* ***asList***
* [@SafeVarargs](https://docs.oracle.com/javase/8/docs/api/java/lang/SafeVarargs.html)

public static <T> [List](https://docs.oracle.com/javase/8/docs/api/java/util/List.html)<T> asList(T... a)

Returns a fixed-size list backed by the specified array. (Changes to the returned list "write through" to the array.) This method acts as bridge between array-based and collection-based APIs, in combination with [Collection.toArray()](https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html#toArray--). The returned list is serializable and implements [RandomAccess](https://docs.oracle.com/javase/8/docs/api/java/util/RandomAccess.html).

This method also provides a convenient way to create a fixed-size list initialized to contain several elements:

List<String> stooges = Arrays.asList("Larry", "Moe", "Curly");

**Type Parameters:**

T - the class of the objects in the array

**Parameters:**

a - the array by which the list will be backed

**Returns:**

a list view of the specified array

Create a List which contains name of city and filter the cities List whose name start with A

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** CityFilterDemoWithOutStream {

**public** **static** List<String> getFilterCities(List<String>cities,String filter){

List<String>filteredCities=**new** ArrayList<String>();

**for**(String city:cities) {

**if**(city.startsWith("A"))

filteredCities.add(city);

}

**return** filteredCities;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<String> cities=Arrays.*asList*("Mumbai","Chennai","Delhi","Kolkata","Hyderabad","Bangalore","Pune","Agra","Ahmedabad");

//filter the city name which starts with A

List<String>filterCities= *getFilterCities*(cities, "A");

System.***out***.println("----All Cities--------");

**for**(String city:cities) {

System.***out***.println(city);

}

System.***out***.println("----Cities Which Starts With A--------");

**for**(String city:filterCities) {

System.***out***.println(city);

}

}

}

----All Cities--------

Mumbai

Chennai

Delhi

Kolkata

Hyderabad

Bangalore

Pune

Agra

Ahmedabad

----Cities Which Starts With A--------

Agra

Ahmedabad

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** java.util.stream.Stream;

**public** **class** CityFilterDemoWithStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<String> cities=Arrays.*asList*("Mumbai","Chennai","Delhi","Kolkata","Hyderabad","Bangalore","Pune","Agra","Ahmedabad");

System.***out***.println("----All Cities--------");

**for**(String city:cities) {

System.***out***.println(city);

}

//Convert list to stream by calling stream() method on collection.

Stream<String> stream=cities.stream();

//filters the City, starts with "A".

//Returns a stream consisting of the elements of this stream that match the given predicate.

stream=stream.filter(city->city.startsWith("A"));

//Collect the output and convert streams to a List.

List<String> filteredCities=stream.collect(Collectors.*toList*());

System.***out***.println("----Cities Which Starts With A--------");

filteredCities.forEach(city->System.***out***.println(city));

}

}

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** java.util.stream.Stream;

**public** **class** CityFilterDemoWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<String> cities=Arrays.*asList*("Mumbai","Chennai","Delhi","Kolkata","Hyderabad","Bangalore","Pune","Agra","Ahmedabad");

List<String> filteredCities= cities.stream() //convert list to stream

.filter(city->city.startsWith("A"))//filters the City, starts with "A".

.collect(Collectors.*toList*()); //collect the output and convert streams to a List

System.***out***.println("---Filtered Cities---");

filteredCities.forEach(city->System.***out***.println(city));

}

}

How to filter the List of Student using Java 8 Stream | Java 8 streams tutorial | Streams in Java 8

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** StudentFilterDemoWithOutStream {

**private** **static** Student findStudentByName(List<Student> studentList, String name) {

Student tempStudent=**null**;

**for**(Student student:studentList) {

**if**(student.getName().equalsIgnoreCase(name))

tempStudent=student;

}

**return** tempStudent;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Student> studentList=Arrays.*asList*(**new** Student(101, "John", "CS"),

**new** Student(102, "Mark", "IT"),

**new** Student(103, "Scott", "CS"),

**new** Student(104, "Sachin", "ME"),

**new** Student(105, "Peter", "EC"));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter Searched Student Name");

String name=sc.next();

Student student=*findStudentByName*(studentList, name);

**if**(student!=**null**)

System.***out***.println(student);

