otlin meets Gadsu

Christoph Pickl

Kotlin Vienna Meetup – 2017-01-31

Agenda

1. **Introduction to Gadsu**
2. Kotlin in the wild
3. Code "*Schmankerln*"
4. Lessons Learned

Agenda

1. Introduction to Gadsu
2. **Kotlin in the wild**
3. Code "*Schmankerln*"
4. Lessons Learned

Agenda

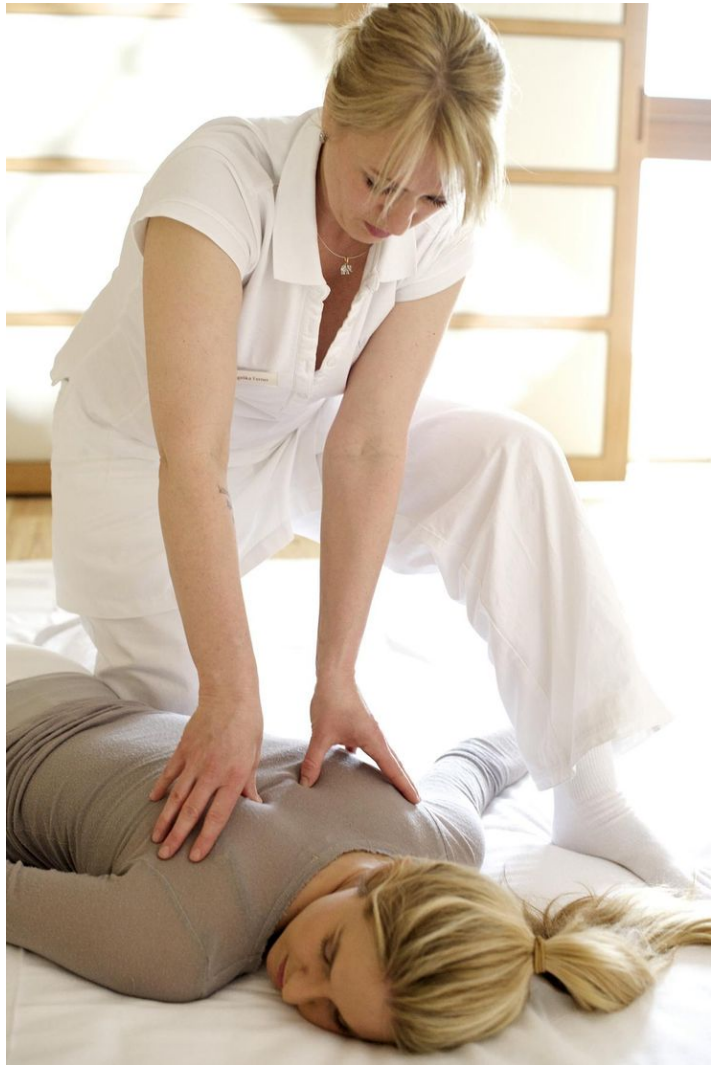
1. Introduction to Gadsu
2. Kotlin in the wild
3. **Code "SchmankerIn"**
4. Lessons Learned

Agenda

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.

Gadsu is . . .

Gadsu is . . .

Gadse

Gadsu is . . .

Gadse

\+ Shiat**su**

Gadsu is . . .

Gadse

\+ Shiat**su**

= **Gadsu**

Shiatsu is . . .

. . . Japanese!

Shiatsu is . . .

. . . Japanese! Food!

Shiatsu is . . .

. . . Japanese! Food! Yam yam!

Shiatsu is . . .

- Massage, Physiotherapy, Bodywork
- Acupuncture, Meridiantherapy
- Nervous system stimulation

Shiatsu is . . .

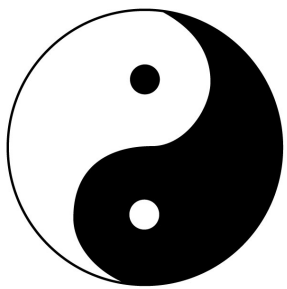
- Massage, Physiotherapy, Bodywork
- Acupuncture, Meridiantherapy
- Nervous system stimulation

- 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

Shiatsu is . . .

- 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

Gadsu can . . .

Features:

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

Gadsu can . . .

