Time & Date Representation in Java

M. Karpinskas

Further Reading

JAVA

- https://www.mkyong.com/java/java-date-and-calendar-examples/
- https://www.tutorialspoint.com/java/java_date_time.htm

JAVA 8

- http://www.journaldev.com/2800/java-8-date-localdate-localdatetime-instant
- https://www.tutorialspoint.com/java8/java8_datetime_api.htm

Date

```
import java.util.Date;

….

Date date = new Date();
```

Date class

- An object of type Date represents an instance in time
- Part of java.util.* (requires import statement)
- A new **Date** object is automatically set to the time of its creation, to the millisecond
- For example:

```
Date now = new Date();   // object now holds value
                         // of the current millisecond
```

The difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

```
System.out.println(sdf.format(new Date(0))); //    1970 01 01  03:00 (LT)
```

long

System.out.println(Long.MAX_VALUE); ------ **292 278 994** m. 08 17 09:12

System.out.println(new Date().getTime());

Max	9223372036854775807
Current	1510054305569

UTC

The official abbreviation for Coordinated Universal Time is UTC. It came about as a compromise between English and French speakers. Coordinated Universal Time in English would normally be abbreviated CUT. Temps Universel Coordonné in French would normally be abbreviated TUC.

Converting to String

- Like many other Java standard classes, Date includes a toString() method, which allows us to output the value of a Date object in the form: `Day Mon dd hh:mm:ss TMZ yyyy`
  - **Day** is a three-letter abbreviation for day of week – e.g. Sat, Sun, etc.
  - **Mon** is a three-letter abbreviation for month of year – e.g. Jan, Feb, etc.
  - **dd** is the one- or two-digit date
  - **hh:mm:ss** is the current time
  - **TMZ** is the time zone – for example,, a Date object created at Kirkwood would have a CDT or CST stamp
  - **yyyy** is the 4-digit year

SimpleDateFormat class

- The java.text.* package contains a class that allows us to output Date objects using other formats – the **SimpleDateFormat** class interacts with Date objects much like the DecimalFormat class interacts with floating-point numbers
- A **Date** object is created, then a **SimpleDateFormat** object is created to describe its output appearance

Example

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

public class DemoDate {
    public static void main (String [] args) {
        Date now;
        SimpleDateFormat fmt;
        now = new Date();
        fmt = new SimpleDateFormat ("dd MMM yyyy");
        System.out.println ("Today's date: " +
                            fmt.format(now));
    }
}
```

SimpleDateFormat patterns – a partial list

Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2006.02.01 AD at 12:08:56 CST
"hh 'o''clock' a, zzzz"	12 o'clock PM, Central Standard Time
"yyyyy.MMMMM.dd hh:mm aaa"	02006.February.01 12:08 PM
"EEE, MMM d, ''yy"	Wed, Feb 4, '06

More examples: http://java.sun.com/j2se/1.5.0/docs/api/

U1

1. Išveskite šiandienos datą (tik datą)
2. Kitoje eilutėje - laiką (tik laiką)

GregorianCalendar class

- Also from java.util.*
- More friendly than Date because it uses named constants
- Incorporates a Date object, accessible via the getTime() method
- Like a Date object, a **GregorianCalendar** object will have the current time as its value by default; it is relatively easy to create an object using a different moment in time

Example

```
GregorianCalendar myDay =
   new GregorianCalendar(1978, 4, 16);
```

- This creates an object that contains the Date value May 16, 1978
- Note that the month of May is represented by a 4 rather than a 5; this is because the first month, January, is represented by a 0

Example

- We can use a predefined constant from the Calendar class to represent the month, as follows:

```
GregorianCalendar myDay =
            new GregorianCalendar
                (1978, Calendar.MAY, 16);
```

- Calendar.MAY is another example of a class constant, like Math.PI

Extracting Date information from a GregorianCalendar object

- Using other predefined constants from the Calendar class and the GregorianCalendar object's get() method, we can extract various pieces of information about the underlying Date value
- The Calendar class constants include Calendar.DAY_OF_WEEK, Calendar.DATE, Calendar.MONTH, Calendar.DAY_OF_YEAR and Calendar.YEAR

Example

```
  GregorianCalendar myDay = new
GregorianCalendar(1978, 4, 16);
        System.out.println(
                (myDay.get(Calendar.MONTH) + 1)
                + "/" + myDay.get(Calendar.DATE)
                + "/" + myDay.get(Calendar.YEAR));

// prints 5/16/1978


        Date data = myDay.getTime();
```

Example 1.1 – Convert Date to String.

