



Java Foundations

Working With Date/Time

Copyright

Introduction to Java Programming by Eric Matthews is licensed under a Creative Commons Attribution 3.0 Unported License. This license allows you to

- Copy, distribute and transmit the work
- Adapt the work
- Make commercial use of the work

Under the following conditions:

You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work

With the understanding that:

Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.

Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.

Other Rights — In no way are any of the following rights affected by the license:

- Your fair dealing or fair use rights, or other applicable copyright exceptions and limitations;
- The author's moral rights;
- Rights other persons may have either in the work itself or in how the work is used, such as publicity or privacy rights.

Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to one or more of the following web pages.

www.DeveloperGeekResources.com

www.EducationAnytime.com

Table of Contents

<i>A quick Blurb</i>	4
<i>DATE/TIME Stuff</i>	5
What is available	5
First Things First	5
Getting Current Date/Time	6
Setting Date/Time	6
Getting Granular (a journey into madness)	6
Back to Sanity? (some More methods).....	8
Reusing GregorianCalendar Object	8
Date Math	8
Formatting Dates	9
Time Zone Madness	10

A QUICK BLURB

The first place you are likely to search to find date/time methods is the date class. You will quickly discover that most of this class has been deprecated. This is Java's Orwellian speak to mean that this code has been superceded by other classes. While you may be tempted to use the deprecated methods of this class I would advise you not to. How long something that is deprecated is going to be supported is anyone's guess. It would really suck if you wrote a lot of code that was no longer supported in a future version. While in the realm of idealism the notion of obsolescence seems to fly in the face of the concept of a guaranteed method. Guaranteed means "as long as someone plans on supporting it".

My intent is to give you examples for some of the essentials in working with date and time. This is not by any means an exhaustive study of the subject. I have mentioned most of the major classes in the JDK for dealing with date and time so that you can do some more digging and hunting on your own.

DATE/TIME STUFF

WHAT IS AVAILABLE

The following is a list of the classes available for working with date and time.

java.util

- Date (mostly deprecated, you should avoid using deprecated methods in this class)
- Calendar
- GregorianCalendar
- TimeZone
- SimpleTimeZone

java.text

- DateFormat
- SimpleDateFormat
- DateFormatSymbols

FIRST THINGS FIRST

```
import java.util.GregorianCalendar;  
import java.util.Calendar;
```

We need to reference the class that we intend on using. We do this by using the import keyword. I have explicitly identified the classes we are using. We could have written the above two statements as...

```
import java.util.*;
```

While this is also acceptable, and the way you will see such references written it can be somewhat problematic. Packages tend to be fairly large. They contain many classes which in turn contain many different methods. With the last convention it makes it more difficult for someone that is unfamiliar with the package to quickly get to the documentation. The former method in my opinion should be the preferred method of doing business, though we do tend to be lazy.

Since GregorianCalendar is a subclass of Calendar we have access to any of the methods in Calendar as well when we instantiate an object of GregorianCalendar type.

```
GregorianCalendar now = new GregorianCalendar();
```

GregorianCalendar is overloaded and will accept any of the following arguments.

```
GregorianCalendar() //current date and time  
GregorianCalendar(int year, int month, int day)  
GregorianCalendar(int year, int month, int day, int hour, int minute)  
GregorianCalendar(int year, int month, int day, int hour, int minute, int second)
```

```
GregorianCalendar(Locale aLocale)
GregorianCalendar(TimeZone zone)
GregorianCalendar(TimeZone zone, Locale aLocale)
```

Getting Current Date/Time

```
GregorianCalendar now = new GregorianCalendar();
```

The above statement creates a new `GregorianCalendar` object. Since we are not passing any arguments we end up with the current date/time. To get the value we use the `getTime()` method, which is a method in the `Calendar` class.

```
System.out.println(dob.getTime());
```

Setting Date/Time

```
GregorianCalendar dob =
new GregorianCalendar(1958, Calendar.NOVEMBER, 23, 8, 32, 12);
```

GETTING GRANULAR (A JOURNEY INTO MADNESS)

Suppose we just want to return a portion of the date/time. The `Calendar` has a plethora of fields we can access. Of course we seem to have a slight predicament with using these based on how we instantiated our class. **Important Note:** We are about to embark on a wild and strange trip into the world of dates in Java. I decided to do it this way so that you could see first hand the daffiness that surrounds this subject. Most literature does not address this subject “head-on” which means that you get to experience the weirdness through the school of personal hard knocks. My goal is to save you the grief I experienced, or as my buddy Bill Clinton would say, “I feel your pain”.

