@techwithvishalraj

# Java

# Lambda expressions

Follow

Share

@techwithvishalraj

# Lambda expressions

- **The Lambda expression is used to provide the implementation of an interface which called as 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.**
- **It helps to iterate, filter and extract data from collection.**

@techwithvishalraj

# 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.**

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

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

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

@techwithvishalraj

# Without Lambda Expression

```
interface Welcome{
    public void hello();
}

public class Test1 {
    public static void main(String[] args) {

        //without lambda, Welcome implementation using anonymous class
        Welcome welcome=new Welcome() {
            public void hello() {
                System.out.println("Hey, Hii :) ");
            }
        };
        welcome.hello();
    }
}
```

output: Hey, Hii :)

@techwithvishalraj

# With Lambda Expression

```
@FunctionalInterface      // it is optional
interface Welcome{
   public void hello();
}

public class Test2 {
   public static void main(String[] args) {

      // You can omit function parentheses
      Welcome welcome=() -> {
                        System.out.println("Hey, Hii :)  ");
      };

      welcome.hello();
   }
}
```

output: Hey, Hii :)

# Lambda Expression with single parameter

@techwithvishalraj

```
@FunctionalInterface
interface Welcome{
   public void hello(String name);
}

public class Test3 {
   public static void main(String[] args) {
      Welcome welcome=(name) -> System.out.println("Hey, " + name);
      welcome.hello("Raju");
   }
}
```

output: Hey, Raju

@techwithvishalraj

# Lambda Expression with multiple parameter

```
@FunctionalInterface
interface Addition{
    public int add(int a, int b);
}

public class Test4 {
    public static void main(String[] args) {
        Addition addition = (x, y)-> x+y;
        System.out.println(addition.add(4, 5));
    }
}
```

output: 9

@techwithvishalraj

# Lambda Expression with or without return keyword

```
@FunctionalInterface
interface Addition{
    public int add(int a, int b);
}

public class Test4 {
 public static void main(String[] args) {

    //if there is only one statement, you may or may not use return keyword.
    Addition addition = (x, y)-> x+y;
    System.out.println(addition.add(4, 5));
```

output: 9

```
    Addition addition1 = (x, y)->{
                System.out.println("adding.....");
                return (x+y);
    };
    System.out.println(addition1.add(4, 5));
 }
}
```

output: adding.....
9

@techwithvishalraj

# Lambda Expression with return keyword must

**You must use return keyword when lambda expression contains multiple statements.**

```
@FunctionalInterface
interface CheckGrade{
    public String check(int a);
}

public class Test5 {
    public static void main(String[] args) {
        CheckGrade checkGrade = (a)-> {
            if(a>=80) return "A";
            else if(a>=60) return "B";
            else if(a>=35) return "C";
            else return "F";
        };

        System.out.println(checkGrade.check(96));
        System.out.println(checkGrade.check(38));
    }
}
```

output: A
C

@techwithvishalraj

# Lambda Expression with Foreach Loop

```
import java.util.*;

public class Test6 {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        list.add(6);
        list.add(2);
        list.add(4);

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

output: 6
2
4

@techwithvishalraj

# Thank you!

vishal-bramhankar

techwithvishalraj

Vishall0317