Lambda Expression Assignment Questions

Assignment Questions:

1. What is the Lambda expression in Java 8?

Ans → Lambda expressions are a new feature introduced in Java 8 that allow the developer to write more concise and expressive code by providing a way to pass behavior as a parameter to a method.

1. Can you pass lambda expressions to a method? When?

Ans → Yes, Lambda expressions can be passed as arguments to methods that accept functional interfaces as parameters. This is one of the main features of Lambda expressions in Java 8 that enables functional programming in Java.

A functional interface is an interface that has only one abstract method and can be used as the target of a Lambda expression. When a Lambda expression is passed as a parameter to a method, it is converted into an instance of the functional interface that matches its signature.

Here is an example of a method that accepts a functional interface as a parameter and a Lambda expression that is passed to the method:

interface MyFunction {

int apply(int a, int b);

}

public static void processNumbers(int x, int y, MyFunction func) {

int result = func.apply(x, y);

System.out.println("Result: " + result);

}

public static void main(String[] args) {

processNumbers(5, 3, (a, b) -> a + b);

}

1. What is the functional interface in Java 8?

Ans → A functional interface in Java 8 is an interface that has only one abstract method and is used as the target for Lambda expressions and method references. Functional interfaces are used extensively in Java 8's new functional programming features to enable the passing of behavior as a parameter to methods.

Functional interfaces are annotated with the @FunctionalInterface annotation, which indicates to the compiler that the interface should only have one abstract method. If an interface is annotated with @FunctionalInterface and has more than one abstract method, the compiler will report an error.

1. Why do we use lambda expression in Java?

Ans → Lambda expressions in Java provide a concise and expressive way to pass behavior as a parameter to a method, enabling more flexible and modular code. There are several reasons why we use Lambda expressions in Java:

1. Improved code readability: Lambda expressions can make code more concise and expressive by removing the need for anonymous inner classes, which can make code easier to read and understand.
2. Enable functional programming: Lambda expressions enable functional programming in Java by providing a way to pass behavior as a parameter to methods, allowing developers to write more declarative and functional code.
3. Simplify code maintenance: Lambda expressions can simplify code maintenance by reducing the number of classes and interfaces required to implement behavior, which can make code easier to refactor and modify.
4. Increase code efficiency: Lambda expressions can help to increase code efficiency by reducing the amount of boilerplate code required to implement common patterns, such as iteration and filtering.
5. Is it mandatory for a lambda expression to have parameters?

Ans → No, it is not mandatory for a lambda expression in Java to have parameters. A lambda expression can have zero, one, or multiple parameters, depending on the functional interface that it is implementing.

If a functional interface has no parameters, the Lambda expression that implements it will also have no parameters. Here is an example of a Lambda expression with no parameters:

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