Java Lambda Expressions

Syllabus:

Functional Programming: Functional Interfaces – Function, BiFunction, Predicate, and Supplier, Lambda Expression Fundamentals, Block Lambda Expressions, Passing Lambda Expressions as Arguments, Lambda Expressions and Exceptions, Variable Capture, Method References.

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code.

Java lambda expression is treated as a function, so compiler does not create .class file.

Functional Interface

Lambda expression provides implementation of *functional interface*. An interface which has only one abstract method is called functional interface. Java provides an anotation *@FunctionalInterface*, which is used to declare an interface as functional interface.

Why use Lambda Expression

- To provide the implementation of Functional interface.
- Less coding.

Java Lambda Expression Syntax

(argument-list) -> {body}

Java lambda expression is consisted of three components.

Argument-list: It can be empty or non-empty as well.

Arrow-token: It is used to link arguments-list and body of expression.

Body: It contains expressions and statements for lambda expression.

Lambda Expression Parameters

There are three Lambda Expression Parameters are mentioned below

1. Zero Parameter
2. Single Parameter
3. Multiple Parameters

1.No Parameter Syntax(Zero Parameters)

```
() -> {
//Body of no parameter lambda
}
```

2.One Parameter Syntax(One Parameters)

```
(p1) -> {
//Body of single parameter lambda
}
```

3.Two Parameter Syntax(Multiple Parameters)

```
(p1,p2) -> {
//Body of multiple parameter lambda
}
```

Let's see a scenario where we are not implementing Java lambda expression. Here, we are implementing an interface without using lambda expression.

Without Lambda Expression

```
interface Drawable{
    public void draw();
}
class Output implements Drawable {
  @Override
  public void draw() {
    System.out.println("drawing...");
  }}
public class LambdaExpressionDemo1 {
  public static void main(String[] args) {
    // without lambda expression
```

```
        Output output = new Output();
        output.draw();
    }
}
```

Java Lambda Expression Example

Now, we are going to implement the above example with the help of Java lambda expression.

```
@FunctionalInterface  //It is optional
interface Drawable{
    public void draw();
}
public class LambdaExpressionDemo2 {
    public static void main(String[] args) {
        // with lambda expression
        Drawable d1 = () -> {
            System.out.println("drawing...");
        };
        d1.draw();
    }
}
```

A lambda expression can have zero or any number of arguments. Let's see the examples:the above is an example for Lambda expression with no parameter

Java Lambda Expression Example: Single Parameter

```
interface Sayable{
    public String say(String name);
}

public class LambdaExpressionDemo3{
    public static void main(String[] args) {
        // Lambda expression with single parameter.
        Sayable s1=(name)->{
            return "Hello, "+name;
        };
        System.out.println(s1.say("Sonoo"));
        // You can omit function parentheses
        Sayable s2= name ->{
            return "Hello, "+name;
        };
        System.out.println(s2.say("Sonoo"));
    }
}
```

Java Lambda Expression Example: Multiple Parameters

```
interface Calculator{
    int add(int a,int b);
}

public class LambdaExpressionDemo4{
    public static void main(String[] args) {

        // Multiple parameters in lambda expression
        Calculator ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Multiple parameters with data type in lambda expression
       Calculator ad2=(int a, int b)->(a+b);
        System.out.println(ad2.add(100,200));
    }
}
```

Java Lambda Expression Example: with or without return keyword

In Java lambda expression, if there is only one statement, you may or may not use return keyword. You must use return keyword when lambda expression contains multiple statements.

```
interface Calculator {
    int add(int a,int b);
}

public class LambdaExpressionDemo5{
    public static void main(String[] args) {
        // Lambda expression without return keyword.
        Calculator ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));

        // Lambda expression with return keyword.
        Calculator ad2=(int a,int b)->{
                        return (a+b);
                        };
        System.out.println(ad2.add(100,200));
    }
}
```

<u>Java Lambda Expression Example: Foreach Loop</u>

```
import java.util.*;
public class LambdaExpressionDemo6{
    public static void main(String[] args) {

        List<String> list=new ArrayList<String>();
        list.add("ankit");
        list.add("mayank");
        list.add("irfan");
        list.add("jai");

        list.forEach(
            (n)->System.out.println(n)
        );
    }
}
```

<u>Java Lambda Expression Example: Multiple Statements</u>

```
@FunctionalInterface
interface Sayable{
    String say(String message);
}

public class LambdaExpressionDemo7{
    public static void main(String[] args) {

        // You can have multiple statements in lambda expression
        Sayable person = (message)-> {
            String str1 = "I would like to say, ";
            String str2 = str1 + message;
            return str2;
        };
            System.out.println(person.say("time is precious."));
    }
}
```

Java Lambda Expression Example: Creating Thread

You can use lambda expression to run thread. In the following example, we are implementing run method by using lambda expression.

```
public class LambdaExpressionDemo8{
    public static void main(String[] args) {

        //Thread Example with lambda
        Runnable r=()->{
                System.out.println("Thread is running...");
        };
        Thread t=new Thread(r);
        t.start();
    }
}
```

Java lambda expression can be used in the collection framework. It provides efficient and concise way to iterate, filter and fetch data. Following are some lambda and collection examples provided.

Java Lambda Expression Example: Comparator

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Product{
```

```
int id;
String name;
float price;
public Product(int id, String name, float price) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
}
}
public class LambdaExpressionDemo9{
public static void main(String[] args) {
        List<Product> list=new ArrayList<Product>();

        //Adding Products
        list.add(new Product(1,"HP Laptop",25000f));
        list.add(new Product(3,"Keyboard",300f));
        list.add(new Product(2,"Dell Mouse",150f));

        System.out.println("Sorting on the basis of name...");

        // implementing lambda expression
        Collections.sort(list,(p1,p2)->{
        return p1.name.compareTo(p2.name);
        });
        for(Product p:list){
                System.out.println(p.id+" "+p.name+" "+p.price);
        }

}
```

Lambda Expression as arguments

A **lambda expression** passed in a method that has an argument of type of **functional interface**. If we need to pass a lambda expression as an argument, the type of parameter receiving the lambda expression argument must be of a functional interface type.

Example 1: Define lambda expressions as method parameters
import java.util.ArrayList;

```
class LambdaExpressionDemo10 {
    public static void main(String[] args) {
        // create an ArrayList
        ArrayList<String> languages = new ArrayList<>();

        // add elements to the ArrayList
        languages.add("java");
        languages.add("swift");
        languages.add("python");
        System.out.println("ArrayList: " + languages);

        // pass lambda expression as parameter to replaceAll() method
        languages.replaceAll(e -> e.toUpperCase());
        System.out.println("Updated ArrayList: " + languages);
    }
}
```