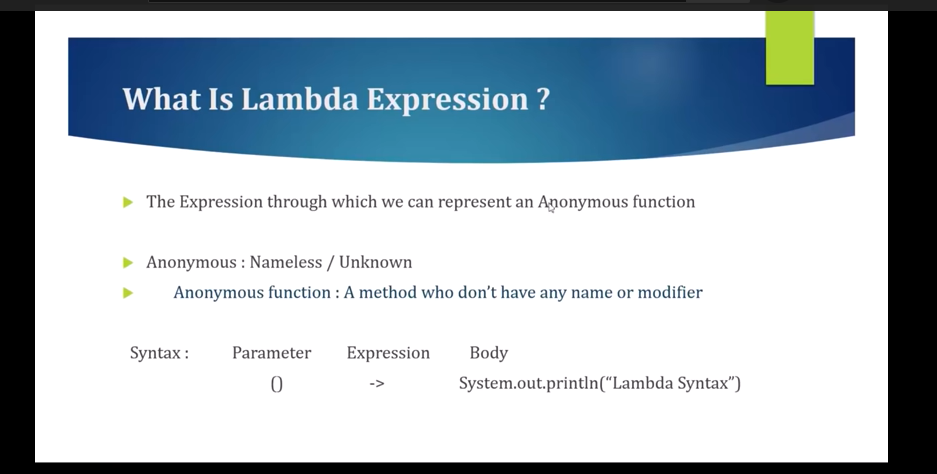
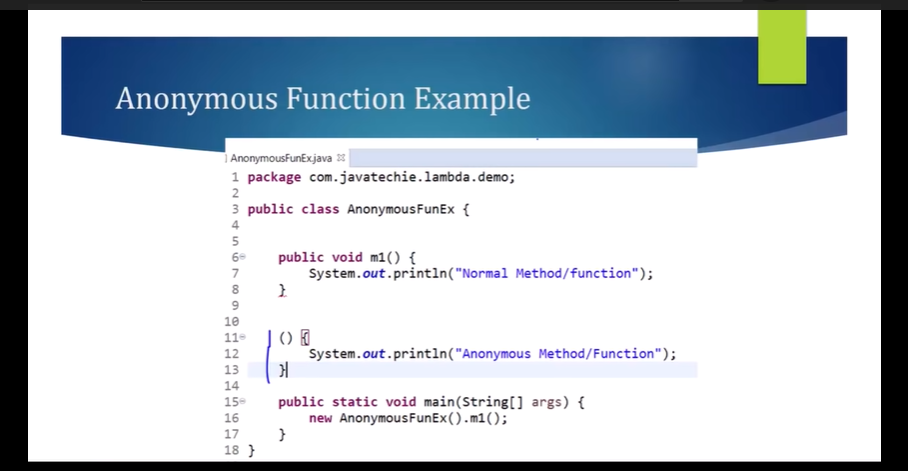
**==================<< Lambda Expression >>=====================**



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

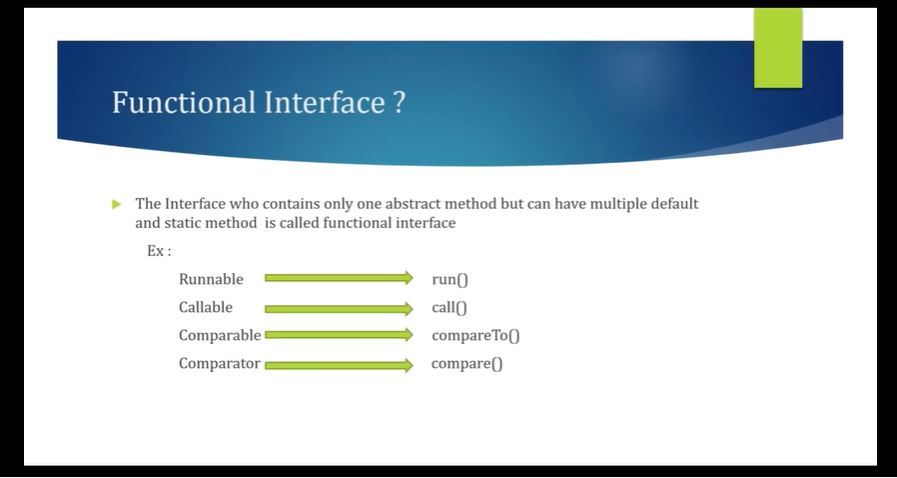
So, here m1() is our normal function and marked one is an Anonymous Function. So, we do have a question like if this method doesn’t have a method name or prefix then how we will call this method using object.

