JAVA - DATE & TIME

http://www.tutorialspoint.com/java/java_date_time.htm Copyright © tutorialspoint.com

Java provides the **Date** class available in **java.util** package, this class encapsulates the current date and time.

The Date class supports two constructors as shown below.

SR.NO	Constructor and Description
1	**Date** This constructor initializes the object with the current date and time.
2	**Date***longmillisec* This constructor accepts an argument that equals the number of milliseconds that have elapsed since midnight, January 1, 1970

Below given are the methods of the date class.

SN	Methods with Description
1	**boolean after***Datedate* Returns true if the invoking Date object contains a date that is later than the one specified by date, otherwise, it returns false.
2	**boolean before***Datedate* Returns true if the invoking Date object contains a date that is earlier than the one specified by date, otherwise, it returns false.
3	**Object clone** Duplicates the invoking Date object.
4	**int compareTo***Datedate* Compares the value of the invoking object with that of date. Returns 0 if the values are equal. Returns a negative value if the invoking object is earlier than date. Returns a positive value if the invoking object is later than date.
5	**int compareTo***Objectobj* Operates identically to compareTo*Date* if obj is of class Date. Otherwise, it throws a ClassCastException.
6	**boolean equals***Objectdate* Returns true if the invoking Date object contains the same time and date as the one specified by date, otherwise, it returns false.
7	**long getTime**

Returns the number of milliseconds that have elapsed since January 1, 1970.

8 **int hashCode**

Returns a hash code for the invoking object.

9 **void setTime*longtime***

Sets the time and date as specified by time, which represents an elapsed time in milliseconds from midnight, January 1, 1970

10 **String toString**

Converts the invoking Date object into a string and returns the result.

Getting Current Date & Time

This is very easy to get current date and time in Java. You can use a simple Date object with *toString* method to print current date and time as follows:

```
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 using toString()
       System.out.println(date.toString());
   }
}
```

This would produce the following result:

```
on May 04 09:51:52 CDT 2009
```

Date Comparison:

There are following three ways to compare two dates:

- You can use getTime to obtain the number of milliseconds that have elapsed since midnight, January 1, 1970, for both objects and then compare these two values.
- You can use the methods before, after, and equals. Because the 12th of the month comes before the 18th, for example, new Date$99, 2, 12$.before$newDate(99, 2, 18)$ returns true.
- You can use the compareTo method, which is defined by the Comparable interface and implemented by Date.

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. For 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 would produce the following result:

```
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 $secondWed. inJuly$
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 given below. For 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 using toString()
     String str = String.format("Current Date/Time : %tc", date );

     System.out.printf(str);
  }
}
```

This would produce the following result:

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

This would produce the following result:

```
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. For 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 would produce the following result:

```
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 *withleadingzeroes*	2004
y	Last two digits of the year *withleadingzeroes*	04
C	First two digits of the year *withleadingzeroes*	20
B	Full month name	February
b	Abbreviated month name	Feb
m	Two-digit month *withleadingzeroes*	02
d	Two-digit day *withleadingzeroes*	03
e	Two-digit day *withoutleadingzeroes*	9
A	Full weekday name	Monday
a	Abbreviated weekday name	Mon
j	Three-digit day of year *withleadingzeroes*	069
H	Two-digit hour *withleadingzeroes*, between 00 and 23	18
k	Two-digit hour *withoutleadingzeroes*, between 0 and 23	18
I	Two-digit hour *withleadingzeroes*, between 01 and 12	06
l	Two-digit hour *withoutleadingzeroes*, between 1 and 12	6
M	Two-digit minutes *withleadingzeroes*	05
S	Two-digit seconds *withleadingzeroes*	19
L	Three-digit milliseconds *withleadingzeroes*	047
N	Nine-digit nanoseconds *withleadingzeroes*	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. For 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:

```
$ java DateDemo
1818-11-11 Parses as Wed Nov 11 00:00:00 GMT 1818
$ java DateDemo 2007-12-01
2007-12-01 Parses as Sat Dec 01 00:00:00 GMT 2007
```

Sleeping for a While:

You can sleep for any period of time from one millisecond up to the lifetime of your computer. For example, following program would sleep for 10 seconds:

```
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 would produce the following result:

```
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 above example once again:

```
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 would produce the following result:

```
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. I did not discuss Calendar class in this tutorial, you can look 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:

SN	Constructor with Description
1	**GregorianCalendar** Constructs a default GregorianCalendar using the current time in the default time zone with the default locale.
2	**GregorianCalendar***intyear, intmonth, intdate* Constructs a GregorianCalendar with the given date set in the default time zone with the default locale.
3	**GregorianCalendar***intyear, intmonth, intdate, inthour, intminute* Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.
4	**GregorianCalendar***intyear, intmonth, intdate, inthour, intminute, intsecond* Constructs a GregorianCalendar with the given date and time set for the default time zone with the default locale.

5	**GregorianCalendar*LocaleaLocale*** Constructs a GregorianCalendar based on the current time in the default time zone with the given locale.
6	**GregorianCalendar*TimeZonezone*** Constructs a GregorianCalendar based on the current time in the given time zone with the default locale.
7	**GregorianCalendar*TimeZonezone, LocaleaLocale*** 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:

SN	Methods with Description
1	**void add*intfield, intamount*** 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*Objectobj*** Compares this GregorianCalendar to an object reference.
5	**int get*intfield*** Gets the value for a given time field.
6	**int getActualMaximum*intfield*** Return the maximum value that this field could have, given the current date.
7	**int getActualMinimum*intfield*** Return the minimum value that this field could have, given the current date.
8	**int getGreatestMinimum*intfield*** Returns highest minimum value for the given field if varies.
9	**Date getGregorianChange** Gets the Gregorian Calendar change date.

10 **int getLeastMaximum***intfield*

Returns lowest maximum value for the given field if varies.

11 **int getMaximum***intfield*

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***intfield*

Returns minimum value for the given field.

16 **int hashCode**

Override hashCode.

17 **boolean isLeapYear***intyear*

Determines if the given year is a leap year.

18 **void roll***intfield, booleanup*

Adds or subtracts *up/down* a single unit of time on the given time field without changing larger fields.

19 **void set***intfield, intvalue*

Sets the time field with the given value.

20 **void set***intyear, intmonth, intdate*

Sets the values for the fields year, month, and date.

21 **void set***intyear, intmonth, intdate, inthour, intminute*

Sets the values for the fields year, month, date, hour, and minute.

22 **void set***intyear, intmonth, intdate, inthour, intminute, intsecond*

Sets the values for the fields year, month, date, hour, minute, and second.

23 **void setGregorianChange***Datedate*

Sets the GregorianCalendar change date.

24 **void setTime***Datedate*

Sets this Calendar's current time with the given Date.

25 **void setTimeInMillis***longmillis*

Sets this Calendar's current time from the given long value.

26 **void setTimeZone***TimeZonevalue*

Sets the time zone with the given time zone value.

27 **String toString**

Return 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 would produce the following result:

```
Date: Apr 22 2009
Time: 11:25:27
The current year is not a leap year
```

For a complete list of constant available in Calendar class, you can refer to standard Java documentation.

Processing math: 100%