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 opertation, 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> l3=l.stream().sorted().collect(Collectors.toList());
```

[5, 10, 15, 20, 25, 30, 35]

```
List<Integer> l4=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().min((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 →

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 MaxMetaspace Size
→ we can set the metaspace upper bounds.

→ MinMetaspaceFreeRatio and maxMetaspacefree ratio
- metadata capacity free after garbage collection to avoid a reduction in the space.

- JVM reduces the chance to get the OutOfMemory Error.

Date : / /

➢ Difference between map() and flatMap() :

① map() :

- It process stream of values.
- It proform 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() :

- It process stream of stream of values
- It proform mapping as well as flattening.
- [[1,2] [3,4] [5,6] [7,8]] → Stream of stream

[1, 2, 3, 4, 5, 6, 7, 8] → this process is called 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 method when the mapper function is producing multiple value for each input value.

Ex→

new Customer (id: 101, name: "nilesh", email: "nilesh@gmail.
Arrays.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.toList());

o/p → [John@gmail.com, smit@gmail.com, nil@gmail.com]

for above

→ stream of stream by using map()

Stock Holding Corporation of India Limited

// 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.getPhoneNumbers().stream())
                    .collect(Collectors.toList());
```

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

* Convert List<Employee> to Map.

① Map<Integer, Employee> empName = list.stream().Collect
(Collectors.toMap(Employee::empId, employee->employee));

② Map<Integer, String> empMap = list.stream().Collect(Collectors.
toMap(Employee::empId, Employee::empName));

③ Map<Integer, 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 perform 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 -

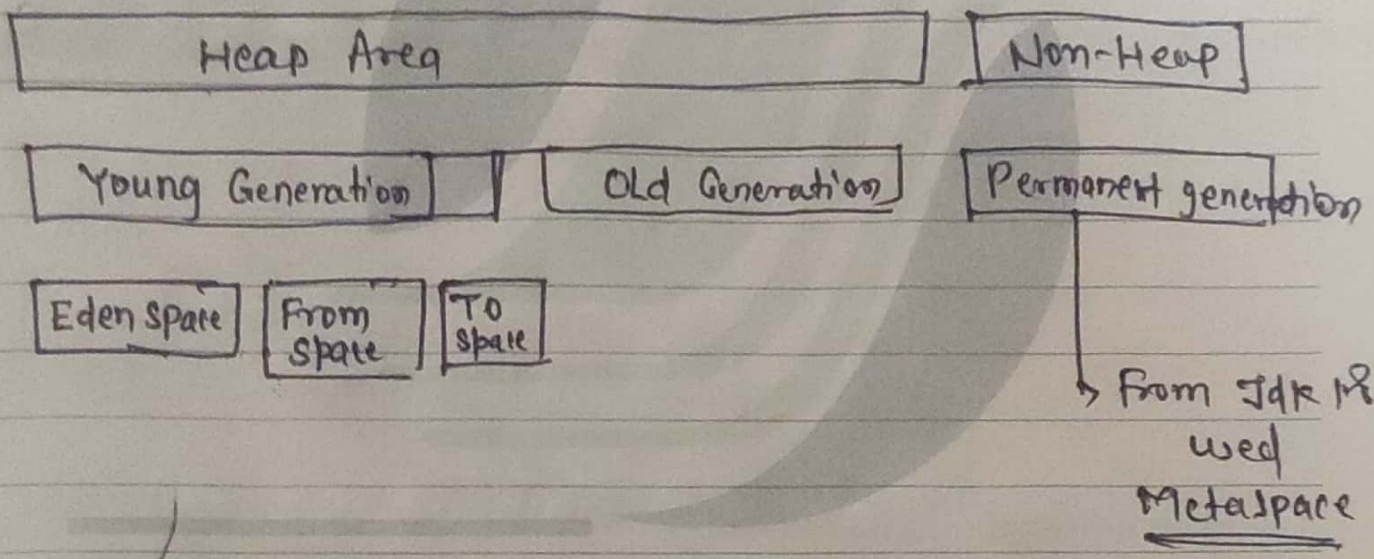
① Student s = new Student();
s = null;
→ By making a reference null.

② register(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 - newely created objects start in the young generation. the young generation divided into to sub category

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

- Eden has all object (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 & Meta space)

↳ back side.

* By using java8. identify the Employee who name is "Aakash"

(1).

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

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

```
Optional<Employee> empl = 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) → 4 types

① Predicate → boolean result
② Consumer → no result
③ function → Input and output
④ Supplier. → no input only supply.

Example -

① Predicate. → (Contain one method test())

```
public class Mainclass
{
    // functional Interface types - 4 types
    // 1. predicate -- boolean result.

    public static void main(String[] args)
    {

    Predicate<String> check = str -> str.length() > 5;

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

    }
}
```

↓ method of predicate function Interface.

o/p - true.

