expedia group™

Kotlin programming - July 2020

Agenda

1. Why kotlin?
2. Kotlin in EG
3. Basics
4. Functions and lambdas
5. Class and Object
6. Collections
7. Coroutines
8. Putting it all together

Goals of this demo

- Introduction to broad and basic concepts of kotlin
- For prior java programmers trying to pickup kotlin
- Doesn’t cover anything deeper
- Any opinions welcome to be discussed
- Stop and ask questions anytime

Kotlin

Named after an island kotlin near St. Petersburg, Russia

Why kotlin?

- Concise
- Interoperable
- Functional programming or/and object oriented
- Tool friendly

Concise

Reduces boilerplate code

Examples

1.
```
data class Date(private val day: Int, private val month: Int,
private val year: Int)
```

2.
```
val list = List(5) {it + 1}
assertThat(list.filter { it%2 == 0 })
    .containsAll(2, 4)
```

Concise

Null safety easier

Ex:

```
var name: String
name = null; //Compilation error

val name: String? = null
println(name.length) //Compilation error
```

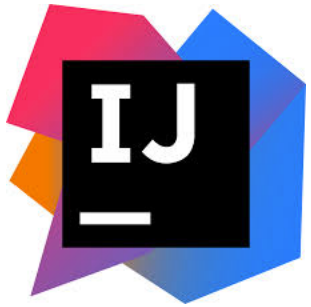
Interoperable

Code in kotlin, compile it to JVM bytecode or JavaScript

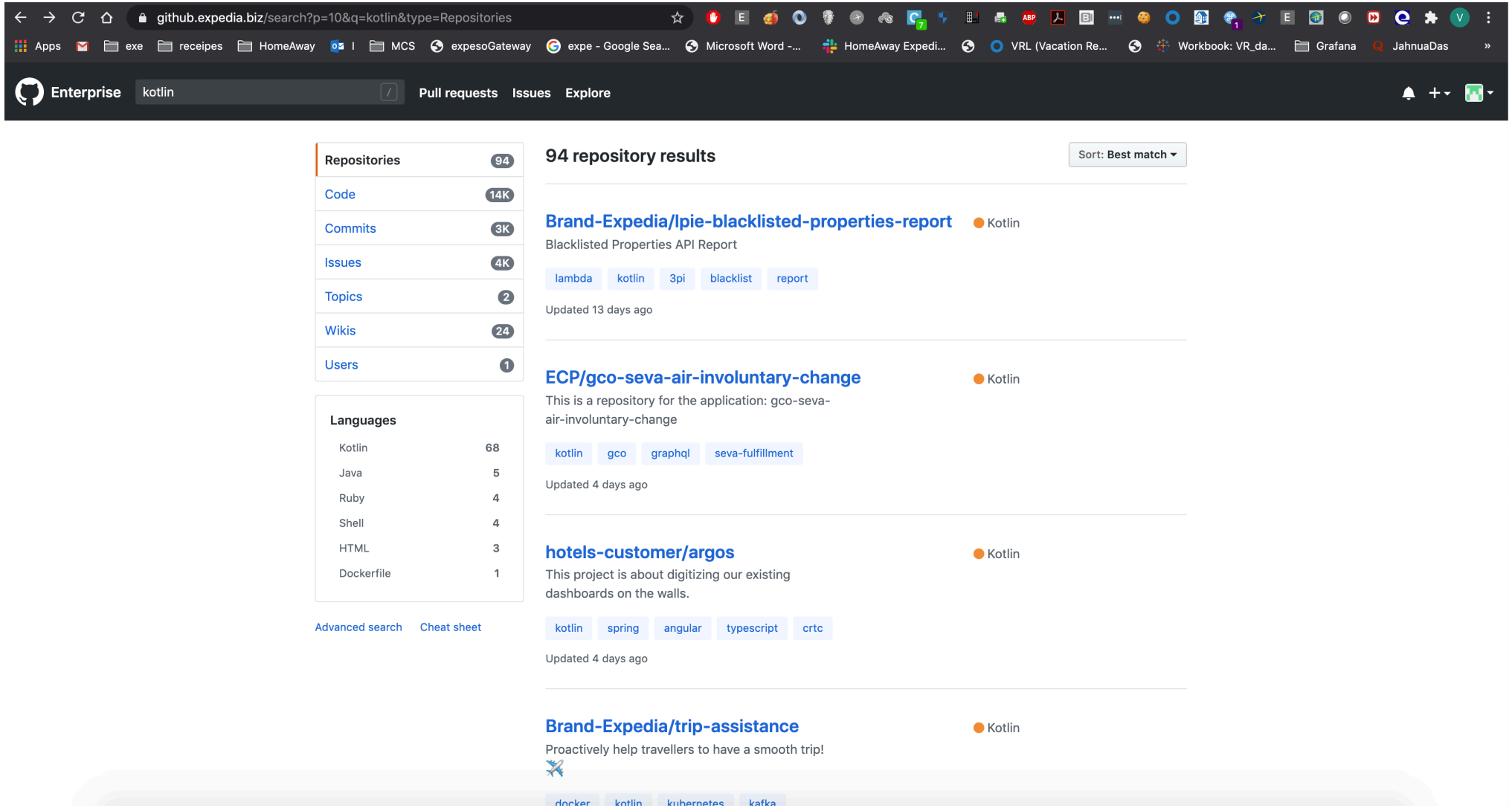
Functional programming or/and object oriented

