Date & Time in Java

Java time → package for date & time in Java

Before Java 8, Java util package used to hold the date time class now these classes are deprecated.

How Java stores a Date?

Date in Java is stored in the form of a long number. This long number holds the time number of milliseconds passed since 1 Jan 1970.

Java assumes that 1900 is the start year which means it calculates years passed since 1900 whenever we ask it for years passed.

~~System.currentTi~~
System.currentTimeMillis(): It returns the no. of milliseconds passed from 1 Jan 1970. Once no. of ms are calculated, we can calculate minutes, seconds & years passed.

Q.1) Is it safe to store the number of milliseconds in a variable of type long?

A1) Yes, it is absolutely safe to store the no. of milliseconds in a variable of type long, because the max. value that can be stored in long is app. 9×10^{19}. You can see that the max. value of long is huge. Therefore, we

do not need to worry about the value of milliseconds. Notice the o/p of the below code; the value of current time millis is 10^6 times smaller than the max. value of long data type.

Code:

```
import java.util.*;

public class CWH extends Thread
    p s v m ( ) {

sout("The max. value of long is: " +
                Long.MAX_VALUE);

sout("The value of current time in ms:" +
    System.currentTimeMillis());
    }
}
```

o/p:
The max. value of long is: 9×10^{19}
The value of current time in ms: 1.6×10^{13}

Date class in Java

→ Date class in Java is available in java.util package

→ This class provides the instant in time with precision of millisecond.

Constructors of the date class:

Date(): This constructor is used when we need an object of current date & time.

Date(long milliseconds):

This constructor creates a date object from the number of milliseconds passed ~~from~~ since Jan 1, 1970.

& Code:

```
import java.util.*;

public class CWH extends Thread {
    p s v m ( ) {

        Date d = new Date();
        sout("The current date is: " + d);

        ~~Date d1 = new Date(1.6×10^13);~~
        Date d1 = new Date(1.6×16^13 l);
        sout("The date calculated from
        milliseconds is: " + d1);

    }
}
```

o/p:

The current date is:

Sun May 23 00:24:17 IST 2021

The date calculated from milliseconds is:

Sun May 23 00:23:59 IST 2021

Methods of date class:

compareTo():

→ Checks for the equality of 2 dates
→ Returns 0, if dates are equal else returns 1.

Code:

```
import java.util.*;

public
public class CWH extends Thread {
    p s v m ( ) {

        Date d = new Date();

Date d1 = new Date(2021, 12, 24); // both
                                  dates are diff.

        sout(d1.compareTo(d));
    }
}
```

o/p
1

→ getTime():
It returns the number of milliseconds passed ~~from~~ since the midnight of Jan 1, 1970.

→ getYear():
Prints the current year

→ getDate():
Prints current Date

Code:

```
import java.util.*;

public class CurrentTimeWithThread {
    p s v m ( ) {

        Date d = new Date();

sout("The no. of milliseconds: " + d.getTime());

sout("Current date: " + d.getDate());

sout("Current Year: " + d.getYear());

    }
}
```

o/p: The number of milliseconds: 6.158×10^{13}
Current date: 23
Current years 122 → 2022 − 1900

Calendar Class in Java

→ The Calendar class in Java provides the methods that helps in converting date let" a specific instant in time.
→ It is an abstract class.
→ Since it is an abstract class, we cannot create an instance of this class with the help of a constructor.
→ We use the static method Calendar.getInstance() in order to implement a subclass.

Constructors of the Calendar class:

Calendar(): This constructor is used to construct a calendar with the default time zone & locale.

Calendar(Time zone, Locale locale): This constructor is used to construct a calendar with the specified time zone & locale.

Methods of calendar class:

→ get(int field):
• This method returns the value of specified calendar field.

→ add(int field, int amount)
• This method is useful for calculating the time before or after of a specified calendar field.

