A Kotlin DSL to code business process

Inside a Spring boot application

Bonitasoft

#Bonitasoft @baptistemesta

About me

About me

- Baptiste Mesta

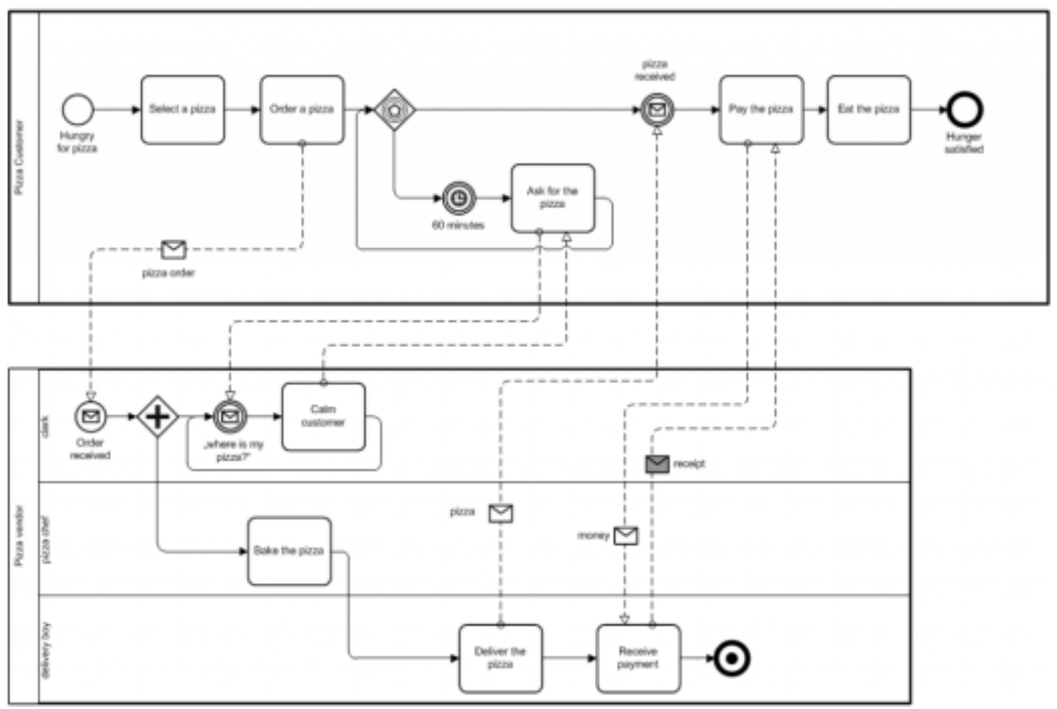
About me

- Baptiste Mesta
- Dev @ Bonitasoft for 10 years
  - BPM Execution engine
  - Developer tooling
  - IA/Analytics module

Process modeling pre-requisite

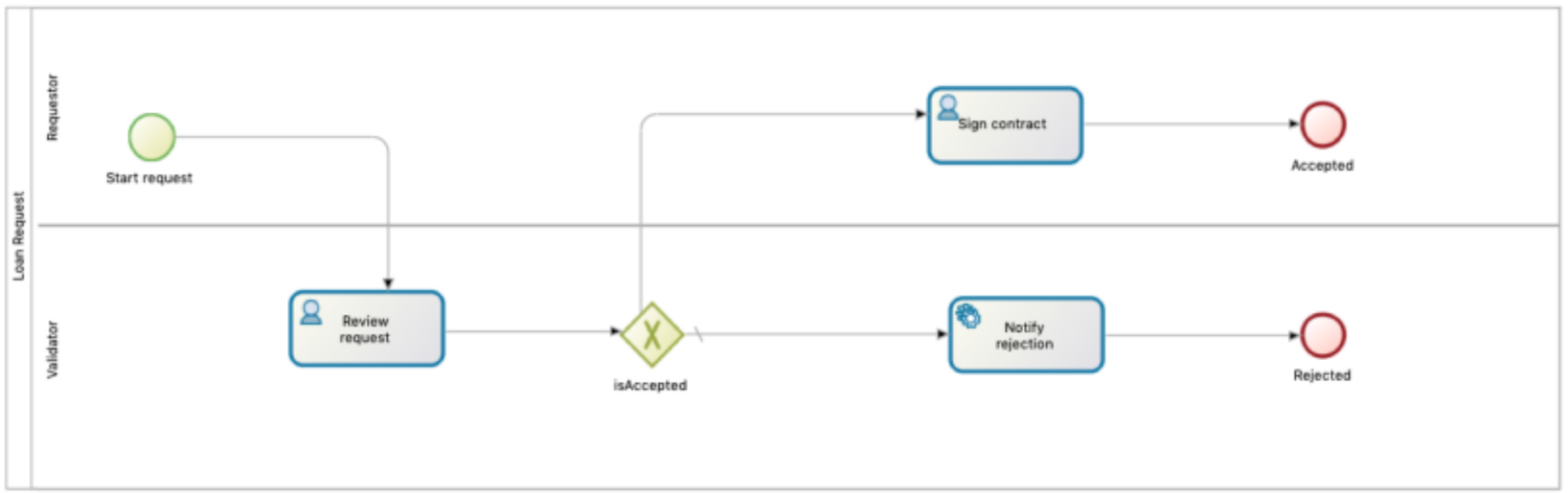
Pre-requisite: BPMN in 2 minutes!

