

If we want to process objects from collection then we used stream API.

- Group of elements
- Array
- collection

Stream is an interface present in java.util.stream package.

We create the object of stream -

```
Stream s = c.Stream();  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class StreamApiD {  
    public static void main(String[] args){  
        ArrayList<Integer> l = new ArrayList<>();  
        l.add(5);  
        l.add(10);  
        l.add(15);  
        l.add(20);  
        l.add(25);  
        l.add(30);  
        l.add(35);  
  
        System.out.println(l);  
  
        List<Integer> l2=l.stream().filter(i->i%2==0).collect(Collectors.toList());  
  
        System.out.println(l2);  
    }  
}  
  
Output - [5, 10, 15, 20, 25, 30, 35]  
[10, 20, 30]
```

By using Stream reduced the number of line code.

example : above example we do the filter operation, by using stream we done with in a single line but normally required 5 to 6 line.

1.

**Filter :-** it return Boolean (predicate), if you filter the data based on some condition then we used the filter in stream api.

It means input from collection is - 10 inputs

Using filter output will be 0 to 10 or <10

Ex- l.stream().filter(predicate)

2.

**Map**:- when we perform same calculation on each object present in collection then we should go for map.

It means input from collection is - 10

Then using map output will be - 10.

Ex- `l.stream().map(Function)`

**3. count()** :- it return the long value. It return how many objects in the stream.

Ex- `long cnt = l.stream().filter(i->i%2==0).count();`

**4. collect()** :- used for collect the stream result.

**5. sorted()** :- used for sort stream object.

1. Sorted() - default sorting order.

Internally call:-

```
sorted((i1,i2)->i1.compareTo(i2)) (natural-Asc)
sorted((i1,i2)->-i1.compareTo(i2)) (reverse natural Desc)
```

```
sorted((i1,i2)->i2.compareTo(i1)) (natural-Asc)
sorted((i1,i2)->-i2.compareTo(i1)) (reverse naturalDesc)
```

2. Sorted(comparator) : customized sorting order.

`Sorted((i1,i2)->((i1<i2)?1:(i1>i2)?-1:0))`

- Compare() method contains two parameters so (i1,i2)

- (*i1*<*i2*)?1 means positive obj1 come before obj2

- (*i1*>*i2*)?-1 means negative obj1 come after obj2

- 0 obj1 and obj2 are equals.

Ex-

```
List<Integer> 13=l.stream().sorted().collect(Collectors.toList());
[5, 10, 15, 20, 25, 30, 35]
```

```
List<Integer> 14=l.stream().sorted((i1,i2)->((i1<i2)?1:(i1>i2)?-1:0)).collect(Collectors.toList());
[35, 30, 25, 20, 15, 10, 5]
```

Ex – increasing length and for equals alphabetical order

```
ArrayList<String> s = new ArrayList();
s.add("A");
s.add("AAA");
s.add("AAAAAA");
s.add("AA");
s.add("AAAAAAA");
s.add("AA");

Comparator<String> c=(i1,i2)->{
    int len1=i1.length();
    int len2=i2.length();
    if(len1<len2) return 1;
    else if(len1>len2) return -1;
    else return i1.compareTo(i2);
};
List<String> ls= s.stream().sorted((i1,i2)->{
    int len1=i1.length();
    int len2=i2.length();
    if(len1<len2) return 1;
    else if(len1>len2) return -1;
    else return i1.compareTo(i2);
}).collect(Collectors.toList());
List<String> ls1=
s.stream().sorted(c).collect(Collectors.toList());
System.out.println(ls);

Output:- [AAAAAAA, AAAAAA, AAA, AA, AA, A]
```

## 6. min() and max() in stream

```
// min(comparator)  max(comparator)

int min = l.stream().min((i1,i2)->i1.compareTo(i2)).get();

int max = l.stream().max((i1,i2)->-i1.compareTo(i2)).get();

System.out.println("Min : "+min+ " Max : "+max);

Output is:- Min : 5 Max : 35
```

## 7. forEach() -

```
l.stream().forEach(function)

l.stream().forEach(System.out::println)
```

```
ex- l.stream().forEach(i->{System.out.println(i*1)});
```

8. `toArray()` -

9. `Stream.of()`:-

Ex-1

```
Stream ss = Stream.of(9,99,999,9999);  
  
OR
```

```
Stream<Integer> ss = Stream.of(9,99,999,9999);  
  
ss.forEach(System.out::println);  
  
output : - 9  
         99  
         999  
         9999
```

Ex-2

```
Integer[] i = {10,20,30,40,50};  
Stream.of(i).forEach(System.out::println);
```

JDK 1.8

Remove memory → PermGen.

