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.

<table>
<thead>
<tr>
<th>SR.NO</th>
<th>Constructor and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Date</strong></td>
</tr>
<tr>
<td></td>
<td>This constructor initializes the object with the current date and time.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Date(longmillisec)</strong></td>
</tr>
<tr>
<td></td>
<td>This constructor accepts an argument that equals the number of milliseconds that have elapsed since midnight, January 1, 1970</td>
</tr>
</tbody>
</table>

Below given are the methods of the date class.

<table>
<thead>
<tr>
<th>SN</th>
<th>Methods with Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>boolean after(Datedate)</strong></td>
</tr>
<tr>
<td></td>
<td>Returns true if the invoking Date object contains a date that is later than the one specified by date, otherwise, it returns false.</td>
</tr>
<tr>
<td>2</td>
<td><strong>boolean before(Datedate)</strong></td>
</tr>
<tr>
<td></td>
<td>Returns true if the invoking Date object contains a date that is earlier than the one specified by date, otherwise, it returns false.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Object clone</strong></td>
</tr>
<tr>
<td></td>
<td>Duplicates the invoking Date object.</td>
</tr>
<tr>
<td>4</td>
<td><strong>int compareTo(Datedate)</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td><strong>int compareTo(Objectobj)</strong></td>
</tr>
<tr>
<td></td>
<td>Operates identically to compareTo(Date) if obj is of class Date. Otherwise, it throws a ClassCastException.</td>
</tr>
<tr>
<td>6</td>
<td><strong>boolean equals(Objectdate)</strong></td>
</tr>
<tr>
<td></td>
<td>Returns true if the invoking Date object contains the same time and date as the one specified by date, otherwise, it returns false.</td>
</tr>
<tr>
<td>7</td>
<td><strong>long getTime</strong></td>
</tr>
</tbody>
</table>
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**
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:

```java
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(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:

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

public class DateDemo {
    public static void main(String args[]) {
```
```java
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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era designator</td>
<td>AD</td>
</tr>
<tr>
<td>y</td>
<td>Year in four digits</td>
<td>2001</td>
</tr>
<tr>
<td>M</td>
<td>Month in year</td>
<td>July or 07</td>
</tr>
<tr>
<td>d</td>
<td>Day in month</td>
<td>10</td>
</tr>
<tr>
<td>h</td>
<td>Hour in A.M./P.M.</td>
<td>12</td>
</tr>
<tr>
<td>H</td>
<td>Hour in day</td>
<td>22</td>
</tr>
<tr>
<td>m</td>
<td>Minute in hour</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>Second in minute</td>
<td>55</td>
</tr>
<tr>
<td>S</td>
<td>Millisecond</td>
<td>234</td>
</tr>
<tr>
<td>E</td>
<td>Day in week</td>
<td>Tuesday</td>
</tr>
<tr>
<td>D</td>
<td>Day in year</td>
<td>360</td>
</tr>
<tr>
<td>F</td>
<td>Day of week in month</td>
<td>2 secondWed. inJuly</td>
</tr>
<tr>
<td>w</td>
<td>Week in year</td>
<td>40</td>
</tr>
<tr>
<td>W</td>
<td>Week in month</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>A.M./P.M. marker</td>
<td>PM</td>
</tr>
<tr>
<td>k</td>
<td>Hour in day</td>
<td>24</td>
</tr>
<tr>
<td>K</td>
<td>Hour in A.M./P.M.</td>
<td>10</td>
</tr>
<tr>
<td>z</td>
<td>Time zone</td>
<td>Eastern Standard Time</td>
</tr>
<tr>
<td>'</td>
<td>Escape for text</td>
<td>Delimiter</td>
</tr>
<tr>
<td>&quot;</td>
<td>Single quote</td>
<td>\</td>
</tr>
</tbody>
</table>

**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:

```java
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 formatted date
        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:
<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>Complete date and time</td>
<td>Mon May 04 09:51:52 CDT 2009</td>
</tr>
<tr>
<td>F</td>
<td>ISO 8601 date</td>
<td>2004-02-09</td>
</tr>
<tr>
<td>D</td>
<td>U.S. formatted date $month$/day/year</td>
<td>02/09/2004</td>
</tr>
<tr>
<td>T</td>
<td>24-hour time</td>
<td>18:05:19</td>
</tr>
<tr>
<td>R</td>
<td>24-hour time, no seconds</td>
<td>18:05</td>
</tr>
<tr>
<td>Y</td>
<td>Four-digit year withleadingzeros</td>
<td>2004</td>
</tr>
<tr>
<td>y</td>
<td>Last two digits of the year withleadingzeros</td>
<td>04</td>
</tr>
<tr>
<td>C</td>
<td>First two digits of the year withleadingzeros</td>
<td>20</td>
</tr>
<tr>
<td>B</td>
<td>Full month name</td>
<td>February</td>
</tr>
<tr>
<td>b</td>
<td>Abbreviated month name</td>
<td>Feb</td>
</tr>
<tr>
<td>m</td>
<td>Two-digit month withleadingzeros</td>
<td>02</td>
</tr>
<tr>
<td>d</td>
<td>Two-digit day withleadingzeros</td>
<td>03</td>
</tr>
<tr>
<td>e</td>
<td>Two-digit day withoutleadingzeros</td>
<td>9</td>
</tr>
<tr>
<td>A</td>
<td>Full weekday name</td>
<td>Monday</td>
</tr>
<tr>
<td>a</td>
<td>Abbreviated weekday name</td>
<td>Mon</td>
</tr>
<tr>
<td>j</td>
<td>Three-digit day of year withleadingzeros</td>
<td>069</td>
</tr>
<tr>
<td>H</td>
<td>Two-digit hour withleadingzeros, between 00 and 23</td>
<td>18</td>
</tr>
<tr>
<td>k</td>
<td>Two-digit hour withoutleadingzeros, between 0 and 23</td>
<td>18</td>
</tr>
<tr>
<td>I</td>
<td>Two-digit hour withleadingzeros, between 01 and 12</td>
<td>06</td>
</tr>
<tr>
<td>I</td>
<td>Two-digit hour withoutleadingzeros, between 1 and 12</td>
<td>6</td>
</tr>
<tr>
<td>M</td>
<td>Two-digit minutes withleadingzeros</td>
<td>05</td>
</tr>
<tr>
<td>S</td>
<td>Two-digit seconds withleadingzeros</td>
<td>19</td>
</tr>
<tr>
<td>L</td>
<td>Three-digit milliseconds withleadingzeros</td>
<td>047</td>
</tr>
<tr>
<td>N</td>
<td>Nine-digit nanoseconds withleadingzeros</td>
<td>047000000</td>
</tr>
<tr>
<td>P</td>
<td>Uppercase nanoseconds withleadingzeros</td>
<td>PM</td>
</tr>
<tr>
<td>p</td>
<td>Lowercase nanoseconds withleadingzeros</td>
<td>pm</td>
</tr>
<tr>
<td>z</td>
<td>RFC 822 numeric offset from GMT</td>
<td>-0800</td>
</tr>
<tr>
<td>Z</td>
<td>Time zone</td>
<td>PST</td>
</tr>
<tr>
<td>s</td>
<td>Seconds since 1970-01-01 00:00:00 GMT</td>
<td>1078884319</td>
</tr>
<tr>
<td>Q</td>
<td>Milliseconds since 1970-01-01 00:00:00 GMT</td>
<td>1078884319047</td>
</tr>
</tbody>
</table>

There are other useful classes related to Date and time. For more details, you can refer to Java.
**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:

```java
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:

```java
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
import java.util.*;

class DiffDemo {
    public static void main(String args[]) {
        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);
    }
}
Here is the list of few useful support methods provided by GregorianCalendar class:

<table>
<thead>
<tr>
<th>SN</th>
<th>Methods with Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>void add(int field, int amount)</code></td>
</tr>
<tr>
<td></td>
<td>Adds the specified signed amount of time to the given time field, based on the calendar’s rules.</td>
</tr>
<tr>
<td>2</td>
<td><code>protected void computeFields()</code></td>
</tr>
<tr>
<td></td>
<td>Converts UTC as milliseconds to time field values.</td>
</tr>
<tr>
<td>3</td>
<td><code>protected void computeTime()</code></td>
</tr>
<tr>
<td></td>
<td>Overrides Calendar Converts time field values to UTC as milliseconds.</td>
</tr>
<tr>
<td>4</td>
<td><code>boolean equals(Object obj)</code></td>
</tr>
<tr>
<td></td>
<td>Compares this GregorianCalendar to an object reference.</td>
</tr>
<tr>
<td>5</td>
<td><code>int getField()</code></td>
</tr>
<tr>
<td></td>
<td>Gets the value for a given time field.</td>
</tr>
<tr>
<td>6</td>
<td><code>int getActualMaximum(int field)</code></td>
</tr>
<tr>
<td></td>
<td>Return the maximum value that this field could have, given the current date.</td>
</tr>
<tr>
<td>7</td>
<td><code>int getActualMinimum(int field)</code></td>
</tr>
<tr>
<td></td>
<td>Return the minimum value that this field could have, given the current date.</td>
</tr>
<tr>
<td>8</td>
<td><code>int getGreatestMinimum(int field)</code></td>
</tr>
<tr>
<td></td>
<td>Returns highest minimum value for the given field if varies.</td>
</tr>
<tr>
<td>9</td>
<td><code>Date getGregorianChange()</code></td>
</tr>
<tr>
<td></td>
<td>Gets the Gregorian Calendar change date.</td>
</tr>
</tbody>
</table>
int getLeastMaximum\textit{intfield}
Returns lowest maximum value for the given field if varies.

int getMaximum\textit{intfield}
Returns maximum value for the given field.

\textbf{Date \text{getTime}}
Gets this Calendar's current time.

\textbf{long \text{getTimeInMillis}}
Gets this Calendar's current time as a long.

\textbf{TimeZone \text{getTimeZone}}
Gets the time zone.

int getMinimum\textit{intfield}
Returns minimum value for the given field.

int \text{hashCode}
Override hashCode.

boolean \text{isLeapYear\textit{intyear}}
Determines if the given year is a leap year.

void \text{roll\textit{intfield}, booleanup}
Adds or subtracts \textit{up/down} a single unit of time on the given time field without changing larger fields.

void \text{set\textit{intfield}, intvalue}
Sets the time field with the given value.

void \text{set\textit{intyear, intmonth, intdate}}
Sets the values for the fields year, month, and date.

void \text{set\textit{intyear, intmonth, intdate, inthour, intminute}}
Sets the values for the fields year, month, date, hour, and minute.

void \text{set\textit{intyear, intmonth, intdate, inthour, intminute, intsecond}}
Sets the values for the fields year, month, date, hour, minute, and second.

void \text{setGregorianChange\textit{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:**

```java
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.