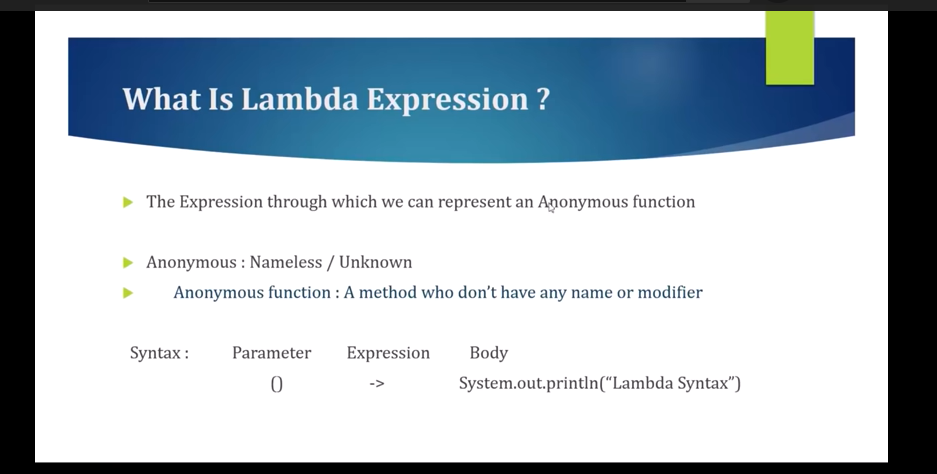
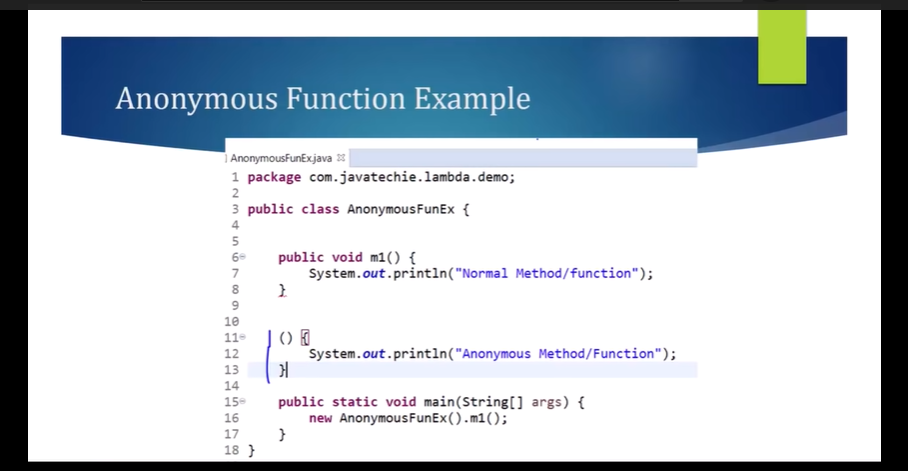
**=========<< Functional Interface & Lambda Expression >>==========**



Using Lambda Expression we can convert our Abstract Method to Anonymous Function.

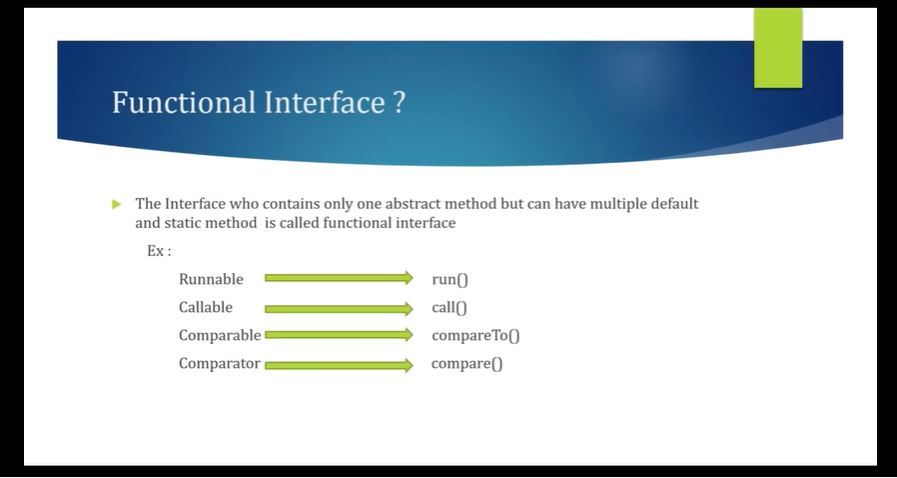
So, here m1() is our normal function and marked one is an Anonymous Function. So, we do have a question like if this method doesn’t have a method name or prefix then how we will call this method using object.

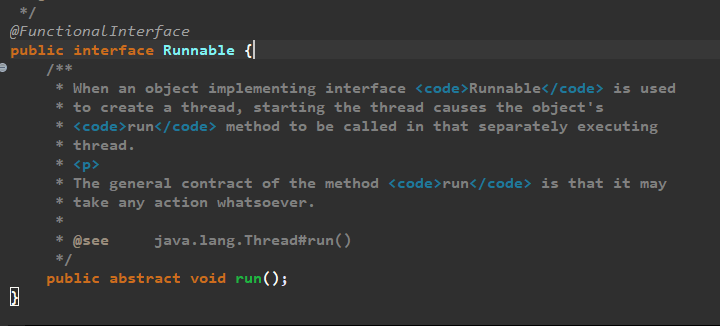
**Note**: for Every method we can’t write the Lambda Expression, Lambda Expression only can be applicable for Functional Interface. So, the Method which is present into a Functional Interface for that only we can write the Lambda Expression.



**What is Functional Interface**????

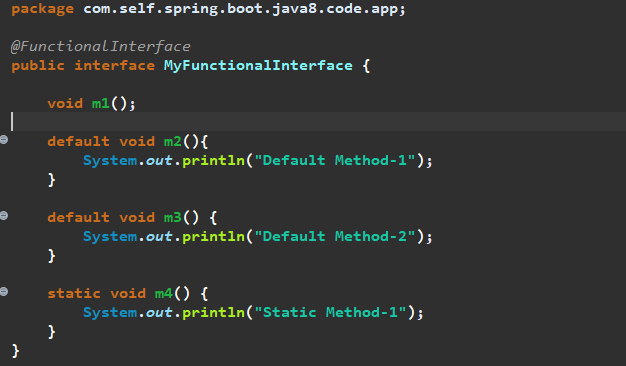
As we all know like In Java 8 Inside an Interface we can write a Default Method as well as the static method, But If the Interface contains only one Abstract Method then only we can say that Interface is a Functional Interface. So, as you observed that Runnable, Callable, Comparable and Comparator are Functional Interface because these all Interface contains only one abstract Method.

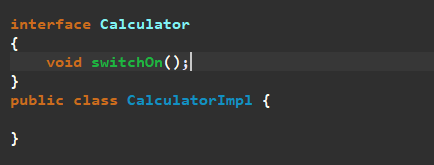




Like this we can write our own Functional Interface. So, it doesn’t matter for the static and default method If the Interface contains only one Abstract method then we can say that Interface is a Functional Interface.

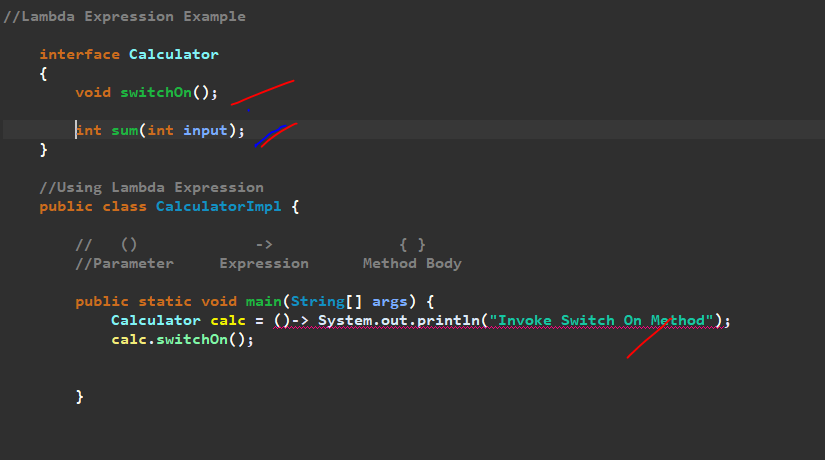
Note: For Functional Interface the Method which is Abstract for that one only we can write Lambda Expression. So, we can convert this m1() method to Anonymous Function using Lambda Expression.





