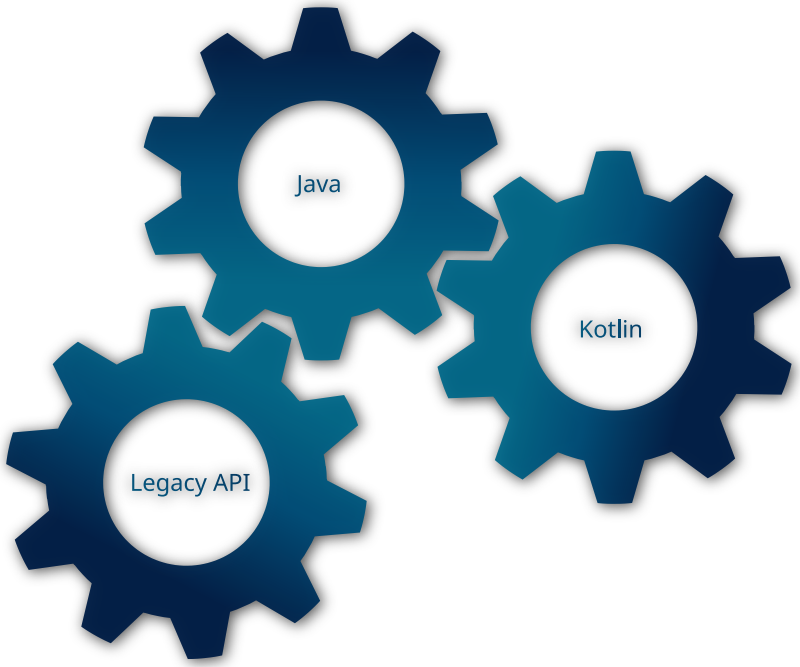


Java Legacy-APIs mit Kotlin bändigen

Roland Innerhofer, Areltis GmbH

Safe harbor / Disclaimer

Die Handlung dieses Talks ist frei erfunden. Etwaige Ähnlichkeiten mit tatsächlichen Begebenheiten oder mit existierenden oder bereits abgelösten APIs wären rein zufällig. Etwaige Urheberrechtsverletzungen sind unbeabsichtigt und die entsprechenden Inhalte werden bei Meldung sofort entfernt.

Safe harbor / Disclaimer

Die Handlung dieses Talks ist frei erfunden. Etwaige Ähnlichkeiten mit tatsächlichen Begebenheiten oder mit existierenden oder bereits abgelösten APIs wären rein zufällig. Etwaige Urheberrechtsverletzungen sind unbeabsichtigt und die entsprechenden Inhalte werden bei Meldung sofort entfernt.

Übersicht

areltis

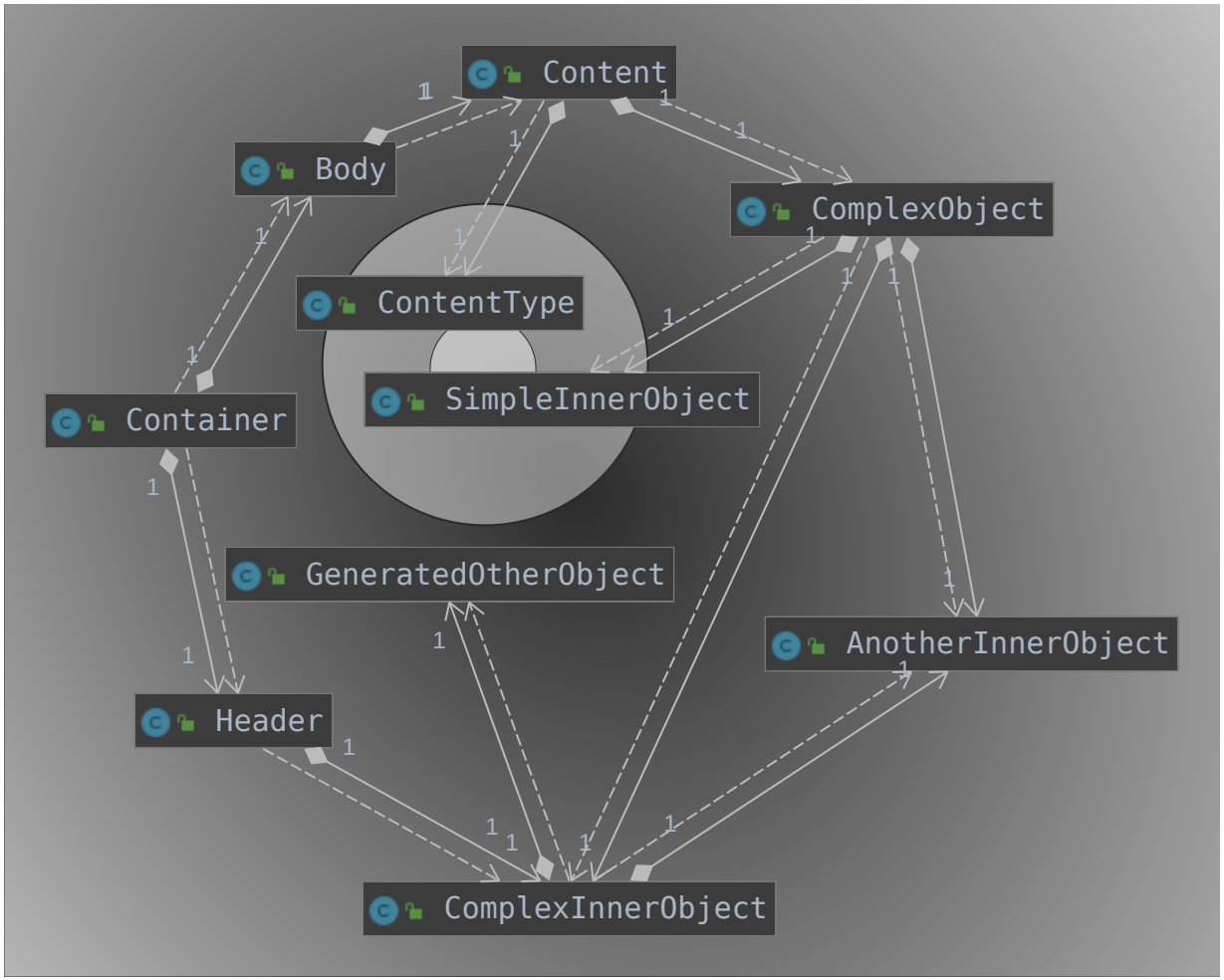
1. Generierte Klassen
2. Server API
3. Prozedural

