Streams

Stream is used to process the data from collection.

- To process the data, streams provided different mechanisms
  - → Filter
  - → Map

- Java.io.* → used to work with files.
- Java.util.* → used to work with collections.

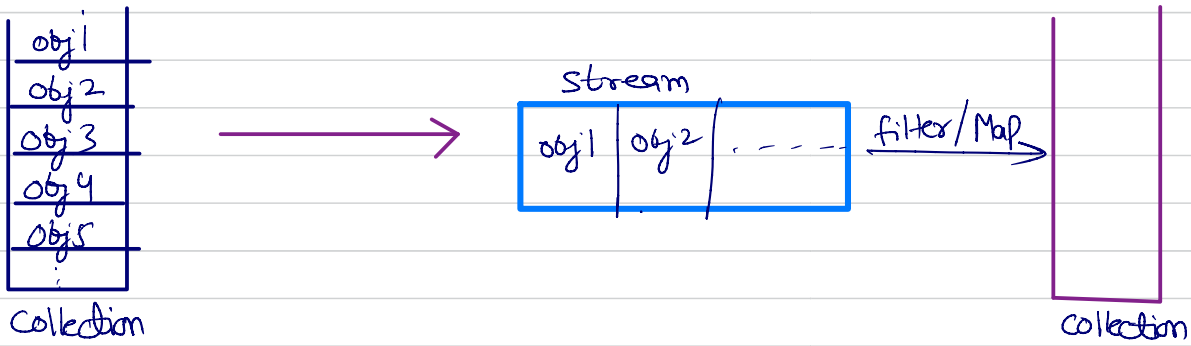
1.) **Filter:-** from the given collection, i will write some condition and based on the condition, i will filter the whole data from the collection and use it for further purposes.

Ex → In an ArrayList, how many numbers are even, store it in some collection (ArrayList, List...).

2.) **Map:-** In map, we will get each and every element/object from the collection, we will apply set of operations in every element and, we can use that data for some other purposes

- → In map, the collection size will not reduce, because every element will map with some operation.
- → But in filter, collection size can reduce.

→ We don't process the data directly from collections, we convert it into stream



→ The data in the collection will not modify, we are just adding it into stream, so here stream is an api.

→ There is a separate class for stream api i.e. Stream, which contains some methods like sorted(), count(), collect() → to collect the data after filter/map and store it in collection, distinct(), foreach(), min(), max().

→ **Collection:-** to represent group of data/objects as single entity.

Filters

```java
package filterInStreams;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class FilterExample1 {

    public static void main(String[] args) {
//      ArrayList<Integer> numbersList = new ArrayList<>();
//      numbersList.add(10);
//      numbersList.add(20);
//      numbersList.add(30);
//      numbersList.add(40);
//      numbersList.add(50);

        List<Integer> numbersList = Arrays.asList(10,20,30,40,50);
        List<Integer> evenNumbersList = new ArrayList<>();

        //without using stream
        for(int number : numbersList) {
            if(number%2==0) {
                evenNumbersList.add(number);
            }
        }

        System.out.println(evenNumbersList);

        //with streams
        //filter method always takes a boolean expression as a parameter
        //predicate Functional Interface
        evenNumbersList = numbersList.stream().filter(n-> n%2==0).collect(Collectors.toList());

        System.out.println(evenNumbersList);

        //if i dont want to store it in variable
        //consumer Functional Interface
        numbersList.stream().filter(n->n%2==0).forEach(n->System.out.println(n));
        //System.out is a static variable, so we can directly access println from it
        numbersList.stream().filter(n->n%2==0).forEach(System.out::println);
    }
}
```

```java
package filterInStreams;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class FilterExample2 {

    public static void main(String[] args) {
        List<String> names = Arrays.asList("Kapil", "Rahul", "Vineet", "Aditya", "javved");
        List<String> longNames = new ArrayList<>();

//      longNames = names.stream().filter(str -> str.length() > 5 && str.length() < 10).collect(Collectors.toList());
//      System.out.println(longNames);

        names.stream().filter(str -> str.length() > 5 && str.length() < 10).forEach(str -> System.out.println(str));

    }
}
```

```java
package filterInStreams;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class FilterExample3 {

    public static void main(String[] args) {
        List<String> words = Arrays.asList("Kapil", null, "Rahul", null,"Vineet", null, "Aditya", "javved");

        List<String> nonNullWords = new ArrayList<>();

        nonNullWords = words.stream().filter(str->str!=null).collect(Collectors.toList());
        System.out.println(nonNullWords);
    }
}
```

```java
package filterInStreams;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

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

    public Product(int id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class FilterExample4 {

    public static void main(String[] args) {

        List<Product> productsList = new ArrayList<>();
        productsList.add(new Product(1, "Table",5000));
        productsList.add(new Product(2, "Chair",65000));
        productsList.add(new Product(3, "Bag",67800));
        productsList.add(new Product(4, "Keyboard",10920));
        productsList.add(new Product(5, "Mouse",400));

        //store the products whose price> 5000,
        List<Product> newProductsList = new ArrayList<>();
        newProductsList = productsList.stream().filter(p->p.price>5000).collect(Collectors.toList());

        for(Product product : newProductsList) {
            System.out.println(product.name + " " + product.price);
        }
        //without storing, just printing the products
        productsList.stream().filter(p->p.price>5000).forEach(product->
          System.out.println(product.name));
    }
}
```