Introduced Memory → metaspace.

① PermGen - (Permanent Generation) is a special heap space separated from main memory heap.

- JVM stores all the static content in this memory section.

this includes - static methods, primitive variable and reference of static objects.

String pool also part of this memory.

default maximum size →

for 32 bit JVM →

~~Perm~~ is 64 MB

for 64 bit JVM →

is 82 MB.

We can change the default size. -

-XX:PermSize=[size]

-XX:MaxPermSize=[ ]

with the limited memory size in PermGen, there is a problem of OutOfMemory error.

- not properly garbage collected so memory leakage problem.

② Metaspace - It has replaced the older PermGen memory space.

- memory size grows automatically by default.

→ MetaspaceSize and MaxMetaspaceSize

→ we can set the metaspace upper bounds.

→ MinMetaspaceFreeRatio and maxMetaspaceFreeRatio

- metadata capacity free after garbage collection to avoid a reduction in the space.

- JVM reduces the chance to get the OutOfMemory Error.

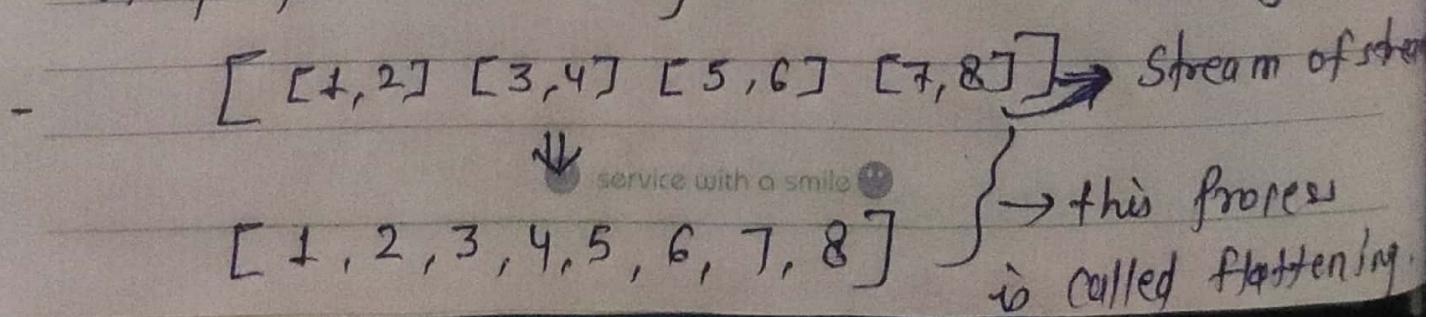
➤ Difference between map() and flatMap();

① map()  $\leftrightarrow$

- It process stream of values.
- It perform only mapping
- It produces single value for each input value  
Ex+ one user have only one email.
- It provide one-to-one mapping.
- Data transformation : From stream to stream.
- use this method when the mapper function is producing a single value for each input value.

② flatMap()  $\leftrightarrow$

- It process stream of stream of values
- It perform mapping as well as flattening .



- It produces multiple value for each input value.  
for ex - one user have multiple phone no.
- It provide one-to-many mapping.
- Data transformation : from Stream < stream to stream>
- use this method when the mapper function is producing multiple value for each input value.