- Kotlin natively supports higher order functions
- Can be used as pure functional programming without class
- Full-fledged Object oriented support too with classes
- Best combine both

Tool friendly



Kotlin in EG


github.expedia.biz/search?p=10&q=kotlin&type=Repositories
Apps
exe
receipes
HomeAway
MCS
expesoGateway
expe - Google Sea...
Microsoft Word -...
HomeAway Expedi...
VRL (Vacation Re...
Workbook: VR_da...
Grafana
JahnuaDas
Enterprise
kotlin
Pull requests
Issues
Explore
Repositories 94
Code 14K
Commits 3K
Issues 4K
Topics 2
Wikis 24
Users 1
Languages
Kotlin 68
Java 5
Ruby 4
Shell 4
HTML 3
Dockerfile 1
Advanced search
Cheat sheet
94 repository results
Sort: Best match
Brand-Expedia/lpie-blacklisted-properties-report
Kotlin
Blacklisted Properties API Report
lambda
kotlin
3pi
blacklist
report
Updated 13 days ago
ECP/gco-seva-air-involuntary-change
Kotlin
This is a repository for the application: gco-seva-air-involuntary-change
kotlin
gco
graphql
seva-fulfillment
Updated 4 days ago
hotels-customer/argos
Kotlin
This project is about digitizing our existing dashboards on the walls.
kotlin
spring
angular
typescript
crtc
Updated 4 days ago
Brand-Expedia/trip-assistance
Kotlin
Proactively help travellers to have a smooth trip!
docker
kotlin
kubernetes
kafka

Basics

Basic Types

- Numbers: Byte, Short, Int, Long, Float, Double
- Boolean
- Char
- String
- Arrays: Array, ByteArray, ShortArray, IntArray,

Variables

```
val language = "kotlin" //val is readonly
language = "java" //compilation error

val x: String
x = "3" //still works first assignment

var language = "kotlin" //var is mutable
language = "java"

companion object{
    const val x = 2
}
```

Variables

Nullable types and Non-Null types

- **Kotlin eliminates NullPointerException**

```
var name: String = "kotlin" // non-null by default
name = null // compilation error

var name: String? = "abc" // can be set null
name = null // ok


var name: String? = "abc" // can be set null
val l = name.length //compilation error

val l = if (name != null) name.length else -1 //null safe way

val l = name?.length //? operator - null safe navigation

val l = name?.length?: -1 //Elvis operator
```

Functions and lambdas

```
fun double(num: Int) = num * 2 //single expression

fun double(num: Int): Int = num * 2 //Compiler can infer return type

fun double(num: Int): Int{
    return num * 2
}
```

Usage: *double*(2)

Functions and lambdas

Default arguments: `fun double(num: Int = 3) = num * 2`

Usage: `double()`

Named argument: `fun sum(a: Int, b: Int) = a + b`

Usage: `sum(b = 3, a = 4)`

Varargs argument: `fun sum(vararg ints: Int) = ints.sum()`

Usage: `sum(1, 2, 3, 4)`

With generic: `fun <T> singletonList(item: T) = listOf(item)`

Functions and lambdas

Functions are first-class, stored as variables, passed as argument and returned from other higher order functions

```
fun <T> returnString(t: T, block: (T) -> String) = block.invoke(t)
fun <T> returnString(t: T, block: (T) -> String) = block(t)
```

Usage:

```
returnString(4, { it.toString() }) //lambda
returnString(4, Int::toString) //method reference
returnString(4, fun(s: Int): String { return s.toString() }) //Anonymous function
```

Functions and lambdas

Extension functions and properties

Adds functions/properties to existing classes without modifying source including jars, standard libraries

