Motivation for Lambdas

The main motivation for introducing lambda expressions in Java is to provide a clear and concise way to represent instances of single-method interfaces (functional interfaces) using an expression. This leads to more readable and maintainable code, especially when dealing with internal iterations, event handling, or passing behavior as parameters.

Benefits of Lambdas:

1. **Conciseness**: Reduces boilerplate code by eliminating the need for anonymous inner classes.
2. **Readability**: Simplifies the code, making it easier to understand.
3. **Functional Programming**: Facilitates functional programming by treating functions as first-class citizens.
4. **Parallel Processing**: Enhances the ability to leverage multi-core architectures by using Stream API and parallel streams.

Lambda Expression Overview

A lambda expression is a concise way to represent an anonymous function (i.e., a function without a name). It has the following syntax:

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

Example:

```
// Traditional way using anonymous inner class
Runnable r1 = new Runnable() {
                @Override
                public void run() {
                    System.out.println("Hello, world!");
                }
            };

// Using lambda expression
```

```
Runnable r2 = () -> System.out.println("Hello,
world!");
```

Key Points:

1. **Parameters**: Optional if there are no parameters or only one parameter without type.
2. **Arrow Token**: `->` separates parameters and the body.
3. **Body**: Can be a single expression or a block of statements.

Lambda Expressions and Functional Interfaces

A functional interface is an interface that contains exactly one abstract method. They can have multiple default or static methods but only one abstract method. The single abstract method defines the target type for a lambda expression.

Example of a Functional Interface:

```
@FunctionalInterface
    interface MyFunctionalInterface {
        void execute();

    }
```

Using a Lambda with a Functional Interface:

```
MyFunctionalInterface myFunc = () ->
System.out.println("Executing...");

          myFunc.execute();
```

Common Functional Interfaces in Java:

1. **Predicate<T>**: Represents a boolean-valued function of one argument.
2. **Function<T, R>**: Represents a function that accepts one argument and produces a result.
3. **Consumer<T>**: Represents an operation that accepts a single input argument and returns no result.

4. **Supplier<T>**: Represents a supplier of results.