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

Technology Stack

- Gradle

Technology Stack

- Gradle
- Swing

Technology Stack

- Gradle
- Swing
- Guice

Technology Stack

- Gradle
- Swing
- Guice
- Spring JDBC, HSQLDB, Flyway

Technology Stack

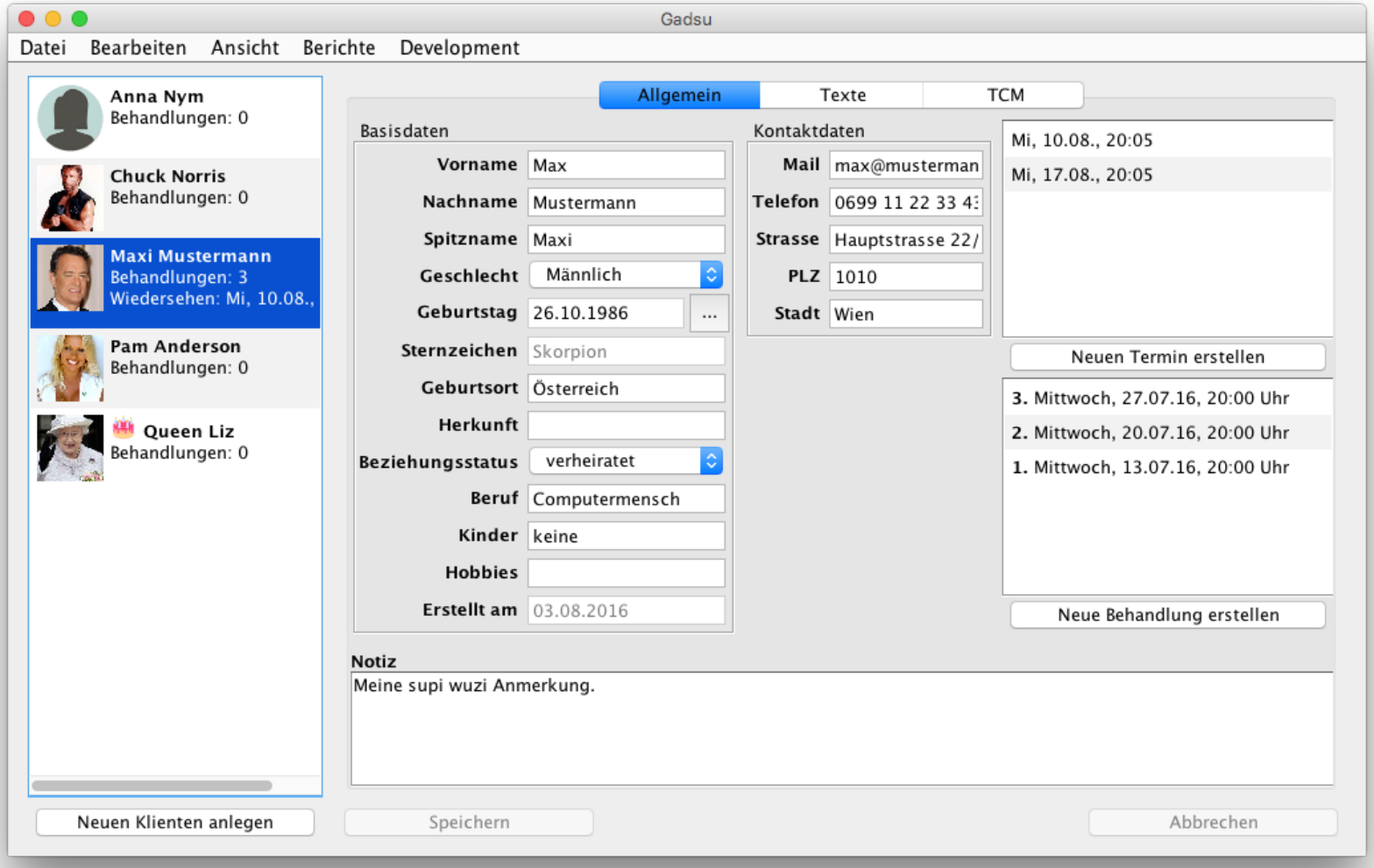
- Gradle
- Swing
- Guice
- Spring JDBC, HSQLDB, Flyway
- Jasper, Pdfbox, Freemarker

Technology Stack

- Gradle
- Swing
- Guice
- Spring JDBC, HSQLDB, Flyway
- Jasper, Pdfbox, Freemarker
- TestNG, Mockito, Hamcrest, UISpec4J

Technology Stack

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



Gadsu got something like 30,000 LoC :)

Let's see some app . . .