Source: `Date1a.java`

The `Calendar` class has a ton of fields you can reference. Here are a few.

```
import java.util.GregorianCalendar;
import java.util.Calendar;

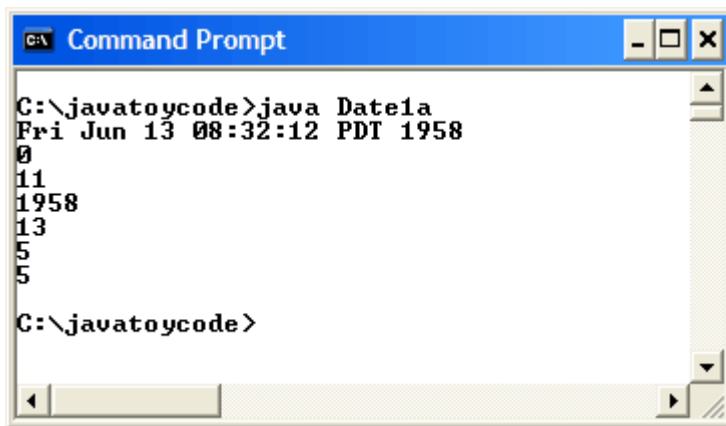
public class Date1a {
    public static void main(String[] args)
    {
        GregorianCalendar dob =
            new GregorianCalendar(1958, Calendar.JUNE, 13, 8, 32, 12);

        System.out.println(dob.getTime());
        //date start at 0 = Jan, not 1...brilliant???
        System.out.println(dob.getMinimum(Calendar.MONTH));
        System.out.println(dob.getMaximum(Calendar.MONTH));
        System.out.println(dob.get(Calendar.YEAR));
        System.out.println(dob.get(Calendar.DATE));
        System.out.println(dob.get(Calendar.MONTH));
    }
}
```

```

        System.out.println(Calendar.JUNE);
    }
}

```



```

C:\javatoycode>java Date1a
Fri Jun 13 08:32:12 PDT 1958
0
11
1958
13
5
5
C:\javatoycode>

```

All appears well until we see the output for MONTH. They decided to start January at 0, so the Month is off by one. Don't think for a minute that you can deal with this through code like...

```
dob.get(Calendar.MONTH+1) //danger, warning will robinson
```

...because this does not return what you would expect.

If you really want to get the month using the field you need to add the value of month to another integer and use that integer. For example:

Source: Date1a2.java

```

import java.util.GregorianCalendar;
import java.util.Calendar;

public class Date1a2 {
    public static void main(String[] args)
    {
        int month = 1;
        GregorianCalendar dob =
            new GregorianCalendar(1958, Calendar.JUNE, 13, 8, 32, 12);

        System.out.println(dob.get(Calendar.YEAR));
        System.out.println(dob.get(Calendar.DATE));
        //lame, lame, lame...
        month = month + dob.get(Calendar.MONTH);
        System.out.println(month);
    }
}

```

There are some interesting side effects we need to cover.

Source: Date1b.java

```
dob.roll(GregorianCalendar.MONTH, 1);
```

```
System.out.println(dob.get(Calendar.MONTH));
```

The roll() method (part of the Calendar class) can be used to change the default value of MONTH from 0 to 1. Doing this under this context is probably not a good idea. Here is why.

Source: Date1b2.java

```
System.out.println(dob.getTime());
dob.roll(dob.MONTH, 1);
System.out.println(dob.getTime()); //whoops, I am now JULY!!!
```

While roll() can be handy for calculating past and future dates it is not a good idea to use it in this context. While we are in the neighborhood, the roll() method will accept any field as its first argument, and the amount you want to “roll” (change).