Übersicht

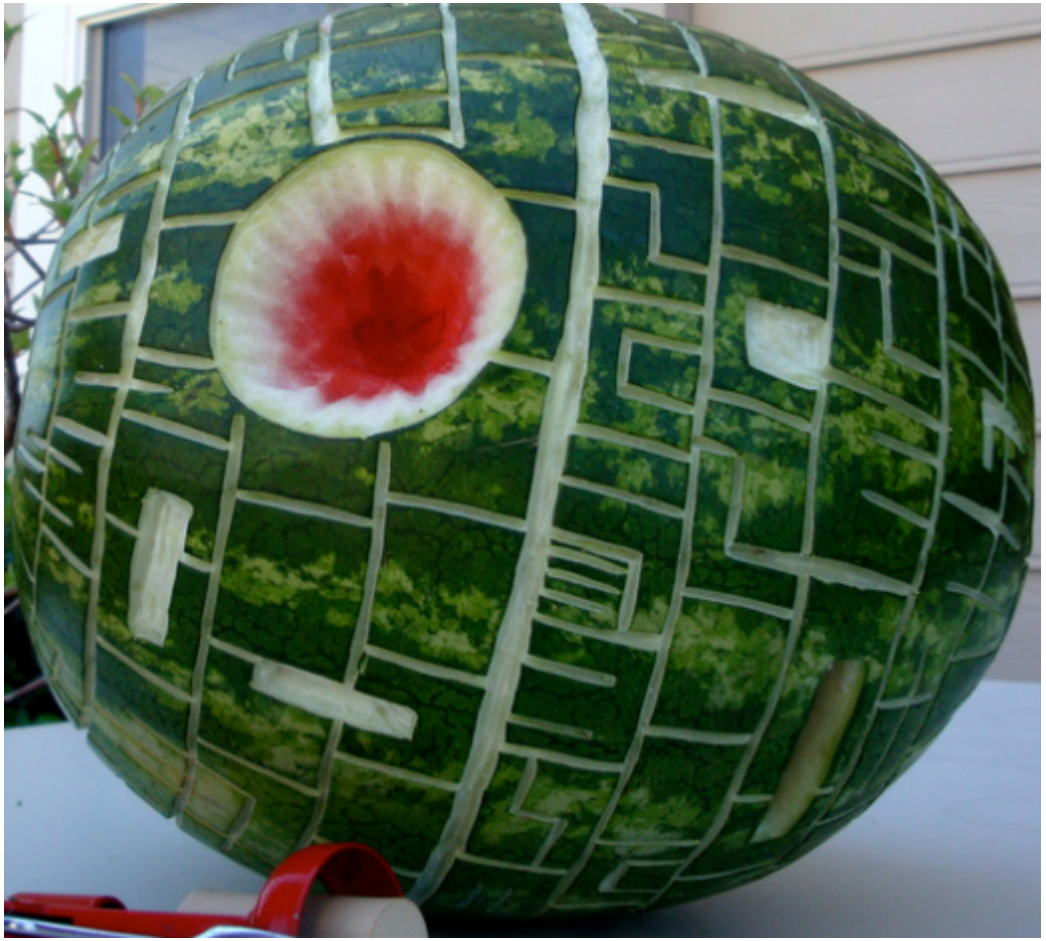
areltis

1. Generierte Klassen
2. Server API
3. Prozedural
4. ~~Live-coding~~
5. ~~viele weitere~~

Generierte Klassen



Generierte Klassen



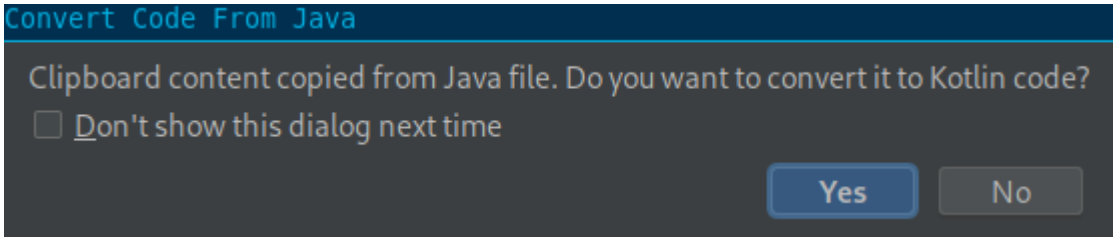
Flickr

Generierte Klassen: Java

```
Container container = new Container();

Header header = new Header();
ComplexInnerObject complexContent = new ComplexInnerObject();
complexContent.setContent(new GeneratedOtherObject());
header.setComplexContent(complexContent);
container.setHeader(header);

Content content = new Content();
ContentType contentType = new ContentType();
content.setContentType(contentType);
content.setComplexObject1(new ComplexObject());
ComplexObject complexObject2 = new ComplexObject();
ComplexInnerObject complexInner2 = new ComplexInnerObject();
complexInner2.setContent(new GeneratedOtherObject());
complexObject2.setComplexInner(complexInner2);
content.setComplexObject2(complexObject2);
ComplexObject complexObject3 = new ComplexObject();
complexObject3.setSimpleInner(new SimpleInnerObject());
content.setComplexObject3(complexObject3);
ComplexObject complexObject4 = new ComplexObject();
complexObject4.setComplexInner(new ComplexInnerObject());

Body body = new Body();
body.setContent(content);

container.setBody(body);
```

Generierte Klassen: Kotlin

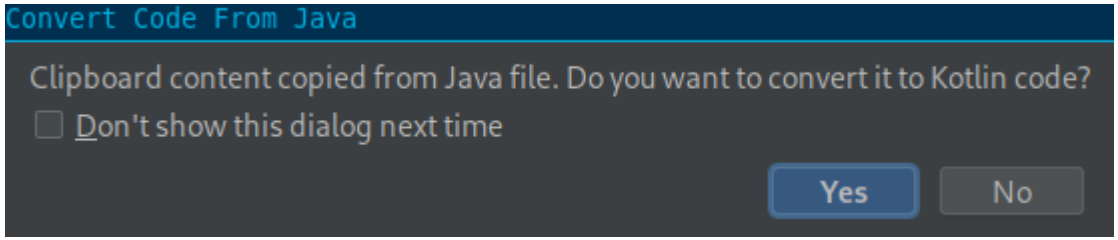
Wie wär's mit automatischer Umwandlung vom Java-Code zu Kotlin? z.B. via Intellij?



Generierte Klassen: Kotlin

areltis

Wie wär's mit automatischer Umwandlung vom Java-Code zu Kotlin? z.B. via Intellij?



