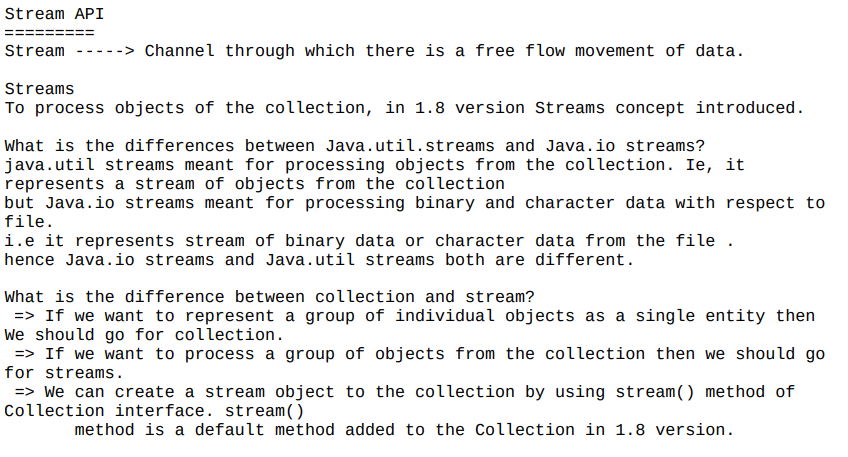
Stream –Channel through which free flow movement of data.

Collection: Group of individual Objects represented as a Single Entity.

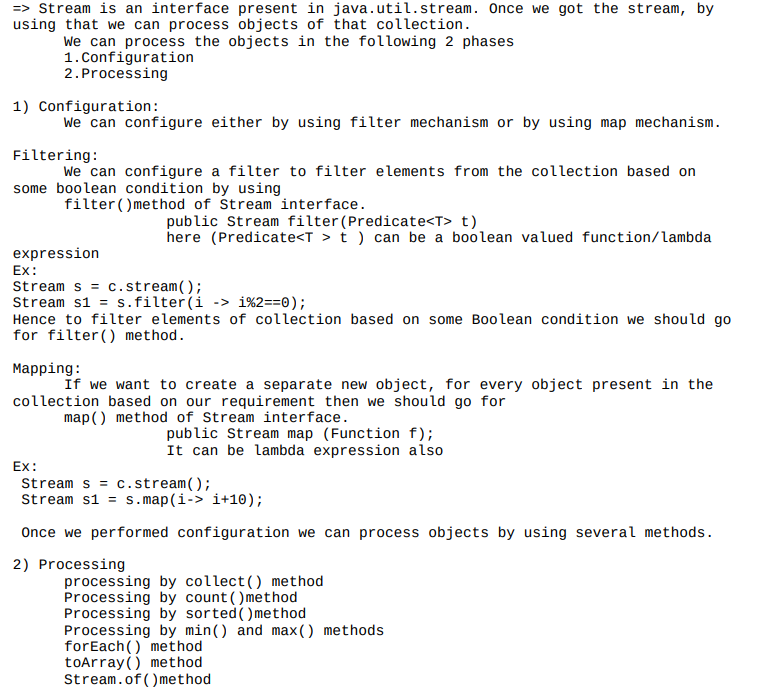


Make sure to import below

import java.util.\*;

import java.util.stream.\*;

Stream is an interface



filter:

package com.chinmay.streamapi;

//

//import java.util.ArrayList;

//import java.util.List;

//import java.util.stream.Collectors;

import java.util.\*;

import java.util.stream.\*;

//import java.util.stream.Collectors;

public class ArrayListStream {

public static void main(String[] args) {

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

al.add(0);

al.add(5);

al.add(10);

al.add(15);

al.add(20);

System.***out***.println(al);//[0,5,10,15,20,25]

//till jdk1.7v

ArrayList<Integer> evenList = new ArrayList<Integer>();

for ( Integer i1: al )

if (i1%2==0)

evenList.add(i1);

System.***out***.println(evenList);//[0,10,20]

//From JDK1.8V we use Streams

//1. Configuration ===> al.stream()

//2. Processing ===> filter(i->i%2==0) ---InterediateOperation

//3.Collect ==>.collect(Collectors.toList());

List<Integer> streamList=al.stream().filter(i->i%2==0).collect(Collectors.*toList*());

System.***out***.println(streamList);

streamList.forEach(System.***out*** :: println);

}

}

map:

package com.chinmay.streamapi;

//

//import java.util.ArrayList;

//import java.util.List;

//import java.util.stream.Collectors;

import java.util.\*;

import java.util.stream.\*;