We are going to write the lambda expression for the switchOn() method. So. Lambda Expression is nothing we just need to convert this Traditional method into Anonymous Method Using Lambda Expression. So let’s write a lambda expression for this switchOn Method.

If we will create 2 Abstract Method then it won’t be a Functional Interface and lambda Expression will throw an Error….



**//Lambda Expression Example**

**interface Calculator**

**{**

**//void switchOn();**

**//void sum(int input);**

**int substract(int i1,int i2);**

**}**

**//Using Lambda Expression**

**public class CalculatorImpl {**

**// () -> { }**

**//Parameter Expression Method Body**

**public static void main(String[] args) {**

**// Calculator calc = ()-> System.out.println("Invoke Switch On Method");**

**// calc.switchOn();**

**// Calculator calculator =(input)->System.out.println("Sum = "+input);**

**// calculator.sum(135);**

**Calculator calculator=(i1,i2)->{**

**if(i2>i1)**

**return (i2-i1);**

**else**

**throw new RuntimeException(i2+" is Less than "+i1);**

**};**

**System.*out*.println(calculator.substract(12, 9));**

**}**

**}**

**//In Our Traditional Approach**

**// public class CalculatorImpl implements Calculator**

**// {**

**// @Override**

**// public void switchOn() {**

**// System.out.println("Invoke Switch On Method");**

**// }**

**//**

**// public static void main(String[] args) {**

**// new CalculatorImpl().switchOn();**

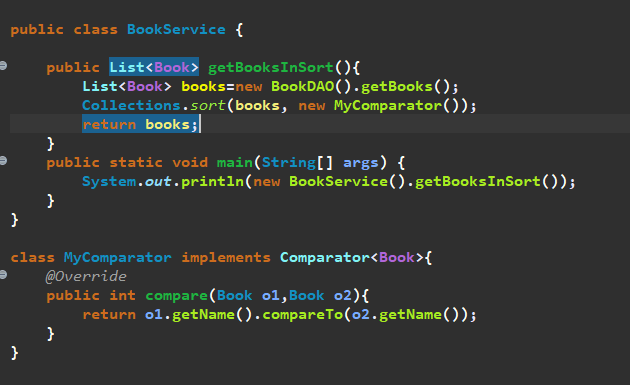
**// }**

**//}**

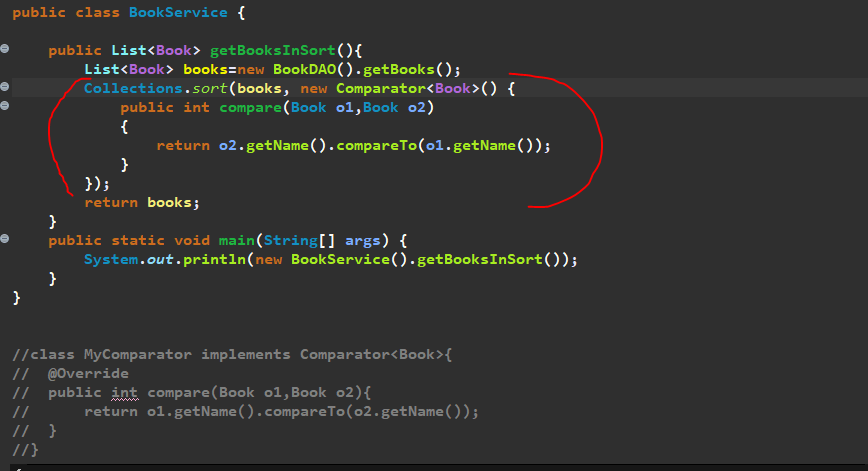
**Real Time Example for Functional Interface & Lambda Expression**

We will take a Book Example and then we will sort Using Comparator using Lambda Expression Java 8…

**Traditional way of Sorting…..**

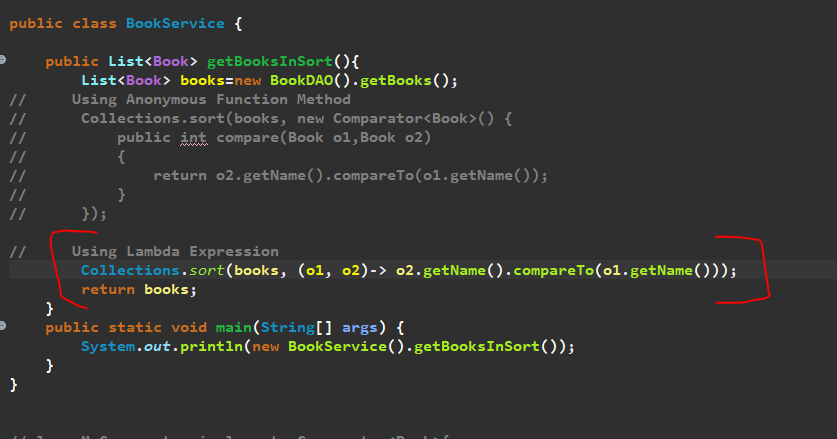


We can write this in another way using Anonymous Function for Functional Interface….Since Comparator is a Functional Interface so we can write Lambda Expression also for the method compare…



Using Lambda Expression…So we need to write the Lambda Expression for the Anonymous Function method compare()…so we don’t want any method name and Prefix…and no need to pass the data type of arguments so just remove…we just need the Expression and the body…since we are passing only one line so no need to give the bracket as well…so we can pass below instead of comparator…

(o1, o2)-> o2.getName().compareTo(o1.getName())

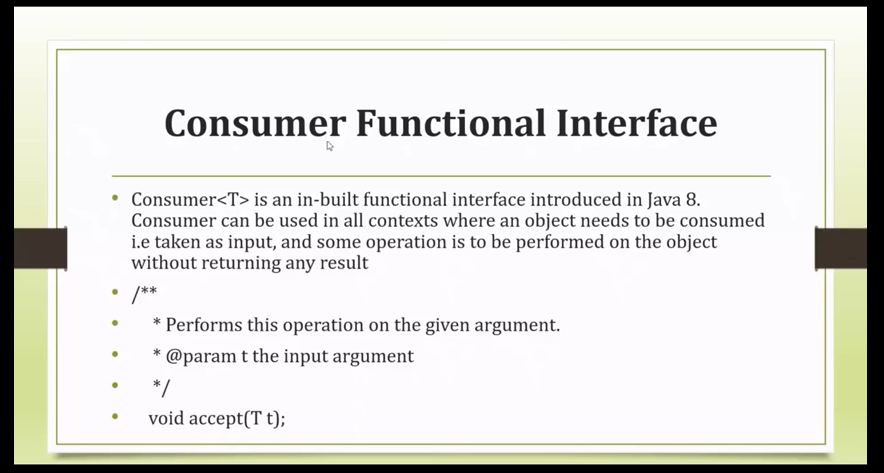


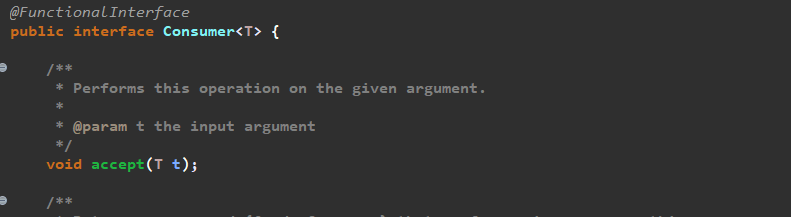
Since Comparator is our Functional Interface that why we are able to represent the compare() method using lambda expression.

