Kotlin DSLs

Agenda

- Definitions
- Examples
- Ingredients
- Live Coding

Domain Specific Languages

A domain-specific language (DSL) is a computer language specialized to a particular application domain.

— *Wikipedia*

Examples

- HTML
- SQL
- Regular Expressions
- ...

Embedded DSLs

Embedded (or internal) DSLs are implemented as libraries which exploit the syntax of their host general purpose language

— *Wikipedia*

Kotlin DSLs

Domain-specific languages embedded in Kotlin

Examples - HTML [1]

```
System.out.appendHTML().html {
    body {
        div {
            a("http://kotlinlang.org") {
                target = ATarget.blank
                +"Main site"
            }
        }
    }
}
```

[1] kotlinx.html, https://github.com/Kotlin/kotlinx.html

Examples - JSON [2]

```
import com.lectra.koson.*

val o = obj {
    "property" to "value"
    "array" to arr[1, 2, 3]
    "null" to null
    "empty" to obj {  }
}
```

[2] Koson, https://github.com/lectra-tech/koson

Examples - Gradle [3]

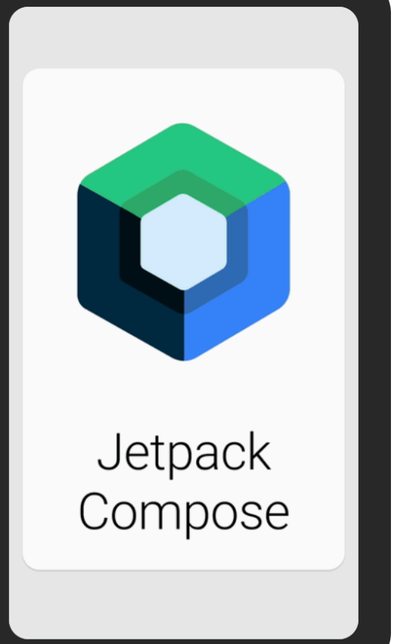
```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
    kotlin("jvm") version "1.6.0"
}
tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "11"
}
repositories {
    mavenCentral()
}
dependencies {
    implementation("com.lectra:koson:1.1.0")
    implementation("org.jetbrains.kotlinx:kotlinx-html-jvm:0.7.3")
}
```

---

[3] Kotlin Gradle DSL, https://docs.gradle.org/current/userguide/kotlin_dsl.html

Examples - Android Layouts [5]

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.h2,
                )
            }
        }
    }
}
```



[5] Jetpack Compose, https://developer.android.com/jetpack/compose

Example - Spring Web Router Config [6]

```
@Bean
fun mainRouter(userHandler: UserHandler) = router {
    accept(TEXT_HTML).nest {
        GET("/") { ok().render("index") }
        GET("/sse") { ok().render("sse") }
        GET("/users", userHandler::findAllView)
    }
    "/api".nest {
        accept(APPLICATION_JSON).nest {
            GET("/users", userHandler::findAll)
        }
        accept(TEXT_EVENT_STREAM).nest {
            GET("/users", userHandler::stream)
        }
    }
    resources("/**", ClassPathResource("static/"))
}
```

---

[6] Spring Framework, https://docs.spring.io/spring-framework

Example - Spring Mock MVC DSL [7]

```
mockMvc.get("/person/{name}", "Lee") {
    secure = true
    accept = APPLICATION_JSON
    headers {
        contentLanguage = Locale.FRANCE
    }
    principal = Principal { "foo" }
}.andExpect {
    status { isOk }
    content { contentType(APPLICATION_JSON) }
    jsonPath("$.name") { value("Lee") }
    content { json("""{"someBoolean": false}""", false) }
}.andDo {
    print()
}
```

---

[7] Spring Framework, https://docs.spring.io/spring-framework

Example - Rest Template Execution DSL [8]

```
fun create(account: Account) {
    restTemplate.runCatching {
        httpPost<String>(createAccountUrl) {
            body {
                model(account)
            }
        }.execute()
    }.onFailure {
        logger.error("Error creating account: ${it.message}", it)
        throw it
    }
}
```

[8] REWE digital codebase

Build your own DSL

Ingredients

Extention functions

Instead of this

```
fun <T> swap(list: MutableList<T>, index1: Int, index2: Int) {
    val tmp = list[index1]
    list[index1] = list[index2]
    list[index2] = tmp
}
val list = mutableListOf(1, 2, 3)
swap(list, 1, 2)
```

Extention functions

... we can write this

```
fun <T> MutableList<T>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
val list = mutableListOf(1, 2, 3)
list.swap(1,2)
```

Function Type with receivers[9]

The type of an extention function

```
fun A.f(b: B): C {
    // ...
}
```

is

```
A.(B) -> C
```

[9] Function literals with receivers

Lambda Expressions with Receivers

So what does this do?

