

LE: it is an expression using which we can provide implementation of an FI.

--Using LE we can represent the object a FI.

2.java.util.function.Consumer<T>:

=====

public void accept(T t);

--this method only accept the object of generic type and does not return anything/

MyConsumer.java:

-----

package com.masai;

import java.util.function.Consumer;

public class MyConsumer implements Consumer<Student>{

    @Override

    public void accept(Student s) {

        System.out.println("Roll is :"+s.getRoll());

        System.out.println("Name is :"+s.getName());

        System.out.println("Marks is :"+s.getMarks());

    }

}

Demo.java:

-----

package com.masai;

import java.util.function.Consumer;

```

public class Demo{

    public static void main(String[] args) {

//          Consumer<Student> c= new MyConsumer();
//
//          c.accept(new Student(10, "N1", 500));
//

        Consumer<Student> c2= s -> {

            System.out.println("Roll is :"+s.getRoll());
            System.out.println("Name is :"+s.getName());
            System.out.println("Marks is :"+s.getMarks());

        };

        c2.accept(new Student(10, "N1", 500));

    }

}

```

forEach method:  
=====

public void forEach(Consumer action); // action for each element of a collection

--this method is a default method belongs to Iterable interface.

--as we know that every collection is iterable (refer the Collection hierarchy diagram)

---so we can call this forEach method on any collection object.

Demo.java:  
-----

package com.masai;

```

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class Demo{

    public static void main(String[] args) {

        List<String> names=Arrays.asList("Amit","Ravi","Sunil","Mukesh");

        //normal for loop
        //enhanced for loop
        //Iterator
        //ListIterator

        names.forEach(name -> System.out.println(name.toUpperCase()));

    }
}

```

example 2:

Demo.java:

-----

```

package com.masai;

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

public class Demo{

    public static void main(String[] args) {

```

```

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

students.add(new Student(10, "N1", 750));
students.add(new Student(20, "N2", 750));
students.add(new Student(30, "N3", 750));
students.add(new Student(40, "N4", 750));


students.forEach(s -> {

    System.out.println(s);
    //write that object to the File(Serialize the object)

});

}

}

```

3. java.util.function.Supplier<T>:  
=====

```
public T get();
```

example

MySupplier.java:

```

-----

package com.masai;

import java.util.function.Supplier;

public class MySupplier implements Supplier<String>{

    @Override
    public String get() {

```

```
        return "This message from the external class";
    }
}
```

Demo.java:

```
-----

package com.masai;

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

public class Demo{

    public static int getANumber() {

        return 1000;
    }

    public static void main(String[] args) {

        Supplier<String> s= new MySupplier();

        String str= s.get();

        System.out.println(str);

        Supplier<String> s2 = () -> "This message from the LE";

        System.out.println(s2.get());

        Supplier<Student> s3 = () -> new Student(10, "N1", 450);

        System.out.println(s3.get());

        Supplier<Integer> s4 = Demo::getANumber;
```

```

        System.out.println(s4.get());

    }

}

```

4.java.util.function.Function<T,R>:

-----

```
public R apply(T t)
```

example :

Getting a Student object and returning the result of that student if marks > 500  
return Pass otherwise return fail

example:

MyFunction.java:

-----

```
package com.masai;
```

```
import java.util.function.Function;
```

```
public class MyFunction implements Function<Student, String>{
```

```
    @Override
```

```
    public String apply(Student s) {
```

```
        //          if(s.getMarks() > 500)
```

```
        //          return "Pass";
```

```
        //          else
```

```
        //          return "fail";
```

```
        return s.getMarks() > 500 ? "Pass" : "fail";
```

```
    }  
}
```

Demo.java:

-----

```
package com.masai;
```

```
import java.util.function.Function;
```

```
public class Demo{
```

```
    public static void main(String[] args) {
```

```
        Function<Student, String> f1= new MyFunction();
```

```
        System.out.println(f1.apply(new Student(10, "N1", 450)));
```

```
        Function<Student, String> f2 = s -> s.getMarks() > 500?"Pass": "Fail";  
        System.out.println(f2.apply(new Student(10, "N1", 850)));
```

```
    }
```

```
}
```