MapStream

```java
package mapStream;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class MapExample1 {

    public static void main(String[] args) {
        List<String> vehicles = Arrays.asList("car", "bus", "truck", "flight", "auto");

        //convert vehicles name in uppercase and store it in other collections

        List<String> vehiclesListInUppercase = new ArrayList<String>();

        //without stream
//      for(String name : vehicles) {
//          name.toUpperCase();
//          vehiclesListInUppercase.add(name);
//      }

        //streams
        vehiclesListInUppercase= vehicles.stream().map(name-> name.toUpperCase()).collect(Collectors.toList());
        System.out.println(vehiclesListInUppercase);
    }
}
```

```java
package mapStream;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class MapExample2 {

    public static void main(String[] args) {
        List<String> vehicles = Arrays.asList("car", "bus", "truck", "flight", "auto");
        //find the length of every word and put it in another collection

        List<Integer> wordLengthsIntegers = new ArrayList<Integer>();
        wordLengthsIntegers = vehicles.stream().map(word-> word.length()).collect(Collectors.toList());
        System.out.println(wordLengthsIntegers);

        //for printing without storing
        vehicles.stream().map(word-> word.length()).forEach(wordlen->System.out.println(wordlen));

        //without wordlen
        vehicles.stream().map(word-> word.length()).forEach(System.out::println);
    }
}
```

```java
package mapStream;

import java.util.Arrays;
import java.util.List;

public class MapExample3 {

    public static void main(String[] args) {
        List<Integer> numbersList = Arrays.asList(10,20,30,40,50);

        //multiply everynumber by 3
        numbersList.stream().map(n->n*3).forEach(num->System.out.println(num));

        //:: represents we println holds a parameter, so no need to specify it
        numbersList.stream().map(n->n*3).forEach(System.out::println);
    }
}
```

```java
package mapStream;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

class Employee {
    int empId;
    String employeeName;
    Float salary;

    public Employee(int empId, String employeeName, Float salary) {
        super();
        this.empId = empId;
        this.employeeName = employeeName;
        this.salary = salary;
    }
}

public class MapExample4 {

    public static void main(String[] args) {
        ArrayList<Employee> employees = new ArrayList<>();
        List<Employee> filteredEmployees = new ArrayList<Employee>();

        employees.add(new Employee(1, "Kapil", 75000f));
        employees.add(new Employee(2, "Rahul", 15000f));
        employees.add(new Employee(3, "Kartik", 145000f));
        employees.add(new Employee(4, "Virat", 1454200f));
        employees.add(new Employee(5, "Rohit", 75000656f));

        // combine filter and map
        filteredEmployees = employees.stream().filter(e -> e.salary > 75000).map(s -> {
            s.salary = s.salary * 1.10f;
            return s;
        }).collect(Collectors.toList());

    }
}

```

FlatMap

→ flatMap is used to work with complex collection (collection containing collection).

