Spring and Kotlin

A Winning Combination

Contact Info

Ken Kousen

Kousen IT, Inc.

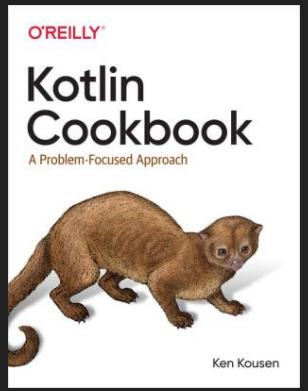
ken.kousen@kousenit.com

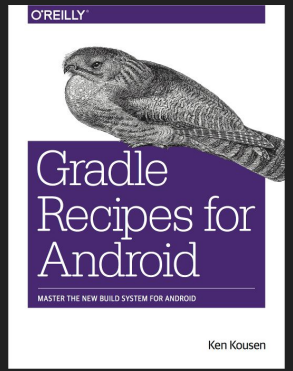
http://www.kousenit.com

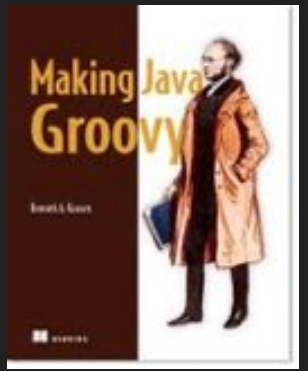
http://kousenit.org (blog)

@kenkousen (twitter)

https://kenkousen.substack.com (newsletter)







Documentation

My repository:
https://github.com/kousen/spring-kotlin

Spring Boot Kotlin support
https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-kotlin

Spring Framework Kotlin support
https://docs.spring.io/spring/docs/5.2.4.RELEASE/spring-framework-reference/languages.html#kotlin

Kotlin docs
https://docs.spring.io/spring-framework/docs/5.2.4.RELEASE/kdoc-api/spring-framework/

Tutorial guide
https://spring.io/guides/tutorials/spring-boot-kotlin/

Plugins

- Need the plugin.spring plugin →
  - Opens Spring-annotated classes for extension
    - @Component, @Async, @Transactional, @Cacheable
  - Enables proxy classes to be generated
- Also plugin.jpa →
  - Generates no-arg constructors for @Entity classes

Testing

`@SpringBootTest` includes `@ExtendWith(SpringExtension.class)`

→ JUnit 5 built in

Can change the lifecycle to one test instance per class

```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
```

or add to src/test/resources/junit-platform.properties

```
junit.jupiter.testinstance.lifecycle.default = per_class
```

Extension functions

- Add functionality to existing classes
- Easy to enhance the Java library to make it more Kotlin friendly

Extension Functions

org.springframework.ui.Model

```
operator fun Model.set(attributeName: String, attributeValue: Any) {
    this.addAttribute(attributeName, attributeValue)
}
```

Means now you can write:

`model["user"] = "World"` → invokes `addAttribute`

DSLs

Kotlin makes it easy to define domain specific languages

- Bean definition DSL
- MockMVC DSL for testing
- Router DSL for reactive streams

MockMVC DSL

Domain Specific Language for tests

```
mockMvc.get("/hello") {
     accept = MediaType.TEXT_PLAIN
     param("name", "Dolly")
}.andExpect {
     status { isOk }
     view { name("hello") }
     model { attribute("user", "Dolly") }
}
```

Router DSL

Can build a router function with a simple syntax

```
@Configuration
class RouterRouterConfiguration {

    @Bean
    fun mainRouter(userHandler: UserHandler) = router {
        accept(TEXT_HTML).nest {
            GET("/") { ok().render("index") }
            GET("/users", userHandler::findAllView)
        }

         // … more on next slide …
```

Router DSL

```
// In addition to previous ...

        "/api".nest {
            accept(APPLICATION_JSON).nest {
                GET("/users", userHandler::findAll)
            }
            accept(TEXT_EVENT_STREAM).nest {
                GET("/users", userHandler::stream)
            }
        }
        resources("/**", ClassPathResource("static/"))
    }
}
```

Reified Generics

```
inline fun <reified T : Any> runApplication(vararg args: String,
    init: SpringApplication.() -> Unit): ConfigurableApplicationContext =
        SpringApplication(T::class.java)
            .apply(init)
            .run(*args)
```

- Instantiate SpringApplication class
- Use scope function apply to initialize
- Use scope function run to execute with arguments

Kotlin data classes

Used everywhere except JPA

```
data class JokeValue(val id: Int,
                     val joke: String,
                     val categories: List<String>)

data class JokeResponse(val type: String,
                        val value: JokeValue)
```

Great for JSON serialization/deserialization

Kotlin and JPA

JPA doesn't play nicely with immutable objects

Rather than use data classes, use regular classes

```
@Entity
@Table(name = "officers")
class Officer(
        @Enumerated(EnumType.STRING)
        val rank: Rank,
        val first: String,
        val last: String,
        @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
        var id: Int? = null)  // nullable last
```

Kotlin data classes

Note you can use data classes with non-JPA

- Spring Data JDBC
- Spring Data MongoDB
- etc.

