Kotlin Lambda

Kotlin Lambdas and Higher-Order Functions (HoFs)

---

1. First-Class Functions

In Kotlin, functions are treated as **first-class citizens**, meaning:

1. **Assign Functions to Variables**: Functions can be stored in variables or data structures.
2. **Pass Functions as Parameters**: Functions can be passed to other functions.
3. **Return Functions**: Functions can be returned from other functions.

This ability forms the foundation of functional programming in Kotlin.

---

2. Lambda Functions

A **lambda function** in Kotlin is an anonymous function, often used as an expression to simplify code. It's a concise way to pass behavior as data.

Basic Syntax of a Lambda Function

```
val lambdaName: (InputType) -> ReturnType = { parameters -> body }
```

- `parameters`: Variables passed to the lambda.
- `body`: The code executed when the lambda is invoked.

---

3. Examples of Lambdas

Example 1: Basic Lambda for Summation

A simple lambda that calculates the sum of two integers:

```
fun main() {
    val sum = { x: Int, y: Int -> x + y }  // Lambda function
    println(sum(3, 99))  // Output: 102
}
```

- **Parameters**: `x: Int`, `y: Int`.
- **Body**: `x + y`.

Example 2: Lambda with a Collection

A lambda function used to find the maximum property value in a list of objects:

```
class Student(val ageNow: Int, val graduationYearNow: Int) {
    val age: Int = ageNow
    val gradYear = graduationYearNow
}

fun main() {
    val students: List<Student> = listOf(
        Student(18, 2025),
        Student(20, 2023),
        Student(21, 2022)
    )

    // Find student with the maximum age
    println(students.maxByOrNull { s -> s.age }?.age)  // Output: 21

    // Find student with the latest graduation year
    println(students.maxByOrNull { s -> s.gradYear }?.gradYear)  // Output:
2025
}
```

Explanation:

- The lambda `{ s -> s.age }` specifies the property to compare in `maxByOrNull`.
- If the list is empty, `?.age` ensures null-safe access.

---

4. Higher-Order Functions (HoFs)

Higher-order functions (HoFs) are functions that:

1. **Accept Functions as Parameters**.

2. **Return Functions as Results**.

Example: Passing Optional Functions

```
fun main() {
    val greet: (String) -> String = { name -> "Hello, $name!" }
    println(greet("Kotlin"))  // Output: Hello, Kotlin!
}
```

Explanation:

- `greet` is a lambda that takes a `String` parameter and returns a greeting.

---

5. Null Safety in Lambdas

Using the Elvis Operator (`?:`)

The **Elvis operator** provides a concise way to handle `null` values.

```
fun main() {
    val sum = { x: Int, y: Int -> x + y }

    val num: Int? = null
    println(num ?: -1)  // Output: -1
}
```

Explanation:

- If `num` is null, the Elvis operator returns `-1` as the default value.

Safe Call Operator (`?.`) in Lambdas

The **safe call operator (`?.`)** is used to call properties or methods on nullable objects without risking a `NullPointerException`.

```
class Simple(val arg: Int) {
    fun getArg(): Int = arg
}

fun main() {
    val simple: Simple? = null
```

```
    println(simple?.getArg() ?: "No value")  // Output: No value
}
```

---

6. Lambdas and Classes

Kotlin lambdas can interact seamlessly with class methods and properties.

Example: Combining Lambdas with Class Methods

```
class Simple(val arg: Int) {
    fun getArg(): Int = arg
}

fun main() {
    val simple1 = Simple(99)
    val simple2: Simple? = null

    println(simple1.getArg())  // Output: 99
    println(simple2?.getArg() ?: "Default Value")  // Output: Default Value
}
```

---

7. Lambdas with Kotlin Collections

Lambdas are commonly used with collection operations like `map`, `filter`, and `reduce`.

Example:

```
fun main() {
    val numbers = listOf(1, 2, 3, 4, 5)

    // Multiply each number by 2
    val doubled = numbers.map { it * 2 }
    println(doubled)  // Output: [2, 4, 6, 8, 10]

    // Filter even numbers
    val evens = numbers.filter { it % 2 == 0 }
    println(evens)  // Output: [2, 4]
}
```

---

8. Key Points

1. **First-Class Functions**: Functions can be treated as objects, passed around, and returned.
2. **Lambda Syntax**:
   - `{ x: Int, y: Int -> x + y }` defines parameters and the body.
3. **Higher-Order Functions (HoFs)**:
   - HoFs accept or return functions, enabling dynamic and reusable logic.
4. **Null Safety**:
   - Use `?.` for safe calls.
   - Use `?:` to provide a default value when encountering `null`.
5. **Collections**:
   - Lambdas simplify data manipulation with operations like `map`, `filter`, and `reduce`.