BPMN Process definition



Our use case

Loan request application



Application based on business processes

Studio to design processes

#Bonitasoft @baptistemesta

Pros of using the full platform

Pros of using the full platform

- Low code development

Pros of using the full platform

- Low code development
- Visual representation of the process

Pros of using the full platform

- Low code development
- Visual representation of the process
- Business data editor

Pros of using the full platform

- Low code development
- Visual representation of the process
- Business data editor
- WYSIWYG editor to create pages and forms

Pros of using the full platform

- Low code development
- Visual representation of the process
- Business data editor
- WYSIWYG editor to create pages and forms
- Platform to administrate processes / users

Pros of using the full platform

- Low code development
- Visual representation of the process
- Business data editor
- WYSIWYG editor to create pages and forms
- Platform to administrate processes / users
- Out of the box user case list

As a developer...

I want everything as code in one place

Another approach

A **DSL** to code processes and execute them using an **Embedded engine**

What is a DSL?

Domain specific language

> *The basic idea of a domain specific language (DSL) is a computer language that's targeted to a particular kind of problem, rather than a general purpose language that's aimed at any kind of software problem.*

— Martin Fowler

External DSLs

aka *"Create a new language from scratch"*

Internal DSLs

aka *"I don't want to spend 2 months on that"*

We already uses DSLs

Spock framework

```
def "should return user given its id and tenant id"() {
    given:
    def expectedUser = aUser(new User(id: 1))
    when:
    def user = repository.findById(1).get()
    then:
    user == expectedUser
}
```

We already uses DSLs

Gradle DSL

```
plugins {
    `kotlin-dsl`
}
repositories {
    jcenter()
}
dependencies {
    compile("commons-httpclient:commons-httpclient:3.1")
}
```

Kotlin DSL

Kotlin lang

kotlin is...

- cross-platform
- statically typed
- executable on the JVM
- designed to interoperate fully with Java
- concise

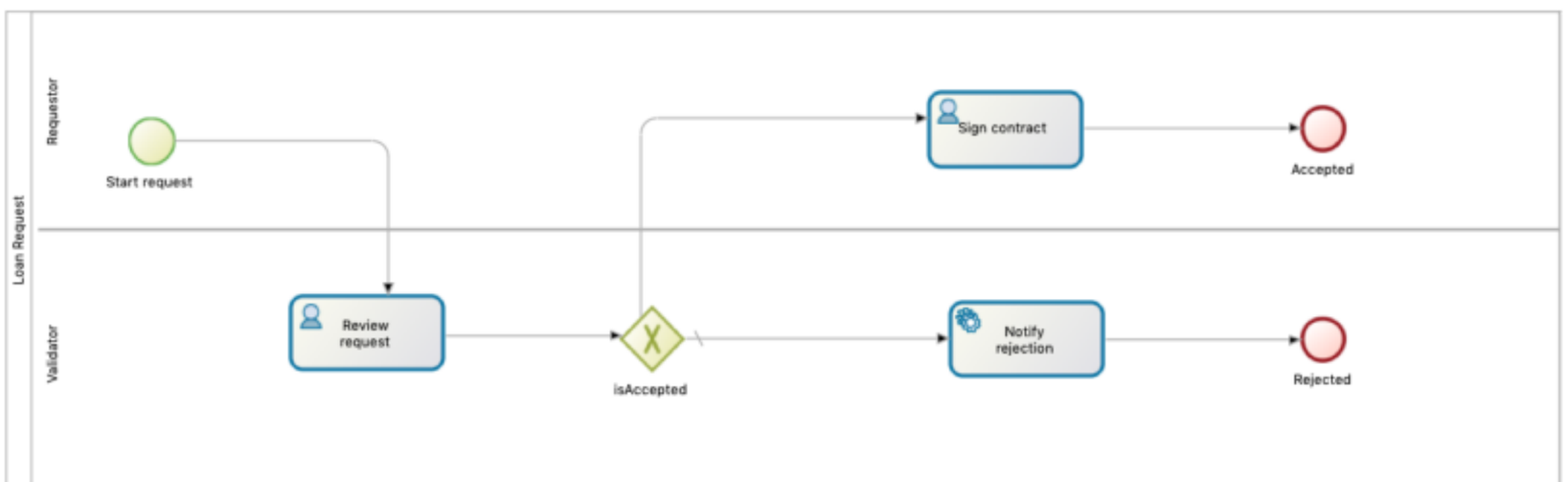
Kotlin makes writing DSL easy

```
html {
    head {
        title {+"Some title"}
    }
    body {
        h1 {+"Some title"}
        p  {+"Some text"}
        a(href = "http://kotlinlang.org") {+"Kotlin"}
    }
}
```

