ARCHUNIT

SOFTWARE-ARCHITEKTUR BEWAHREN

12.11.2022

Johannes Thorn

JOHANNES THORN

- Softwareentwickler
- Problemlöser

Schwerpunkte

- Java-Technologien
- Leichtgewichtige Architekturarbeit

Kontakt und Fragen

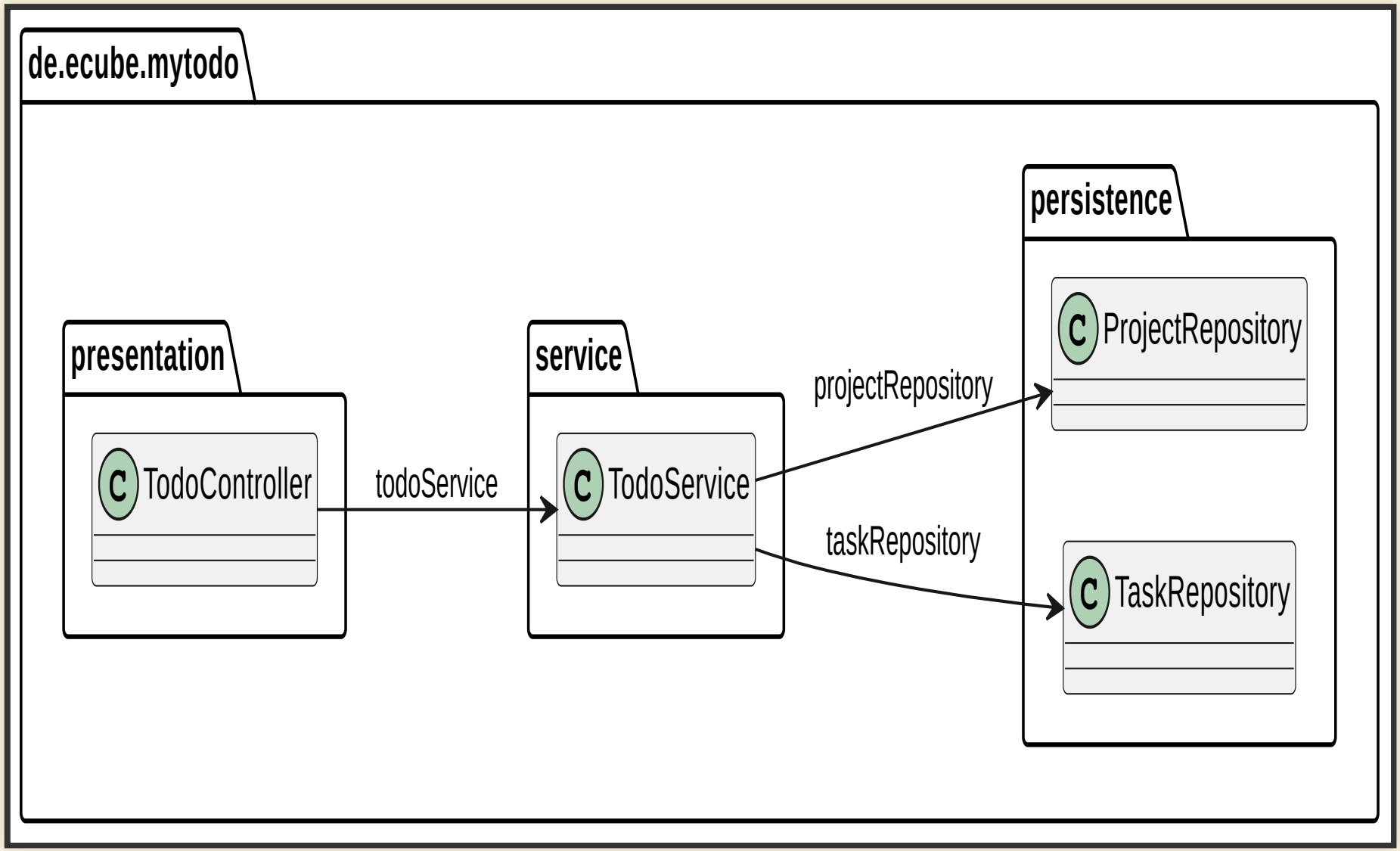
Johannes.Thorn@eCube.de
@HerrStachel

WAS IST SOFTWAREARCHITEKTUR

Die Architektur eines Softwaresystems besteht aus seinen Strukturen, der Zerlegung in Komponenten, deren Schnittstellen und Beziehungen untereinander.

