1.Once stream is used u cant use it again. Its same like iterator

Couple of short circuiting operations are anyMatch() and findFirst().

The Diff between Collections and stream is, stream wont store data , whereas collection stores data

**Intermediate operations** do not actually perform any filtering, but instead creates a new stream that, when traversed, contains the elements of the initial stream that match the given predicate.

## Reference links

<https://stackoverflow.com/questions/68692822/how-to-get-first-and-last-day-in-specific-week-in-month>

<https://kodejava.org/how-do-i-get-the-first-sunday-of-the-year-in-java/>

<https://www.baeldung.com/java-8-date-time-intro>

for legend

<https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>

<https://www.baeldung.com/java-string-to-date#:~:text=Date%20object%3A,parse(dateInString)%3B>

Intermediate Operations - Types

Intermediate operations are further divided as

* stateless operations
* stateful operations

**Stateless operations**, such as filter and map, retain no state from previously seen element when processing a new element. Each element can be processed independent of operations on other elements.

**Stateful operations**, such as distinct and sorted, may incorporate state from previously seen elements when processing new elements.

##### Terminal Operation

Terminal operations may traverse the stream to produce a result or a side-effect.

After the terminal operation is performed, the stream pipeline is considered consumed, and can no longer be used.

Terminal operations are eager, completing their traversal of the data source and processing of the pipeline before returning

Few examples for terminal operation

forEach(), collect(), match(), count() and reduce()

StartsWith Demo

==============

String firstMatchedName = memberNames.stream() .filter((s) -> s.startsWith("L")) .findFirst().get();

# Methods in stream

Allmatch(Predicate p), anyMatch(Predicate p),noneMatch(Predicate p)

mapToInt

# Date and Time API in java8

New series of date and time APIs are created in the new java.time package.

Following are some of the important classes introduced in java.time package −

**Local** − Simplified date-time API with no complexity of timezone handling.

**Zoned** − Specialized date-time API to deal with various timezones

<https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html>

As such, a clock can be used instead of [System.currentTimeMillis()](https://docs.oracle.com/javase/8/docs/api/java/lang/System.html" \l "currentTimeMillis--) and [TimeZone.getDefault()](https://docs.oracle.com/javase/8/docs/api/java/util/TimeZone.html" \l "getDefault--).

long millis = Clock.*systemUTC*().millis();

LocalDate l=LocalDate.*of*(2011, Month.*FEBRUARY*,29);

LocalTime lt=LocalTime.*of*(12,60,60,200)

## How to get any country time as below

LocalTime lt=LocalTime.now(ZoneId.of("GMT"));

## Date Formatting

Old date time formatter

===========================

import java.text.SimpleDateFormat;

private static final DateFormat sdf = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

New Date time Formatter

=========================

import java.time.format.DateTimeFormatter;

public static void main(String[] args) {

Date date = new Date();

System.out.println(sdf.format(date));

Calendar cal = Calendar.getInstance();

System.out.println(sdf.format(cal.getTime()));

### **//from LocalDateTime to string conversion**

LocalDateTime now = LocalDateTime.now();

private static final DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy/MM/dd HH:mm:ss");

System.out.println(dtf.format(now));

LocalDate localDate = LocalDate.now();

System.out.println(DateTimeFormatter.ofPattern("yyy/MM/dd").format(localDate));

### **//String to LocalDate conversion**

String now = "2017-06-13 12:30";

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

LocalDateTime formatDateTime = LocalDateTime.parse(now, formatter);

System.out.println("Before : " + now);

System.out.println("After : " + formatDateTime);

System.out.println("After : " + formatDateTime.format(formatter));

}

}

### **Get Instant from Clock**

Instant i=Clock.*systemUTC*().instant();

long millis = Clock.systemUTC().millis();

### **From instant to localDateTime**

Instant instant = Instant.now();

System.out.println("Instant : " + instant);

LocalDateTime ldt = LocalDateTime.ofInstant(instant, ZoneOffset.UTC);

LocalDateTime localDateTime = instant.atZone(defaultZoneId).toLocalDateTime();

ZonedDateTime zonedDateTime = instant.atZone(defaultZoneId);

### **From Local Date time to Instant**

Instant instant = date.toInstant();

### **Convert Date to LocalDate**

import java.time.\*;

import java.util.Date;

public class DateToJavaTime {

public static void main(String[] args) {

Date date = new Date();

//Convert to LocalDate

LocalDate localDate = date.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();

System.out.println(localDate.toString());

//Convert to LocalDateTime

LocalDateTime localDateTime = date.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime();

System.out.println(localDateTime);

}

### **Temporal Adjusters**

LocalDate with1 = localDate.with(TemporalAdjusters.lastDayOfMonth());

LocalDate with2 = localDate.with(TemporalAdjusters.next(DayOfWeek.MONDAY));