(2) Consumer :: Consumer modifies data no output.
- it contain method accept()

```
public class Person
{
    Private string name;

    // Setter and getter.
}

Public class Mainclass
{
    public static void main (String args[])
    {
        Person p = new person();

    Consumer<Person> check = t -> t.setName ("play Java");

        check.accept(p)

        S.O.pln (p.getName());
    }
}
```

o/p - play Java.

③ Function ⇒ It work on Input and Provide Output

It Contain method apply()

```
public class Test
{
    P. S. V. main(String args[])
    {
        Function<Integer, String> test = t -> t*10+"data";
                                    ↓          ↓
                                  Input      Output
        S.O.p(test.apply(2));
    }
}
```

O/P - 20 data

④ Supplier ⇒ It provide only Output.

It Contain one method get()

Ex:- Public class Mainclass

```
{
    p. s. v. main(String[] args)
    {
        Supplier<Double> A = () → Math.random();
        s.o.ph(A.get()); // no Input.
    }
}
```

O/P- 0.1369.

Stock Holding Corporation of India Limited

☆ Java 8- summingInt() & summingLong().

→ is a method of Collector class.

Ex→

```
class Employee
{
    Private int id;
    private int deptId;
    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.O.Pln ("total emp salary" + total);
}
}
```

Note:3 for long summingLong (

☆ Get second highest salary using Java 1.8.

```
Optional<Emp> e = l.stream().sorted(Comparator.comparing(
                    -Double(Emp::getSalary).reversed()).
                    skip(1).findFirst();

S.O.pln(emp.get());
```

Date :

☆ Java 8- Program, Print only Id, whose Salary gater 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, "Alok", 60000));
        p.add(new Product(3, "Atul", 80000));
        p.add(new Product(4, "Nik", 90000));

