**Java calendar class: add/subtract Year, months, days, hour, minutes**

[**JAVA DATE**](http://beginnersbook.com/category/technology/java-guide/java-date/)

Java’s Calendar class provides a set of methods for manipulation of temporal information. In addition to fetch the system’s current date and time, it also enables functionality for date and time arithmetic.

**Adding Time Period (Months and days) to a Date**

Suppose you want to add a time period to a date. How will you do so? Say, if time period is four months and the date to which time span needs to be added is today itself? For this, we will have to calculate the number of days the current month and in the intervening months. Also, you will have to take care of end of year and leap year to arrive at final result. Quite complex!!

Here is where **Calendar class** comes into picture and can provide you with an easy API to use to implement such a complex calculation with ease.

Let’s make the above problem a little more complex. Now, suppose we want to add four months and 5 days to 6th March 2009. So, how will the Calendar class help? Let’s see..

**Example:**

package calendarusage;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;

public class CalendarUsage {

/\*\*

\* main class

\* @param args

\*/

public static void main(String[] args) {

CalendarUsage calUsage = new CalendarUsage();

calUsage.performCalendarArithmetic();

}

/\*\*

\* This method performs Calendar Arithmetic by adding 4 months

\* and 5 days to 6th March 2009 and prints in two date formats

\*

\*/

private void performCalendarArithmetic() {

//Set calendar to 6th March 2009

Calendar cal = new GregorianCalendar(2009, Calendar.MARCH, 6);

//Add 4months and 5days

cal.add(Calendar.MONTH, 4);

cal.add(Calendar.DAY\_OF\_MONTH, 5);

//Define Format of date

SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy");

String date = sdf.format(cal.getTime());

System.out.println(date);

//Define another Format of date

sdf = new SimpleDateFormat("d MMM yyyy hh:mm aaa");

date = sdf.format(cal.getTime());

System.out.println(date);

}

}

In the method performCalendarArithmetic(), we have first set the date as6th March 2009 and then we have added 4 months to the months which will take the calendar to 6th July. To this date we have added 5 days. The result is 11th July 2009.

**The output of the above code is:**

11 Jul 2009

11 Jul 2009 12:00 AM

Another Case: Now, suppose we want to add 12 hours 34 minutes to 6th March 2009. We can do this as follows:

**Example:**

package calendarusage;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;

public class CalendarUsage {

/\*\*

\* main class

\* @param args

\*/

public static void main(String[] args) {

CalendarUsage calUsage = new CalendarUsage();

calUsage.addTime();

}

private void addTime() {

//set calendar to 6th March 2009

Calendar calendar = new GregorianCalendar(2009,Calendar.MARCH,6, 1,0);

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

SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy hh:mm aaa");

String date = sdf.format(calendar.getTime());

System.out.println(date);

// add 14h 55min

calendar.add(Calendar.HOUR,12);

calendar.add(Calendar.MINUTE,34);

System.out.println(date);

}

}

**The output is:**

Current Date::

6 Mar 2009 01:00 AM

6 Mar 2009 01:00 AM

**Subtracting Time Periods (Months, Days, hours, minutes) from Date**

Subtraction is same as addition. Only care that needs to be taken is that the value should be taken as negative. Suppose, we want to subtract 4 months,5 days,12 hours and 24 minutes. This can be done as follows:

**Example:**

package calendarusage;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;

public class CalendarUsage {

/\*\*

\* main class

\* @param args

\*/

public static void main(String[] args) {

CalendarUsage calUsage = new CalendarUsage();

calUsage.subtractTime();

}

private void subtractTime() {

//set calendar to 6th March 2009

Calendar calendar = new GregorianCalendar(2009, Calendar.MARCH, 6, 1, 0);

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

SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy hh:mm aaa");

String date = sdf.format(calendar.getTime());

System.out.println(date);

//Subtract 4 months,5 days,12 hours and 24 minutes

calendar.add(Calendar.MONTH, -4);

calendar.add(Calendar.DAY\_OF\_MONTH, -5);

calendar.add(Calendar.HOUR, -12);

calendar.add(Calendar.MINUTE, -24);

System.out.println("After subtracting 4 months,5 days,12 hours and 24 minutes::");

date = sdf.format(calendar.getTime());

System.out.println(date);

}

}

**The output is:**

Current Date::

6 Mar 2009 01:00 AM

After subtracting 4 months,5 days,12 hours and 24 minutes::

31 Oct 2008 12:36 PM

See the Calendar object automatically takes care of adjusting the year.

**Rolling**

In last example, we saw that Calendar object automatically took care of adjusting the year when an overflow occurred. However, there might be instances where we don’t want this to happen. In that case, Calendar class provides a method called roll().  
See, how it works:

**Example:**

package calendarusage;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;

public class CalendarUsage {

/\*\*

\* main class

\* @param args

\*/

public static void main(String[] args) {

CalendarUsage calUsage = new CalendarUsage();

calUsage.roll();

}

private void roll() {

//initialize calendar

Calendar calendar = new GregorianCalendar(2008, Calendar.DECEMBER,1);

System.out.println("Starting date is: ");

SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy");

String date = sdf.format(calendar.getTime());

System.out.println(date);

System.out.println("After rolling 1 month, ending date is: ");

calendar.roll(Calendar.MONTH, 1);

date = sdf.format(calendar.getTime());;

System.out.println(date);

}

}

**The output is:**

Starting date is:

1 Dec 2008

After rolling 1 month, ending date is:

1 Jan 2008

Notice that the year has not changed. Normally, if we have added a month to 1 Dec 2008, the resulting date should have been 1st Jan 2009. However, when we used roll() method, it only changed month without automatically adjusting the year.

**Date Comparison using compareTo() method**

Calendar class has a method called compareTo() which lets you compare two dates to find which one is earlier.

It returns:  
1. A value less than zero if the date and time of the input Calendar object is later than that of the calling Calendar object.  
2. A value greater than zero if the reverse is true  
3. A value of 0 if the two Calendar objects represent the same date

**Example:**

package calendarusage;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.GregorianCalendar;

public class CalendarUsage{

public static void main(String[] args)

{

// initialize two calendars

Calendar calendar1 = new GregorianCalendar(2007,Calendar.JANUARY,1,0,0,0);

Calendar calendar2 = new GregorianCalendar(2007,Calendar.JANUARY,1,0,1,0);

// define date format

String date1 = null;

String date2 = null;

SimpleDateFormat sdf = new SimpleDateFormat("d MMM yyyy hh:mm aaa");

// compare dates

if((calendar1.compareTo(calendar2)) < 0)

{

date1 = sdf.format(calendar1.getTime());

date2 = sdf.format(calendar2.getTime());

}

else

{

date1 = sdf.format(calendar2.getTime());

date2 = sdf.format(calendar1.getTime());

}

System.out.println(date1 + " occurs before " + date2);

System.out.println(date2 + " occurs after " + date1);

}

}

**Here’s the output:**

1 Jan 2007 12:00 AM occurs before 1 Jan 2007 12:01 AM

1 Jan 2007 12:01 AM occurs after 1 Jan 2007 12:00 AM