

java8 -- Notepad

File Edit Format View Help

<https://www.geeksforgeeks.org/remove-elements-from-a-list-that-satisfy-given-predicate-in-java/?ref=lbp>

peek() -- a non-terminal operation. It can be useful for debugging or logging, or for performing some kind of side effect on the stream elements

peek will not interrupt the flow, you can continue to operate on the flow later, foreach will interrupt the flow, only traversal

```
Stream.of("apple", "banana", "cherry", "date", "elderberry")
    .peek(s -> System.out.println("Before filter: " + s))
    .filter(s -> s.length() > 5)
    .peek(s -> System.out.println("After filter: " + s))
    .forEach(System.out::println);
```

```
Before filter: apple
Before filter: banana
After filter: banana
banana
Before filter: cherry
After filter: cherry
cherry
Before filter: date
Before filter: elderberryAfter filter: elderberry
elderberry
```

spring cloud config server -- externalize configuration in distributed system
we can read properties from git

The functional interface also known as Single Abstract Method Interface was introduced to facilitate Lambda functions. Since a lambda function can only provide the implementation for 1 method it is mandatory for the functional interface to have ONLY one abstract method.
predicate, function, runnable, callable

=====
marker interface -- no method present --
An interface that does not contain methods, fields, and constants is known as marker interface. In other words, an empty interface is known as marker interface or tag interface. It delivers the run-time type information about an object

Cloneable Interface
Serializable Interface
Remote Interface

Ln 1, Col 1

100%

Windows (CRLF)



*java8-- Notepad

File Edit Format View Help

```
map and filter both are intermediate functions..both will return stream  
filter() method checks a condition and passes the object or value to the next step if the condition is true.  
=====
```

StringJoiner
it is used to construct a sequence of characters separated by a delimiter. Now, you can create string by passing delimiters like comma(,), hyphen(-)

```
StringJoiner joinNames = new StringJoiner(","); // passing comma(,) as delimiter  
// Adding values to StringJoiner  
joinNames.add("Rahul");  
joinNames.add("Raju");  
joinNames.add("Peter");  
joinNames.add("Raheem");  
  
System.out.println(joinNames);  
Rahul,Raju,Peter,Raheem
```

flatMap

```
List<Integer> flatList= number.stream().flatMap(list -> list.stream())  
.collect(Collectors.toList());
```

parallel stream

```
List<String> list = Arrays.asList("Hello ",  
"G", "E", "E", "K", "S!");  
list.parallelStream().forEach(System.out::print);
```

output--ES!KGHello

```
List<String> list= Arrays.asList("Hello ","G", "E", "E", "K", "S!");  
  
// using parallelStream() method for parallel stream  
list.parallelStream().forEachOrdered(System.out::print);
```

output--Hello GEEKS!

Ln 54, Col 1

100%

28°C Sunny



```
*java8 - Notepad
File Edit Format View Help
list<Integer> list = Arrays.asList(3, 4, 6, 12, 20);
boolean ans = list.stream().allMatch(n > n % 3 == 0);
Output: true
Lambda exp
(i, j) -> System.out.println()

The Lambda expression is used to provide the implementation of an interface which has functional interface.
Function interface is used to provide reference to lambda exp.
Function interface only 1 abstract method, many default method. @FunctionalInterface
method reference is replacement of lambda exp.

default method-
public interface InterfaceDemo(){
//void print()
default void print()
{ sysout("123")
}
}

and suppose there r 3 classes
public class InterfaceDemoImpl1 implements InterfaceDemo{}
public class InterfaceDemoImpl2 implements InterfaceDemo{}
public class InterfaceDemoImpl3 implements InterfaceDemo{ create main method here and create obj of InterfaceDemoImpl2 and through taht obj
call print method from InterfaceDemo. }

here all 3 classes r implementing InterfaceDemo. all method wh we ll add in InterfaceDemo it should compulsary present in all 3
implementing classes..if we l] not include any it ll give compilation error. if there r 100 methods then code becomes complex.so if we will add
default in function then no need to add method in all implementing classes and it reduces code.

if we want to change functionality of default method in any class(InterfaceDemoImpl2) then we can override taht function and add our own
logic to achieve that. sysout("123jhkjhhh")

default is not access modifier like public, private, protected. default is keyword
in switch case we ll use default..and it never used in classes/interface
```

Ln 54, Col 1

100% Windows (CRE)



*java8-- Notepad
File Edit Format View Help

default is not access modifier like public, private, protected. default is keyword
in switch case we'll use default..and it never used in classes/interface

we can't use default keyword in classes where its implementing interface. we can use default only in interface.

Intermediate and Terminal Operations?

Intermediate operations are those operations that return Stream itself, allowing for further operations on a stream. An intermediate operation can only process data when there is a terminal operation. Some of the intermediate operations are filter, map and flatMap.

