java.util.Date vs java.sql.Date
==========

```
public class Test {
	public static void main(String[] args) {

		//To use Date in general operations like printing the date and time
		java.util.Date utilDate = new java.util.Date();
		System.out.println(utilDate);


		long l = utilDate.getTime();//It is giving the information of utilDate
in milliseconds from 1970
		System.out.println(l+"ms");

		//To use Date in DB operations like insert,update,delete query we use
sqlDate
		java.sql.Date sqlDate = new java.sql.Date(l);
		System.out.println(sqlDate);
	}
}
```

Output

```
Sun Oct 30 10:05:33 IST 2022
1667107558145ms
2022-10-30(yyyy-mm-dd)
```

Difference b/w java.util.Date and java.sql.Date
==========

java.util.Date

- => It is a utility class to handles Date in our java program.
- => It represents both Date and Time

java.sql.Date

- => It is designed class to handle Dates w.r.t DB operations
- => It represents only Date,but not Time.

Note: In sql package

Time(C)	represents only => Time value
TimeStamp(C)	represents both => Date and Time value

Date and Time API: (Joda-Time API)

Until Java 1.7 version the classes present in Java.util package to handle Date and Time (like Date, Calendar,
TimeZone etc) are not up to the mark with respect to convenience and performance.

To overcome this problem in the 1.8 version oracle people introduced Joda-Time API. This API developed by joda.org and available in Java in the form of "java.time" package.

```
# program for to display System Date and time.
import Java.time.*;
public class DateTime {
		public static void main(String[] args) {
				LocalDate date = LocalDate.now();
				System.out.println(date);

				LocalTime time=LocalTime.now();
```

```
                        System.out.println(time);
            }
}
Output
======
2022-10-30
11:15:41.698
```

Once we get LocalDate object we can call the following methods on that object to retrieve Day,month and year values separately.

Ex:

```
import java.time.*;
 class Test {
       public static void main(String[] args) {
                  LocalDate date = LocalDate.now();
                  System.out.println(date);
                  int dd =   date.getDayOfMonth();
                  int mm = date.getMonthValue();
                  int yy = date.getYear();
                  System.out.println(dd+"..."+mm+"..."+yy);
                  System.out.printf("\n%d-%d-%d",dd,mm,yy);
      }
}
Output
2022-10-30
30...10...2022
30-10-2022
```

Once we get LocalTime object we can call the following methods on that object.

```
import java.time.*;
class Test {
       public static void main(String[] args) {
                  LocalTime time = LocalTime.now();
                  int h = time.getHour();
                  int m = time.getMinute();
                  int s = time.getSecond();
                  int n = time.getNano();
                  System.out.printf("\n%d:%d:%d:%d",h,m,s,n);
      }
}
Output
9:22:31:795000000
```

Note::

If we want to represent both Date and Time then we should go for LocalDateTime object.

```
LocalDateTimedt = LocalDateTime.now();
System.out.println(dt);
O/p: 2015-11-23T12:57:24.531
```

We can represent a particular Date and Time by using LocalDateTime object as follows.

Ex:

```
 LocalDateTime dt1 = LocalDateTime.of(1995,Month.APRIL,28,12,45);
 sop(dt1);
```

Ex:

```
 LocalDateTime dt1=LocalDateTime.of(1995,04,28,12,45);
```

```
 Sop(dt1);
 Sop(“After six months:”+dt.plusMonths(6));
 Sop(“Before six months:”+dt.minusMonths(6));
```

ZoneId
=====
To Represent Zone:
ZoneId object can be used to represent Zone.
Ex:

```
import Java.time.*;
  class ProgramOne {
             public static void main(String[] args) {
                        ZoneId zone = ZoneId.systemDefault();
                        System.out.println(zone);
             }
}
```

Output
Asia/Calcutta

We can create ZoneId for a particular zone as follows
Ex:

```
 ZoneId la = ZoneId.of("America/Los_Angeles");
 ZonedDateTimezt = ZonedDateTime.now(la);
 System.out.println(zt);
```

Output
2022-10-29T23:19:59.718-07:00[America/Los_Angeles]

Period Object:
Period object can be used to represent quantity of time
Ex:

```
 LocalDate today = LocalDate.now();
 LocalDate birthday = LocalDate.of(1994,01,3);
 Period p = Period.between(birthday,today);
 System.out.printf("age is %d year %d months %d
days",p.getYears(),p.getMonths(),p.getDays());
```

Output
age is 28 year 9 months 27 days

write a program to check the given year is leap year or not

rule for leap year
==========
=> A year may be a leap year if it is evenly divisible by 4.
=> Years that are divisible by 100 (century years such as 1900 or 2000) cannot be leap years unless they are also
      divisible by 400.

```
import Java.time.*;
 public class Leapyear {
      public static void main(String[] args){
             int n = Integer.parseInt(args[0]);
             Year y = Year.of(n);
              if(y.isLeap())
                  System.out.printf("%d is Leap year",n);
             else
                  System.out.printf("%d is not Leap year",n);
```

```
      }
}


Note:
    Date -> LocalDate(C)
    Time -> LocalTime(C)
    Date & Time -> LocalDateTime(c)

      now() --->Current information
        of()     --> user specific information

      ZoneId -> Setting up the particular zone to fetch the information
       ZondDateTime-> To get the Date and time information of any zone.

       Period ---> To find difference b/w 2 date Objects
      Year    --->  To check whether the supplied year is leapYear or not.
```