For other Interface we can do like that only for functional Interface we can write lambda expression…

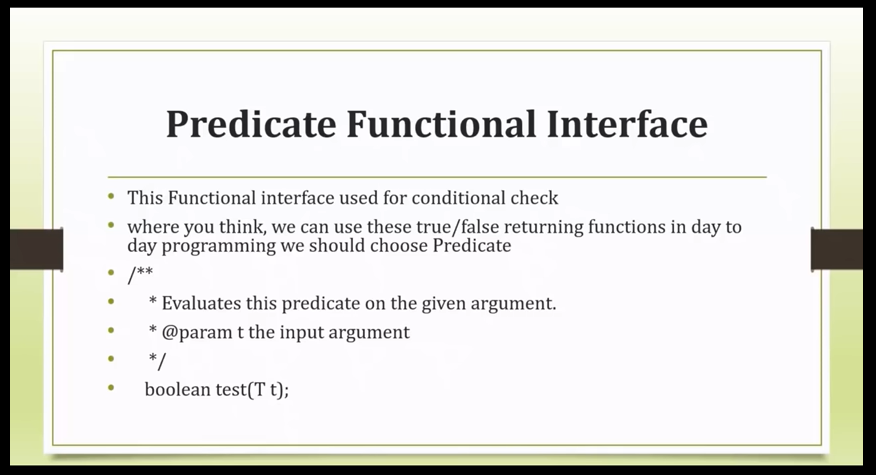
**==================<< END of Lambda Expression >>=====================**

**====== Predefined Functional Interfaces : Consumer, Predicate & Supplier =======**

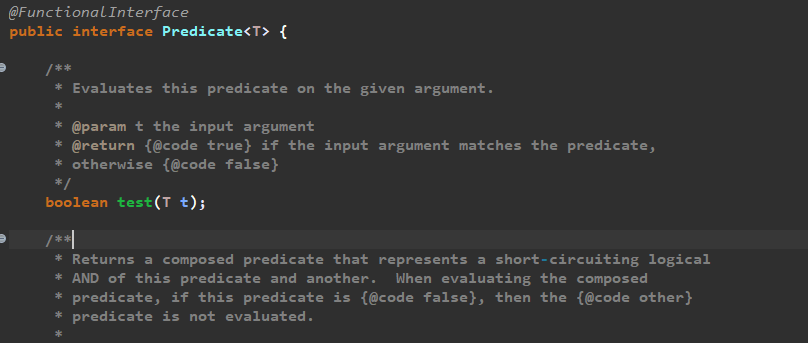




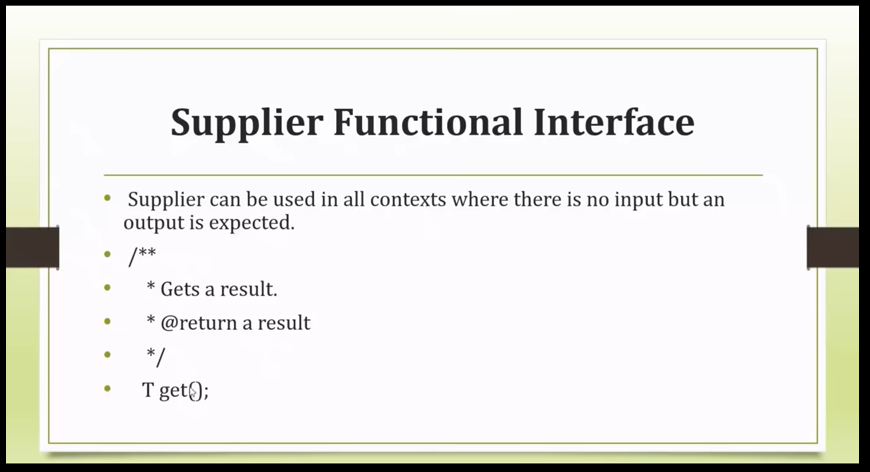
Note: Consumer is a Functional Interface which contains one abstract method. Abstract method which is taking input as generic object it can be any type of object and it is not returning anything, so the return type is void that means pass the input and perform some operations based on input or simply we can say either we can print, or we can do manipulation logic based on that given input.



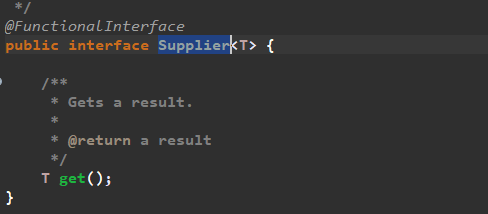
So Based on the Input Argument it will always do the conditional check or perform some conditional operation and return output as a true or false as a Boolean value.



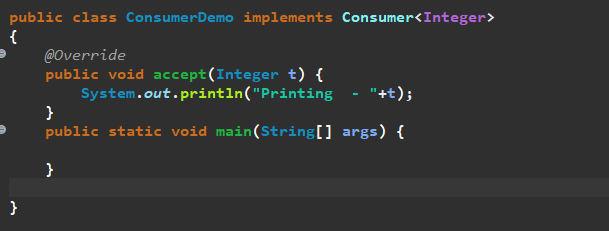
So, this Functional Interface contain one abstract method test, which take object as an input and after performing the operation it will return Boolean(true/false) as an output.



Supplier Functional Interface contain one abstract method that is get() it doesn’t have any input parameter, but it always returns as a Generic Type object. SO always It expect a return type.



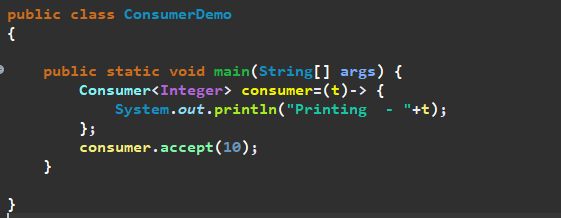
For example, just consume we are using list of 10 object, and we are performing some operation on it after that we are not returning anything so If I want to return some dummy result we can use Supplier, In that case. As an example, we are using list of 10 employee record object we are performing some percentile operations on it then we want to return some dummy result then we can go for a Supplier.



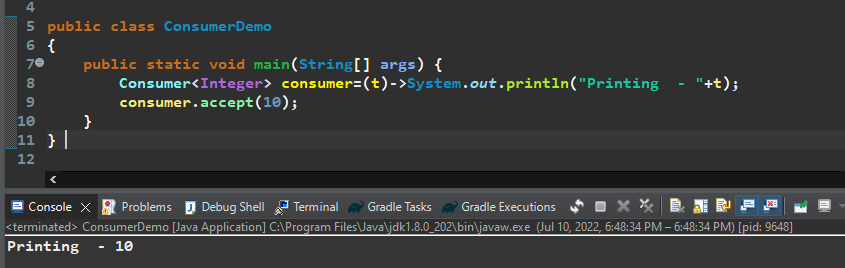
This accept is an abstract method which present into a Consumer Functional Interface so we can write a Lambda expression for it.so we can write a lambda expression for this accept method.

So, lets remove the implements Statement. Let’s copy paste accept method into a main method. So, we want to make it Anonymous Function, so we don’t want any method name and prefix lets remove it. And no need to specify data type also so just remove it. So now we have the parameter we have the method body only we have to add the lambda expression. And then we need to assign it to the functional Interface i.e here Consumer. So, for the method of Functional Interface only we can write the lambda expression.

Consumer Interface contain the accept method so directly we can call using Interface reference.

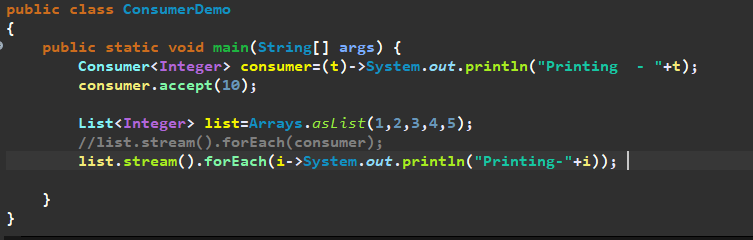


Since, only one argument so no need to give the bracket as well…



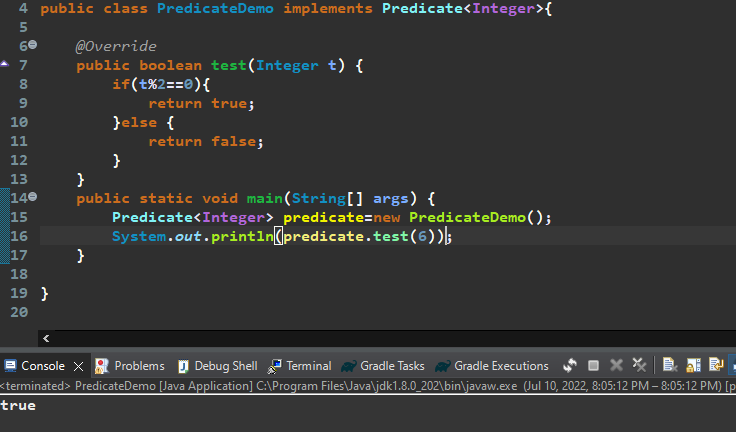
So this accept method only take one input parameter and perform operations on it , it will not return any output.

So, In Java8 Stream API there is a method called forEach() method which always accept the Consumer Functional Interface. And consumer only can perform operation does not return anything, so we are just printing the value. So, instead of passing the consumer reference we can directly pass the lambda expression of the Consumer Functional Interface.



