10 Interesting things in Kotlin

@im_Ntiwari

#1
Extension functions

- Works on decorator pattern
- Helps extends the functionality of the class
- No need to inherit the property
- Also works on property of the class

```kotlin
fun makeFirstCharacterCaps(s: String): String {
    return s.substring(0, 1).toUpperCase() +
            s.substring(1)
}
```

```kotlin
println(makeFirstCharacterCaps("kotlin"))
```

Kotlin

```kotlin
fun String.makeFirstCharacterCaps(): String {
   return this.substring(0, 1).toUpperCase() +
           this.substring(1)
}
```

```kotlin
println("kotlin".makeFirstCharacterCaps())
```

Kotlin

#2
Higher order functions and Lambdas

- Takes function as a parameters
- Can return result as function
- Can store result in a function

```
fun fetchEvenList(list: List<Int>): List<Int> {
    return list.filter { it % 2 == 0 }.toList()
}
```

```
fetchEvenList(mutableListOf(1, 2, 3, 4, 5, 6)).forEach {
    print(it)
}
```

```
2 4 6
```

```
fun fetchEvenList(list: List<Int>, eventList: (list: List<Int>) -> Unit) {
    val result = list.filter { it % 2 == 1 }.toList()
    eventList(result)
}
```

```
fetchEvenList(mutableListOf(1, 2, 3, 4, 5, 6)) { it ->
    it.forEach { print(it) }
}
```

```
2 4 6
```

- Lambda expression are **function literal**
- Surrounded by curly braces { }

```
fun fetchEvenList(list: List<Int>): List<Int> {
    return list.filter { it % 2 == 0 }.toList()
}
```

Let’s combine all of them

```
fun List<Int>.fetchEventList(eventList: (list: List<Int>) -> Unit) {
    eventList(this.filter { it % 2 == 0 }.toList())
}
```

#3
Inline functions

- Body of the function get in-lined with code
- Reduces memory overhead
- Works well for smaller and repeated function body
- Needs **inline** keyword on the function

```
fun String.capsFirsCharacter(caps: (result: String) -> Unit){
    caps(this.substring(0, 1).toUpperCase())
}
```

```
"kotlin".capsFirsCharacter {
    println(it)
}
```

```
public final class KotlinPresentationKt {
   public static final void main(@NotNull String[] args) {
      capsFirsCharacter("kotlin", (Function1)null.INSTANCE);
    }

   public static final void capsFirsCharacter(@NotNull String $this$capsFirsCharacter, @NotNull Function1 
caps) {
      ...
      String var10001 = $this$capsFirsCharacter.substring(var3, var4);
        ...
   }
```

```
inline fun String.capsFirsCharacter(caps: (result: String) -> Unit){
    caps(this.substring(0, 1).toUpperCase())
}
```

```
"kotlin".capsFirsCharacter {
    println(it)
}
```

```
public final class KotlinPresentationKt {
    public static final void main(@NotNull String[] args) {
        ...
        String var10000 = $this$capsFirsCharacter$iv.substring(var4, var5);
        ...
    }
}
```

#4
Delegates

- Transfer the request to helpers to serve

- Kotlin does using **by** keyword on classes + properties

```
interface Base {
    fun print()
}


class Concrete(val s: String): Base {
    override fun print() {
        println(s)
    }
}


class Delegate(d: Base): Base by d
```

```
Delegate(Concrete("kotlin")).print()
```

```
kotlin
```

```
val tvMobile by lazy {
    findViewById(R.id.mobile_tv) as TextView
}
```

```
var name : String by notNull()
```

#5
Destructuring declaration

```
data class Customer(val name: String, val mobile: String)

val (name, mobile) = Customer("Kotlin", "70210***64")

println(name)                    Kotlin
println(mobile)              70210***64
```

```
val (name) = Customer("Kotlin", "70210***64")
println(name)                    Kotlin
```

```
val customer = Customer("Kotlin", "70210***64")

println(customer.component1())              Kotlin
println(customer.component2())               70210***64
```

