

04

04-4. 인라인 함수

Functional Programming

인라인 함수

인라인 함수란?

- 고차함수의 런타임시 성능이슈 문제

```
fun hoFunTest(argFun: (x1: Int, x2: Int) -> Int){  
    argFun(10, 20)  
}  
fun main(args: Array<String>) {  
    val result = hoFunTest { x1, x2 -> x1 + x2 }  
}
```

```
public static final void hoFunTest(@NotNull Function2 argFun) {  
    Intrinsics.checkNotNull(argFun, "argFun");  
    argFun.invoke(Integer.valueOf(10), Integer.valueOf(20));  
}  
  
public static final void main(@NotNull String[] args) {  
    Intrinsics.checkNotNull(args, "args");  
    hoFunTest((Function2)null.INSTANCE);  
}
```

인라인 함수

인라인 함수란?

- inline 함수는 함수 선언 앞에 inline 예약어를 추가한 함수
- 컴파일 단계에서 정적으로 포함

```
inline fun hoFunTest(argFun: (x1: Int, x2: Int) -> Int){  
    argFun(10, 20)  
}  
fun main(args: Array<String>) {  
    hoFunTest { x1, x2 -> x1 + x2 }  
}
```

```
public static final void main(@NotNull String[] args) {  
    //.....  
    int x2 = 20;  
    int x1 = 10;  
    int var10000 = x1 + x2;  
}
```

인라인 함수

노인라인

- `noinline` 예약어를 이용하여 `inline`에 적용되는 람다함수와 적용되지 않는 람다함수를 구분

```
inline fun inlineTest(argFun1: (x: Int) -> Int, argFun2: (x: Int) -> Int){  
    argFun1(10)  
    argFun2(10)  
}  
fun main(args: Array<String>) {  
    inlineTest({it * 10}, {it * 20})  
}
```

```
public static final void main(@NotNull String[] args) {  
    Intrinsics.checkNotNull(args, "args");  
    int it = 10;  
    int var10000 = it * 10;  
    it = 10;  
    var10000 = it * 20;  
}
```

인라인 함수

노인라인

- `noinline` 예약어를 이용하여 `inline`에 포함하지 않아도 되는 함수타입의 매개변수를 명시적으로 선언

```
inline fun inlineTest(argFun1: (x: Int) -> Int, noinline argFun2: (x: Int) -> Int){  
    argFun1(10)  
    argFun2(10)  
}  
fun main(args: Array<String>) {  
    inlineTest({it * 10}, {it * 20})  
}
```

```
public static final void main(@NotNull String[] args) {  
    Intrinsics.checkNotNull(args, "args");  
    Function1 argFun2$iv = (Function1)null.INSTANCE;  
    int it = 10;  
    int var10000 = it * 10;  
    argFun2$iv.invoke(Integer.valueOf(10));  
}
```

인라인 함수

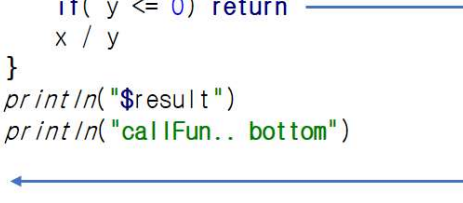
논로컬 반환

- Non-local return 이란 람다함수 내에서 람다함수를 포함하는 함수를 벗어나게 하고자 하는 기법
- 코틀린에서는 return 을 이름이 정의된 일반함수와 Anonymous Function 에서만 사용이 가능하며 람다함수에서는 사용 불가

```
fun inlineTest2(argFun: (x: Int, y: Int) -> Int): Int {  
    return argFun(10, 0)  
}
```

```
fun callFun() {  
    println("callFun.. top")  
    val result = inlineTest2 { x, y ->  
        if (y <= 0) return //error  
        x / y  
    }  
    println("$result")  
    println("callFun.. bottom")  
}
```

```
fun callFun() {  
    println("callFun.. top")  
    val result = inlineTest2 { x, y ->  
        if (y <= 0) return  
        x / y  
    }  
    println("$result")  
    println("callFun.. bottom")  
}
```