→ Returns stream of objects.

```java
package flatMap;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class FlatMapExample1 {

    public static void main(String[] args) {
        List<Integer> lists = Arrays.asList(1, 2, 3, 4, 5, 6, 7);
        List<Integer> finallists = new ArrayList<Integer>();

        // map
        finallists = lists.stream().map(x -> x + 10).collect(Collectors.toList());
        System.out.println(finallists);

        // flatMap
        List<Integer> lists1 = Arrays.asList(1, 2);
        List<Integer> lists2 = Arrays.asList(3, 4);
        List<Integer> lists3 = Arrays.asList(5, 6);
        List<Integer> lists4 = Arrays.asList(7, 8);

        List<List<Integer>> combinedLists = Arrays.asList(lists1, lists2, lists3, lists4);
        List<Integer> updatedCombinedLists = new ArrayList<Integer>();

        updatedCombinedLists = combinedLists.stream().flatMap(x -> x.stream().map(n -> n + 10))
                .collect(Collectors.toList());
        System.out.println(updatedCombinedLists);
    }
}
```

```java
package flatMap;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class FlatMapExample2 {

    public static void main(String[] args) {
        List<String> teamA = Arrays.asList("Virat", "Rohit", "Jaddu");
        List<String> teamB = Arrays.asList("David Warner", "Maxwell", "Cummins");
        List<String> teamC = Arrays.asList("Kane", "Mitchell", "Anderson");

        List<List<String>> playersInWC2023 = Arrays.asList(teamA, teamB, teamC);

        // before Java8
        for (List<String> team : playersInWC2023) {
            for (String player : team) {
                System.out.println(player);
            }
        }
        // in Java 8, using flatMap
        List<String> combinedPlayerList = playersInWC2023.stream().flatMap(t -> t.stream())
                .collect(Collectors.toList());
        System.out.println(combinedPlayerList);
    }
}
```

```java
package flatMap;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

class Student {
    String name;
    int sid;
    char grade;

    Student(String name, int sid, char grade) {
        super();
        this.name = name;
        this.sid = sid;
        this.grade = grade;
    }
}

public class FlatMatExample3 {

    public static void main(String[] args) {

        List<Student> studentList1 = new ArrayList<Student>();
        studentList1.add(new Student("Kapil", 1, 'A'));
        studentList1.add(new Student("Rahul", 2, 'B'));
        studentList1.add(new Student("Kapil", 3, 'C'));

        List<Student> studentList2 = new ArrayList<Student>();
        studentList2.add(new Student("Ajay", 4, 'A'));
        studentList2.add(new Student("Vineet", 5, 'B'));
        studentList2.add(new Student("Alok", 6, 'C'));

        // Add both collection in one single collection
        List<List<Student>> combinedList = Arrays.asList(studentList1, studentList2);
        List<String> studentList = combinedList.stream().flatMap(stulist -> stulist.stream())
          .map(s -> s.name).collect(Collectors.toList());
        System.out.println(studentList);
    }
}
```

```java
package OtherStreams;

import java.lang.reflect.Array;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class DistinctStreamDemo {

    public static void main(String[] args) {
        List<String> vehicleslList = Arrays.asList("bike", "car", "bike", "auto", "plane", "cycle", "metro");

        //store distinct elements in another collection
        List<String> distinctVehicleList = vehicleslList.stream().distinct().collect(Collectors.toList());
        System.out.println(distinctVehicleList);

        //to print without storing
        vehicleslList.stream().distinct().forEach(value->System.out.println(value));

        //count
        long count = vehicleslList.stream().distinct().count();
        System.out.println(count);

        //limit()
        List<String> limitedVehicles=vehicleslList.stream().limit(2).collect(Collectors.toList());
        System.out.println(limitedVehicles);
    }
}
```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.List;

public class ToArrayExample {

    public static void main(String[] args) {

        List<String> stringlList = Arrays.asList("A", "B", "C", "1", "2","3","4");
        Object arr[] = stringlList.stream().toArray();