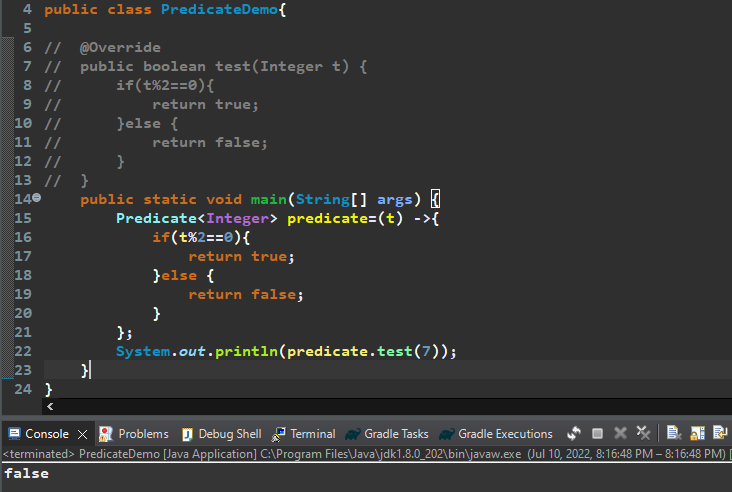
So forEach internally uses the Consumer Functional Interfaces, Consumer Internally uses the accept method…

**Predicate Example:-**

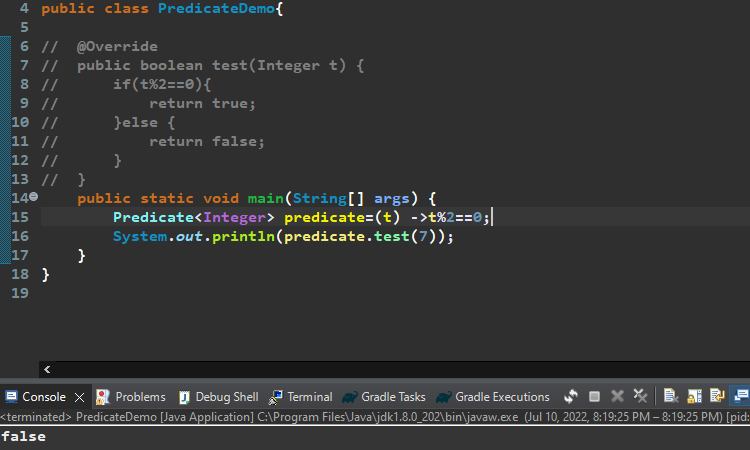


Now let’s make this traditional way into lambda expression…

So lets remove the implements Predicate Functional Interface. Let’s copy test method inside a main method then let’s remove the method name, data type and put Interface reference.

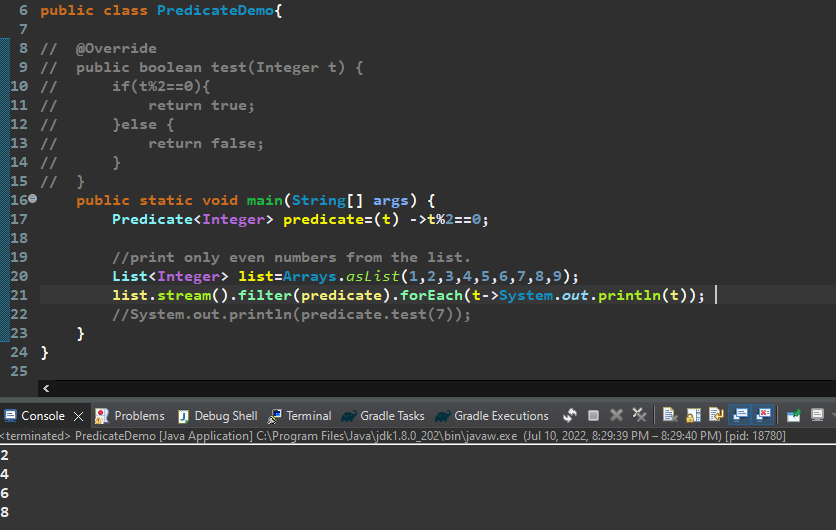


So, Predicate always returns either true or false based on our conditional check…



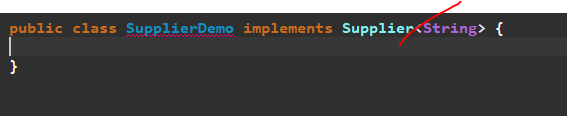
SO, for an Example Stream API filter() method always use a Predicate Functional Interface….

Print only the Even Numbers from the list of Integers…..

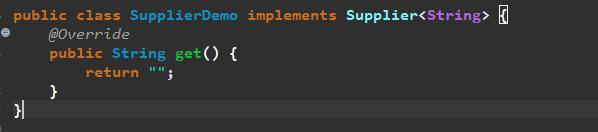


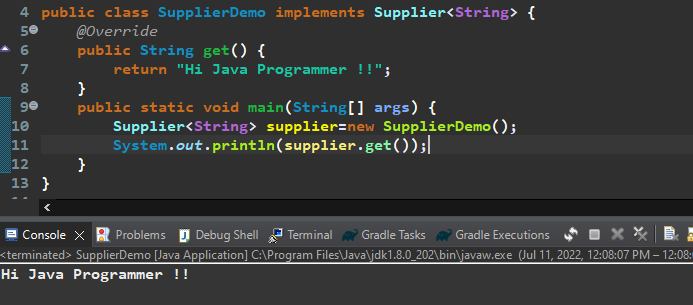
Let’s understand the **Supplier Functional Interface**…

So, Supplier which method doesn’t have any argument but it always expect some output.

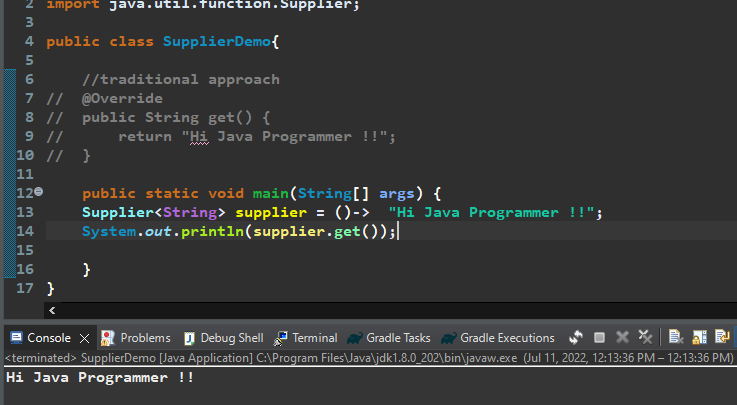


Means it will always return me the String value…

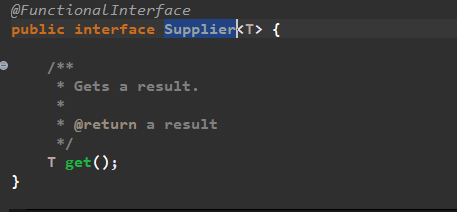




So, now lets convert this traditional approach to lambda expression…

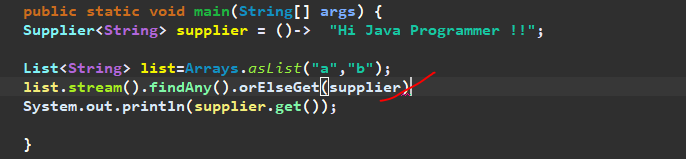


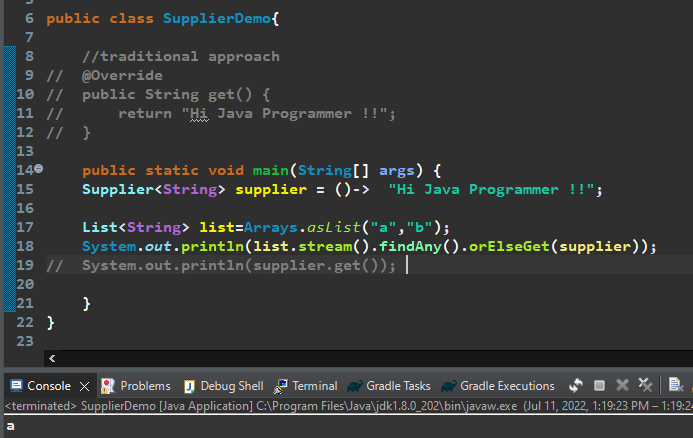
Supplier Functional Interface contain one method get() which always return a generic type.