→ getWeeksInWeekYear():

- Returns the number of weeks

→ getMaximum(int field)

- Returns maximum value for specified calendar field

Code:

```
import java.util.*;

public class CWH extends Thread{
    p s vm ( ) {

        Calendar c = Calendar.getInstance();
        sout("Calendar Type: " + c.getCalendarType());
sout("Current year: " + c.get(Calendar.YEAR));
sout("Current day: " + c.get(Calendar.DAY_OF_WEEK));
sout("Current second: " + c.get(Calendar.SECOND));

sout("Current date: " + c.getTime());
c.add(Calendar.YEAR, 4);
sout("After 4 years : " + c.getTime());

sout("Weeks in a year : " + c.getWeeksInWeekYear());

sout("The maximum number of weeks in a year: "
        + c.getMaximum(Calendar.WEEK_OF_YEAR));

    }
}
```

Calendar Type: gregory
Current Year: 2021
Current Day: 1
Current second: 3
Current date: Sun May 23 12:14:24 IST 2021
After 4 yrs: Fri May 23 12:14:24 IST 2025
Weeks in a year: 52
The maximum no. of weeks in a year: 53

Gregorian Calendar class & Time zone

- Gregorian Calendar class is the concrete subclass of Calendar class.
- This class supports both Julian and Gregorian calendar systems

⇒ Difference bet^n Calendar & Gregorian Calendar class

The Calendar class is a abstract class. So, the instance of this class can not be instantiated. Therefore, we need to use the static method Calendar.getInstance() to initialize the object of the Calendar class.

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

Since GregorianCalendar class is a concrete subclass, it can be initialized as:

GregorianCalendar gcal = new GregorianCalendar();

Constructors of GregorianCalendar class

1) GregorianCalendar():

This constructor is used to initialize an object with the current time in default time-zone.

2) GregorianCalendar(int year, int month, int day)

This constructor is used to initialize an object with the date specified as parameters in the default timezone & default locale.

3) GregorianCalendar(int year, int month, int day, int hours, int minutes, int seconds)

Initializes object with more specific date & time in default time-zone & locale

4) GregorianCalendar(Locale locale):
Current date-time in default timezone & specified locale.

5) GregorianCalendar(TimeZone timezone)
Current date & time in default locale & specified time-zone

6) GregorianCalendar(TimeZone timeZone, Locale locale):
specified locale & time-zone

METHODS OF GREGORIAN CLASS

→ isLeapYear(int year):
- checks if year passed as parameter is leap year or not
- returns boolean

→ roll(int field, boolean up)
- This method subtracts a given single unit of time from the specified time field.
- true = rolls up the value by 1
- false = rolls down the value by 1

→ hashcode():
- This method returns the hashcode of calendar object.

Code:

```
import java.util.*;

public class CWH extends Thread {

    psvm() {

        GregorianCalendar c = new GregorianCalendar();
        sout(c.isLeapYear(2000));
        sout(c.isLeapYear(2021));

sout("Date before rolling: " + c.getTime());
c.
c.roll(Calendar.MONTH, true);
c.roll(Calendar.DATE, false);
c.roll(Calendar.YEAR, true);

sout("Date after rolling: " + c.getTime());

sout("The hashcode of this calendar is:" +
                                c.hashcode());

    }
}
o/p
true
false
Date before rolling: Wed May 06
                    07:53:24 IST 2021
Date after rolling: Sat Jun 25 07:53:24
                    IST 2022
The hashcode of for this calendar is: 1358707903
```

JAVA.TIME API

Date & time features in Java is primarily supported by 2 packages:

→ java.util → java.time

The package java.time was added with release of Java 8

Class

⟹ Classes of java.time:

Clock class:

→ This class provides access to current instant, date & time-zone using time-zone.

→ Clock class is an abstract class therefore it is not possible to create instance of clock class.

⟹ Some methods of the clock class:

→ abstract ZoneId getZone():
This method returns the time zone being used to create date & time objects

→ abstract Instant instant():
Returns current instant of clock

⟹ Duration class:

- This class is used to measure time in seconds & nano-seconds
- This class is immutable

Methods of Duration Class.

- boolean isNegative(): This method is used to check whether duration is negative
- isZero(): This method is used to check if the duration is zero. Returns boolean value.

