

Kotlin meets Gadsu

Christoph Pickl

Kotlin Vienna Meetup – 2017-01-31

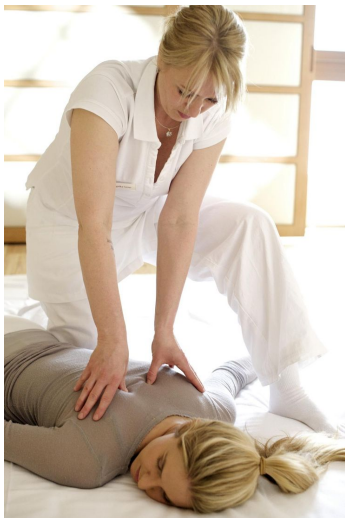


- 1 **Introduction to Gadsu**
- 2 **Kotlin in the wild**
- 3 **Code “Schmankerln”**
- 4 **Lessons Learned**

Introduction to Gadsu



This is a cat.



This is Shiatsu . . .



... so is this.



Gadse
+ Shiatsu
= **Gadsu**

Shiatsu is ...

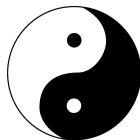


... Japanese! Food! Yam yam!





- 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
 - Body and mind seen as a unit, not separated from each other



Taiji Symbol, Theory of Yin and Yang

Volunteers?!



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 ;)*

Gadsu


Datei Bearbeiten Ansicht Berichte Development




Anna Nym
Behandlungen: 0




Chuck Norris
Behandlungen: 0



Maxi Mustermann
Behandlungen: 3
Wiedersehen: Mi, 10.08., 20:05



Pam Anderson
Behandlungen: 0



Queen Liz
Behandlungen: 0

Allgemein Texte TCM

Basisdaten

Vorname

Nachname

Spitzname

Geschlecht

Geburtstag ...

Sternzeichen

Geburtsort

Herkunft

Beziehungsstatus

Beruf

Kinder

Hobbies

Erstellt am

Kontaktdaten

Mail

Telefon

Strasse

PLZ

Stadt

Mi, 10.08., 20:05

Mi, 17.08., 20:05

Neuen Termin erstellen

3. Mittwoch, 27.07.16, 20:00 Uhr

2. Mittwoch, 20.07.16, 20:00 Uhr

1. Mittwoch, 13.07.16, 20:00 Uhr

Neue Behandlung erstellen

Notiz

Meine supi wuzi Anmerkung.

Neuen Klienten anlegen Speichern Abbrechen

Gadsu got something like 30,000 LoC :)


Let's see some app . . .

Kotlin in the wild



```
1 apply plugin: "kotlin"
2 buildscript {
3     ext.kotlin_version = '1.0.6'
4     dependencies {
5         classpath "org.jetbrains.kotlin:
6             kotlin-gradle-plugin:$kotlin_version"
7     }
8 }
9 dependencies {
10     compile "org.jetbrains.kotlin:
11         kotlin-stdlib:$kotlin_version"
12     compile "org.jetbrains.kotlin:
13         kotlin-reflect:$kotlin_version"
14 }
```




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

⚙ 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



```
1 language: kotlin
2 sudo: false
3 jdk:
4   - oraclejdk8
5 before_install:
6   - "chmod +x gradlew"
7   - "export DISPLAY=:99.0"
8   - "sh -e /etc/init.d/xvfb start"
9 script:
10   - "./gradlew test ..."
11 notifications:
12   email:
13     - "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
14 14
15 15 +
16 16
17 17
18 18
@@ -18 +19 @@

bind(GMailApi::class.java).to(GMailApiImpl::class.java).`in`(Scopes
bind(MailSender::class.java).to(MailSenderImpl::class.java).`in`(Sc
bind(AppointmentConfirmationner::class.java).to(AppointmentConfirmat
bind(MailView::class.java).to(MailSwingView::class.java).`in`(Scope
bind(MailController::class.java).asEagerSingleton()

codecov.io

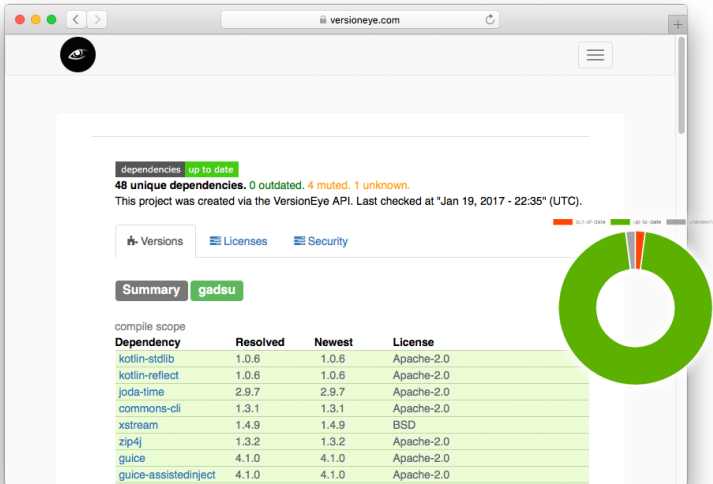


Gradle Configuration:

```
1 plugins {  
2     id 'jacoco'  
3     id 'com.github.kt3k.coveralls'  
4 }  
5 jacocoTestReport {  
6     reports {  
7         xml.enabled = true  
8     }}  
9 }
```

Travis Configuration:

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



versioneye.com



Gradle Configuration:

```
1 plugins {  
2     id "org.standardout.versioneye"  
3     version "1.4.0"  
4 }
```

Gradle Properties:









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

Travis Configuration:

```
1 script:  
2 - "./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 ..
```

Code Schmankerln

Null Handling

Trust no one!



```
1 @Data
2 class Contact {
3     String mail;
4 }
5 @Data
6 class Client {
7     Contact contact;
8     public boolean hasContact() {
9         return contact != null;
10    }
11 }
12
13 if (client.hasContact()) {
14     process(client.getContact());
15 }
```

Explicit handling in Kotlin



```
1 data class Contact(val mail: String)
2 data class Client(val contact: Contact?)
3
4 client.contact.ifNotNull(::process)
5 client.contact?.mail ?: "default@mail.at"
```

Write your own, custom *if-loop*:

```
1 fun <T> T?.ifNotNull(action: (T) -> Unit):
2     T? {
3     if (this == null) {
4         return null
5     }
6     action(this)
7     return this
8 }
```

Extension Methods



Implement a straight-forward, *clean domain object*:

```
1 package at.cpickl.gadsu.client
2
3 data class Client(
4     val id: String,
5     val name: String
6 )
```



Persistence specific functionality:

```
1 package at.cpickl.gadsu.persistence
2
3 data class ClientDbc(
4     val TXT_ID: String,
5     val TXT_NAME: String
6 )
7 fun Client.toDbc() =
8     ClientDbc(id, name)
9
10 class ClientRepo {
11     fun save(client: Client) {
12         saveSomewhere(client.toDbc())
13     }
14 }
```



Or for those masochists out there who prefer **nullables**:

```
1 package at.cpickl.gadsu.persistence
2
3 fun Client?.toDbo() =
4     if (this == null) null
5     else ClientDbo(id, name)
6
7 val client: Client? = ....
8 val dbo = client?.toDbo()
9     ?: ClientDbo.defaultInstance()
```



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

```
1 fun <T : JComponent> T.bold(): T {  
2     font = font.deriveFont(Font.BOLD)  
3     return this  
4 }  
5  
6 val myLabel = JLabel("text").bold().italic()  
7 val myTextField = JTextField("text").bold()  
8 val myTextArea = JTextArea("text").bold()  
9  
10 val panel = JPanel().transparent()
```


Extension Properties



Possible replacement of common **test factories**:

```
1 package at.cpickl.gadsu.test
2
3 val Client.Companion.testee1: Client
4     get() = Client(
5         id = "",
6         name = "Max Muster"
7     )
```

Unfortunately Kotlin requires to have some *placeholder*:

```
1 package at.cpickl.gadsu.client
2
3 data class Client( ... ) {
4     companion object {}
5 }
```



Use those testees in your **tests**:

```
1 package at.cpickl.gadsu.test
2
3 @Test class ClientIT {
4
5     @Inject lateinit var repo: ClientRepo
6
7     fun 'reference test scoped testee'() {
8         repo.save(
9             Client.testee1.copy(name = "Otto")
10        )
11        // ... assertions ...
12    }
13 }
```

λ

Functional done right!



Java 8 being veeery verbose as always:

```
1 List<Integer> numbers = asList(1, 2, 3);
2 List<String> numbers2 = numbers
3     .stream()
4     .filter(i -> i % 2 == 0)
5     .map(Object::toString)
6     .collect(toList());
```

Kotlin (implicit it variable)

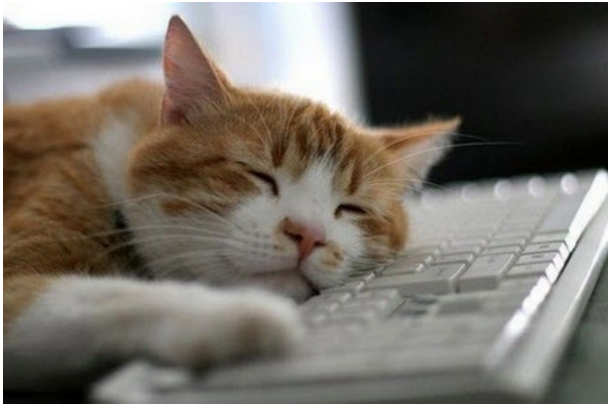
```
1 val numbers = listOf(1, 2, 3)
2 val numbers2 = numbers
3     .filter { it % 2 == 0 }
4     .map(Int::toString)
```



Java sometimes behaves like good old *Aunt Chatty*.



- `fun <T, R> T.let(f: (T) -> R): R = f(this)`
- `fun <T> T.apply(f: T.() -> Unit): T { f(); return this }`
- `fun <T, R> with(receiver: T, f: T.() -> R): R = receiver.f()`
- `fun <T, R> T.run(f: T.() -> R): R = f()`



Sometimes I feel so lazy ...



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

```
1 public class NaiveSingleton {
2     private Object lazyField = null;
3
4     public Object getLazyField() {
5         if (lazyField == null) {
6             lazyField = expensiveInit();
7         }
8         return lazyField;
9     }
10 }
```



```
1 public class Java8 {  
2     private Supplier<Object> lazyField=() -> {  
3         Object value = expensiveInit();  
4         lazyField = () -> value;  
5         return value;  
6     };  
7  
8     public Object getLazyField() {  
9         return lazyField.get();  
10    }  
11 }
```



```
1 class LazyKotlin {  
2     val lazyField by lazy {  
3         expensiveInit()  
4     }  
5 }  
6  
7 // part of stdlib:  
8 fun <T> lazy(initializer: ()->T): Lazy<T> =  
9     SynchronizedLazyImpl(initializer)
```

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

Delegates



Given the existing classes:

```
1 interface Step {  
2     fun take()  
3 }  
4  
5 class StepImpl : Step {  
6     override fun take() {}  
7 }
```

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



```
1 public class MyService implements Step {  
2  
3     private final Step step;  
4  
5     public MyService(Step step) {  
6         this.step = step;  
7     }  
8  
9     @Override public void take() {  
10         step.take();  
11     }  
12 }
```



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

Standard delegates in Kotlin:

- **lazy**
- observable
- map properties



Let's do the Swing!



```
1 class ClientTabMain(modifications, ...) {
2     val fields = Fields<Client>(modifications)
3     val inpNote = fields.newTextArea(
4         "Notiz", { it.note },
5         ViewNames.Client.InputNote, bus)
6     init {
7         add(VFillFormPanel()).apply {
8             addFormInput(inpNote)
9         }}
10    fun isModified(client: Client) =
11        fields.isAnyModified(client)
12    fun updateFields(client: Client) {
13        fields.updateAll(client)
14    }
15 }
```



Dispatch events via EventBus and subscribe in Controller:

```
1 open class ClientViewController @Inject
2   constructor(val bus: EventBus, ...) {
3
4   @Subscribe open fun onAppStartupEvent(
5     event: AppStartupEvent) {
6     reinitClients()
7     bus.post(ChangeMainContentEvent(view))
8     ...
9   }
```



```
1 @Test(groups = arrayOf("uiTest"))
2 class ClientUiTest : UiTest() {
3     fun 'save client should change UI'() {
4         val client = Client.unsavedValidInstance()
5         with(driver) {
6             assertSaveButtonTextEquals("Neu anlegen")
7
8             saveClient(client)
9             assertSaveButtonTextEquals("Speichern")
10
11             assertListContains(client)
12             assertListSelected(client)
13             ...
14 }
```



