functional interfaces and lambdas are features introduced in Java 8 to support functional programming and enable more concise and expressive code. Here's an overview of functional interfaces and lambdas in Java:

Functional Interfaces: A functional interface is an interface that defines a single abstract method, known as the functional method. Functional interfaces serve as the foundation for lambda expressions in Java. The **java.util.function** package provides several predefined functional interfaces, such as **Predicate**, **Consumer**, **Function**, and **Supplier**. These interfaces encapsulate common functional patterns and allow you to pass behavior as arguments or return types.

Lambda Expressions: A lambda expression is a concise way to represent an anonymous function. It allows you to express functionality directly in-line without the need for writing a separate method or class. Lambda expressions are commonly used in combination with functional interfaces.

Syntax of a Lambda Expression: A lambda expression consists of three parts: a parameter list, an arrow token (**->**), and a body. The parameter list specifies the inputs to the lambda expression, the arrow token separates the parameter list from the body, and the body contains the code to be executed.

Here's the general syntax of a lambda expression:

(parameter1, parameter2, ...) -> {

// Body of the lambda expression

// Perform some operations using the parameters

// Return a value (if necessary)

}

Example Usage: Let's take an example of using a functional interface and a lambda expression:

import java.util.function.Predicate;

public class LambdaExample {

public static void main(String[] args) {

// Create a Predicate functional interface using a lambda expression

Predicate<Integer> isEven = (number) -> number % 2 == 0;

// Use the Predicate to test if a number is even

System.out.println(isEven.test(4)); // Output: true

System.out.println(isEven.test(7)); // Output: false

}

}