Generierte Klassen: Kotlin

```kotlin
val container = Container()

val header = Header()
val complexContent = ComplexInnerObject()
complexContent.content = GeneratedOtherObject()
header.complexContent = complexContent
container.header = header

val content = Content()
val contentType = ContentType()
content.contentType = contentType
content.complexObject1 = ComplexObject()
val complexObject2 = ComplexObject()
val complexInner2 = ComplexInnerObject()
complexInner2.content = GeneratedOtherObject()
complexObject2.complexInner = complexInner2
content.complexObject2 = complexObject2
val complexObject3 = ComplexObject()
complexObject3.simpleInner = SimpleInnerObject()
content.complexObject3 = complexObject3
val complexObject4 = ComplexObject()
complexObject4.complexInner = ComplexInnerObject()

val body = Body()
body.content = content

container.body = body
```

Generierte Klassen: Java vs Kotlin

```java
Container container = new Container();

Header header = new Header();
ComplexInnerObject complexContent = new ComplexI
complexContent.setContent(new GeneratedOtherObje
header.setComplexContent(complexContent);
container.setHeader(header);

Content content = new Content();
ContentType contentType = new ContentType();
content.setContentType(contentType);
content.setComplexObject1(new ComplexObject());
ComplexObject complexObject2 = new ComplexObject
ComplexInnerObject complexInner2 = new ComplexIn
complexInner2.setContent(new GeneratedOtherObjec
complexObject2.setComplexInner(complexInner2);
content.setComplexObject2(complexObject2);
ComplexObject complexObject3 = new ComplexObject
complexObject3.setSimpleInner(new SimpleInnerObj
content.setComplexObject3(complexObject3);
ComplexObject complexObject4 = new ComplexObject
complexObject4.setComplexInner(new ComplexInner

Body body = new Body();
body.setContent(content);

container.setBody(body);
```

```kotlin
val container = Container()

val header = Header()
val complexContent = ComplexInnerObject()
complexContent.content = GeneratedOtherObject()
header.complexContent = complexContent
container.header = header

val content = Content()
val contentType = ContentType()
content.contentType = contentType
content.complexObject1 = ComplexObject()
val complexObject2 = ComplexObject()
val complexInner2 = ComplexInnerObject()
complexInner2.content = GeneratedOtherObject()
complexObject2.complexInner = complexInner2
content.complexObject2 = complexObject2
val complexObject3 = ComplexObject()
complexObject3.simpleInner = SimpleInnerObject()
content.complexObject3 = complexObject3
val complexObject4 = ComplexObject()
complexObject4.complexInner = ComplexInnerObject()

val body = Body()
body.content = content

container.body = body
```

Generierte Klassen: Java vs Kotlin

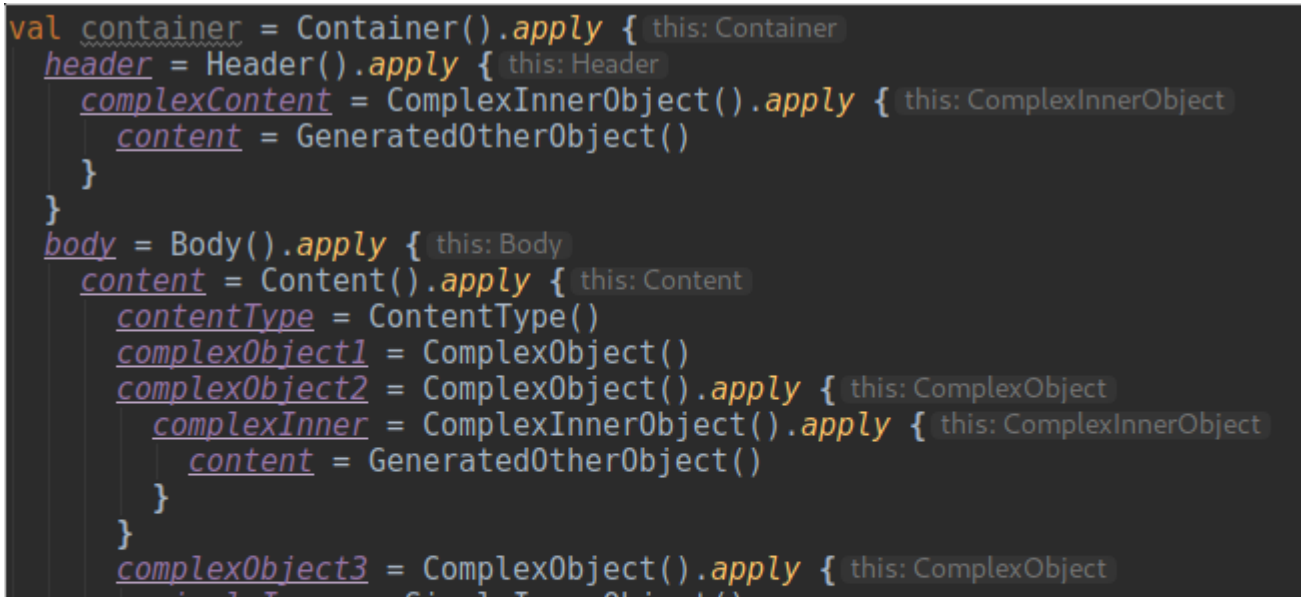
```java
Container container = new Container();

Header header = new Header();
ComplexInnerObject complexContent = new Complex
complexContent.setContent(new GeneratedOtherObje
header.setComplexContent(complexContent);
container.setHeader(header);

Content content = new Conte
ContentType contentType = 
content.setContentType(con
content.setComplexObject1(
ComplexObject complexObjec
ComplexInnerObject complex
complexInner2.setContent(ne
complexObject2.setComplexIn
content.setComplexObject2(
ComplexObject complexObjec
complexObject3.setSimpleInn
content.setComplexObject3(
ComplexObject complexObjec
complexObject4.setComplexIn

Body body = new Body();
body.setContent(content);

container.setBody(body);
```

```kotlin
val container = Container()

val header = Header()
val complexContent = ComplexInnerObject()
complexContent.content = GeneratedOtherObject()
header.complexContent = complexContent
container.header = header

ent()
ContentType()
e = contentType
ect1 = ComplexObject()
 = ComplexObject()
 = ComplexInnerObject()
ent = GeneratedOtherObject()
plexInner = complexInner2
ect2 = complexObject2
 = ComplexObject()
pleInner = SimpleInnerObject()
ect3 = complexObject3
 = ComplexObject()
plexInner = ComplexInnerObject()

val body = Body()
body.content = content

container.body = body
```

Generierte Klassen: Kotlins apply

```
val container = Container().apply { this: Container
  header = Header().apply { this: Header
    complexContent = ComplexInnerObject().apply { this: ComplexInnerObject
      content = GeneratedOtherObject()
    }
  }
  body = Body().apply { this: Body
    content = Content().apply { this: Content
      contentType = ContentType()
      complexObject1 = ComplexObject()
      complexObject2 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject().apply { this: ComplexInnerObject
          content = GeneratedOtherObject()
        }
      }
      complexObject3 = ComplexObject().apply { this: ComplexObject
        simpleInner = SimpleInnerObject()
      }
      val complexObject4 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject()
      }
    }
  }
}
```

Generierte Klassen: Kotlins apply



```
val container = Container().apply { this: Container
  header = Header().apply { this: Header
    complexContent = ComplexInnerObject().apply { this: ComplexInnerObject
      content = GeneratedOtherObject()
    }
  }
  body = Body().apply { this: Body
    content = Content().apply { this: Content
      contentType = ContentType()
      complexObject1 = ComplexObject()
      complexObject2 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject().apply { this: ComplexInnerObject
          content = GeneratedOtherObject()
        }
      }
      complexObject3 = ComplexObject().apply { this: ComplexObject
```

kotlin-stdlib / kotlin / apply

apply

Common JVM JS Native

```
inline fun <T> T.apply(block: T.() -> Unit): T
```

Calls the specified function block with `this` value as its receiver and returns `this` value.

