UNIT 1

JAVA DATE AND TIME

1.INTRODUCTION

The java.time, java.util, java.sql and java.text packages contains classes for representing date and time. Following classes are important for dealing with date in Java.

Java 8 Date/Time API

Java has introduced a new Date and Time API since Java 8. The java.time package contains Java 8 Date and Time classes.

- java.time.LocalDate class
- java.time.LocalTime class
- java.time.LocalDateTime class
- java.time.MonthDay class
- java.time.OffsetTime class
- java.time.OffsetDateTime class
- java.time.Clock class
- java.time.ZonedDateTime class
- java.time.ZoneId class
- java.time.ZoneOffset class
- java.time.Year class
- java.time.YearMonth class
- java.time.Period class
- java.time.Duration class
- java.time.Instant class
- java.time.DayOfWeek enum
- java.time.Month enu

Classical Date/Time API

But classical or old Java Date API is also useful. Let's see the list of classical Date and Time classes.

- java.util.Date class
- java.sql.Date class
- java.util.Calendar class
- java.util.GregorianCalendar class
- java.util.TimeZone class
- java.sql.Time class
- java.sql.Timestamp class

Formatting Date and Time

We can format date and time in Java by using the following classes:

- **java.text.SimpleDateFormat class**

Java Date and Time APIs

Java provide the date and time functionality with the help of two packages java.time and java.util. The package java.time is introduced in Java 8, and the newly introduced classes tries to overcome the shortcomings of the legacy java.util.Date and java.util.Calendar classes.

Classical Date Time API Classes

The primary classes before Java 8 release were:

Java.lang.System: The class provides the currentTimeMillis() method that returns the current time in milliseconds. It shows the current date and time in milliseconds from January 1st 1970.

java.util.Date: It is used to show specific instant of time, with unit of millisecond.

java.util.Calendar: It is an abstract class that provides methods for converting between instances and manipulating the calendar fields in different ways.

java.text.SimpleDateFormat: It is a class that is used to format and parse the dates in a predefined manner or user defined pattern.

java.util.TimeZone: It represents a time zone offset, and also figures out daylight savings.

Drawbacks of existing Date/Time API's

1. **Thread safety:** The existing classes such as Date and Calendar does not provide thread safety. Hence it leads to hard-to-debug concurrency issues that are needed to be taken care by developers. The new Date and Time APIs of Java 8 provide thread safety and are immutable, hence avoiding the concurrency issue from developers.
2. **Bad API designing:** The classic Date and Calendar APIs does not provide methods to perform basic day-to-day functionalities. The Date and Time classes introduced in Java 8 are ISO-centric and provides number of different methods for performing operations regarding date, time, duration and periods.
3. **Difficult time zone handling:** To handle the time-zone using classic Date and Calendar classes is difficult because the developers were supposed to write the logic for it. With the new APIs, the time-zone handling can be easily done with Local and ZonedDate/Time APIs.

NEW DATE TIME API IN JAVA 8

The new date API helps to overcome the drawbacks mentioned above with the legacy classes. It includes the following classes:

java.time.LocalDate: It represents a year-month-day in the ISO calendar and is useful for representing a date without a time. It can be used to represent a date only information such as a birth date or wedding date.

java.time.LocalTime: It deals in time only. It is useful for representing human-based time of day, such as movie times, or the opening and closing times of the local library.

java.time.LocalDateTime: It handles both date and time, without a time zone. It is a combination of LocalDate with LocalTime.

java.time.OffsetTime: It handles time with a corresponding time zone offset from Greenwich/UTC, without a time zone ID.

java.time.OffsetDateTime: It handles a date and time with a corresponding time zone offset from Greenwich/UTC, without a time zone ID.

java.time.Clock : It provides access to the current instant, date and time in any given time-zone. Although the use of the Clock class is optional, this feature allows us to test your code for other time zones, or by using a fixed clock, where time does not change.

java.time.Duration : Difference between two instants and measured in seconds or nanoseconds and does not use date-based constructs such as years, months, and days, though the class provides methods that convert to days, hours, and minutes.

java.time.Period : It is used to define the difference between dates in date-based values (years, months, days).

2.JAVA LOCALDATE CLASS.

