# Stream API

AUGUST, 2023

expleo

# Introduction

- A stream is a **sequence of objects** that supports various methods which can be **pipelined to produce the desired result**.
- To perform a computation, stream operations are composed into a **stream pipeline**.
- A stream pipeline consists of a **source, zero or more intermediate operations** (which transform a stream into another stream), and a **terminal operation** that ends the use of a stream.

# Introduction

- A **stream pipeline** consists of:
  - **A source**: An array, a collection, a generator function, an I/O channel
  - **Zero or more intermediate operations**, which transform a stream into another stream filter
  - **A terminal operation**, which produces a result or side effect count() or forEach

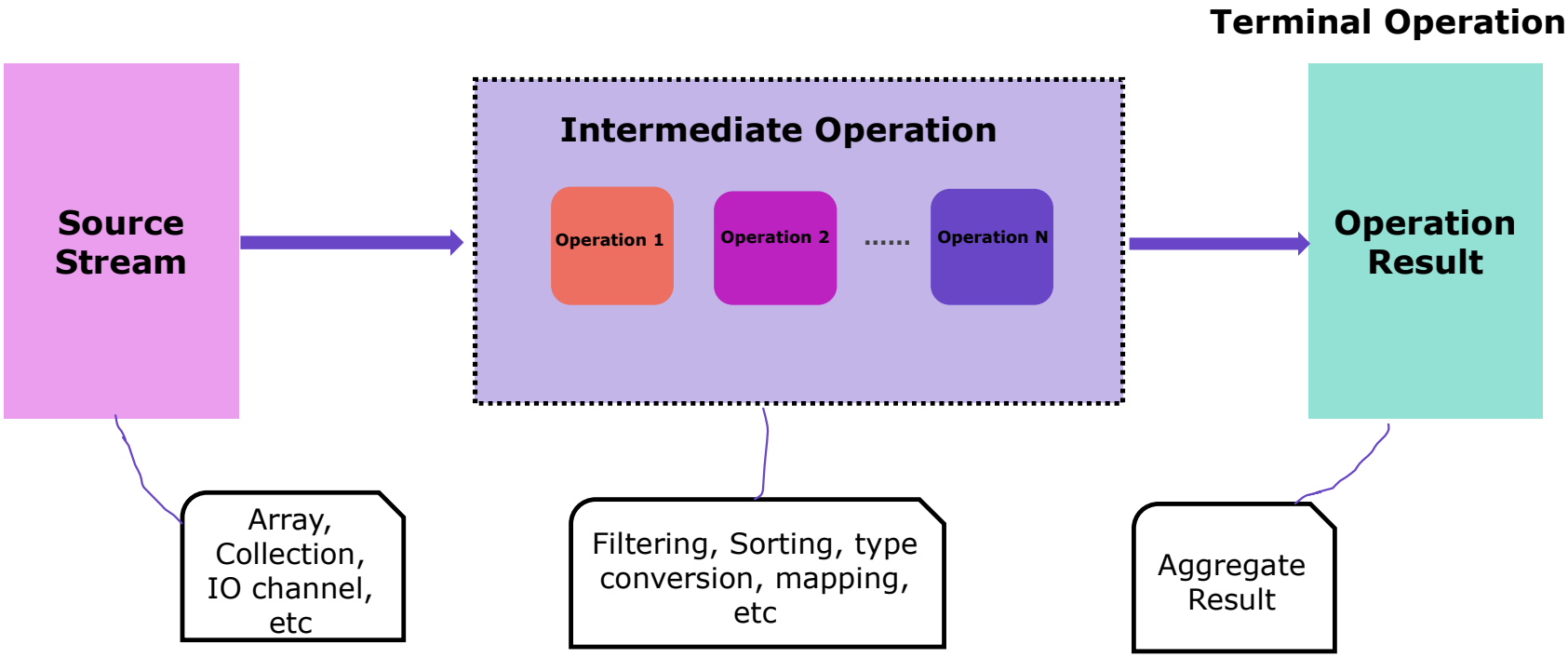
# Introduction

- In Java, **java.util.Stream interface** represents a stream on which one or more operations can be performed.
- Stream **operations** are either **intermediate or terminal**.
- Computation on the **source data is performed** only when the **terminal operation is initiated**, and source elements are consumed only as needed.