```
1 class ClientDriver(  
2     test: UiTest, window: Window) {  
3  
4     val list = window.getListBox(  
5         ViewNames.Client.List)!!  
6     val createButton = window.getButton(  
7         ViewNames.Client.CreateButton)!!  
8  
9     fun assertNoChangesDetected() {  
10         test.assertThat(  
11             test.not(saveButton.isEnabled))  
12         test.assertThat(  
13             test.not(cancelButton.isEnabled))  
14     }  
15     ...
```

Let's see some code . . .

Lessons Learned



- *Mostly* as good as for Java
- **IntelliJ** support feels already superb
- **Build** system support (Gradle Script Kotlin!)
- Static code analysis tools **missing**
- Syntax highlighting mostly **missing**



- 1 **Null handling** is a MUST!



Remember me?!

8 reasons to use Kotlin



- 1 **Null handling** is a MUST!
- 2 **Extension methods** for improved auto completion
- 3 (Constructor) Properties
- 4 **Lambdas** done properly
- 5 (Local) Type **inference**
- 6 Named and default **arguments**
- 7 Compact **syntax**: No semicolon, new
- 8 **Data** classes replaces Lombok

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 young padawan 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{}`

Apply – Who is this?



```
1 class MainPanel : JPanel() {  
2     init {  
3         val subPanel = JPanel()  
4         subPanel.background = Color.RED  
5         add(subPanel)  
6     }  
7 }
```

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

Apply – This is not this anymore!



```
1 class MainPanel : JPanel() {  
2     init {  
3         JPanel().apply {  
4             background = Color.RED  
5             add(this)  
6         }  
7     }  
8 }
```

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?!
- PS: Kotlin 1.1 will come with a new function called `also()`

```
1 JPanel().also {  
2     this.add(it)  
3 }
```



- 1 Compiler plugins (open-by-default, no-arg ctor)
- 2 Coroutines
- 3 Type aliases
- 4 Type inference for getters
- 5 Bound callable references
- 6 Local delegated properties & Inline properties
- 7 Inheritance for data classes
- 8 Subclasses of sealed classes in the same file
- 9 Destructuring in lambdas
- 10 Underscore for unused parameters
- 11 Underscore in numeric literals



- Visit Gadsu **website**:
<https://github.com/christophpickl/gadsu>
- \LaTeX **sources** of slides:
https://github.com/christophpickl/gadsu_meetup
- Kotlin Vienna **Meetup**:
<https://www.meetup.com/Kotlin-Vienna>
- Kotlin **Slack** Channel:
<https://kotlinlang.slack.com/messages/vienna/>

One more thing . . .

OBLIGATORY

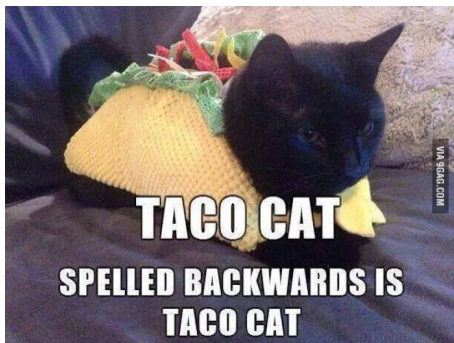
MEME

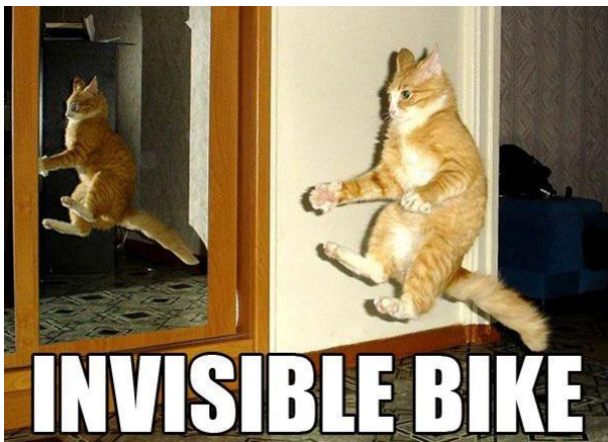
memegenerator.net











INVISIBLE BIKE



