Java Programming Language

Date and Time - Creation, Operation and Formatting

Introduction

There are many Java classes available for date/time and it becomes pretty confusing, in particular, the `Date` and `Calendar` clases. The documentation is not very clear, and I have to look into the source codes to understand the salient features.

java.util.Date and java.text.SimpleDateFormat

`Date` is sufficient if you need only a *current timestamp* in your application, and you do not need to operate on dates, e.g., one-week later. You can further use `SimpleDateFormat` to control the date/time display format.

The class `java.util.Date` represents a specific instance in time, with millisecond precision. It has two constructors:

- `Date()`: Allocate a `Date` instance with the current time.
- `Date(long millisSinceEpoch)`: Allocate a `Date` instance with the given time.

From the source code, the no-arg constructor invokes `System.currentTimeMillis()` to get the milliseconds since January 1, 1970, 00:00:00 GMT (known as "epoch") and stores in a private variable `fastTime`. Take note that epoch is *absolute* and does not depends on local time zones.

Most of the methods in `Date` are deprecated (for lack of internationalization support), except:

- `long getTime()`: returns the number of milliseconds since epoch.
- `String toString()`: returns a date/time string in *local* time-zone using the *default locale* in the format: `dow mon dd hh:mm:ss zzz yyyy`, where `dow` is the day of week (Sun, ..., Sat), `mon` is the month (Jan, ..., Dec), zzz is the time zone. Take note that although `Date` is represented based on the absolute epoch, the `toString()` displays the local time, according to the default time zone.

The `Date`'s `toString()` method has a fixed date/time display format. You can use `SimpleDateFormat` to control the display format.

EXAMPLE:

```
import java.text.SimpleDateFormat;
import java.util.Date;

public class DateTest {
   public static void main(String[] args) {
      Date now = new Date();
      System.out.println("toString(): " + now);  // dow mon dd hh:mm:ss zzz yyyy

      // SimpleDateFormat can be used to control the date/time display format:
      //   E (day of week): 3E or fewer (in text xxx), >3E (in full text)
      //   M (month): M (in number), MM (in number with leading zero)
      //              3M: (in text xxx), >3M: (in full text full)
      //   h (hour): h, hh (with leading zero)
      //   m (minute)
      //   s (second)
      //   a (AM/PM)
      //   H (hour in 0 to 23)
      //   z (time zone)
      SimpleDateFormat dateFormatter = new SimpleDateFormat("E, y-M-d 'at' h:m:s a z");
      System.out.println("Format 1:   " + dateFormatter.format(now));

      dateFormatter = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
      System.out.println("Format 2:   " + dateFormatter.format(now));
```

```
      dateFormatter = new SimpleDateFormat("EEEE, MMMM d, yyyy");
      System.out.println("Format 3:   " + dateFormatter.format(now));
   }
}
```

```
toString(): Sat Sep 25 21:27:01 SGT 2010
Format 1:   Sat, 10-9-25 at 9:27:1 PM SGT
Format 2:   Sat 2010.09.25 at 09:27:01 PM SGT
Format 3:   Saturday, September 25, 2010
```

Summary:

- `Date` class is *sufficient* if you just need *a simple timestamp*.
- You could use `SimpleDateFormat` to control the date/time display format.

Use `java.util.Calendar` class if you need to extract year, month, day, hour, minute, and second, or manipulating these field (e.g., 7 days later, 3 weeks earlier).

Use `java.text.DateFormat` to format a `Date` (form `Date` to text) and parse a date string (from text to `Date`). `SimpleDateForamt` is a subclass of `DateFormat`.

`Date` is legacy class, which does not support internationalization. `Calendar` and `DateFormat` support locale (you need to consider locale only if you program is to be run in many countries concurrently).

java.util.Calendar & java.util.GregorianCalendar

The `Calendar` class provides support for:

1. maintaining a set of calendar fields such as `YEAR`, `MONTH`, `DAY_OF_MONTH`, `HOUR`, `MINUTE`, `SECOND`, `MILLISECOND`; and
2. manipulating these calendar fields, such as getting the date of the previous week, roll forward by 3 days.