Generierte Klassen: Java vs Kotlin

```java
Container container = new Container();

Header header = new Header();
ComplexInnerObject complexContent = new ComplexInnerObject();
complexContent.setContent(new GeneratedOtherObject());
header.setComplexContent(complexContent);
container.setHeader(header);

Content content = new Content();
ContentType contentType = new ContentType();
content.setContentType(contentType);
content.setComplexObject1(new ComplexObject());
ComplexObject complexObject2 = new ComplexObject();
ComplexInnerObject complexInner2 = new ComplexInnerObject();
complexInner2.setContent(new GeneratedOtherObject());
complexObject2.setComplexInner(complexInner2);
content.setComplexObject2(complexObject2);
ComplexObject complexObject3 = new ComplexObject();
complexObject3.setSimpleInner(new SimpleInnerObject());
content.setComplexObject3(complexObject3);
ComplexObject complexObject4 = new ComplexObject();
complexObject4.setComplexInner(new ComplexInnerObject());

Body body = new Body();
body.setContent(content);

container.setBody(body);
```

```kotlin
val container = Container().apply { this: Container
  header = Header().apply { this: Header
    complexContent = ComplexInnerObject().apply { this: ComplexInnerObject
      content = GeneratedOtherObject()
    }
  }
  body = Body().apply { this: Body
    content = Content().apply { this: Content
      contentType = ContentType()
      complexObject1 = ComplexObject()
      complexObject2 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject().apply { this: ComplexInnerObject
          content = GeneratedOtherObject()
        }
      }
      complexObject3 = ComplexObject().apply { this: ComplexObject
        simpleInner = SimpleInnerObject()
      }
      val complexObject4 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject()
      }
    }
  }
}
```

Generierte Klassen: Kotlins apply

aber...

```
val container = Container().apply { this: Container
  header = Header().apply { this: Header
    complexContent = ComplexInnerObject().apply { this: ComplexInnerObject
      content = GeneratedOtherObject()
    }
  }
  body = Body().apply { this: Body
    content = Content().apply { this: Content
      contentType = ContentType()
      complexObject1 = ComplexObject()
      complexObject2 = ComplexObject().apply { this: ComplexObject
        complexInner = ComplexInnerObject().apply { this: ComplexInnerObject
          content = GeneratedOtherObject()
          body = Body()
        }
      }
      complex                                                    this: ComplexObject
        simpl
      }
```

```
fake.generated.Container
public void setBody(Body body)
java-legacy-kotlin.main
```

Generierte Klassen: Kotlin

@DslMarker & type-safe-builders

```kotlin
@DslMarker
@Target(AnnotationTarget.TYPE)
annotation class BuilderMarker
```

