Java Lambda Expressions

Java lambda Expressions

- 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.

Java lambda Expressions

- 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.

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

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

  3) Body: It contains expressions and statements for lambda expression.

- **No Parameter Syntax**

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

- **One Parameter Syntax**

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

- **Two Parameter Syntax**

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

Without Lambda Expression

```
interface Drawable{
public void draw();
}
public class LambdaExpressionExample {
public static void main(String[] args) {
 int width=10;
 //without lambda, Drawable implementation using anonymous class
Drawable d=new Drawable(){
public void draw(){System.out.println("Drawing "+width);}
};
d.draw();
}
}
```

Output:
Drawing 10

Java Lambda Expression Example

```
@FunctionalInterface  //It is optional
interface Drawable{
 public void draw();
}
public class LambdaExpressionExample2 {
public static void main(String[] args) {
 int width=10;
//with lambda
Drawable d2=()->{
 System.out.println("Drawing "+width);
};
d2.draw();
}
}
```

Java Lambda Expression Example: No Parameter

```
interface Sayable{
    public String say();
}
public class LambdaExpressionExample3{
public static void main(String[] args) {
    Sayable s=()->{
        return "I have nothing to say.";
    };
    System.out.println(s.say());
} }
```

Java Lambda Expression Example: Single Parameter

```
interface Sayable{
    public String say(String name);
}
public class LambdaExpressionExample4{
    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 Addable{
    int add(int a,int b);
}
public class LambdaExpressionExample5{
    public static void main(String[] args) {
        // Multiple parameters in lambda expression
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));
        // Multiple parameters with data type in lambda expression
        Addable ad2=(int a,int b)->(a+b);
        System.out.println(ad2.add(100,200));
    }
}
```

Java Lambda Expression Example: with or without return keyword

```
interface Addable{
    int add(int a,int b);
}
 public class LambdaExpressionExample6 {
    public static void main(String[] args) {
            // Lambda expression without return keyword.
        Addable ad1=(a,b)->(a+b);
        System.out.println(ad1.add(10,20));
            // Lambda expression with return keyword.
        Addable ad2=(int a,int b)->{
                        return (a+b);
                        };
        System.out.println(ad2.add(100,200));
    } }
```

Java Lambda Expression Example: Multiple Statements

```
@FunctionalInterface
interface Sayable{
    String say(String message);
}
  public class LambdaExpressionExample8{
    public static void main(String[] args) {
        // You can pass 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

```
public class LambdaExpressionExample9{
    public static void main(String[] args) {
        //Thread Example without lambda
        Runnable r1=new Runnable(){
            public void run(){
                System.out.println("Thread1 is running...");
            } };
        Thread t1=new Thread(r1);
        t1.start();
        //Thread Example with lambda
        Runnable r2=()->{
                System.out.println("Thread2 is running...");
        };
        Thread t2=new Thread(r2);
        t2.start();  } }
```

Java Lambda Expression Example: Event Listener

```
public class LambdaEventListenerExample {
    public static void main(String[] args) {
        JTextField tf=new JTextField();
        tf.setBounds(50, 50,150,20);
        JButton b=new JButton("click");
        b.setBounds(80,100,70,30);
        // lambda expression implementing here.
        b.addActionListener(e-> {tf.setText("hello swing");});
          JFrame f=new JFrame();
        f.add(tf);f.add(b);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setLayout(null);
        f.setSize(300, 200);
        f.setVisible(true);
    } }
```

Lambda expression exception handling example

```
interface IFuncInt {
    int func(int num1, int num2) throws Exception;
}
public class LambdaExceptionDemo {
    public static void main(String[] args){
        IFuncInt funcInt = (num1, num2) -> {
            int result = num1 +  num2;
            throw new Exception();
        };
        try {
            System.out.println("" + funcInt.func(6, 7));
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace(); }}}
```