List<Integer> list = P.stream().filter((n)→n.getPrice()>
                                70000).map(m→m.id).collect
                                (Collectors.toList());
    S.O.pln(list);
      or
    list.forEach(s→ S.O.pln(s));
  }
}
```

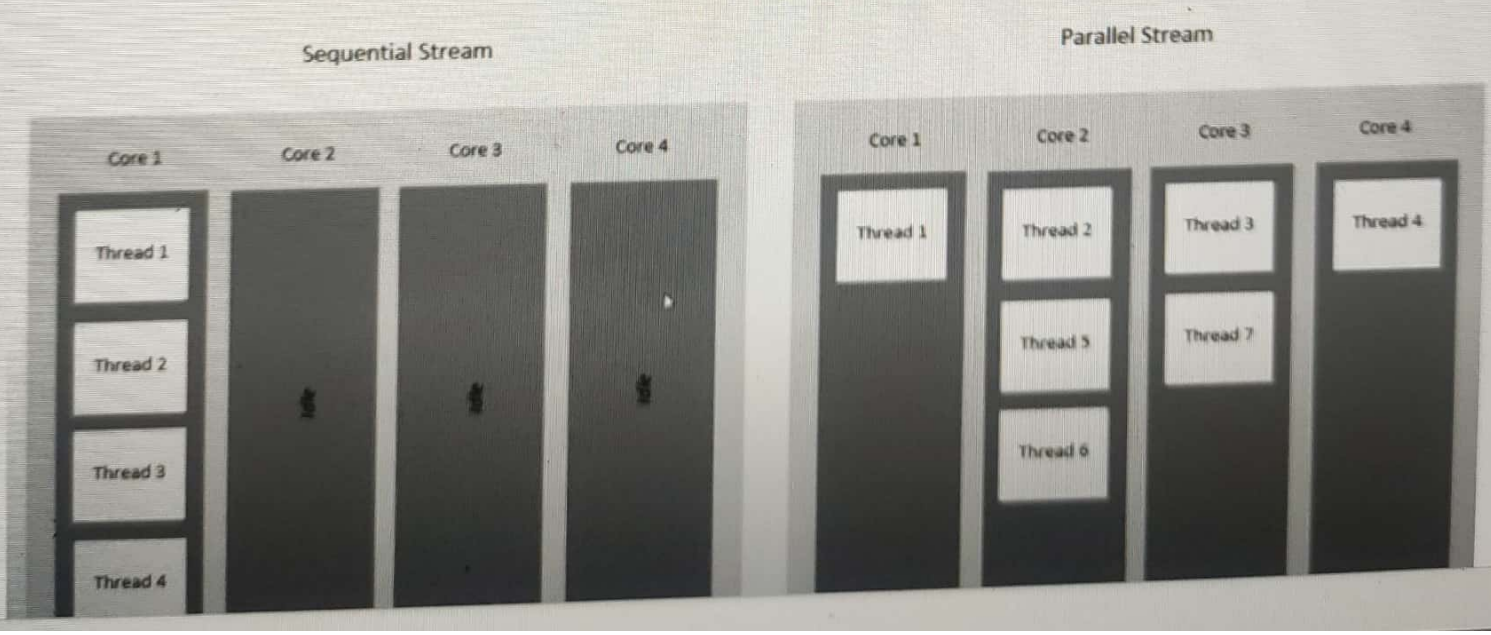
O/P-> 3 } → id, whose Price greater
4 } than 70,000.

Sequential Stream and Parllel Stream :-

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

Parllel 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



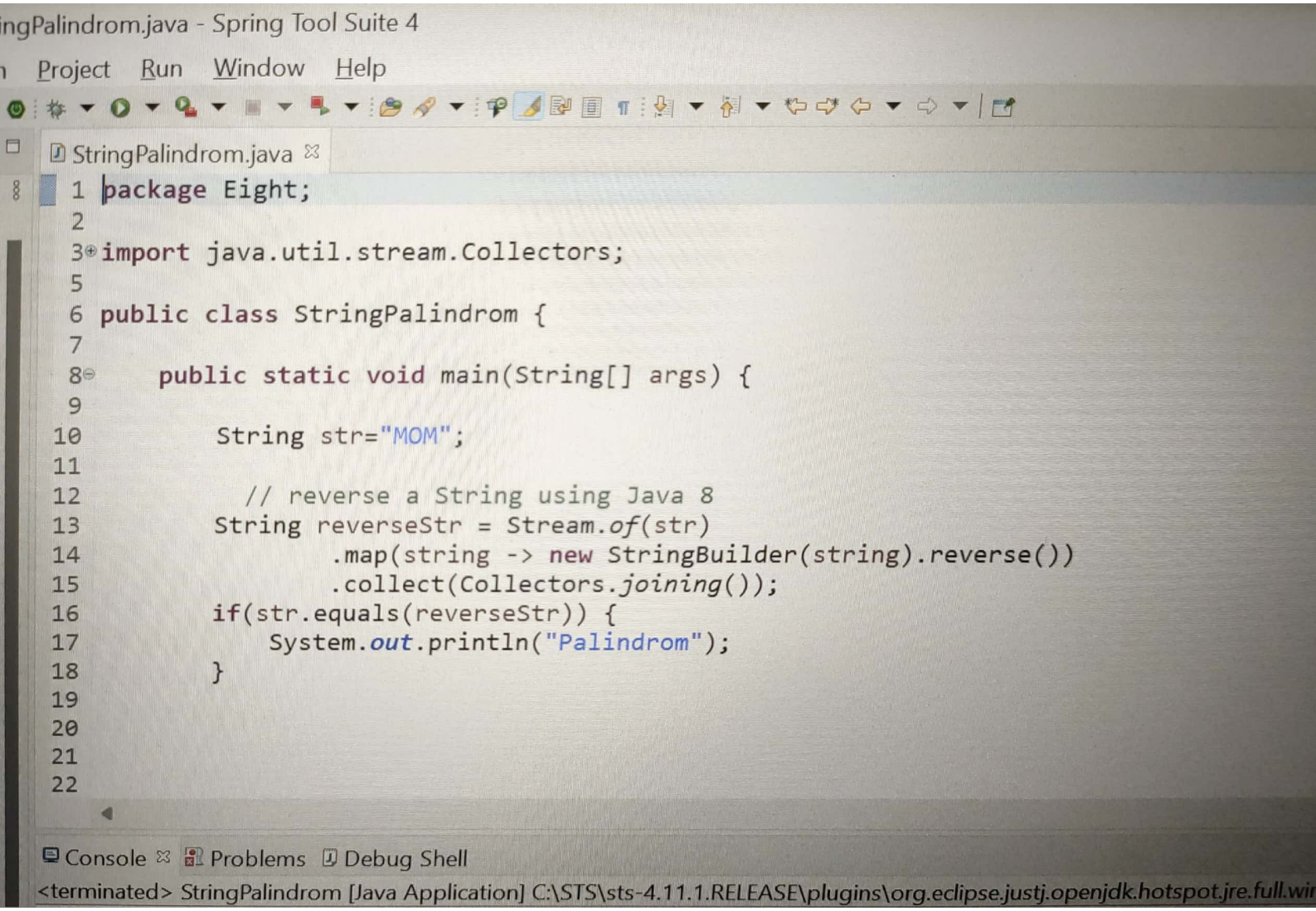
```
    public static void main(String[] args) {
        long start=0;
        long end=0;