#6
Collections

- Kotlin collections are very comprehensive
- Some common includes list, set, and maps
- Located in **kotlin.collections** package

```
 val set = setOf("android", "ios", "web", "four")
 var emptySet = mutableSetOf<String>()


val map = mapOf("name" to "kotlin", "mobile" to "70210***64")
val emptyMap = mutableMapOf<String, String>()


val squared = MutableList(3) { it * it }
println(squared)                [0, 1, 4]
```

```kotlin
val platforms = listOf("android", "ios", "web", "windows")

val languages = listOf("kotlin", "swift", "react")

println(platforms.zip(languages))
```

```
[(android, kotlin), (ios, swift), (web, react)]
```

```kotlin
val platforms = listOf("android", "ios", "web", "windows")

println(platforms.associateBy { it.first().toUpperCase() })
```

```
{A=android, I=ios, W=windows}
```

```kotlin
println(platforms.associateBy(keySelector = { it.first().toUpperCase() }, valueTransform =
{ it.length < 4 }))
```

```
{A=false, I=true, W=false}
```

7
Couroutines

- Patterns to simplify the asynchronous style of code

- Help reduce overhead of long running task blocking mainthread or Main Safe

- Lightweight

- Uses dispatcher MAIN, IO,DEFAULT to identify threads

How to start ?

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)

   GlobalScope.launch(Dispatchers.Main) {
      triggerApiAndDisplay()
   }

}
```

```
suspend fun triggerApiAndDisplay(){
   val apiResult = triggerApi()
   display(apiResult)
}
```

```
// Non Blocking API
suspend fun triggerApi()=
   withContext(Dispatcher.IO) {
      ..
   }
}
```

```
# display on Main
fun display() {
    ..
}
```

#8
Scope functions

- Makes code more concise and easier to write
- Usage depends on adoption and selection by developers in project
- Like .let {}, .apply{}, .run {}, with{}, also {}

Function	Object reference	Return value	Is extension function
`let`	`it`	Lambda result	Yes
`run`	`this`	Lambda result	Yes
`run`	-	Lambda result	No: called without the context object
`with`	`this`	Lambda result	No: takes the context object as an argument.
`apply`	`this`	Context object	Yes
`also`	`it`	Context object	Yes

```
val customer = Customer("kotlin", "70210***64")
       .apply { this.name = this.name.capsFirsCharacter() }

println(customer)
```

```
Customer(name=Kotlin, mobile=70210***64)
```

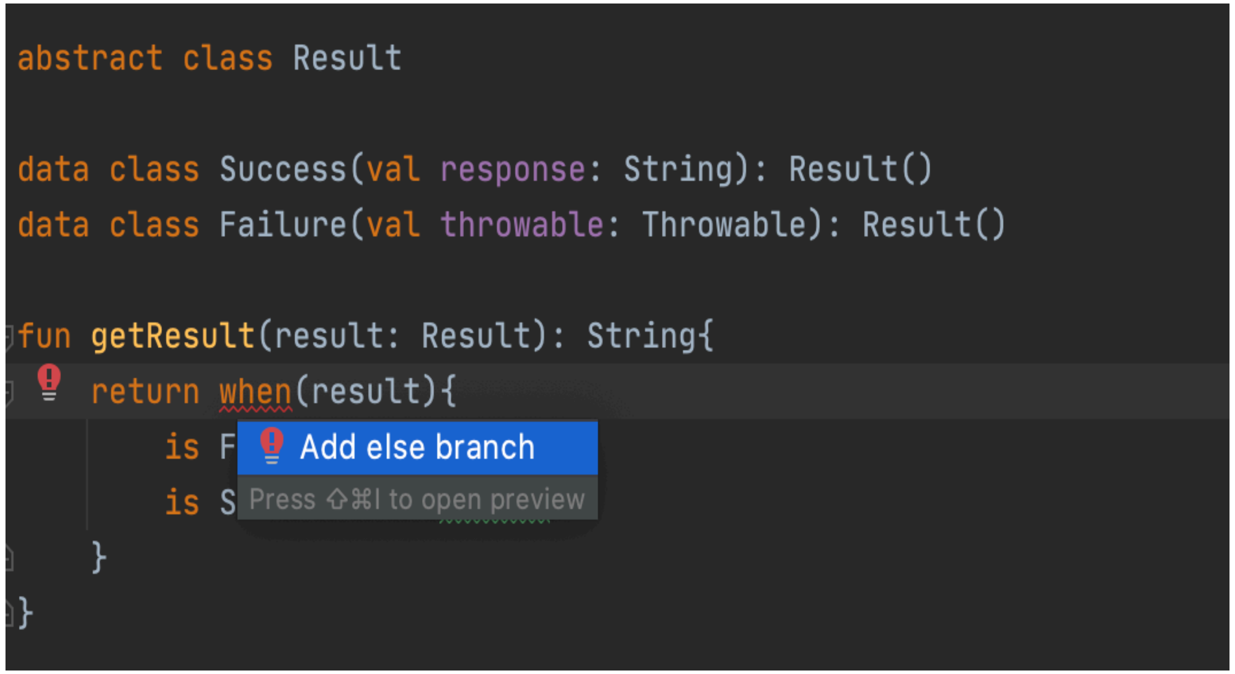
```
val customer = Customer("kotlin", "70210***64")
       .let {
          it.name = it.name.capsFirsCharacter()
          it.name
       }

