Stream API

By Debarjit Mohanty

Stream API

Collection: A Group of Objects as a single Entity, It means a single unit of objects.

Stream: If we want to process the object from a collection, then we should use a Stream.

Package: java. util. stream

Collection to Stream: Stream S = collection. stream();

In Stream API, operations are classified into two operation:

1. Intermediate Operations:

Transform a stream into another stream.

Examples are: filter, map, distinct, sorted, limit etc.

2. Terminal Operations:

It produce a result and terminate the stream

Examples are: forEach, collect, reduce, count etc.

Stream API Methods

- stream()
- filter()
- map()
- collect()
- count()
- sorted()
- max() and min()
- forEach()
- toArray()
- Arrays. stream()
- Stream. of()

Filter:

Syntax: Stream filteredStream = originalStream.filter(element->/*predicate */);

- It is used to filter the data from stream.
- And create a new stream.
- filter takes predicate as an argument, which returns Boolean types (true/false)
- It is intermediate operation.

Q: Find the even number from ArrayList using stream.

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

public class ClassDemo {

		public static void main(String[] args) {
	List<Integer> list=List.of(1,2,3,4,5,6);

		//List<Integer> result = list.stream().filter(i->i%2==0).collect(Collectors.toList());

		 List<Integer> result = list.stream().filter(i->i%2==0).toList();

		System.out.println(result);

		}
}
```

- **Map**

Syntax: Stream mappedStream = originalStream.map(element->/*transformation function */);

- Map is used to transform each element of Stream.
- And returns a new Stream
- Map takes function as an argument, the return type based on the types of data.
- It is intermediate operation.

Q: Multiply by two for each element from the list.

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

public class ClassDemo {

		public static void main(String[] args) {
	List<Integer> list=List.of(1,2,3,4,5,6);

		List<Integer> result = list.stream().map(i->i*2).toList();

		 System.out.println(result);

		}
}
```

Q: Select only passed student.

Q: Add the 5 grace marks to all the failed student.

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

public class ClassDemo {

public static void main(String[] args) {
	List<Integer> mark=Arrays.asList(40,35,65,19,75,85,30,24,47);
```

```
        List<Integer> passed = mark.stream().filter(i->i>35).toList();
        System.out.println(passed);

        List<Integer> graceMark = mark.stream().filter(i->i<=35).map(i->i+5).toList();
        System.out.println(graceMark);
        }
}
```

- **count():**

To count the number of elements in the Stream

Q: Get the total number of failed students.

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

public class ClassDemo {

        public static void main(String[] args) {
        List<Integer> mark=Arrays.asList(40,35,65,19,75,85,30,24,47);

        long failed_std = mark.stream().filter(i-> i<35).count();
        System.out.println(failed_std);

        }
}
```

- **sorted():**

To sort the order of elements in the Stream.

- Sorted according to natural order.(Asc)

Q: Sort the element from stream.

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

public class ClassDemo {

        public static void main(String[] args) {
          List<Integer> mark=Arrays.asList(40,35,65,19,75,85,30,24,47);

         List<Integer> sorted_mark = mark.stream().sorted().toList();
        System.out.println(sorted_mark);

        List<String> name=Arrays.asList("rajesh", "hemant", "debarjit", "srinivas");

        List<String> name_sort = name.stream().sorted().toList()    ;
        System.out.println(name_sort);
    }
}
```

If we want sort in descending order, then how to perform?

if we want customize sorting order then we should go for **Comparator**.

- **comparator** is a Functional Interface.
- It has compare(obj1, obj2) method.

return -ve; if obj1 has come before obj2.

return +ve; if obj1 has come after obj2.

return 0; if obj1 and obj2 are equal

For Descending Order: (a , b) -> (a < b) ? 1 : (a > b) ? -1 : 0

- **sorted():** According to default natural sorting order
- **sorted d(Comparator):** For customized sorting order

Q: Sort the element in descending order using comparator from stream.

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

public class ClassDemo {

        public static void main(String[] args) {

List<Integer> mark=Arrays.asList(40,35,65,19,75,85,30,24,47);

      // List<Integer> sorted_mark = mark.stream().sorted((a,b)->(a<b)?1:(a>b)?-1:0).toList();

        List<Integer> sorted_mark = mark.stream().sorted((a,b)->b.compareTo(a)).toList();
        System.out.println(sorted_mark);
    }
}
```

Comparable

Method: **compareTo(obj1)**:

List.stream( ).sorted((a , b) -> a.compareTo(b)) . toList( );

If want to reverse the sorting then put the "-" symbol as below

List.stream( ).sorted ((a , b) -> -a.compareTo(b)). toList();

Q: Sort the element based on the length of the ArrayList

1st way

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class ClassDemo {

        public static void main(String[] args) {
                List<String> list = Arrays.asList("A", "AAA", "BB", "BBBBB", "AAAAAA");
                  List<String> result = list.stream().sorted().toList();  //sort in alphabetical order

                  Comparator<String> c = (a, b) -> {
                                int l1 = a.length();
                                int l2 = b.length();
```

```
                    / *         if (l1 < l2)  return -1;
                                else if ( l1 > l2  )  return 1;
                                else  return 0;   */              // one way using if_else

                      return Integer.compare( l1 , l2 );   //second way
                    };
                    List<String> sortedString = list.stream().sorted(c).toList();  //sort in length wise

                    System.out.println(sortedString);

          }
}
```

2nd way

```
Comparator<String> c = (a, b) -> {
                                int l1 = a.length();
                                int l2 = b.length();
                     return Integer.compare( l1,  l2 );
};
//List<String> sortedString = list.stream().sorted((a,b)->Integer.compare(a.length(),b.length())).toList();  //one way