Effektive Softwarearchitekturen (Hanser Verlag)
— Dr. Gernot Starke

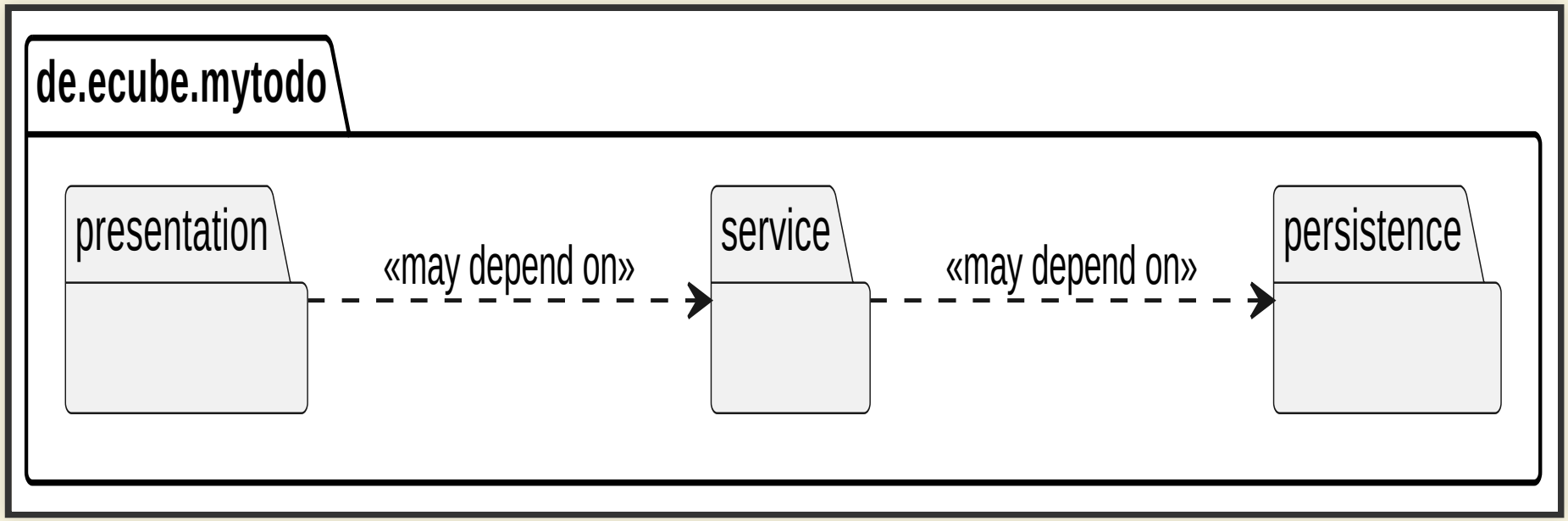
BEISPIEL FÜR SOLCHE STRUKTUREN



WARUM MACHEN WIR DAS ÜBERHAUPT?