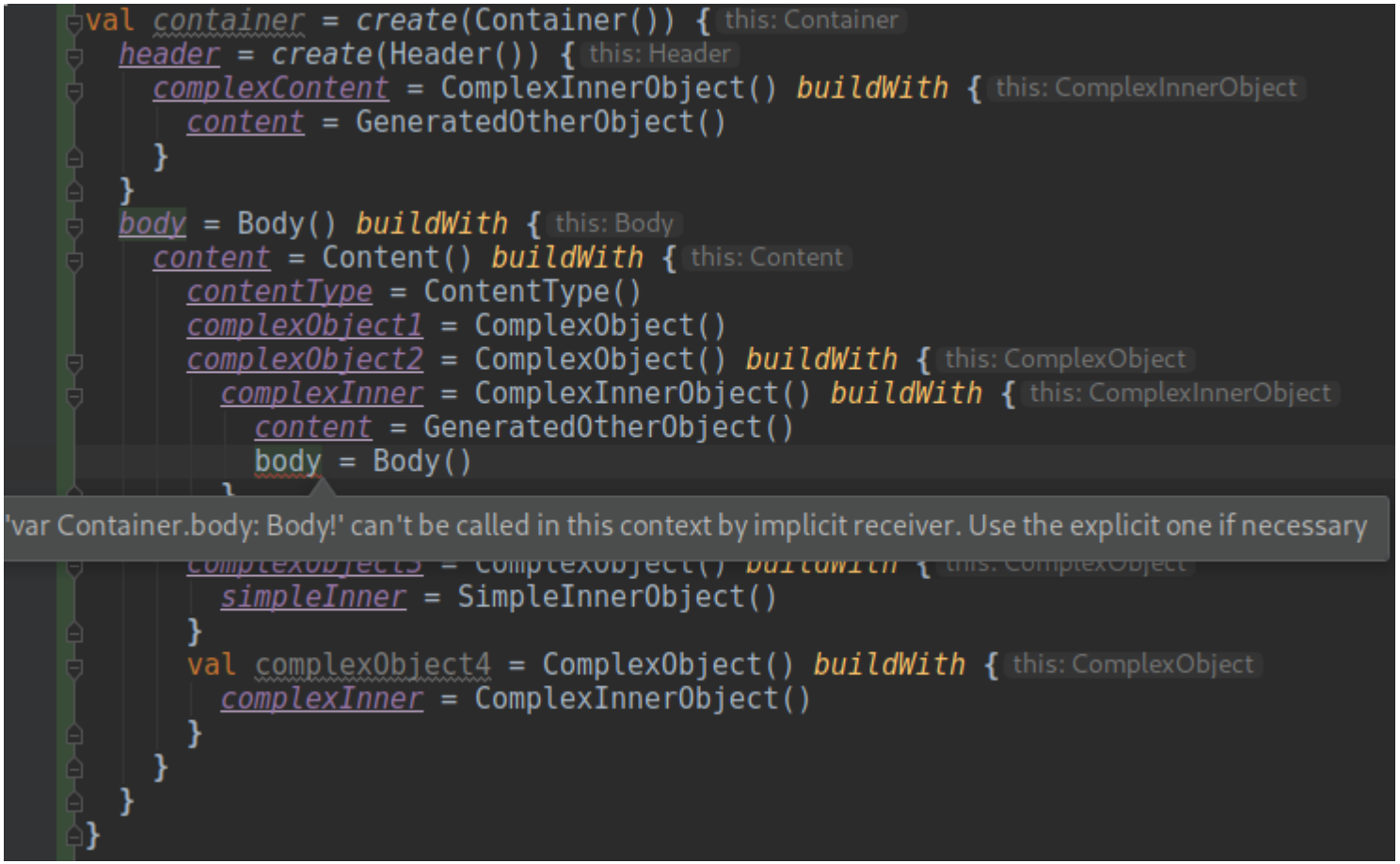
Generierte Klassen: Kotlin

@DslMarker & type-safe-builders

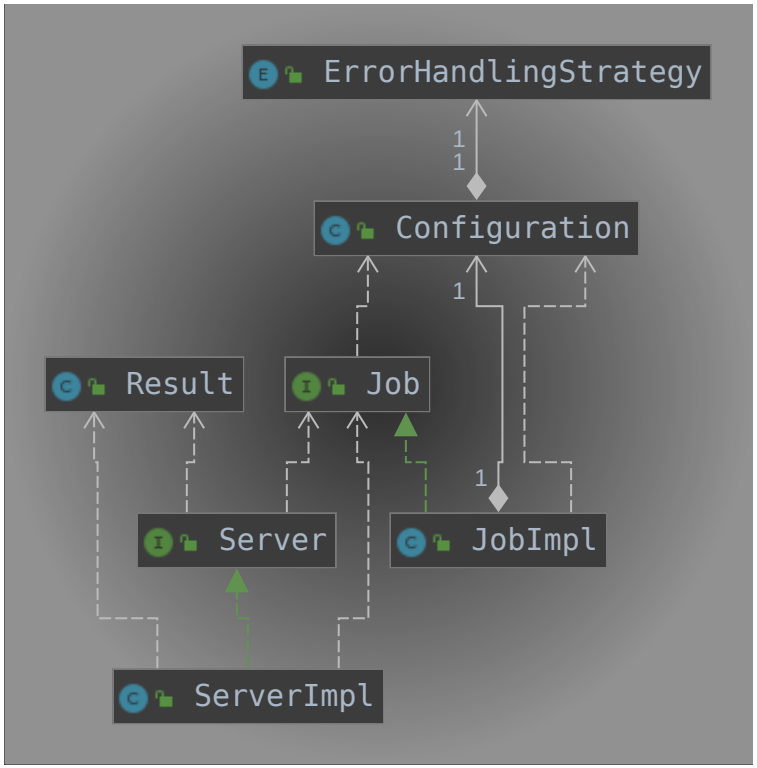
```kotlin
@DslMarker
@Target(AnnotationTarget.TYPE)
annotation class BuilderMarker
```

```kotlin
inline infix fun <T> T.buildWith(block: (@BuilderMarker T).() -> Unit) = apply(block)

inline fun <T> create(t: T, block: (@BuilderMarker T).() -> Unit) = t.buildWith(block)
```

Generierte Klassen: Kotlin



Server API



Server API: Java

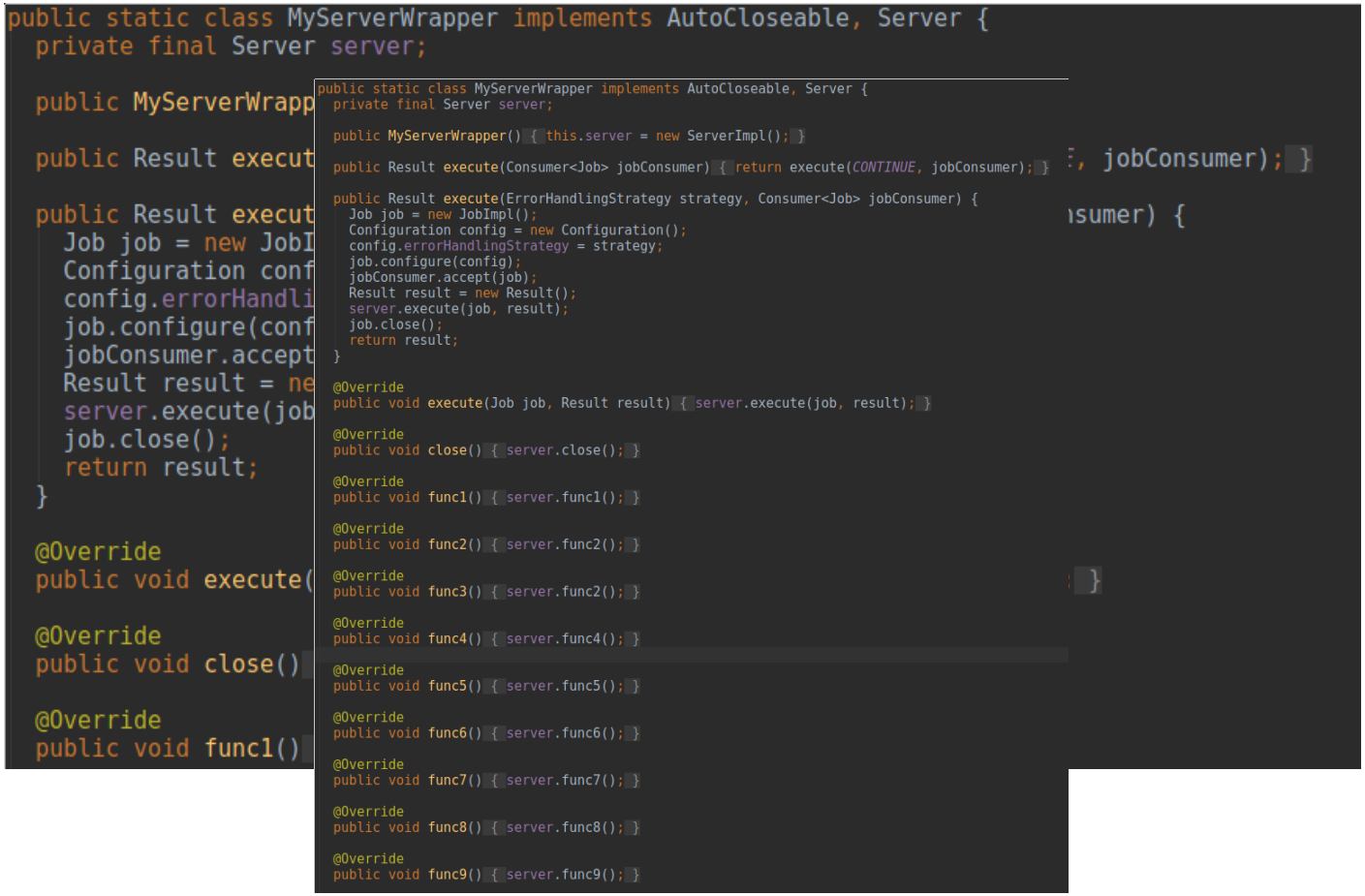
```java
Server server = new ServerImpl();

{
  Job job = new JobImpl();
  job.doSome( work: "work");
  Configuration config = new Configuration();
  config.errorHandlingStrategy = FAIL;
  job.configure(config);
  Result result = new Result();
  server.execute(job, result);
  job.close();
}
{
  Job job = new JobImpl();
  job.doSome( work: "otherWork");
  Configuration config = new Configuration();
  config.errorHandlingStrategy = CONTINUE;
  job.configure(config);
  Result result = new Result();
  server.execute(job, result);
  job.close();
}

server.close();
```

Server API: Java - Wrapper/Delegate?

areltis

```java
public static class MyServerWrapper implements AutoCloseable, Server {
  private final Server server;

  public MyServerWrapper() { this.server = new ServerImpl(); }

  public Result execute(Consumer<Job> jobConsumer) { return execute(CONTINUE, jobConsumer); }

  public Result execute(ErrorHandlingStrategy strategy, Consumer<Job> jobConsumer) {
    Job job = new JobImpl();
    Configuration config = new Configuration();
    config.errorHandlingStrategy = strategy;
    job.configure(config);
    jobConsumer.accept(job);
    Result result = new Result();
    server.execute(job, result);
    job.close();
    return result;
  }

  @Override
  public void execute(Job job, Result result) { server.execute(job, result); }

  @Override
  public void close() { server.close(); }

  @Override
  public void func1() { server.func1(); }
```