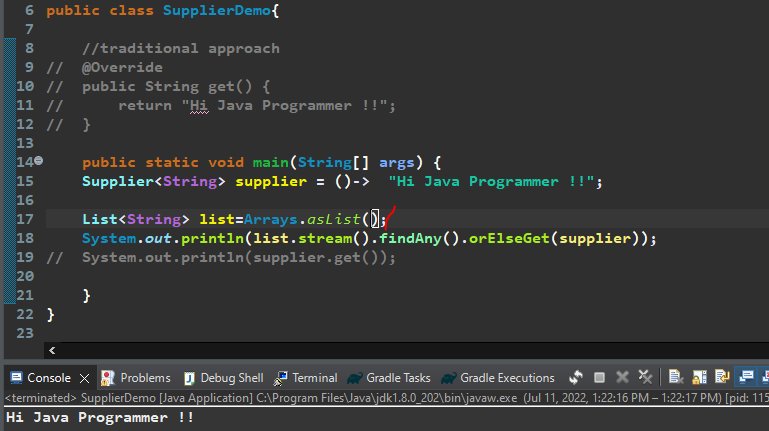
So, Supplier has only the return type no method argument….

In Real time Scenario In java 8 if we are doing filtering process and we are not getting any result, but you want some dummy data or some meaningful response in such case you can go for Supplier. orElseGet (Supplier) method get input as a supplier…..



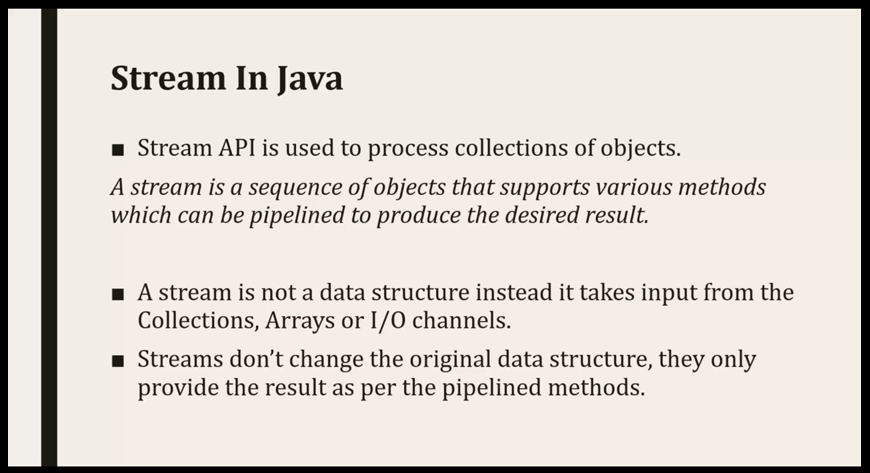


Since we do have a value in list i.e., a and b so it’s printing any one value…. but if list doesn’t contain any values… then supplier provide the dummy value. As list is empty…

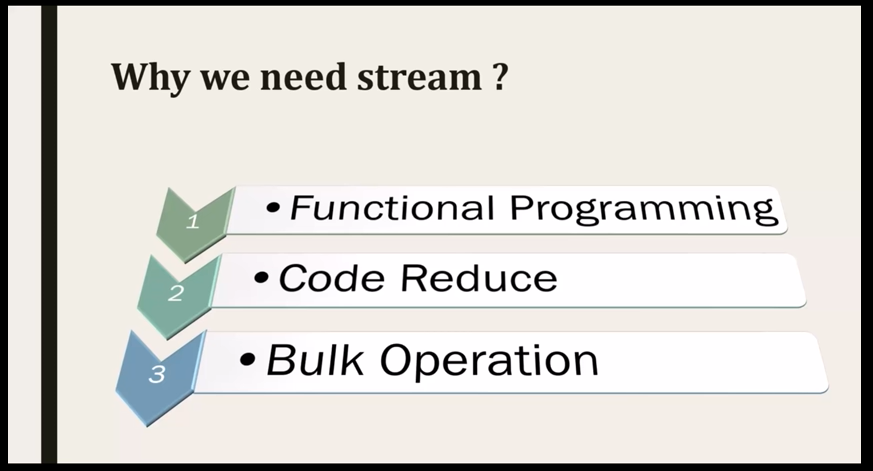


**===================End of Consumer, Predicate & Supplier==================**

**===========Java 8 Stream API I forEach &filter Method Example==========**



So, Stream won’t change any data structure it just performs operations based on pipeline provided method and gives a desired Result.

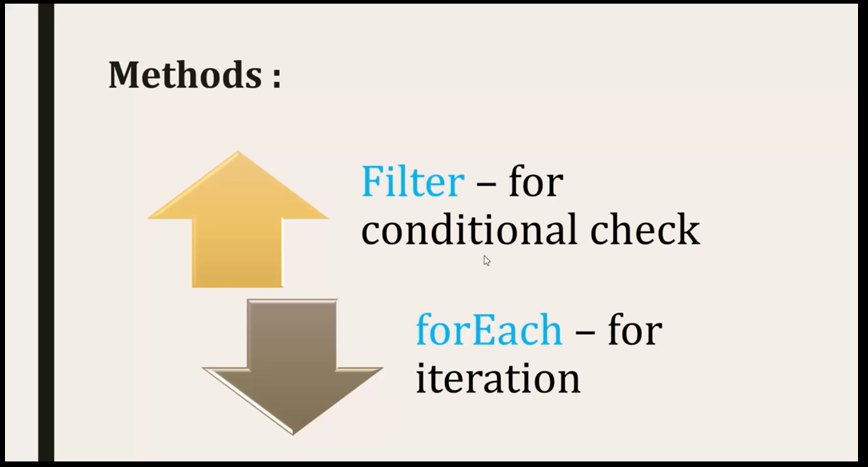


1. first advantage is we can achieve Functional Programming, means If we have a Functional Interface then I can represent that with lambda expression.

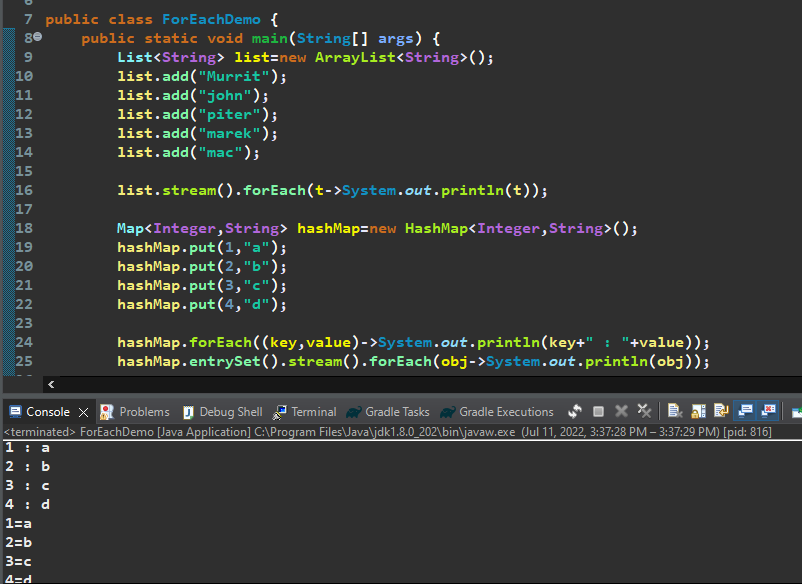
2. Second one is Code Reduce If you are using the lambda expression then length of the code will be minimized. And already we are aware for pipeline methods so if u do method chaining then the length of the code will reduce.

3. Third one is bulk operation, If I have a collection with huge data and I want to perform some operation then I should go for Stream API.

Because as per our Traditional Approach Stream will gives us a Better Performance



Let’s see Internal Structure for Filter and ForEach Method…..



Note:-

hashMap.forEach( Bi-Consumer) -> It takes input as Bi-Consumer means 2 input one output

hashMap.entrySet().stream().forEach(Consumer)

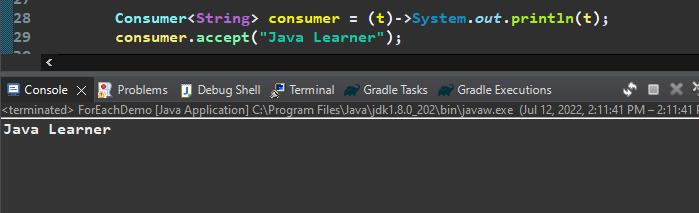
list.stream().forEach(Consumer)

list.forEach(Consumer)

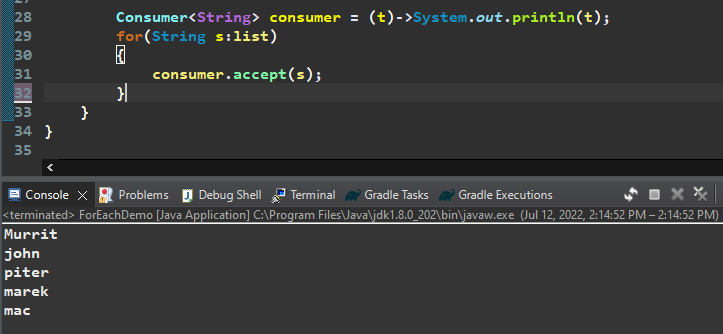
**\*\* How forEach() method works Internally?**

