Lambda Expressions in Java

A **lambda expression** in Java is a short way to write an implementation of a **functional interface**.
It was introduced in **Java 8** mainly to make code more concise, especially when working with collections, streams, and event handling.

---

1. First understand the problem

Suppose we want to create a method that prints a message.

Without lambda:

```
interface Greeting {
    void sayHello();
}

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

        Greeting g = new Greeting() {
            @Override
            public void sayHello() {
                System.out.println("Hello!");
            }
        };

        g.sayHello();
    }
}
```

This works, but there is a lot of code for a very simple operation.

With a lambda:

```
interface Greeting {
    void sayHello();
}

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

        Greeting g = () -> {
            System.out.println("Hello!");
        };

        g.sayHello();
    }
}
```

Even shorter:

```
Greeting g = () -> System.out.println("Hello!");
```

That's the main purpose of lambda expressions: **less boilerplate code**.

---

2. Lambda Expression Syntax

The general syntax is:-

```
(parameters) -> expression
```

or

```
(parameters) -> {
   statements;
}
```

For example:

```
(int a, int b) -> a + b
```

Here:

- `(int a, int b)` → parameters
- `->` → lambda operator
- `a + b` → expression/result

Another example:

```
() -> System.out.println("Hello");
```

Here there are **no parameters**.

3. Simple Example

Let's create a functional interface Mylambda.java file code :-

```
interface Calculator {

    int add(int a, int b);

}

public class Mylambda {


    public static void main(String[] args) {


        // Lambda expression

        Calculator c = (a, b) -> a + b;


        // Calling the add() method

        System.out.println(c.add(10, 20));

    }

}
```

Output:-

30

Note:-

What is a Functional Interface?

This is **very important** for understanding lambdas.
A **functional interface** is an interface that has exactly **one abstract method**.

Example:

```
interface Calculator {
    int add(int a, int b);
}
```

It has only one abstract method:
add()
Therefore, it is a functional interface.

Lambda With One Parameter

```
@FunctionalInterface

interface Printer {

    void print(String message);

}


public class Main1 {


    public static void main(String[] args) {


        // Lambda expression

        Printer p = message -> System.out.println(message);


        // Calling the print() method

        p.print("Hello");

    }
```

```
}
```

Lambda With Multiple Parameters

```
@FunctionalInterface
interface Calculator {
    int multiply(int a, int b);
}

public class Main2 {

    public static void main(String[] args) {

        // Lambda expression with multiple parameters
        Calculator c = (a, b) -> a * b;

        // Calling the multiply() method
        System.out.println(c.multiply(5, 4));
    }
}
```