        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));
    }
}
```

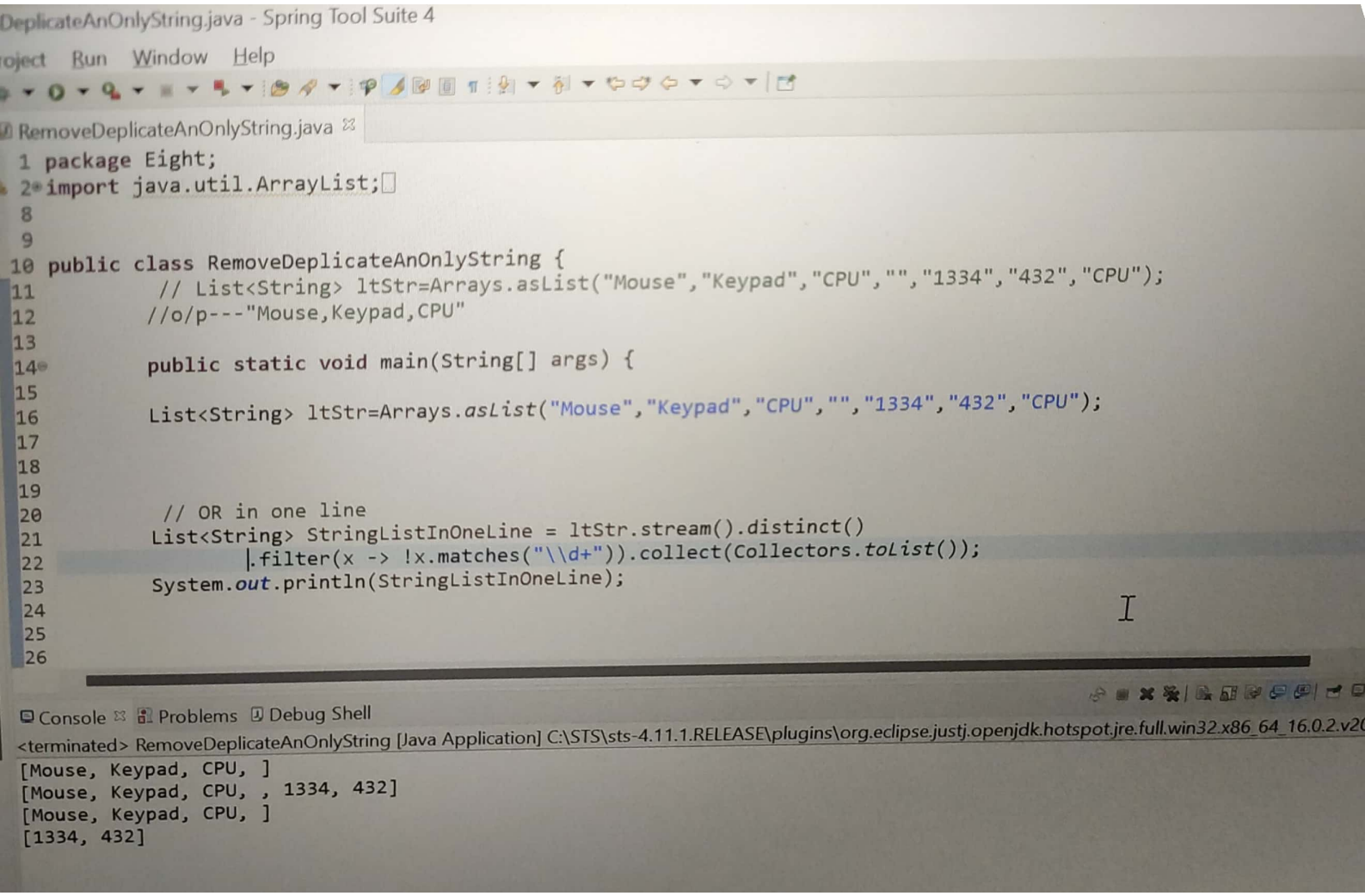
Sequential Stream or plain Stream took : 73 milliSeconds (ms)

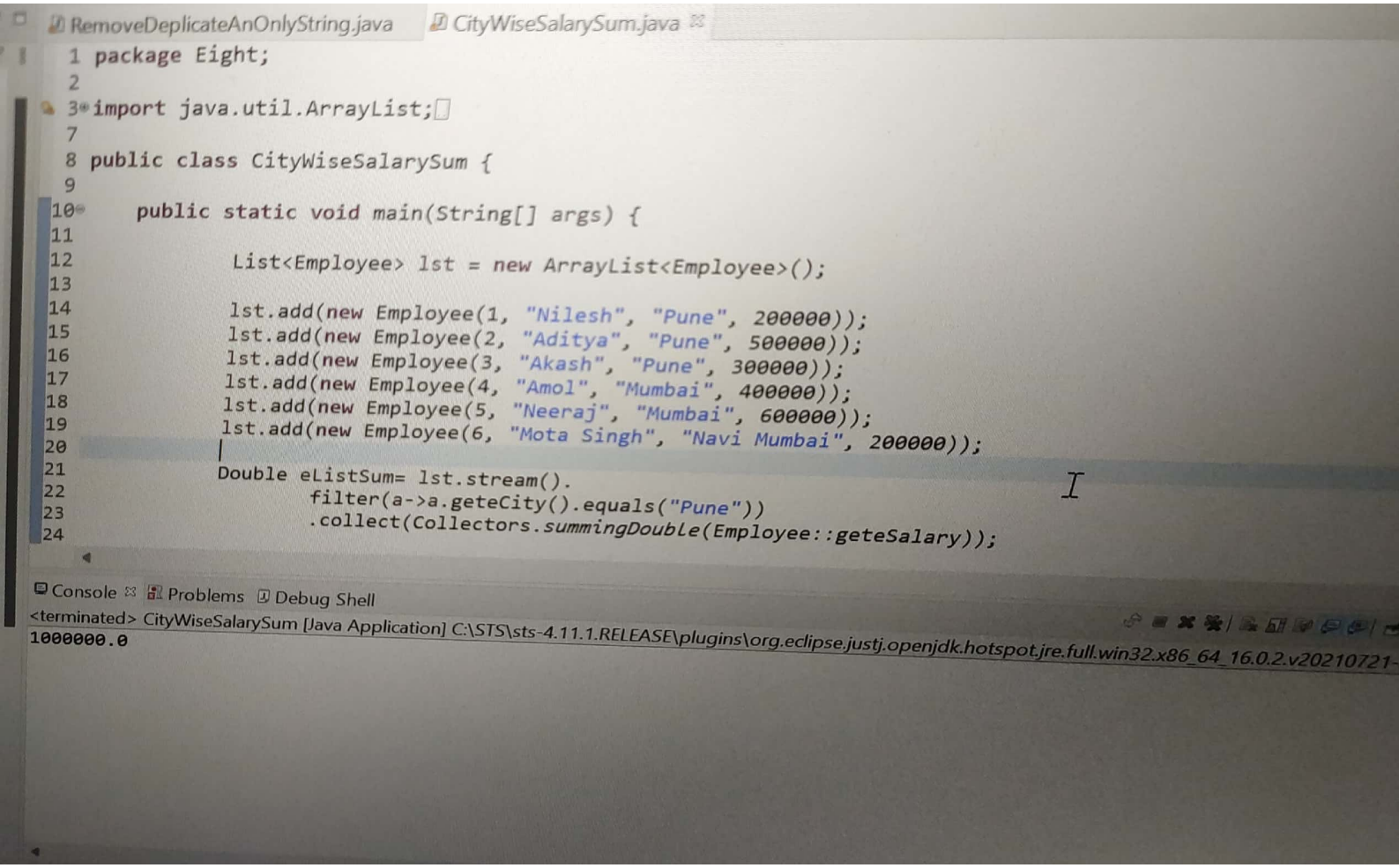