List<String> sortedString = list.stream().sorted(Comparator.comparingInt(String :: length)). toList();  //second way


          System.out.println(sortedString);
```

Q. Sort the element based on the length of the ArrayList in reverse order

```
Comparator<String> c = (a, b) -> {
                                int l1 = a.length();
                                int l2 = b.length();
                     return Integer.compare( l2,  l1 );   //only modify this line remaining same
};
          List<String> sortedString = list.stream().sorted(c). toList();
            System.out.println(sortedString);
```

- **min( ) & max( ):**

- Both method takes the comparator as an argument.
- And based on the comparator result it will return the value.

➢ **min (comparator)** will return 1st element from the comparator result;

➢ **max (comparator)** will return last element from the comparator result;

```
                List<Integer> list = Arrays.asList(10,20,50,54,12,11,94);

                    Integer min = list.stream().min((a,b)->Integer.compare(a,b)).get();

                    Integer max= list.stream().max((a,b)->Integer.compare(a,b)).get();

                       System.out.println(min+"\n"+max);
```

```
List<String> list = Arrays.asList("A", "AAA", "BB", "BBBBB", "AAAAAA");
Comparator<String> c = (a, b) -> {
                                int l1 = a.length();
                                int l2 = b.length();
                     return Integer.compare( l1,  l2 );
};
          String min = list.stream().min(c).get();
          String max = list.stream().max(c).get();

           System.out.println(sortedString);
```

- **forEach():**
  - To perform an action for each element of this stream
  - It is terminal operation.
  - It's similar to for loop

```
List<Integer> list = Arrays.asList(10,20,50,54,12,11,94);

        list.stream().forEach(i->System.out.print(i));  //one way
        list.forEach(System.out::println); //second way
```

- **toArray():**
  - It returns an array containing elements of this stream.
  - It is a terminal operation.

Q: How to convert Stream of object into Array?

Integer[ ] i=list. stream ( ). toArray (Integer[ ]::new);

Q: How to convert Array to Stream?

Arrays. stream (array) or Stream. of (arr)

```
Integer[ ] arr={10,20,30,40,50};
Arrays.stream(arr);
```

Q: How to find even no from an Array ?

```
Integer[ ] arr={10,20,30,40,50};
Arrays.stream(arr).filter(i->i%2==0).forEach(System.out::println);
```

- **Stream. of (args):**

Arguments should be any type either arrays or any group of elements

```
Stream<?> items=stream.of(9 ,44, "aaa", 46, "bb", 56);
Items.forEach(System.out::println);
```

Interview Questions

Q: Find the employee who has maximum salary?

Q: Find the employee whose has secônd highest salary?

Q: Find the employee who is most senior based on joining date?

Q: Count the employee based on the gender?

Employee.java

```
package com.pack1;


import java.util.Date;

public class Employee {

	private String name;
	private double salary;
	private Date joiningdate;
	private String gender;

	public Employee(String name, double salary, Date joiningdate, String gender) {
		super();
		this.name = name;
		this.salary = salary;
		this.joiningdate = joiningdate;
		this.gender = gender;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public double getSalary() {
		return salary;
	}

	public void setSalary(double salary) {
		this.salary = salary;
	}

	public Date getJoiningdate() {
		return joiningdate;
	}

	public void setJoiningdate(Date joiningdate) {
		this.joiningdate = joiningdate;
	}

	public String getGender() {
		return gender;
	}

	public void setGender(String gender) {
		this.gender = gender;
	}

	@Override
	public String toString() {
	return "Employee [name=" + name + ", salary=" + salary + ", joiningdate=" + joiningdate + ", gender=" +
gender+ "]";
	}

}
```

testEmp.java

```
package com.pack1;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.Comparator;
import java.util.Date;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class testEmp {
	public static void main(String[] args) throws ParseException {
		SimpleDateFormat dateFormat = new SimpleDateFormat("dd-mm-yyyy");
		Date joiningDateDev = dateFormat.parse("01-01-2022");
		Date joiningDateSrinivas = dateFormat.parse("15-04-2023");
		Date joiningDateKavya = dateFormat.parse("25-12-2023");
		Date joiningDatePriya = dateFormat.parse("08-09-202022");

	List<Employee> emp = Arrays.asList(new Employee("Dev", 20000, joiningDateDev, "M"),
					new Employee("Srinivas", 15000, joiningDateSrinivas, "M"),
					new Employee("Kavya", 15000, joiningDateKavya, "F"),
				new Employee("Priya", 18000, joiningDatePriya, "F")

		);

	    // Find the employee who has maximum salary?
	    //	Employee maxSalary = emp.stream().max((a, b) ->
				Double.compare(a.getSalary(), b.getSalary())).get();    //one way

		Employee maxSalary1 = emp.stream().
				max(Comparator.comparingDouble(Employee::getSalary)).get();

		System.out.println(maxSalary1);

	    // Find the employee whose has secônd highest salary?
		Optional<Employee> secMaxSal = emp.stream().
				sorted((a, b) -> Double.compare(b.getSalary(), a.getSalary()))
				.skip(1).findFirst();
		System.out.println(secMaxSal);

	     //  Find the employee who is most senior based on joining date?
		Optional<Employee> senior = emp.stream()
				.min((a, b) -> a.getJoiningdate().compareTo(b.getJoiningdate()));
		System.out.println(senior);

	     // Count the employee based on the gender?
		Map<String, Long> gender = emp.stream()
			.collect(Collectors.groupingBy(Employee::getGender, Collectors.counting()));
		System.out.println(gender);
	}
}
```