115. Important Note - Plan of learning this Section

Selenium coding with java streams:

Java streams are introduced from version 8 which comes with many helper methods to write simplified and optimized code.

In this section, I will explain How to use java streams concept for selenium Automation.

This section is divided in to 2 parts

1st part : Understand java streams from scratch with examples. – 1hr session

2nd part : Introduce java streams into selenium Web Automation

116. Learn everything about Java Streams

Java streams, Lambda Expressions for Building Optimized Code

What are Streams ? - Streams -  stream is nothing but a collection of strings

Stream API is new feature available from java 8

By using streams, we can perform various aggregate operations on the data returned from collections classes by drastically reduce the complexity of code.

What is Lambda Expressions ?

Lambda Expressions introduce the new arrow operator -> into java. It divides the lambda expressions in two parts :

The left side specifies the parameters required by the expression, which could also be empty if no parameters are required.

The right side is the lambda body which specifies the actions of the lambda expression.

The working of stream can be explained in three stages :

1. Create a stream
2. Perform intermediate operations on the initial stream to transform it into another stream and so on on further intermediate operations.
3. Perform terminal operation on the final stream to get the result.

An important characteristic of intermediate operations is laziness.

When executing this code snippet, nothing is printed to the console. This is because intermediate operations will only be executed when a terminal operation is present.

Note: The aggregate operations that we perform on the collection, array or any other data source do not change the data of the source, they simply return a new stream.

first of all, let me write one program to store the elements in the list.

And after storing the elements in the list, you have to find out what all elements which are stored in the list starts with Alphabet A.

Basically, I'm going to create one data structure with different names.

After you store, you should get the count of number of names present in the that particular list, starting with the letter A.

same program. I'll do it in streams and show you how we can optimize our existing code.

So our goal is to pull out the names.Start with Alphabet A.

// count the number of names starting with alphabet A in list

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

It's just creating an object for a class array list.

And here we are clearly telling that this list will only have strings.

Let's add the names into this arraylist.

names.add(“Abhijeet”);

names.add(“don”);

names.add(“Alekhya”);

names.add(“Adam”);

names.add(“Ram”);

int count =0;

So we have added five names into this list.

Now, if I want to get the count of number of names, starting with Alphabet A first I need to iterate through a for loop.

To go through each and every name of a list and then figure out for which item in that list we are having. The name starts with Alphabet A.

To iterate through each and every item in the array list. So I am starting from zero and still the size of that array. You see that names is an object right of this list.

Total five names are stored so it will iterate five times.

So each time, if I want to grab every element, unique element of this array list, I can do it with name start get of a. So for the first time I will be zero.

Note : So get method in ArrayList is basically used to pull the element which is present in the list.

So whatever I get for each and every loop iteration, I will carefully store it in one string.

Let's say it has actual something, anything variable name.

For(int i=0;i<names.size();i++)

{

String actual=names.get(i);

If(actual.startswith(“A”))

{

Count++;

}

}

System.out.println(count);

So this is how you will do when this coding test is given with the regular java style.

So same program, let's say.How can we optimize it?

Very minimal and do it in one or two lines with the help of streams.

Public void streamfilter()

{

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

names.add(“Abhijeet”);

names.add(“don”);

names.add(“Alekhya”);

names.add(“Adam”);

names.add(“Ram”);

Long c=names.stream().filter(s->s.startsWith(“A”)).count();

System.out.println(c);

s is the variable

So a whole list is in this.Names object, right?

Name start.Assign this array list to streams first.

Streams -  stream is nothing but a collection of strings.

So all you are ArrayList.We will be moving into stream so that stream can process it very quick and it do the operations parallelly. So in this stream API there is a method called filter.

whatever list you have and make it stream compatible by writing

dot stream so that it will be stream compatible and which can also support stream methods.

Now dot filter is something which will filter based upon your condition.

Now you have to tell by which condition you want to filter this ArrayList and give the result.

s->s.startsWith(“A’)).count(); - lambda expression 2types 1st page definition

Long is a datatype

C is a new created variable

You have done the whole logic in one single line instead of having for loop.If conditions creating variables

Stream.of(“Abhijeet”,”Don”,”Alekhya”,”Adham”,”Ram”).filter(s->s.startsWith(“A”)).count();

Now, instead of creating one collection and converting that into stream and then applying, you can directly create a stream with the stream API where we are cursor on this and you need to import this Java.util.stream.

this current operation will only come into picture when if filter operation return true.

If it does not return true count method will not get executed.

Note : 1. there is no life for intermediate operation. If there is no terminal op.

