KREATIVER EINSATZ VON KOTLIN

Jax, 26.04.2018, Mainz

@RenePreissel

https://github.com/rpreissel/kotlin-creative.git

KTOR - ASYNCHRONOUS SERVER

```
@Location("/") class Index()
@Location("/hello/{name}") class Hello(val name: String)

get<Index> {
    call.respondHtml {
        body {
            h1 {
                +"Greeter"
            }
            div {
                a(locations.href(Hello("Rene"))) {
                    +"Say Hello"
                }
            }
        }
    }
}
```

ANKO - ANDROID LIBRARIES

```
override fun createView(ui: AnkoContext<Context>): View = with(ui) {
    constraintLayout {
        hello = textView("Hello World")

        applyConstraintSet {
            hello {
                connect(TOP to TOP of PARENT_ID margin dip(100),
                        LEFT to LEFT of PARENT_ID,
                        RIGHT to RIGHT of PARENT_ID,
                        BOTTOM to BOTTOM of PARENT_ID)
            }
        }
    }
}
```

INHALT

- Kotlin-Intro
- Kotlin-Syntax-Features mit Beispielen
  - KLogging, KotlinJS/React
  - Exposed, Gradle
  - Kodein, Anko, Ktor
  - ...

KOTLIN-INTRO

```
class Circle(val radius: Double) {
    fun diameter(): Double {
        return radius * 2
    }
    val circumference: Double
        get() {
            return diameter() * PI
        }
    val area: Double
        get() = PI * radius * radius
}

fun main(args: Array<String>) {
    val circle = Circle(42.0)
    println(circle.radius)
    println("Diameter: ${circle.diameter()} Circumference: ${circle.circumference}"
}
```

KOTLIN-LOGGING MIT COMPANION

```kotlin
class Circle(val radius: Double) {
    fun diameter(): Double {
        logger.info("compute diameter")
        return radius * 2
    }

    companion object : KLogging()
}

open class KLogging : KLoggable {
    override val logger: KLogger = logger()
}

fun main(args: Array<String>) : Unit {
    Circle.logger.info("companion access / e.g. static access")
}
```

LAMBDAS

```kotlin
class Circle(val radius: Double) {
    val circumference: Double
        get() {
            logger.info {
                "compute circumference"
            }
            return diameter() * PI
        }

    companion object : KLogging()
}
```

```kotlin
fun info(msg: () -> Any?) {
    if (isInfoEnabled)
        info(msg().toString())
}
```

TRACING

```
class Circle(val radius: Double) {
    fun diameter(): Double
        = radius * 2
}
```

```
class Circle(val radius: Double) {
    fun diameter(): Double = try {
        logger.info("diameter - begin")
        radius * 2
    } finally {
        logger.info("diameter - end")
    }

    companion object : KLogging()
}
```

TRACING MIT LAMBDAS

```
class Circle(val radius: Double) {
    fun diameter(): Double = withTracing("diameter") {
        radius * 2
    }
    companion object : MyLogging()
}
```

```
open class MyLogging : KLogging() {
    fun <T> withTracing(
        prefix: String, block: () -> T
    ): T = try {
        logger.info("$prefix - begin")
        block()
    } finally {
        logger.info("$prefix - end")
    }
}
```

TRACING INNERHALB VON FUNKTIONEN

```
class Circle(val radius: Double) {
    val area: Double
        get() = withTracing("area") {
            logger.info("area - before radius square")
            val r2 = radius * radius
            logger.info("area - after radius square")
            PI * r2
        }

    companion object : MyLogging()
}
```

TRACING MIT KONTEXT

```
class Circle(val radius: Double) {
    val area: Double
        get() = withTracing("area") {
            trace("before radius square")
            val r2 = radius * radius
            trace("after radius square")
            PI * r2
        }
}
```

```
class TracingContext(val logger: KLogger, val prefix: String) {
    fun trace(msg: String) {
        logger.info("$prefix - $msg")
    }
}
```

LAMBDA WITH RECEIVER - LAMBDA++

```
open class MyLogging : KLogging() {
    fun <T> withTracing(
        prefix: String, block: TracingContext.() -> T
    ): T = try {
        logger.info("$prefix - begin")
        val context = TracingContext(logger, prefix)
        context.block()
    } finally {
        logger.info("$prefix - end")
    }
}
```

```
val area: Double
    get() = withTracing("area") { // 'this' ist ein TracingContext
        trace("before radius square")
        ...
```