        //System.out.println(arr.length);

        for(Object data: arr) {
            System.out.println(data);
        }
    }
}
```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

//Stream Methods

// sorted() anyMatch() allMatch() noneMatch()
//findAny() findFirst()

public class SortedStreamExample {

    public static void main(String[] args) {

        List<Integer> list1 = Arrays.asList(5,6,7,1,3,2,4,8,0,19);
        List<Integer> sortedList = list1.stream().sorted().collect(Collectors.toList());
        System.out.println(sortedList);

        //ReverseOrderSorting
        List<Integer> reverseSortedList =
          list1.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());
        System.out.println(reverseSortedList);
    }
}
```

```java
package OtherStreams;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Set;

import javax.management.ValueExp;

//anyMatch
//allMatch
//noneMatch
public class AnyMatchAllMatchNoneMatch {

    public static void main(String[] args) {

        Set<String> fruitSet = new HashSet<String>();
        fruitSet.add("One Mango");
        fruitSet.add("Two Pineapples");
        fruitSet.add("One Papaya");
        fruitSet.add("Three Oranges");
        fruitSet.add("Two bananas");

        // will take predicate as parameter
        // anymatch will return true, if any entry starts with One
        boolean result = fruitSet.stream().anyMatch(value -> {
            return value.startsWith("One");
        });
        System.out.println(result); // true

        // allMatch will return true, if all entries start with One
        boolean result2 = fruitSet.stream().allMatch(value -> {
            return value.startsWith("One");
        });
        System.out.println(result2); // false

        // noneMatch will return true, if none of the entries start with One
        boolean result3 = fruitSet.stream().noneMatch(value -> {
            return value.startsWith("One");
        });
        System.out.println(result3); // false

    }
}
```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

//findAny //findFirst
public class FindAnyFindFirst {

    public static void main(String[] args) {

        List<String> list  = Arrays.asList("one","two", "one", "three");

        //findAny store any element, if collection is not empty
        Optional<String> ele =list.stream().findAny();

        System.out.println(ele.get()); //one or NoSuchElementException

        //findFirst will find the first element from the stream
        Optional<String> ele2 = list.stream().findFirst();
        System.out.println(ele2.get()); //one or NoSuchElementException
    }
}
```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class MiscExample2 {

    public static void main(String[] args) {

        List<Integer> numbersList = Arrays.asList(1,2,3,4,5,6,7,8,9,10);

        //count()
        long numberOfEvenNumbers = numbersList.stream().filter(num->num%2==0).count();
        System.out.println(numberOfEvenNumbers);

        //min
        Optional<Integer> min= numbersList.stream().min((val1,val2) -> {return val1.compareTo(val2);});
        System.out.println(min.get());

        //max
        Optional<Integer> max= numbersList.stream().max((val1,val2) -> {return val1.compareTo(val2);});
        System.out.println(max.get());


        //reduce
        List<String> stringlList = Arrays.asList("A", "B", "C", "1", "2","3","4");


        Optional<String> reduced = stringlList.stream().reduce((value, combinedValue)-> {
            return combinedValue+value;}
        );
        System.out.println(reduced.get());
    }
}
```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class ConcatenateStreamsExample {

    public static void main(String[] args) {
        List<String> animalsList = Arrays.asList("Lion", "Tiger", "Giraffe");
        List<String> birdsList = Arrays.asList("parrot", "peacock", "pigeon");

        Stream<String> stream1 = animalsList.stream();
        Stream<String> stream2 = birdsList.stream();

        //concat streams
        List<String> finalliList=
          Stream.concat(stream1,stream2).collect(Collectors.toList());
        System.out.println(finalliList);
    }
}

```

```java
package OtherStreams;

import java.util.Arrays;
import java.util.List;

class Student {

    String name;
    int score;

    public Student(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
        return score;
    }
}

public class ParallelStream {

    public static void main(String[] args) {

        List<Student> studentList = Arrays.asList(new Student("Kapil", 90), new Student("Rahul", 80),
                new Student("Vineet", 70), new Student("Aditya", 80));

        //sequential stream
        studentList.stream().filter(x -> x.getScore() >= 80).limit(3)
                .forEach(stu -> System.out.println(stu.getName() + " " + stu.getScore()));

        //parallel stream
        studentList.parallelStream().filter(x -> x.getScore() >= 80).limit(3)
        .forEach(stu -> System.out.println(stu.getName() + " " + stu.getScore()));

        //convert stream() -> parallelStream
        studentList.stream().parallel().filter(x -> x.getScore() >= 80).limit(3)
        .forEach(stu -> System.out.println(stu.getName() + " " + stu.getScore()));
    }
}
```