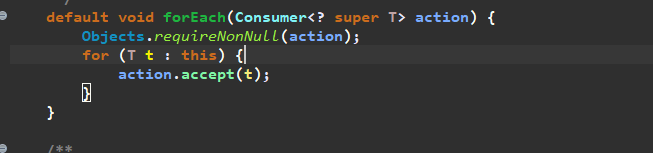
As we know forEach() consume argument as a Consumer Functional Interface. And the Consumer Interface have the method called the accept() method which consume one parameter and doesn’t return any value. I am just taking the input and printing it…that’s what foreach() is doing…

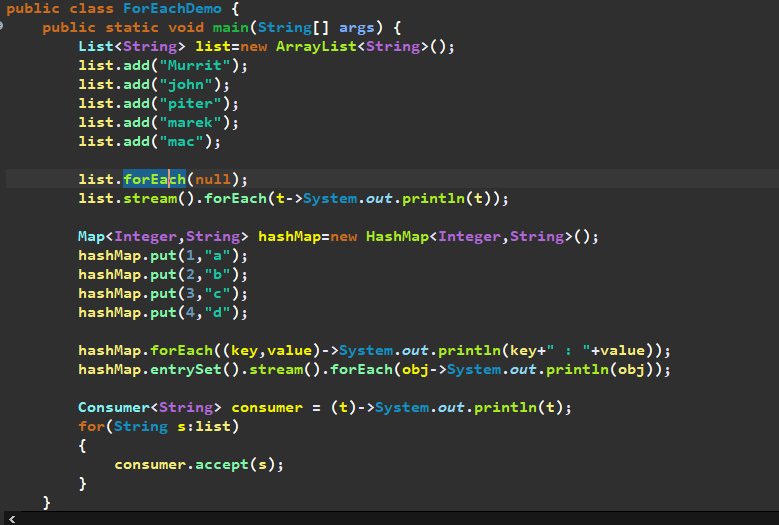




So what ever we are passing to accept method its simply print the result, same thing forEach() method also do it just iterate the list and call consumer.accept() method to print all the results…





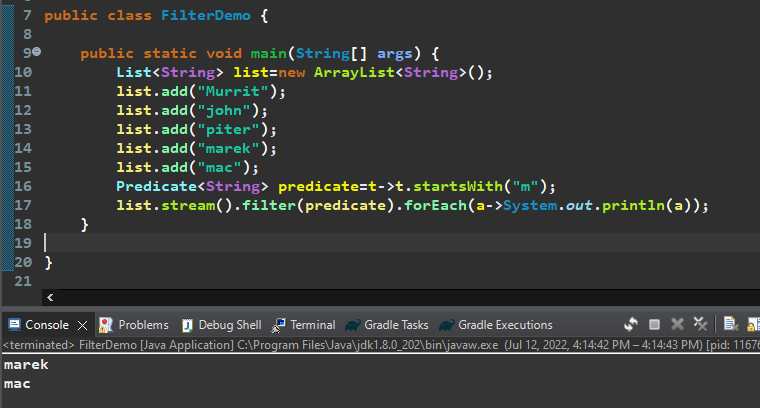


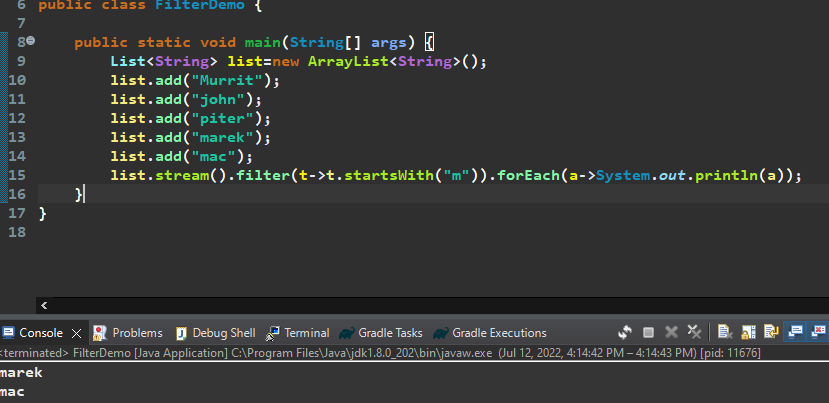
**Let’s discuss for filter () method: -**

Filter 🡪 Conditional Check

Suppose I have a list of Objects and I want to Iterate it based on some condition…

We need to find the string which starts with M…

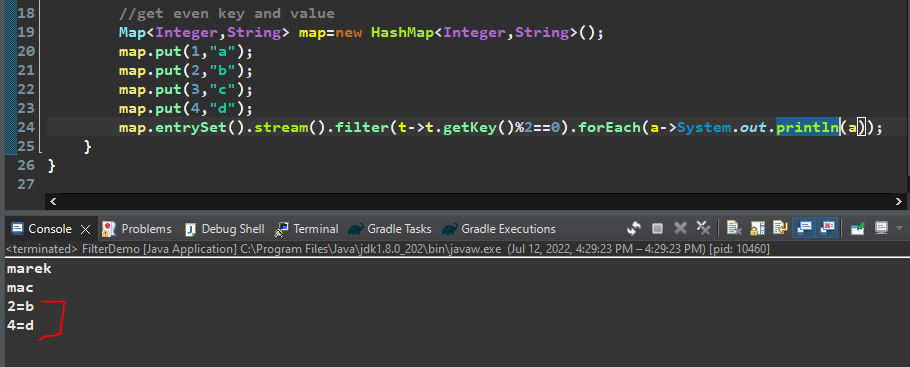




So, Filter internally using the Predicate Functional Interface, Predicate Internally having one abstract method i.e test () method, so we are writing lambda expression for that test method…

Let’s filter the map…

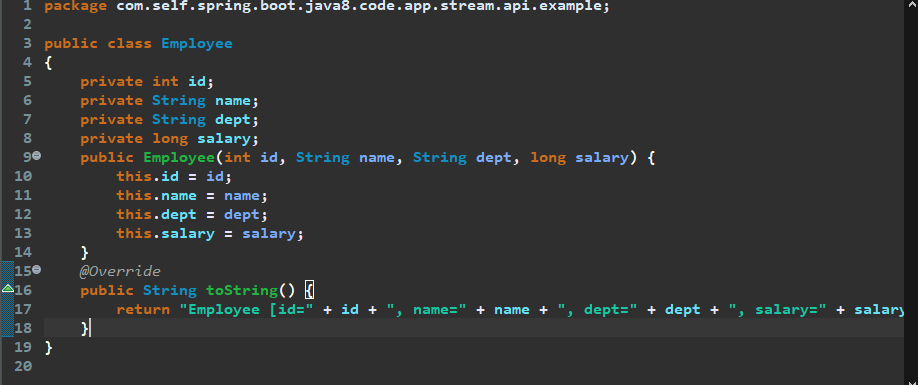
Get the Even key from the maps….