In contrast, terminal operations terminate the pipeline and initiate stream processing. The stream is passed through all intermediate operations during terminal operation call. Terminal operations include forEach, reduce, Collect and sum.

diamond problem due to multiple inheritance---multiple inheritance is not possible in java bcoz of diamond
public interface interfacedemo1{ default void m1() {sysout("m1 interface") } }
public interface interfacedemo2{ default void m1() {sysout("m2 interface") } }
public class interfacedemoImpl1 implements interfacedemo1, interfacedemo2{}

in above condition interfacedemoImpl1 will get confuse wh default method i should call.and diamond problem will come.

to overcome this we can use below

```
public class interfacedemoImpl1 implements interfacedemo1, interfacedemo2{  
    interfacedemo1.super.m1(); //it'll call interfacedemo1 m1 method  
    or interfacedemo2.super.m1(); //it'll call interfacedemo2 m1 method  
    create main method here and create obj of interfacedemoImpl1 and call interfacedemoImpl1.m1()  
}
```

static methods in java 8---

can't overridden, can't inherit, can't overload
if we will define any static method in interface then that method we can't access in its child implementing classes. its only allowed to use in interface only.

but in java 8 it is possible to use static method in implementing classes. for that we have to give,

```
public class interfacedemoImpl1 implements interfacedemo1{  
    interfacedemo1.staticfunctionname() }  
before java 8 we have to create class, create obj and then use static method..but interface doesn't require any constructor, obj creation so performance is high.
```

Ln 54, Col 1 100% Windows (CRLF)

28°C Sunny



*java8-- Notepad

File Edit Format View Help

Predicate in java 8

-isEqual(Object targetRef) : Returns a predicate that tests if two arguments are equal according to Objects.equals(Object, Object).
-and(Predicate other) : Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
-negate() : Returns a predicate that represents the logical negation of this predicate
-or(Predicate other) : Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
-test(T t) : Evaluates this predicate on the given argument.boolean test(T t)

when we ll use filter its default return type is Predicate.

its a predefined functional interface already present in java 8 hving only 1 abstarcet method ,name is test() abstarcet method. it returns boolean value true/false.

to call method we hv to give predicate nam.test()method .

```
public class predicatedemo{  
public static void main(string[] args){  
predicate<string> xyz = s -> s.length >= 5; //return true if lenght is greater than 5  
sysout("length is--"+xyz.test("jfjdfjhfj")); ///jfjdfjhfj is input string and taken in S variable  
}}  
predicate<string>, predicate<int>, predicate<emp>--anything but only 1 argument  
retun type mandatory boolean
```

predicate chaining

```
Predicate<Integer> greaterThanTen = (i) -> i > 10;  
  
// Creating predicate  
Predicate<Integer> lowerThanTwenty = (i) -> i < 20;  
boolean result = greaterThanTen.and(lowerThanTwenty).test(15);  
System.out.println(result);
```

function in java 8

its a predefined functional interface already present in java 8 hving only 1 abstarcet method named apply() abstarcet method.

Ln 54, Col 1

100% Windows (C)



28°C Sunny

*java8-- Notepad
File Edit Format View Help

function in java 8

its a predefined functional interface already present in java 8 having only 1 abstract method named apply() abstract method.
to call method we have to give functionname.apply()method . its return output/object

```
public class Functiondemo{  
    public static void main(String[] args){  
        Function<Integer, Integer> xyz = s -> s*s; //return square of 4..here 1st integer is input var and 2nd integer is return type(output)  
        System.out.println("square is--"+xyz.apply(4)); //4 is input and taken in S variable  
    }  
}
```

to make square of each element of array using stream. it gives all records from list/array

```
List<Integer> arlist = new ArrayList<Integer>();  
arlist.add(1);  
arlist.add(2);  
arlist.add(3);  
arlist.stream().map(i -> i*i).foreach(x -> System.out.println(x));
```

filter is used to filter stream
less no of records

```
arlist.stream().filter(i -> i%2 == 0).foreach(x -> System.out.println(x));
```

Given a list of employees, you need to filter all the employee whose age is greater than 20 and print the employee names

```
List<String> employeeFilteredList = employeeList.stream()  
    .filter(e -> e.getAge() > 20)  
    .map(Employee::getName)  
    .collect(Collectors.toList());
```

how to find duplicate values from list using java 8

data in emp table	srno	emp name	age
	1	asd	23
	2	afgf	21
	3	cbfb	27

Ln 54, Col 1 100% Windows (CRLF)

28°C Sunny

Type here to search

```
arlist.stream().filter(i -> i/2 == 0).foreach(x -> sysout(x));  
-----  
Given a list of employees, you need to filter all the employee whose age is greater than 20 and print the employee names  
List<String> employeeFilteredList = employeeList.stream()  
    .filter(e -> e.getAge() > 20)  
    .map(Employee::getName)  
    .collect(Collectors.toList());
```

how to find duplicate values from list using java 8

data in emp table	srno	emp name	age
	1	asd	23
	2	afgf	21
	3	cbfb	27
	4	asd	20

```
@getmapping(path="/findall")  
public responseentity<> getall(){  
list<string> namesrepo = repo.findall().stream().map(emp -> emp.getname()).collect(Collectors.toList());  
set<string> uniquenames = new HashSet<>();  
set<string> duplicatenames = namesrepo.stream().filter(name -> !uniquenames.add(name)).collect(Collectors.toSet());  
return new responseentity<>(uniquenames ,httpsstatus.ok); //gives asd afgf cbfb as output  
//return new responseentity<>duplicatenames ,httpsstatus.ok); // gives asd as output  
}
```