KOTLINJS / REACT

- KotlinJS transpiliert nach JavaScript
- Wrapper für React ist vorhanden
- Interne DSL für HTML (ähnlich JSX) vorhanden

LAMBDA++ IN KOTLINJS / REACT

```kotlin
div("App-header") {
    key = "header"
    logo()
    h2 {
        +"Welcome to React with Kotlin"
    }
}
p("App-intro") {
    key = "intro"
    +"To get started, edit "
    code { +"app/App.kt" }
    +" and save to reload."
}
```

LAMBDA++

- Expliziter Kontext (`this`) für einen Code-Block
- Nutzen:
  - Bereitstellen von speziellen Funktionen im Code-Block
  - Vermeidung von wiederholenden expliziten Parametern
  - Umsetzung eines Builder-Patterns - Zusammensetzen von komplexen Objekten

EXTENSION-FUNKTIONEN

```kotlin
val url: URL = StringUtil.toURL("http://localhost:8080")

object StringUtil {
    fun toURL(s: String) = URL(s)
}
```

```kotlin
fun String.toURL() = URL(this)

val url = "http://localhost:8080".toURL()
```

EXTENSION ANSTELLE VON VERERBUNG

```kotlin
open class MyLogging : KLogging() {
    fun <T> withTracing(
        prefix: String,
        block: TracingContext.() -> T
    ): T = ...
}

class Circle(val radius: Double) {
    fun diameter(): Double = withTracing("diameter") {
        radius * 2
    }

    companion object : MyLogging()
}
```

EXTENSION ANSTELLE VON VERERBUNG

```
fun <T> KLogging.withTracing(
    prefix: String,
    block: TracingContext.() -> T
): T = ...

class Circle(val radius: Double) {
    fun diameter(): Double = withTracing("diameter") {
        radius * 2
    }

    companion object : KLogging()
}
```

EXTENSIONS IN KOTLINJS/REACT

```
div("App-header") {
    key = "header"
    logo()
    h2 {
        +"Welcome to React with Kotlin"
    }
}
```

```
fun RBuilder.logo(height: Int = 100) {
    div("Logo") {
        attrs.style = js {
            this.height = height
        }
        img(alt = "React logo.logo", src = reactLogo, classes = "Logo-react") {}
    }
}
```

EXTENSION-FUNKTIONEN

- Zusätzliche Funktionen an vorhandenen Typen
- Nutzen:
  - Vermeidung von Util-Klassen
  - Vermeidung von Vererbung
  - Umsetzung von erweiterbareren DSLs

LOKALE EXTENSION-FUNKTIONEN

```kotlin
val area: Double
    get() = withTracing("area") {
        val r2 = radius * radius
        trace("dump r2: $r2")
        PI * r2
    }
```

```kotlin
val area: Double
    get() = withTracing("area") {
        PI * (radius * radius).dump("dump r2")
    }
```

LOKALE EXTENSION-FUNKTIONEN

```
class TracingContext(val logger: KLogger, val prefix: String) {
    ...
    fun <T> T.dump(msg: String): T {
        trace("$msg: $this") // entspricht: this@TracingContext.log("$msg: $this")
        return this
    }
}
```

```
val area: Double
    get() = withTracing("area") {
        PI * (radius * radius).dump("dump r2")
    }
```

LOKALE EXTENSIONS IN KOTLINJS/REACT

```kotlin
interface TickerState : RState {
    var secondsElapsed: Int
}

class Ticker(props: TickerProps) : RComponent<TickerProps, TickerState>(props) {
    override fun TickerState.init(props: TickerProps) {
        secondsElapsed = props.startFrom
    }

    override fun RBuilder.render() {
        p {
            +"This app has been running for ${state.secondsElapsed} seconds."
        }
    }
}
```

LOKALE EXTENSION-FUNKTIONEN

- Zusätzliche Funktionen **nur** in einem bestimmten Kontext
- Nutzen:
  - Bereitstellen von Hilfsfunktionen in einem bestimmten Kontext
  - Implizite Übergabe von Kontext (`this`) an Hilfsfunktionen
  - Impliziter Aufruf von Methoden bei zwei Typen (zwei `this`-Zeiger)

