Day-12 JA-111 Batch

Topics

- **Date and Time API**
- **Collection Framework Introduction**

Date and Time API in java

The data and time related classes are part of java.time package. Some important classes of this package are

1. The LocalDate class
2. The LocalTime class
3. The LocalDateTime class
4. The DateTimeFormatter class of package java.time.format package
5. The Duration class
6. The Period class
7. The ChronoUnit Enum

1. The LocalDate class

- The LocalDate represents a date in ISO format (yyyy-MM-dd) without time. It can be used to store dates like birthdays and paydays.
- This class is immutable and thread-safe.
- This class is final so not possible to create subclass of the same

Here is list of some methods of LocalDate class

- **static LocalDate now():** Obtains the current date from the system clock in the default time-zone.
- **static LocalDate of(int year, int month, int dayOfMonth):** Obtains an instance of LocalDate from a year, month and day.
- **static LocalDate parse(CharSequence text):** Obtains an instance of LocalDate from a text string such as 2007-12-03.
- **static LocalDate parse(CharSequence text, DateTimeFormatter formatter):** Obtains an instance of LocalDate from a text string using a specific formatter.
- **boolean isAfter(ChronoLocalDate other):** Checks if this date is after the specified date.
- **boolean isBefore(ChronoLocalDate other):** Checks if this date is before the specified date.
- **boolean isEqual(ChronoLocalDate other):** Checks if this date is equal to the specified date.
- **boolean isLeapYear():** Checks if the year is a leap year.
- **LocalDate plusDays(long daysToAdd):** Returns a copy of this LocalDate with the specified number of days added.
- **LocalDate plusMonths(long monthsToAdd):** Returns a copy of this LocalDate with the specified number of months added.
- **LocalDate plusWeeks(long weeksToAdd):** Returns a copy of this LocalDate with the specified number of weeks added.
- **LocalDate plusYears(long yearsToAdd):** Returns a copy of this LocalDate with the specified number of years added.

- **LocalDate minus(TemporalAmount amountToSubtract):** Returns a copy of this date with the specified amount subtracted.
- **LocalDate minusDays(long daysToSubtract):** Returns a copy of this LocalDate with the specified number of days subtracted.
- **LocalDate minusMonths(long monthsToSubtract):** Returns a copy of this LocalDate with the specified number of months subtracted.
- **LocalDate minusWeeks(long weeksToSubtract):** Returns a copy of this LocalDate with the specified number of weeks subtracted.
- **LocalDate minusYears(long yearsToSubtract):** Returns a copy of this LocalDate with the specified number of years subtracted.
- **LocalDateTime atTime(LocalTime time):** Combines this time with a date to create a LocalDateTime.
- **int getDayOfMonth():** Gets the day-of-month field.
- **int getDayOfYear():** Gets the day-of-year field.
- **int getYear():** Gets the year field.

2. The LocalTime class

- A time without a time-zone in the ISO-8601 calendar system, such as 10:15:30.
- LocalTime is an immutable date-time object that represents a time, often viewed as hour-minute-second. Time is represented to nanosecond precision.
- This class does not store or represent a date or time-zone. Instead, it is a description of the local time as seen on a wall clock.