Ex -

```
new Customer( id: 101, name: "nilesh", email: "nilesh@gmail.com",
               phones: asList("39934042", "39568989"))
```

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

```
{   // customer → customer.getEmail() // one to one  
    // mapping
```

```
List <string> email = customers.stream().map( customer  
→ getEmail ()).collect (Collectors. to  
List ()),
```

O/P → [john@gmail.com, smit@gmail.com, nilesh@gmail.com]

for  
above

→ Stream of stream by using map()

1/ Customer → customer.getPhoneNumbers() → One to many mapping

List<List<String>> phoneNo = customers.stream().map(  
customer.getPhoneNumbers()).collect  
(Collectors.toList());

O/p → [[39798978, 3894789], [8945789, 989748]]

→ Stream of stream by using flatMap()

List<String> phones = customers.stream().flatMap(  
customer.getPhoneNumber().stream())  
• collect(Collectors.toList());

O/p → [39798978, 3894789, 8945789, 989748]

## \* Convert List < Employee > to Map.

- ①  $\text{Map} < \text{Integer}, \text{Employee} >$  empName = list.stream().collect(Collectors.toMap(Employee :: empId, employee -> employee))
- ②  $\text{Map} < \text{Integer}, \text{String} >$  empMap = list.stream().collect(Collectors.toMap(Employee :: empId, Employee :: empName));
- ③  $\text{Map} < \text{Integer}, \text{String} >$  emp = list.stream().collect(Collectors.toMap(x -> x.getId(), x -> x.getName));

\* Garbage Collection in Java - what is GC and how it works in the JVM.

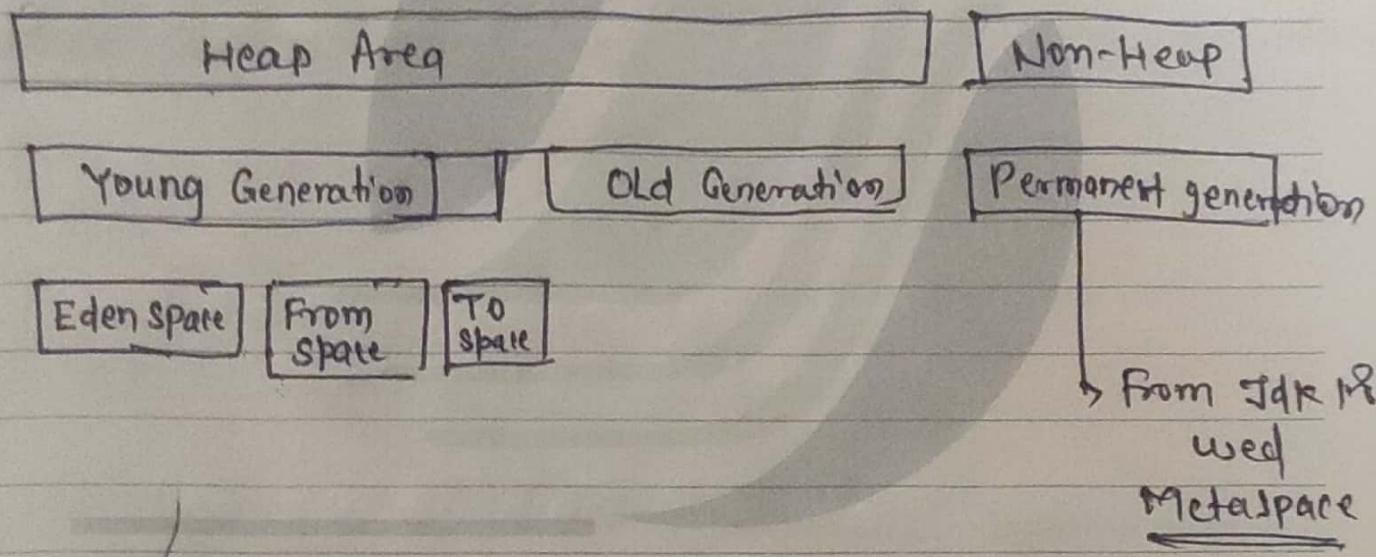
- Garbage collection is used to increase memory by destroying the unused object.
- Java garbage collection is the process by which java program performs automatic memory management.
- When java program run on the JVM, object created on the heap.
- The heap memory consists of two types of objects-
  - ① Live - these objects are being used.
  - ② Dead - these objects are no longer used.

when there are no reference to an object, it is assumed to be dead.

Candidate for garbage collection -

- ①  $\text{Student } s = \text{new Student}();$        $s = \underline{\text{null}};$       } By making a reference null.
- ②  $\text{register}(\text{new Student}());$       } By using an anonymous object.

