Java 8 Stream

← Prev Next →

Java provides a new additional package in Java 8 called java.util.stream. This package consists of classes, interfaces and enum to allows functional-style operations on the elements. You can use stream by importing java.util.stream package.

Stream provides following features:

- Stream does not store elements. It simply conveys elements from a source such as a data structure, an array, or an I/O channel, through a pipeline of computational operations.
- Stream is functional in nature. Operations performed on a stream does not modify it's source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection.
- Stream is lazy and evaluates code only when required.
- The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source.

You can use stream to filter, collect, print, and convert from one data structure to other etc. In the following examples, we have apply various operations with the help of stream.

Java Stream Interface Methods

Methods	Description
boolean allMatch(Predicate<? super T> predicate)	It returns all elements of this stream which match the provided predicate. If the stream is empty then true is returned and the predicate is not evaluated.
boolean anyMatch(Predicate<? super T> predicate)	It returns any element of this stream that matches the provided predicate. If the stream is empty then false is returned and the predicate is not evaluated.
static <T> Stream.Builder<T> builder()	It returns a builder for a Stream.
<R,A> R collect(Collector<? super T,A,R> collector)	It performs a mutable reduction operation on the elements of this stream using a Collector. A Collector encapsulates the functions used as arguments to collect(Supplier, BiConsumer, BiConsumer), allowing for reuse of collection strategies and composition of collect operations such as multiple-level grouping or partitioning.
<R> R collect(Supplier<R> supplier, BiConsumer<R,? super T> accumulator, BiConsumer<R,R> combiner)	It performs a mutable reduction operation on the elements of this stream. A mutable reduction is one in which the reduced value is a mutable result container, such as an ArrayList, and elements are incorporated by updating the state of the result rather than by replacing the result.
static <T> Stream<T> concat(Stream<? extends T> a, Stream<? extends T> b)	It creates a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream. The resulting stream is ordered if both of the input streams are ordered, and parallel if either of the input streams is parallel. When the resulting stream is closed, the close handlers for both input streams are invoked.
long count()	It returns the count of elements in this stream. This is a special case of a reduction.
Stream<T> distinct()	It returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.
static <T> Stream<T> empty()	It returns an empty sequential Stream.
Stream<T> filter(Predicate<? super T> predicate)	It returns a stream consisting of the elements of this stream that match the given predicate.
Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)	It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)	It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)	It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping
predicate)	predicate.
Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be

	returned.
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)	It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)	It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)	It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper) predicate)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping predicate.
Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.

<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper) | It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper) | It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper) | It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper) predicate) | It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping predicate.
Optional<T> findAny() | It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst() | It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.

<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper) | It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper) | It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper) | It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper) predicate) | It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping predicate.
Optional<T> findAny() | It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst() | It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.

<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper) | It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper) | It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)

| IntStream flatMapToInt(Function<? super T,? | It returns an IntStream consisting of the results of replacing each element of this stream |

extends IntStream> mapper)	with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper) predicate)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping predicate.
Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)	It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)	It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)	It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper) predicate)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping predicate.

- Type Inference
- Numeric Literals
- Java 7 JDBC
- ✓ Java 4/5 Features
- Java Assertion
- Java For-each Loop
- Java Varargs
- Java Static Import
- Java Autoboxing
- Java Enums
- Java Annotations
- Java Generics
- ✓ RMI
- ✓ Internationalization

Optional<T> findAny()	It returns an Optional describing some element of the stream, or an empty Optional if the stream is empty.
Optional<T> findFirst()	It returns an Optional describing the first element of this stream, or an empty Optional if the stream is empty. If the stream has no encounter order, then any element may be returned.
<R> Stream<R> flatMap(Function<? super T,? extends Stream<? extends R>> mapper)	It returns a stream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
DoubleStream flatMapToDouble(Function<? super T,? extends DoubleStream> mapper)	It returns a DoubleStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have placed been into this stream. (If a mapped stream is null an empty stream is used, instead.)
IntStream flatMapToInt(Function<? super T,? extends IntStream> mapper)	It returns an IntStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping function to each element. Each mapped stream is closed after its contents have been placed into this stream. (If a mapped stream is null an empty stream is used, instead.)
LongStream flatMapToLong(Function<? super T,? extends LongStream> mapper)	It returns a LongStream consisting of the results of replacing each element of this stream with the contents of a mapped stream produced by applying the provided mapping