Syntax:

```
roll(int field, int amount);
```

Example:

```
nextappt.roll(nextappt.DATE, 30);
```

BACK TO SANITY? (SOME MORE METHODS)

Reusing GregorianCalendar Object

It is possible to pass a new date into a GregorianCalendar object you have created.

Source: Date2.java

```
GregorianCalendar dob =
new GregorianCalendar(1958, Calendar.NOVEMBER, 23, 8, 32, 12);

System.out.println(dob.getTime());

dob.clear();
dob.set(1972, Calendar.APRIL, 1);
System.out.println(dob.getTime());
```

Date Math

Instead of using the roll() method, you can use the add() method to calculate date in the past or future. Here is an example of this.

Source: Date3a.java

```

next_appt.add(GregorianCalendar.YEAR, 1); // year from now
System.out.println(next_appt.getTime());

next_appt.add(GregorianCalendar.DATE, -10); // 10 days ago
System.out.println(next_appt.getTime());

next_appt.add(GregorianCalendar.MONTH, -3); // 3 months ago
System.out.println(next_appt.getTime());

```

FORMATTING DATES

The `DateFormat` and `SimpleDateFormat` classes are used to format dates. I am not going to go into much detail here, but will provide one example.

Source: `DateFormat.java`

```

import java.text.SimpleDateFormat;

. . .

GregorianCalendar dob =
new GregorianCalendar(1958, Calendar.NOVEMBER, 23, 8, 32, 12);
GregorianCalendar now = new GregorianCalendar();

SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yy kk:mm:ss.S");
System.out.println(sdf.format(dob.getTime()));

```

In this example I am creating an instance of the `SimpleDateFormat` class. This class has an overloaded constructor which can be set to a template of your choosing, or a default template (there are also some other choices as well). I have chosen to set up a template. Once I have done this I call the `format()` method passing in the results of the `getTime()` method from the `GregorianCalendar` class. Here is the table from the Javadoc for this class to save you the lookup.

Symbol	Meaning	Presentation	Example
G	Era designator	Text	AD
y	Year	Number	1996
M	Month in year	Text & Number	July & 07
d	Day in month	Number	10
h	Hour in am/pm (1~12)	Number	12
H	Hour in day (military) (0~23)	Number	0
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
E	Day in week	Text	Tuesday

D	Day in year	Number	190
F	Day of week in month	Number	2 (2 nd Wed in Jul)
w	Week in year	Number	27
W	Week in month	Number	2
a	am/pm marker	Text	PM
k	Hour in day (1~24)	Number	24
K	Hour in am/pm (0~11)	Number	0
z	Time zone	Text	Pacific Standard Time
'	Escape for text	Delimiter	
''	Escape for quote	Literal	'

TIME ZONE MADNESS

There is likely nothing more convoluted on this planet to express how screwed up humans are than time. Don't even get me started on this insane subject. I suppose much of my dismay at Java's dealing with date and time can really be attributed to just how screwed up time really is.

Source: TimeZone1.java

```
String[] tzones = TimeZone.getAvailableIDs();

for (int i=0; i<tzones.length; ++i)
{
    TimeZone tz = TimeZone.getTimeZone(tzones[i]);

    System.out.println(tz.getID() + " " + tz.getRawOffset());
}
```

In this example I am creating a String array and assigning it to the `getAvailableIDs()` method of the `TimeZone` class. I then loop through my array getting each time zone and printing it out. Also, for laughs, I decided to print the time zone offset.

Source: TimeZones.java

```
GregorianCalendar now = new GregorianCalendar();

for (int i=0; i<timezn.length; ++i)
{
    TimeZone tz = TimeZone.getTimeZone(timezn[i]);
    DateFormat dteformat = DateFormat.getDateTimeInstance();
    dteformat.setTimeZone(tz);

    System.out.println(timezn[i] + dteformat.format(now.getTime()));
}
```

This example prints out the current time for each time zone.