- ★ → Java garbage collection is an automatic process. Programmer does not need to explicitly delete the objects.
- Each JVM can implement its own version of garbage collection.
- you can use -Xms and -Xmx to max and min heap size.



→ Young Generation - newly created objects start in the young generation. The young generation divided into three sub categories

- ① Eden space
- ② From space
- ③ To space

- Eden has all objects (live and dead)
- all dead objects are removed from Eden, all live objects are moved to From Space
- So at any time, one (from & To) space is always empty.

Stock Holding Corporation of India Limited

Date: / /

When spaces ; reach a certain threshold , then move  
the object to old generation.

→ Permanent Generation (PermGen & Metaspace)

↳ back side,

\* By using Java 8. identify the Employee whose name is "Aakash".

①.

```
List<Employee> emp = new ArrayList<Employee>();
```

```
emp.add(new Employee(103, "Aakash", 27));  
emp.add(new Employee(104, "Nilash", 25));  
emp.add(new Employee(105, "Atul", 28));
```

Optional<Employee> empf = emp.stream().filter(e → e.getEmpName()  
•.equals("Aakash")).findFirst();

② Remove the duplicate from List :

```
List<Integer> list = new ArrayList<>();
```

```
list.add(1);  
list.add(2);  
list.add(1);  
list.add(2);
```

```
List<Integer> list1 = list.stream().distinct().collect(Collectors.toList());
```

\* functional Interface - Java 8. (types)  $\rightarrow$  4 types

- ① Predicate  $\rightarrow$  boolean result
- ② consumer  $\rightarrow$  no result
- ③ function  $\rightarrow$  Input and output
- ④ supplier  $\rightarrow$  no input only supply.

example -

① Predicate.  $\rightarrow$  (contain one method test())

public class Mainclass  
{

|| functional Interface types - 4 types  
|| i. predicate -- boolean result.

public static void main(String[] args)

Predicate <String> check = str  $\rightarrow$  str.length() > 5;

s.o.println(check.test("nileshG"));

{  
↓  
method of predicate function  
interface.

o/p - true.

② Consumer  $\Rightarrow$  consumer modifies data no output.  
- it contain method accept()

public class Person  
{

    private string name;

    // Setters and getters.

public class MainClass

{  
    public static void main (String args[])

        Person p = new Person();

    Consumer<Person> check = t  $\rightarrow$  t.setName ("play Java");

    check.accept(p)

    System.out.println (p.getName());

}

}

o/p - play Java.

③ function  $\Rightarrow$  It works on Input and provide output  
It contain method apply()

public class Test

{

P. S. V. main(string args[])

{

Function < Integer, string > test = t  $\rightarrow$  t \* 10 + 'data'  
↓              ↓  
Input          Output

S.O. pIn(test.apply(2));

}

Output -

20 data

④ Supplier  $\Rightarrow$  It provide only output.

It contain one method get()

Q. public class Mainclass

{  
    f. s. v. main(String[] args)  
    {

Supplier<Double> A = () → Math.random();

s.o.pn (s.get()); // no input.

}  
}

O/P- 0.1369,

\* Java 8 - SummingInt() & SummingLong().

→ is a method of Collector class.

Ex -

class Employee

{  
private int id;  
private int empId;  
private String name;  
private int salary;

}

public class Test

{

p. s. v. main (String[] args)

{

int total = emp.stream().collect (Collectors.summing

- Int (Employee :: getSalary));

s.out ("total emp salary" + total);

{}

Note:- for long summingLong()

★ Get Second highest salary using Dotski.e.

www.shell.com

Date :

Optional<Emp> e = l.stream().sorted(Comparator.comparingDouble(Emp::getSalary).reversed()).skip(1).findFirst();  
s.o.println(emp.get());

**★ Java 8 -** program, print only Id, whose salary greater than 70,000.

public class StreamTest  
{

public static void main(String args[]){

List<Product> p = new ArrayList();

p.add(new Product(1, "Nil", 50000));  
 p.add(new Product(2, "Aak", 60000));  
 p.add(new Product(3, "Atul", 80000));  
 p.add(new Product(4, "Nk", 90000));