**else**

System.***out***.println("Student Not Existed!");

}

}

Enter Searched Student Name

john

Student [rollno=101, name=John, branch=CS]

##### [--> [Open Declaration](eclipse-open:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream(Stream.class%E2%98%83Stream~findAny)](eclipse-open:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream(Stream.class%E2%98%83Stream~findAny)[Optional](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream(Stream.class%E2%98%83Stream~findAny%E2%98%82java.util.Optional)<[Student](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream(Stream.class%E2%98%83Stream~findAny%E2%98%82Student)> [java](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava).[util](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util).[stream](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream).[Stream](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util.stream(Stream.class%E2%98%83Stream).findAny()

#### findAny

[Optional](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html)<[T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)> findAny()

Returns an [Optional](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html) describing some element of the stream, or an empty Optional if the stream is empty.

This is a [short-circuiting terminal operation](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps).

The behavior of this operation is explicitly nondeterministic; it is free to select any element in the stream. This is to allow for maximal performance in parallel operations; the cost is that multiple invocations on the same source may not return the same result. (If a stable result is desired, use [findFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#findFirst--) instead.)

**Returns:**

an Optional describing some element of this stream, or an empty Optional if the stream is empty

**Throws:**

[NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) - if the element selected is null

**See Also:**

[findFirst()](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#findFirst--)

##### [--> [Open Declaration](eclipse-open:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util(Optional.class%E2%98%83Optional~orElse~TT;)](eclipse-open:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util(Optional.class%E2%98%83Optional~orElse~TT;)[Student](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util(Optional.class%E2%98%83Optional~orElse~TT;%E2%98%82Student) [java](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava).[util](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util).[Optional](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util(Optional.class%E2%98%83Optional).orElse([Student](eclipse-javadoc:%E2%98%82=StreamProject/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_131%5C/lib%5C/rt.jar%3Cjava.util(Optional.class%E2%98%83Optional~orElse~TT;%E2%98%82Student) other)

#### orElse

public [T](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html) orElse([T](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html) other)

Return the value if present, otherwise return other.

**Parameters:**

other - the value to be returned if there is no value present, may be null

**Returns:**

the value, if present, otherwise other

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** StudentFilterDemoWithStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Student> studentList=Arrays.*asList*(**new** Student(101, "John", "CS"),

**new** Student(102, "Mark", "IT"),

**new** Student(103, "Scott", "CS"),

**new** Student(104, "Sachin", "ME"),

**new** Student(105, "Peter", "EC"));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter Searched Student Name");

String name=sc.next();

Student student=studentList.stream() ////convert list to stream

.filter(s->s.getName().equalsIgnoreCase(name))

.findAny() // If 'findAny' then return found

.orElse(**null**); //If not found, return null

System.***out***.println(student);

}

}

Enter Searched Student Name

peter

Student [rollno=105, name=Peter, branch=EC]

How to convert Stream of Student to Stream of String using map method of Java 8 Stream

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

#### map

<R> [Stream](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)<R> map([Function](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)<? super [T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html),? extends R> mapper)

Returns a stream consisting of the results of applying the given function to the elements of this stream.

This is an [intermediate operation](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps).

**Type Parameters:**

R - The element type of the new stream

**Parameters:**

mapper - a [non-interfering](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#NonInterference), [stateless](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Statelessness) function to apply to each element

**Returns:**

the new stream

find the Student Branch Name by providing Student Name

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** StudentFilterDemoWithStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Student> studentList=Arrays.*asList*(**new** Student(101, "John", "CS"),

**new** Student(102, "Mark", "IT"),

**new** Student(103, "Scott", "CS"),

**new** Student(104, "Sachin", "ME"),

**new** Student(105, "Peter", "EC"));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter Searched Student Name");

String name=sc.next();

String branch=studentList.stream() //convert list to stream

.filter(s->s.getName().equalsIgnoreCase(name))

.map(Student::getBranch)//convert Stream<Student> to Stream<String>

.findAny() // If 'findAny' then return found

.orElse(**null**); //If not found, return null

System.***out***.println(branch);

}

}

filter the Products Details with and without Java 8 Stream | Java 8 streams tutorial

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** ProductFilterWithOutStream {