```
val printMe: String.() -> Unit = { println(this) }

"Stefan".printMe()
```

Lambda Expressions with Receivers

Now let's assume we have a class HTML

```
class HTML {
    fun head() {}
    fun body() {}
}
```

...

Lambda Expressions with Receivers

... and a function `html`

```
fun html(init: HTML.() -> Unit): HTML {
    val html = HTML()
    html.init()
    return html
}
```

Lambda Expressions with Receivers

... then we can write

```
html({
    this.head()
    this.body()
})
```

Lambda Expressions with Receivers

`this` can be omitted:

```
html({
    head()
    body()
})
```

Lambda Expressions with Receivers

We apply syntactic sugar:

```
html() {
    head()
    body()
}
```

Lambda Expressions with Receivers

And again:

```
html {
    head()
    body()
}
```

Bingo!

Lambda Expressions with Receivers

Now let's go from here

```
class HTML {
    fun head() {}
    fun body() {}
}
```

Lambda Expressions with Receivers

... to here:

```
class HTML {
    fun head(init: Head.() -> Unit): Head {
        val head = Head(); head.init(); return head
    }
    fun body(init: Body.() -> Unit): Body {
        val body = Body(); body.init(); return body
    }
}

class Head { fun title() { } }

class Body { fun p() { } }
```

Lambda Expressions with Receivers

Then we can write

```
html {
    head {
        title()
    }
    body {
        p()
    }
}
```

... and so on.

Scope control

But …

```
html {
    head {
        head {
            title()
        }
    }
    // ...
}
```

would also be OK. :-(

DSL Markers for the rescue!

Let's control receiver scope with `@DslMarker`:

```
@DslMarker
annotation class HtmlTagMarker

@HtmlTagMarker
class HTML { // ...
}

@HtmlTagMarker
class Head { // ...
}

@HtmlTagMarker
class Body { // ...
}
```

DSL Markers

If we now try

```
30 html {
31     head {
32         head {
33             title()
34         }
35     }
      // ...
}
```

…

DSL Markers

... we will get

```
$ ./gradlew compileKotlin
e: .../html.kt: (32, 13): 'fun head(init: Head.() -> Unit):
  Head' can't be called in this context by implicit receiver.
  Use the explicit one if necessary
> Task :compileKotlin FAILED
```

DSL Markers

Well, we still could write

```
html {
    head {
        this@html.head {
            title()
        }
    }
    // ...
}
```

... but who would do this?

The whole thing ...

... could be found here: Type-Safe builders.

More Ingredients: Invoke

With

```
class MyClass {
    operator fun invoke() { // ...
    }
}
```

one can write

```
val myObj = MyClass()
myObj() // will call invoke
```

Invoke for DSLs

With

```
class Family {
    operator fun invoke(body: Family.() -> Unit) { body() }
    fun addMember(name: String) {}
}
```

one can write

```
val family = Family()
family {
    addMember("Mom"); addMember("Dad"); addMember("Kid")
}
```

More Ingredients

- Operator Overloading
- Infix Functions

More Examples for DSLs

- https://dzone.com/articles/kotlin-dsl-from-theory-to-practice
  convert Test Data Builder to Kotlin DSL
- https://kotlinexpertise.com/create-dsl-with-kotlin/
  create a DSL for setting up a TLS connection
- https://kotlinexpertise.com/java-builders-kotlin-dsls/
  convert Java builders for Android Material Drawer to Kotlin DSL
- https://blog.codecentric.de/2018/06/kotlin-dsl-apache-kafka/
  A simple example of Kotlin DSL for Apache Kafka producer and consumer (German)

Slides

https://speakerdeck.com/stefanscheidt/kotlin-dsls

Sources

https://github.com/stefanscheidt/kotlin-dsl

Thank you!

@stefanscheidt on Twitter and GitHub