1. Image by Steve Buissinne from Pixabay

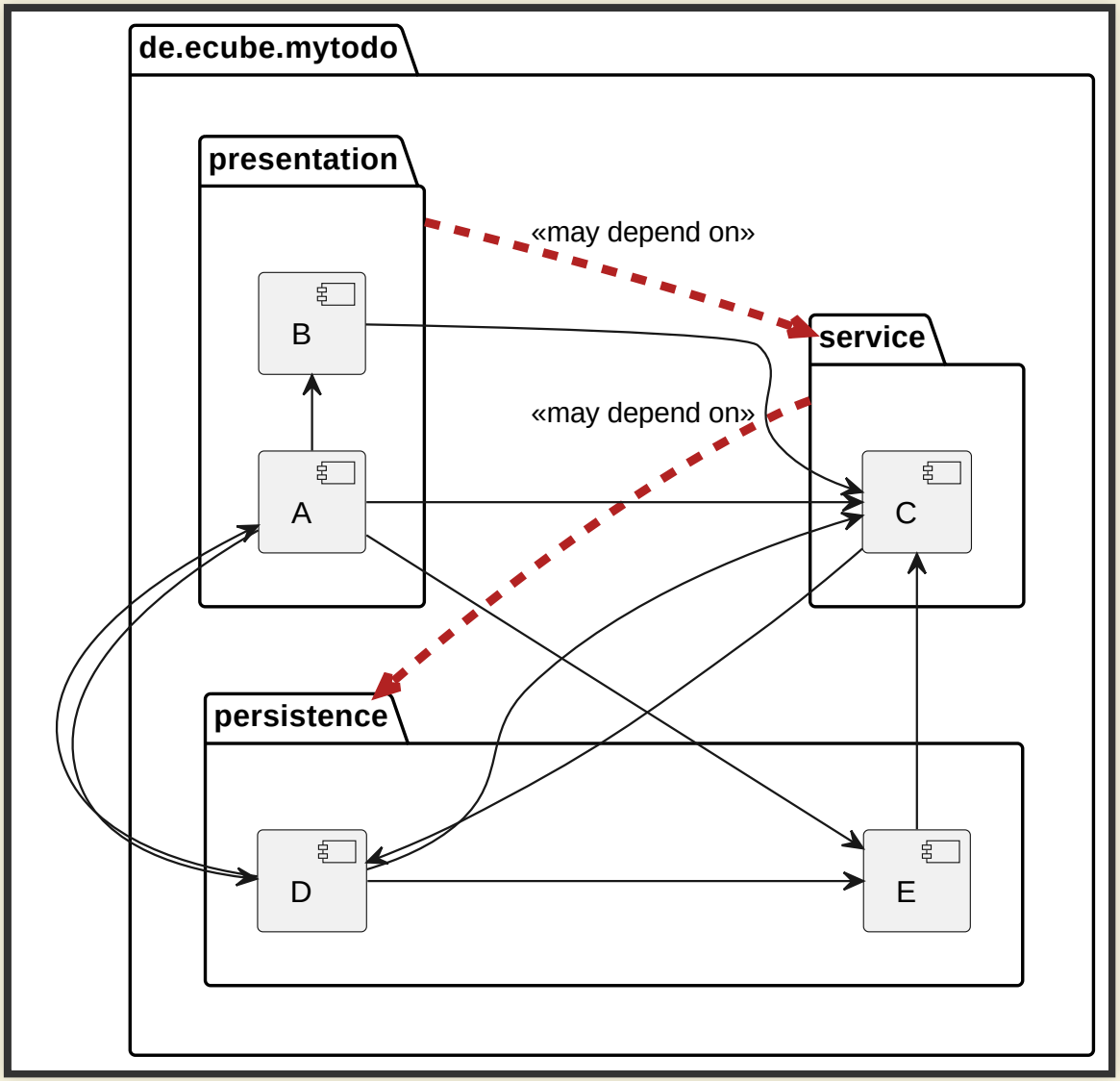
DER PLAN



- niemand darf auf `presentation` zugreifen
- nur `presentation` darf auf `service` zugreifen
- nur `service` darf auf `persistence` zugreifen

A FEW
MOMENTS LATER

DER SALAT


de.ecube.mytodo
presentation
«may depend on»
B
service
«may depend on»
A
C
persistence
D
E

DIE LÖSUNG

Macht es explizit!

ARCHUNIT ALS ANTWORT

- Eine Test-Bibliothek
- Testen / Validieren von *Architekturregeln* direkt im Code

Maven

```
<dependency>
    <groupId>com.tngtech.archunit</groupId>
    <artifactId>archunit</artifactId>
    <version>1.0.0</version>
    <scope>test</scope>
</dependency>
```

Gradle

```
dependencies {
    testImplementation 'com.tngtech.archunit:archunit:1.0.0'
}
```

UNSERE ERSTE REGEL

nur `presentation` darf auf `service` zugreifen

```
public class MyArchitectureTest {
  @Test
  public void service_should_only_be_accessed_by_presentation() {
    JavaClasses importedClasses = new ClassFileImporter() (1)
        .importPackages("de.ecube.mytodo");

    ArchRule rule = classes().that().resideInAPackage("..service..") (2)
        .should().onlyBeAccessed().byAnyPackage(
            "..presentation..", "..service..");

    rule.check(importedClasses); (3)
  }
}
```