println(customer)
```

```
Kotlin
```

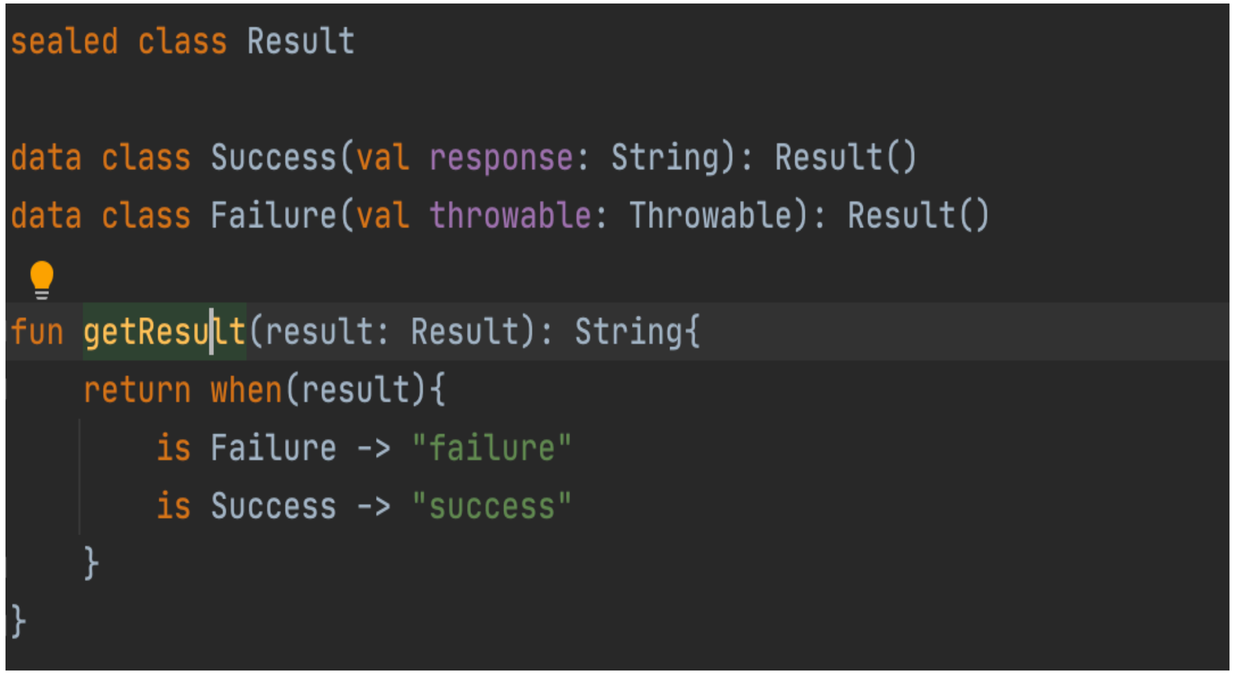
#9
Sealed Classes

- Limits class Hierarchies
- More control on inheritance of the class
- Subclasses are known at compile time
- Hierarchies in the same file

```
enum class Result(message: String) {
    SUCCESS(message = "success"),
    FAILURE(message = "failure", throwable)
}

# Not possible to add state of the FAILURE



sealed class Result {
    data class Success(val response: String) : KotlinPresentationSealed()
    class Failure(throwable: Throwable): KotlinPresentationSealed()
}

println(KotlinPresentationSealed.Success("{}"))
println(KotlinPresentationSealed.Failure(Exception("Invalid request")))


