Using Higher Order Functions in Kotlin

Kevin Jones

@kevinrjones www.rocksolidknowledge.com

Higher Order Function

A function that takes another function as an argument

Declaring Functions

```
val action = { println("Hello, World") }

val calc = { x: Int, y: Int -> x*y }
```

Function Types

$$\overbrace{\texttt{(Int, Int)}}^{\text{parameters}}\ \texttt{->}\ \overbrace{\texttt{Int}}^{\text{return}}$$

Declaring Functions

```
val action: () -> Unit = { println("Hello, World") }

val calc: (Int, Int) -> Int = { x, y -> x*y }
```

Calling Functions

```
fun <T> first(items: List<T>, predicate: (T) -> Boolean) {
    for(item in items) {
        if(predicate (item)) return item
    }
    throw ...
}
```

Inlining Functions

Lambdas map to anonymous classes

Extra class and method created each time

This is expensive

Enter inlining

Demo

Inlining Functions

Not every function can be inlined

If Lambda is used directly then can inline

Lambda cannot be stored in a variable for later use

Kotlin collection operations are inlined

- map, filter etc

The same operations on sequences are not

Summary

Higher-order functions give us great flexibility

However there can be a performance overhead

Use inlining to get around this

Not all calls can be inlined