JAVA – DATE, TIME

DATE CLASS

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

The Date class supports two constructors. The first constructor initializes the object with the current date and time.

```
Date( )
```

The following constructor accepts one argument that equals the number of milliseconds that have elapsed since midnight, January 1, 1970

```
Date(long millisec)
```

Once you have a Date object available, you can call any of the following support methods to play with dates:

	Methods with Description
1	boolean **after**(Date date) 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**(Date date) 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**(Date date) 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**(Object obj) Operates identically to compareTo(Date) if obj is of class Date. Otherwise, it throws a ClassCastException.
6	boolean **equals**(Object date) 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**(long time) 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:

```
Mon May 11 09:51:52 CDT 2013
```

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(new Date (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 2013.11.10 at 08:52:01 AM CST
```

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 (second Wed. in July)
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, 2014
```

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, 2014
```

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 (with leading zeroes)	2004
y	Last two digits of the year (with leading zeroes)	04
C	First two digits of the year (with leading zeroes)	20
B	Full month name	February
b	Abbreviated month name	Feb
m	Two-digit month (with leading zeroes)	02
d	Two-digit day (with leading zeroes)	03
e	Two-digit day (without leading zeroes)	9
A	Full weekday name	Monday
a	Abbreviated weekday name	Mon

j	Three-digit day of year (with leading zeroes)	069
H	Two-digit hour (with leading zeroes), between 00 and 23	18
k	Two-digit hour (without leading zeroes), between 0 and 23	18
I	Two-digit hour (with leading zeroes), between 01 and 12	06
l	Two-digit hour (without leading zeroes), between 1 and 12	6
M	Two-digit minutes (with leading zeroes)	05
S	Two-digit seconds (with leading zeroes)	19
L	Three-digit milliseconds (with leading zeroes)	047
N	Nine-digit nanoseconds (with leading zeroes)	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

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 = "1818-11-11";
      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
```