인라인 함수

논로컬 반환

- 고차함수가 inline 으로 선언되어 있다면 이 고차함수에 적용되는 람다함수 내에는 return 구문을 사용 가능

```
inline fun inlineTest2(argFun: (x: Int, y: Int) -> Int): Int{
    return argFun(10, 0)
}
fun callFun() {
    println("callFun.. top")
    val result = inlineTest2 { x, y ->
        if( y <= 0) return
        x / y
    }
    println("$result")
    println("callFun.. bottom")
}
fun main(args: Array<String>) {
    callFun()
}
```

인라인 함수

크로스 인라인

- inline 되는 고차함수라고 하더라도 대입되는 람다함수가 고차함수 내에서 다른 객체에 대입되어 사용이 된다면 return 사용에 제약

```
open class TestClass {  
    open fun some() {}  
}  
fun inlineTest3(argFun: () -> Unit){  
    val obj = object : TestClass() {  
        override fun some() = argFun()  
    }  
}
```

```
open class TestClass {  
    open fun some() {}  
}  
inline fun inlineTest3(argFun: () -> Unit){  
    val obj = object : TestClass() {  
        override fun some() = argFun()//error  
    }  
}
```


인라인 함수

크로스 인라인

- `crossinline` 은 함수가 `inline` 으로 선언되었다고 하더라도 대입되는 람다함수에 `return` 이 사용되지 않게 하기 위한 예약어

```
inline fun inlineTest3(crossinline argFun: () -> Unit){
    val obj = object : TestClass() {
        override fun some() = argFun()
    }
}

fun crossInlineTest(){
    println("aaa")
    inlineTest3 {
        return //error
    }
}
```

인라인 함수

라벨로 반환

- 람다에서 return 을 이용하여 자신이 대입되는 고차함수의 수행을 끝내야 하는 경우

```
val array = arrayOf(1,-1, 2)
fun arrayLoop() {
    println("loop top..")
    array.forEach {
        println(it)
    }
    println("loop bottom..")
}
arrayLoop()
```

실행결과

loop top..

1

-1

2

loop bottom..

인라인 함수

라벨로 반환

- 람다함수에 return 구문을 추가

```
val array = arrayOf(1, -1, 2)
fun arrayLoop() {
    println("loop top..")
    array.forEach {
        if(it < 0) return
        println(it)
    }
    println("loop bottom..")
}
arrayLoop()
```

실행결과

loop top..

1

인라인 함수

라벨로 반환

- 데이터를 출력하는 람다함수만 벗어나게 하고 싶은 경우

```
val array = arrayOf(1, -1, 2)
fun arrayLoop() {
    println("loop top..")
    array.forEach aaa@{
        if(it < 0) return@aaa
        println(it)
    }
    println("loop bottom..")
}
arrayLoop()
```

실행결과

loop top..

1

2

loop bottom..

인라인 함수

라벨로 반환

- 자동으로 고차함수에 label이 추가되고 그 이름을 이용 가능

```
val array = arrayOf(1, -1, 2)
fun arrayLoop() {
    println("loop top..")
    array.forEach {
        if(it < 0) return@forEach
        println(it)
    }
    println("loop bottom..")
}
arrayLoop()
```

실행결과

loop top..

1

2

loop bottom..

인라인 함수

라벨로 반환

```
inline fun hoTest(argFun: (String)->Unit){  
    argFun("hello")  
    argFun("kim")  
    argFun("kkang")  
}  
fun labelTest(){  
    println("test top....")  
    hoTest {  
        if(it.length < 4) return@hoTest  
        println(it)  
    }  
    println("test bottom....")  
}
```

실행결과

test top....

hello

kkang

test bottom....



감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.



윌리엄 셰익스피어
William Shakespeare