Day 3: Java Stream API, with functional programming

Java Stream API:

This API is also introduced in java 8. This API belongs to **java.util.stream** package.

The Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

java.util.stream package contains some library classes and interfaces by using which we can perform functional style of programming on the group of objects(collection of data).

This API has one main interface:

```
java.util.stream.Stream
```

Note:- Object of this **Stream** interface represents sequence of object from a source like collections.

The feature of java stream:

- The stream does not stores the elements, it only represents elements in a sequence.

  Example: wire does not store the electricity.
- It holds only objects, primitives are not allowed.
- Operation(filtering) performed on the stream does not modify its source.

  Example: filtering a stream obtained from a source(collection) produces a new stream with the filtered element rather than removing the elements from the source collection.
- With the help of stream obj we can perform various operations on the collection of objects in functional style, like filtering some elements, transform some elements, manipulate, sort, etc.
- Stream is lazy and evaluates code only when required.
- The elements of a stream are only visited once during the life of a stream. a new stream must be generated to revisit the same elements of the source.

From java 8, Collection interface provides following method to gets a Stream obj.

public default Stream<T> stream();

Note: We can use Stream as generic type for example:

Stream<String>: represents sequence of String object from a source(Collection object).

Stream<Employee>: represents sequence of Employee object from a source(Collection object).

Methods in Stream interface:

There are two types of methods in **Stream** interface:

1. Intermediate methods
2. Terminal methods

1. **Intermediate methods:** these methods returns a new stream object, these intermediate methods never gives the final result.
   most commonly used intermediate methods are:

   public Stream map(Function f);

   public Stream filter(Predicate p);

2. **Terminal methods:**- stream object returns a result only when terminal methods are called on the stream object, the terminal methods consumes that stream object and after that we can't use that stream object again.

   some of the terminal methods are:

   public void forEach(Consumer c);

   public R collect(Collector c);

   public Optional<T> min(Comparator c);

   public Optional<T> max(Comparator c);

   public boolean anyMatch(Predicate p)

   public boolean allMatch(Predicate p)

   public long count();

   Example1: forEach method:

```
    ArrayList<String> al = new ArrayList<String>();

    al.add("one");
    al.add("one1");
    al.add("one2");
    al.add("one3");

    Stream<String> ss=al.stream();

    ss.forEach(i->System.out.println(i));
```

Difference Between Collection.stream().forEach() and Collection.forEach()

Collection.stream().forEach()	Collection.forEach()
Collection.stream().forEach() is also used for iterating the collection but it first converts the collection to the stream and then iterates over the stream of collection.	Collection.forEach() uses the collections iterator.
Unlike Collection.forEach() it does not execute in any specific order, i.e. the order is not defined.	If always execute in the iteration order of iterable, if one is specified.
During structure modification in the collection, the exception will be thrown later.	If we perform any structural modification in the collection by using the collection.forEach() it will immediately throw an exception.
If iteration is happening over the synchronized collection, then it does not lock the collection.	If iteration is happening over the synchronized collection, then it locks the collection and holds it across all the calls.

Example 2: print all the Student marks from the List of Student using stream.

```
public class Main {

        public static void main(String[] args) {

            List<Student> students = new ArrayList<>();

            students.add(new Student(10,"Name1",850));
            students.add(new Student(12,"Name2",750));
            students.add(new Student(13,"Name3",650));
            students.add(new Student(14,"Name4",950));
            students.add(new Student(15,"Name5",750));

            students.stream().forEach(s -> System.out.println(s.getMarks()));

        }
}
```

Example: filter() method:

This method take Predicate object as an argument and filter the stream based on the Predicate condition, and returns the filtered elements in a another stream object.

It will not filter the elements from the source collection object.