//Filter the product, whose price is less than 20000

**static** List<Product> findProductDetails(List<Product>productList,**double** price)

{

List<Product> filteredProduct=**new** ArrayList<Product>();

**for**(Product product:productList) {

**if**(product.getPrice()<price)

filteredProduct.add(product);

}

**return** filteredProduct;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the Price");

**double** price=sc.nextDouble();

List<Product>filteredProductList=*findProductDetails*(productList, price);

**for**(Product product:filteredProductList) {

System.***out***.println(product);

}

}

}

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the Price");

**double** price=sc.nextDouble();

List<Product> filteredProductList=productList.stream() //convert list to stream

.filter(product->product.getPrice()<price) // Filter the product, whose price is less than given price

.collect(Collectors.*toList*());// collecting as list

filteredProductList.forEach(product->System.***out***.println(product));

}

}

Enter the Price

30000

Product [id=1, name=RealMe Mobile, price=15000.0]

Product [id=2, name=Samsung Mobile, price=20000.0]

Product [id=4, name=Nokia Mobile, price=12000.0]

Product [id=5, name=Oppo Mobile, price=10000.0]

List out all Product Name whose price is less than given price.

.

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**static** List<String> findProductDetails(List<Product>productList,**double** price)

{

List<String> filteredProductNameList=**new** ArrayList<String>();

**for**(Product product:productList) {

**if**(product.getPrice()<price)

filteredProductNameList.add(product.getName());

}

**return** filteredProductNameList;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the Price");

**double** price=sc.nextDouble();

List<String> productNameList=*findProductDetails*(productList, price);

System.***out***.println(productNameList);

}

}

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the Price");

**double** price=sc.nextDouble();

List<String> filteredProductNameList=productList.stream() //convert list to stream

.filter(product->product.getPrice()<price) // Filter the product, whose price is less than given price

.map(product->product.getName()) // fetching Name

.collect(Collectors.*toList*());// collecting as list

System.***out***.println(filteredProductNameList);

}

}

Enter the Price

30000

[RealMe Mobile, Samsung Mobile, Nokia Mobile, Oppo Mobile]

Count all product whose price is given price.

#### count

long count()

Returns the count of elements in this stream. This is a special case of a [reduction](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Reduction) and is equivalent to:

return mapToLong(e -> 1L).sum();

This is a [terminal operation](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps).

**Returns:**

the count of elements in this stream

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter the Price");

**double** price=sc.nextDouble();

**long** countOfProducts =productList.stream() //convert list to stream

.filter(product->product.getPrice()<price) // Filter the product, whose price is less than given price

.count(); // Returns the count of elements in this stream.

System.***out***.println("CountOfProducts = " + countOfProducts);

}

}

Enter the Price

30000

CountOfProducts = 4

finding the sum of all the product price using reduce method of Java 8 Stream

#### reduce

[Optional](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html)<[T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)> reduce([BinaryOperator](https://docs.oracle.com/javase/8/docs/api/java/util/function/BinaryOperator.html" \o "interface in java.util.function)<[T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)> accumulator)

Performs a [reduction](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Reduction) on the elements of this stream, using an [associative](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Associativity) accumulation function, and returns an Optional describing the reduced value, if any. This is equivalent to:

boolean foundAny = false;

T result = null;

for (T element : this stream) {

if (!foundAny) {

foundAny = true;

result = element;

}

else

result = accumulator.apply(result, element);

}

return foundAny ? Optional.of(result) : Optional.empty();

but is not constrained to execute sequentially.

The accumulator function must be an [associative](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Associativity) function.

This is a [terminal operation](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps).

**Parameters:**

accumulator - an [associative](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Associativity), [non-interfering](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#NonInterference), [stateless](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Statelessness) function for combining two values

**Returns:**

an [Optional](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html) describing the result of the reduction

**Throws:**

[NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) - if the result of the reduction is null

**See Also:**

[reduce(Object, BinaryOperator)](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#reduce-T-java.util.function.BinaryOperator-), [min(Comparator)](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#min-java.util.Comparator-), [max(Comparator)](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#max-java.util.Comparator-)

reduce method takes a sequence of input elements

\* and combines them into a single summary result

\* by repeated operation

Using reduce method we can perform Sum, min, max, average,

\* and string concatenation.