A DSL for processes from scratsh

+

The process


Loan Request
Requestor
Validator
Start request
Review request
isAccepted
Sign contract
Notify rejection
Accepted
Rejected

Kotlin Features for DSL

Function literals with receiver

```
fun html(init: HTML.() -> Unit): HTML {
    val html = HTML()  // create the receiver object
    html.init()        // pass the receiver object to the lambda
    return html
}
```

```
fun html(init: HTML.() -> Unit): HTML
```

Name of the function

Object on which we do something

```
html {
    //call function on HTML object
}
```

@DslMarker

```
@DslMarker
annotation class ProcessDSLMarker
```

```
html {
    head {
        head {} // should be forbidden
    }
    // ...
}
```

```
The general rule:
- an implicit receiver may *belong to a DSL @X*
    if marked with a corresponding DSL marker annotation
- two implicit receivers of the same DSL
    are not accessible in the same scope
- the closest one wins
- other available receivers are resolved as usual,
     but if the resulting resolved call binds to such a receiver,
     it's a compilation error
```

Expression body

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

```
fun data(init: DataContainer.() -> Unit) {
    DataContainer(builder).init()
}
```

becomes

```
fun data(init: DataContainer.() -> Unit) = DataContainer(builder).init
```

Default arguments

```
// Bad
fun foo() = foo("a")
fun foo(a: String) { ... }

// Good
fun foo(a: String = "a") { ... }
```

```
automaticTask("name"){
}
```

becomes

```
automaticTask("name")
```

Infix functions

```
infix fun Int.shl(x: Int): Int { ... }

// calling the function using the infix notation
1 shl 2

// is the same as
1.shl(2)
```

```
text("type").describedAs("type of the loan")
```

becomes

```
text("type") describedAs "type of the loan"
```

Extension function

```
fun MutableList<Int>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}
```

```
actor("requester") mappedToUser "john"
```

becomes

```
"requester" mappedToUser "john"
```

Property with getter

```
val isEmpty: Boolean
    get() = this.size == 0
```

```
text("type") describedAs "type of the loan"
```