Server API: Java - Wrapper/Delegate?



```java
public static class MyServerWrapper implements AutoCloseable, Server {
  private final Server server;

  public MyServerWrapper() { this.server = new ServerImpl(); }

  public Result execute(Consumer<Job> jobConsumer) { return execute(CONTINUE, jobConsumer); }

  public Result execute(ErrorHandlingStrategy strategy, Consumer<Job> jobConsumer) {
    Job job = new JobImpl();
    Configuration config = new Configuration();
    config.errorHandlingStrategy = strategy;
    job.configure(config);
    jobConsumer.accept(job);
    Result result = new Result();
    server.execute(job, result);
    job.close();
    return result;
  }

  @Override
  public void execute(Job job, Result result) { server.execute(job, result); }

  @Override
  public void close() { server.close(); }

  @Override
  public void func1() { server.func1(); }

  @Override
  public void func2() { server.func2(); }

  @Override
  public void func3() { server.func2(); }

  @Override
  public void func4() { server.func4(); }

  @Override
  public void func5() { server.func5(); }

  @Override
  public void func6() { server.func6(); }

  @Override
  public void func7() { server.func7(); }

  @Override
  public void func8() { server.func8(); }

  @Override
  public void func9() { server.func9(); }
```

Server API: Java - Wrapper/Delegate!

```
try (MyServerWrapper server = new MyServerWrapper()) {
  server.execute(FAIL, job -> job.doSome( work: "work"));
  server.execute(CONTINUE, job -> job.doSome( work: "otherWork"));
  server.execute(job -> job.doSome( work: "continueWork"));
}
```

Server API: Java - vorher/nachher

```java
Server server = new ServerImpl();

{
  Job job = new JobImpl();
  job.doSome( work: "work");
  Configuration config = new Configuration();
  config.errorHandlingStrategy = FAIL;
  job.configure(config);
  Result result = new Result();
  server.execute(job, result);
  job.close();
}
{
  Job job = new JobImpl();
  job.doSome( work: "otherWork");
  Configuration config = new Configuration();
  config.errorHandlingStrategy = CONTINUE;
  job.configure(config);
  Result result = new Result();
  server.execute(job, result);
  job.close();
}

server.close();
```

```java
try (MyServerWrapper server = new MyServerWrapper()) {
  server.execute(FAIL, job -> job.doSome( work: "work"));
  server.execute(CONTINUE, job -> job.doSome( work: "otherWork"));
  server.execute(job -> job.doSome( work: "continueWork"));
}
```

Server API: Kotlin?

areltis

Server API: Kotlin

delegation + extension function

```
class MyKotlinServerWrapper(private val server: Server = ServerImpl())
  : AutoCloseable, Server by server

inline fun Server.execute(strategy: ErrorHandlingStrategy = CONTINUE,
                          executeJob: Job.() -> Unit) =
    JobImpl().apply { this: JobImpl
      configure(Configuration().apply { this: Configuration
        errorHandlingStrategy = strategy
      })
    }.also(executeJob)
        .let { job ->
          job.begin()
          ^let Result().also { it: Result
            execute(job, it)
            job.close()
          }
        }
```

- delegation via `by`-keyword ➞ keine umständlichen Overrides nötig
- extension function für einen Typ via `fun <Typ>.<Funktionsname>`

Server API: Kotlin
Wrapper und `AutoClosable`

```
MyKotlinServerWrapper().use { server ->
  server.execute(FAIL) { this: Job
    doSome( work: "work")
  }
  server.execute(CONTINUE) { this: Job
    doSome( work: "otherWork")
  }
   ^use server.execute { this: Job
    doSome( work: "continueWork")
  }
}
```

```
MyKotlinServerWrapper().use { it: MyKotlinServerWrapper
  it.execute(FAIL) { this: Job
    doSome( work: "work")
  }
  it.execute(CONTINUE) { this: Job
    doSome( work: "otherWork")
  }
   ^use it.execute { this: Job
    doSome( work: "continueWork")
  }
}
```

`use` entspricht try-with-resources

Server API: Kotlin
ohne Wrapper ebenso verwendbar

```kotlin
val server = ServerImpl()
server.execute(FAIL) { this: Job
  doSome( work: "work")
}
server.execute(CONTINUE) { this: Job
  doSome( work: "otherWork")
}
server.execute { this: Job
  doSome( work: "continueWork")
}
```

- dann aber ohne try-with-resources-Funktionalität
- folglich `server.close()` wieder nötig

Server API: Kotlin
ohne Wrapper ebenso verwendbar

```kotlin
val server = ServerImpl()
server.execute(FAIL) { this: Job
  doSome( work: "work")
}
server.execute(CONTINUE) { this: Job
  doSome( work: "otherWork")
}
server.execute { this: Job
  doSome( work: "continueWork")
}
```

```kotlin
MyKotlinServerWrapper().use { server ->
  server.execute(FAIL) { this: Job
    doSome( work: "work")
  }
  server.execute(CONTINUE) { this: Job
    doSome( work: "otherWork")
  }
  ^use server.execute { this: Job
    doSome( work: "continueWork")
  }
}
```

- dann aber ohne try-with-resources-Funktionalität
- folglich `server.close()` wieder nötig

Server API: Java vs Kotlin

```java
try (MyServerWrapper server = new MyServerWrapper()) {
  server.execute(FAIL, job -> job.doSome( work: "work"));
  server.execute(CONTINUE, job -> job.doSome( work: "otherWork"));
  server.execute(job -> job.doSome( work: "continueWork"));
}
```

```kotlin
MyKotlinServerWrapper().use { server ->
  server.execute(FAIL) { this: Job
    doSome( work: "work")
  }
  server.execute(CONTINUE) { this: Job
    doSome( work: "otherWork")
  }
   ^use server.execute { this: Job
    doSome( work: "continueWork")
  }
}
```

Server API: Java vs Kotlin

```java
try (MyServerWrapper server = new MyServerWrapper()) {
  server.execute(FAIL, job -> job.doSome( work: "work"));
  server.execute(CONTINUE, job -> job.doSome( work: "otherWork"));
  server.execute(job -> job.doSome( work: "continueWork"));
}
```

```kotlin
val server = ServerImpl()
server.execute(FAIL) { this: Job
  doSome( work: "work")
}
server.execute(CONTINUE) { this: Job
  doSome( work: "otherWork")
}
server.execute { this: Job
  doSome( work: "continueWork")
}
```

