10.14-Lamda Expression with return type

Introduction to Lambda Expression with Return Type

A **lambda expression** can also return values. When using lambda expressions with methods that have a return type, the syntax can be optimized to reduce code significantly.

Below, we will explore how a method with two parameters and a return type is implemented first using an **anonymous inner class** and then optimized using a **lambda expression.**

Example: Anonymous Inner Class with Return Type

```
interface A {
    int add(int i, int j);
}
public class MyClass {
  public static void main(String args[]) {
    A obj = new A() {
            public int add(int i, int j) {
                return i + j;
            }
        };
        int result = obj.add(5, 4);
        System.out.println("The addition is: " +
result);
}
```

Output:

```
The addition is: 9
```

Explanation:

- In the above example, we are using an **anonymous inner class** to implement the add() method of the A interface.
- The add() method takes two integer parameters (i and j) and returns their sum.
- This approach works, but the code is somewhat **verbose.**

Optimized Example: Lambda Expression with Return Type

Now, we will optimize the same implementation using a lambda expression.

```
interface A {
    int add(int i, int j);
}
public class MyClass {
  public static void main(String args[]) {
   A obj = (i, j) -> i + j;
        int result = obj.add(5, 4);
        System.out.println("The addition is: " +
result);
}
```

Output:

```
The addition is: 9
```

Explanation:

- The **lambda expression** (i, j) -> i + j directly implements the add() method.
- No need for the return keyword, as the expression itself acts as the return value.

- We don't need to specify the data type of the parameters (i and j) because they are inferred automatically.
- This version of the code is significantly shorter and more readable compared to the anonymous inner class implementation.

Key Points to Remember About Lambda Expressions with Return Type:

1. **Return Type Omission**:
    - If the lambda body contains a single return expression, the return keyword can be omitted. The result of the expression is automatically returned.
2. **Parameter Type Omission**:
    - If the method parameters have the same type, we can omit the parameter type in the lambda expression. For example, both i and j are integers, so their types are inferred automatically.

Usage of Lambda Expressions in Collections

Lambda expressions can also be used as parameters when working with collections. They are particularly useful in functional operations like **filtering**, **mapping**, and **sorting**.

Important Points to Remember:

- **Lambda expressions** can only be used with **functional interfaces**. A functional interface has exactly one abstract method. If an interface has more than one method, using a lambda expression would lead to ambiguity, as it wouldn't be clear which method the lambda is implementing.
- Lambda expressions **simplify code** by reducing the need for boilerplate code, especially when implementing interfaces with single methods.
- **Type Inference**: Java automatically infers parameter types in lambda expressions, making the code more concise.

Additional Points to Consider:

- **Code Readability**: While lambda expressions reduce the amount of code, it's essential to use them wisely. Overusing lambda expressions in complex logic might make the code harder to understand.
- **Functional Programming**: Lambda expressions are part of the broader functional programming capabilities introduced in Java 8. They work seamlessly with **streams**, **functional interfaces**, and **method references**.