Introduced in Java 8, 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.  
The features of Java stream are –

* A stream is not a data structure instead it takes input from the Collections, Arrays or I/O channels.
* Streams don’t change the original data structure, they only provide the result as per the pipelined methods.
* Each intermediate operation is lazily executed and returns a stream as a result, hence various intermediate operations can be pipelined. Terminal operations mark the end of the stream and return the result.

Different Operations On Streams-  
**Intermediate Operations:**

1. **map:**The map method is used to returns a stream consisting of the results of applying the given function to the elements of this stream.  
   **List number = Arrays.asList(2,3,4,5); // collection**List **square** = number.stream().map(x->x\*x) //stream API

.collect(Collectors.toList()); //stream API

1. **filter:** The filter method is used to select elements as per the Predicate passed as argument.  
   List names = Arrays.asList("Reflection","Collection","Stream",”Saravanan”);  
   List result =

names.stream().filter(s->s.startsWith("S"))

.collect(Collectors.toList());

1. **sorted:** The sorted method is used to sort the stream.  
   List names = Arrays.asList("Reflection","Collection","Stream");

List result =

names.stream().sorted()

.collect(Collectors.toList());

**Terminal Operations:**

1. **collect:** The collect method is used to return the result of the intermediate operations performed on the stream.  
   List number = Arrays.asList(2,3,4,5,3);  
   Set square = number.stream().map(x->x\*x).collect(Collectors.toSet());
2. **forEach:** The forEach method is used to iterate through every element of the stream.  
   List number = Arrays.asList(2,3,4,5); //collection is created  
   number.stream().map(x->x\*x) //Stream API

.forEach(y->System.out.println(y)); //Stream API

1. **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.

List number = Arrays.asList(2,3,4,5);  
int even = number.stream().filter(x->x%2==0)

.reduce(0,(ans,i)-> ans+i);

System.out.println(even);

**Important Points/Observations:**

1. A stream consists of source followed by zero or more intermediate methods combined together (pipelined) and a terminal method to process the objects obtained from the source as per the methods described.
2. Stream is used to compute elements as per the pipelined methods without altering the original value of the object.

Demo :

import java.util.\*;

import java.util.stream.\*;

class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

// 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);

// demonstration of sorted method

List<String> show =

names.stream().sorted().collect(Collectors.toList());

System.out.println(show);

// create a list of integers

List<Integer> numbers = Arrays.asList(2,3,4,5,2);

// collect method returns a set

Set<Integer> squareSet =

numbers.stream().map(x->x\*x).collect(Collectors.toSet());

System.out.println(squareSet);

// demonstration of forEach method

number.stream().map(x->x\*x).forEach(y->System.out.println(y));

// demonstration of reduce method

int even =

number.stream().filter(x->x%2==0).reduce(0,(ans,i)-> ans+i);

System.out.println(even);

}

}

Output :

Hello, World!

[4, 9, 16, 25]

[Stream]

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

49

16

25

6

Example 1 :

import java.util.\*;

import java.util.stream.\*;

class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

int[] number={4,1,3,60,16,20}; //collections

//Find min of given elements // normal java code without streams

int min=number[0];

for(int i=0; i<number.length;i++)

{

if(min>number[i]) {

min=number[i];

}

}

System.out.println(min);

// the above code is without streams home work this code to be converted using stream API

}

}

int[] numbers={4,1,13,90,16,2,0};

try

{

int m=IntStream.of(numbers)

.min()

.getAsInt();

System.out.println(m);

}catch(Exception e) {System.out.println(e);}

int[] numbers={4,1,13,90,16,2,0};

try

{

IntStream.of(numbers)

.min()

.ifPresent(m->System.out.println(m)); //lambda expression

}catch(Exception e) {System.out.println(e);}

(or)

try

{

IntStream.of(numbers)

.min()

.ifPresent(System.out::println); // replace lambda with static method reference.

}catch(Exception e) {System.out.println(e);}

int[] numbers={};

try

{

int m=IntStream.of(numbers)

.min()

.getAsInt();

System.out.println(m);

}catch(Exception e) {System.out.println(e);}

java.util.NoSuchElementException: No value present

int[] numbers={4,5,2,6,10,23,2};

try

{

IntStream.of(numbers)

.min()

.ifPresent(System.out::println);

}catch(Exception e) {System.out.println(e);}

IntSummaryStatistics stat=

IntStream.of(numbers).summaryStatistics();

System.out.println(stat.getMin());

System.out.println(stat.getMax());

System.out.println(stat.getAverage());

System.out.println(stat.getCount());

System.out.println(stat.getSum());

Example 2: Find 3 distinct smallest numbers from the given array.

int[] numbers={4,5,1,6,10,23,2};

//clone

int[] copy=Arrays.copyOf(numbers, numbers.length);

//sort

Arrays.sort(copy);

for(int i=0;i<3;i++)

{

System.out.println(copy[i]);

}

Output :

1

2

4

int[] numbers={4,5,2,6,10,23,2};

IntStream.of(numbers) //creates the stream

.distinct() // process the stream

.sorted() // process the stream

.limit(3) // process the stream

.forEach(System.out::println); // consume the stream

2

4

5

int[] numbers={4,5,2,6,10,23,2};

int s=IntStream.of(numbers) //creates the stream

.distinct() // process the stream

.sorted() // process the stream

.limit(3) // process the stream

.sum(); //consume the stream

System.out.println(s);

11

Or

average()

count()

min()

max()

//Apply same final operations as before

3 distinct parts of stream

* 1. Create
  2. Process
  3. Consume

[4,5,2,6,10,23,2] create

[4,5,2,6,10,23] distinct

[2,4,5,6,10,23] sort

[2,4,5] limit

11 sum

IntStream.range(1,100)

.forEach(System.out::println);

//print 1 to 99

IntStream.of(numbers); // from array

IntStream.range(1,101); //1 to 100

IntStream.rangeClosed(1,100); // 1..100

IntStream.generate(supplier()); // from supplier

IntStream.of(numbers).distinct(); //distinct

IntStream.of(numbers).sorted(); // sort

IntStream.of(numbers).limit(3); // get first 3

IntStream.of(numbers).skip(3); // skip first 3

IntStream.of(numbers).filter(num->num%2==0); // only even

IntStream.of(numbers).map(num->num\*2); // double each

IntStream.of(numbers).boxed(); //convert each num to Integer

IntStream.range(1,100).forEach(System.out::println); // print 1 to 99

IntStream.range(1,100).toArray(); // collect into array

List<Integer> l =IntStream.range(1,100).boxed().collect(Collectors.toList()); // collect into list

boolean b =IntStream.of(numbers).anyMatch(num->num%2==1); // is any num odd

boolean b =IntStream.of(numbers).allMatch(num->num%2==1); // is all num odd