Java Stream Iterating Example

You can use stream to iterate any number of times. Stream provides predefined methods to deal with the logic you implement. In the following example, we are iterating, filtering and passed a limit to fix the iteration.

```
import java.util.stream.*;
public class JavaStreamExample {
    public static void main(String[] args){
        Stream.iterate(1, element->element+1)
        .filter(element->element%5==0)
        .limit(5)
        .forEach(System.out::println);
    }
}
```

Output:

```
5
10
15
20
25
```

Java Stream Example: Filtering and Iterating Collection

In the following example, we are using filter() method. Here, you can see code is optimized and very concise.

```
import java.util.*;
class Product{
    int id;
    String name;
```

```
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        // This is more compact approach for filtering data
        productsList.stream()
                        .filter(product -> product.price == 30000)
                        .forEach(product -> System.out.println(product.name));
    }
}
```

Output:

```
Dell Laptop
```

Java Stream Example : reduce() Method in Collection

This method takes a sequence of input elements and combines them into a single summary result by repeated operation. For example, finding the sum of numbers, or accumulating elements into a list.

In the following example, we are using reduce() method, which is used to sum of all the product prices.

```
import java.util.*;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        // This is more compact approach for filtering data
        Float totalPrice = productsList.stream()
                .map(product->product.price)
                .reduce(0.0f,(sum, price)->sum+price);  // accumulating price
        System.out.println(totalPrice);
        // More precise code
        float totalPrice2 = productsList.stream()
                .map(product->product.price)
                .reduce(0.0f,Float::sum);   // accumulating price, by referring method of Float class
        System.out.println(totalPrice2);

    }
}
```

Output:

```
201000.0
201000.0
```

Java Stream Example: Sum by using Collectors Methods

We can also use collectors to compute sum of numeric values. In the following example, we are using Collectors class and it?s specified methods to compute sum of all the product prices.

```
import java.util.*;
import java.util.stream.Collectors;
class Product{
    int id;
    String name;
```

```
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        // Using Collectors's method to sum the prices.
        double totalPrice3 = productsList.stream()
                    .collect(Collectors.summingDouble(product->product.price));
        System.out.println(totalPrice3);

    }
}
```

Output:

```
201000.0
```

Java Stream Example: Find Max and Min Product Price

Following example finds min and max product price by using stream. It provides convenient way to find values without using imperative approach.

```
import java.util.*;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        // max() method to get max Product price
        Product productA = productsList.stream().max((product1, product2)->product1.price > product2.price ? 1: -1).get();
        System.out.println(productA.price);
        // min() method to get min Product price
        Product productB = productsList.stream().min((product1, product2)->product1.price > product2.price ? 1: -1).get();
        System.out.println(productB.price);

    }
}
```

Output:

```
90000.0
25000.0
```

Java Stream Example: count() Method in Collection

```
import java.util.*;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class JavaStreamExample {
```

```
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));
        // count number of products based on the filter
        long count = productsList.stream()
                .filter(product->product.price<30000)
                .count();
        System.out.println(count);
    }
}
```

Output:

```
3
```

stream allows you to collect your result in any various forms. You can get you result as set, list or map and can perform manipulation on the elements.