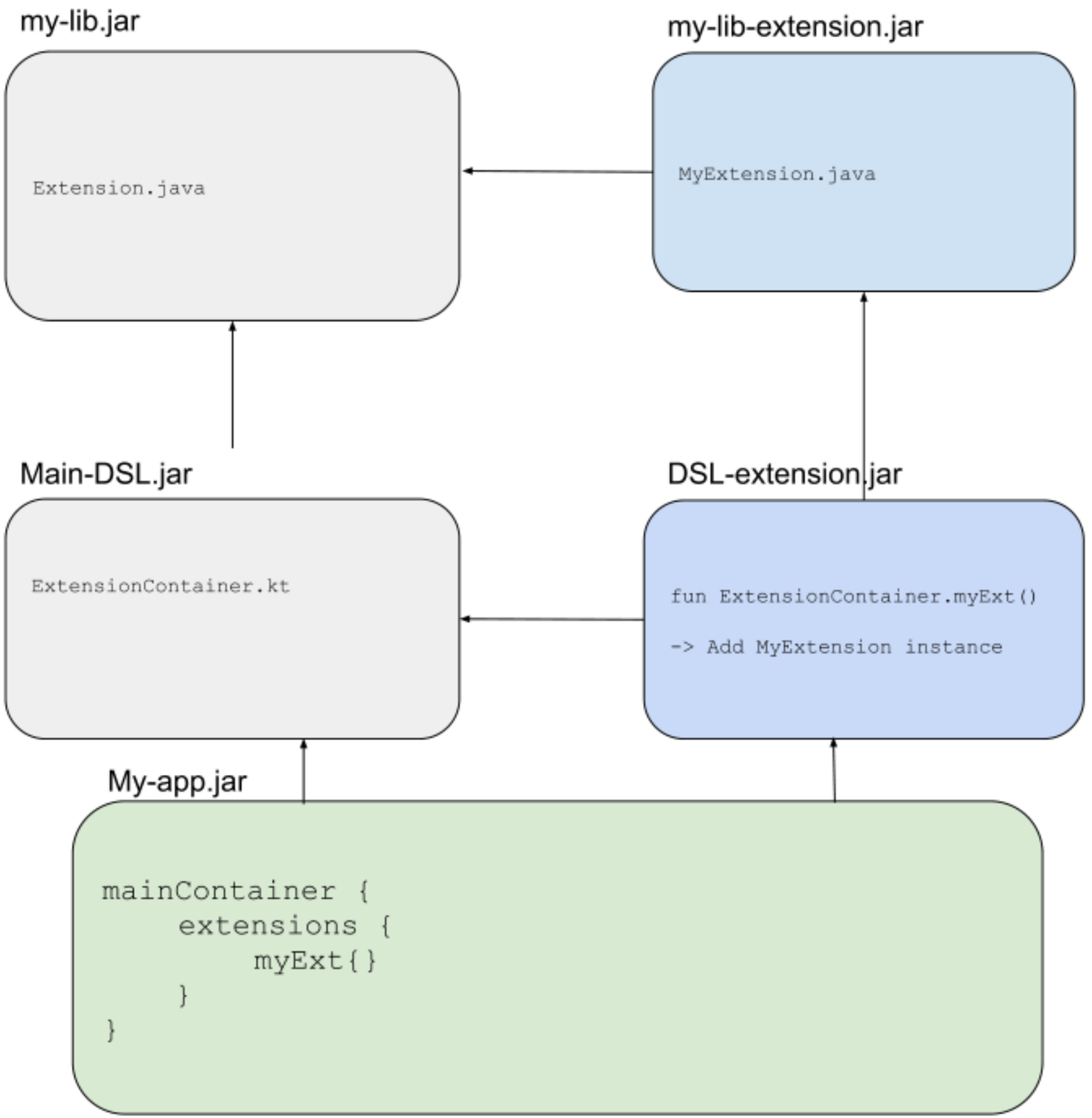
becomes

```
text named "type" describedAs "type of the loan"
```

"Dynamic DSL"

(name not official)

Leverage extension function to create extensible DSLs


my-lib.jar
Extension.java
my-lib-extension.jar
MyExtension.java
Main-DSL.jar
ExtensionContainer.kt
DSL-extension.jar
fun ExtensionContainer.myExt()
-> Add MyExtension instance
My-app.jar
mainContainer {
extensions {
myExt{}
}
}

DSL as file

Spring boot configuration

```
@ConditionalOnResource(resources = "classpath:process")
```

Programmatically execute script

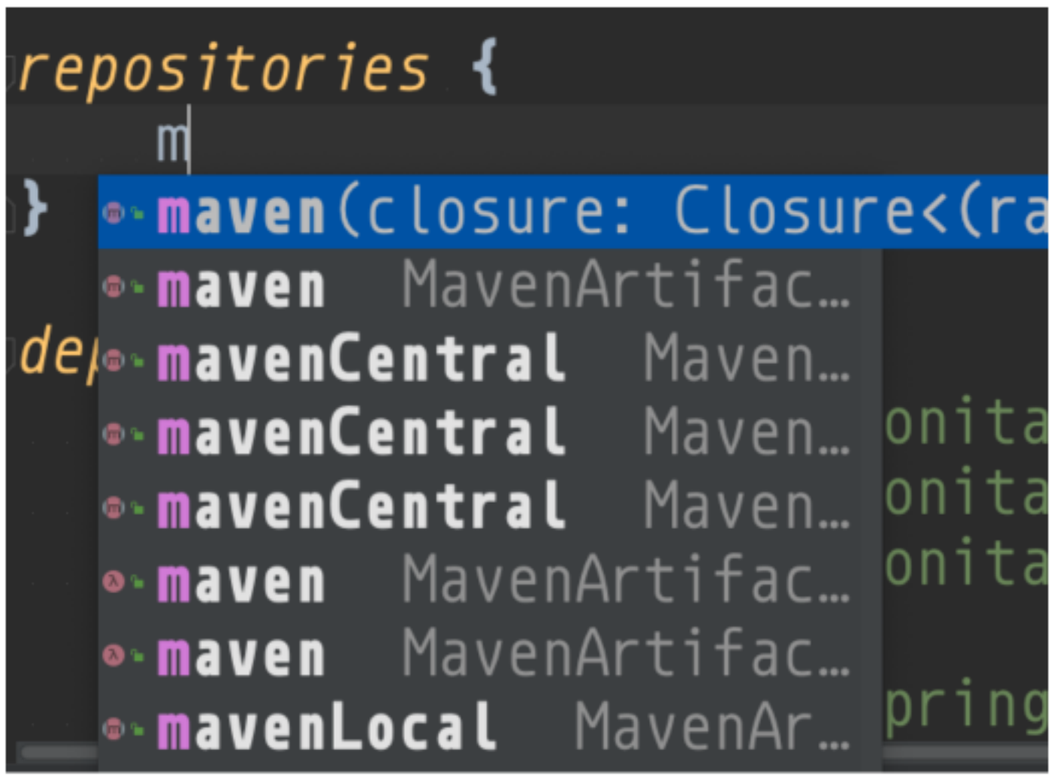
```
ScriptEngine scriptEngine = scriptEngineManager
            .getEngineByExtension("kts");
BusinessArchive bar = (BusinessArchive) scriptEngine.eval(content);
deployBar(apiClient, bar);
```