`Calendar` provides internationalization support.

`Calendar` is a abstract class, and you cannot use the constructor to create an instance. Instead, you use the static method `Calendar.getInstance()` to instantiate an implementation sub-class.

- `Calendar.getInstance()`: return a `Calendar` instance based on the current time in the default time zone with the default locale.
- `Calendar.getInstance(TimeZone zone)`
- `Calendar.getInstance(Locale aLocale)`
- `Calendar.getInstance(TimeZone zone, Locale aLocale)`

Looking into the source code reveals that: `getInstance()` returns a `GregorianCalendar` instance for all locales, (except `BuddhistCalendar` for Thai ("`th_TH`") and `JapaneseImperialCalendar` for Japanese ("`ja_JP`")).

The most important method in `Calendar` is `get(int calendarField)`, which produces an `int`. The `calendarField` are defined as `static` constant and includes:

- `get(Calendar.DAY_OF_WEEK)`: returns 1 (`Calendar.SUNDAY`) to 7 (`Calendar.SATURDAY`).
- `get(Calendar.YEAR)`: year
- `get(Calendar.MONTH)`: returns 0 (`Calendar.JANUARY`) to 11 (`Calendar.DECEMBER`).
- `get(Calendar.DAY_OF_MONTH)`, `get(Calendar.DATE)`: 1 to 31
- `get(Calendar.HOUR_OF_DAY)`: 0 to 23
- `get(Calendar.MINUTE)`: 0 to 59
- `get(Calendar.SECOND)`: 0 to 59
- `get(Calendar.MILLISECOND)`: 0 to 999
- `get(Calendar.HOUR)`: 0 to 11, to be used together with `Calendar.AM_PM`.
- `get(Calendar.AM_PM)`: returns 0 (`Calendar.AM`) or 1 (`Calendar.PM`).
- `get(Calendar.DAY_OF_WEEK_IN_MONTH)`: `DAY_OF_MONTH` 1 through 7 always correspond to `DAY_OF_WEEK_IN_MONTH` 1; 8 through 14 correspond to `DAY_OF_WEEK_IN_MONTH` 2, and so on.
- `get(Calendar.DAY_OF_YEAR)`: 1 to 366
- `get(Calendar.ZONE_OFFSET)`: GMT offset value of the time zone.

- `get(Calendar.ERA)`: Indicate AD (`GregorianCalendar.AD`), BC (`GregorianCalendar.BC`).

A date in `Calendar` can be represented as:

```
YEAR + MONTH + DAY_OF_MONTH
YEAR + MONTH + WEEK_OF_MONTH + DAY_OF_WEEK
YEAR + MONTH + DAY_OF_WEEK_IN_MONTH + DAY_OF_WEEK
YEAR + DAY_OF_YEAR
YEAR + DAY_OF_WEEK + WEEK_OF_YEAR
```

A time in `Calendar` can be represented as:

```
HOUR_OF_DAY
AM_PM + HOUR
```

Example:

```
// Get the year, month, day, hour, minute, second
import java.util.Calendar;
public class GetYMDHMS {
   public static void main(String[] args) {
      Calendar cal = Calendar.getInstance();
      // You cannot use Date class to extract individual Date fields
      int year = cal.get(Calendar.YEAR);
      int month = cal.get(Calendar.MONTH);      // 0 to 11
      int day = cal.get(Calendar.DAY_OF_MONTH);
      int hour = cal.get(Calendar.HOUR_OF_DAY);
      int minute = cal.get(Calendar.MINUTE);
      int second = cal.get(Calendar.SECOND);

      System.out.printf("Now is %4d/%02d/%02d %02d:%02d:%02d\n",  // Pad with zero
          year, month+1, day, hour, minute, second);
   }
}
```

`Calendar` has these setters and operations:

- `void set(int calendarField, int value)`
- `void set(int year, int month, int date)`
- `void set(int year, int month, int date, int hour, int minute, int second)`
- `void add(int field, int amount)`: Adds or subtracts the specified amount of time to the given calendar field, based on the calendar's rules.
- `void roll(int calendarField, boolean up)`: Adds or subtracts (up/down) a single unit of time on the given time field without changing larger fields.
- `void roll(int calendarField, int amount)`: Adds the specified (signed) amount to the specified calendar field without changing larger fields.

