**Features of Java 8**

Lambda expression:

* It is an anonymous function/ Nameless
* Without return type
* Without modifiers
* Eg. (int a, int b)-> system.out.println(a+b); or (a, b)-> system.out.println(a+b);
* Eg. (int n)->{return n\*n;} , (int n)->n\*n; , (n)->n\*n; n->n\*n; all are optional
* Lambda expression vs Anonymous inner class

Functional Interface:

* Contains single abstract method.
* Used to invoke lambda expression.
* Eg. Runnable : run(), Callable : call(), Comparable : compareTo(), ActionListener : actionPerformed(), consumer: accept();
* Can contains any number of default and static methods but it should contain only one abstract method.
* @FunctionalInterface: optional
* Eg. Public void m1(): -> abstract method.
* Functional interface in case inheritance valid only if child contains one abstract which is derived from parent functional interface.
* supports lambda exp

String Class: Java 8 introduced StringJoiner class:

* A screenshot of a computer code

  AI-generated content may be incorrect.
* Always prefer to use StringBuffer class for cantactination. Use append() then need to convert stringbuilder object using toString(). Eg String s= sBuilder.toString(); StringBuffer is Synchronized. [ we can use StringBuilder instead of StringBuffer because StringBuilder is faster than StringBuffer.]
* StringJoiner : Used to contactination, also it built with separator, we can also add prefix and a postfix, join()
* A screenshot of a computer code

  AI-generated content may be incorrect.

DateAndTimeAPI: It is thread safe, Immutable , does not have setter methods

All the classes use [Factory Pattern](https://www.digitalocean.com/community/tutorials/factory-design-pattern-in-java) and [Strategy Pattern](https://www.digitalocean.com/community/tutorials/strategy-design-pattern-in-java-example-tutorial) for better handling. Once you have used the methods in one of the classes, working with other classes won’t be hard.

-Old Date time api lacks of following things :

* Not thread safe :
* Mutable
* Two Date related classes:

java.util.Date[Which handles machine related Date classes] ,

java.sql.Date [ Which handles sql / database related date classes]

It creates confusion while converting one date object to another.

-Java 8 Date Time API consists of following packages.

1. **java.time**: This is the base package of the new Java Date Time API. All the major base classes are part of this package, such as LocalDate, LocalTime, LocalDateTime, Instant, Period, Duration, etc. All of these classes are immutable and thread-safe. Most of the time, these classes will be sufficient for handling common requirements.
2. **java.time.chrono**: This package defines generic APIs for non-ISO calendar systems. We can extend AbstractChronology class to create our own calendar system.
3. **java.time.format**: This package contains classes used for formatting and parsing date-time objects. Most of the time we would not be directly using them because of principle classes in java.time package provides formatting and parsing methods.
4. **java.time.temporal**: This package contains temporal objects and we can use it to find out the specific dates or times related to the date/time objects. For example, we can use these to find out the first or last day of the month. You can identify these methods easily because they always have the format “withXXX”.
5. **java.time.zone Package**: This package contains classes for supporting different time zones and their rules.

-java.time: Java 8 introduced , It includes classes Local, Zoned

-Instant class : is used to represent the specific time **instant** on the current timeline.

-Duration Class: It deals with time based amount of time.

-LocalDate\LocalTime and LocalDateTime class: Use it when time zones are NOT required.

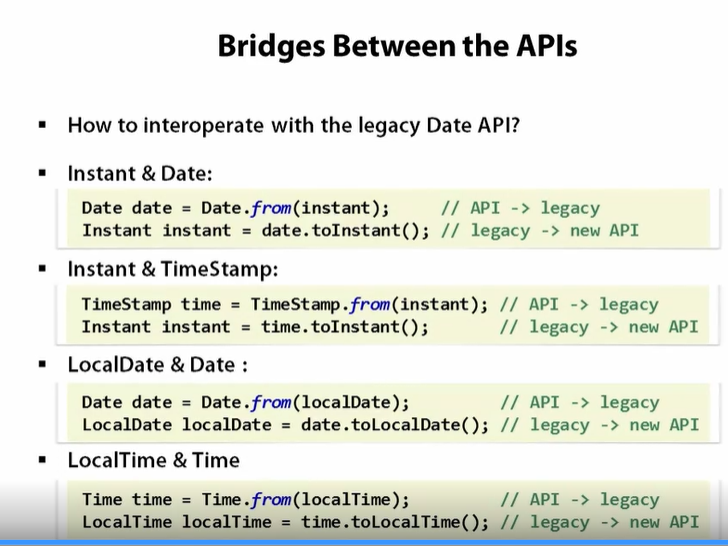
-Period class: deals with date based amount of time

-ZonedDateTime class: Use it when time zones are to be considered.

-DateTimeFormatter: proposes a set of predefined formatters, available as constants.

-**ChronoUnits Enum :**java.time.temporal.ChronoUnit enum is added in Java 8 to replace integer values used in old API to represent day, month etc.