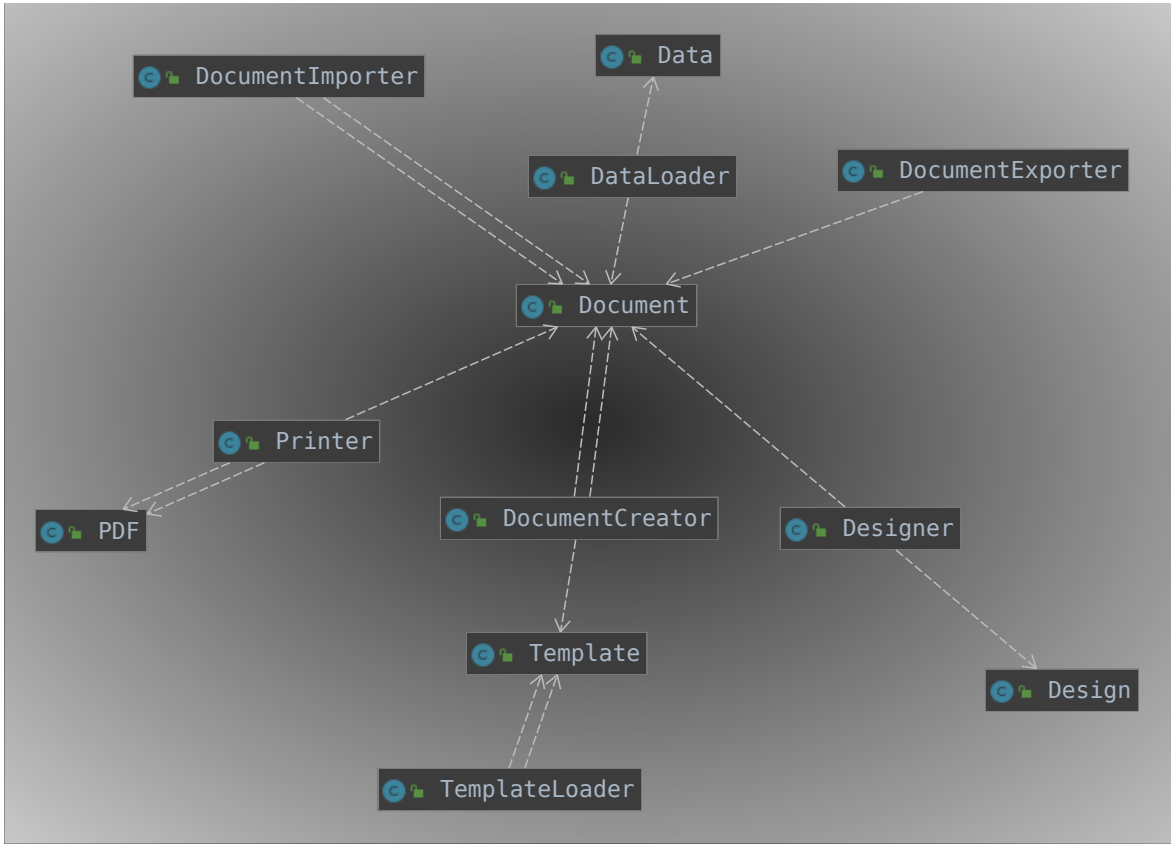
```kotlin
MyKotlinServerWrapper().use { server ->
  server.execute(FAIL) { this: Job
    doSome( work: "work")
  }
  server.execute(CONTINUE) { this: Job
    doSome( work: "otherWork")
  }
  ^use server.execute { this: Job
    doSome( work: "continueWork")
  }
}
```

Server API: Java vs Kotlin

```java
public static class MyServerWrapper implements AutoCloseable, Server {
  private final Server server;

  public MyServerWrapper() { this.server = new ServerImpl(); }

  public Result execute(Consumer<Job> jobConsumer) { return execute(CONTINUE, jobConsumer); }

  public Result execute(ErrorHandlingStrategy strategy, Consumer<Job> jobConsumer) {
    Job job = new JobImpl();
    Configuration config = new Configuration();
    config.errorHandlingStrategy = strategy;
    job.configure(config);
    jobConsumer.accept(job);
    Result result = new Result();
    server.execute(job, result);
    job.close();
    return result;
  }

  @Override
  public void execute(Job job, Result result) { server.execute(job, result); }

  @Override
  public void close() { server.close(); }

  @Override
  public void func1() { server.func1(); }

  @Override
  public void func2() { server.func2(); }

  @Override
  public void func3() { server.func2(); }

  @Override
  public void func4() { server.func4(); }

  @Override
  public void func5() { server.func5(); }

  @Override
  public void func6() { server.func6(); }

  @Override
  public void func7() { server.func7(); }

  @Override
  public void func8() { server.func8(); }

  @Override
  public void func9() { server.func9(); }
```

```kotlin
class MyKotlinServerWrapper(private val server: Server = ServerImpl())
  : AutoCloseable, Server by server

inline fun Server.execute(strategy: ErrorHandlingStrategy = CONTINUE,
                          executeJob: Job.() -> Unit) =
    JobImpl().apply { this: JobImpl
      configure(Configuration().apply { this: Configuration
        errorHandlingStrategy = strategy
      })
    }.also(executeJob)
      .let { job ->
        job.begin()
        ^let Result().also { it: Result
          execute(job, it)
          job.close()
        }
      }
```

Prozedural

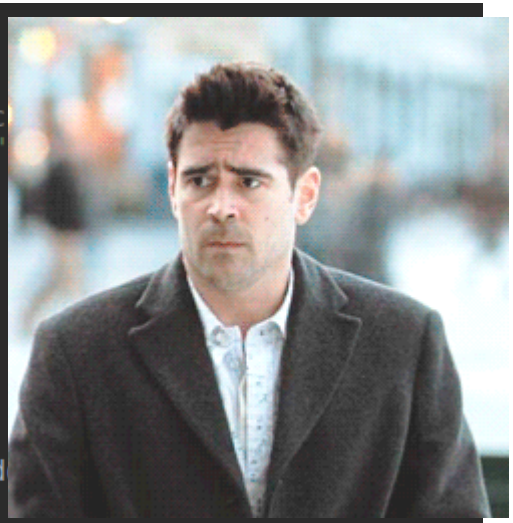


Prozedural: Java

```
Document doc;
Document doc2;

{ // erstes Dokument kreieren: load template > create doc > prepare design > load data
  Template template = TemplateLoader.loadTemplate( string: "demo");

  doc = DocumentCreator.createDocument(template);
  Designer.prepareDesign(doc, new Design());
  DataLoader.loadData(doc, new Data());
}
{ // zweites Dokument importieren
  doc2 = importer.importDocument(reader);
}

Document combinedDoc = DocumentCreator.combineDocuments(doc, doc2);

exporter.exportDocument(combinedDoc, writer);

PDF pdf = Printer.printToPdf(combinedDoc);
```

Prozedural: Java

```
Document doc;
Document doc2;

{ // erstes Dokument kreieren: load template > create doc
  Template template = TemplateLoader.loadTemplate( string: "

  doc = DocumentCreator.createDocument(template);
  Designer.prepareDesign(doc, new Design());
  DataLoader.loadData(doc, new Data());
}
{ // zweites Dokument importieren
  doc2 = importer.importDocument(reader);
}

Document combinedDoc = DocumentCreator.combineDocuments(d

exporter.exportDocument(combinedDoc, writer);

PDF pdf = Printer.printToPdf(combinedDoc);
```