```
SimpleDateFormat sdf = new
SimpleDateFormat("dd/M/yyyy");
String date = sdf.format(new Date());
System.out.println(date); //15/10/2013
```

Example 1.2 – Convert String to Date.

```
SimpleDateFormat sdf = new SimpleDateFormat("dd-M-yyyy
hh:mm:ss");
String dateInString = "31-08-1982 10:20:56";
Date date = sdf.parse(dateInString);
System.out.println(date); //Tue Aug 31 10:20:56 SGT 1982
```

Example 1.3 – Get current date time

```
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");
Date date = new Date();
System.out.println(dateFormat.format(date)); //2013/10/15 16:16:39
```

Example 1.4 – Convert Calendar to Date

```
Calendar calendar = Calendar.getInstance();
        Date date =  calendar.getTime();
```

Example 2.3 – Set a date&time manually.

```
SimpleDateFormat sdf = new SimpleDateFormat("yyyy MMM dd
HH:mm:ss");

Calendar calendar = new GregorianCalendar(2013,1,28,13,24,56);
System.out.println("#1. " + sdf.format(calendar.getTime()));

//update a date
calendar.set(Calendar.YEAR, 2014);
calendar.set(Calendar.MONTH, 11);
calendar.set(Calendar.MINUTE, 33);

System.out.println("#2. " + sdf.format(calendar.getTime()));
```

U2

1. Išveskite Jūsų gimimo datą

U2

```
package date;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Scanner;
public class U2Date {

        public static void main(String[] args) throws ParseException {
                Calendar cal = new GregorianCalendar();

                cal.set(Calendar.YEAR, 2000);
                cal.set(Calendar.MONTH, Calendar.AUGUST);
                cal.set(Calendar.DATE, 9);

                SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");
                Date myDate = cal.getTime();
                System.out.println("Mano data: " + formatter.format(myDate));

                Scanner sc = new Scanner(System.in);

                System.out.println("Įveskite datą 'yyyy-mm-dd' formatu");
                String ivestas = sc.nextLine();

                myDate = formatter.parse(ivestas);
                System.out.println("Mano data: " + formatter.format(myDate));
        }

}
```

public static long currentTimeMillis()

Returns the current time in milliseconds. Note that while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger. For example, many operating systems measure time in units of tens of milliseconds.
See the description of the class Date for a discussion of slight discrepancies that may arise between "computer time" and coordinated universal time (UTC).

Returns:

the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

See Also:

Date

public static long nanoTime()

Returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.
<..>
For example, to measure how long some code takes to execute:

```
long startTime = System.nanoTime();
 // ... the code being measured ...
 long estimatedTime = System.nanoTime() - startTime;
```

To compare two nanoTime values

```
long t0 = System.nanoTime();
 ...
 long t1 = System.nanoTime();
```

one should use t1 - t0 < 0, not t1 < t0, because of the possibility of numerical overflow.

Returns:

the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds

Since:

1.5

https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime()

PVZ

long yourmilliseconds = System.currentTimeMillis();

SimpleDateFormat sdf = new SimpleDateFormat("MMM dd,yyyy HH:mm");

Date resultdate = new **Date(yourmilliseconds)**;

System.out.println(sdf.format(resultdate));

Date

