Lambda Expressions in Java ☕

Lambda expressions are a powerful feature introduced in Java 8 that allow for a more concise and functional approach to programming. They enable developers to write instances of single-method interfaces (functional interfaces) in a more readable and expressive manner. This document provides an overview of lambda expressions, their syntax, and practical examples to illustrate their usage.

What is a Lambda Expression?

A lambda expression is essentially an anonymous function that can be used to implement a method defined by a functional interface. It provides a clear and concise way to represent a single method interface using an expression rather than an entire class.

Syntax of Lambda Expressions

The basic syntax of a lambda expression is as follows:

```
(parameters) -> expression
```

or

```
(parameters) -> { statements; }
```

- **parameters**: The input parameters for the function.
- **->**: The arrow token that separates the parameters from the body.
- **expression**: The body of the lambda, which can be a single expression or a block of statements.

Example of Lambda Expressions

1. Basic Example

Here's a simple example of a lambda expression that takes two integers and returns their sum:

```
// Functional interface
interface MathOperation {
    int operation(int a, int b);
}

// Using lambda expression
MathOperation addition = (a, b) -> a + b;

public class LambdaExample {
    public static void main(String[] args) {
        System.out.println("Sum: " + addition.operation(5, 3)); // Output: Sum:
8
    }
}
```

2. Using Lambda with Collections

Lambda expressions are particularly useful when working with collections. For instance, you can use them to filter and sort lists easily:

```
import java.util.Arrays;
import java.util.List;

public class LambdaWithCollections {
    public static void main(String[] args) {
        List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David");

        // Using lambda to filter names that start with 'A'
        names.stream()
             .filter(name -> name.startsWith("A"))
             .forEach(System.out::println); // Output: Alice
    }
}
```

3. Using Lambda with Runnable

Lambda expressions can also be used to create instances of functional interfaces like **Runnable**:

```
public class LambdaRunnable {
    public static void main(String[] args) {
        Runnable runnable = () -> System.out.println("Running in a thread");

        Thread thread = new Thread(runnable);
        thread.start(); // Output: Running in a thread
    }
}
```

Conclusion

Lambda expressions in Java provide a streamlined way to implement functional interfaces, making code more readable and maintainable. They are particularly beneficial when working with collections and multi-threading. By adopting lambda expressions, developers can leverage the power of functional programming in Java, leading to cleaner and more efficient code.

Vamsi Mangalagiri