Funciones lambda

Funciones de orden superior

Kotlin es un lenguaje orientado a objetos pero introduce características existentes en los lenguajes funcionales que nos permiten crear un código más claro y expresivo.

Una de las características del paradigma de la programación funcional son las funciones de orden superior.

Las funciones de orden superior son funciones que pueden recibir como parámetros otras funciones y/o devolverlas como resultados.

Veremos una serie de ejemplos muy sencillos para ver como Kotlin implementa el concepto de funciones de orden superior y a medida que avancemos en el curso podremos ver las ventajas de este paradigma.

Ejercicio

Definir una función de orden superior llamada operar. Llegan como parámetro dos enteros y una función. En el bloque de la función llamar a la función que llega como parámetro y enviar los dos primeros parámetros.

La función retorna un entero.

```
fun operar(v1: Int, v2: Int, fn: (Int, Int) -> Int) : Int {
    return fn(v1, v2)
}

fun sumar(x1: Int, x2: Int) = x1 + x2

fun restar(x1: Int, x2: Int) = x1 - x2

fun multiplicar(x1: Int, x2: Int) = x1 * x2

fun dividir(x1: Int, x2: Int) = x1 / x2

fun main(parametro: Array<String>) {
    val resu1 = operar(10, 5, ::sumar)
    println("La suma de 10 y 5 es $resu1")
    val resu2 = operar(5, 2, ::sumar)
    println("La suma de 5 y 2 es $resu2")
    println("La resta de 100 y 40 es ${operar(100, 40, ::restar)}")
    println("El producto entre 5 y 20 es ${operar(5, 20,
::multiplicar)}")
    println("La división entre 10 y 5 es ${operar(10, 5, ::dividir)}")
}
```

Funciones lambda

Una expresión lambda es cuando enviamos a una función de orden superior directamente una función anónima.

Funciones lambda

Es más común enviar una expresión lambda en lugar de enviar la referencia de una función como vimos en el concepto anterior.

Ejemplo

Definir una función de orden superior llamada operar. Llegan como parámetro dos enteros y una función. En el bloque de la función llamar a la función que llega como parámetro y enviar los dos primeros parámetros.

Desde la función main llamar a operar y enviar distintas expresiones lambdas que permitan sumar, restar y elevar el primer valor al segundo.

```
fun operar(v1: Int, v2: Int, fn: (Int, Int) -> Int) : Int{
    return fn(v1, v2)
}

fun main(parametro: Array<String>) {
    val suma = operar(2, 3, {x, y -> x + y})
    println(suma)
    val resta = operar(12, 2, {x, y -> x - y})
    println(resta)
    var elevarCuarta = operar(2, 4, {x, y ->
        var valor = 1
        for(i in 1..y)
            valor = valor * x
        valor
    })
    println(elevarCuarta)
}
```

Ejemplo

Confeccionar una función de orden superior que reciba un vector de enteros y una función con un parámetro de tipo **Int** y que retorne un **Boolean**.

La función debe analizar cada elemento del vector llamando a la función que recibe como parámetro, si retorna un true se pasa a mostrar el elemento.

En la función **main** definir un vector de enteros de 10 elementos y almacenar valores aleatorios comprendidos entre 0 y 99.

Imprimir del vector

- Los valores múltiplos de 2
- Los valores múltiplos de 3 o de 5
- Los valores mayores o iguales a 50
- Los valores comprendidos entre 1 y 10, 20 y 30, 90 y 95

Funciones lambda

```
fun imprimirSi(vector: IntArray, fn:(Int) -> Boolean) {
    for(elemento in vector)
        if (fn(elemento))
            print("$elemento ")
    println();
}

fun main(parametro: Array<String>) {
    val vector1 = IntArray(10)
    for(i in vector1.indices)
        vector1[i] = ((Math.random() * 100)).toInt()
    println("Imprimir los valores múltiplos de 2")
    imprimirSi(vector1) {x -> x % 2 == 0}
    println("Imprimir los valores múltiplos de 3 o de 5")
    imprimirSi(vector1) {x -> x % 3 == 0 || x % 5 ==0}
    println("Imprimir los valores mayores o iguales a 50")
    imprimirSi(vector1) {x -> x >= 50}
    println("Imprimir los valores comprendidos entre 1 y 10, 20 y 30,
90 y 95")
    imprimirSi(vector1) {x -> when(x) {
        in 1..10 -> true
        in 20..30 -> true
        in 90..95 -> true
        else -> false
    }}
    println("Imprimir todos los valores")
    imprimirSi(vector1) {x -> true}
}
```