Lambda Expression

Lambda expressions in Java are a feature introduced in **Java 8** (2014)

It allows you to create **anonymous functions** (i.e., functions without a name) and treat them as **first-class citizens**.
*(A **first-class citizen** here means that something (like a function) that can be:*

- ***Stored in variables***
- ***Passed as arguments***
- ***Returned from other functions**)*

Syntax -
(parameters) -> expression
Or
(parameters) -> { statements; }

Example - Runnable with Lambda
Runnable r = () -> System.out.println("Hello from a thread!");
new Thread(r).start();

�� Key Characteristics

1. Can be assigned to variables or passed as arguments.

```
import java.util.function.Function;
public class Main {
   public static void main(String[] args) {
       Function<Integer, Integer> square = x -> x * x;
       System.out.println(square.apply(5)); // Output: 25
   }
}
```

Here in the above code the lambda expression is being assigned to a variable named square.

```
import java.util.function.Function;
```

```
public class Main {
    static void printResult(int x, Function<Integer, Integer> func) {
        System.out.println("Result: " + func.apply(x));
    }

    public static void main(String[] args) {
        printResult(4, x -> x * x); // Output: Result: 16
    }
}
```

Here in this code the lambda expression is being passed to a function named ***printResult.***

2. **Only usable with functional interfaces (interfaces with one abstract method). Using**

Custom Functional Interface

```
@FunctionalInterface
interface MyOperation {
    int operate(int x);
}

public class Main {
    public static void main(String[] args) {
        MyOperation doubleIt = x -> x * 2;
        System.out.println(doubleIt.operate(6)); // Output: 12
    }
}
```

Using Built-in Functional Interface

```
import java.util.function.Predicate;

public class Main {
    public static void main(String[] args) {
        Predicate<String> isLong = s -> s.length() > 5;
        System.out.println(isLong.test("Hello")); // false
        System.out.println(isLong.test("Welcome")); // true
    }
}
```

3. **Makes code more concise and readable.**

Functional Interface

A **functional interface** in Java is an interface that has **exactly one abstract method**. It's designed to represent a **single "function" behavior**, which makes it perfect for use with **lambda expressions**.

✅ Key Points about Functional Interfaces:

Feature	**Description**
�� One abstract method	Only one method without a body (abstract), so lambdas can implement it.
�� Can have default/static methods	But still only **one abstract method** is allowed. ��
Marked with @FunctionalInterface	Optional but recommended for compiler checks.

Example -

```
@FunctionalInterface
interface MyFunction {
   int doSomething(int x);
}
```

A Simple code vs ***A Functional interface and Lambda Expression code***

A simple java code to write a square function -

```
public class Main
{
   int sq(int x)
   {
      return x*x;
   }

   public static void main(String[] args) {
     Main m = new Main();
          System.out.println(m.sq(5));
   }
}
```

Code with same functionality using Functional Interface and Lambda Expressions -

```
@FunctionalInterface
interface MyFuctionalInterface{
   int apply(int x);
}

public class Main
{
         public static void main(String[] args) {
            MyFuctionalInterface fi = (x) -> x*x;
    System.out.println(fi.apply(5));
         }
}
```

Differences -

Code Comparison

Aspect Traditional Method Lambda + Functional Interface Code Method in a class

Lambda implementing an interface **Style** Object-oriented Functional programming

Boilerplate More code for small tasks Less code, more concise **Flexibility** Tied to a class

Can be passed around easily **Reusability** Harder to reuse dynamically Easily reused or passed as parameter **Performance** Slightly better (no indirection) Slight overhead (via interface)