In Java, there are three types of method references that can be used as a shorthand notation for lambda expressions:

1. Reference to a Static Method: This type of method reference refers to a static method. It is denoted by the class name followed by **::** and the method name. The static method should have a compatible parameter list with the functional interface's abstract method.

Example:

// Reference to a static method

Function<Integer, Integer> square = MyClass::square;

// Equivalent lambda expression

Function<Integer, Integer> square = x -> MyClass.square(x);

Reference to an Instance Method of a Particular Object: This type of method reference refers to an instance method of a specific object. It is denoted by the object instance followed by **::** and the method name. The instance method should have a compatible parameter list with the functional interface's abstract method.

Example:

// Reference to an instance method of a particular object

List<String> names = Arrays.asList("John", "Jane", "Mike");

Consumer<String> printName = System.out::println;

names.forEach(printName);

// Equivalent lambda expression

List<String> names = Arrays.asList("John", "Jane", "Mike");

Consumer<String> printName = name -> System.out.println(name);

names.forEach(printName);

Reference to an Instance Method of an Arbitrary Object of a Particular Type: This type of method reference refers to an instance method of an arbitrary object of a specific type. It is denoted by the class name followed by **::** and the method name. The instance method should have a compatible parameter list with the functional interface's abstract method.

Example:

// Reference to an instance method of an arbitrary object of a particular type

Function<String, Integer> lengthFunc = String::length;

// Equivalent lambda expression

Function<String, Integer> lengthFunc = str -> str.length();

Refactoring lambdas into method references can make your code more concise and improve its readability. Here are a few guidelines for refactoring lambdas into method references:

1. Identify the Lambda Expression: Locate the lambda expression that you want to refactor. Look for lambda expressions that represent a simple method call or invocation.
2. Determine the Method Signature: Examine the parameters and return type of the lambda expression. Ensure that the method reference you choose has the same parameter types and return type.
3. Choose the Appropriate Method Reference Type: Select the appropriate method reference type based on the target object or class.
   * Reference to a Static Method: If the lambda expression calls a static method, refactor it into a reference to a static method.
   * Reference to an Instance Method of a Particular Object: If the lambda expression calls an instance method on a specific object, refactor it into a reference to that instance method.
   * Reference to an Instance Method of an Arbitrary Object of a Particular Type: If the lambda expression calls an instance method on an object of a specific type, refactor it into a reference to that instance method.
4. Refactor the Lambda Expression: Replace the lambda expression with the appropriate method reference, using the syntax for the chosen method reference type.

Here's an example that demonstrates the refactoring of a lambda expression into a method reference:

import java.util.function.Predicate;

public class RefactorLambdaExample {

public static void main(String[] args) {

// Lambda expression

Predicate<String> isNotEmptyLambda = str -> !str.isEmpty();

boolean result1 = isNotEmptyLambda.test("Hello");

System.out.println("Result 1: " + result1);

// Refactor into method reference

Predicate<String> isNotEmptyMethodRef = String::isEmpty;

boolean result2 = isNotEmptyMethodRef.test("Hello");

System.out.println("Result 2: " + result2);

}

}