Working With Dates

[Posted By: Fadel]

The Date Class:

The Date class is available at java.util package, it works with milliseconds starting from January 1, 1970 00:00:00.000 GMT. as the Gregorian calendar specifies. In fact, most of the Date class constructors and methods are deprecated. The non deprecated constructors are Date() that returns the current system date and Date(long date) that returns a Date object based on the amount of milliseconds provided by the long value.On the other side, the non deprecated methods are as following:

`boolean after(Date when)`	Tests if this date is after the specified date.
`boolean before(Date when)`	Tests if this date is before the specified date.
`long getTime()`	Returns the number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this Date object.

```
public static void Method1(){
  Date A = new Date();
  Date B= new Date(A.getTime()+(5*24*60*60*1000));
  System.out.println(A);
  System.out.println(B);
  System.out.println("A is before B : "+A.before(B));
  System.out.println("A is after B : "+A.after(B));
}
```

The first line at the above method creates Date object A based on the current system time. where the number 1000 milliseconds * 60 seconds * 60 minutes * 24 hours is to calculate the amount of milliseconds per day, multiplying this value by 5 shows the amount of milliseconds for five days, and by adding this value to the previous day number of milliseconds [A.getTime()] will conclude the number of milliseconds of the fifth days after the current system date. Finally, feeding the long value to the date constructor of object B.

Formatting Date Values :

SimpleDateFormat is a concrete class for formatting Date objects. it is offered by java.text package that provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages. SimpleDateFormat is a direct child of java.text.DateFormat that is also a direct child of java.text.Format class, the following example illustrates the use of SimpleDateFormat Class.

```
public static void Method2(){
  SimpleDateFormat sdf;
  sdf=new SimpleDateFormat("yyyyMMddhhmmss");
  Date Today = new Date();
  String Mystr= sdf.format(Today);
  System.out.println(Mystr);
  sdf = new SimpleDateFormat("yyyy");
  int year= Integer.parseInt(sdf.format(Today));
  System.out.println("Next Year:"+(++year));
}
```

The possible pattern values to feed into the SimpleDateFormat Objects are as follows:

Letter	Date or Time Component
`G`	Era designator (AD;BC)
`y`	Year (2010; 10)
`M`	Month in year (0-11)(July; Jul; 07)
`w`	Week in year (1-52)
`W`	Week in month (1-6)
`D`	Day in year (1-365)
`d`	Day in month (1-31)
`F`	Day of week in month (In which week of the month the day is in)
`E`	Day in week (Tuesday; Tue)
`a`	Am/pm marker
`H`	Hour in day (0-23)
`k`	Hour in day (1-24)
`K`	Hour in am/pm (0-11)
`h`	Hour in am/pm (1-12)
`m`	Minutes
`s`	Seconds
`S`	Millisecond
`z`	General Time zone
`Z`	Time zone

A better way of using the SimpleDateFormat object is by declaring it as a static final object; for instance:

```
private static final SimpleDateFormat FORMAT1 = new SimpleDateFormat("dd-MMM-yyyy");
private static final SimpleDateFormat FORMAT2 = new SimpleDateFormat("EEE dd, MMM, yyyy");
private static final SimpleDateFormat FORMAT3 = new SimpleDateFormat("EEE, d MMM yyyy HH:mm:ss Z");
```

And then use the above static objects with your code as follows:

```
public static void Method3(){
  Date a = new Date();
  System.out.println(FORMAT1.format(a));
  System.out.println(FORMAT2.format(a));
  System.out.println(FORMAT3.format(a));
}
```

<u>Calendar Class & Date Arithmetic:</u>

[Java Documentation]The Calendar class is an abstract class that provides methods for converting between a specific instant in time and a set of calendar fields such as YEAR, MONTH, DAY_OF_MONTH, HOUR, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instant in time can be represented by a millisecond value that is an offset from the Epoch, January 1, 1970 00:00:00.000 GMT (Gregorian).

The class also provides additional fields and methods for implementing a concrete calendar system outside the package. Those fields and methods are defined as protected.

Like other locale-sensitive classes, Calendar provides a class method, getInstance, for getting a generally useful object of this type. Calendar's getInstance method returns a Calendar object whose calendar fields have been initialized with the current date and time:

```
    Calendar rightNow = Calendar.getInstance();
```

The Calendar getTime() method returns a Date object representing this Calendar's time value (millisecond offset from the Epoch").

```
private static final SimpleDateFormat FORMAT1 = new SimpleDateFormat("dd-MMM-yyyy");

public static void Method4(){
  Calendar aa = Calendar.getInstance();
  System.out.println(aa.getTime());
  System.out.print(aa.get(Calendar.YEAR)+"|");
  System.out.print(aa.get(Calendar.MONTH)+"|");
  System.out.print(aa.get(Calendar.DATE)+"|");
  System.out.println(aa.get(Calendar.HOUR)+"|");
  aa.set(Calendar.YEAR, 2012);
  aa.set(Calendar.DATE,1);
  aa.set(Calendar.MONTH,11);
  System.out.println(FORMAT1.format(aa.getTime()));
  aa.set(Calendar.MONTH,aa.get(Calendar.MONTH)+ 13);
  System.out.println(FORMAT1.format(aa.getTime()));
}
```

The last set() method in the above code will shift the date by a full year plus two more months. Therefore, a value within the month field greater than 11 will float over the year value. Same idea applies to day value, if it exceeds the current month last day value it will float over the month value where the month will be shifted one step and the remaining day value will be applied. In case, the day value exceeds 2 months - it will not assume all months ends with 31 days; in fact, it will calculate each month limit separately and what is left will be applied as day value.

The following example setting a zero and a negative day values

```
public static void Method5(){
  Calendar a = Calendar.getInstance();
  a.set(2010,11,1);
  System.out.println(FORMAT1.format(a.getTime()));
  a.set(Calendar.DATE,0);
  System.out.println(FORMAT1.format(a.getTime()));
  a.set(Calendar.DATE,-1);
  System.out.println(FORMAT1.format(a.getTime()));
}
```

```
Output:
  01-Dec-2010
  30-Nov-2010
  30-Oct-2010
```

Remember that the day value starts from 1 (not like months value which starts from 0), and by putting Zero you are shifting the date to the last day of the previous month. And that is why you get the 30-Nov-2010 as result. However, the being at Nov. and setting the day to be -1 i.e. two days before the first day of Nov. will get you back one more month plus two days.

The best way to go back one day each time is to get the current Date value, subtract it by One, and set it back again.

```
public static void Method6(){
  Calendar a = Calendar.getInstance();
  a.set(2010,11,1);
  System.out.println(FORMAT1.format(a.getTime()));
  a.set(Calendar.DATE,a.get(Calendar.DATE)-1);
  System.out.println(FORMAT1.format(a.getTime()));
  a.set(Calendar.DATE,a.get(Calendar.DATE)-1);
  System.out.println(FORMAT1.format(a.getTime()));
}
Output:
  01-Dec-2010
  30-Nov-2010
  29-Nov-2010
```