LocalDate class declaration

Let's see the declaration of java.time.LocalDate class.

```
public final class LocalDate extends Object
implements Temporal, TemporalAdjuster, ChronoLocalDate, Serializable
```

Java does not have a built-in Date class, but we can import the java.time package to work with the date and time API. The package includes many date and time classes. For example:

Class	Description
LocalDate	Represents a date (year, month, day (yyyy-MM-dd))

To display the current date, import the java.time.LocalDate class, and use its now() method:

PROGRAM

```
import java.time.LocalDate;  // import the LocalDate class

public class Main {

  public static void main(String[] args) {

    LocalDate myObj = LocalDate.now();  // Create a date object

    System.out.println(myObj);  // Display the current date

  }

}
```

OUTPUT

```
2023-03-27
```

3.LOCALTIME CLASS

Java LocalTime class is an immutable class that represents time with a default format of hour-minute-second. It inherits Object class and implements the Comparable interface.

LocalTime class declaration

Let's see the declaration of java.time.LocalTime class.

```
public final class LocalTime extends Object
implements Temporal, TemporalAdjuster, Comparable<LocalTime>, Serializable
```

To display the current time (hour, minute, second, and nanoseconds), import the java.time.LocalTime class, and use its now() method:

Class	Description
LocalTime	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))

PROGRAM

```
import java.time.LocalTime;  // import the LocalTime class


public class Main {

  public static void main(String[] args) {

    LocalTime myObj = LocalTime.now();

    System.out.println(myObj);

  }

}
```

OUTPUT

```
15:23:25.977146
```

4. DATE AND TIME

class declaration

Let's see the declaration of java.time.LocalDateTime class.

```
public final class LocalDateTime extends Object
implements Temporal, TemporalAdjuster, ChronoLocalDateTime<LocalDate>, Serializable
```

To display the current date and time, import the java.time.LocalDateTime class, and use its now() method:

Class	Description
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)

PROGRAM

```
import java.time.LocalDateTime;  // import the LocalDateTime class


public class Main {
  public static void main(String[] args) {
    LocalDateTime myObj = LocalDateTime.now();
    System.out.println(myObj);
  }
}
```

OUTPUT

```
2023-03-27T15:31:01.840750
```

5.JAVA MONTHDAY

The **java.time.MonthDay** class represents a month-day in the ISO-8601 calendar system, such as --12-03.

Class declaration:

Following is the declaration for **java.time.MonthDay** class −

```
public final class MonthDay
   extends Object
implements TemporalAccessor, TemporalAdjuster, Comparable<MonthDay>, Serializable
```

PROGRAM

```
import java.time.*;
public class MonthDayExample1 {
  public static void main(String[] args) {
    MonthDay month = MonthDay.now();
    LocalDate date = month.atYear(1994);
    System.out.println(date);
  }
}
```

OUTPUT

1994-01-17

6.OFFSETTIME

OffsetTime is an immutable date-time object that represents a time, often viewed as hour-minute-second-offset. This class stores all time fields, to a precision of nanoseconds, as well as a zone offset. For example, the value "13:45.30.123456789+02:00" can be stored in an OffsetTime.

This is a value-based class; use of identity-sensitive operations (including reference equality (==), identity hash code, or synchronization) on instances of OffsetTime may have unpredictable results and should be avoided. The equals method should be used for comparisons.

Class declaration

Following is the declaration for **java.time.OffsetTime** class −

```
public final class OffsetTime
   extends Object
      implements Temporal, TemporalAdjuster, Comparable<OffsetTime>, Serializable
```

Field

Following are the fields for **java.time.OffsetTime** class −

- **static OffsetTime MAX** − The maximum supported OffsetTime, '23:59:59.999999999-18:00'.
- **static OffsetTime MIN** − The minimum supported OffsetTime, '00:00:00+18:00'.

PROGRAM

```
import java.time.OffsetTime;
import java.time.temporal.ChronoField;
public class OffsetTimeExample1 {
  public static void main(String[] args) {
    OffsetTime offset = OffsetTime.now();
    int h = offset.get(ChronoField.HOUR_OF_DAY);
    System.out.println(h);
    int m = offset.get(ChronoField.MINUTE_OF_DAY);
    System.out.println(m);
    int s = offset.get(ChronoField.SECOND_OF_DAY);
    System.out.println(s);
  }
}
```

OUTPUT

```
16
970
58224
```

7.JAVA OFFSETDATETIME CLASS

Java OffsetDateTime class is an immutable representation of a date-time with an offset. It inherits Object class and implements the Comparable interface.

OffsetDateTime class is used to store the date and time fields, to a precision of nanoseconds.

Class declaration

Following is the declaration for **java.time.OffsetDateTime** class −

```
public final class OffsetDateTime
   extends Object
      implements Temporal, TemporalAdjuster, Comparable<OffsetDateTime>, Serializable
```

Field

Following are the fields for **java.time.OffsetDateTime** class −

- **static OffsetDateTime MAX** − The maximum supported OffsetDateTime, '+999999999-12-31T23:59:59.999999999+18:00'.
- **static OffsetDateTime MIN** − The minimum supported OffsetDateTime, '-999999999-01-01T00:00:00-18:00'.