otlin in the wild

build.gradle

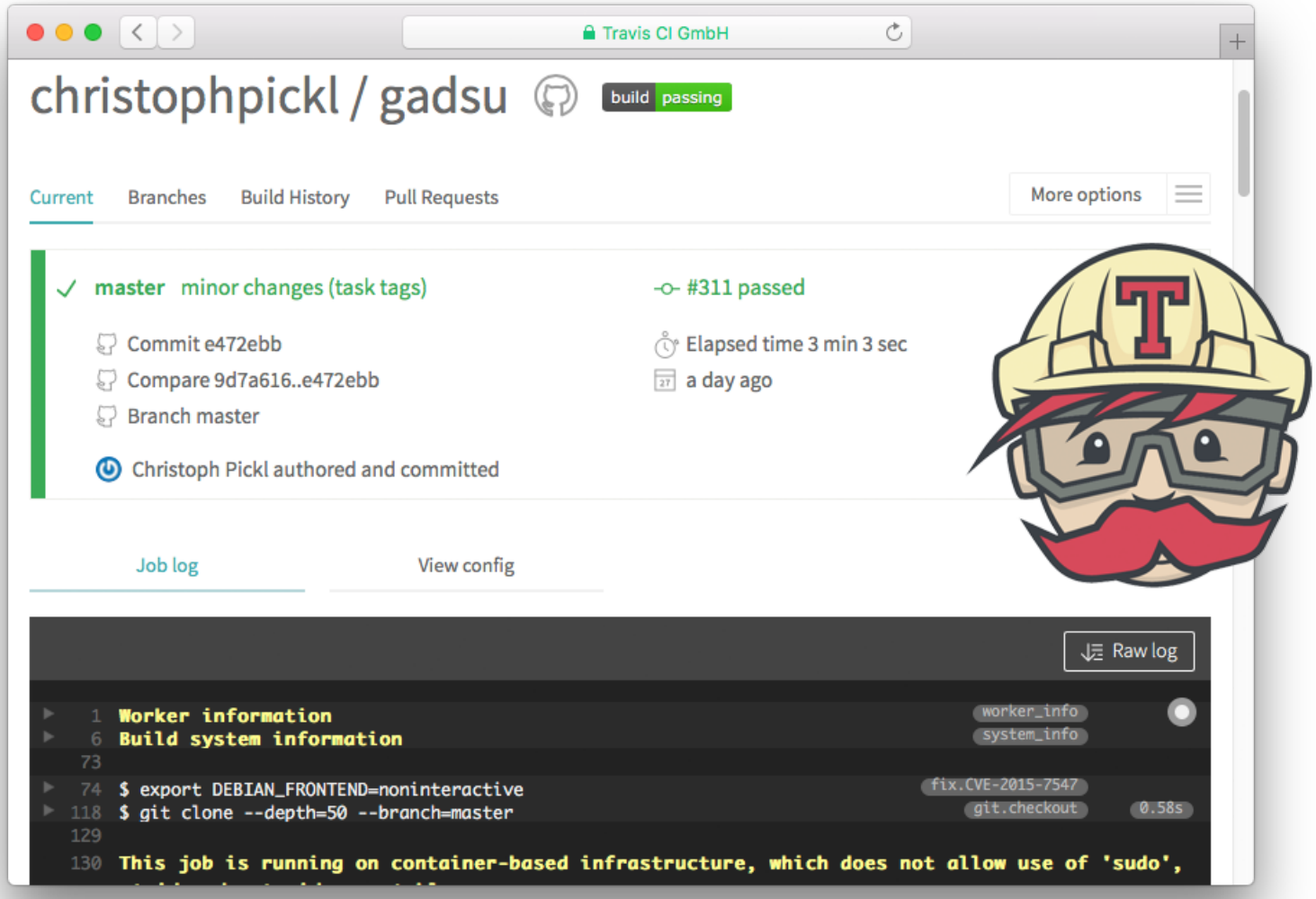
```
1 apply plugin: "kotlin"
```

build.gradle

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

build.gradle

```
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.org

.travis.yml