Here is list of some methods of LocalTime class

- **static LocalTime now():** Obtains the current time from the system clock in the default time-zone.
- **static LocalTime of(int hour, int minute):** Obtains an instance of LocalTime from an hour and minute.
- **static LocalTime of(int hour, int minute, int second):** Obtains an instance of LocalTime from an hour, minute and second.
- **static LocalTime of(int hour, int minute, int second, int nanoOfSecond):** Obtains an instance of LocalTime from an hour, minute, second and nanosecond.
- **static LocalTime parse(CharSequence text):** Obtains an instance of LocalTime from a text string such as 10:15.
- **static LocalTime parse(CharSequence text, DateTimeFormatter formatter):** Obtains an instance of LocalTime from a text string using a specific formatter.
- **long toNanoOfDay():** Extracts the time as nanos of day, from 0 to 24 * 60 * 60 * 1,000,000,000 - 1.
- **int toSecondOfDay():** Extracts the time as seconds of day, from 0 to 24 * 60 * 60 - 1.
- **String toString():** Outputs this time as a String, such as 10:15.
- **LocalDateTime atDate(LocalDate date):** Combines this time with a date to create a LocalDateTime.
- **int compareTo(LocalTime other):** Compares this time to another time.
- **int getHour():** Gets the hour-of-day field.
- **int getMinute():** Gets the minute-of-hour field.
- **int getSecond():** Gets the second-of-minute field.
- **int getNano():** Gets the nano-of-second field.
- **boolean isAfter(LocalTime other):** Checks if this time is after the specified time.

- **boolean isBefore(LocalTime other):** Checks if this time is before the specified time.
- **LocalTime plusHours(long hoursToAdd):** Returns a copy of this LocalTime with the specified number of hours added.
- **LocalTime plusMinutes(long minutesToAdd):** Returns a copy of this LocalTime with the specified number of minutes added.
- **LocalTime plusSeconds(long secondstoAdd):** Returns a copy of this LocalTime with the specified number of seconds added.
- **LocalTime plusNanos(long nanosToAdd):** Returns a copy of this LocalTime with the specified number of nanoseconds added.
- **LocalTime minusHours(long hoursToSubtract):** Returns a copy of this LocalTime with the specified number of hours subtracted.
- **LocalTime minusMinutes(long minutesToSubtract):** Returns a copy of this LocalTime with the specified number of minutes subtracted.
- **LocalTime minusNanos(long nanosToSubtract):** Returns a copy of this LocalTime with the specified number of nanoseconds subtracted.
- **LocalTime minusSeconds(long secondsToSubtract):** Returns a copy of this LocalTime with the specified number of seconds subtracted.
- **String format(DateTimeFormatter formatter):** Formats this time using the specified formatter.

3. The LocalDateTime class

- A date-time without a time-zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30.
- LocalDateTime is an immutable date-time object that represents a date-time, often viewed as year-month-day-hour-minute-second.
- Other date and time fields, such as day-of-year, day-of-week and week-of-year, can also be accessed. Time is represented to nanosecond precision.
- Most of methods are common to LocalDate and LocalTime class

4. The DateTimeFormatter class

- This class of java.time.format package provides the main application entry point for printing and parsing and provides common implementations of DateTimeFormatter:
  - Using predefined constants, such as ISO_LOCAL_DATE
  - Using pattern letters, such as uuuu-MMM-dd
  - Using localized styles, such as long or medium

Here is list of some methods of DateTimeFormatter class

- **static DateTimeFormatter ofPattern(String pattern):** Creates a formatter using the specified pattern.
- **static DateTimeFormatter ofPattern(String pattern, Locale locale):** Creates a formatter using the specified pattern and locale.

Some predefined formats for

Formatter	Description	Example
BASIC_ISO_DATE	Basic ISO date	'20111203'
ISO_LOCAL_DATE	ISO Local Date	'2011-12-03'
ISO_OFFSET_DATE	ISO Date with offset	'2011-12-03+01:00'
ISO_DATE	ISO Date with or without offset	'2011-12-03+01:00'; '2011-12-03'
ISO_LOCAL_TIME	Time without offset	'10:15:30'
ISO_OFFSET_TIME	Time with offset	'10:15:30+01:00'
ISO_TIME	Time with or without offset	'10:15:30+01:00'; '10:15:30'
ISO_LOCAL_DATE_TIME	ISO Local Date and Time	'2011-12-03T10:15:30'
ISO_OFFSET_DATE_TIME	Date Time with Offset	2011-12-03T10:15:30+01:00'
ISO_ZONED_DATE_TIME	Zoned Date Time	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_DATE_TIME	Date and time with ZoneId	'2011-12-03T10:15:30+01:00[Europe/Paris]'
ISO_ORDINAL_DATE	Year and day of year	'2012-337'
ISO_WEEK_DATE	Year and Week	2012-W48-6'
ISO_INSTANT	Date and Time of an Instant	'2011-12-03T10:15:30Z'
RFC_1123_DATE_TIME	RFC 1123 / RFC 822	'Tue, 3 Jun 2008 11:05:30 GMT'