List<Integer> list = p.stream().filter(n->n.getPrice() >

70000).map(m->m.getId()).collect(Collectors.toList());

s.o.println(list);

or

list.forEach(s->s.o.println(s));

}

o P>

3  
4

3  
4

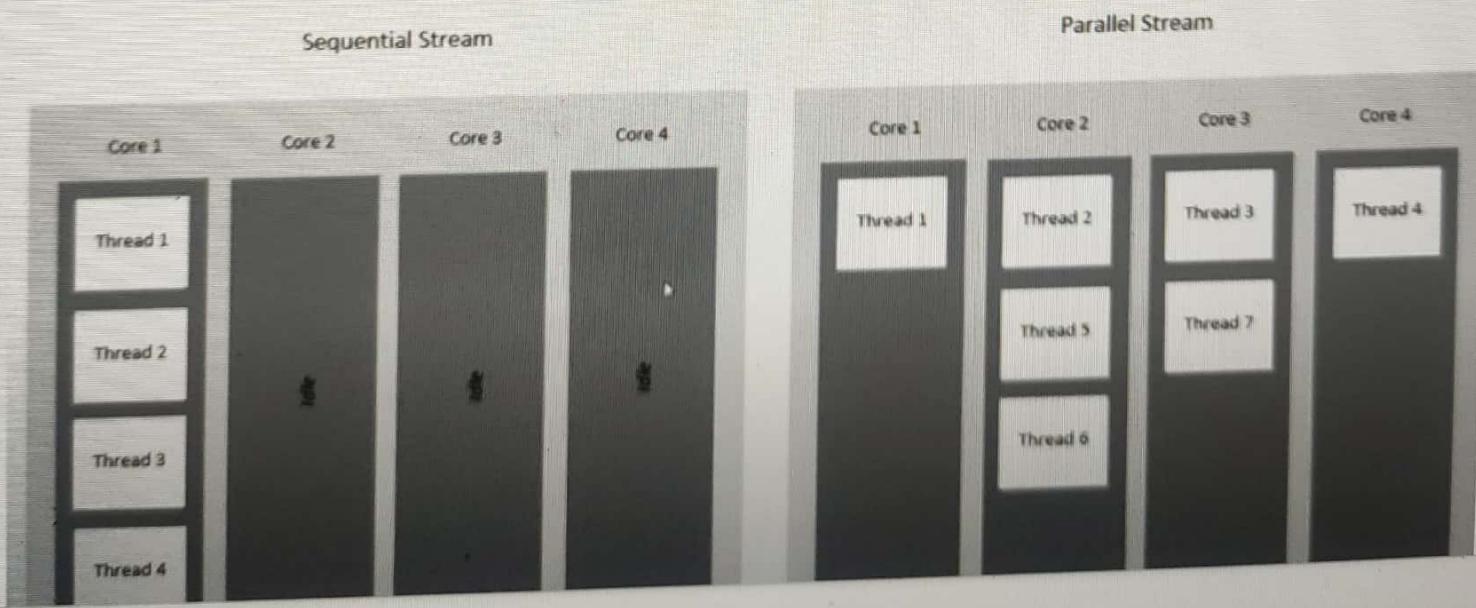
→ id, whose Price greater  
than 70,000.

## **Sequential Stream and Parallel Stream :-**

**Sequential Stream** : java code process in one stream that are executed in single core

**Parallel Stream** : java code divided into multiple stream that are executed in separate cores, final results is the combination of the individual outcomes.

- Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel streams, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes



```
public static void main(String[] args) {  
    long start=0;  
    long end=0;  
  
    start=System.currentTimeMillis();  
    IntStream.range(1,100).forEach(System.out::println);  
    end=System.currentTimeMillis();  
    System.out.println("Plain stream took time : "+(end-start));  
  
    System.out.println("=====");  
    start=System.currentTimeMillis();  
    IntStream.range(1,100).parallel().forEach(System.out::println);  
    end=System.currentTimeMillis();  
    System.out.println("Parallel stream took time : "+(end-start));  
}  
}
```

Sequential Stream or plain Stream took : 73 milliseconds (ms)

Parallel stream took : 23 milliseconds (ms)

In Sequence Stream : order will be maintained like – 1,2,3,4,5...., bcz single core is used.