PROGRAM

```
import java.time.OffsetDateTime;
public class OffsetDateTimeExample1 {
public static void main(String[] args) {
OffsetDateTime offsetDT = OffsetDateTime.now();
System.out.println(offsetDT.getDayOfMonth());
}
}
```

OUTPUT:

```
18
```

8.JAVA CLOCK CLASS

The **java.time.Clock** class provides access to the current instant, date and time using a time-zone.

The aim of the Clock class is to allow you to plug in another clock whenever you need it. Instead of using a static method, applications utilise an object to get the current time. It simplifies the testing process. A method that requires a current instant can take a Clock as a parameter.

Class Declaration

Let's see the declaration of java.time.Clock class.

public abstract class Clock **extends** Object

Method	Description
abstract ZoneId getZone()	It is used to get the time-zone being used to create dates and times.
abstract Instant instant()	It is used to get the current instant of the clock.
static Clock offset(Clock baseClock, Duration offsetDuration)	It is used to obtain a clock that returns instants from the specified clock with the specified duration added

PROGRAM

```
import java.time.Clock;
public class ClockExample1 {
  public static void main(String[] args) {
    Clock c = Clock.systemDefaultZone();
    System.out.println(c.getZone());
  }
}
```

OUTPUT:

```
Asia/Calcutta
```

9.JAVA YEAR CLASS

Java Year class is an immutable date-time object that represents a year. It inherits the Object class and implements the Comparable interface.

Class declaration

Following is the declaration for **java.time.Year** class −

```
public final class Year
   extends Object
      implements Temporal, TemporalAdjuster, Comparable<Year>, Serializable
```

Field

Following are the fields for **Java.time.Period** class −

- **static int MAX_VALUE** − The maximum supported year, '+999,999,999'.
- **static int MIN_VALUE** − The minimum supported year, '-999,999,999'.

Method	Description
LocalDate atDay(int dayOfYear)	It is used to combine this year with a day-of-year to create a LocalDate.

PROGRAM

```
import java.time.Year;
```

```
public class YearExample1 {
public static void main(String[] args) {
Year y = Year.now();
System.out.println(y);
}
}
```

OUTPUT:

```
2023
```

10.JAVA YEARMONTH CLASS

Java YearMonth class is an immutable date-time object that represents the combination of a year and month. It inherits the Object class and implements the Comparable interface.

Class declaration

Following is the declaration for **java.time.YearMonth** class −

```
public final class YearMonth
   extends Object
      implements Temporal, TemporalAdjuster, Comparable<YearMonth>, Serializable
```

Methods of Java YearMonth

Method	Description
Temporal adjustInto(Temporal temporal)	It is used to adjust the specified temporal object to have this year-month.

PROGRAM

```
import java.time.YearMonth;
```

```
public class YearMonthExample1 {
public static void main(String[] args) {
YearMonth ym = YearMonth.now();
System.out.println(ym);
}
}
```

OUTPUT:

```
2023-01
```

11.JAVA PERIOD CLASS

Java Period class is used to measures time in years, months and days. It inherits the Object class and implements the ChronoPeriod interface.

Class declaration

Following is the declaration for **java.time.Period** class −

```
public final class Period
   extends Object
      implements ChronoPeriod, Serializable
```

Field

Following are the fields for **Java.time.Period** class −

- **static Period ZERO** − Constant for a Period of zero.

Methods of Java Period

Method	Description
Temporal addTo(Temporal temporal)	It is used to add this period to the specified temporal object.

PROGRAM

```
import java.time.*;
```

```
import java.time.temporal.Temporal;
public class PeriodExample1 {
public static void main(String[] args) {
Period period = Period.ofDays(24);
Temporal temp = period.addTo(LocalDate.now());
System.out.println(temp);
 }
}
```

OUTPUT:

```
2017-02-24
```

12.JAVA DURATION CLASS

The **java.time.Duration** 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.

Class declaration

Following is the declaration for **java.time.Duration** class −

```
public final class Duration
   extends Object
      implements TemporalAmount, Comparable<Duration>, Serializable
```

Methods of Java Duration

Method	Description
Temporal addTo(Temporal temporal)	It is used to add this duration to the specified temporal object.

PROGRAM

```
import java.time.*;
import java.time.temporal.ChronoUnit;
public class DurationExample1 {
public static void main(String[] args) {
Duration d = Duration.between(LocalTime.NOON,LocalTime.MAX);
System.out.println(d.get(ChronoUnit.SECONDS));
}
}
```

OUTPUT:

```
43199
```