**Java8 Lambda Expression**

In this article we will discuss about Java8 most debated feature Lambda Expression.

What is a Lambda Expression?

In brief I can say Lambda Expression is way of passing functions in to a function. Where Passing function is being used by function where it is pass.

Passing function act as a Callback function.

In Java 8 Version Java adopt the Lambda expression actually this is the first step towards functional programming. In functional programing we can pass a function in another function that is called **Higher Order Function.**

The benefits of this is Lambda Expression is

As you can pass a function into an another function or I can say you can pass behavior in to a function (think function as a behavior). So It is on caller hand what behavior you can want to do or I can say the strategy is on caller’s hand so it can be altered easily. Prior to this If you want to pass a behavior probably you will create an anonymous class and wrap that behavior in that class and pass it to the another method.

To be specific prior to Lambda expression we will solve this problem that is passing behavior in to a method by anonymous class.

Think about Thread and Runnable interface. In Runnable interface there is a run method where we define the behavior or the set activities which should be executed when thread is spawned.

And we pass this in Thread constructor so when Thread is in running state it performs the activity written in Runnable. But to do we has to create an anonymous class and override public void run then pass it in Thread class.

This not right way to pass a function rather what we did we pass a object where the function is in wrapped form.

Java8 try to ge rid of this now no need to create that class directly you can pass a method body to a method.

Let’s take an Example

Suppose I need to compute an Area of Circle and Rectangle and print them so How to do it by lambda Expression.

First I create two functional interface, It just a Java interface and it has one method where I specify the behavior template method.

**package** com.example.java;

**public** **interface** Shape {

**public** **int** execute(**int** a,**int** b);

}

**package** com.example.java;

**public** **interface** Print {

**public** **void** print(**int** value);

}

Now I create a Class where we define Rectangle and Circle compute behavior and print behavior by Lambda Expression

package com.example.impl;

import com.example.java.Print;

import com.example.java.Shape;

public class ShapeClosure {

public void computeArea(Shape shape,Print print,int length,int breadth)

{

int result = shape.execute(length, breadth);

print.print(result);

}

public static void main(String[] args) {

ShapeClosure closure = new ShapeClosure();

Shape circle = (a,b)->(int)3.14f\*a\*b;

Print area = (result)->System.out.println("Area of Circle " + result);

closure.computeArea((a,b)->a\*b, (result)->System.out.println("Area of Rectangle " + result), 10, 20);

closure.computeArea(circle, area, 10, 5);

}

}

Look at the ShapeClosure class here I create a ComputeArea Function, the function which will take Shape and Print function as a callback function. In order to do that

Look at the ComputeArea function arguments, it takes shape and Print functional interface as argument. Because this functional interface says about the behavior of the function

Shape interface behavior is execute which takes two int argument and return a int

Print interface behavior is print which takes an int argument and print the value.

In compute are we call this behavior or Other way I can say in a function I use callback funtion’s behaviour which is main motto of functional programming

Now look these lines in main method

Shape circle = (a,b)->(int)3.14f\*a\*b;

Print area = (result)->System.out.println("Area of Circle " + result);

closure.computeArea((a,b)->a\*b, (result)->System.out.println("Area of Rectangle " + result), 10, 20);

closure.computeArea(circle, area, 10, 5);

Here I use Lambda expression as shape’s execute method takes two int’s and compute them so

**(a,b)->(int)3.14f\*a\*b;**

**Two arguments int a and b (Don’t need to say data type from functional interface java takes the argument type.)**

**Now define the body of execute as (int)3.14f\*a\*b;**

**So these line is (a,b)->(int)3.14f\*a\*b;**

**Is equivalent to**

**public int execute(int a,int b)**

**{**

**return (int)3.14f\*a\*b;**

**}**

**Same for print , I pass them in computeArea along with data’s which is main intent of functional programming**

closure.computeArea(circle, area, 10, 5);

On Other hand I can directly pass them in computeArea like this

closure.computeArea((a,b)->a\*b, (result)->System.out.println("Area of Rectangle " + result), 10, 20);

when you run the program. You will see Output

Area of Rectangle 200

Area of Circle 150