Parallel stream took : 23 miliseconds (ms)

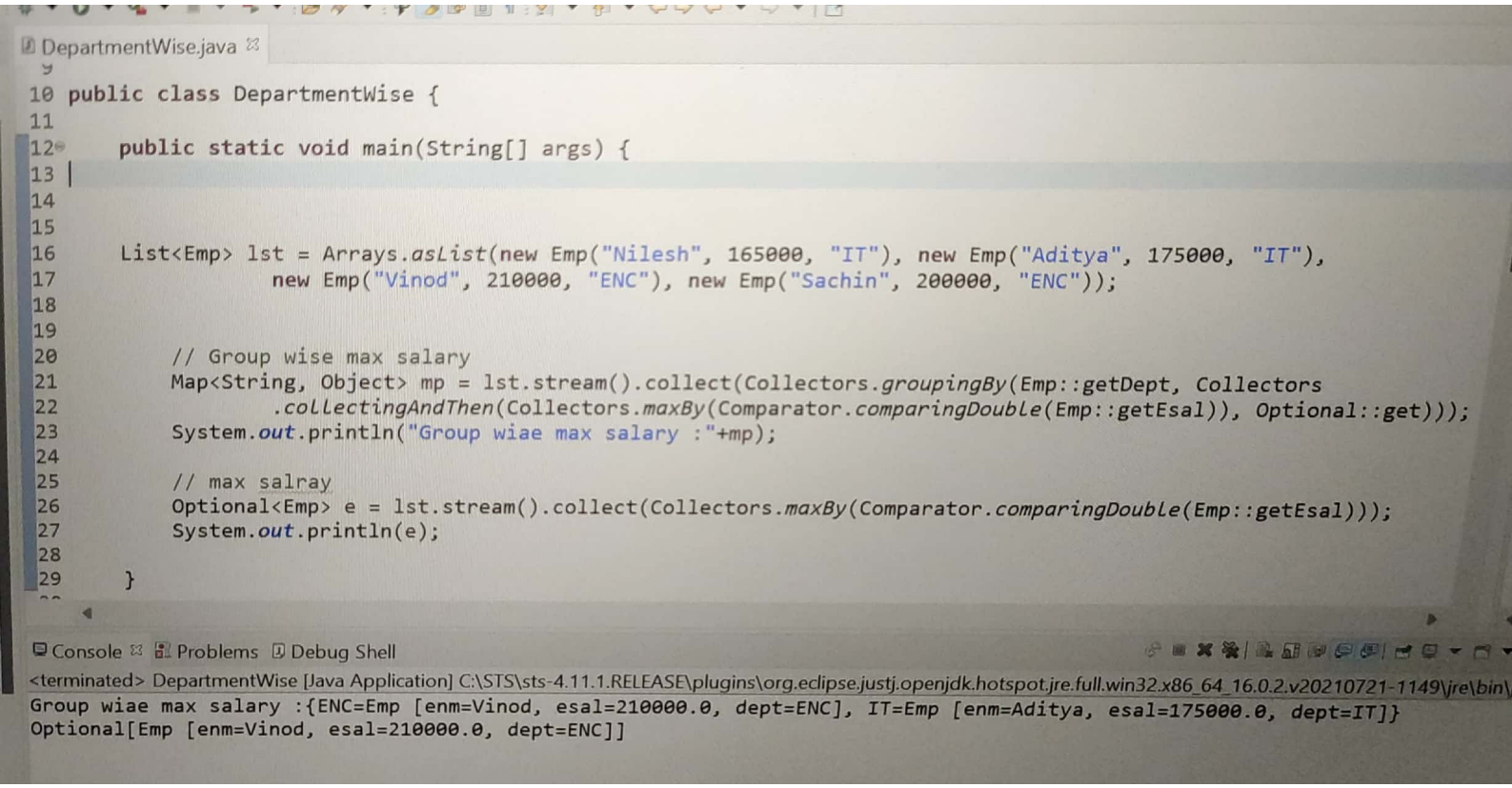
In Sequence Stream : order will be maintained like – 1,2,3,4,5...., bcz single core is used.

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