```
long yourmilliseconds = System.currentTimeMillis();
```

```
long yourmilliseconds = 0;
```

```
Date resultdate = new Date(yourmilliseconds);
```

```
Thu Jan 01 03:00:00 EET 1970
```

EET – Eastern European Time (Time Zone Abbreviation)

Most of **Europe** uses three standard **time zones**. From west to **east** the **time zones**are Western **European Time** (WET) which is UTC/GMT +0, Central **European Time**(CET) which is UTC/GMT+1 and **Eastern European Time** (EET) which is UTC/GMT +2.

U3

Metodas su dviem datos parametrais, kuris pasako ar pirma data yra didesnė (ateityje) už antrąją

U4 datos

Metodas, kuris turi parametrą datos tipo ir taip pat grąžina datą.

Grąžinama data gaunama - prie parametro pridėjus vienerius metus.

U4 datos

Metodas, kuris turi parametrą datos tipo ir taip pat grąžina datą.

Grąžinama data gaunama - prie parametro pridėjus vienerius metus.

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

Date f;

...

cal.setTime(f);

cal.add(Calendar.YEAR, n); // Where n is int

f = cal.getTime();
```

Java 8 Date

OSA Java 8

LocalDate Contains just a date—no time and no time zone. A good example of LocalDate is your birthday this year. It is your birthday for a full day regardless of what time it is.

LocalTime Contains just a time—no date and no time zone. A good example of LocalTime is midnight. It is midnight at the same time every day.

LocalDateTime Contains both a date and time but no time zone. A good example of LocalDateTime is “the stroke of midnight on New Year’s.” Midnight on January 2 isn’t nearly as special, and clearly an hour after midnight isn’t as special either.

Ready to create your first date and time objects?

```
System.out.println(LocalDate.now());

System.out.println(LocalTime.now());

System.out.println(LocalDateTime.now());


LocalDate date1 = LocalDate.of(2015, Month.JANUARY, 20);

LocalDate date2 = LocalDate.of(2015, 1, 20);


LocalTime time1 = LocalTime.of(6, 15); // hour and minute

LocalTime time2 = LocalTime.of(6, 15, 30); // + seconds

LocalTime time3 = LocalTime.of(6, 15, 30, 200); // + nanoseconds
```

Formatting Dates and Times

```
LocalDate date = LocalDate.of(2020, Month.JANUARY, 20);

System.out.println(date.getDayOfWeek()); // MONDAY

System.out.println(date.getMonth()); // JANUARY

System.out.println(date.getYear()); // 2020

System.out.println(date.getDayOfYear()); // 20
```

If you don’t want to use one of the predefi ned formats, you can create your own. For example, this code spells out the month:

```
DateTimeFormatter f = DateTimeFormatter.ofPattern("MMMM dd, yyyy, hh:mm");

System.out.println(dateTime.format(f)); // January 20, 2020, 11:12
```

Parsing Dates and Times

```
DateTimeFormatter f = DateTimeFormatter.ofPattern("MM dd yyyy");

LocalDate date = LocalDate.parse("01 02 2015", f);

LocalTime time = LocalTime.parse("11:22");

System.out.println(date); // 2015-01-02

System.out.println(time); // 11:22
```

U5 LocalDateTime

Vartotojo paprašyti įvest datą

U5

```
public class U5LocalTateTime {


    public static void main(String[] args) {

        LocalDateTime mano = LocalDateTime

                .parse(new Scanner(System.in).nextLine(),

                DateTimeFormatter.ofPattern("yyyy-MM-dd.HH:mm"));

        System.out.println("data: " + mano);


    }


}
```

U5

```
package date;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;
public class U5LocalTateTime {
	public static void main(String[] args) {
		String ivestasTekstas = new Scanner(System.in).nextLine();
		DateTimeFormatter formateris =
DateTimeFormatter.ofPattern("yyyy-MM-dd.HH:mm");
		LocalDateTime mano = LocalDateTime.parse(ivestasTekstas,
formateris);
		System.out.println("data: " + mano);
		System.out.println("data: " + mano.format(formateris));


	}

}
```

U6 LocalDateTime

Prideti 2 val. 15 min;

U6

```
LocalDateTime laikas = LocalDateTime.now();

laikas = laikas.plusHours(2).plusMinutes(15);

System.out.println("laikas: " + laikas);
```