Coroutines

Very popular in Kotlin development

Makes async calls easy to write
(but that doesn't make async coding easy)

Same basic idea as webflux module

Coroutines

Example from
https://docs.spring.io/spring/docs/5.2.4.RELEASE/spring-framework-reference/languages.html#controllers

```
@RestController
class CoroutinesRestController(client: WebClient, banner: Banner) {

    @GetMapping("/suspend")
    suspend fun suspendingEndpoint(): Banner {
        delay(10)
        return banner
    }
```

Some Conclusions

- Kotlin simplifies Java in many ways
- Data classes are great for JSON
- Extension functions are easy to write and use
- Reified generics are helpful
- DSLs are available in Spring
- Getting easier to use in Spring all the time
- Coroutines are promising, but you can wait on that

Additional Kotlin Info

For those who are interested,
the remaining slides discuss Kotlin features

Kotlin

JetBrains created and maintains the language

Provides null safety at the compiler level

Statically typed and statically bound by default

Runs on the JVM → Clean interoperability with Java

Coroutines for concurrency

Kotlin

Home page is https://kotlinlang.org

Many code simplifications borrowed from other languages

- Closures similar to Groovy
- Typing system similar to Scala
- Coroutines from several other languages

Learning Kotlin

http://play.kotlinlang.org/ → online script engine

Kotlin Koans → https://kotlinlang.org/docs/tutorials/koans.html

Kotlin reference → https://kotlinlang.org/docs/reference/

Kotlin idioms → https://kotlinlang.org/docs/reference/idioms.html

Demonstrates good practices and usage patterns

Basic Syntax

Types declared after the variable, separated by a colon

```
val s : String
```

var and val:

`var` is a variable (mutable)

`val` is a value (immutable, i.e., final)

Basic Syntax

Variables are non-null by default

Must declare nullable types using "?"

```
val s : String?
```

Implies "s" can be assigned null; not true otherwise

Data Classes

Classes defined using the keyword "data"

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

(That's the entire class)

Data classes have:

- generated getters (and setters for var properties)
- toString, equals, hashCode
- `copy()` method
- componentN() methods

Functions

Functions defined with the "fun" keyword

```
fun main(args: Array<String>) { … }

fun main() { … }  // Kotlin 1.3+
```

If function consists of single statement, can use assignment

```
fun sayHello(name: String) = println("Hello, $name!")
```

(note: semicolons not needed)

Functions

Return type shown after signature

```
fun sum(a: Int, b:Int) : Int {
    return a + b
}
```

Simpler:

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

Return type inferred

(Use "Unit" return type for Java "void")

Functions

Support default parameters

```
fun read(b: Array<Byte>, off: Int = 0, len: Int = b.size) {
    ...
}
```

Override defaults by supplying actual values

if

"if" clause returns value automatically

```
val max = if (a > b) a else b
```

Acts like Java ternary operator (which isn't supported)

when

Like a Java switch statement with a return

```
var ans = when (x % 2) {
    0 -> "$x is even"
    1 -> "$x is odd"
    else -> "Houston, we have a problem..."
}
println(ans)
```

when statement must be exhaustive

when

Works with many options, including ranges

```
when (x) {
    in 1..10 -> print("$x is in the range 1..10")
    is Float -> print("$x is a Float")
    !in 10..20 -> print("x is outside 10..20")
    else -> print("none of the above")
}
```

for

Traditional Java for loop not supported

Use for-in loop

```
for (item in collection) print(item)

for (item: Int in ints) {
    // …
}
```

for

Looping over arrays, using indices

```
for (i in array.indices) {
    print(array[i])
}
```

Looping over maps, use destructuring

```
for ((index, value) in array.withIndex()) {
    println("the element at $index is $value")
}
```

Destructuring

Very common with maps

```
for ((key, value) in map) { println("$key maps to $value") }
```

On classes, uses componentN methods

val (name, dob) = person

invokes person.component1(), person.component2(), ...

data classes automatically supply componentN methods

Elvis operator

Can use ?: as in Groovy

If value is not null, use it, otherwise default

```
val s = person.name ?: "World"
```

Lambdas

Kotlin supports lambda expressions

```
inline fun <T, R : Comparable<R>> Iterable<T>.maxBy(
    selector: (T) -> R
): T?
```

```
val nameToAge = listOf("Alice" to 42, "Bob" to 28, "Carol" to 51)
val oldestPerson = nameToAge.maxBy { it.second }
println(oldestPerson) // (Carol, 51)
```

Lambdas

Basic syntax:

```
val add = { x: Int, y: Int -> x + y }
```

Can declare return type (optional here)

```
val add: (Int, Int) -> Int = { x, y -> x + y }
add(2, 3) // 5
```

If single argument, default is "it"

```
ints.filter { it > 0 }
```

Classes and Objects

Classes are defined as usual

Don't need new to instantiate

```
val customer = Customer("Fred", "flintstone@slatequarry.com")
```

Classes and Objects

To extend, class must be declared "open"

Functions must also have "open" or you can't override them

```
open class Base {
    open fun v() {}
    fun nv() {}
}
class Derived() : Base() {
    override fun v() {}
}
```

Classes and Objects

object → singleton; From reference manual:

```
object CarFactory {
    val cars = mutableListOf<Car>()

    fun makeCar(horsepower: Int): Car {
        val car = Car(horsepower)
        cars.add(car)  // we should talk about this
        return car
    }
}
```

Classes and Objects

Treat like static in Java (which Kotlin does not support):

```
val car = CarFactory.makeCar(150)
println(CarFactory.cars.size)
```

Note:

Could also define the property and method inside
companion object

of Car, rather than separate singleton

Classes and Objects

Companion objects are singletons inside classes

```
class MyClass {
    companion object {
        fun instance() = MyClass()
    }
}

val instance = MyClass.instance()
```

Classes and Objects

Note default access for everything is public

Also can put functions inside a file without a class ("script")

Becomes part of the generated class

Extension functions

Can add methods to existing classes

Good for optional methods

```
fun String.isPalindrome() =
    this.toLowerCase().replace("""[\W+]""".toRegex(), "")
        .let { it == it.reversed() }
```

See misc.palindrome.kt in GitHub repo

Sequences

Methods like "map", "filter" are added to collections

The "`asSequence()`" method converts collection to sequence

- Like Java streams
- Evaluated element at a time
- No data processed unless there is a terminal expression