LocalDate class:

→ This class is useful for representing dates in year, month, day format.
→ Dates can be represented without time

Methods of LocalDate class

- compareTo(): This method compares the equality of 2 dates.
  Returns boolean value
- withYear(int year): This method returns a copy of the LocalDate but alters year with the value of year passed as argument.

LocalTime class.

→ This class helps us to represent the time without the dates

→ Instances of local time class are mutable

⇒ Methods of Local Time

- **LocalTime plusHours(long hoursToAdd):**

→ Returns a copy of local time with specified number of Hours added.

- **LocalTime minusMinutes(long minutesToSub):**

→ Returns a copy of local time with specified no. of minutes subtracted.

Code:

```
import java.time.*;

public class CWH extends Thread{

    p s v m ( ) {

    ~~clock~~
     Clock cl = Clock.systemDefaultZone();
     sout ln(cl.getZone());
     sout ln(cl.Instant());

Duration d1 = Duration.between(
                    LocalTime.MIN, LocalTime.NOON);
 sout(d1.isNegative());
```

```
Duration d2 = Duration.between(LocalTime.MAX,
                               LocalTime.MIN);
sout(d2.isNegative());
sout(d2.isZero());

Duration d3 = Duration.between(LocalTime.MIN,
                               LocalTime.MIDNIGHT);
sout(d3.isZero());

LocalDate d = LocalDate.now();
sout(d);

LocalDate d = LocalDate.parse("2021-05-27");

LocalDate d1 = LocalDate.parse("2021-05-26");

LocalDate d2 = LocalDate.parse("2021-05-26");

sout(d1.equals(d));
sout(d2.equals(d1));

sout(d.withYear(2001));

LocalTime t = LocalTime.now();
sout(t);

LocalTime t1 = LocalTime.of(13,18,29);
sout("Time before :" + t);

LocalTime t2 = LocalTime.t.plusHours(5);
sout("After adding 5 hrs :" + t2);
```

```
LocalTime t3 = t1.minusMinutes(8);
sout("After sub. 8 minutes: " + t3);
        }
    }
```

o/p

Asia/Calcutta
2021-05-26T06:43:05.064840700Z

false
true

false
true

2024-05-26

false
true

2001-05-27
13:13:36.198479100

Time before: 13:18:29
After adding 5hrs: 18:18:29
After sub. 8 min.: 18:10:29

Date-Time formatter in Java

→ This class helps us to print & parse date date & time in our desired format.
→ The format method of the DateTimeformatter class is used to format the dates using our desired format.

Syntax:

~~public~~
public String format(DateTimeFormatter formatter)

Parameter:

The object of the formatter to be used is passed, and it cannot be null.

Exception:

This method throws DateTimeException.

Return Value:

Returns the string in format specified by user.
→ In addition to the format, formatters can be created with desired Locale, Chronology, ZoneId & DecimalStyle.

Predefined formatters of Date-Time class.

→ ISO_LOCAL_DATE:

- Formates the dates acc. to international std.

→ ISO_WEEK_DATE:
Returns the number of weeks & years

→ ISO_ORDINAL_DATE:
Returns the year & day of year.

Symbol	Meaning	Presentation	Examples
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
D	day-of-year	number	189

Code:

```
import java.time.*;

public class CWH extends Thread {
    p s v m ( ) {

LocalDateTime dt = LocalDateTime.now();
sout("Current date: " + dt);

DateTimeFormatter df = DateTimeFormatter.of
                       Pattern("dd.MM.yyyy");
                  // format

String myDate = dt.format(df); // Creating
                Date string using date & format

sout("After formatting:" + myDate);

DateTimeFormatter df1 = DateTimeFormatter.
                        ISO_LOCAL_DATE;
  // Formatting date in ISO format

String date1 = dt.format(df1);
sout("ISO format: " + date1);

DateTimeFormatter df2
```

y

y

o/p

Current date: 2021-05-26T18:15:42
554864400

After formatting: 26.05.2024

ISO format: 2021-05-26