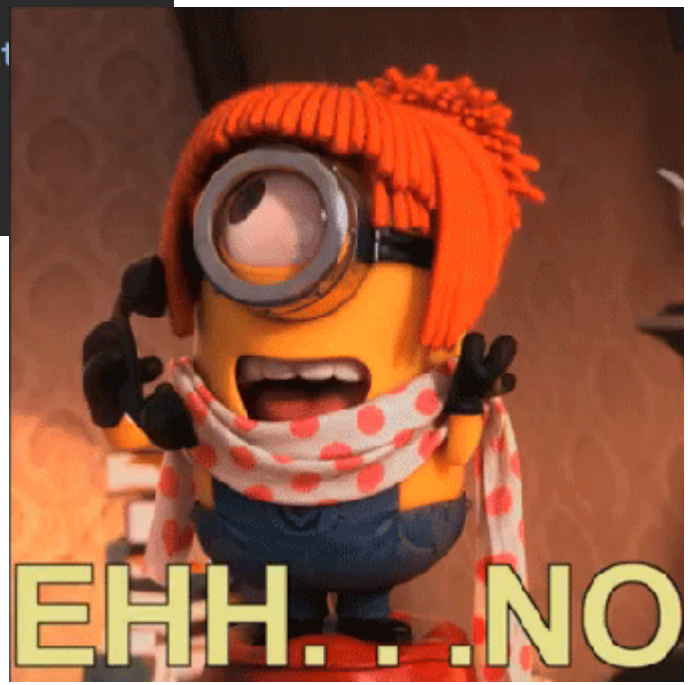
- fast alles statische Aufrufe
- ohne Dokumentation unklar welche Hilfsklasse wann zum Zug kommt

Prozedural: Kotlin

```kotlin
val pdf = loadTemplate( string: "demo")
    .let(::createDocument)
    .also { it: Document!
      Designer.prepareDesign(it, Design())
      DataLoader.loadData(it, Data())
    }.let { it: Document!
      arrayOf(it, importer.importDocument(reader))
    }
    .let { it: Array<Document!>
      DocumentCreator.combineDocuments(*it)
    }.also { it: Document!
      exporter.exportDocument(it, writer)
    }
    .let(::printToPdf)
```

Prozedural: Kotlin

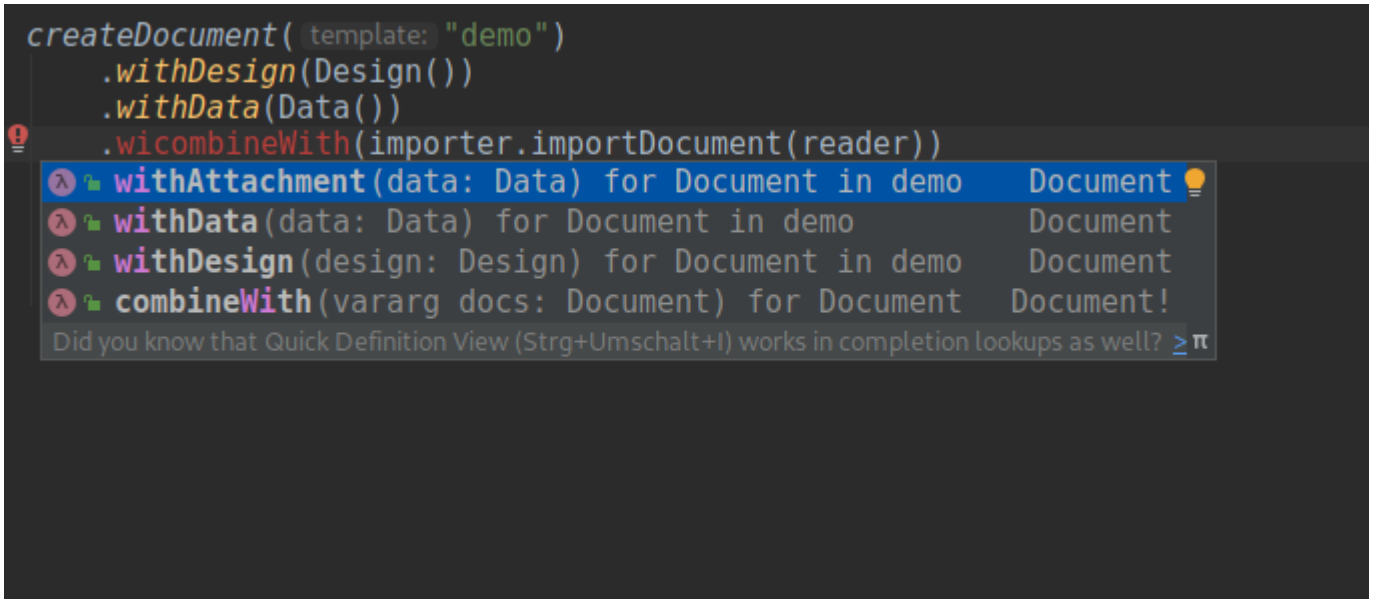
```kotlin
val pdf = loadTemplate( string: "demo")
    .let(::createDocument)
    .also { it: Document!
      Designer.prepareDesign(it, Design())
      DataLoader.loadData(it, Data())
    }.let { it: Document!
      arrayOf(it, importer.importDocument(reader))
    }
    .let { it: Array<Document!>
      DocumentCreator.combineDocuments(*it
    }.also { it: Document!
      exporter.exportDocument(it, writer)
    }
    .let(::printToPdf)
```



Prozedural: Kotlin - fluent api

```
createDocument( template: "demo")
    .withDesign(Design())
    .withData(Data())
    .combineWith(importer.importDocument(reader))
    .also { it: Document!
      exporter.exportDocument(it, writer)
    }
    .toPdf()
```

Prozedural: Kotlin - fluent api



Prozedural: Kotlin - extension functions

```kotlin
fun createDocument(template: String) = loadTemplate(template)
    .let(DocumentCreator::createDocument)

fun Document.withData(data: Data) = also { it: Document
  DataLoader.loadData(it, data)
}

fun Document.withDesign(design: Design) = also { it: Document
  Designer.prepareDesign(it, design)
}

fun Document.combineWith(vararg docs: Document) =
    DocumentCreator.combineDocuments( ...doc: this, *docs)

fun Document.toPdf() = let(Printer::printToPdf)
```

Prozedural: Java vs Kotlin

```java
Document doc;
Document doc2;

{ // erstes Dokument kreieren: load template > create doc > prepare design > load data
  Template template = TemplateLoader.loadTemplate( string: "demo");

  doc = DocumentCreator.createDocument(template);
  Designer.prepareDesign(doc, new Design());
  DataLoader.loadData(doc, new Data());
}
{ // zweites Dokument importieren
  doc2 = importer.importDocument(reader);
}

Document combinedDoc = DocumentCreator.combineDocuments(doc, doc2);

exporter.exportDocument(combinedDoc, writer);

PDF pdf = Printer.printToPdf(combinedDoc);
```

```kotlin
createDocument( template: "demo")
    .withDesign(Design())
    .withData(Data())
    .combineWith(importer.importDocument(reader))
    .also { it: Document!
      exporter.exportDocument(it, writer)
    }
    .toPdf()
```

areltis

Danke für die Aufmerksamkeit