Topics to cover:

1. Lambda expressions

    1. Lambda expression

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 the collection library. It helps to iterate, filter and extract data from collections.

The Lambda expression is used to provide the implementation of an interface which has a 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 the compiler does not create a .class file.

Functional Interface

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

Why use Lambda Expression

1. To provide the implementation of Functional interface.
2. Less coding.

Java Lambda Expression Syntax

```
(argument-list) -> {body}
```

Java lambda expression consists 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

}
```

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();

}

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

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 LambdaExpressionExample2 {

   public static void main(String[] args) {

      int width=10;



      //with lambda

      Drawable d2=()->{

         System.out.println("Drawing "+width);
```

```
        };
        d2.draw();
    }
}
```

Output:

```
Drawing 10
```

A lambda expression can have zero or any number of arguments. Let's see the examples:

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());
```

```
}

}
```

Output:

I have nothing to 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"));
    }
}
```

Output:

```
Hello, Sonoo
Hello, 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));

    }

}
```

Output:

```
30

300
```

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 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));
    }
}
```

Output:

```
30
300
```

Java Lambda Expression Example: Foreach Loop

```
import java.util.*;
public class LambdaExpressionExample7{
```

```
    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)

        );

    }

}
```

Output:

```
ankit
mayank
irfan
jai
```

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."));
    }
}
```

Output:

```
I would like to 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 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();
    }
}
```

Output:

```
Thread1 is running...
Thread2 is running...
```

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) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
public class LambdaExpressionExample10{
    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);
        }
```

```
        }
    }
```

Output:

```
Sorting on the basis of name...
2 Dell Mouse 150.0
1 HP Laptop 25000.0
3 Keyboard 300.0
```