```
1 language: kotlin
```

.travis.yml

```
language: kotlin
sudo: false
```

.travis.yml

```
language: kotlin
sudo: false
jdk:
  - oraclejdk8
```

.travis.yml

```yaml
language: kotlin
sudo: false
jdk:
  - oraclejdk8
before_install:
  - "chmod +x gradlew"
```

.travis.yml

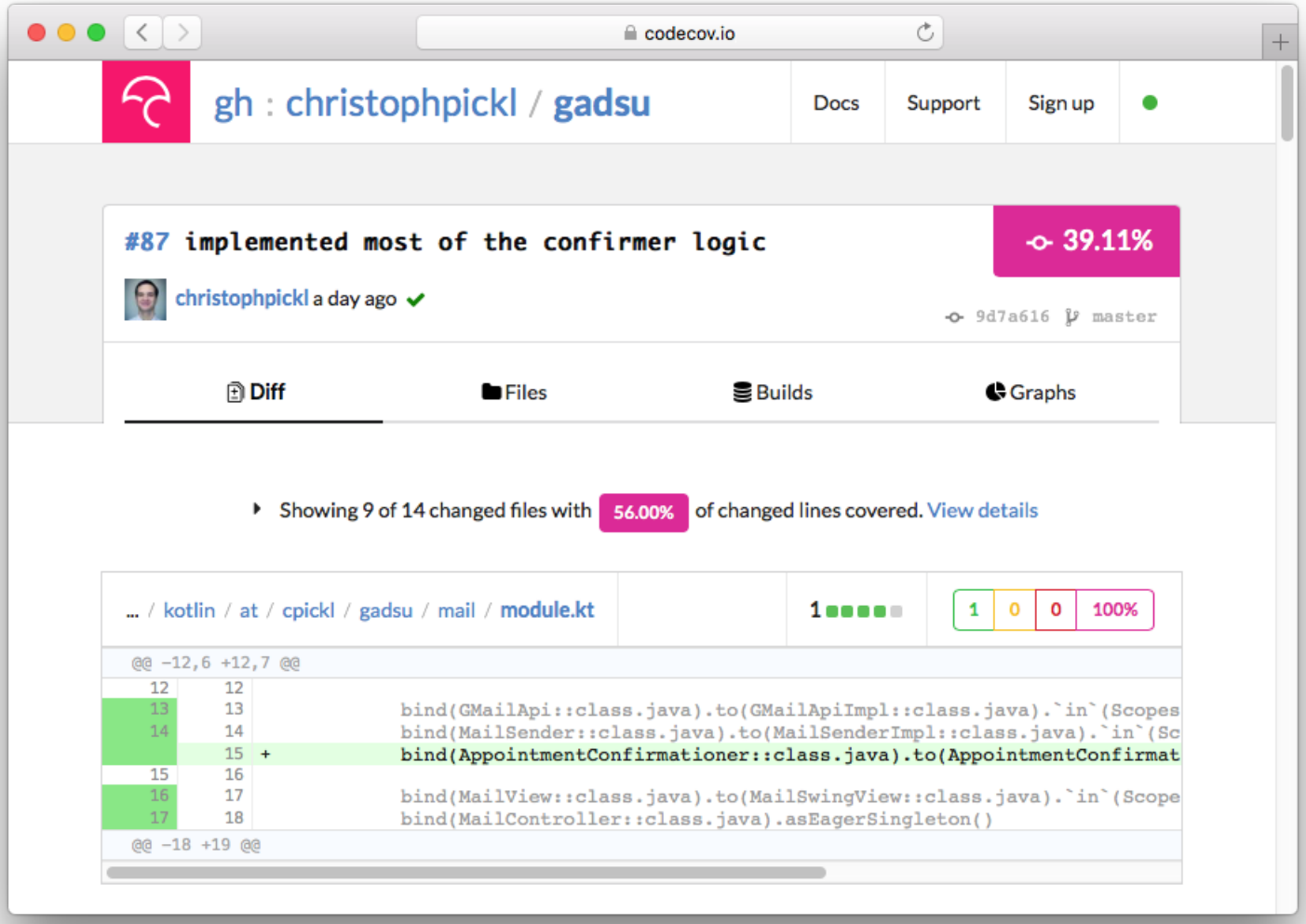
```
language: kotlin
sudo: false
jdk:
  - oraclejdk8
before_install:
  - "chmod +x gradlew"
  - "export DISPLAY=:99.0"
  - "sh -e /etc/init.d/xvfb start"
```

.travis.yml

```
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 ..."
```

.travis.yml

```
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 : christophpickl / gadsu
Docs
Support
Sign up
#87 implemented most of the confirmer logic
39.11%
christophpickl 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
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(AppointmentConfirmationer::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

Codecov

Gradle Configuration:

```
plugins {
  id 'jacoco'
  id 'com.github.kt3k.coveralls'
}
```

Codecov

Gradle Configuration:

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

Codecov

Gradle Configuration:

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

Travis Configuration:

```
script:
  - "./gradlew ... jacocoTestReport ..."
```

Codecov

Gradle Configuration:

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

Travis Configuration:

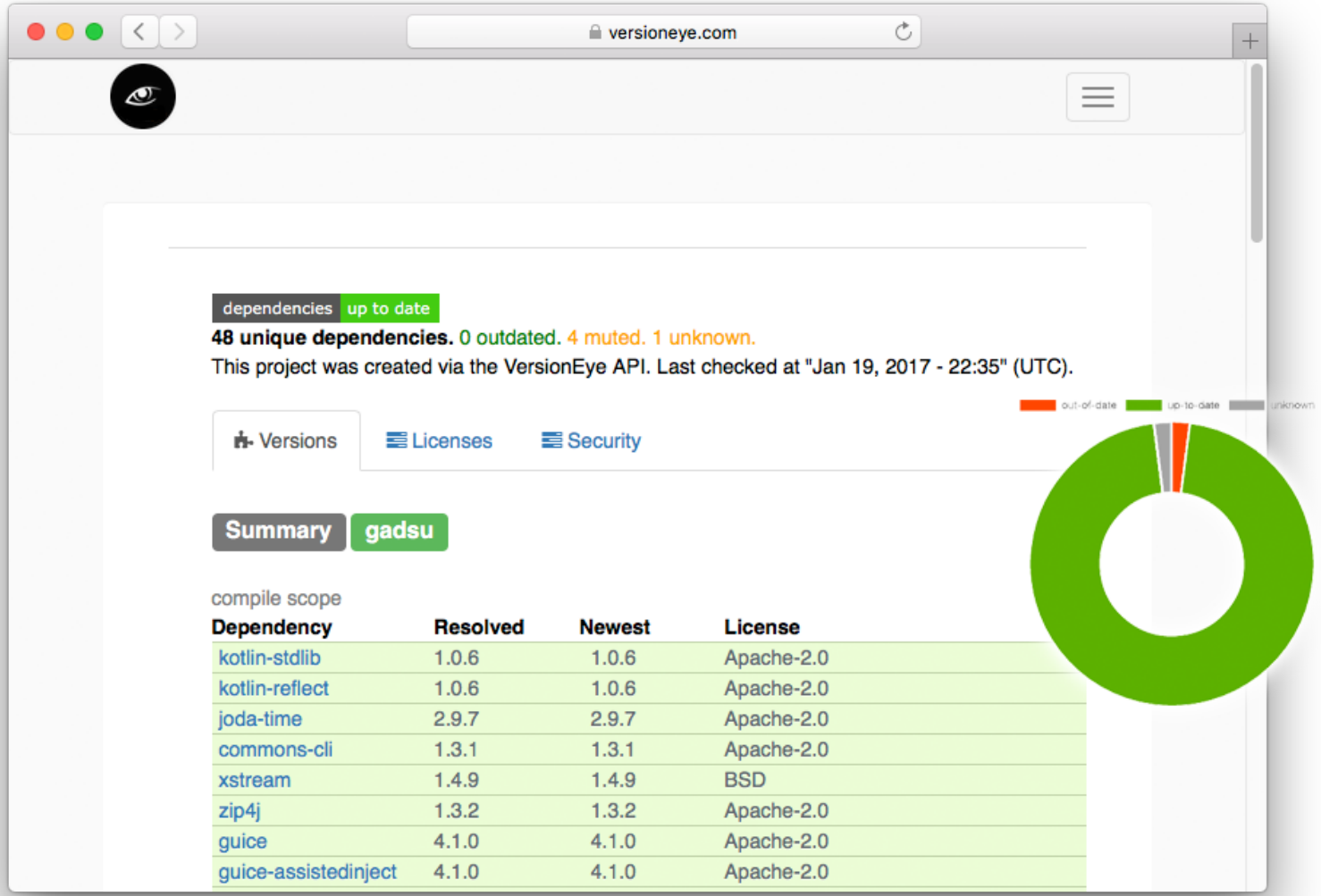
```
script:
  - "./gradlew ... jacocoTestReport ..."
after_success:
  - bash <(curl -s https://codecov.io/bash)
```

VersionEye GitHub Integration

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      }
```



versioneye.com

VersionEye

Gradle Configuration:

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

VersionEye

Gradle Configuration:

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

Gradle Properties:

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

VersionEye

Gradle Configuration:

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

Gradle Properties:

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

Travis Configuration:

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

Code Schmankerln

Null Handling

Trust no one!

```
@Data
class Contact {
    String mail;
}
```

Trust no one!

```
@Data
class Contact {
    String mail;
}
@Data
class Client {
  Contact contact;
  public boolean hasContact() {
    return contact != null;
  }
}
```

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

```kotlin
data class Contact(val mail: String)
```

Explicit handling in Kotlin

```kotlin
data class Contact(val mail: String)
data class Client(val contact: Contact?)
```

Explicit handling in Kotlin

```kotlin
data class Contact(val mail: String)
data class Client(val contact: Contact?)

client.contact.ifNotNull(::process)
```

Explicit handling in Kotlin

```kotlin
data class Contact(val mail: String)
data class Client(val contact: Contact?)

client.contact.ifNotNull(::process)
client.contact?.mail ?: "default@mail.at"
```

Explicit handling in Kotlin

```kotlin
data class Contact(val mail: String)
data class Client(val contact: Contact?)

client.contact.ifNotNull(::process)
client.contact?.mail ?: "default@mail.at"
```

Write your own, custom *if-loop*:

```kotlin
fun <T> T?.ifNotNull(action: (T) -> Unit):
  T? {
  if (this == null) {
    return null
  }
  action(this)
  return this
}
```

Extension Methods

Domain Object

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 Extensions

Persistence specific functionality:

```
1 package at.cpickl.gadsu.persistence
2
3 data class ClientDbo(
4   val TXT_ID: String,
5   val TXT_NAME: String
6 )
```

Persistence Extensions

Persistence specific functionality:

```
1 package at.cpickl.gadsu.persistence
2
3 data class ClientDbo(
4   val TXT_ID: String,
5   val TXT_NAME: String
6 )
7 fun Client.toDbo() =
8         ClientDbo(id, name)
9
```

Persistence Extensions

Persistence specific functionality:

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

Nullable Persistence Extensions

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)
```

