Java Collectors

**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 + "]";

}

}

Java Collectors Example: Fetching data as a List

**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

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

.map(prod->prod.getName())//fetching the name of product

.collect(Collectors.*toList*());//collecting as a List

prodListName.forEach(name->System.***out***.println(name));

}

}

Java Collectors Example: Fetching data as a Set

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]

Java Collectors Example: Fetching data as a Map

**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** FetchingDataAsMap {

**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));

}

}

Java Collectors Example: using sum method

Finding sum of all product price

#### summingDouble

public static <T> [Collector](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)<T,?,[Double](https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html)> summingDouble([ToDoubleFunction](https://docs.oracle.com/javase/8/docs/api/java/util/function/ToDoubleFunction.html)<? super T> mapper)

Returns a Collector that produces the sum of a double-valued function applied to the input elements. If no elements are present, the result is 0.

The sum returned can vary depending upon the order in which values are recorded, due to accumulated rounding error in addition of values of differing magnitudes. Values sorted by increasing absolute magnitude tend to yield more accurate results. If any recorded value is a NaN or the sum is at any point a NaN then the sum will be NaN.

**Type Parameters:**

T - the type of the input elements

**Parameters:**

mapper - a function extracting the property to be summed

**Returns:**

a Collector that produces the sum of a derived property

**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));

Double sumPrices = productList.stream() //convert list to stream

.collect(Collectors.*summingDouble*(prod->prod.getPrice()));

System.***out***.println("Sum of prices: "+sumPrices);

}

}

Java Collectors Example: Getting Product Average Price

#### averagingDouble

public static <T> [Collector](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collector.html)<T,?,[Double](https://docs.oracle.com/javase/8/docs/api/java/lang/Double.html)> averagingDouble([ToDoubleFunction](https://docs.oracle.com/javase/8/docs/api/java/util/function/ToDoubleFunction.html)<? super T> mapper)

Returns a Collector that produces the arithmetic mean of a double-valued function applied to the input elements. If no elements are present, the result is 0.

The average returned can vary depending upon the order in which values are recorded, due to accumulated rounding error in addition of values of differing magnitudes. Values sorted by increasing absolute magnitude tend to yield more accurate results. If any recorded value is a NaN or the sum is at any point a NaN then the average will be NaN.

**Implementation Note:**

The double format can represent all consecutive integers in the range -253 to 253. If the pipeline has more than 253 values, the divisor in the average computation will saturate at 253, leading to additional numerical errors.

**Type Parameters:**

T - the type of the input elements

**Parameters:**

mapper - a function extracting the property to be summed

**Returns:**

a Collector that produces the sum of a derived property

**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));

Double average = productList.stream() //convert list to stream

.collect(Collectors.*averagingDouble*(prod->prod.getPrice()));

System.***out***.println("Average price is: "+average);

}

}

**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));

Long noOfElements = productList.stream()

.collect(Collectors.*counting*());

System.***out***.println("Total elements : "+noOfElements);

}

}