JAVA Programming

Course Instructor: **Dr. N Nandini Devi**

School of Computer Science

UPES Dehradun

TOPICs to be discussed

- Lambda Expressions in Java
- Functional Interfaces
- Some Predefined Functional Interfaces
- Benefits of Lambda Expressions
- Limitations of Lambda Expressions
- Comparison with Anonymous Inner Class

Let’s START …!!!

Lambda Expressions

- Lambda expressions were introduced in **Java 8** to bring functional programming features into **Java**.
- They allow you to write concise and readable code by expressing instances of **functional interfaces** in a single line.

What is a Lambda Expression?

- A lambda expression is essentially an anonymous function—a block of code without a name that can be passed around and executed.
- It provides a clear and concise way to implement a **single abstract method (SAM)** of a **functional interface**.
- **Syntax:**

```
(parameters) -> { body }
```

Lambda Expressions (Example)

➢ **Lambda Expression with a single statement**

```
interface A{
void show();
}
public  class Demo{
         public static void main(String[] args){
                  A obj = new A(){
                  public void show(){

System.out.println("Hello World!!");
                  }
         }
}
```

```
interface A{
void show();
}
public  class Demo{
         public static void main(String[] args){
                  A obj = ()->      {

System.out.println("Hello World!!");
                  };
         }
}
```

Lambda Expressions (Example)

- **Lambda Expression with a multiple statements**

```
Comparator<Integer> comparator = (a, b) -> {
    if (a > b) return 1;
    else if (a < b) return -1;
    else return 0;
};
```

- **Lambda Expression with parameters**

```
interface MathOperation {
    int operation(int a, int b);
}

MathOperation addition = (a, b) -> a + b;
                  //Usage
System.out.println(addition.operation(5, 3));
```

Functional Interface

- A **functional interface** is an interface with exactly one abstract method. Examples include:
  - **Runnable** (method: run())
  - **Comparator** (method: compare())
  - **Callable** (method: call())

- **Custom functional interfaces** can be created using the `@FunctionalInterface` annotation.

```
@FunctionalInterface
interface Greeting {
    void sayHello(String name);
}

Greeting greeting = (name) -> System.out.println("Hello, " + name);
greeting.sayHello("Alice");
```

Output:

```
Hello Alice
```

Predefined Functional Interfaces in java.util.function

- **Predicate<T>: Represents a condition (method: test(), returns boolean)**

```
Predicate<Integer> isEven = (n) -> n % 2 == 0;
System.out.println(isEven.test(4));
```

Output:

```
true
```

- **Function<T, R>: Transforms input of type T to output of type R. (method: apply())**

```
Function<String, Integer> length = (s) -> s.length();
System.out.println(length.apply("Lambda"));
```

Output:

```
6
```

- **Consumer<T>: Performs an operation on a given input. (method: accept())**

```
Consumer<String> print = (s) -> System.out.println(s);
print.accept("Hello, World!");
```

Output:

```
Hello, World!
```

Predefined Functional Interfaces in java.util.function

- **Supplier<T>: Provides a result of type T without any input. (method: get())**

```
Supplier<Double> randomValue = () -> Math.random();

System.out.println(randomValue.get());
```

Output:

A random double value

- **BiFunction<T, U, R>: Takes two inputs and produces a result. (method: apply())**

```
BiFunction<Integer, Integer, String> sumDescription =
                                                       (a, b) -
> "Sum is " + (a + b);
System.out.println(sumDescription.apply(3, 5));
```

Output:

Sum is 8

Benefits of Lambda Expressions

- **Conciseness:**

  Simplifies the implementation of **functional interfaces**.

- **Readability:**

  Reduces boilerplate code, making code more readable.

- **Less Overhead:**

  No need for anonymous inner classes.

- **Improved Functional Programming:**

  Works seamlessly with **Java Stream API** for data processing.

Limitations of Lambda Expressions

- **Limited to Functional Interfaces:**

  Works only with interfaces having one abstract method.

- **Code Readability for Complex Logic:**

  Overusing lambdas with complex logic can make the code harder to read.

- **Debugging:**

  Tracing errors in lambda expressions can be more challenging than traditional methods.

Comparison with Anonymous Inner Class

- Lambda expressions are often a concise replacement for anonymous inner classes:

```
Runnable r = new Runnable() {
    public void run() {
        System.out.println("Running with anonymous class");
    }
};
```

```
Runnable r = () -> System.out.println("Running with Lambda");
```

Summary

Today, we learned about

- Lambda Expressions in Java (Single statement, multiple statements, parameterized)
- Functional Interfaces
- Some Predefined Functional Interfaces (Predicate, Function, Consumer, Supplier, BiFunction)
- Benefits of Lambda Expressions
- Limitations of Lambda Expressions
- Comparison with Anonymous Inner Classes

Thank You!