Example:

```
data class Date(val day: Int, val month: Int, val year: Int)


fun Int.beween(lower: Int, upper: Int): Boolean {
    return this.absoluteValue in (lower + 1) until upper
}

fun Date.isValid(): Boolean {
    return this.day.beween(0, 31)
            && this.month.beween(0, 13)
}
```

Functions and lambdas

Suspend functions

- **Non blocking – can be suspended and resumed later**
- **Can be invoked from coroutine**
- **Can invoke other suspended functions**
- **Basis for coroutines**

Example:

```
suspend fun callRemoteService(): String {
    delay(1000L) // assume this is time taken for remote call
    return "done"
}

var data: String
runBlocking {
    data = callRemoteService()
}
```

Co(operative)routines

Lightweight concurrency model over threads for asynchronous/non blocking execution

- **Threads are objects and have overhead of scheduling, stack maintenance when used explicitly**
- **Threads have pre-emptive scheduling and corresponds to native thread**
- **Coroutine is cooperative multitasking either running or suspended**
- **Coroutine still require a thread when running**

Co(operative)routines

Example:

```
suspend fun callRemoteService(): String {
    delay(1000L) // assume this is time taken for remote call
    return "done"
}
```

Launch with global scope: more overhead to use this scope always

```
var data: String = ""
GlobalScope.launch {
    data = callRemoteService()
}
```

Co(operative)routines

Example structured concurrency:

```
suspend fun callRemoteService(): String {
    delay(1000L) // assume this is time taken for remote call
    return "done"
}
```

Launch with specific scope:

```
launch {
    data = callRemoteService()
}
```

Co(operative)routines

Launch with specific scope and use join to wait nonblock:

```
var data: String = ""
val job = launch {
    data = callRemoteService()
}
job.join()



withTimeout(400) {
    data = callRemoteService()
}
```

Co(operative)routines

```
val one = async { callRemoteService1() }
val two = async { callRemoteService2() }
assertThat(one.await() + two.await()).isEqualTo("done1done2")
```

Co(operative)routines

Dispatchers and threads

```
launch { // context of the parent, main runBlocking coroutine
    println("main runBlocking      : I'm working in thread ${Thread.currentThread().name}")
}
launch(Dispatchers.Unconfined) { // not confined -- will work with main thread
    println("Unconfined            : I'm working in thread ${Thread.currentThread().name}")
}
launch(Dispatchers.Default) { // will get dispatched to DefaultDispatcher
    println("Default               : I'm working in thread ${Thread.currentThread().name}")
}
launch(newSingleThreadContext("MyOwnThread")) { // will get its own new thread
    println("newSingleThreadContext: I'm working in thread
${Thread.currentThread().name}")
}
```

Source: https://kotlinlang.org/docs/reference/coroutines/coroutine-context-and-dispatchers.html

Class and Object

```
data class Date(private val day: Int, private val month: Int, private val year: Int)


class Person(val firstName: String) {
    fun print() = print(firstName)
}


Person("kotlin")
Person("kotlin").firstName
Person("kotlin").print()
```

Class and Object

```
class Person(val firstName: String) {
    private lateinit var lastName: String

    constructor(firstName: String, lastName: String): this(firstName) {
        this.lastName = lastName
    }
    fun print() = println(firstName)
}
```

Class and Object

Default final class

```
class IamFinal(val behavior: String = "NoInheritence")
```

To open inheritance

```
open class Name(val name: String) {
    open fun print() {
        println(name)
    }
}

class FullName(private val firstName: String, private val lastName: String) : Name(firstName) {
    override fun print() {
        super.print()
        println("$firstName $lastName")
    }
}
```