All letters 'A' to 'Z' and 'a' to 'z' are reserved as pattern letters. The following pattern letters are defined:

Symbol	Meaning	Presentation	Examples
G	era	text	AD; Anno Domini; A
u	year	year	2004; 04
y	year-of-era	year	2004; 04
D	day-of-year	number	189
M/L	month-of-year	number/text	7; 07; Jul; July; J
d	day-of-month	number	10
Q/q	quarter-of-year	number/text	3; 03; Q3; 3rd quarter
Y	week-based-year	year	1996; 96
w	week-of-week-based-year	number	27
W	week-of-month	number	4
E	day-of-week	text	Tue; Tuesday; T
e/c	localized day-of-week	number/text	2; 02; Tue; Tuesday; T
F	week-of-month	number	3
a	am-pm-of-day	text	PM
h	clock-hour-of-am-pm (1-12)	number	12
K	hour-of-am-pm (0-11)	number	0
k	clock-hour-of-am-pm (1-24)	number	0
H	hour-of-day (0-23)	number	0
m	minute-of-hour	number	30
s	second-of-minute	number	55
S	fraction-of-second	fraction	978
A	milli-of-day	number	1234
n	nano-of-second	number	987654321
N	nano-of-day	number	1234000000
V	time-zone ID	zone-id	America/Los_Angeles; Z; -08:30
z	time-zone name	zone-name	Pacific Standard Time; PST
O	localized zone-offset	offset-O	GMT+8; GMT+08:00; UTC-08:00;
X	zone-offset 'Z' for zero	offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15;
x	zone-offset	offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15;
Z	zone-offset	offset-Z	+0000; -0800; -08:00;

An Example

```
package com.masai.sprint3;
import java.time.*;
import java.time.format.*;

public class DateTimeDemo {
  public static void main(String[] args) {
    LocalDate today = LocalDate.now();
    LocalDate tomorrow = today.plusDays(1);
    LocalDate yesterday = today.minusDays(1);

    System.out.println("Yesterday is " + yesterday);
    System.out.println("Today is " + today);
    System.out.println("Tomorrow is " + tomorrow);
    System.out.println(today.getYear() + " is a leap year? " +
```

```
today.isLeapYear());
    System.out.println(today + " is before " + yesterday +  "?: " +
today.isBefore(yesterday));
    System.out.println(today + " is after " + yesterday +  "?: " +
today.isAfter(yesterday));

    LocalDate indepedanceDay = LocalDate.of(1947, 8, 15);
    System.out.println("Independance day of India is " +
indepedanceDay);
    //Code to convert date to a specific format
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-
yyyy");
    String text = indepedanceDay.format(formatter);
    System.out.println("The date is " + text);

    //Code to convert textual date to to LocalDate object
    //text is the date from which LocalDate object has to be created
    //formatter is the format of the input date format
    String republicDayInText = "26-01-1950";
    LocalDate republicDay = LocalDate.parse(republicDayInText,
formatter);
    System.out.println("The republic day of india is " + republicDay);

    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

    LocalTime timeNow = LocalTime.now();
    System.out.println("The time is " + timeNow);
    System.out.println("The time in minutes is " +
timeNow.toSecondOfDay());
    LocalDateTime ldt = timeNow.atDate(LocalDate.now());
    System.out.println("The date and time is " +
ldt.format(DateTimeFormatter.ISO_LOCAL_DATE_TIME));
  }
}
```

