# Implementing Runnable Interface: Legacy vs Lambda

## Introduction

In this tutorial, we will implement the \*\*Runnable interface\*\* using two different methods: the legacy approach using anonymous classes and the modern approach using \*\*Lambda expressions\*\*. We will compare both and discuss the advantages of one over the other.

## Project Setup

1. \*\*IDE\*\*: \*\*IntelliJ IDEA\*\* is used for the development. Version 2024.3.1 is being used at the time of recording.

2. \*\*Java version\*\*: Java \*\*21\*\* is selected. Ensure your IDE is updated to support Java 21.

3. Create a new project with the name \*\*Java Eight\*\* to explore Java 8 features.

4. Create a package called \*\*lambdas\*\* for this implementation.

## Runnable Interface

1. The \*\*Runnable interface\*\* has a single method: `run()`, which takes no input and returns no output.

2. It is commonly used with \*\*Thread class\*\* to execute tasks in parallel.

3. \*\*@FunctionalInterface\*\* annotation was introduced in JDK 1.8 to mark interfaces with a single abstract method.

4. \*\*Runnable\*\* is one of the pre-existing functional interfaces in Java.

## Legacy Implementation Using Anonymous Classes

1. The first implementation uses the \*\*anonymous class\*\* to implement the `Runnable` interface.

2. We define the `run()` method inside the anonymous class and pass it to a \*\*Thread\*\* instance.

### \*\*Code Implementation (Legacy way):\*\*

public class RunnableExample {  
 public static void main(String[] args) {  
 Runnable runnable = new Runnable() {  
 @Override  
 public void run() {  
 System.out.println("Inside runnable 1.");  
 }  
 };  
 new Thread(runnable).start();  
 }  
}

3. The program works and prints the output: `Inside runnable 1.`

## Lambda Implementation of Runnable

1. In Java 8, we can implement the `Runnable` interface using \*\*Lambda expressions\*\*.

2. \*\*Lambda syntax\*\* simplifies the code, making it more concise and readable.

### \*\*Code Implementation (Lambda way):\*\*

public class RunnableLambdaExample {  
 public static void main(String[] args) {  
 Runnable runnableLambda = () -> System.out.println("Inside runnable 2.");  
 new Thread(runnableLambda).start();  
 }  
}

3. This implementation works similarly and prints the output: `Inside runnable 2.`

## Comparison of Legacy and Lambda Implementations

### \*\*Legacy Approach (Anonymous Class):\*\*

- Requires multiple lines of code to create an instance of `Runnable` and pass it to a `Thread`.

- Involves creating an anonymous class, making the code longer and harder to read.

- Typically involves more \*\*boilerplate code\*\*.

### \*\*Lambda Approach:\*\*

- \*\*Lambda expressions\*\* simplify the code by eliminating the need for anonymous classes.

- Lambda expressions are \*\*more concise\*\* and \*\*readable\*\*.

- With Lambdas, we avoid verbosity, making the code \*\*easier to maintain\*\*.

## Simplified Lambda Expression

1. In cases where the `run()` method consists of a \*\*single statement\*\*, Lambda expressions can be even more simplified.

2. For example, no need for curly braces when there is only one statement.

### \*\*Simplified Code Example:\*\*

Runnable runnableLambda = () -> System.out.println("Inside runnable 3.");  
new Thread(runnableLambda).start();

3. This one-liner shows how concise Lambda expressions can be, even when creating threads with the `Runnable` interface.

## Direct Lambda Usage in Thread

1. Instead of creating a `Runnable` variable, you can directly pass the Lambda expression to the `Thread` constructor.

2. This further reduces the verbosity of the code.

### \*\*Code Example:\*\*

new Thread(() -> System.out.println("Inside runnable 4.")).start();

3. This concise implementation works in the same way but avoids creating a separate `Runnable` variable.

## Handling Multiple Statements in Lambda

1. If your Lambda expression involves \*\*multiple statements\*\*, you need to use curly braces to encapsulate the body.

### \*\*Code Example with Multiple Statements:\*\*

Runnable runnableLambda = () -> {  
 System.out.println("Inside runnable 5.");  
 System.out.println("Multiple statements in lambda.");  
};  
new Thread(runnableLambda).start();

2. When Lambda involves multiple statements, curly braces are required to indicate the start and end of the body.

## Summary

1. \*\*Lambda expressions\*\* simplify the code, making it more concise and readable compared to traditional anonymous classes.

2. \*\*Lambda\*\* eliminates the need for creating separate classes, reducing boilerplate code.

3. In cases of \*\*single-line statements\*\*, no curly braces are needed.

4. For \*\*multiple statements\*\*, curly braces are mandatory in Lambda.

5. In this tutorial, we've shown the difference between legacy and Lambda implementations using the `Runnable` interface.

### \*\*End of Tutorial\*\*

Thank you for watching!