Class and Object

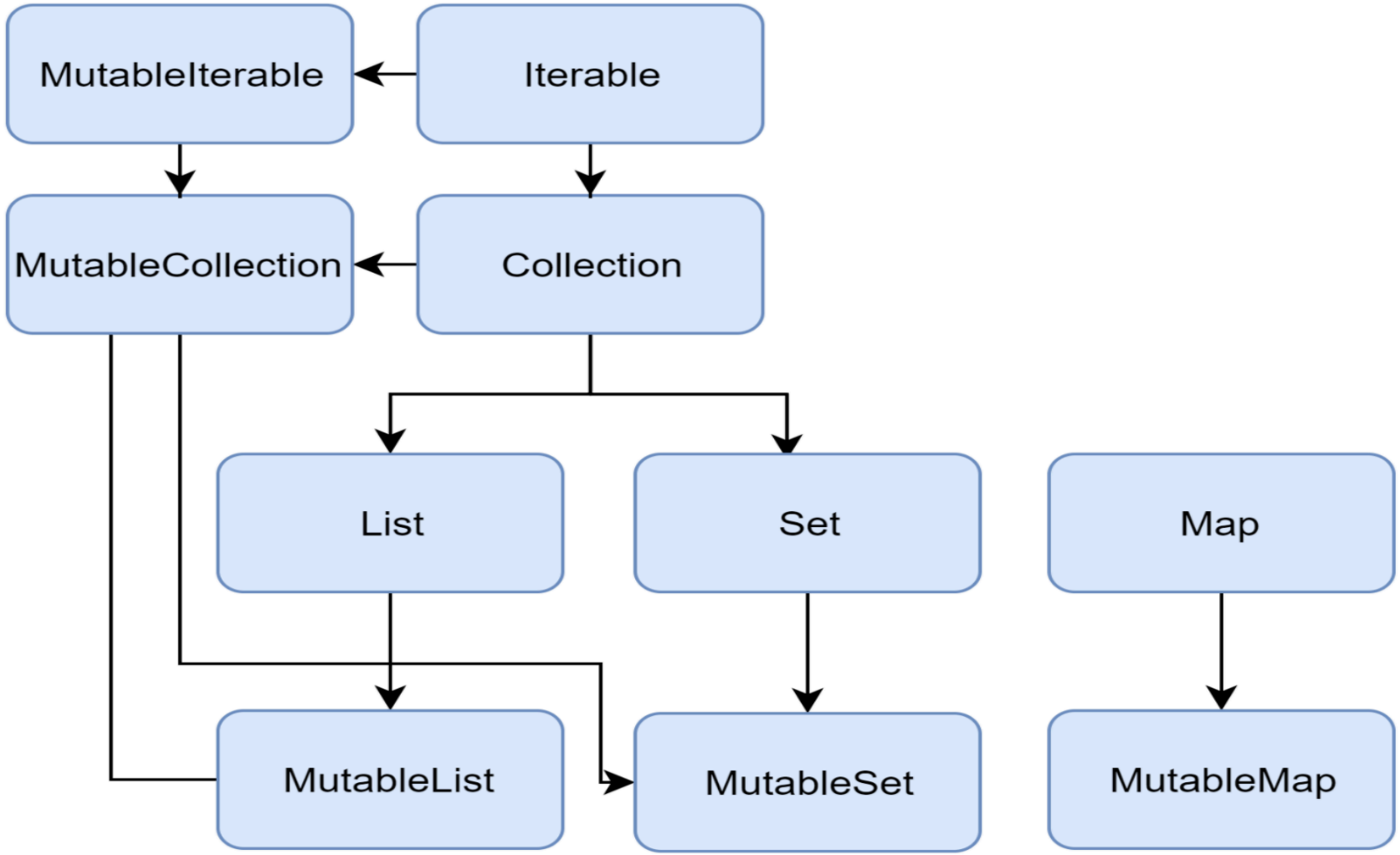
Singleton:

```
object Singleton {
    fun getObject(): String { //returns singleton
        return "service-name"
    }
}

val serviceName = Singleton.getObject()
```

Companion:

```
class MyClass {
    companion object Factory {
        fun create(): MyClass = MyClass() //factory
    }
}

val instance = MyClass.create() //Different instance each time
```

Collections



Collections

Mutable:

```
mutableListOf(1, 2, 3) //add supported
mutableSetOf(1, 2, 3)
mutableMapOf(1 to "a", 2 to "b")
```

Immutable:

```
listOf(1, 2, 3) //add not supported
setOf(1, 2, 3)
mapOf(1 to "a", 2 to "b")
```

Collections

List operations

Initialization: `val list = List(5) {it + 1} //Contains 1, 2, 3, 4, 5`

Filtering: `list.filter { it%2 == 0 } //2, 4`

Filtering: `list.filterNot { it%2 == 0 } //1, 3, 5`

List addition: `listOf(1, 2, 3) + listOf(4, 5) //1, 2, 3, 4, 5`

List subtraction: `listOf(1, 2, 3) - listOf(2) //1, 3`

Collections

Map

Access by key:

```
mapOf("one" to 1, "two" to 2)["one"] //1
mapOf("one" to 1, "two" to 2).get("one") //1
mapOf("one" to 1, "two" to 2).getOrDefault("four", 4) //4
```

Filtering:

```
mapOf("one" to 1, "two" to 2).filter{(k,v)->k.equals("one") && v.equals(1)} //contains "one", 1
```

addition: `mapOf("one" to 1) + mapOf("two" to 2)` // contains "one" -> 1, "two" -> 2

subtraction: `mapOf("one" to 1, "two" to 2) - mapOf("one" to 1)` // contains "two" -> 2

Vishwanath Sundareshan