2. terminal op will execute only if internal op (filter) returns true

3. we can create stream

4. how to use filter in stream API

what this lambda expression tells the right side, specify the action.

If that action is more than one line, then you can actually open the braces and write a whole code inside it.

Long d = Stream.of(“Abhijeet”,”Don”,”Alekhya”,”Adham”,”Ram”).filter(s->

{

s.startsWith(“A”);

return true;

}).count();

System.out.println(d);

}

 now my requirement is to print.

Print all the names present in the list with the name length is greater than 4

1. first convert names into stream.

names.stream().filter(s->s.length()>4).forEach(s->system.out.println(s));

2 results will get abijeet and alekya but in want only one result to print.

names.stream().filter(s->s.length()>4).limit(1).forEach(s->System.***out***.println(s));

 I'm going to introduce one more method called map.

 I have a new requirement.Of printing. Print the names which has last letter as “a” with upper case

So if you want to manipulate your stream, then there is a method called map which can map your original stream to. The value or how you want to manipulate it.

Map method  will actually help us to modify our stream filter results.

@Test

Public void streamMAP()

{

// Print the names which has last letter as “a” with upper case

Stream.of(“Abhijeet”,”Don”,”Alekhya”,”Adham”,”Rama”).filter(s->s.endsWith(“a”))map(s->s.toUpperCase()).forEach(s->System.out.println(s));

}

//Cosole output : ALEKHYA, RAMA

So we have discussed two terminal operations, I guess limit and count.

Similarly, there is one more terminal operation sort.

You can either do this stream of or create an ArrayList and send that name start stream.

Or if you have arrays, you can also convert that into ArrayList.

Do you know that how to convert arrays to ArrayList?

You have a simple method called arrays dot as list, so you can actually pass the your array into this list argument and that will be convert to the list and you can use streams for it.

// print names which have first letter as “a” and upper case has sorted

List<String> names = Arrays.*asList*("Azbhijeet","Don","Alekhya","Adam","Rama");

names.stream().filter(s->s.startsWith("A")).sorted().map(s->s.toUpperCase()).forEach(s->System.***out***.println(s));

let us see how can we merge to array lists?

Let's say you have list A and list B and if I ask you to merge into one whole list by merging all the elements from list A and list B and then sorted, how do you do that?

So first of all, you have to convert names list and names1 list into streams, which is obvious.Thereafter you have a method called streams.concat

So by converting these two lists into streams, we we can utilize the catch stream method which will help us to concatenate two streams, which are nothing but two lists now.

So this will create one new stream by combining stream one and stream two.

Stream<String> newStream =Stream.concat(names.stream(),names1.stream());

newStream.sorted().forEach(s->System.out.println(s));

Now, so when you combine this two lists, you got one big list. Right now we want to check whether Adam is present in that list or not. If it is present, your script should return.True. If not, it should return false. How do you do that?

You may think that filter may help you, but if you observe carefully, filter itself will not work. If you can say that as s. Equals ignore scares.

Find the adam. And if you print, Adam will print in the output.

But you have to tell our written boolean expression that if it doesn't just return me true, if not return me false. I don't want to return any name to be printed in the output.

So if your requirement is to check whether any text or whether anything is present inside our list,then you have to use match argument.So that match argument will return true based upon your condition. If condition is matches, it will return true so that you can directly print that.

Boolean flag=newStream.anyMatch(s->s.equalsIgnoreCase(“Adam”));

System.out.println(flag);

Assert.assertTrue(flag);

so there is one more method called collect.

So this collect method is basically used to collect your results and convert it back into any list. Till now we have taken the list and then we converted that into streams and then we applied, modify it, did many modifications, and then we printed it in the output.

So let's convert back that into new list.

you work in real time frameworks, you have to do manipulation and preparing a new list and send it. You don't want to see anything in output

When I say collect method, we'll collect all the results, whatever it returned from any method.

Public void streamcollect()

{

List<String> ls=Stream.of("Abhijeet","Don","Alekya","Adam","Rama").filter(s->s.endsWith("a")).map(s->s.toUpperCase()).collect(Collectors.toList());

System.out.println(ls.get(0));

Distinct method method simply will get the result of only distinct elements in this list, and it will avoid all the duplicates. It will only print the unique, distinct numbers.

List<Integer>Values=Arrays.*asList*(3,2,2,7,5,1,9,7);

//print unique number from this array

//sort the array – 3rd index – 1,2,3,5,7,9

Values.stream().distinct().forEach(s->System.***out***.println(s));

List<Integer> li=Values.stream().distinct().sorted().collect(Collectors.*toList*());

System.***out***.println(li.get(2));