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

Double totalPrice = productList.stream()

.map(product->product.getPrice())

.reduce(0.0,(sum, price)->sum+price); // accumulating price

System.***out***.println("Sum of All Product Price ="+totalPrice);

}

}

**Terminal operations[min and max] in Java 8 Stream**

#### max

[Optional](https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html)<[T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)> max([Comparator](https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html)<? super [T](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)> comparator)

Returns the maximum element of this stream according to the provided Comparator. This is a special case of a [reduction](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Reduction).

This is a [terminal operation](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#StreamOps).

**Parameters:**

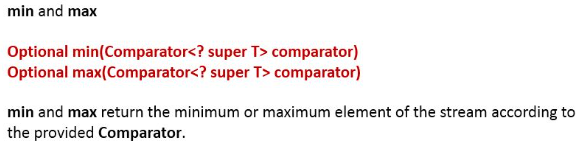
comparator - a [non-interfering](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#NonInterference), [stateless](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html#Statelessness) Comparator to compare elements of this stream

**Returns:**

an Optional describing the maximum element of this stream, or an empty Optional if the stream is empty

**Throws:**

[NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) - if the maximum element is null



Stream.max() returns the maximum element of the stream based on the provided Comparator. A Comparator is a comparison function

**Exception :** This method throws ***NullPointerException*** if the maximum element is null.

**import** java.util.ArrayList;

**public** **class** StreamDemo5 {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

ArrayList<Integer>numberList=**new** ArrayList<Integer>();

numberList.add(10);

numberList.add(20);

numberList.add(45);

numberList.add(2);

numberList.add(0);

numberList.add(50);

System.***out***.println("number List: "+numberList);

Integer i1=numberList.stream().min((n1,n2)->n1.compareTo(n2)).get();

System.***out***.println("Minimum value in List as Asc Order : "+i1);

Integer i2=numberList.stream().min((n1,n2)->-n1.compareTo(n2)).get();

System.***out***.println("Minimum value in List as Desc Order : "+i2);

Integer i3=numberList.stream().max((n1,n2)->n1.compareTo(n2)).get();

System.***out***.println("Max value in List as Asc Order : "+i3);

Integer i4=numberList.stream().max((n1,n2)->-n1.compareTo(n2)).get();

System.***out***.println("Max value in List as Desc Order : "+i4);

}

}

finding the max produce price or min product price using Java 8 Stream | Java 8 streams tutorial

**public** **class** Product {

**private** **int** id;

**private** String name;

**private** **double** price;

**public** Product(**int** id, String name, **double** price) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.price = price;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **double** getPrice() {

**return** price;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

@Override

**public** String toString() {

**return** "Product [id=" + id + ", name=" + name + ", price=" + price + "]";

}

}

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 12000),

**new** Product(5, "Oppo Mobile", 10000));

// max() method to get max Product price

Product maxPriceProduct=productList.stream()

.max((prod1,prod2)->prod1.getPrice()>prod2.getPrice()?1:-1).get();

System.***out***.println("Max price product = "+maxPriceProduct);

// min() method to get min Product price

Product minPriceProduct = productList.stream()

.min((product1, product2)->

product1.getPrice() > product2.getPrice() ? 1: -1).get();

System.***out***.println("Min price product = "+minPriceProduct);

}

}

Max price product = Product [id=3, name=Sony Mobile, price=30000.0]

Min price product = Product [id=5, name=Oppo Mobile, price=10000.0]

filter the productList and get the result as a Set using Java 8 Stream | Java 8 streams

List out all product unique price based on given price

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** java.util.Set;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 10000),

**new** Product(5, "Oppo Mobile", 10000));

Scanner sc=**new** Scanner(System.***in***);

System.***out***.println("Enter Price");

**double** price=sc.nextDouble();

Set<Double> productPriceSet =productList.stream()//convert list to stream

.filter(product->product.getPrice()<price)// filter product on the base of price

.map(product->product.getPrice())

.collect(Collectors.*toSet*());//collect it as Set(remove duplicate elements)

System.***out***.println(productPriceSet);

}

}

Enter Price

30000

[20000.0, 10000.0, 15000.0]

**Converting the productList to map |Product price act as key and name act as value in map|using Java 8 Stream | Java 8 streams tutorial | Java 8 streams**

