LAMBDA EXPRESSION IN JAVA

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

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.)

Lambda Expression?

1. Lambda is a anonymous function:
   - ✧ No name
   - ✧ No access modifier
   - ✧ No return type

   Normal Function in Java:

```
public void printMessage() {
    System.out.println("Hello, this is normal function");
}
```

   Converted into Lambda Expression

```
()->{System.out.println("Hello, this is Lembda Expression");}
```

Why use Lambda Expression?

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

Important Rules of Lambda:

1. If the body of Lambda Expression contain only one statement then curly braces are optional.
2. Java Compiler also inter the type of variable passed in arguments, hence type is optional.

Java Lambda Expression Syntax:

(argument-list) -> {body}

Java lambda expression is consisted of three components:

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

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

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

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. (ex- Runnable, Callable, Comparable etc.)

Example:

```java
@FunctionalInterface
public interface MyFunInterface {
    public abstract void printData();
}
```

#How to execute Interface Abstract Method

- Create Separate class, implement the interface abstract method and create the object of class and call the implemented method:

a) Interface with one abstract method:

```java
@FunctionalInterface
public interface MyFunInterface {
    public abstract void printData();
}
```

b) class implements interface - MyFunInterface:

```java
public class MyFuncInterfaceImpl implements MyFunInterface {

    @Override
    public void printData() {
        // TODO Auto-generated method stub
        System.out.println("Hello from impl of myFunInterface");
    }
}
```

c) main method to create the object and call the implemented method:

```java
public class LamdaMain {

    public static void main(String[] args) {

        MyFuncInterfaceImpl impl = new MyFuncInterfaceImpl();
        impl.printData();
    }
}
```

- Anonymous class for implementing interface:

a) Interface with one abstract method:

```java
@FunctionalInterface
public interface MyFunInterface {
    public abstract void printData();
}
```

b) main method with anonymous class to provide implementation of interface abstract method:

```
public class LamdaMain {

    public static void main(String[] args) {

        MyFunInterface impl = new MyFunInterface() {

            @Override
            public void printData() {
                // TODO Auto-generated method stub
                System.out.println("Hello from impl of myFunInterface");
            }
        };

        impl.printData();
    }
}
```

- Lambda Expression with Functional interface:

a) Interface with one abstract method:

```
@FunctionalInterface
public interface MyFunInterface {
    public abstract void printData();
}
```

b) main method with lambda expression to provide implementation of functional interface:

```
public class LamdaMain {

    public static void main(String[] args) {

        MyFunInterface impl=()->System.out.println("Hello from impl of myFunInterface");
        impl.printData();
    }
}
```

[Reference- w3schools, geeksforgeeks, javatpoint]

#Lambda Expression with multiple parameters:

a) Functional Interface

```
public interface TwoArgFunInterface {

    void getSum(int a, int b);
}
```

b) main method with Lambda Expression

```
public class LamdaMain {

    public static void main(String[] args) {

        TwoArgFunInterface impl = (a, b) -> System.out.println("Sum- " + (a + b));
        impl.getSum(100, 200);
    }
}
```

#Lambda Expression with and 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.

a) Functional Interface:

```
public interface TwoArgFunInterface {

    int getSum(int a, int b);
}
```

b) Single Statement (without return keyword)

```
public class LamdaMain {

    public static void main(String[] args) {

        TwoArgFunInterface impl = (a, b) -> a + b;
        System.out.println(impl.getSum(100, 200));
    }
}
```

c) Multiple Statements (with return keyword)

```
public class LamdaMain {

    public static void main(String[] args) {

        TwoArgFunInterface impl = (a, b) -> {
            return (a + b);
        };
        System.out.println(impl.getSum(100, 200));
    }
}
```