Output

```
Yesterday is 2022-12-05
Today is 2022-12-06
Tomorrow is 2022-12-07
2022 is a leap year? false
2022-12-06 is before 2022-12-05?: false
2022-12-06 is after 2022-12-05?: true
Independance day of India is 1947-08-15
The date is 15-08-1947
The republic day of india is 1950-01-26
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
The time is 13:18:25.877815100
The time in minutes is 47905
The date and time is 2022-12-06T13:18:25.877815100
```

```
The date and time is 2022-12-06T13:18:25.8778151
```

5. The Duration class

- This class models a quantity or amount of time in terms of seconds and nanoseconds. It can be accessed using other duration-based units, such as minutes and hours.
- In addition, the DAYS unit can be used and is treated as exactly equal to 24 hours, thus ignoring daylight savings effects.
- The range of a duration requires the storage of a number larger than a long. To achieve this, the class stores a long representing seconds and an int representing nanosecond-of-second, which will always be between 0 and 999,999,999. The model is of a directed duration, meaning that the duration may be negative.
- This class is immutable and thread-safe.

Some important methods of Duration class are-

- **Duration abs():** Returns a copy of this duration with a positive length.
- **int compareTo(Duration otherDuration):** Compares this duration to the specified Duration.
- **int getNano():** Gets the number of nanoseconds within the second in this duration.
- **long getSeconds():** Gets the number of seconds in this duration.
- **boolean isNegative():** Checks if this duration is negative, excluding zero.
- **boolean isZero():** Checks if this duration is zero length.
- **Duration minusDays(long daysToSubtract):** Returns a copy of this duration with the specified duration in standard 24 hour days subtracted.
- **Duration minusHours(long hoursToSubtract):** Returns a copy of this duration with the specified duration in hours subtracted.
- **Duration minusMinutes(long minutesToSubtract):** Returns a copy of this duration with the specified duration in minutes subtracted.
- **Duration minusSeconds(long secondsToSubtract):** Returns a copy of this duration with the specified duration in seconds subtracted.
- **Duration minusMillis(long millisToSubtract):** Returns a copy of this duration with the specified duration in milliseconds subtracted.
- **Duration minusNanos(long nanosToSubtract):** Returns a copy of this duration with the specified duration in nanoseconds subtracted.
- **long toDays():** Gets the number of days in this duration.
- **long toHours():** Gets the number of hours in this duration.
- **long toMillis():** Converts this duration to the total length in milliseconds.
- **long toMinutes():** lets the number of minutes in this duration.
- **long toNanos():** Converts this duration to the total length in nanoseconds expressed as a long.

6. The Period class

- A date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'.
- This class models a quantity or amount of time in terms of years, months and days.
- The Period class is useful for daylight saving time

Most of the methods are common for Period and Duration class

An Example

```
package com.masai.sprint3;
import java.time.*;

public class DurationPeriodDemo {
  public static void main(String[] args) {
    LocalDateTime eveningTime = LocalDateTime.of(2022, 8, 15, 22, 00);
    LocalDateTime nextDayMorningTime = LocalDateTime.of(2022, 8, 16, 12,
00);
    Duration duration = Duration.between(eveningTime,
nextDayMorningTime);
    System.out.println("Duration in days is " + duration.toDays());
    System.out.println("Duration in hours is " + duration.toHours());
    System.out.println("Duration in minutes is " +
duration.toMinutes());
    System.out.println("Duration in seconds is " +
duration.toSeconds());
    System.out.println("Duration in nanoseconds is " +
duration.toNanos());

    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-");

    Period period = Period.between(LocalDate.now(),
LocalDate.now().plusDays(475));
    System.out.println("Period is years is " + period.getYears());
    System.out.println("Period is months is " + period.getMonths());
    System.out.println("Period is days is " + period.getDays());
  }
}
```

Output

```
Duration in days is 0
Duration in hours is 14
Duration in minutes is 840
Duration in seconds is 50400
Duration in nanoseconds is 50400000000000
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Period is years is 1
Period is months is 3
Period is days is 19
```

The ChronoUnit Enum

- This ChronoUnit enum belongs to java.time.temporal package. it is the most useful structure in java.time API.
- ChronoUnit is an enum that implements the TemporalUnit interface, which provides the standard units used in the Java Date Time API.
- With the help of this enum also we can get the difference between 2 dates in more precise way.