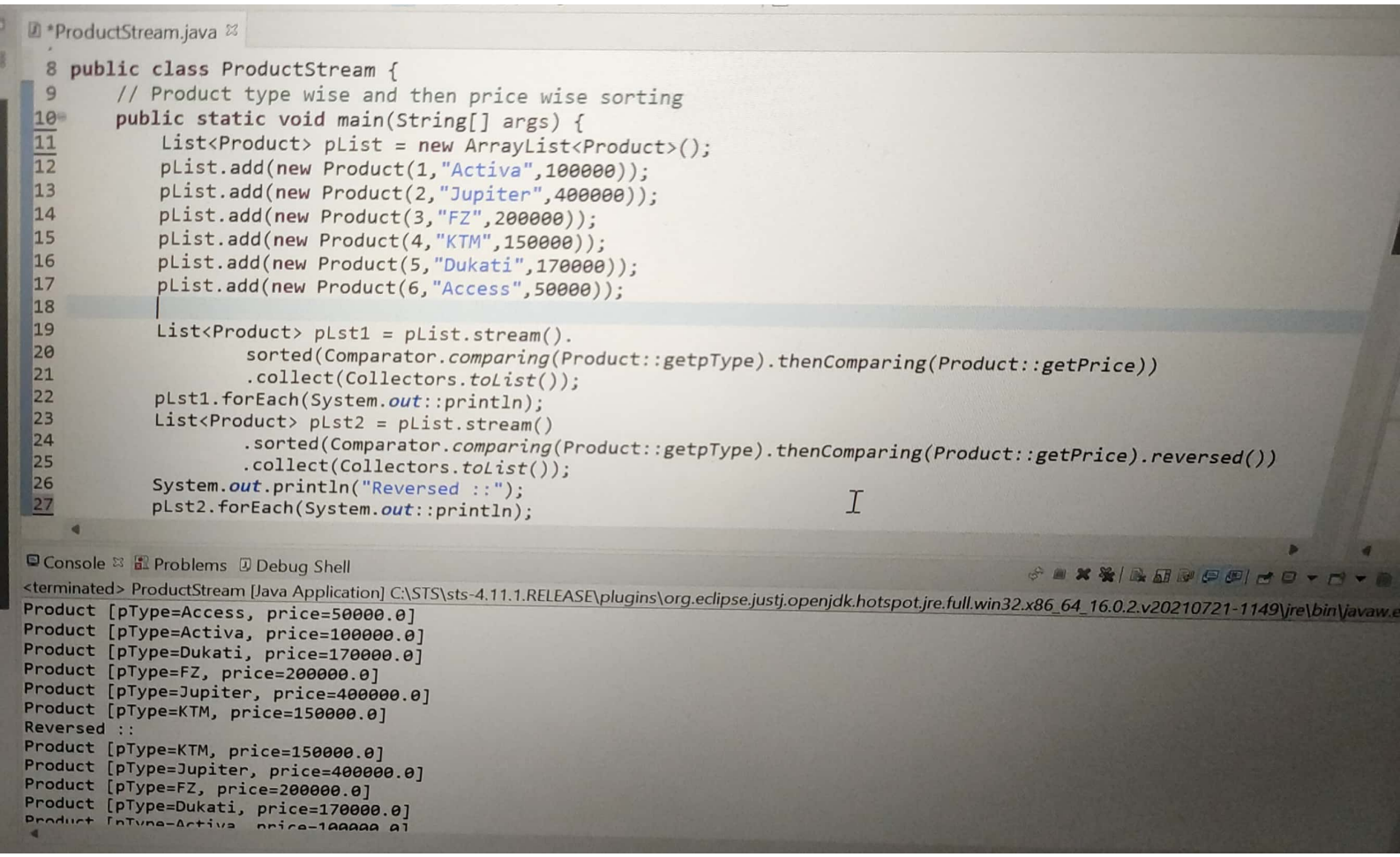
DepartmentWise.java | JavaDuplicated1.java

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

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.v20210721-114