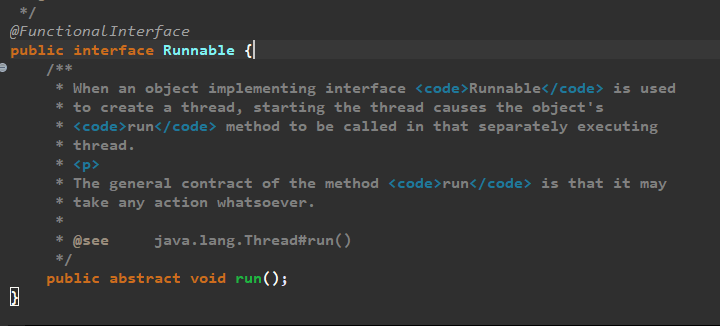
**Note**: for Every method we can’t write the Lambda Expression, Lambda Expression only can be applicable for Functional Interface. So, the Method which is present into a Functional Interface for that only we can write the Lambda Expression.



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

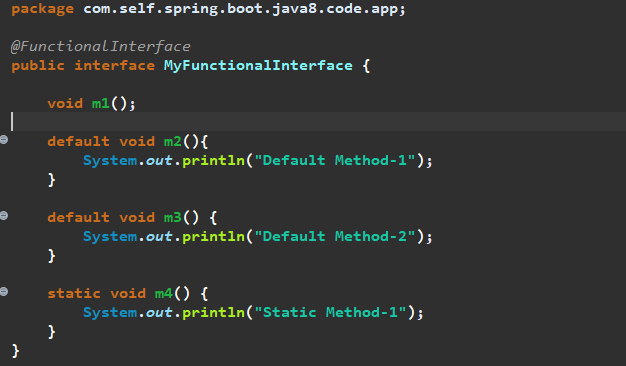
As we all know like In Java 8 Inside an Interface we can write a Default Method as well as the static method, But If the Interface contains only one Abstract Method then only we can say that Interface is a Functional Interface. So, as you observed that Runnable, Callable, Comparable and Comparator are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

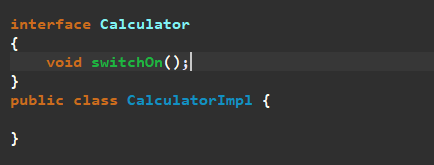




Like this we can write our own Functional Interface. So, it doesn’t matter for the static and default method If the Interface contains only one Abstract method then we can say that Interface is a Functional Interface.

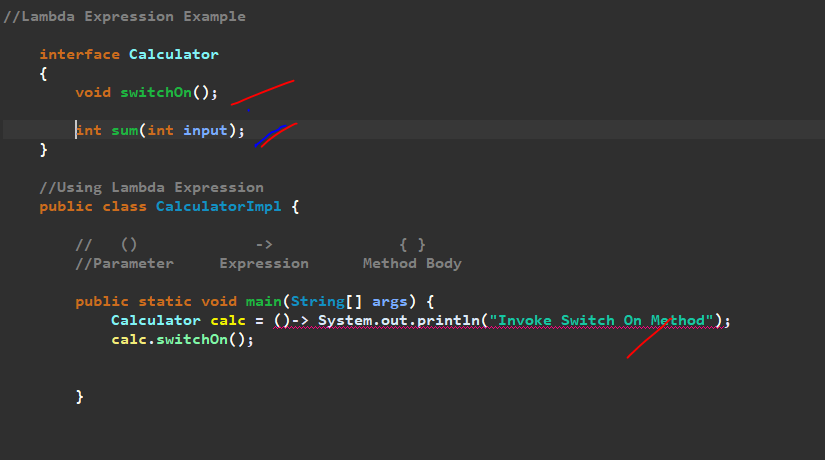
Note: For Functional Interface the Method which is Abstract for that one only we can write Lambda Expression. So, we can convert this m1() method to Anonymous Function using Lambda Expression.