#States of result allowed
```



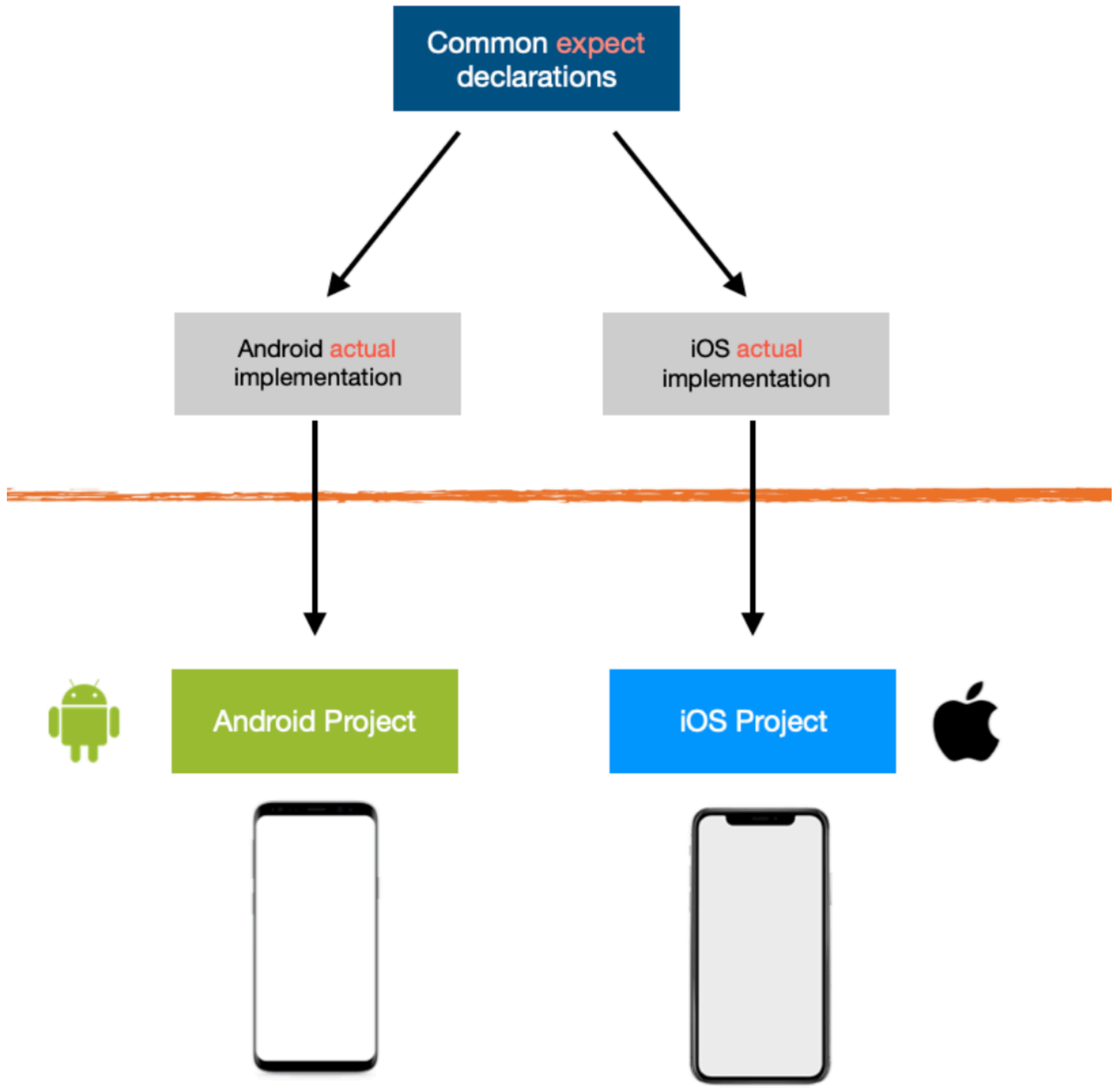
```kotlin
abstract class Result

data class Success(val response: String): Result()
data class Failure(val throwable: Throwable): Result()