```
lInt list ::[5, 3, 4, 1, 3, 7, 2, 9, 9, 4]
[3, 4, 9]
3
4
9
```



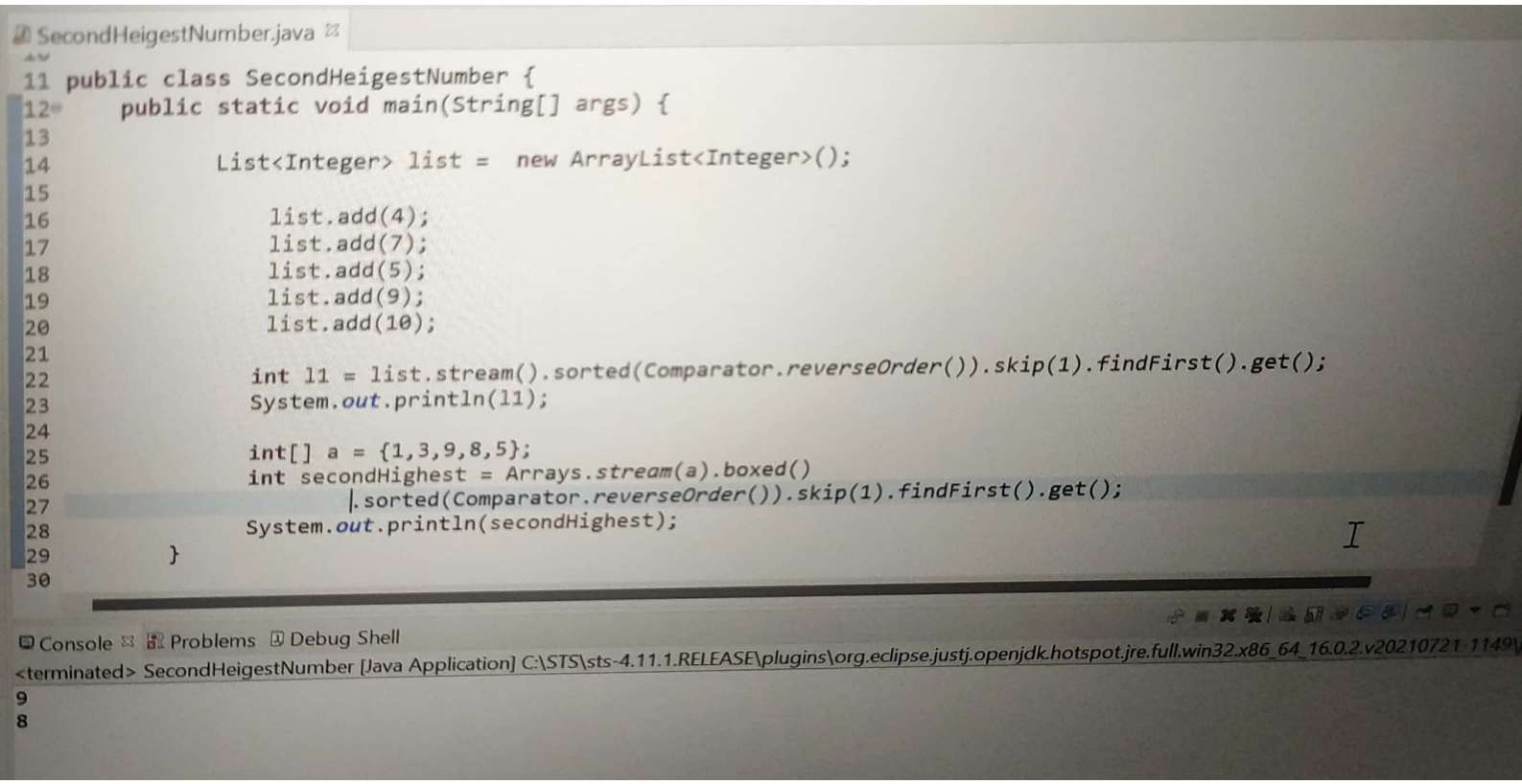
RemoveDoublicate.java