Java Stream api:

=====

--this api is introduced in java 1.8

--this api belongs to "java.util.stream" package

--this api is different from IO stream, this IO-stream api belongs to java.io package and java.nio package

here we represent flow of data between peripherals (input output devices) in the form of bytes or characters.

this java.io stream represents flow of data in bytes or characters

--this java.util.stream package contains some library classes and interfaces by using which we can perform functional style of programming on a group of objects(Collection of data) in the form of Objects.

this java.util stream represents flow of data in the form of objects.

\*\*this api has one main interface :

java.util.stream.Stream(I)

Note: object of this Stream interface represents flow/sequence of objects from a source like collection objects.

Feature of Stream :

=====

1.stream does not store the elements, it only represents elements in a sequence  
ex: wire does not hold/store the electricity

2.it represent only flow of objects , not the primitives.

3. operations (filtering/mapping,etc) performed on the stream object does not modify its source.

ex: filtering a stream obtained from a source (collection) produces a new stream with the filtered elements rather than removing the elements from the source collection.

4. with the help of stream object we can perform various operations on the collection data in functional style, like filtering some elements , printing some elements, transforming some elements,etc.

--Collection interface provides 2 methods to get a Stream object.

1. Stream<T> stream();

2. Stream<T> parallelStream(); // this stream obj is used on multithreaded application.



methods of the Stream(I) interface:

-----

there are 2 types of method in the Stream interface:

1. intermediate methods

2. terminal methods

1. intermediate methods: these methods return a new Stream object, instead of a final output.

--these methods never give the final output.

some of the commonly used intermediate methods are:

map() , filter() method

2. terminal methods:- stream objects return the final output only when terminal method is called on the stream object.

these methods consume that stream object. and after that we can not re-use that stream obj again.

Note: if we try to use a consumed stream obj once again , then we will get an exception.

some of the commonly used terminal methods are :

forEach(Consumer action) // similar to the Iterable interface forEach method

collect()

min()

max()

count()

get()

anyMatch()

allMatch()

example:

Demo.java:

-----

```
package com.masai;
```

```
import java.util.Arrays;
```

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

```
import java.util.stream.Stream;
```

```
public class Demo{
```

```
    public static void main(String[] args) {
```

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

```
        Stream<String> str1= list.stream();
```

```
        str1.forEach(s -> System.out.println(s)); // terminal method
```

```
        str1.forEach(s -> System.out.println(s)); // Runtime exception
```

```
    }
```

```
}
```

filter() methods:

=====

--it is one the intermediate method.

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

example:

Demo.java:

-----

```
package com.masai;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;
```

```
public class Demo{
```

```
    public static void main(String[] args) {
```

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

```
        students.add(new Student(10, "N1", 750));
        students.add(new Student(12, "N2", 450));
        students.add(new Student(13, "N3", 650));
        students.add(new Student(14, "N4", 850));
        students.add(new Student(15, "N5", 410));
```

```
        //from the above list get another list of students whose marks is
        //less than 500.
```

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

```
        Stream<Student> str2= str1.filter(s -> s.getMarks() < 500);
```

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

```
        students.stream()
            .filter(s -> s.getMarks() < 500)
            .forEach(s -> System.out.println(s));
```

```
    }  
}
```

--creating another list based on filtered elements instead of printing them on the console.