INFIX-FUNKTIONEN

```
class Circle(val x: Double, val y: Double, val radius: Double) {
    fun intersects(other: Circle): Boolean ...
}

val c1 = Circle(x = 100.0, y = 100.0, radius = 50.0)
val c2 = Circle(x = 75.0, y = 75.0, radius = 5.0)
println(c1.intersects(c2))
```

```
class Circle(val x: Double, val y: Double, val radius: Double) {
    infix fun intersects(other: Circle): Boolean ...
}

val c1 = Circle(x = 100.0, y = 100.0, radius = 50.0)
val c2 = Circle(x = 75.0, y = 75.0, radius = 5.0)
println(c1 intersects c2)
```

EXPOSED - KOTLIN SQL LIBRARY

- DSL für SQL Zugriffe

```kotlin
object Addresses : Table() {
    val id = integer("id").autoIncrement().primaryKey()
    val city = varchar("city", 50)
}

Addresses.insert {
    it[city] = "Hamburg"
}
```

INFIX-FUNKTIONEN IN EXPOSED

```
object Persons : Table() {
    val id = integer("id").autoIncrement().primaryKey()
    val name = varchar("name", length = 50)
    val addressId = integer("address_id") references Addresses.id
}

val personId = Persons.insert {
    it[name] = "Rene"
    it[addressId] = hamburgId
} get Persons.id

Persons.deleteWhere { Persons.name like "%e" }

val allNamesWithCities = (Persons innerJoin Addresses)
            .slice(Persons.name, Addresses.city)
            .selectAll()
```

INFIX-FUNKTIONEN

- Funktionen wie Operatoren aufrufen
- Nutzen:
  - Vermeidung von Klammern
  - 'Fluent' APIs
  - Gut geeignet für algebraische Domänen

DELEGATED PROPERTIES

```kotlin
class Circle(val radius: Double) {
    val diameter: Double by lazy {
        radius * 2
    }
    val circumference: Double by lazy {
        diameter * PI
    }
}
```

```kotlin
fun <T> lazy(initializer: () -> T): Lazy<T> = ...

operator fun <T> Lazy<T>.getValue(thisRef: Any?,
                                  property: KProperty<*>): T = value
```

GRADLE - BUILD-TOOL

- Unterstützt Kotlin(Script) als Build-Language

```
plugins {
    id("org.jetbrains.kotlin.jvm") version "1.2.30"
}
configure<JavaPluginConvention> {
    sourceCompatibility = JavaVersion.VERSION_1_8
}

val clean by tasks

val helloTask by tasks.creating {
    dependsOn(clean)
    doLast { println("Hello") }
}
```

DELEGATED PROPERTIES IN GRADLE

```
val clean = tasks["clean"]

val helloTask = tasks.create("helloWorld") {
    dependsOn(clean)
    doLast { println("Hello") }
}
```

```
val clean by tasks

val helloTask by tasks.creating {
    dependsOn(clean)
    doLast { println("Hello") }
}
```

DELEGATED PROPERTIES

- Delegieren der lesenden und schreibenden Property-Zugriffe
- Nutzen:
  - Implementierung von verschiedenen Entwurfsmuster (Proxy, Observer, etc)
  - Vermeidung der doppelter Nennung von Variablennamen

OPERATOREN ÜBERLADEN

```
fun main(args: Array<String>) {
    val c1 = Circle(x = 100.0, y = 100.0, radius = 50.0)
    val c2 = Circle(x = 75.0, y = 75.0, radius = 5.0)
    println(c1 intersects c2)
    println(c1 % c2)
}

class Circle(val x: Double, val y: Double, val radius: Double) {
    infix fun intersects(other: Circle): Boolean ...

    operator fun mod(other: Circle): Boolean = intersects(other)
}
```

OPERATOREN IN KOTLINJS/REACT

```
div("App-header") {
    key = "header"
    logo()
    h2 {
        +"Welcome to React with Kotlin"
    }
}
p("App-intro") {
    key = "intro"
    +"To get started, edit "
    code { +"app/App.kt" }
    +" and save to reload."
}
```

OPERATOREN IN GRADLE

```
tasks {
    "worldTask" {
        dependsOn(helloTask)
        doLast { println("World") }
    }
}
```

```
class NamedDomainObjectContainerScope<T : Any>(
    operator fun String.invoke(configuration: T.() -> Unit): T =
        this().apply(configuration)
    ...
```

```
tasks {
    "worldTask"(Zip::class) {
    ...
```