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();**

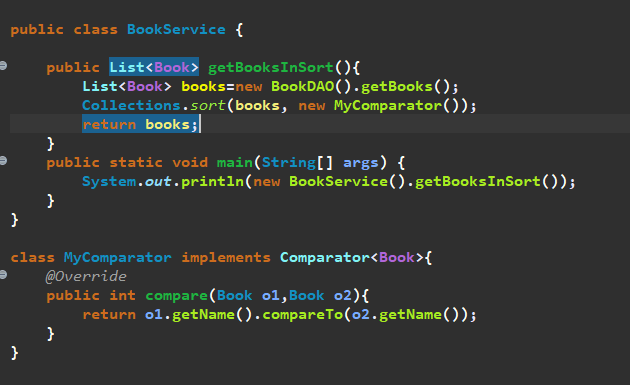
**// }**

**//}**

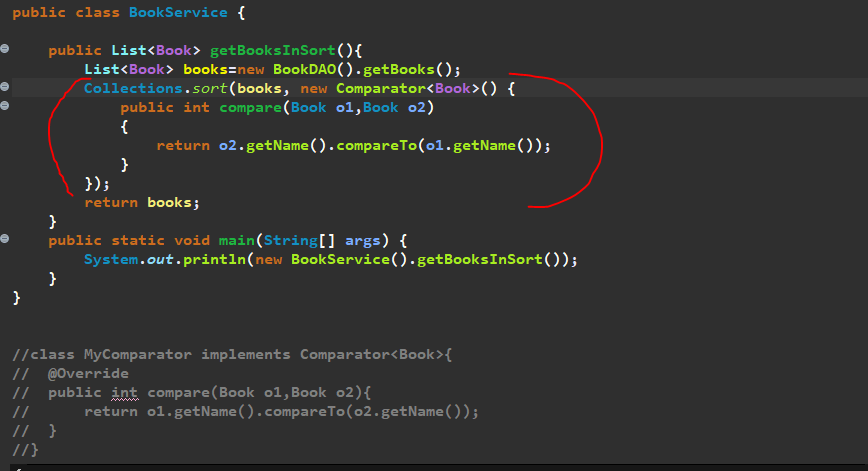
**Real Time Example for Functional Interface & Lambda Expression**

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

**Traditional way of Sorting…..**

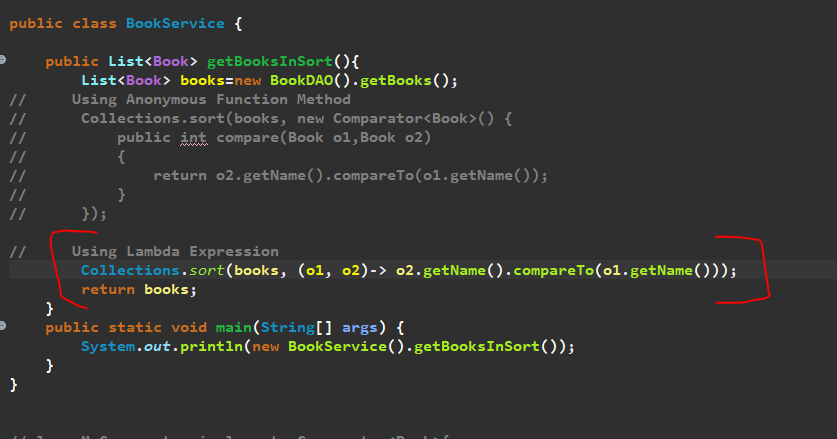


We can write this in another way using Anonymous Function for Functional Interface….Since Comparator is a Functional Interface so we can write Lambda Expression also for the method compare…



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 >>=====================**