An Example:

```
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

class ChronoMain {
  public static void main(String args[]) {
    LocalDateTime oldDate = LocalDateTime.of(1982, 5, 31, 10, 20, 55);
    LocalDateTime newDate = LocalDateTime.of(2016, 9, 9, 10, 21, 56);

    System.out.println(oldDate);
    System.out.println(newDate);

    System.out.println(ChronoUnit.YEARS.between(oldDate, newDate) + "
years");
    System.out.println(ChronoUnit.MONTHS.between(oldDate, newDate) + "
months");
    System.out.println(ChronoUnit.WEEKS.between(oldDate, newDate) + "
weeks");
    System.out.println(ChronoUnit.DAYS.between(oldDate, newDate)+ "
days");
    System.out.println(ChronoUnit.HOURS.between(oldDate, newDate) + "
hours");
    System.out.println(ChronoUnit.MINUTES.between(oldDate, newDate) + "
minutes");
    System.out.println(ChronoUnit.SECONDS.between(oldDate, newDate) + "
seconds");
    System.out.println(ChronoUnit.MILLIS.between(oldDate, newDate) + "
milis");
    System.out.println(ChronoUnit.NANOS.between(oldDate, newDate) + "
nano");
  }
}
```

Output

```
1982-05-31T10:20:55
2016-09-09T10:21:56
34 years
411 months
1788 weeks
12520 days
300480 hours
18028801 minutes
1081728061 seconds
1081728061000 milis
1081728061000000000 nano
```

Collection Framework

If we want to group together the objects of a class then we have used array of objects.

Example:

```
String[] names = new String[3];  //Create array of references
names[0] = "Chitranshi";  //Assign object to each reference one by one
names[1] = "Anu";
names[2] = "Bhakti";
```

Limitation with the Array

1. Arrays are having fixed size in nature. In case of arrays, we are able to add the elements up to the specified size only, we are unable to add the elements over its size, if we are trying to add elements over its size then JVM will rise an exception like "ArrayIndexOutOfBoundsException".
2. Array does not have predefined methods to perform searching and sorting operations over the elements, in case of arrays to perform searching and sorting operations developers have to provide their own logic.
3. In array if you want to keep distinct elements only, want to keep the elements sorted always despite of several insertions and deletions then developer has to write enormous amount of code and writing highly efficient and error free code require a lot of skill & experience.

Solution: Collection framework in java

What is collection?

- A collection is simply an object which has group multiple elements into a single unit.
- Collections are used to store retrieve, manipulate, and communicate aggregate data.

What collection framework contains?

Interfaces: Java provides a set of interfaces that form a hierarchy. Interface allows to manipulate collection independent to their representation.

Implementations: It has a set of classes that provides implementation of interfaces. In simple words we can say that these classes provides reusable data structure.

Algorithms: Collection framework provides a wide range of methods that perform a set of useful and most widely used operation. Collection framework provides reusable functionality.

What collection framework provides?

Collection framework provides a unified approach for representing and manipulating collections. Collection framework is designed to meet following goals

1. Provides high performance implementation of most widely used collection object. It provides implementation of dynamic arrays, linked list and hash tables hence programmer does not need to develop their own code to implement these.
2. Provides high degree of interoperability and work in similar manner for different type of collections.
3. Designed with keeping flexibility in mind, It is very easy to extend and adapt inbuilt interface.

Tip: All classes and interfaces related to the collection framework are in the **java.util** package.

Collection framework contains two sections:-

1. Normal Collection (multiple objects are handled individually)
2. Map (multiple objects are handled in the form of pair (key-value)).

Let us start with some discussion about normal Collection.