# Needs

- The ability to write functions at a **more abstract level** which can **reduce code bugs**, compact functions into **fewer and more readable lines of code.**

# Introduction



# Stream Creation

### Stream of *Collection*

```
Collection<String> collection = Arrays.asList("a", "b", "c");
Stream<String> streamOfCollection = collection.stream();
```

### Stream of Array

```
Stream<String> streamOfArray = Stream.of("a", "b", "c");
```

- **Empty Stream**

```
Stream<String> streamEmpty = Stream.empty();
```

# Stream Creation

- **Stream.generate():** The generate() method accepts a Supplier<T> for element generation. As the resulting **stream is infinite**, the developer should **specify** the **desired size**, or the generate() method will work until it reaches the memory limit:

```
Stream<String> streamGenerated = Stream.generate(() -> "element").limit(10);
```

- Another way of creating an **infinite stream** is by using the ***iterate()*** method:

```
Stream<Integer> streamIterated = Stream.iterate(40, n -> n + 2).limit(20);
```

- The first element of the resulting stream is the first parameter of the *iterate()* method. When creating every following element, the specified function is applied to the previous element.

# Intermediate Operation

- **map**: The map method is typically used to **extract data from a field** and **perform** a **calculation or operation**. The results of the mapping operation are returned as a stream.
  - **Syntax:** map(Function<? super T,? extends R> mapper)
- A Function takes one generic and returns something else.
- Primitive versions of map
  - mapToInt() mapToLong() mapToDouble()

# Intermediate Operation

- **filter**: The filter method is used to **select elements as per the Predicate** passed as argument.
  - **Syntax**: Stream<T> filter(Predicate<? super T> predicate)
- **predicate**: It takes Predicate reference as an **argument**. Predicate is a **functional interface**. So, you can also pass **lambda expression** here.

# Intermediate Operation

- **sorted**: The sorted method can be used to **sort stream elements** based on their **natural order**.
  - **Syntax**: sorted(Comparator<? super T> comparator)
- Returns a stream consisting of the elements **sorted according to the Comparator.**

# Terminal Operation

- **collect**: The collect method allows you to save the results of all the **filtering, mapping, and sorting** that takes place in a **pipeline**.
- It takes a Collectors class as a parameter. The Collectors class provides a number of ways to return the elements left in a pipeline.
  - **Syntax**: collect(Collector<? super T,A,R> collector)
- **Examples**
  - stream().collect(Collectors.toList());
  - stream().collect(Collectors.toMap());

# Terminal Operation

- **forEach:** The forEach method is used to **iterate through every element** of the stream.
- **reduce:** The reduce method is used to **reduce the elements of a stream to a single value**. The reduce method takes a BinaryOperator as a parameter.

# Example #1

```
/*** This example demonstrates simple Stream API*/
import java.util.stream.*;
public class StreamExample{
   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
```

Stream API

# Example #2

```
/*** This example demonstrates use of simple Stream API*/
import java.util.*;
import java.util.stream.*;
class StreamExample{
        public static void main(String args[]) {
        // create a list of integers
        List<Integer> number = Arrays.asList(2,3,4,5);
        // demonstration of map method
List<Integer> square = number.stream().map(x -> x*x).collect(Collectors.toList());
        System.out.println(square);
        // create a list of String
        List<String> names =Arrays.asList("Reflection","Collection","Stream");
        // demonstration of filter method
List<String> result = names.stream().filter(s->s.startsWith("S")) .collect(Collectors.toList());
        System.out.println(result);
```

# Example #2

```
        // demonstration of sorted method
         List<String> show =names.stream().sorted().collect(Collectors.toList());
        System.out.println(show); //[Collection, Reflection, Stream]
        List<Integer> numbers = Arrays.asList(2,3,4,2);
        // collect method returns a set
Set<Integer> squareSet =numbers.stream().map(x->x*x).collect(Collectors.toSet());
        System.out.println(squareSet);  //[16, 4, 9]
        // demonstration of forEach method
        numbers.stream().map(x->x*x).forEach(y->System.out.println(y));  // 4, 9, 16
        // demonstration of reduce method
        int even =numbers.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);
        System.out.println(even);  //8
        }
}
```

# Example #2

**Output:**

```
[4, 9, 16, 25]
[Stream]
[Collection, Reflection, Stream]
[16, 4, 9]
4
9
16
4
8
```

# Quiz


**1) The newly introduced Streams API is available in which package of java 8:**

**a) java.io.streams**

**b) java.io.stream**

**c) java.util.streams**

**d) java.util.stream**

**d) java.util.stream**

# Quiz


**2) What is the purpose of filter method of stream in java 8?**

**a) Iterate each element of the stream.**

**b) Map each element to its corresponding result.**

**c) Eliminate elements based on a criteria**

**d) None of these Above**

**c) Eliminate elements based on a criteria**

# Quiz


**3) Most of the stream operations return stream itself so that their result can be pipelined.**

**a) True**

**b) False**

**a) True**

# Quiz


**4) In Java 8 Streams, Which is aggregate operation?**

**a) filter**

**b) map**

**c) foreach**

**d) All of these Above**

**d) All of these Above**