But in Parallel Stream : order will not be maintained, bcz different core is used.

## StringPalindrom.java - Spring Tool Suite 4

File Project Run Window Help



StringPalindrom.java

```
1 package Eight;
2
3 import java.util.stream.Collectors;
4
5
6 public class StringPalindrom {
7
8     public static void main(String[] args) {
9
10         String str="MOM";
11
12         // reverse a String using Java 8
13         String reverseStr = Stream.of(str)
14             .map(string -> new StringBuilder(string).reverse())
15             .collect(Collectors.joining());
16
17         if(str.equals(reverseStr)) {
18             System.out.println("Palindrom");
19         }
20
21
22 }
```

Console Problems Debug Shell

<terminated> StringPalindrom [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\lib\rt.jar

## RemoveDuplicateAnOnlyString.java - Spring Tool Suite 4

```
Project Run Window Help
RemoveDuplicateAnOnlyString.java ✘
1 package Eight;
2 import java.util.ArrayList;
3
4
5 public class RemoveDuplicateAnOnlyString {
6     // List<String> ltStr=Arrays.asList("Mouse", "Keypad", "CPU", "", "1334", "432", "CPU");
7     //o/p---"Mouse,Keypad,CPU"
8
9
10    public static void main(String[] args) {
11        List<String> ltStr=Arrays.asList("Mouse", "Keypad", "CPU", "", "1334", "432", "CPU");
12
13
14        // OR in one line
15        List<String> StringListInOneLine = ltStr.stream().distinct()
16            .filter(x -> !x.matches("\\d+")).collect(Collectors.toList());
17        System.out.println(StringListInOneLine);
18
19
20
21
22
23
24
25
26
```

Console Problems Debug Shell

```
<terminated> RemoveDuplicateAnOnlyString [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20200722-1258
[Mouse, Keypad, CPU, ]
[Mouse, Keypad, CPU, , 1334, 432]
[Mouse, Keypad, CPU, ]
[1334, 432]
```

```
RemoveDuplicateAnOnlyString.java CityWiseSalarySum.java
1 package Eight;
2
3 import java.util.ArrayList;
4
5 public class CityWiseSalarySum {
6
7     public static void main(String[] args) {
8
9         List<Employee> lst = new ArrayList<Employee>();
10
11         lst.add(new Employee(1, "Nilesh", "Pune", 200000));
12         lst.add(new Employee(2, "Aditya", "Pune", 500000));
13         lst.add(new Employee(3, "Akash", "Pune", 300000));
14         lst.add(new Employee(4, "Amol", "Mumbai", 400000));
15         lst.add(new Employee(5, "Neeraj", "Mumbai", 600000));
16         lst.add(new Employee(6, "Mota Singh", "Navi Mumbai", 200000));
17
18         Double eListSum= lst.stream()
19             .filter(a->a.getCity().equals("Pune"))
20             .collect(Collectors.summingDouble(Employee::geteSalary));
21
22
23
24 }
```

Console Problems Debug Shell

<terminated> CityWiseSalarySum [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-1000000.0

```
DepartmentWise.java ✘
10 public class DepartmentWise {
11
12     public static void main(String[] args) {
13
14
15
16     List<Emp> lst = Arrays.asList(new Emp("Nilesh", 165000, "IT"), new Emp("Aditya", 175000, "IT"),
17         new Emp("Vinod", 210000, "ENC"), new Emp("Sachin", 200000, "ENC"));
18
19
20     // Group wise max salary
21     Map<String, Object> mp = lst.stream().collect(Collectors.groupingBy(Emp::getDept, Collectors
22         .collectingAndThen(Collectors.maxBy(Comparator.comparingDouble(Emp::getEsal)), Optional::get)));
23     System.out.println("Group wise max salary :" + mp);
24
25
26     // max salary
27     Optional<Emp> e = lst.stream().collect(Collectors.maxBy(Comparator.comparingDouble(Emp::getEsal)));
28     System.out.println(e);
29 }
```