UNSERE ERSTE REGEL MIT JUNIT 5

```
@AnalyzeClasses(packages = "com.ecube.mytodo") (1)
public class MyArchitectureTest {

  @ArchTest
  public static final ArchRule (2)
      service_should_only_be_accessed_by_presentation =
    classes().that().resideInAPackage("..service..")
      .should().onlyBeAccessed().byAnyPackage(
          "..presentation..", "..service..");

}
```

UNSER ERSTER FEHLSCHLAG

AssertionError

```
java.lang.AssertionError: Architecture Violation [Priority: MEDIUM] -
Rule
'classes that reside in a package '..service..'
should only be accessed by classes that reside in a package
'..presentation..', '..service..'
was violated (1 times):
Method <de.ecube.persistence.E.persist()>
calls method <de.ecube.service.C.doSomething()> in (E.java:14)
```

REGELBESCHREIBUNGEN

```
classes that reside in a package '..service..'
should only be accessed by classes that reside in a package
'..presentation..', '..service..'
```

VIOLATIONS

```
Method <de.ecube.persistence.E.persist()>
calls method <de.ecube.service.C.doSomething()> in (E.java:14)

Method <de.ecube.persistence.E.persist()>
calls method <de.ecube.service.C.doSomethingElse()> in (E.java:17)
```

MEHRSCHICHTIGER AUFBAU

Library API:
Abstraktionshilfen

Lang API:
Regeldefinitionen

Core API:
Basisfunktionalität

LIBRARY API

LAYERED ARCHITECTURE

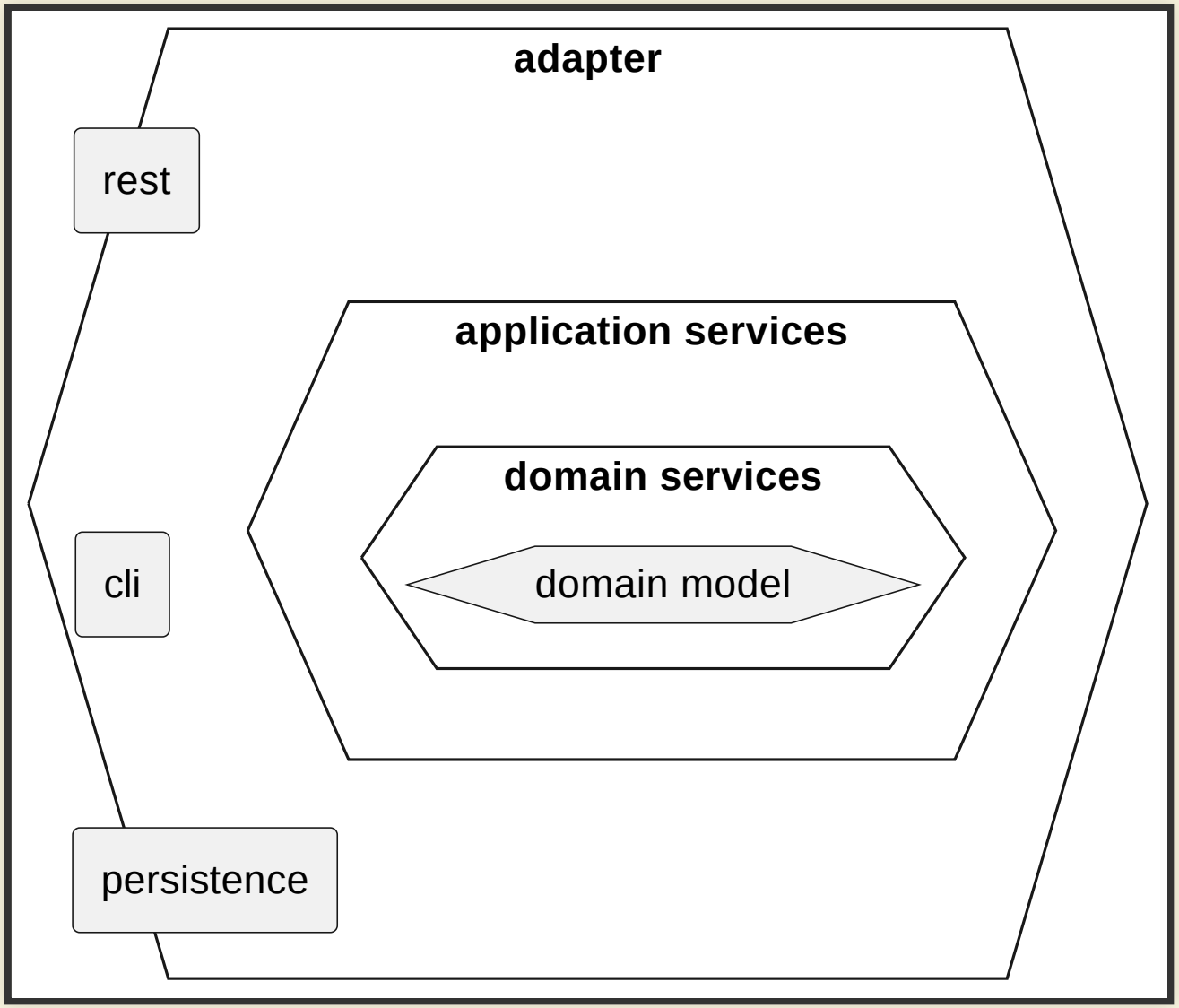
- niemand darf auf `presentation` zugreifen
- nur `presentation` darf auf `service` zugreifen
- nur `service` darf auf `persistence` zugreifen

