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

java.util.Date

- It is a utility class to handle 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)	=> Time value
TimeStamp(C)	=> Date and Time value

Stream API in Java

Streams

To process objects of the collection, in 1.8 version Streams concept introduced.

What is the difference between Java.util.streams and Java.io streams?

java.util streams meant for processing objects from the collection. Ie, it represents a stream of objects from the collection but Java.io streams are meant for processing binary and character data with respect to file. i.e it represents a stream of binary data or character data from the file. Hence Java.io streams and Java.util streams both are different.

What is the difference between collection and stream?

- If we want to represent a group of individual objects as a single entity then We should go for collection.
- If we want to process a group of objects from the collection then we should go for streams.
- We can create a stream object to the collection by using the stream() method of the Collection interface.
- stream() method is a default method added to the Collection in 1.8 version.

```
default Stream stream()
Ex: Stream s = c.stream();
import java.util.*;
import java.util.stream.*;

public class Test {
 public static void main(String[] args) {
   ArrayList<Integer> al = new ArrayList<Integer>();
   al.add(0);
   al.add(5);
   al.add(10);
   al.add(15);
   al.add(20);
   al.add(25);

   System.out.println(al);
```

```
        ArrayList<Integer> doubleList = new ArrayList<Integer>();
          for ( Integer i1: al )
                        doubleList.add(i1*2);
          System.out.println(doubleList);

          List<Integer> streamList = al.stream().map(I→I*2).collect(Collectors.toList());
    System.out.println(streamList);
  }
}
```

=> Stream is an interface present in java.util.stream. Once we get the stream, by using that we can process objects of that collection.
We can process the objects in the following 2 phases
1.Configuration
2.Processing

1) Configuration:
We can configure either by using a filter mechanism or by using map mechanism.

Filtering:
We can configure a filter to filter elements from the collection based on some boolean condition by using filter()method of Stream interface.
public Stream filter(Predicate<T> t)
here (Predicate<T > t) can be a boolean valued function/lambda expression
Ex:
Stream s = c.stream();
Stream s1 = s.filter(i -> i%2==0);
Hence to filter elements of collection based on some Boolean condition we should go for filter() method.

Mapping:
If we want to create a separate new object, for every object present in the collection based on our requirement then we should go for
map() method of Stream interface.
public Stream map (Function f);
It can be lambda expression also
Ex:
Stream s = c.stream();
Stream s1 = s.map(i-> i+10);
Once we perform configuration we can process objects by using several methods.

2) Processing
processing by collect() method
Processing by count()method
Processing by sorted()method
Processing by min() and max() methods
forEach() method
toArray() method
Stream.of()method

1. collect()
This method collects the elements from the stream and adds the specified to the collection indicated

(specified) by argument.

2. count()
This method returns the number of elements present in the stream.
 public long count()

3. Processing by sorted()method
If we sort the elements present inside the stream then we should go for the sorted() method.
The sorting can either default to natural sorting order or customized sorting order specified by comparator.
  sorted()- default natural sorting order
  sorted(Comparator c)-customized sorting order.

4. Processing by min() and max() methods
min(Comparator c): returns minimum value according to specified comparator.
max(Comparator c): returns maximum value according to the specified comparator.

5. forEach() method
This method will not return anything.
This method will take lambda expression as argument and apply that lambda expression for each element present in the stream.

6. toArray() method
We can use toArray() method to copy elements present in the stream into specified array

7. Stream.of()method
We can also apply a stream for a group of values and for arrays.
Ex:
Stream s=Stream.of(99,999,9999,99999);
s.forEach(System.out:: println);

Double[] d={10.0,10.1,10.2,10.3};
Stream s1=Stream.of(d);
s1.forEach(System.out :: println);

Enums

We can use enum to define a group of named constants.
Example 1:
```
enum Month
{
 JAN,FEB,MAR, ... DEC; //; ⟶optional
}
Example 2:
enum Color
{
 RED,BLUE,GREEN;
}
```