OPERATOREN

- Eigene Operatoren für Datentypen definieren
- Nutzen:
  - Kompakterer Code
  - Eher geeignet für mathematische Domänen

WEITERE BEISPIELE

KODEIN - DEPENDENCY INJECTION

```
val kodein = Kodein {
    constant("dburl") with "jdbc:h2:mem:singleton"

    bind<DataSource>() with singleton {
        JdbcDataSource().apply {
            setURL(instance("dburl"))
        }
    }
}

val datasource: DataSource = kodein.direct.instance()
```

DELEGATED PROPERTIES IN KODEIN

```
class DatabaseService(override val kodein: Kodein) : KodeinAware {
    val dataSource: DataSource by instance()
    val dbUrl: String by instance("dburl")
}
```

```
fun main(args: Array<String>) {
    val kodein = Kodein {
        constant("dburl") with "jdbc:h2:mem:singleton"
        bind<DataSource>() with singleton {
            JdbcDataSource().apply { setURL(instance("dburl")) }
        }
    }
    val databaseService = DatabaseService(kodein)
}
```

ANKO - ANDROID EXTENSIONS

```
override fun createView(ui: AnkoContext<Context>): View = with(ui) {
    constraintLayout {
        hello = textView("Hello World")

        applyConstraintSet {
            hello {
                connect(TOP to TOP of PARENT_ID margin dip(100),
                        LEFT to LEFT of PARENT_ID,
                        RIGHT to RIGHT of PARENT_ID,
                        BOTTOM to BOTTOM of PARENT_ID)
            }
        }
    }
}
```

KTOR - ASYNCHRONOUS SERVER

```
@Location("/") class Index()
@Location("/hello/{name}") class Hello(val name: String)

fun Application.main() {
    install(Locations)

    routing {
        get<Index> {
            call.respondHtml {
                body {
                    div {
                        a(locations.href(Hello("Rene"))) {
                            +"Say Hello"
                        }
...
```

COMPANION-FACTORY

```
fun <B, ...> install(feature: ApplicationFeature<...>,
                     configure: B.() -> Unit = {})

open class Locations(val application: Application,
                     val routeService: LocationRouteService) {

    companion object Feature : ApplicationFeature<...> {
        override fun install(pipeline: Application,
                             configure: Locations.() -> Unit): Locations {
            val routeService = LocationAttributeRouteService()
            return Locations(pipeline, routeService).apply(configure)
        }
    }
}

install(Locations)
```

KOOBY/JOOBY - WEB FRAMEWORK

```kotlin
class App: Kooby({
    get("/") {
        val name = param("name").value("Kotlin")
        "Hello $name!"
    }
})

fun main(args: Array<String>) {
    org.jooby.run(::App, *args)
}
```

SPEK - SPECIFICATION FRAMEWORK

```
object AppTest : Spek({
    jooby(App()) {
        describe("Get /") {
            given("no queryParameter") {
                it("should return Kotlin as the default name") {
                    get("/")
                        .then()
                        .assertThat().statusCode(Status.OK.value())
                        .extract().asString()
                        .let {
                            //Kluent Infix Method
                            it shouldEqual "Hello Kotlin!"
                        }
                }
            }
...
```

WAS KONNTE ICH NICHT ZEIGEN

- Delegation Interfaces für "Mehrfachvererbung"
  (siehe Tests in Squash)
- `@DslMarker` um Probleme mit Verschachtelung zu vermeiden
  (siehe `ContextDsl` in Ktor)
- Inline / `@PublishedApi` um eigene effiziente Kontrollstrukturen zu bauen
- Inline Reified um auf die Klasse von generische Typparameter zuzugreifen
- Interfaces als Anker für Extension-Funktionen
  (siehe `AnkoLogger`)

FRAGEN?

@RenePreissel
rene.preissel@etosquare.de
www.etosquare.de

LINKS I

https://kotlinlang.org/

https://ktor.io

https://github.com/Kotlin/anko

https://github.com/MicroUtils/kotlin-logging

https://github.com/JetBrains/create-react-kotlin-app

https://github.com/JetBrains/Exposed

LINKS II

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

https://github.com/Kodein-Framework/Kodein-DI

https://jooby.org/doc/lang-kotlin

http://spekframework.org

http://rest-assured.io

https://github.com/MarkusAmshove/Kluent

https://github.com/orangy/squash