**Assignment Lambada Expression**

Q1. What is the lambda expression of Java 8+

A lambda expression in Java 8+ is a way to write anonymous functions in a concise manner. It allows you to treat functionality as a method argument or store it in a variable, enabling more functional programming techniques.

**Syntax of a Lambda Expression:**

The basic syntax of a lambda expression is as follows:

(parameters) -> expression

Or, if the body has multiple statements:

(parameters) -> {

// body

}

Basic Example (No parameters):

Runnable r = () -> System.out.println("Hello, World!");

Q2.Can you pass lambda expressions to a method? When+

Yes, you can pass lambda expressions to methods in Java 8+ when the method expects an argument of a functional interface type. A functional interface is an interface with exactly one abstract method, and lambda expressions are perfect for implementing these interfaces on the fly.

**When to Pass Lambda Expressions to a Method:**

1. **When a Method Expects a Functional Interface:** If a method has a parameter that is a functional interface, you can pass a lambda expression directly as an argument.
2. **When You Need to Pass Behavior to a Method:** Lambdas are useful for passing behavior (code logic) to methods. This is common in APIs like collections, streams, and custom methods that work on specific tasks.

Q3. What is the functional interface in Java 8?

In Java 8, a **functional interface** is an interface that contains exactly one abstract method. These interfaces are designed to be used with lambda expressions, method references, and constructor references. Functional interfaces enable a more functional style of programming in Java.

**Key Characteristics of Functional Interfaces:**

1. **Single Abstract Method (SAM):** A functional interface has only one abstract method. This is sometimes referred to as the Single Abstract Method (SAM) rule.
2. **Default and Static Methods:** Functional interfaces can have any number of default or static methods. These methods do not count as abstract methods.
3. **@FunctionalInterface Annotation:** While optional, it’s a good practice to use the @FunctionalInterface annotation to explicitly declare an interface as a functional interface. This annotation ensures that the interface cannot have more than one abstract method, helping to prevent accidental violations of the SAM rule.

Q4. Why do we use lambda expressions in Java

Lambda expressions in Java 8+ are used to enable a more concise, readable, and functional style of programming. They allow you to treat functionality as a method argument or store it in a variable, significantly reducing boilerplate code and making certain operations easier and more intuitive to express.

**Key Reasons to Use Lambda Expressions in Java:**

1. **Conciseness:**
   * Lambda expressions allow you to write less code. They eliminate the need for anonymous classes that were commonly used before Java 8 to implement functional interfaces. This reduces the verbosity of your code and makes it easier to read and maintain.

// Before Java 8 (using anonymous class)

new Thread(new Runnable() {

@Override

public void run() {

System.out.println("Hello, World!");

}

}).start();

// Using Lambda Expression

new Thread(() -> System.out.println("Hello, World!")).start();

2. **Functional Programming Support:**

* Lambda expressions are the foundation of functional programming in Java. They enable the use of higher-order functions (functions that take other functions as arguments or return them as results), making it easier to implement behaviors like map-reduce, filtering, and more.

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

// Using lambda with Stream API to filter and print even numbers

numbers.stream()

.filter(n -> n % 2 == 0)

.forEach(n -> System.out.println(n));

Q5. 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 be used with or without parameters, depending on the requirements of the functional interface they are implementing.

* **No Parameters:** Lambda expressions can be used without parameters if the functional interface’s method does not require any.
* **Single Parameter:** Lambda expressions can include a single parameter when the functional interface’s method takes one parameter.
* **Multiple Parameters:** Lambda expressions can handle multiple parameters if needed by the functional interface’s method.

Lambda expressions are flexible and can be adapted to the needs of the functional interface they are intended to implement, making them a powerful tool for writing concise and expressive code.