### **ZonedDateTime**

withZoneSameInstant() method

* The **java.time.ZonedDateTime.withZoneSameInstant**(ZoneId zone) method returns a copy of the date-time with a different time-zone, retaining the instant.
* Following is the declaration for **java.time.ZonedDateTime.withZoneSameInstant**(ZoneId zone) method.
* **public ZonedDateTime withZoneSameInstant**(ZoneId zone)
* The following example program shows the usage of **java.time.ZonedDateTime.withZoneSameInstant**(ZoneId zone) method.

import java.time.ZoneId;

import java.time.ZonedDateTime;

public class ZoneMethod {

public static void main(String[] args) {

ZonedDateTime date = ZonedDateTime.parse("2020-08-28T19:10:38.492+05:30[Asia/Kolkata]");

ZonedDateTime result = date.withZoneSameInstant(ZoneId.of("Asia/Tokyo"));

System.out.println(result);

}

}

The output is

The same instant fot the given zone Asia/Tokyo is : 2020-08-28T22:40:38.492+09:00[Asia/Tokyo]

Just add zone to the current Date and time

ZoneId zoneId=ZoneId.of("Asia/Singapore");  
ZonedDateTime zdt=ZonedDateTime.of(LocalDateTime.now(),zoneId);  
System.out.println(zdt);

### **Add few days/months to localDateTime**

LocalDate ldt = LocalDate.*parse*(stringInputDate, dtf);  
ldt= ldt.plus(days, ChronoUnit.*DAYS*)  
 .plus(months, ChronoUnit.*MONTHS*)  
.plus(years,ChronoUnit.*YEARS*);

### **Copying the date**

LocalDate date = LocalDate.of(year,month,1);

LocalDate firstOfWeek = LocalDate.from(date);

### Difference between 2 dates

LocalDate date1 = LocalDate.now();

LocalDate date2 = date1.plus(1, ChronoUnit.MONTHS);

Period period = Period.between(date2, date1);

## Code samples

### Get First Sunday of any year august

**import** **static** java.time.temporal.TemporalAdjusters.firstInMonth;

**public** **class** FirstSundayOfTheYear {

public static String friendShipDay(int inputYear) {  
 DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("dd-MM-yyyy");

LocalDate august = LocalDate.*of*(inputYear, Month.*AUGUST*, 1);  
 LocalDate with = august.with(TemporalAdjusters.*firstInMonth*(DayOfWeek.*SUNDAY*));  
  
 return with.format(dtf);  
}  
public static void main(String[] args) {  
 System.*out*.println(*friendShipDay*(2022));  
}

### **Now whats time in US, UK,Australia**

Given a certain in india may be like – 12 june-1994 – 9:36 AM and what that’s time

Adi India time, a second ki US lo time entha., and UK lo time entha

*//Build DATE and time with asia time zone.*DateTimeFormatter dtf=DateTimeFormatter.*ofPattern*("yyyy-MM-dd HH:mm:ss");  
LocalDateTime parse = LocalDateTime.*parse*("2020-08-27 10:40:50",dtf);  
ZonedDateTime zonedDT= parse.atZone(ZoneId.*of*("Asia/Calcutta"));  
  
*//aa samayam lo america lo time entha / ade samayanaki london lo time entha*

*// santu puttindi 29 october 1996 7:00 pm IST apudu US lo time entha*ZonedDateTime zonedDateTime = zonedDT.withZoneSameInstant(ZoneId.*of*("Canada/Central"));  
System.*out*.println(zonedDateTime);

API

## TemporalAdjuster

**All Known Implementing Classes:**

[DayOfWeek](https://docs.oracle.com/javase/8/docs/api/java/time/DayOfWeek.html)[Instant](https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html),  [LocalDate](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html" \o "class in java.time), [LocalDateTime](https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html), [LocalTime](https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html" \o "class in java.time) , [Month](https://docs.oracle.com/javase/8/docs/api/java/time/Month.html), [MonthDay](https://docs.oracle.com/javase/8/docs/api/java/time/MonthDay.html" \o "class in java.time), [OffsetDateTime](https://docs.oracle.com/javase/8/docs/api/java/time/OffsetDateTime.html" \o "class in java.time), [OffsetTime](https://docs.oracle.com/javase/8/docs/api/java/time/OffsetTime.html" \o "class in java.time),  [Year](https://docs.oracle.com/javase/8/docs/api/java/time/Year.html), [YearMonth](https://docs.oracle.com/javase/8/docs/api/java/time/YearMonth.html" \o "class in java.time), [ZoneOffset](https://docs.oracle.com/javase/8/docs/api/java/time/ZoneOffset.html" \o "class in java.time)

# Repeatable annotations

public class App {

@Repeatable(value = Cars.class )

public @interface Manufacturer {

String value();

};

/cars will just have manufacturer array

@Retention( RetentionPolicy.RUNTIME )

public @interface Cars {

Manufacturer[] value() default{};

}

@Manufacturer( "Mercedes Benz")

@Manufacturer( "Toyota")

@Manufacturer( "BMW")

@Manufacturer( "Range Rover")

public interface Car {

}

semaphores