

Kotlin meets Gadsu

Christoph Pickl

Kotlin Vienna Meetup – 2017-01-31



- 1 **Introduction to Gadsu**
- 2 **Kotlin in the wild**
- 3 **Code “Schmankerln”**
- 4 **Lessons Learned**

Introduction to Gadsu



This is a cat.



This is Shiatsu . . .



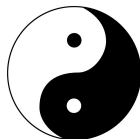
... so is this.



Gadse
+ Shiatsu
= **Gadsu**



- Massage, Physiotherapy, Bodywork
- Acupuncture, Meridiantherapy
- Nervous system stimulation
- Based on the **T**raditional **C**hinese **M**edicine
 - Concept of **Qi** flowing through the body and everything
 - Every aspect of the human grouped into **5 Elements**
 - Body and mind seen as a unit, not separated from each other



Taiji Symbol, Theory of Yin and Yang



Features:

- Client database
- Manage medical records
- Generate reports
- Google integration
- Auto update, auto backup

Roadmap:

- Pain indicator, 5 Elements
- Statistics
- TCM intelligence
- Doodle integration
- Invoicing



- Gradle
- Swing
- Guice
- Spring JDBC
- HSQLDB + Flyway
- Jasper, Pdfbox
- Freemarker
- TestNG, Mockito, Hamcrest
- UISpec4J
- *Initial implementation used Kotlin 0.6 ;)*

Let's have a look . . .

Kotlin in the wild



```
apply plugin: "kotlin"
buildscript {
    ext.kotlin_version = '1.0.6'
    dependencies {
        classpath "org.jetbrains.kotlin:kotlin-gradle-plugin:$kotlin_version"
    }
}
dependencies {
    compile "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
    compile "org.jetbrains.kotlin:kotlin-reflect:$kotlin_version"
}
```

Travis CI GmbH

christophpickl / gadsu

build: passing

Current Branches Build History Pull Requests More options

✓ master minor changes (task tags) -o- #311 passed

- Commit e472ebb
- Compare 9d7a616...e472ebb
- Branch master

⌚ Elapsed time 3 min 3 sec

📅 a day ago

👤 Christoph Pickl authored and committed

Job log View config

Raw log

```
1 Worker information
6 Build system information
73
74 $ export DEBIAN_FRONTEND=noninteractive
118 $ git clone --depth=50 --branch=master
129
130 This job is running on container-based infrastructure, which does not allow use of 'sudo',
```

travis-ci.org



```
language: kotlin
sudo: false
jdk:
  - oraclejdk8
before_install:
  - "chmod +x gradlew"
  - "export DISPLAY=:99.0"
  - "sh -e /etc/init.d/xvfb start"
script:
  - "./gradlew test ..."
notifications:
  email:
    - "MLtravis@gadsu.com"
```


codecov.io

gh : christophickl / gadsu

[Docs](#)
[Support](#)
[Sign up](#)

#87 implemented most of the confirmer logic

christophickl a day ago ✓

9d7a616 master

[Diff](#)
[Files](#)
[Builds](#)
[Graphs](#)

Showing 9 of 14 changed files with **56.00%** of changed lines covered. [View details](#)

-- / kotlin / at / cpickl / gadsu / mail / module.kt		1	0	0	100%
@@ -12,6 +12,7 @@					
12	12				
13	13	bind(GMailApi::class.java).to(GMailApiImpl::class.java).`in` (Scopes			
14	14	bind(MailSender::class.java).to(MailSenderImpl::class.java).`in` (Sc			
	15 +	bind(AppointmentConfirmationner::class.java).to(AppointmentConfirmat			
15	16				
16	17	bind(MailView::class.java).to(MailSwingView::class.java).`in` (Scope			
17	18	bind(MailController::class.java).asEagerSingleton()			
@@ -18 +19 @@					