Why using a DSL?

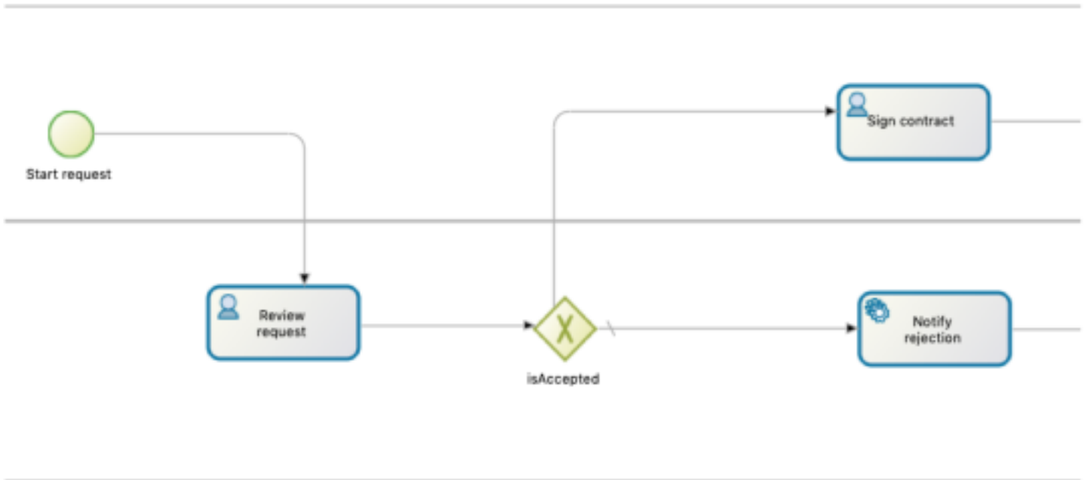
Validation

```
15  dependencies {
16      implemntation("org.bonitasoft.eng
17      implementation("org.bonitasoft.en
18      implementation("org.bonitasoft.co
19
20      implementation("org.springframewo
```

Discovery


repositories {
m
}
maven(closure: Closure<(ra
maven MavenArtifac…
mavenCentral Maven…
mavenCentral Maven…
mavenCentral Maven…
maven MavenArtifac…
maven MavenArtifac…
mavenLocal MavenAr…
onita
onita
onita
pring

Speak the language of your domain




```
businessArchive {
    process("Request Loan", "1.0") {
    val requester = initiator("requester")
    val validator = actor("validator")
    data {
        boolean named "accepted"
    }
    contract {
        text named "type"
        integer named "amount"
    }
    userTask("Review request", validator) {
        contract {
            boolean named "accept"
            text named "reason"
        }
    }
    userTask("Sign contract", requester)
    exclusiveGateway("isAccepted")
    automaticTask("Notify reject")
    transitions {
        normal from "Review request" to "isAccepted"
        conditional("accepted") from "isAccepted" to "Sign contract"
        default from "isAccepted" to "Notify reject"
    }

  }
}
```

Feedbacks on writing a DSL

Use Kotlin… 90% of the job is done

Write how it should look like first

```
process {
    data {
        text named "myData"
    }
}
```

- Don’t try to use all features
- Focus on usability and discovery

- Throw away your POCs
- Once your confortable rewrite everything

A good way to learn kotlin

Thank you

bonitasoft/bonita-engine

baptistemesta.github.io/process-kotlin-dsl-slides

github.com/bonitasoft-labs/process-kotlin-dsl-request-loan

@bonitasoft @baptistemesta