Demo.java:

-----

```
package com.masai;
```

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

```
public class Demo{
```

```
    public static void main(String[] args) {
```

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

```
        students.add(new Student(10, "N1", 750));  
        students.add(new Student(12, "N2", 450));  
        students.add(new Student(13, "N3", 650));  
        students.add(new Student(14, "N4", 850));  
        students.add(new Student(15, "N5", 410));
```

```
        //from the above list get another list of students whose marks is  
        //less than 500.
```

```
        //        Stream<Student> str1= students.stream();  
        //  
        //        Stream<Student> str2= str1.filter(s -> s.getMarks() < 500);  
        //  
        //        str2.forEach(s -> System.out.println(s));
```

```

        List<Student> filteredList= students.stream()
                                                    .filter(s ->
s.getMarks() < 500)
.collect(Collectors.toList());

        System.out.println(students);
        System.out.println(filteredList);

    }

}

```

map() method:

-----

--it is also a intermediate method.

--this method is used to transform the object.

--this method takes java.util.function.Function(I) object as an argument and map/transform the element to a new element and returns the mapped elements in another stream.

exmaple :

```
package com.masai;
```

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

```
public class Demo{
```

```
    public static void main(String[] args) {
```

```

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

students.add(new Student(10, "N1", 750));
students.add(new Student(12, "N2", 450));
students.add(new Student(13, "N3", 650));
students.add(new Student(14, "N4", 850));
students.add(new Student(15, "N5", 410));

// Stream<Student> str1= students.stream();
//
//
// Stream<Student> str2= str1.map(s -> {
//
//         Student s2 = new Student(s.getRoll(), s.getName(), s.getMarks()+50);
//
//         return s2;
//
//     });
//
// List<Student> modifiedStudents= str2.collect(Collectors.toList());

List<Student> modifiedList= students.stream()
                                .map(s
-> new Student(s.getRoll(), s.getName(),s.getMarks()+50))
                                .collect(Collectors.toList());

modifiedList.forEach(s -> System.out.println(s));

}

}

```

min and max methods:

=====

these methods are the terminal methods which will takes a Comparator object, using which we can decide max and min elements.

--this min() and max() method will return the minimum and maximum object in the form of "java.util.Optional" class object.

--this Optional class introduced in java1.8 and it is basically used to avoid the NullPointerException

--to get the element from this Optional class ,we need to call get() method.

ex:

Demo.java:

-----

```
package com.masai;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```
public class Demo{
```

```
    public static void main(String[] args) {
```

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

```
        students.add(new Student(10, "N1", 750));
        students.add(new Student(12, "N2", 450));
        students.add(new Student(13, "N3", 650));
        students.add(new Student(14, "N4", 850));
```

```

        students.add(new Student(15, "N5", 410));

//        Stream<Student> str1= students.stream();
//
//
//        Optional<Student> opt= str1.min((s1,s2) -> s1.getMarks() > s2.getMarks() ?
+1:-1 );
//
//        Student s= opt.get();
//
//
//        System.out.println(s);

        Student minStudent= students
                                .stream()
                                .min((s1,s2) -> s1.getMarks() > s2.getMarks() ? +1: -1)
                                .get();

        System.out.println(minStudent);

    }
}

```

count() method:  
=====

```
package com.masai;
```

```

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

```

```
public class Demo{
```

```
    public static void main(String[] args) {
```



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

```
students.add(new Student(10, "N1", 750));  
students.add(new Student(12, "N2", 450));  
students.add(new Student(13, "N3", 650));  
students.add(new Student(14, "N4", 850));  
students.add(new Student(15, "N5", 410));
```

```
long result= students.stream().filter(s-> s.getMarks() < 500).count();
```

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

```
}
```

```
}
```

allMatch() anyMatch, nonMatch()

=====

--these methods takes the Predicate object and returns boolean

ex:

```
package com.masai;
```

```
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;
```

```
public class Demo{
```

```
public static void main(String[] args) {  
  
    List<Student> students = new ArrayList<>();  
  
    students.add(new Student(10, "N1", 750));  
    students.add(new Student(12, "N2", 450));  
    students.add(new Student(13, "N3", 650));  
    students.add(new Student(14, "N4", 850));  
    students.add(new Student(15, "N5", 410));  
  
    boolean result= students.stream().allMatch(s -> s.getMarks() < 1000);  
  
    System.out.println(result);  
  
}  
  
}
```