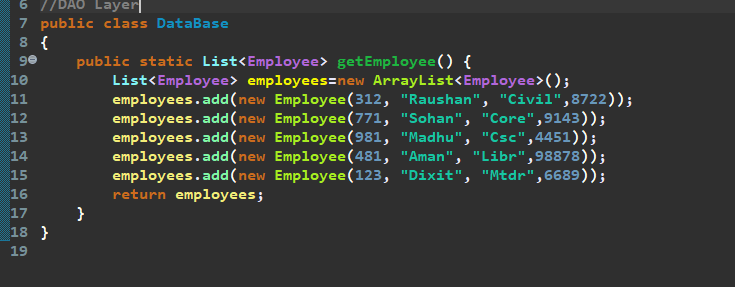


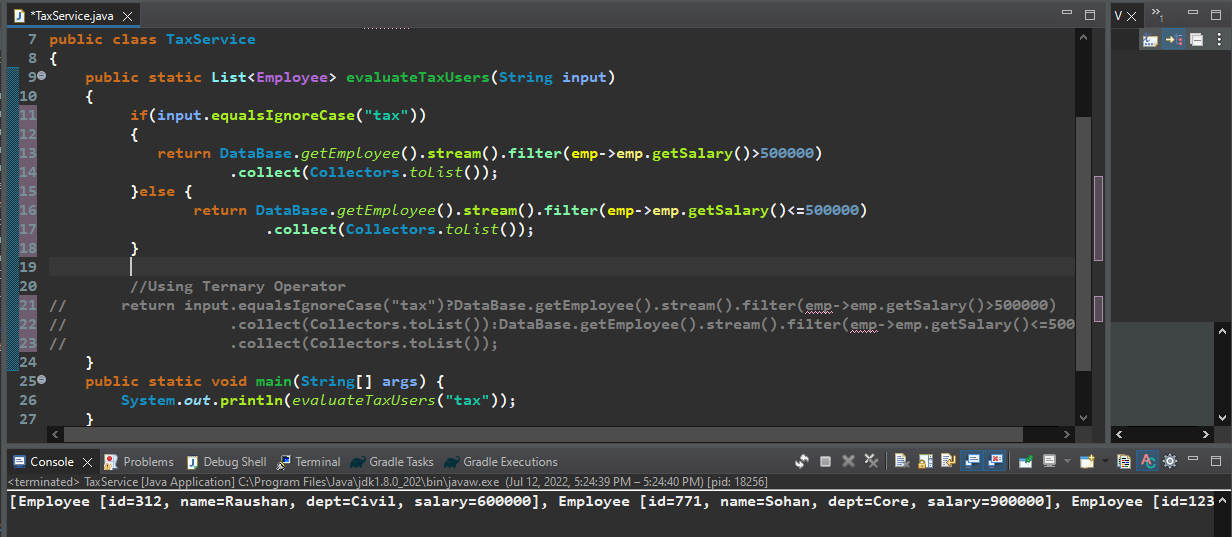
Realtime Example for Stream & filter-----

// I want to write a logic for An Employee who is eligible to Pay the Tax and who is not Eligible to Pay the Tax.

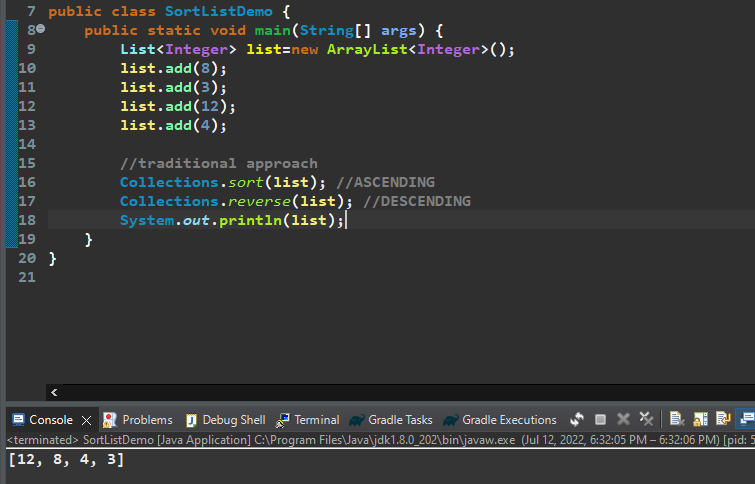
An Employee who is having Salary more than 5 Lakhs is Eligible to Pay the Tax.

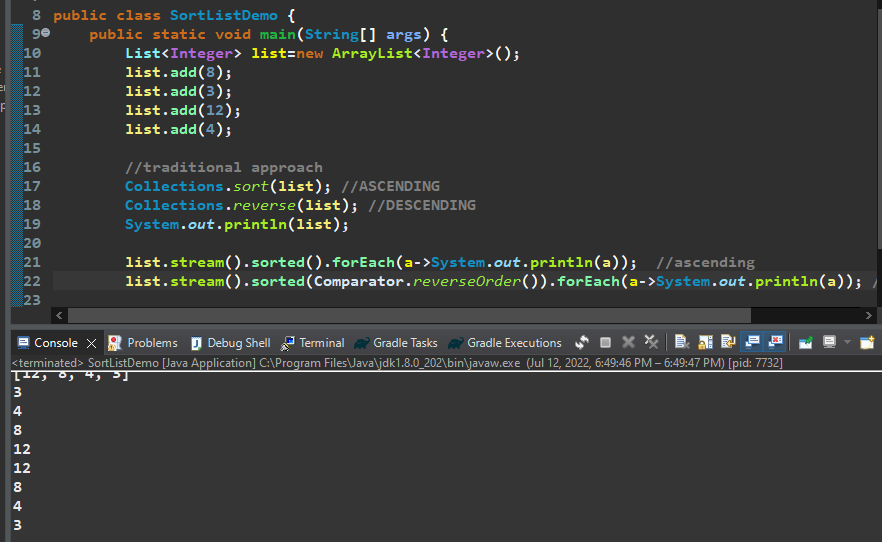




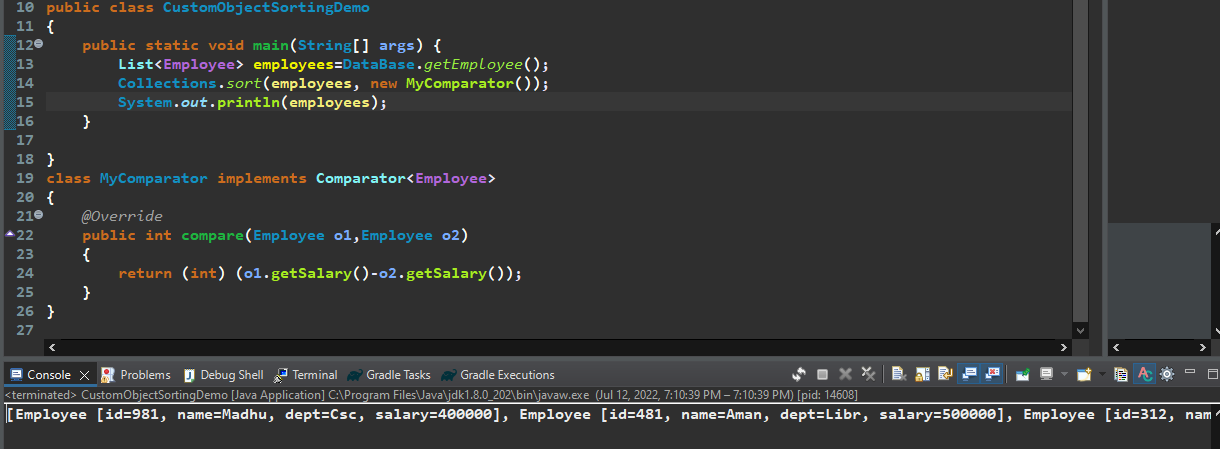






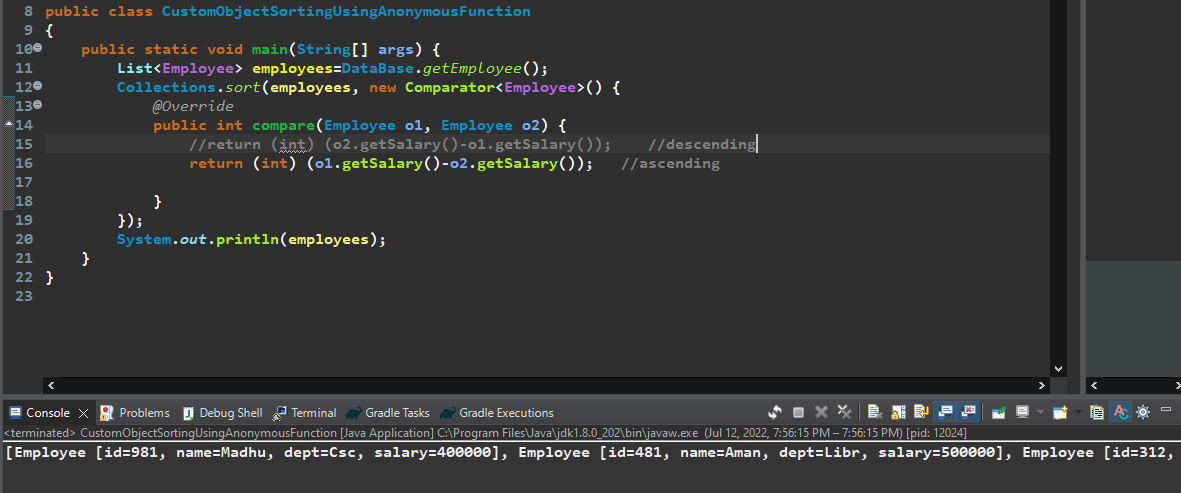


Suppose If we have a Custom Object and we want to sort based on few of the Fields…



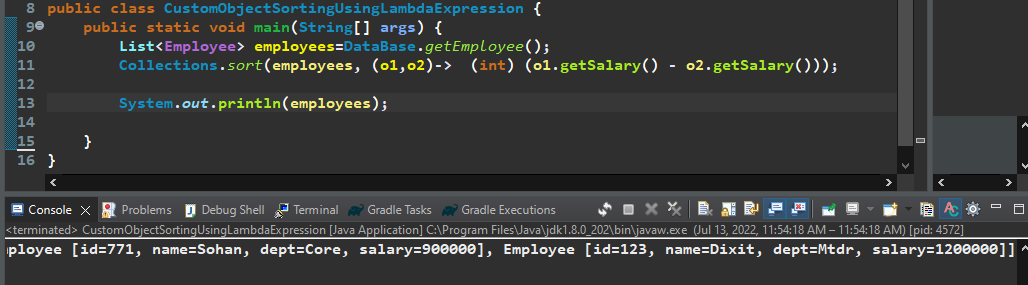
So instead of writing the traditional way …AS My Comparator is an Anonymous Function and Comparator is our Functional Interface so we can convert compare () method using lambda expression…

Now we are going to represent the Anonymous Function of that Interface….

