10.13-lamda Expression

Introduction to Lambda Expression

Lambda expressions provide a clear and concise way to represent the implementation of a **functional interface**. A functional interface is one that contains only a single abstract method, and with lambda expressions, we don't have to implement the method in a traditional way. Instead, we simply write the logic of the method, significantly reducing code.

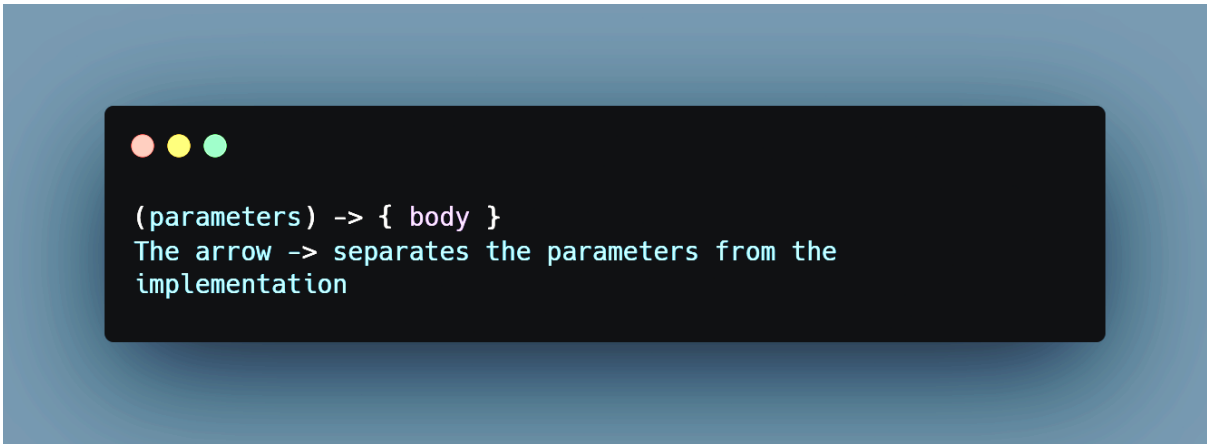
Lambda expressions were introduced in **Java SE 8** and are designed to make coding more efficient by reducing boilerplate code, especially when working with interfaces that have only one abstract method.

Functional Interface and Lambda Expression

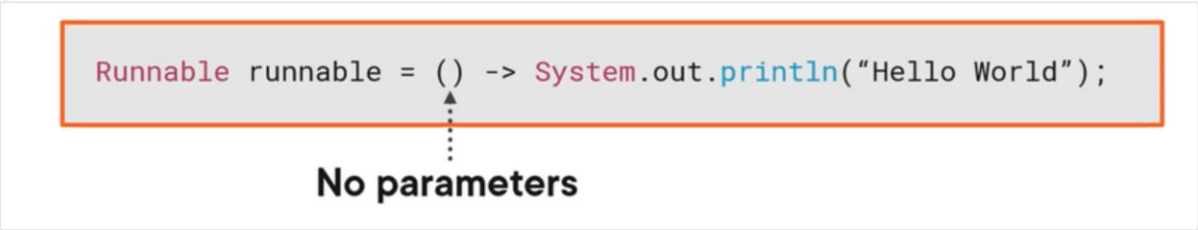
- A **functional interface** is an interface with a single abstract method, and lambda expressions allow us to provide a clean implementation of this method.
- The functional interface can be implemented using an anonymous inner class, but **lambda expressions** simplify this further.

Syntax of a Lambda Expression

Lambda expressions use the syntax:



Example of Lambda Expression with Zero Parameters:



Code example:

```
interface A {
    void show();
}

public class Demo {
    public static void main(String[] args)
{       A obj = () -> {
            System.out.println("in show");
        };
        obj.show();
    }
}
```

Output:

```
in show
```

Explanation:
In this example, we implemented the show() method of the A interface using a lambda expression. No need for an anonymous inner class or separate class for the implementation—just the logic within the lambda expression.

Why Use Lambda Expressions?

1. **Implementation of Functional Interface**: Lambda expressions provide a quick and readable way to implement functional interfaces.
2. **Less Coding**: They reduce the lines of code, especially for single-method interfaces.
3. **Syntactical Sugar**: This reduction in code and improved syntax readability is often referred to as "syntactical sugar," making code easier to read and maintain.

Lambda Expression with Parameters

When an interface method accepts parameters, lambda expressions can still be used to implement them easily.

Example of Lambda Expression with One Parameter:

```
interface A {
    void show(int i);
}

public class Demo {
    public static void main(String[] args) {
        A obj =  i -> System.out.println("in show: " +
i);     obj.show(5);
    }
}
```

Output:

```
in show: 5
```

Explanation:
In this example, we implemented the show(int i) method using a lambda expression that accepts an integer parameter i. Notice that the type of the parameter (int) is included before the parameter. If the method has only one parameter, we don't need parentheses around the parameter, and the type can also be omitted.

Simplified Lambda Expression With Two Parentheses:

```
interface A {
    void show(int i, int j);
}

public class Demo {
  public static void main(String args[]) {
    A obj = (i,j) -> System.out.println("in show: " + i +","+
j);     obj.show(5,6);

  }
}
```

Output:

```
in show: 5,6
```

Explanation:
Here, the two parameters type int and parentheses around the parameter i and j are necessary. This no parenthesis is allowed when there is only one parameter, making the code more concise.

Key Features of Lambda Expressions

- **No need to instantiate interfaces**: The compiler automatically handles the instantiation, making the code more readable.
- **Simplicity**: Instead of writing a full method implementation, the developer just focuses on the logic.
- **Efficiency**: Lambda expressions work efficiently with functional interfaces, reducing the need for boilerplate code.

FAQ's

What are Lambda Expressions in Java?

Lambda expressions are a short block of code that:

- Accept inputs as parameters.
- Return a result or perform an action without returning anything.

Is It Good to Use Lambda Expressions in Java?

Yes, lambda expressions improve code readability, reduce boilerplate, and make it easier to interact with functional interfaces, streams, and APIs introduced in Java 8 and later.

What Are the Drawbacks of Lambda Expressions?

The main limitation is that lambda expressions can only be used with functional interfaces (i.e., interfaces that have only one abstract method).