Console ✘ Problems ✘ Debug Shell

```
<terminated> DepartmentWise [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\Group wise max salary :{ENC=Emp [enm=Vinod, esal=210000.0, dept=ENC], IT=Emp [enm=Aditya, esal=175000.0, dept=IT]}Optional[Emp [enm=Vinod, esal=210000.0, dept=ENC]]
```

```
DepartmentWise.java JavaDuplicated1.java
1 package org;
2 import java.util.Arrays;
3
4 public class JavaDuplicated1 {
5
6     public static void main(String[] args) {
7
8         // 3, 4, 9
9         List<Integer> list = Arrays.asList(5, 3, 4, 1, 3, 7, 2, 9, 9, 4);
10
11        Set s = new HashSet<>();
12        Set duplicate = list.stream().filter(n -> !s.add(n)).collect(Collectors.toSet()); // .forEach(System.out::println);
13
14        List<Integer> lInt = list.stream().filter(n -> !s.add(n)).collect(Collectors.toList());
15
16        System.out.println("lInt list ::" + lInt);
17
18        // !s.add(n) -> duplicate elements which is not added into set
19
20        System.out.println(duplicate);
21        duplicate.forEach(System.out::println);
22        // result.forEach(System.out::println);
23
24
25
26 }
```

```
Console Problems Debug Shell
<terminated> JavaDuplicated1 (1) [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149
lInt list ::[5, 3, 4, 1, 3, 7, 2, 9, 9, 4]
[3, 4, 9]
3
4
9
```

```
*ProductStream.java ✘
8 public class ProductStream {
9     // Product type wise and then price wise sorting
10    public static void main(String[] args) {
11        List<Product> pList = new ArrayList<Product>();
12        pList.add(new Product(1, "Activa", 100000));
13        pList.add(new Product(2, "Jupiter", 400000));
14        pList.add(new Product(3, "FZ", 200000));
15        pList.add(new Product(4, "KTM", 150000));
16        pList.add(new Product(5, "Dukati", 170000));
17        pList.add(new Product(6, "Access", 50000));
18    }
19    List<Product> pLst1 = pList.stream()
20        .sorted(Comparator.comparing(Product::getPType).thenComparing(Product::getPrice))
21        .collect(Collectors.toList());
22    pLst1.forEach(System.out::println);
23    List<Product> pLst2 = pList.stream()
24        .sorted(Comparator.comparing(Product::getPType).thenComparing(Product::getPrice).reversed())
25        .collect(Collectors.toList());
26    System.out.println("Reversed ::");
27    pLst2.forEach(System.out::println);
```

Console ✘ Problems ✘ Debug Shell

```
<terminated> ProductStream [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v20210721-1149\jre\bin\javaw.exe
Product [pType=Access, price=50000.0]
Product [pType=Activa, price=100000.0]
Product [pType=Dukati, price=170000.0]
Product [pType=FZ, price=200000.0]
Product [pType=Jupiter, price=400000.0]
Product [pType=KTM, price=150000.0]
Reversed :::
Product [pType=KTM, price=150000.0]
Product [pType=Jupiter, price=400000.0]
Product [pType=FZ, price=200000.0]
Product [pType=Dukati, price=170000.0]
Product [pType=Activa, price=100000.0]
```

## RemoveDuplicate.java

```
1 package Eight;
2
3 public class RemoveDuplicate {
4
5     public static void main(String[] args) {
6
7         String str ="abbcccdeeek";
8
9         char[] ch = str.toCharArray();
10
11        String str1="";
12        for(char c : ch) {
13            |
14            if(str.indexOf(c) == str.lastIndexOf(c)) {
15                str1=str1 + " " + c;
16            }
17        }
18        System.out.println(str1);
19
20    }
21
```

I

Console Problems Debug Shell

<terminated> RemoveDuplicate [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.jdt.adk

```
SecondHeigestNumber.java ✘
11 public class SecondHeigestNumber {
12     public static void main(String[] args) {
13         List<Integer> list = new ArrayList<Integer>();
14         list.add(4);
15         list.add(7);
16         list.add(5);
17         list.add(9);
18         list.add(10);
19
20         int l1 = list.stream().sorted(Comparator.reverseOrder()).skip(1).findFirst().get();
21         System.out.println(l1);
22
23         int[] a = {1,3,9,8,5};
24         int secondHighest = Arrays.stream(a).boxed()
25             .sorted(Comparator.reverseOrder()).skip(1).findFirst().get();
26         System.out.println(secondHighest);
27     }
28 }
```