```java
package Eight;

public class RemoveDoublicate {

    public static void main(String[] args) {

        String str ="abbcccdeeek";

        char[] ch = str.toCharArray();

        String str1="";
        for(char c : ch) {

            if(str.indexOf(c) == str.lastIndexOf(c)) {
                str1=str1 + "" + c;
            }
        }
        System.out.println(str1);

    }
```

Console | Problems | Debug Shell

<terminated> RemoveDoublicate [Java Application] C:\STS\sts-4.11.1.RELEASE\plugins\org.eclipse.jus

adk



SingleAndParlleStream.java

```java
package Eight;

import java.util.stream.IntStream;

public class SingleAndParlleStream {

    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(" sequential stream time :: "+(end-start));

        start=System.currentTimeMillis();
        IntStream.range(1, 100).parallel().forEach(System.out::println);
        end=System.currentTimeMillis();

        System.out.println("Parllel stream time :: "+(end-start));
    }
```

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
Parllel stream time :: 12
```

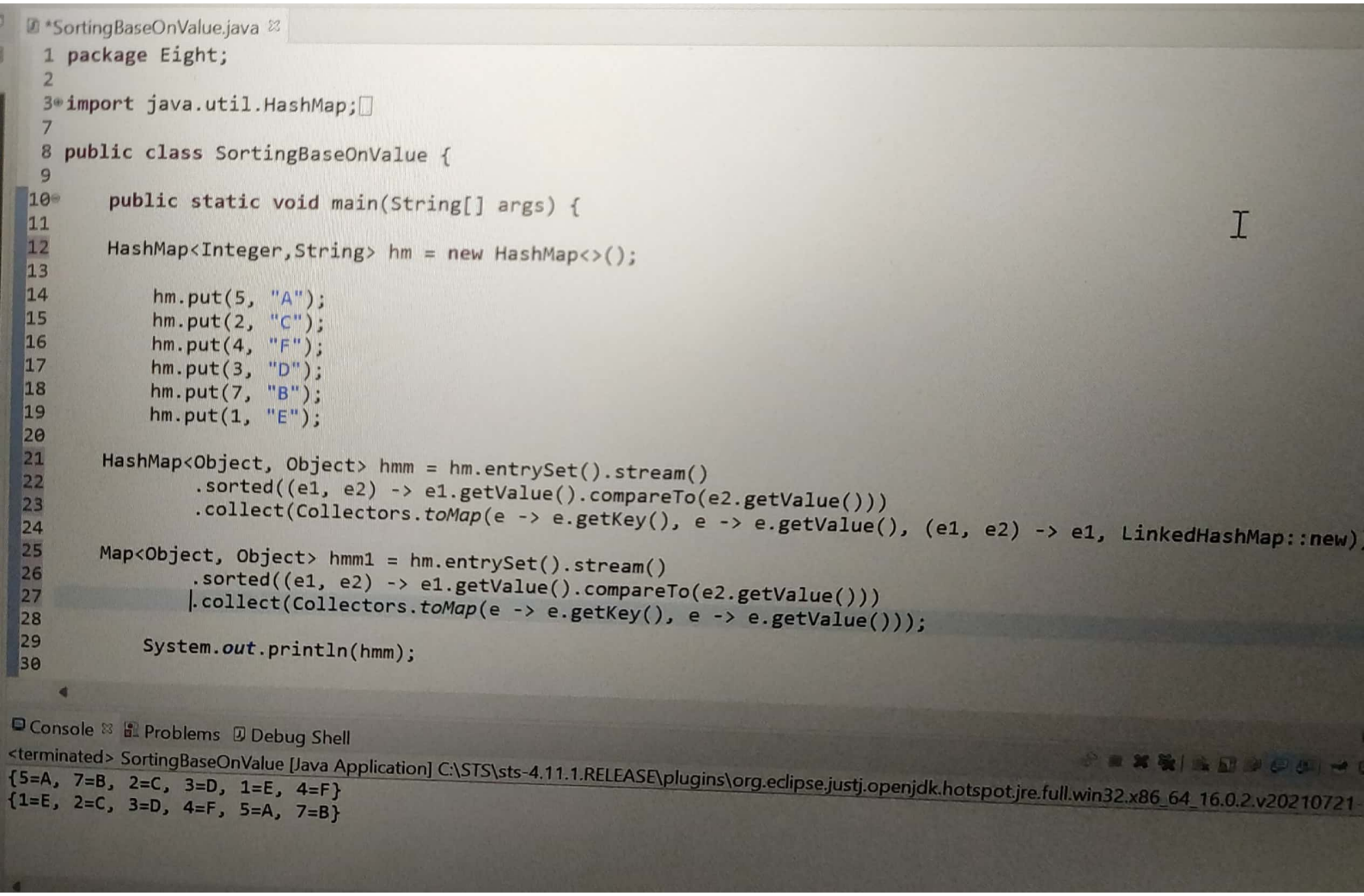
CharacterCount.java

```
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
```

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\jre

```
{1=1, A=3, 2=1, B=3, 3=1}
```



*SortingBaseOnValue.java

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