fun getResult(result: Result): String{
    return when(result){
        is F
        is S
    }
}
```

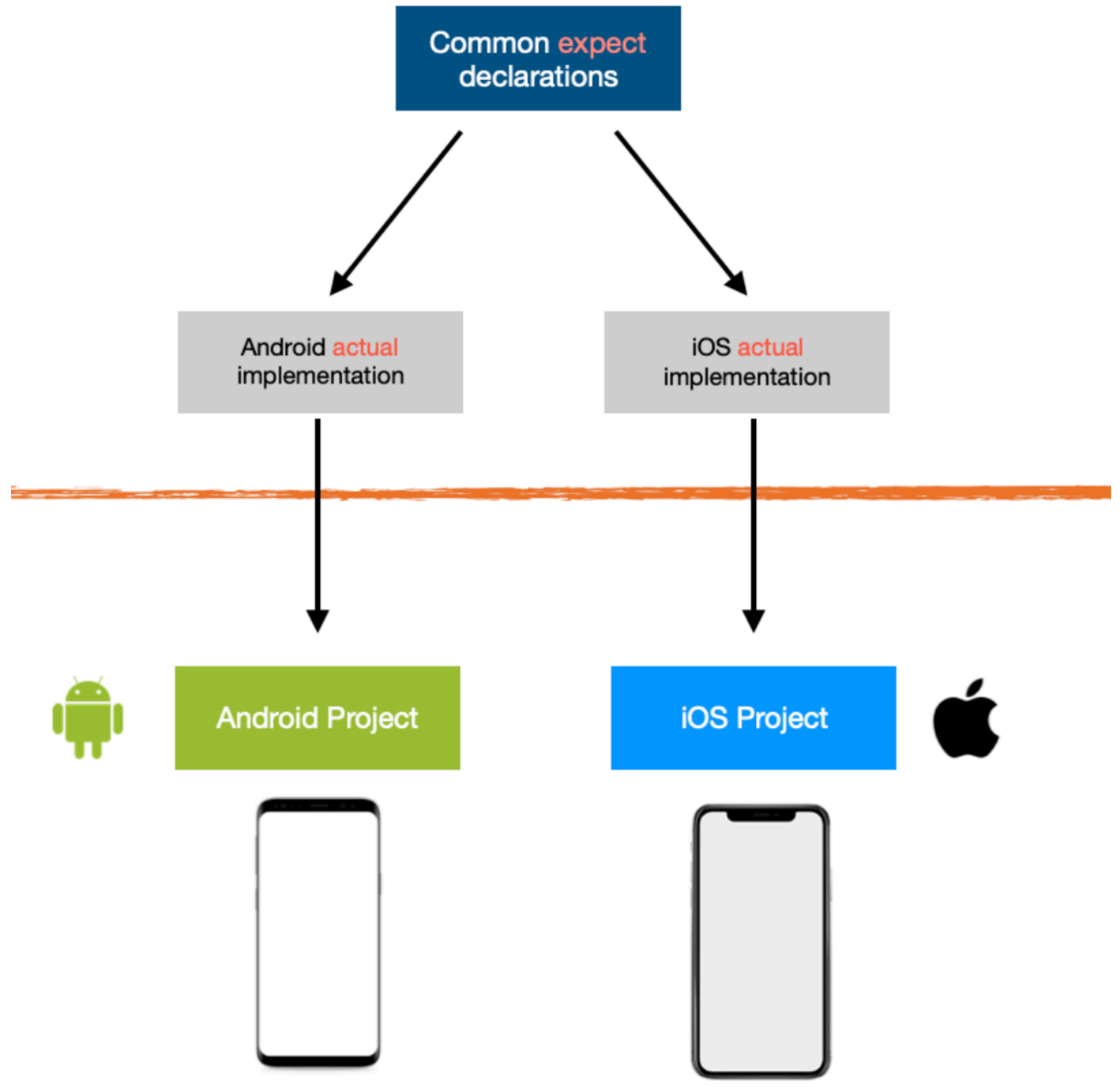


```kotlin
sealed class Result

data class Success(val response: String): Result()
data class Failure(val throwable: Throwable): Result()

fun getResult(result: Result): String{
    return when(result){
        is Failure -> "failure"
        is Success -> "success"
    }
}
```

#10
Kotlin Multi-Platform

- Share common code among platforms android, ios, web
- Task like network calls, long running coroutine task, serialization
- Single source of truth for common code


Common expect declarations
Android actual implementation
iOS actual implementation
Android Project
iOS Project


Common expect declarations
Android actual implementation
iOS actual implementation
Android Project
iOS Project

```
expect fun log(message: String)

#android

actual fun log(message: String){
    Log.d("android", message)
}


#ios
actual fun log(message: String){
    NSLog(message)
}


#jvm
actual fun log(message: String){
    println(message)
}
```

ANY QUESTIONS ?

SHOOT !!

THANK YOU