//import java.util.stream.Collectors;

public class StreamMapMethodExample {

public static void main(String[] args) {

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

al.add(0);

al.add(5);

al.add(10);

al.add(15);

al.add(20);

System.***out***.println(al);//[0,5,10,15,20,25]

//till jdk1.7v

ArrayList<Integer> evenList = new ArrayList<Integer>();

for ( Integer i1: al )

if (i1%2==0)

evenList.add(i1);

System.***out***.println(evenList);//[0,10,20]

//From JDK1.8V we use Streams

//1. Configuration ===> al.stream() --Creating a Stream

//2. Processing ===> map(i->i\*2==0) ---InterediateOperation (There can be any number of intermediate Operations

//3.Collect ==>.collect(Collectors.toList()); --terminal Operation

// map-> for every object, if a new object has to be created then we can with Map

List<Integer> streamList=al.stream().map(i->i\*2).collect(Collectors.*toList*());

System.***out***.println(streamList);

streamList.forEach(System.***out*** :: println);

streamList.forEach(i->System.***out***.print(i+" "));

}

}

Example 2:

package com.chinmay.streamapi;

import java.util.\*;

import java.util.stream.\*;

public class StreamAPIMapExample2 {

public static void main(String args[])

{

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

names.add("sachin");

names.add("saurav");

names.add("dhoni");

names.add("dravid");

names.add("kohli");

names.add("raina");

System.***out***.println(names);

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

for(String name:names)

{

upperCaseNames.add(name.toUpperCase());

}

System.***out***.println(upperCaseNames);

System.***out***.println();

Stream<String> StringStream=names.stream();

List<String> upperStream=StringStream.map(name->name.toUpperCase()).collect(Collectors.*toList*());

upperStream.forEach(n->System.***out***.println(n));

upperStream.forEach(System.***out***::println);

}

}

sortedmethod:

package com.chinmay.streamapi;

//Comparable(Predefined API for natural sorting order) -> compareTo(Object obj)

//Comparator(for userdefined class for customized sorting order)-> compare(Obj1,Obj2)

import java.util.\*;

import java.util.stream.\*;

public class SortedMethodExample {

public static void main(String[] args) {

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

al.add(10);

al.add(0);

al.add(15);

al.add(5);

al.add(20);

System.***out***.println("Before sorting :: "+al);

//using Stream api

List<Integer> resultList=al.stream().sorted().collect(Collectors.*toList*());//As we have not specified any Parameters sorting order will be Considered

System.***out***.println("After sorting :: "+resultList);

//As we want Customized order we have provided the implementation for compareTo() i.e (i1,i2)->i2.compareTo(i1)

List<Integer> customizedResult = al.stream().sorted((i1,i2)->i2.compareTo(i1)).collect(Collectors.*toList*());

System.***out***.println("After sorting :: "+customizedResult); }

}

min and max method:(terminal Operation)

package com.chinmay.streamapi;

import java.util.ArrayList;

public class MinMaxMethod {

public static void main(String[] args) {

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

al.add(10);

al.add(20);

al.add(0);

al.add(5);

al.add(25);

al.add(15);

System.***out***.println(al);

/\*\*

\* //To get the Minimum and Maximum first the elements should be in Ascending Order

\* (i1,i2)->i1.compareTo(i2) --This will allow us to have elements in Ascending Sorted Order.

\*/

Integer minValue = al.stream().min((i1,i2)->i1.compareTo(i2)).get();

System.***out***.println(minValue);

Integer maxValue = al.stream().max((i1,i2)->i1.compareTo(i2)).get();

System.***out***.println(maxValue);

}

}

toArray() method:

package com.chinmay.streamapi;

import java.util.\*;

import java.util.stream.\*;

public class ToArrayStreamMethod {

public static void main(String[] args) {

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

al.add(10);

al.add(20);

al.add(0);

al.add(5);

al.add(25);

al.add(15);

System.***out***.println(al);

Object[] obj=al.stream().toArray();

//System.out.println(obj);

for(Object o:obj)

{

System.***out***.println(o);

}

Integer[] obj1=al.stream().toArray(Integer[]::new);

System.***out***.println();

for(Integer i:obj1)

{

System.***out***.println(i);

}

}

}

Stream.of():

package com.chinmay.streamapi;

import java.util.stream.Stream;

public class StreamDotOfExample {

public static void main(String[] args) {

Stream<Integer> str=Stream.*of*(9,99,999,9999);

str.forEach(i->System.***out***.println(i));

Double d[]=new Double[] {10.0,10.1,10.2,10.3};

Stream<Double> str1=Stream.*of*(d);

str1.forEach(di->System.***out***.println(di));

}

}