Other frequently-used methods are:

- `Date getTime()`: return a `Date` object based on this `Calendar`'s value.
- `void setTime(Date date)`
- `long getTimeInMills()`: Returns this `Calendar's` time value in milliseconds.
- `void setTimeInMillis(long millis)`
- `void setTimeZone(TimeZone value)`

Conversion between Calendar and Date

You can use `getTime()` and `setTime()` to convert between `Calendar` and `Date`.

```
Date getTime(): Returns a Date object representing this Calendar's time value
void setTime(Date aDate): Sets this Calendar's time with the given Date instance
```

EXAMPLE:

```
import java.util.Date;
import java.util.Calendar;
import java.text.SimpleDateFormat;

public class CalendarTest {
```

```
   public static void main(String[] args) {
      Calendar cal = Calendar.getInstance();   // GregorianCalendar
      System.out.println("Calendar's toString() is : " + cal + "\n");
      System.out.println("Time zone is: " + cal.getTimeZone() + "\n");

      // An Easier way to print the timestamp by getting a Date instance
      Date date = cal.getTime();
      System.out.println("Current date and time in Date's toString() is : " + date + "\n");

      // Print Calendar's field
      System.out.println("Year  : " + cal.get(Calendar.YEAR));
      System.out.println("Month : " + cal.get(Calendar.MONTH));
      System.out.println("Day of Month : " + cal.get(Calendar.DAY_OF_MONTH));
      System.out.println("Day of Week  : " + cal.get(Calendar.DAY_OF_WEEK));
      System.out.println("Day of Year  : " + cal.get(Calendar.DAY_OF_YEAR));
      System.out.println("Week of Year : " + cal.get(Calendar.WEEK_OF_YEAR));
      System.out.println("Week of Month : " + cal.get(Calendar.WEEK_OF_MONTH));
      System.out.println("Day of the Week in Month : " + cal.get(Calendar.DAY_OF_WEEK_IN_MONTH));
      System.out.println("Hour  : " + cal.get(Calendar.HOUR));
      System.out.println("AM PM : " + cal.get(Calendar.AM_PM));
      System.out.println("Hour of the Day : " + cal.get(Calendar.HOUR_OF_DAY));
      System.out.println("Minute : " + cal.get(Calendar.MINUTE));
      System.out.println("Second : " + cal.get(Calendar.SECOND));
      System.out.println();

      // Manipulating Dates
      Calendar calTemp;
      calTemp = (Calendar) cal.clone();
      calTemp.add(Calendar.DAY_OF_YEAR, -365);
      System.out.println("365 days ago, it was: " + calTemp.getTime());

      calTemp = (Calendar) cal.clone();
      calTemp.add(Calendar.HOUR_OF_DAY, 11);
      System.out.println("After 11 hours, it will be: " + calTemp.getTime());

      // Roll
      calTemp = (Calendar) cal.clone();
      calTemp.roll(Calendar.HOUR_OF_DAY, 11);
      System.out.println("Roll 11 hours, it will be: " + calTemp.getTime());
      System.out.println();
   }
}
```

java.util.GregorianCalendar

The calendar that we use today, called *Gregorian calendar*, came into effect in October 15, 1582 in some countries and later in other countries. It replaces the *Julian calendar*. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400 (removing 3 leap years (or 3 days) for every 400 years). Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.

`java.util.Calendar` is an abstract class. `Calendar.getInstance()` returns an implementation class `java.util.GregorianCalendar` (except locales of "`th`" and "`jp`"). In Java, this `GregorianCalendar` handles both the Gregorian calendar as well as the Julian calendar, including the cut over.