Java Stream Example : Convert List into Set

```
import java.util.*;
import java.util.stream.Collectors;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();

        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        // Converting product List into Set
        Set<Float> productPriceList =
            productsList.stream()
            .filter(product->product.price < 30000)   // filter product on the base of price
            .map(product->product.price)
            .collect(Collectors.toSet());   // collect it as Set(remove duplicate elements)
        System.out.println(productPriceList);
    }
}
```

Output:

```
[25000.0, 28000.0]
```

Java Stream Example : Convert List into Map

```
import java.util.*;
import java.util.stream.Collectors;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}

public class JavaStreamExample {
    public static void main(String[] args) {
        List<Product> productsList = new ArrayList<Product>();
```

```
        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        // Converting Product List into a Map
        Map<Integer,String> productPriceMap =
           productsList.stream()
                   .collect(Collectors.toMap(p->p.id, p->p.name));

        System.out.println(productPriceMap);
    }
}
```

Output:

```
{1=HP Laptop, 2=Dell Laptop, 3=Lenevo Laptop, 4=Sony Laptop, 5=Apple Laptop}
```

Method Reference in stream

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

class Product{
    int id;
    String name;
    float price;

    public Product(int id, String name, float price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public int getId() {
        return id;
    }
    public String getName() {
        return name;
    }
    public float getPrice() {
        return price;
    }
}

public class JavaStreamExample {

    public static void main(String[] args) {

        List<Product> productsList = new ArrayList<Product>();

        //Adding Products
        productsList.add(new Product(1,"HP Laptop",25000f));
        productsList.add(new Product(2,"Dell Laptop",30000f));
        productsList.add(new Product(3,"Lenevo Laptop",28000f));
        productsList.add(new Product(4,"Sony Laptop",28000f));
        productsList.add(new Product(5,"Apple Laptop",90000f));

        List<Float> productPriceList =
              productsList.stream()
                      .filter(p -> p.price > 30000) // filtering data
                      .map(Product::getPrice)       // fetching price by referring getPrice method
                      .collect(Collectors.toList());  // collecting as list
        System.out.println(productPriceList);
    }
}
```

Output:

```
[90000.0]
```

Next Topic Java 8 Stream Filter

← Prev Next →

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

Learn Latest Tutorials

Splunk | SPSS | Swagger | Transact-SQL | Tumblr | ReactJS

Regex | Reinforcement Learning | R Programming | RxJS | React Native | Python Design Patterns

Python Pillow | Python Turtle | Keras

Preparation

Aptitude | Reasoning | Verbal Ability | Interview Questions | Company Questions

Trending Technologies

Artificial Intelligence | AWS | Selenium | Cloud Computing | Hadoop | ReactJS

Data Science | Angular 7 | Blockchain | Git | Machine Learning | DevOps

B.Tech / MCA

DBMS | Data Structures | DAA | Operating System | Computer Network | Compiler Design

Computer Organization | Discrete Mathematics | Ethical Hacking | Computer Graphics | Software Engineering | Web Technology

Cyber Security | Automata | C Programming | C++ | Java | .Net

Python | Programs | Control System | Data Mining | Data Warehouse

- Digital Marketing
- On Page and Off Page SEO
- PPC
- Content Development
- Corporate Training
- Classroom and Online Training
- Data Entry

Training For College Campus

JavaTpoint offers college campus training on Core Java, Advance Java, .Net, Android, Hadoop, PHP, Web Technology and Python. Please mail your requirement at hr@javatpoint.com.
Duration: 1 week to 2 week

Like/Subscribe us for latest updates or newsletter 

LEARN TUTORIALS

Learn Java
Learn Data Structures
Learn C Programming
Learn C++ Tutorial
Learn C# Tutorial
Learn PHP Tutorial
Learn HTML Tutorial
Learn JavaScript Tutorial
Learn jQuery Tutorial
Learn Spring Tutorial

OUR WEBSITES

Javatpoint.com
Hindi100.com
Lyricsia.com
Quoteperson.com
Jobandplacement.com

OUR SERVICES

Website Development
Android Development
Website Designing
Digital Marketing
Summer Training
Industrial Training
College Campus Training

CONTACT

Address: G-13, 2nd Floor, Sec-3
Noida, UP, 201301, India
Contact No: 0120-4256464, 9990449935

Contact Us
Subscribe Us
Privacy Policy
Sitemap

About Me