay**

```
public int getDay()
```

Returns the day of the week. This method assigns days of the week with the values 0–6, with 0 being Sunday.

o **setDay**

```
public void setDay(int day)
```

Sets the day of the week.
Parameters:
day – the value of the day if the week

o **getHours**

```
public int getHours()
```

Returns the hour. This method assigns the value of the hours of the day to range from 0 to 23, with midnight equal to 0.

o **setHours**

```
public void setHours(int hours)
```

Sets the hours.
Parameters:
hours – the hour value

o **getMinutes**

```
public int getMinutes()
```

Returns the minute. This method assigns the minutes of an hour to be any value from 0 to 59.

o **setMinutes**

```
public void setMinutes(int minutes)
```

Sets the minutes.
Parameters:
minutes – the value of the minutes

o **getSeconds**

```
public int getSeconds()
```

Returns the second. This method assigns the seconds of a minute to values of 0–59.

o **setSeconds**

```
public void setSeconds(int seconds)
```

Sets the seconds.
Parameters:
seconds – the second value

o **getTime**

```
public long getTime()
```

Returns the time in milliseconds since the epoch.

o **setTime**

```
public void setTime(long time)
```

Sets the time.
Parameters:
time – The new time value in milliseconds since the epoch.

o **before**

```
public boolean before(Date when)
```

Checks whether this date comes before the specified date.
Parameters:
when – the date to compare
Returns:
true if the original date comes before the specified one; false otherwise.

o **after**

```
public boolean after(Date when)
```

Checks whether this date comes after the specified date.
Parameters:
when – the date to compare
Returns:
true if the original date comes after the specified one; false otherwise.

o **equals**

```
public boolean equals(Object obj)
```

Compares this object against the specified object.
Parameters:
obj – the object to compare with
Returns:
true if the objects are the same; false otherwise.
Overrides:
equals in class Object

o **hashCode**

```
public int hashCode()
```

Computes a hashCode.
Overrides:
hashCode in class Object

o **toString**

```
public String toString()
```

Converts a date to a String, using the UNIX ctime conventions.
Overrides:
toString in class Object

o **toLocaleString**

```
public String toLocaleString()
```

Converts a date to a String, using the locale conventions.

o **toGMTString**

```
public String toGMTString()
```

Converts a date to a String, using the Internet GMT conventions.

o **getTimezoneOffset**

```
public int getTimezoneOffset()
```

Return the time zone offset in minutes for the current locale that is appropriate for this time. This value would be a constant except for daylight savings time.
