**Java  Lambda Expressions:**

Java, lambda expressions are a way to represent a concise block of code that can be treated as an object. Lambda expressions were introduced in Java 8 and are used to implement functional interfaces, which are interfaces with a single abstract method (SAM). Lambda expressions provide a more concise way to write anonymous classes for implementing these interfaces.

Here's the basic syntax of a lambda expression:

```java
(parameter_list) -> { body }
```

- `parameter_list`: This is a comma-separated list of parameters (if any) that the lambda expression can take.
- `->`: This is called the lambda operator, which separates the parameter list from the body of the lambda expression.
- `{ body }`: This is the code block that contains the implementation of the lambda expression.

Now, let's see two examples of using lambda expressions in Java.

### Example 1: Simple Lambda Expression

In this example, we'll create a lambda expression that takes two integers and returns their sum. We'll use the `FunctionalInterface` `BinaryOperator` to represent a binary operation.

```java
import java.util.function.BinaryOperator;

public class LambdaExample1 {
    public static void main(String[] args) {
        // Lambda expression to add two integers
        BinaryOperator<Integer> add = (a, b) -> a + b;

        int result = add.apply(5, 3);
        System.out.println("Result: " + result); // Output: Result: 8
    }
}
```

In this code, we define a lambda expression `(a, b) -> a + b` that takes two integers `a` and `b` as parameters and returns their sum. We then use this lambda expression with the `apply` method to add two integers.

### Example 2: Lambda Expression with a List

In this example, we'll use a lambda expression to perform an operation on each element of a list.

```java
import java.util.ArrayList;
import java.util.List;

public class LambdaExample2 {
    public static void main(String[] args) {
        List<String> languages = new ArrayList<>();
        languages.add("Java");
        languages.add("Python");
        languages.add("C++");

        // Lambda expression to print each element of the list
        languages.forEach(language -> System.out.println(language));
    }
}
```

In this code, we have a list of programming languages, and we use a lambda expression `language -> System.out.println(language)` with the `forEach` method to print each element of the list.

Lambda expressions provide a more concise and expressive way to work with functional interfaces and perform operations on collections or data. They are a fundamental feature of modern Java programming, especially in functional programming and stream processing.