Domain Specific Languages
entwerfen mit Kotlin

https://gitpitch.com/dxfrontiers/kotlin-gherkin-dsl
https://github.com/dxfrontiers/kotlin-gherkin-dsl

Frank Scheffler

@maverick_1601

https://blog.digitalfrontiers.de
https://www.digitalfrontiers.de
“Wer hat schon mal mit Kotlin gearbeitet?"
fun main() {
    val str1 = "Hello!"
    // str1 = "Hello World!"

    // val str2: String = null

    var str3: String?
    // println(str3)
    str3 = "Hello World!"
    println(str3)
    println(str3)
    str3 = null
    println(str3)
}
fun greet(to: String): String {
    return "Hello $to!"
}

fun divide(a: Int, b: Int) = a / b

fun String.printGreeting() {
    this.let(::greet).run(::println)
}

fun main() {
    println(greet("Frank"))
    println(divide(a = 4, b = 2))
    "Frank".let(::greet)
    .let(::println)
fun `doInTransaction`(action: () -> Unit) {
  // begin tx
  try {
    action()
    // commit tx
  } catch (t: Throwable) {
    // rollback tx
    throw t
  }
}

fun `main`() {
  val `txAction`: () -> Unit = {
    println("Hello Transaction!")
  }
  `doInTransaction`(txAction)
}
class TxContext { /* ... */ }

fun <T> doInTransactionWithResult(action: (TxContext) -> T): T {
    val tx = TxContext()
    // begin tx
    return runCatching { action(tx) }
        .onSuccess {
            /* commit tx */
        }
        .onFailure {
            /* rollback tx */
        }
        .getOrThrow()
}

fun main() {
    val str = doInTransactionWithResult { "Hello Transaction: $it"
    println(str)
}
fun <T> doInTxContextWithResult(action: TxContext() -> T): T {
    val tx = TxContext()
    // begin tx
    return runCatching { tx.action() }
        .onSuccess { /* commit tx */ }
        .onFailure { /* rollback tx */ }
        .getOrThrow()
}

fun main() {
    val str = doInTxContextWithResult { "Hello Transaction: $this" }
    println(str)
}
Domain Specific Languages

"DSLs are small languages, focused on a particular aspect of a software system. You can't build a whole program with a DSL, but you often use multiple DSLs in a system mainly written in a general purpose language."

- Martin Fowler with Rebecca Parsons
Domain Specific Languages

“DSLs come in two main forms: external and internal. An external DSL is a language that's parsed independently of the host general purpose language: good examples include regular expressions and CSS.”

- Martin Fowler with Rebecca Parsons
Internal DSLs are a particular form of API in a host general purpose language, often referred to as a fluent interface. The way mocking libraries, such as JMock, define expectations for tests are good examples of this.

- Martin Fowler with Rebecca Parsons
Reducing Boiler-Plate Code

```kotlin
interface Database {
    // be sure to close
    fun connect(): Connection
    fun close(c: Connection)

    interface Connection {
        fun <T> queryFor(query: String): T
    }
}

fun boilerPlated() {
    val db = myDb("jdbc:mysql:localhost:12356/test")
    val connection = db.connect()
    try {
        val result = connection.queryFor<Int>("SELECT ....")
    }
}
```
Separation of Declaration and Execution

object Users: Table("USERS") {
    val name: String = column("NAME")
    val surname: String = column("SURNAME")
    val age: Int = column("AGE")
}

fun main() {
    val statement = from(Users).where {
        Users::name eq "Hugo"
        Users::age gt 18
    }.select {
        Users::name alias "name"
        Users::surname alias "nachname"
    }
    statement.execute(myDb("jdbc:mysql:localhost:12356/test"))
}
Advantages of DSLs

**Predictable**
Domain logic is expressed declaratively, while execution is controlled by the DSL.

**Safer**
Restricted dialect and clearer scoping reduces unsafe operations.

**Easier to Understand**
Domain experts are able to learn the DSL in less time.

**More Expressive**
Domain code clearly expresses what should be done. Domain specific errors are easier to understand.

**Portable**
Domain logic execution can be ported to new platforms, if necessary.
A Gherkin Test DSL

```
GIVEN
state
before
behavior is applied

WHEN
specifies
the behavior part

THEN
defines
changes or outcome expected
```

"Gherkin uses a set of special keywords to give structure and meaning to executable specifications."

- Cucumber
import assertk.assertThat
import assertk.assertions.isEqualTo
import org.junit.jupiter.api.Test

class Calculator {
    fun add(a: Int, b: Int) = a + b
}

class CalculatorTest {
    @Test
    fun `adds up numbers (gherkin)`() {
        // Additional test code here
    }
}
package de.digitalfrontiers.kotlin.gherkin

import assertk.Assert
import assertk.assertAll
import assertk/assertThat
import assertk.assertions.contains
import assertk.assertions.endsWith
import assertk.assertions.startsWith

typealias given<T> = Given<T>

class Given<S>(private val setup: () -> S) {
    fun <R> on(block: S.() -> R): Result<R> = Result { setup().block() }
}
Repeated Test Execution

```kotlin
@Test
fun `provides random multiples in range (repeatedly)`() {
    given {
        Calculator()
    }.on(repeat = 100) {
        randomMultiple(9)
    }.thenAssert {
        isBetween(1, 100)
    }
}
```
package de.digitalfrontiers.kotlin.gherkin

import org.junit.jupiter.api.Test
import java.util.*

class Greeter(localeProvider: () -> Locale = Locale::getDefault) {
    val greeting: String = when (localeProvider.invoke().language
        "en" -> "Hello"
        "de" -> "Hallo"
        "fr" -> "Bonjour"
        else -> TODO("not yet implemented")
    }

    fun sayHello(name: String) = "$greeting $name!"
}
The Gherkin DSL

```kotlin
package de.digitalfrontiers.kotlin.gherkin

import assertk.Assert
import assertk.assertAll
import assertk.assertThat
import assertk.assertions.contains
import assertk.assertions.endsWith
import assertk.assertions.startsWith

typealias given<T> = Given<T>

class Given<S>(private val setup: () -> S) {

    fun <R> on(block: S.() -> R): Result<R> =
        Result { setup().block() }
}
```
Kotlin DSL Building Blocks

1. Capturing declarations using Lambdas
2. Contexts using Lambdas with Receivers
3. Scoped extension functions (should)
4. Infix notation for improved readability
Wofür werden Sie morgen eine Kotlin DSL erstellen?
Wofür werden Sie morgen eine **Kotlin DSL** erstellen?

---

**Domain Specific Languages entwerfen mit Kotlin**

[@maverick_1601](https://github.com/maverick1601)

[@dxfrontiers](https://www.digitalfrontiers.de)

---

[https://gitpitch.com/dxfrontiers/kotlin-gherkin-dsl](https://gitpitch.com/dxfrontiers/kotlin-gherkin-dsl)

[https://github.com/dxfrontiers/kotlin-gherkin-dsl](https://github.com/dxfrontiers/kotlin-gherkin-dsl)

---

@dxfrontiers

WeAreHiring

https://blog.digitalfrontiers.de

https://www.digitalfrontiers.de
Domain Specific Languages
entwerfen mit Kotlin

Frank Scheffler