**1. What is the lambda expression of Java 8+?**Lambda expressions in Java 8+ are a concise way to represent a functional interface's implementation. They provide a clear and streamlined syntax for defining small, anonymous functions. Lambda expressions eliminate the need for boilerplate code like defining anonymous inner classes for single-method interfaces.

### **Syntax:**

(parameters) -> expression

(parameters) -> { statements; }

### **Example:**

import java.util.Arrays;

import java.util.List;

public class LambdaExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("John", "Alice", "Bob");

// Using a lambda expression to print each name

names.forEach(name -> System.out.println(name));

}

}

// Output:

// John

// Alice

// Bob

**2. Can you pass lambda expressions to a method? When?**Yes, lambda expressions can be passed to a method as arguments when the method expects a parameter of a functional interface type. This is particularly useful for operations like filtering, sorting, or performing actions on collections.

### **Example:**

import java.util.Arrays;

import java.util.List;

public class LambdaArgumentExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

// Passing a lambda expression to a method

numbers.forEach(num -> System.out.println(num \* num));

}

}

// Output:

// 1

// 4

// 9

// 16

// 25

### **When to Use:**

* When dealing with functional-style programming tasks.
* When working with APIs like Stream and Collections.
* When implementing behavior as arguments for reusable methods.

**3. What is the functional interface in Java 8?**A **functional interface** in Java 8 is an interface that contains exactly one abstract method. It can have default and static methods but must have only one abstract method to be compatible with lambda expressions. Functional interfaces are annotated with @FunctionalInterface to ensure compliance.

### **Example of a Functional Interface:**

@FunctionalInterface

interface Greeting {

void sayHello(String name);

}

### **Example Usage with Lambda Expression:**

java

public class FunctionalInterfaceExample {

public static void main(String[] args) {

Greeting greeting = name -> System.out.println("Hello, " + name);

greeting.sayHello("Alice");

}

}

// Output: Hello, Alice

**4. Why do we use lambda expressions in Java?**Lambda expressions are used to achieve cleaner, more concise code and promote functional programming in Java. They provide several benefits:

* **Reduction in Boilerplate Code:** Simplifies implementation of single-method interfaces.
* **Improved Readability:** Allows focus on behavior rather than structure.
* **Enhanced API Usage:** Works seamlessly with Java 8 APIs like Stream and Optional.
* **Parallelism:** Simplifies writing parallel processing logic with the Streams API.

**5. Is it mandatory for a lambda expression to have parameters?**No, it is not mandatory for a lambda expression to have parameters. Lambda expressions can have zero, one, or multiple parameters depending on the functional interface they implement.

### **Example of Zero Parameters:**

java

@FunctionalInterface

interface NoParamFunction {

void execute();

}

public class LambdaNoParam {

public static void main(String[] args) {

NoParamFunction action = () -> System.out.println("Lambda with no parameters!");

action.execute();

}

}

// Output: Lambda with no parameters!

### **Example of Multiple Parameters:**

java

@FunctionalInterface

interface MathOperation {

int operate(int a, int b);

}

public class LambdaMultipleParams {

public static void main(String[] args) {

MathOperation addition = (a, b) -> a + b;

System.out.println("Sum: " + addition.operate(10, 20));

}

}

// Output: Sum: 30