Console Problems Debug Shell

<terminated> SecondHeigestNumber [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-1149

9  
8

SingleAndParlleStream.java

```
1 package Eight;
2
3 import java.util.stream.IntStream;
4
5 public class SingleAndParlleStream {
6
7     public static void main(String[] args) {
8         long start=0;
9         long end=0;
10        start=System.currentTimeMillis();
11        IntStream.range(1, 100).forEach(System.out::println);
12        end=System.currentTimeMillis();
13        System.out.println(" sequential stream time :: "+(end-start));
14
15        start=System.currentTimeMillis();
16        IntStream.range(1, 100).parallel().forEach(System.out::println);
17        end=System.currentTimeMillis();
18
19        System.out.println("Parallel stream time :: "+(end-start));
20    }
21}
```

Console Problems Debug Shell

```
<terminated> SingleAndParlleStream [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.2.v202
60
61
23
24
36
11
12
19
20
21
Parallel stream time :: 12
```

CharacterCount.java

```
1 package Eight;
2
3 import java.util.Map;
4 import java.util.function.Function;
5 import java.util.stream.Collectors;
6 import java.util.stream.IntStream;
7
8 public class CharacterCount {
9     public static void main(String[] args) {
10         String word = "AAABBB123";
11
12         IntStream s = word.codePoints();
13
14         Map<String, Long> charCount = s.mapToObj(Character::toString)
15             .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()));
16
17         System.out.println(charCount);
18     }
19 }
20
```

I

Console Problems Debug Shell

<terminated> CharacterCount (1) [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-1149\re

```
*SortingBaseOnValue.java */
1 package Eight;
2
3 import java.util.HashMap;
4
5 public class SortingBaseOnValue {
6
7     public static void main(String[] args) {
8
9         HashMap<Integer, String> hm = new HashMap<>();
10
11        hm.put(5, "A");
12        hm.put(2, "C");
13        hm.put(4, "F");
14        hm.put(3, "D");
15        hm.put(7, "B");
16        hm.put(1, "E");
17
18        LinkedHashMap<Object, Object> hmm = hm.entrySet().stream()
19            .sorted((e1, e2) -> e1.getValue().compareTo(e2.getValue()))
20            .collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue(), (e1, e2) -> e1, LinkedHashMap::new));
21
22        Map<Object, Object> hmm1 = hm.entrySet().stream()
23            .sorted((e1, e2) -> e1.getValue().compareTo(e2.getValue()))
24            .collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue()));
25
26        System.out.println(hmm);
27
28
29
30 }
```

Console Problems Debug Shell  
<terminated> SortingBaseOnValue [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-  
{5=A, 7=B, 2=C, 3=D, 1=E, 4=F}  
{1=E, 2=C, 3=D, 4=F, 5=A, 7=B}

TwoDarraySum.java

```
6
7 public class TwoDarraySum {
8     public static void main(String[] args) {
9         //2 D Array Sum
10        int[][] twoDArray = {{2,5,9},{3,5,5}};
11        Integer add = Arrays.stream(twoDArray).flatMapToInt(arr->Arrays.stream(arr)).sum();
12        System.out.println(add);
13
14        // 1 D Array Sum
15        Integer[] a = {5,10,15};
16        List<Integer> l= Arrays.asList(a);
17        Integer add1 = l.stream().collect(Collectors.summingInt(Integer::intValue));
18
19        System.out.println(add1);
20
21        // 1 D Array Sum
22        List<Integer> lst = Arrays.asList(1, 2, 3, 4, 5);
23        Integer add2 = lst.stream().reduce(0, Integer::sum);
24
25        System.out.println(add2);
26
27        //Starting with 4
28        Integer[] a1 = {5,10,15,50,44,43};
29        List<Integer> l1= Arrays.asList(a1);
30
31        l1.stream().filter(n->(n%10)==4).collect(Collectors.toList()).forEach(System.out::println);
32
```

Console Problems Debug Shell

<terminated> TwoDarraySum [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-

29  
30  
15  
44