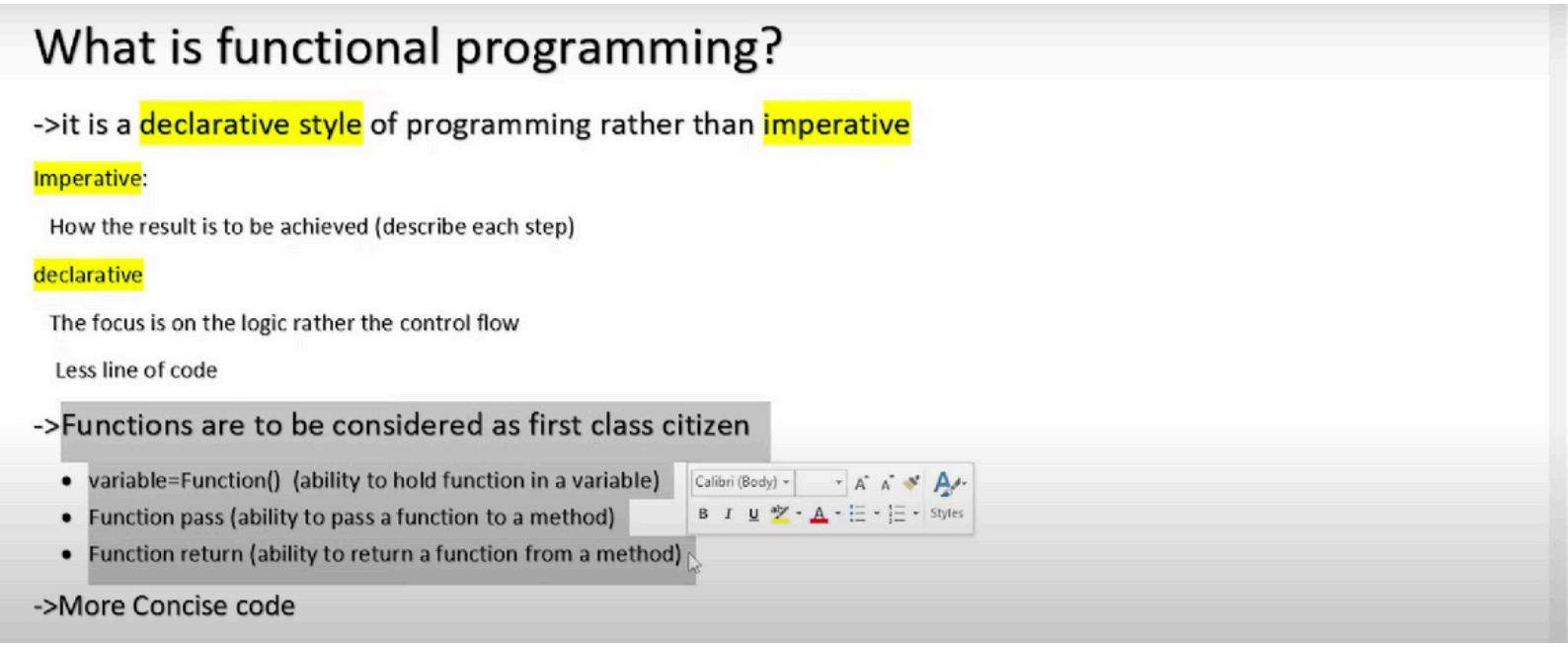
Lec 48 JAVA 8 P4:



What is Functional Programming:

This video explains how lambda expressions in Java make code shorter, more efficient, and enable a style of programming called functional programming. It highlights the difference between imperative(procedure focused) and declarative(logic focused) programming and shows how functions can be treated like variables—held, passed, and returned (first class citizen) —making code easier to reuse and maintain.

Lambda Expressions and Functional Interfaces

Lambda expressions are a way to write concise code for single-method interfaces, called functional interfaces. They reduce the need for verbose(**using more words**

than necessary to express an idea or convey information) anonymous inner classes and make the code more efficient.

Benefits of Lambda Expressions

Lambda expressions reduce lines of code, improve efficiency, and enable functional programming, which is a different way of thinking about how to write programs.

Introduction to Functional Programming

Functional programming is a declarative style where you focus on what you want to achieve (the logic), not on every step to get there. This contrasts with the imperative style, which details each step.

Declarative vs. Imperative Programming

In imperative programming, you specify all steps to achieve a result. In declarative (functional) programming, you describe the desired result and let the language handle the steps.

Example: Logic-Focused Code

By using lambda expressions, you can write just the core logic without specifying method names, data types, or steps, making the code much simpler and focused.

Functional Programming in Java

Functional programming is not unique to Java but is supported in Java. It emphasizes writing less code and focusing on logic, which can be applied in many programming languages.

Functions as First-Class Citizens

In functional programming, functions can be stored in variables, passed as arguments, and returned from methods—just like objects(first class citizen). Java supports this from version 8 onward.

Criteria for Functional Programming

For a language to support functional programming, it must allow holding, passing, and returning functions. Lambda expressions in Java fulfill all these criteria, making code more reusable.

Example: Passing and Reusing Functions

You can define a lambda function once and use it in multiple places by passing it as a parameter, which avoids repeating the same logic and reduces redundancy.

Returning Functions from Methods

Methods can return lambda expressions, allowing you to create reusable logic blocks that can be used whenever needed, further promoting code reuse.

Practical Use: Multithreading with Lambdas

Lambda expressions can simplify multithreading by replacing the need for separate classes or anonymous inner classes to implement Runnable, making thread creation and management easier.

Direct Lambda Usage in Threads

Instead of creating variables for lambdas, you can pass them directly where needed, such as in thread constructors, further reducing code length and complexity.

Practice and Syntax Awareness

Regular practice with lambda expressions helps avoid syntax errors and makes it easier to transition from older Java coding styles. Comparing with anonymous inner classes can help understand the differences.

Preview of Stream API

The next topic to be covered is the Stream API, which builds on the concepts of lambda expressions and functional programming to process collections more efficiently.