**Summing Java Streams Api**

summing java streams api

Pipelines and streams are introduced in **Java SE 8** and it enriches the Java collections API. Java Stream API package is to support **functional style operations** such as map-reduce transformations on collections. To get full documentation click [this](https://docs.oracle.com/javase/8/docs/api/java/util/stream/package-summary.html). Understanding the way stream api works **Lambda expressions** in java is prerequisite.

**Stream**

*Stream is a sequence of elements supporting sequential and parallel aggregate operations.*

**Operations** — **Intermediate and terminal** are two types of operations in a stream. Intermediate operation returns a **new stream** and terminal operation **consumes a stream**.

**Example Stream**

System.out.println("Stream Api");  
List<String> names = Arrays.asList("Abdallah", "Mahmoud", "Ahmed");  
 names.stream().filter(name -> name.length() > 5)  
 .forEach(System.out::println);

**Output:**

Stream Api  
Ahmed

**Java Streams Operations**  
Java stream operation are of two types intermediate and terminal:

**Intermediate Operations** : *returns a new stream*

* map()
* filter()
* sorted()
* distinct()
* limit()

**Terminal Operations**:*consumes the stream pipeline*

* sum()
* count()
* average()
* collect()
* reduce()

To read more about operations in stream api click [here](https://javapapers.com/java/java-stream-api/#:~:text=Operations%20%E2%80%93%20Intermediate%20and%20terminal%20are,terminal%20operation%20consumes%20a%20stream) or java streams api [here](https://www.baeldung.com/java-8-streams).

*reduce():*

It applies a binary operator (accumulator) to each element in the stream, where the first operand is the return value of the previous application, and the second one is the current stream element.

**Primitive Examples**

public Integer getSumByUsingReduce(List<Integer> integers) {  
 return integers.stream().reduce(0, (a, b) -> a + b);  
 }

*Using Java Accumulator*

public Integer getSumByUsingJavaAccumulator(  
List<Integer> integers) {  
 return integers.stream().reduce(0, Integer::sum);  
 }

**SumUtils class**

public class SumUtils {  
 public static int add(int a, int b){  
 return a+b;  
 }  
}

*Using Customized Accumulator*

public Integer getSumByUsingCustomizedAccumulator(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .reduce(0, SumUtils::add);  
 }

*collect():*

public Integer getSumByUsingSum(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .mapToInt(Integer::intValue)  
 .sum();

*MapValues:*

public Integer getSumByUsingMapValues(  
 Map<Integer, Integer> map) {  
 return map  
 .values()  
 .stream()  
 .mapToInt(Integer::valueOf).sum();  
 }

*Extracting:*

public Integer getSumByExtractingIntegersFromString(  
 String str) {  
  
 return Arrays.stream(str.split(" "))  
 .filter((s) -> s.matches("\\d+"))  
 .mapToInt(Integer::valueOf)  
 .sum();  
 }

**StreamSumByUsingPrimitive Class**

package com.riigsoft.streamapi.sum;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.Map;  
import java.util.stream.Collectors;  
  
public class StreamSumByUsingPrimitive {  
  
 public Integer getSumByUsingReduce(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .reduce(0, (a, b) -> a + b);  
 }  
  
 public Integer getSumByUsingCollect(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .collect(Collectors  
 .summingInt(Integer::intValue));  
 }  
  
 public Integer getSumByUsingJavaAccumulator(  
 List<Integer> integers) {  
 return integers.stream().reduce(0, Integer::sum);  
 }  
  
 public Integer getSumByUsingCustomizedAccumulator(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .reduce(0, SumUtils::add);  
 }  
  
 public Integer getSumByUsingSum(  
 List<Integer> integers) {  
 return integers  
 .stream()  
 .mapToInt(Integer::intValue)  
 .sum();  
 }  
  
 public Integer getSumByUsingMapValues(  
 Map<Integer, Integer> map) {  
 return map  
 .values()  
 .stream()  
 .mapToInt(Integer::valueOf).sum();  
 }  
  
 public Integer getSumByExtractingIntegersFromString(  
 String str) {  
  
 return Arrays.stream(str.split(" "))  
 .filter((s) -> s.matches("\\d+"))  
 .mapToInt(Integer::valueOf)  
 .sum();  
 }  
}

**Object Examples**

*Product Class*

package com.riigsoft.streamapi.sum;  
  
public class Product {  
 private int id;  
 private int price;  
  
 public Product(int id, int price) {  
 this.id = id;  
 this.price = price;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public void setId(int id) {  
 this.id = id;  
 }  
  
 public int getPrice() {  
 return price;  
 }  
  
 public void setPrice(int price) {  
 this.price = price;  
 }  
}

**Calculating sum of product prices**:  
*reduce():*

public Integer getSumByUsingReduce(List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, (a, b) -> a + b);  
 }

*Reduce with Reference*

public Integer getSumByUsingReduceWithMethodReference(  
 List<Product> products) {  
 return products.stream()  
 .map(Product::getPrice)  
 .reduce(0, (a, b) -> a + b);  
 }

*collect():*

public Integer getSumByUsingCollect(List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .collect(Collectors.summingInt(Integer::intValue));

*Java Accumulator*

public Integer getSumByUsingJavaAccumulator(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, Integer::sum);  
 }

*Customized Accumulator*

public Integer getSumByUsingCustomizedAccumulator(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, SumUtils::add);  
 }

*sum():*

public Integer getSumByUsingSum(List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .mapToInt(Integer::intValue).sum();  
 }

**StreamSumByUsingObject Class**

package com.riigsoft.streamapi.sum;import java.util.List;  
import java.util.stream.Collectors;public class StreamSumByUsingObject {  
 public Integer getSumByUsingReduce(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, (a, b) -> a + b);  
 }  
 public Integer getSumByUsingReduceWithMethodReference(  
 List<Product> products) {  
 return products.stream()  
 .map(Product::getPrice)  
 .reduce(0, (a, b) -> a + b);  
 } public Integer getSumByUsingCollect(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .collect(Collectors  
 .summingInt(Integer::intValue));  
 } public Integer getSumByUsingJavaAccumulator(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, Integer::sum);  
 } public Integer getSumByUsingCustomizedAccumulator(  
 List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .reduce(0, SumUtils::add);  
 }  
 public Integer getSumByUsingSum(List<Product> products) {  
 return products.stream()  
 .map(product -> product.getPrice())  
 .mapToInt(Integer::intValue).sum();  
 }}