Example: filter the students from the List of Student whose marks is greater than 800.

```
import java.util.ArrayList;
import java.util.List;

public class Main {

        public static void main(String[] args) {

            List<Student> students = new ArrayList<>();
            students.add(new Student(10,"Name1",850));
            students.add(new Student(12,"Name2",750));
            students.add(new Student(13,"Name3",650));
            students.add(new Student(14,"Name4",950));
            students.add(new Student(15,"Name5",750));

            //Stream<Student> str1 = students.stream();

            //Stream<Student> str2 = str1.filter(s -> s.getMarks() > 800);

            //str2.forEach( s -> System.out.println(s.getName()));

      students.stream()
          .filter(s -> s.getMarks() > 800)
            .forEach(s -> System.out.println(s.getName()));


        }
}
```

We Problem:

Create another List of Student whose marks is greater than 800 from a List of Student

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
public class Main {

        public static void main(String[] args) {

            List<Student> students = new ArrayList<>();
            students.add(new Student(10,"Name1",850));
            students.add(new Student(12,"Name2",750));
            students.add(new Student(13,"Name3",650));
            students.add(new Student(14,"Name4",950));
            students.add(new Student(15,"Name5",750));

            //Stream<Student> str1 = students.stream();

            //Stream<Student> str2 = str1.filter(s -> s.getMarks() > 800);

            //List<Student> anotherList =  str2.collect(Collectors.toList());
```

```
            List<Student> anotherList= students.stream()
                                                .filter(s -> s.getMarks() > 800)
                                                .collect(Collectors.toList());
        }
}
```

map() method:

This method takes a Function object as an argument and map the element to the new element and returns the newly mapped elements in another stream object.

Example1: from a list of name, generate a new list which has name with welcome msg:-

```
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
public class Main {

        public static void main(String[] args) {

            ArrayList<String> al = new ArrayList<String>();

            al.add("ramesh");
            al.add("suresh");
            al.add("mukesh");
            al.add("ajay");

            Stream<String> ss=al.stream();


            //List list=ss.map(s->{return "welcome "+s;}).collect(Collectors.toList());

            //or without using return keyword

            List list=ss.map(s-> "welcome "+s).collect(Collectors.toList());


            list.stream().forEach(s->System.out.println(s));

        }
}
```

Example: getting List of Uppercase String from the List of lowercase string:

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

public class Main {

    public static void main(String[] args) {

        List<String> citiesL= Arrays.asList("delhi","mumbai","chennai","kolkata");
```

```
        List<String> citiesU =  citiesL.stream().map( city -> city.toUpperCase()).collect(Collectors.toList());

        System.out.println(citiesL);
        System.out.println(citiesU);


    }
}
```

You Problem:

Convert a list of String into a list of Integer(length of that string) and then filter all even numbers inside another List.

Adding all the marks of students:

```
import java.util.ArrayList;
import java.util.stream.Collectors;
public class Main {

    public static void main(String[] args) {

        ArrayList<Student> al=new ArrayList<Student>();

        al.add(new Student(10, "n1", 852));
        al.add(new Student(12, "n2", 854));
        al.add(new Student(13, "n3", 851));
        al.add(new Student(14, "n4", 856));
        al.add(new Student(15, "n5", 858));

        int x=al.stream().collect(Collectors.summingInt(s->s.getMarks()));

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

Using Allmatch, Anymatch, noneMatch methods:

These methods take a Predicate object as an argument.

Example:

```
public class Main {

    public static void main(String[] args) {

        ArrayList<Student> al=new ArrayList<Student>();

        al.add(new Student(10, "n1", 852));
        al.add(new Student(12, "n2", 854));
        al.add(new Student(13, "n3", 851));
        al.add(new Student(14, "n4", 856));
        al.add(new Student(15, "n5", 858));
```

```
        boolean b=al.stream().allMatch(s->s.getMarks()>800);

        System.out.println(b);

    }
}
```

References:

https://www.geeksforgeeks.org/

https://www.javatpoint.com/

https://docs.oracle.com/javase/tutorial/java