`GregorianCalendar` has the following constructors:

- `GregorianCalendar()`: using the current time, with the default time zone and locale.
- `GregorianCalendar(int year, int month, int dayOfMonth)`: with the default time zone and locale.
- `GregorianCalendar(int year, int month, int dayOfMonth, int hourOfDay, int minute, int second)`
- `GregorianCalendar(TimeZone zone, Locale aLocale)`: using current time.
- `GregorianCalendar(TimeZone zone)`
- `GregorianCalendar(Locale aLocale)`

For example,

```
Calendar cal1 = new GregorianCalendar();  // allocate an instance and upcast to Calendar
Calendar cal2 = new GregorianCalendar(2010, 9, 26);  // allocate with the specified date
cal2.get(Calendar.DAY_OF_WEEK);                      // 1 (Sunday) to 7 (Saturday)
```

java.text.DateFormat & java.text.SimpleDateFormat

Read Java Tutorial, Internationalization ⇒ Formatting ⇒ Dates and Times at http://java.sun.com/docs/books/tutorial/i18n/format/dateintro.html.

`java.text.DateFormat` is an abstract class for formats (from date to text) and parses (from text to date) date/time in a text-language-independent manner. `SimpleDateFormat` is an implementation subclass. Date formatter operates on `Date` object.

To use the date formatter, first create a `DateFormat` object for the desired date/time format, and then use the `format()` method to produce a date/time string.

To use the `DateFormat`, use one of these static factory method to create an instance:

- `DateFormat.getDateTimeInstance()`: use the default style and locale to format date and time.
- `DateFormat.getDateTimeInstance(int dateStyle, int timeStyle, Locale aLocale)`: style includes `DateFormat.FULL`, `LONG`, `MEDIUM`, and `SHORT`.
- `DateFormat.getInstance()`: uses `SHORT` style for date and time.
- `DateFormat.getDateInstance()`, `DateFormat.getDateInstance(int style, Locale aLocale)`: date only.
- `DateFormat.getTimeInstance()`, `DateFormat.getTimeInstance(int style, Locale aLocale)`: time only.

To parse a text string int `Date`, use:

```
DateFormat formatter = ....
Date myDate = formatter.parse(myString);
```

EXAMPLE:

```
import java.util.Date;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Locale;

public class DateFormatTest {

   public static void main(String[] args) {
      Date now = new Date();

      // Use Date.toString()
      System.out.println(now);

      // Use DateFormat
      DateFormat formatter = DateFormat.getInstance(); // Date and time
      String dateStr = formatter.format(now);
      System.out.println(dateStr);
      formatter = DateFormat.getTimeInstance();        // time only
      System.out.println(formatter.format(now));

      // Use locale
      formatter = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL, Locale.FRANCE);
      System.out.println(formatter.format(now));

      // Use SimpleDateFormat
      SimpleDateFormat simpleFormatter = new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
      System.out.println(simpleFormatter.format(now));
   }
}
```

```
Sun May 16 19:38:41 SGT 2010
5/16/10 7:38 PM
7:38:41 PM
dimanche 16 mai 2010 19 h 38 SGT
Sun 2010.05.16 at 07:38:41 PM SGT
```

Measuring Time

Many applications (such as games and animation) require good timing control.

Java provides these `static` methods in `System` class:

`System.currentTimeMillis()`: Returns the current time in milliseconds since January 1, 1970 00:00:00 GMT (known as "epoch"), in `long`.

```
// Measuring elapsed time
long startTime = System.currentTimeMillis();
// The code being measured
  .......
long estimatedTime = System.currentTimeMillis() - startTime;
```

`System.nanoTime()`: Returns the current value of the most precise available system timer, in nanoseconds, in `long`. Introduced in JDK 1.5. `nanoTime()` is meant for measuring relative time interval instead of providing absolute timing.

```
// Measuring elapsed time
long startTime = System.nanoTime();
// The code being measured
  .......
long estimatedTime = System.nanoTime() - startTime;
```

Take note that milli is 10^{-3}=0.001, nano is 10^{-9}=0.000000001. There is no micro (10^{-6}=0.000001) timer.

REFERENCES & RESOURCES

- Java Tutorial, Internationalization ⇒ Formatting ⇒ Dates and Times at http://java.sun.com/docs/books/tutorial/i18n/format/dateintro.html.
- JDK source codes: `java.util.Date`, `java.util.Calendar`, `java.util.GregorianCalendar`.

Latest version tested: JDK 1.6
Last modified: September, 2010