Nullable Persistence Extensions

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()
```

Extend Swing Components

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
```

Extend Swing Components

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

Testee 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 )
```

Testee 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 }
```

Integration Testee

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:

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

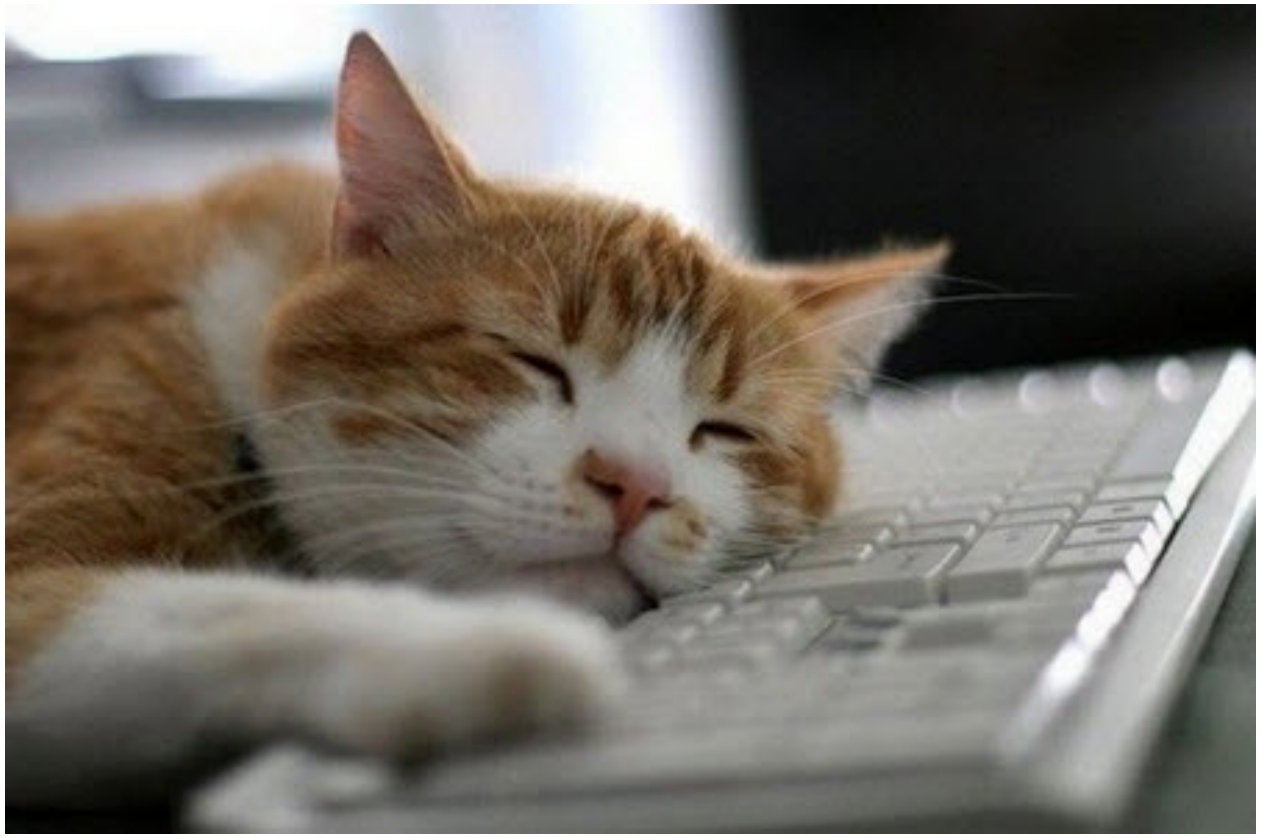
Kotlin (implicit `it` variable)

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

Java sometimes behaves like good old *Aunt Chatty*.

Kotlin Basic Functionals?!

- `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 . . .

Lazy in Java

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

Lazy in Java

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 }
```

Lazy in Java8

```
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 }
```

Lazy in Kotlin

```
class LazyKotlin {
  val lazyField by lazy {
    expensiveInit()
  }
}
```

Lazy in Kotlin

```kotlin
class LazyKotlin {
  val lazyField by lazy {
    expensiveInit()
  }
}

// part of stdlib:
fun <T> lazy(initializer: ()->T): Lazy<T> =
  SynchronizedLazyImpl(initializer)
```

Lazy in Kotlin

```kotlin
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 explicity.