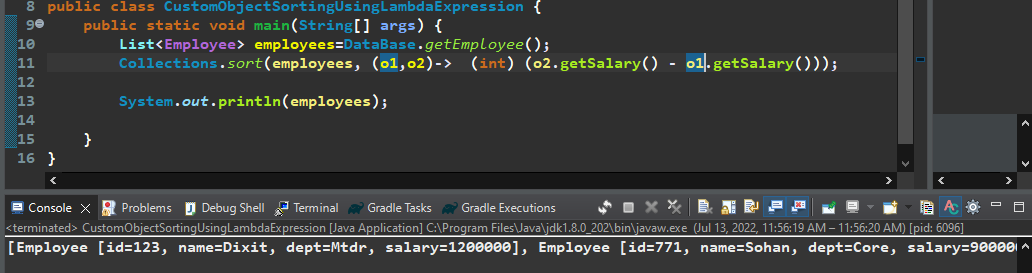


Now lets convert this Anonymous Function into lambda expression

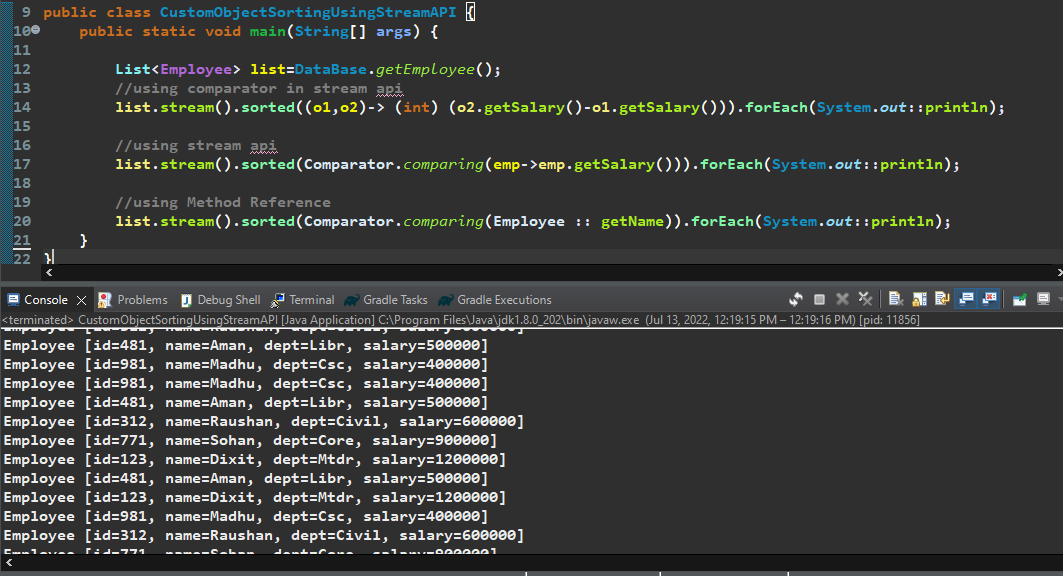
So, this comparator is having only one compare () method with 2 arguments so we don’t need any method name data type we just need parameter, so instead of anonymous function we can write a lambda expression….

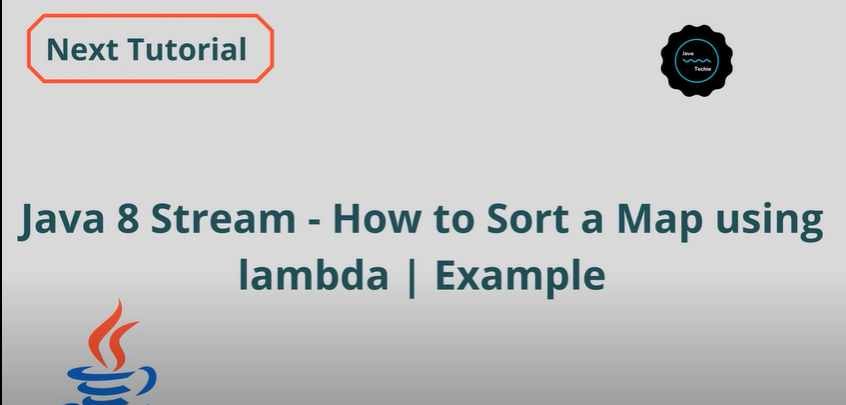


Just changing the object reference we are getting into descending order….

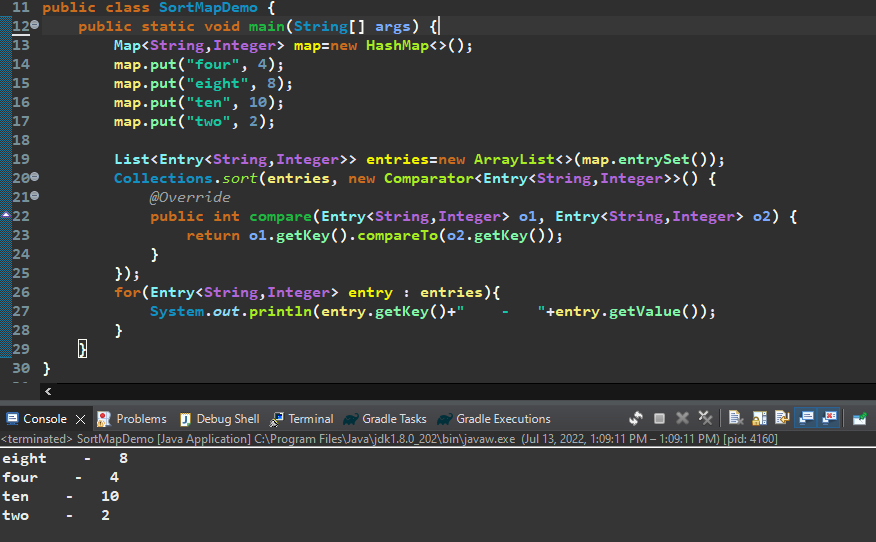


Now let’s see how we can sort using stream API





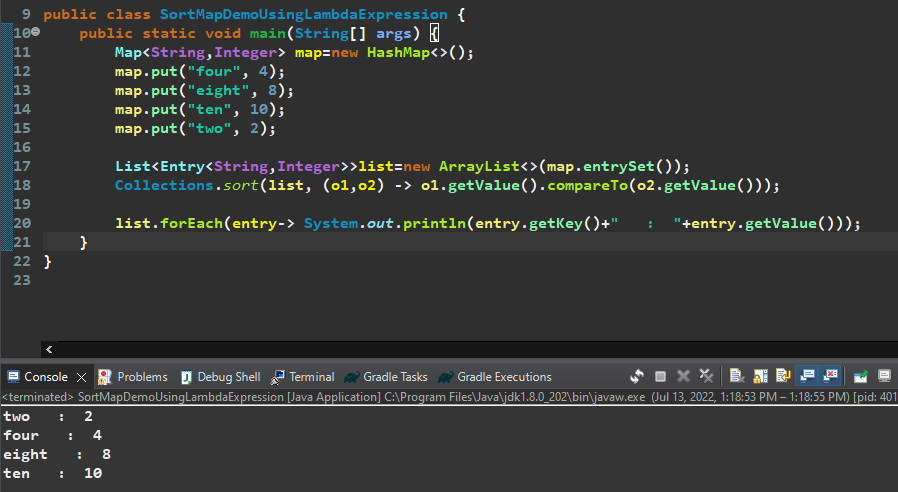
Using Traditional Approach –



Using Lambda Expression lets transform Comparator Functional Interface….

So, no need to give method name, data type ….

Let’s sort based on map values….



Let’s Sort Map Using Stream API----