codecov.io

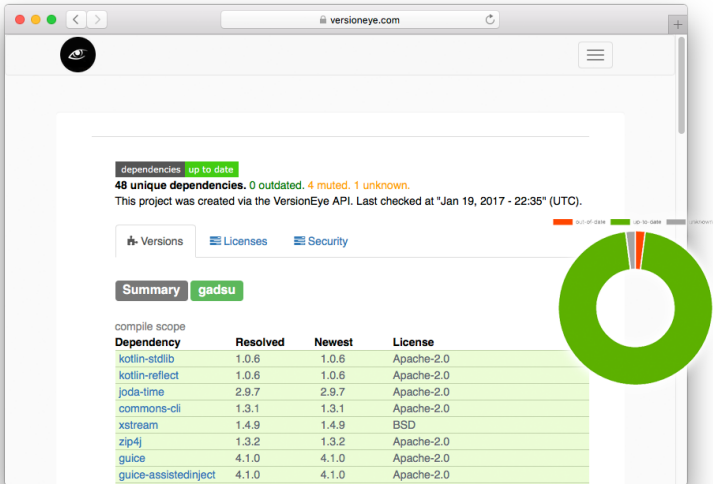


Gradle Configuration:

```
plugins {  
    id 'jacoco'  
    id 'com.github.kt3k.coveralls'  
}  
jacocoTestReport {  
    reports {  
        xml.enabled = true  
    }  
}}
```

Travis Configuration:

```
script:  
- "./gradlew ... jacocoTestReport ..."  
after_success:  
- bash <(curl -s https://codecov.io/bash)
```



versioneye.com



Gradle Configuration:

```
plugins {  
    id "org.standardout.versioneye"  
    version "1.4.0"  
}
```

Gradle Properties:









```
versioneye.projectid=572880644a0f...00b78206
```

Travis Configuration:

```
script:  
- "./gradlew ... versioneye-update ..."
```



Display coverage data via the [Codecov Browser Extension](#):



 christophpickl	#87 send confirmation mail UI	Latest commit df2a2c3 19 days ago
..		
 view	#87 send confirmation mail UI	19 days ago
 xprops	#71 reuse render text logic	a month ago
 events.kt	#76 client CRUD options context menu vs menu bar	a month ago 50.00%
 model.kt	#87 send confirmation mail UI	19 days ago 91.53%
 module.kt	refactoring xprops; db test infra	9 months ago 100.00%
 persistence.kt	#78 new client fields (main objective, symptoms, syndroms, five eleme...	a month ago 75.56%
 service.kt	#76 client CRUD options context menu vs menu bar	a month ago 66.04%

```
47 // extension methods
48 fun ImageIcon.toMyImage(): MyImage = ImageIconImage(this)
49 fun BufferedImage.toMyImage(): MyImage = ImageIconImage(ImageIcon(this))
50 fun File.toMyImage(): MyImage = FileImage(this)
51 fun String.toMyImage(): MyImage = ClasspathImage(this)
52 fun ByteArray.toMyImage(): MyImage = this.readBufferedImage().toMyImage()
53
54 val Gender.defaultImage: MyImage get() =
55     when(this) {
56         Gender.MALE -> MyImage.DEFAULT_PROFILE_MAN
57         Gender.FEMALE -> MyImage.DEFAULT_PROFILE_WOMAN
58         else -> MyImage.DEFAULT_PROFILE_ALIEN
59     }
59 ..
```



- Auto convert from Java to Kotlin
- False unused warning on companion extensions : (
- TODO3
- Automatic replace

```
val text = JTextField()  
val width = text.getWidth()
```

 Use property access syntax 

Code Schmankerln



The *neutral* **domain object**:

```
package at.cpickl.gadsu.client

data class Client(
    val id: String,
    val name: String
)
```




Persistence specific functionality:

```
package at.cpickl.gadsu.persistence

data class ClientDbc(
    val TXT_ID: String,
    val TXT_NAME: String
)

fun Client.toDbc() =
    ClientDbc(id, name)

class ClientRepo {
    fun save(client: Client) {
        saveSomewhere(client.toDbc())
    }
}
```