Delegates

Class Delegation

Given the existing classes:

```
interface Step {
  fun take()
}
```

Class Delegation

Given the existing classes:

```
interface Step {
  fun take()
}

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

Class Delegation

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.

Reimplement in Java

```java
public class MyService implements Step {

  private final Step step;

  public MyService(Step step) {
    this.step = step;
  }

  @Override public void take() {
    step.take();
  }
}
```

Delegate by Kotlin

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

Delegate by Kotlin

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

Standard delegates in Kotlin:

- **lazy**
- observable
- map properties

Let's do the Swing!

Common Swing Component

```
1 class ClientTabMain(modifications, ...) {
```

Common Swing Component

```
1 class ClientTabMain(modifications, ...) {
2   val fields = Fields<Client>(modifications)
```

Common Swing Component

```
class ClientTabMain(modifications, ...) {
  val fields = Fields<Client>(modifications)
  val inpNote = fields.newTextArea(
    "Notiz", { it.note },
    ViewNames.Client.InputNote, bus)
```

Common Swing Component

```
class ClientTabMain(modifications, ...) {
  val fields = Fields<Client>(modifications)
  val inpNote = fields.newTextArea(
    "Notiz", { it.note },
    ViewNames.Client.InputNote, bus)
  init {
    add(VFillFormPanel().apply {
      addFormInput(inpNote)
    })}
```

Common Swing Component

```
class ClientTabMain(modifications, ...) {
  val fields = Fields<Client>(modifications)
  val inpNote = fields.newTextArea(
    "Notiz", { it.note },
    ViewNames.Client.InputNote, bus)
  init {
    add(VFillFormPanel().apply {
      addFormInput(inpNote)
    })}
  fun isModified(client: Client) =
    fields.isAnyModified(client)
```

Common Swing Component

```
class ClientTabMain(modifications, ...) {
  val fields = Fields<Client>(modifications)
  val inpNote = fields.newTextArea(
    "Notiz", { it.note },
    ViewNames.Client.InputNote, bus)
  init {
    add(VFillFormPanel().apply {
      addFormInput(inpNote)
    })}
  fun isModified(client: Client) =
    fields.isAnyModified(client)
  fun updateFields(client: Client) {
    fields.updateAll(client)
  }
}
```

Event Driven Swing

Dispatch events via `EventBus` and subscribe in Controller:

```
1 open class ClientViewController
```

Event Driven Swing

Dispatch events via `EventBus` and subscribe in Controller:

```
1 open class ClientViewController @Inject
2   constructor(val bus: EventBus, ...) {
```

Event Driven Swing

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) {
```

Event Driven Swing

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       ...
```

Spec4J UI Test

```
@Test(groups = arrayOf("uiTest"))
class ClientUiTest : UiTest() {
 fun `save client should change UI`() {
```

Spec4J UI Test

```
@Test(groups = arrayOf("uiTest"))
class ClientUiTest : UiTest() {
 fun `save client should change UI`() {
  val client = Client.unsavedValidInstance()
```

Spec4J UI Test

```
1 @Test(groups = arrayOf("uiTest"))
2 class ClientUiTest : UiTest() {
3  fun `save client should change UI`() {
4   val client = Client.unsavedValidInstance()
5   with(driver) {
```

Spec4J UI Test

```
@Test(groups = arrayOf("uiTest"))
class ClientUiTest : UiTest() {
 fun `save client should change UI`() {
  val client = Client.unsavedValidInstance()
  with(driver) {
   assertSaveButtonTextEquals("Neu anlegen")
```

Spec4J UI Test

```
@Test(groups = arrayOf("uiTest"))
class ClientUiTest : UiTest() {
 fun `save client should change UI`() {
  val client = Client.unsavedValidInstance()
  with(driver) {
   assertSaveButtonTextEquals("Neu anlegen")

   saveClient(client)
   assertSaveButtonTextEquals("Speichern")
```

Spec4J UI Test

```
@Test(groups = arrayOf("uiTest"))
class ClientUiTest : UiTest() {
 fun `save client should change UI`() {
  val client = Client.unsavedValidInstance()
  with(driver) {
   assertSaveButtonTextEquals("Neu anlegen")

   saveClient(client)
   assertSaveButtonTextEquals("Speichern")

   assertListContains(client)
   assertListSelected(client)
   ...
```

Spec4J UI Driver

```
1 class ClientDriver(
2   test: UiTest, window: Window) {
```

Spec4J UI Driver

```
class ClientDriver(
  test: UiTest, window: Window) {

  val list = window.getListBox(
    ViewNames.Client.List)!!
  val createButton = window.getButton(
    ViewNames.Client.CreateButton)!!
```

Spec4J UI Driver

```
class ClientDriver(
  test: UiTest, window: Window) {

  val list = window.getListBox(
    ViewNames.Client.List)!!
  val createButton = window.getButton(
    ViewNames.Client.CreateButton)!!