[before java8, while using number to represent month one issue is occurring like if I want user starting for 0 then it 0 will represent 1st month, but if user starts with 1 then it will consider 1 as January.]



Optional Class:

-Before java 8, to avoid null pointer exception developer need to use null checks means if statements to check null pointer. To handle/avoid this scenario, Java 8 introduces Optional class.

-It is like a container which store value, and it has functions which states a value if present. Also represents a state if value is not present it will throws NoSuchElementException.

-Generic type of class, Final class

-container may or may not contain non-null value.

-when to use: when method is created in class which returns results which may or may not be null use there so that whenever any another client wants to access method then it will return correct result.

-no need to use Optional with Instance variables, use with getters and setters

Multithreading:

* Implementing by using Runnable interface or Thread class.

Collections:

* Comparator Interface
* compare(obje1,obj2) method : if obj1 before obj2 then -ve else +ve or if equal then 0.
* compareTo(): for string

Default method(): we can define the method inside interface using default keyword.

Eg: default void show(){…….}

* we can override this method in case of multiple inheritance we need to define method again in child class.
* Parent Class have more power than parent interface
* If we override Object class methods using default then it will give error.
* Introduced to provide backward compatibility so that existing interfaces can use the lambda expressions without implementing the methods in the implementation class.
* Also known as Defender methods, virtual extension methods
* interface has one or multiple implementations, if one or more methods are added to the interface, all the implementations will be forced to implement them too. Otherwise, the design will just break down.
* Default interface methods are an efficient way to deal with this issue. They allow us to add new methods to an interface that are automatically available in the implementations. Therefore, we don’t need to modify the implementing classes.

Static methods():

Eg static void example(){…..}

ForEach loop:

* External loops: for(int i: values){…}
* Internal loops: values.forEach(i-> System.out.println(i));
* allows us to **increase the degree of**[**cohesion**](https://en.wikipedia.org/wiki/Cohesion_(computer_science)) of a design by putting together related methods in one single place without having to create an object.
* static methods in interfaces make it possible to group related utility methods, without having to create artificial utility classes that are simply placeholders for static methods.

Stream API:

* Steam(): it converts normal list method into steams eg values.stream().forEach(System.out.println()), here values is a list name.
* Methods in stream api: Intermediate method-filter(), map() ,Terminate methods- findFirst(), forEach()
* Once stream use then it can not reuse
* used to process collections of objects.
* If we want to process objects from the collection then we used stream.
* Methods in stream api:
* Default natural sorting is implemented by using comparable interface [ compareTo()]

Steps to create->intermediate operation-> terminal operation

Different ways to create a stream

version wise stream functionality

Stream intermediate operations:

filter: used to proccess elements of a stram and return only those that meet a specified condition

-Filter names with length greater than 3

-Filter employees older than 30

map: used to transform each element in a stream into another object or value

- multiply each number by 2 using map

-Extract the names of employees using map

flatMap: used to flatten a stream of collections into a single stream.It is particularly useful when dealing with nested collections

-flatten the list of fruits

-find all the fruit names that start with a from list oflist

sorted: used to sort the elements of a stream

-sort number in natural order

-sort name in reverse order

limit(n): used to truncate a stream to contain at most n elements. useful when you want to process only a subset of a stream

- Get the first 3 elements using limit

skip: used to skip the first n element of a stream and process the remaining elements. useful when you want to skip a certain number of elements at the begining of the stream

-skip the 1st 3 element

- filter even numbers and skip the first 2

peek: used for debugging or performing a non-terminal operation (side-effects) on each element as it flows through the stream

-debuggingthe stream

distinct: used to filter out duplicate elements from stream, ensuring that only unique elements remain. This method works by comparing elements using their equals()

-Removing duplicate string

-Remove duplicate person from list

Date time API:

* Sql and java import confusion
* Old – immutable
* LocalDate.now() – only for date
* LocalTime.now()- only time
* Of() method
* Also see time zone using ZoneId.getAvailableZoneIds();

Predicate Interface:

* Boolean result
* Test() function
* Eg.Predicate<String> checklength -> str.length()>5;

Sysout(checklength.test(“Priyankasahane”));

Consumer Interface: no result

* modifies data
* accept();

Function Interface: input and output

* apply()

Supplier Interface: no input only supply

* get()

Predicate: boolean test( T t);

java.util.function.Consumer; void accept(T t) : use the parameter but not interested in return value

java.util.function.Function; Function<T,R> : R apply(T t): transform the input into an output( types ca be different)

java.util.function.Predicate;

java.util.function.Supplier; T get():when you want to supplya values without any input

BiConsumer: void accept(T t, U u), BiFunction: R apply( T t, U u), BiPredicate: boolean test(T t, U u), UnaryOperator:T apply(T t): transform the input into an output(types are the same), BinaryOperator: T apply(T t1, T t2)

if we used local variable inside lambda then it will treated as final or effectively final it means we can not modify that variable before and after lambda fun declaration