Add a **fluent API** to an existing classes:

```
fun <T : JComponent> T.bold(): T {  
    font = font.deriveFont(Font.BOLD)  
    return this  
}  
  
val myLabel = JLabel("text").bold()  
val myTextField = JTextField("text").bold()  
val myTextArea = JTextArea("text").bold()
```

Extension Properties #1



Possible replacement of common **test factories**:

```
package at.cpickl.gadsu.test

val Client.Companion.testee1: Client
    get() = Client(
        id = "",
        name = "Max Muster"
    )
```

Sadly requires to have some *placeholder*:

```
package at.cpickl.gadsu.client

data class Client( ... ) {
    companion object {}
}
```



Use those testees in your **tests**:

```
package at.cpickl.gadsu.test

@Test class ClientIT {

    @Inject lateinit var repo: ClientRepo

    fun 'reference test scoped testee'() {
        repo.save(
            Client.testee1.copy(name = "Otto")
        )
        // ... assertions ...
    }
}
```



TODO TODO TODO TODO TODO TODO TODO

Java 8

```
Collection, filter, map, STREAM!, collector
```

Kotlin (implicit `it` variable)

```
Collection, filter, map
```



Sometimes I feel so lazy ...



Given there is a *very expensive* expensiveInit() method:

```
public class NaiveSingleton {  
    private Object lazyField = null;  
  
    public Object getLazyField() {  
        if (lazyField == null) {  
            lazyField = expensiveInit();  
        }  
        return lazyField;  
    }  
}
```



```
public class Java8 {  
    private Supplier<Object> lazyField=() -> {  
        Object value = expensiveInit();  
        lazyField = () -> value;  
        return value;  
    };  
  
    public Object getLazyField() {  
        return lazyField.get();  
    }  
}
```




```
class LazyKotlin {  
    val lazyField by lazy {  
        expensiveInit()  
    }  
}  
  
// part of stdlib:  
fun <T> lazy(initializer: ()->T): Lazy<T> =  
    SynchronizedLazyImpl(initializer)
```

Thanks to type inference, we don't need to specify types explicitly.



Given the existing classes:

```
interface Step {  
    fun take()  
}  
  
class StepImpl : Step {  
    override fun take() {}  
}
```

We now want some new service to implement this interface, but **delegate** all its methods to the StepImpl implementation.



```
public class MyService implements Step {  
  
    private final Step step;  
  
    public MyService(Step step) {  
        this.step = step;  
    }  
  
    @Override public void take() {  
        step.take();  
    }  
}
```



```
class MyService(step: Step) : Step by step
```

Standard delegates in Kotlin:

- lazy
- observable
- map properties

Lessons Learned



- Mostly same as for Java
- Build system support (gradle with kotlin coming!)
- Static code analysis tools missing
- Syntax highlighting mostly missing



- Null handling is a MUST!
- Extension methods (for better auto completion)
- (Constructor) Properties
- Lambdas done properly
- (Local) Type inference
- Named and default arguments
- Compact syntax: No semicolon, new
- Data classes

Kotlin is a great language ... but!



- Data classes are still somehow restricted in their usage (v1.1)
- Paradigm shift of final-by-default clashes with existing libs
- Requires developers to be more disciplined
 - Several classes in one (big) file gets common
 - Explicit type declaration for documentation
 - Overuse of single-expression functions
 - Overuse of functionals like `apply{}`



```
class MainPanel : JPanel() {  
    init {  
        val subPanel = JPanel()  
        subPanel.background = Color.RED  
        add(subPanel)  
    }  
}
```

We can refactor this to get rid of the **variable reference**.

Apply – This is not this anymore!



```
class MainPanel : JPanel() {  
    init {  
        JPanel().apply {  
            background = Color.RED  
            add(this)  
        }  
    }  
}
```

IllegalArgumentException: adding container's parent to itself

The solution is to use: `this@MainPanel.add(this)`

But what happens if there are **two nested** JPanels?!



- Visit the website:
<https://github.com/christophpickl/gadsu>
- \LaTeX sources of the slides:
https://github.com/christophpickl/gadsu_meetup
- Kotlin-Vienna Usergroup:
<https://www.meetup.com/Kotlin-Vienna>