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.Optional;

**import** java.util.Scanner;

**import** java.util.Set;

**import** java.util.stream.Collectors;

**public** **class** ProductFilterWithOutStream {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Product>productList=Arrays.*asList*(**new** Product(1, "RealMe Mobile", 15000),

**new** Product(2, "Samsung Mobile", 20000),

**new** Product(3, "Sony Mobile", 30000),

**new** Product(4, "Nokia Mobile", 10000),

**new** Product(5, "Oppo Mobile", 10000));

//Converting Product List into a Map

Map<String, Double> productPriceMap = productList.stream()

.collect(Collectors.*toMap*(p -> p.getName(), p -> p.getPrice()));

productPriceMap.forEach((k,v)->System.***out***.println(k+" : "+v));

}

}

Nokia Mobile : 10000.0

Oppo Mobile : 10000.0

RealMe Mobile : 15000.0

Sony Mobile : 30000.0

Samsung Mobile : 20000.0

find out the unique cities names in upper case of all student

**public** **class** Address {

**private** String country;

**private** String city;

**public** Address(String country, String city) {

**super**();

**this**.country = country;

**this**.city = city;

}

**public** String getCountry() {

**return** country;

}

**public** **void** setCountry(String country) {

**this**.country = country;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

**public** **class** Student {

**int** rollno;

String name;

**int** age;

Address address;

**public** Student(**int** rollno, String name, **int** age, Address address) {

**super**();

**this**.rollno = rollno;

**this**.name = name;

**this**.age = age;

**this**.address = address;

}

**public** **int** getRollno() {

**return** rollno;

}

**public** **void** setRollno(**int** rollno) {

**this**.rollno = rollno;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** Address getAddress() {

**return** address;

}

**public** **void** setAddress(Address address) {

**this**.address = address;

}

}

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

import java.util.stream.Collectors;

public class StudentFilterDemoWithOutStream {

private static Student findStudentByName(List<Student> studentList, String name) {

Student tempStudent=null;

for(Student student:studentList) {

if(student.getName().equalsIgnoreCase(name))

tempStudent=student;

}

return tempStudent;

}

public static void main(String[] args) {

// TODO Auto-generated method stub

List<Student> studentList=Arrays.asList(new Student(101, "John", 30,

new Address("USA", "New York")),

new Student(102, "Mark", 40,

new Address("USA", "Houston")),

new Student(103, "Scott", 50,

new Address("germany", "Berlin")),

new Student(104, "Sachin", 20,

new Address("India", "Delhi")),

new Student(105, "Peter", 35,

new Address("germany", "Hamburg")));

List<String>countryNameList=studentList.stream()

.map(student->student.getAddress())// Stream<Student> to Stream<Address>

.map(address->address.getCountry())// Stream<Address> to Stream<Name>

.map(String::toUpperCase)// Convert name as upper case

.distinct()// Get the unique elements[i.e. name]

.collect(Collectors.toList()); // collect the result as a list

System.out.println(countryNameList);

}

}

Listout those students which belongs to particular country?

Show the unique names of all the students who belong in the given country.

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

import java.util.stream.Collectors;

public class StudentFilterDemoWithOutStream {

public static void main(String[] args) {

// TODO Auto-generated method stub

List<Student> studentList=Arrays.asList(new Student(101, "John", 30,

new Address("USA", "New York")),

new Student(102, "Mark", 40,

new Address("USA", "Houston")),

new Student(103, "Scott", 50,

new Address("germany", "Berlin")),

new Student(104, "Sachin", 20,

new Address("India", "Delhi")),

new Student(105, "Peter", 35,

new Address("germany", "Hamburg")));

Scanner sc=new Scanner(System.in);

System.out.println("Enter country Name");

String cntryname=sc.next();

List<String>studentNameList=studentList.stream() //convert list to stream

.filter(student->student.getAddress().getCountry().equalsIgnoreCase(cntryname))// Filter student based on country

.map(student->student.getName())//Stream<Student> to Stream<Name>

.map(String::toUpperCase)// Convert name as upper case

.distinct()// Get the unique elements[i.e. name]

.collect(Collectors.toList());// collect the result as a list

System.out.println(studentNameList);

}

}