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.
* 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

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

Java Lambda Expression Syntax

1. (argument-list) -> {body}
2. 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**

1. () -> {
2. //Body of no parameter lambda
3. }

**One Parameter Syntax**

1. (p1) -> {
2. //Body of single parameter lambda
3. }

**Two Parameter Syntax**

1. (p1,p2) -> {
2. //Body of multiple parameter lambda
3. }

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();

        d.draw();

}

**public** **void** draw(){System.out.println("Drawing "+width);}

    }

Output:

Drawing 10

Java Lambda Expression Example

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

**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

1. **interface** Sayable{
2. **public** String say();
3. }
4. **public** **class** LambdaExpressionExample3{
5. **public** **static** **void** main(String[] args) {
6. Sayable s=()->{
7. **return** "I have nothing to say.";
8. };
9. System.out.println(s.say());
10. }
11. }

Output:

I have nothing to say.

Java Lambda Expression Example: Single Parameter

1. **interface** Sayable{
2. **public** String say(String name);
3. }
5. **public** **class** LambdaExpressionExample4{
6. **public** **static** **void** main(String[] args) {
8. // Lambda expression with single parameter.
9. Sayable s1=(nm)->{
10. **return** "Hello, "+nm;
11. };
12. System.out.println(s1.say("Sonoo"));
14. // You can omit function parentheses
15. Sayable s2= name ->{
16. **return** "Hello, "+name;
17. };
18. System.out.println(s2.say("Sonoo"));
19. }
20. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=LambdaExpressionExample4)

Output:

Hello, Sonoo

Hello, Sonoo

Java Lambda Expression Example: Multiple Parameters

1. **interface** Addable{
2. **int** add(**int** a,**int** b);
3. }
5. **public** **class** LambdaExpressionExample5{
6. **public** **static** **void** main(String[] args) {
8. // Multiple parameters in lambda expression
9. Addable ad1=(a,b)->{(a+b);
10. System.out.println(ad1.add(10,20))};
12. // Multiple parameters with data type in lambda expression
13. Addable ad2=(**int** a,**int** b)->{(a+b);
14. System.out.println(ad2.add(100,200));
15. }
16. }