  fun assertNoChangesDetected() {
    test.assertThat(
      test.not(saveButton.isEnabled))
    test.assertThat(
      test.not(cancelButton.isEnabled))
  }
  ...
```

Lessons Learned

Tooling infrastructure grows

- *Mostly* as good as for Java

Tooling infrastructure grows

- *Mostly* as good as for Java
- **IntelliJ** support feels already superb

Tooling infrastructure grows

- *Mostly* as good as for Java
- **IntelliJ** support feels already superb
- **Build** system support (Gradle Script Kotlin!)

Tooling infrastructure grows

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

Tooling infrastructure grows

- *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**

8 reasons to use Kotlin

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


I DON'T ALWAYS CODE IN JAVA
BUT WHEN I DO, IT'S A NULLPOINTEREXCEPTION

Remember me?!

8 reasons to use Kotlin

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

8 reasons to use Kotlin

1. **Null handling** is a MUST!
2. **Extension methods** for improved auto completion

8 reasons to use Kotlin

1. **Null handling** is a MUST!
2. **Extension methods** for improved auto completion
3. (Constructor) Properties

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

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**

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**

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 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

Kotlin is a great language . . . BUT!

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?

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

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!

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

Apply – This is not this anymore!

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

```
IllegalArgumentException: adding container's parent to itself
```

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)`

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`?!

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

```
JPanel().also {
  this.add(it)
}
```

11 Kotlin 1.1 News

1. Compiler plugins (open-by-default, no-arg ctor)

11 Kotlin 1.1 News

1. Compiler plugins (open-by-default, no-arg ctor)
2. Coroutines

11 Kotlin 1.1 News

1. Compiler plugins (open-by-default, no-arg ctor)
2. Coroutines
3. Type aliases

11 Kotlin 1.1 News

1. Compiler plugins (open-by-default, no-arg ctor)
2. Coroutines
3. Type aliases
4. Type inference for getters

11 Kotlin 1.1 News

1. Compiler plugins (open-by-default, no-arg ctor)
2. Coroutines
3. Type aliases
4. Type inference for getters
5. Bound callable references

11 Kotlin 1.1 News

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

11 Kotlin 1.1 News

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

11 Kotlin 1.1 News

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

11 Kotlin 1.1 News

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

11 Kotlin 1.1 News

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 Kotlin 1.1 News

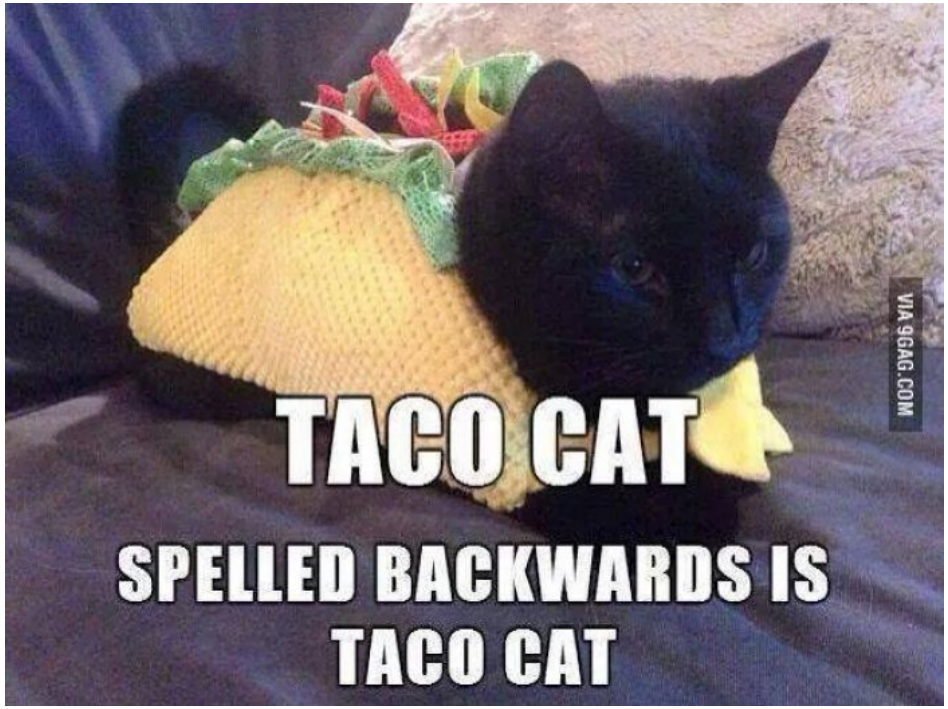
1. Compiler plugins (open-by-default, no-arg ctor)
2. Coroutines
3. Type aliases
4. Type inference for getters
5. Bound callable references
6. Local delegated properties & Inline properties
7. Inheritance for data classes
8. Subclasses of sealed classes in the same file
9. Destructuring in lambdas
10.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

Links

- 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


TACO CAT
SPELLED BACKWARDS IS
TACO CAT
VIA 9GAG.COM


INVISIBLE BIKE


I HATE
CAT MEMES