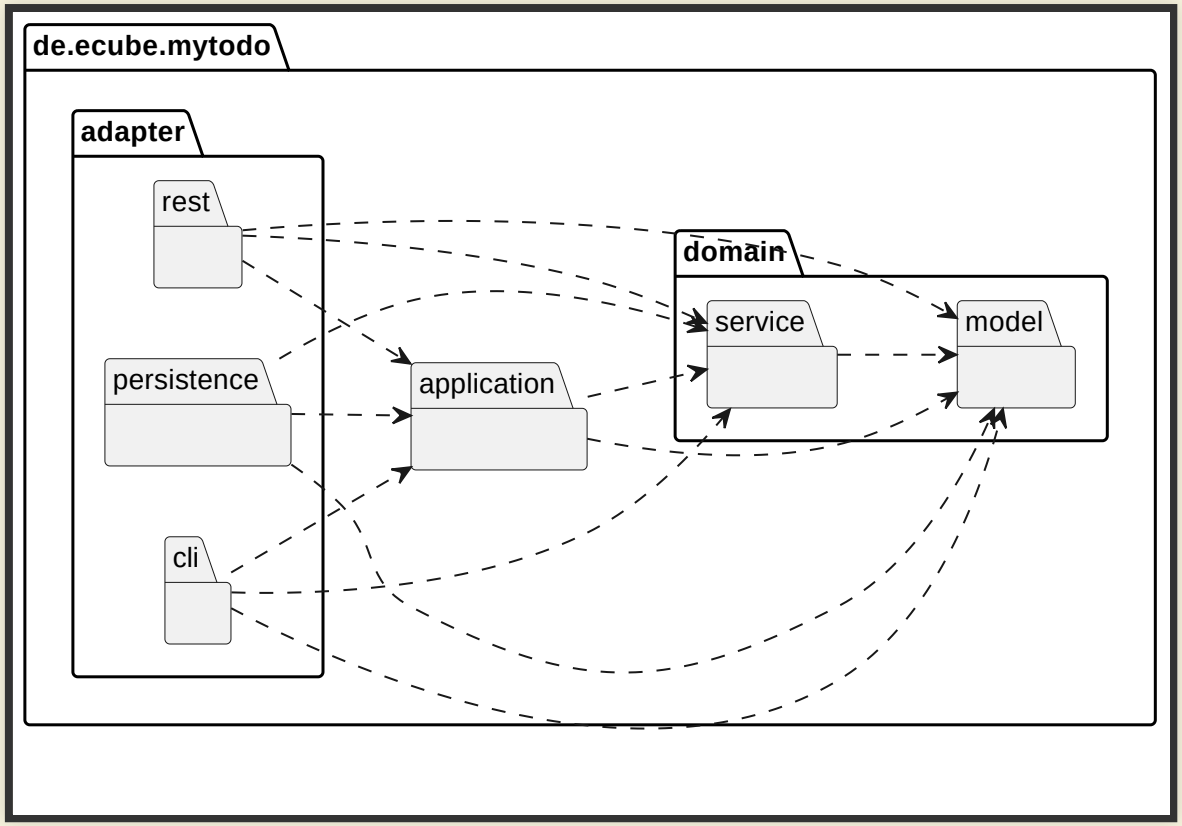
LAYERED ARCHITECTURE

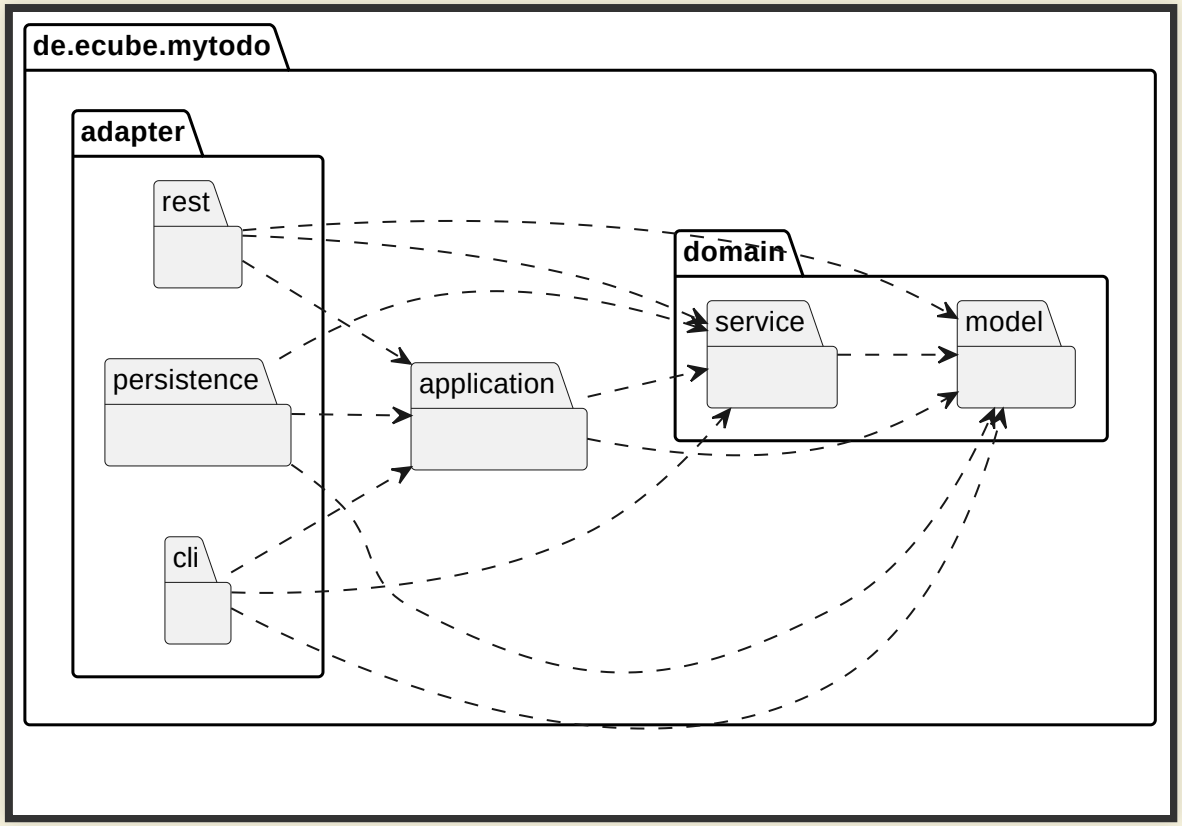
- *niemand darf auf `presentation` zugreifen*
- *nur `presentation` darf auf `service` zugreifen*
- *nur `service` darf auf `persistence` zugreifen*

```
layeredArchitecture().consideringAllDependencies()
    .layer("Presentation").definedBy("..presentation..")
    .layer("Service").definedBy("..service..")
    .layer("Persistence").definedBy("..persistence..")

    .whereLayer("Presentation").mayNotBeAccessedByAnyLayer()
    .whereLayer("Service").mayOnlyBeAccessedByLayers("Presentation")
    .whereLayer("Persistence").mayOnlyBeAccessedByLayers("Service");
```

ONION ARCHITECTURE (DDD)

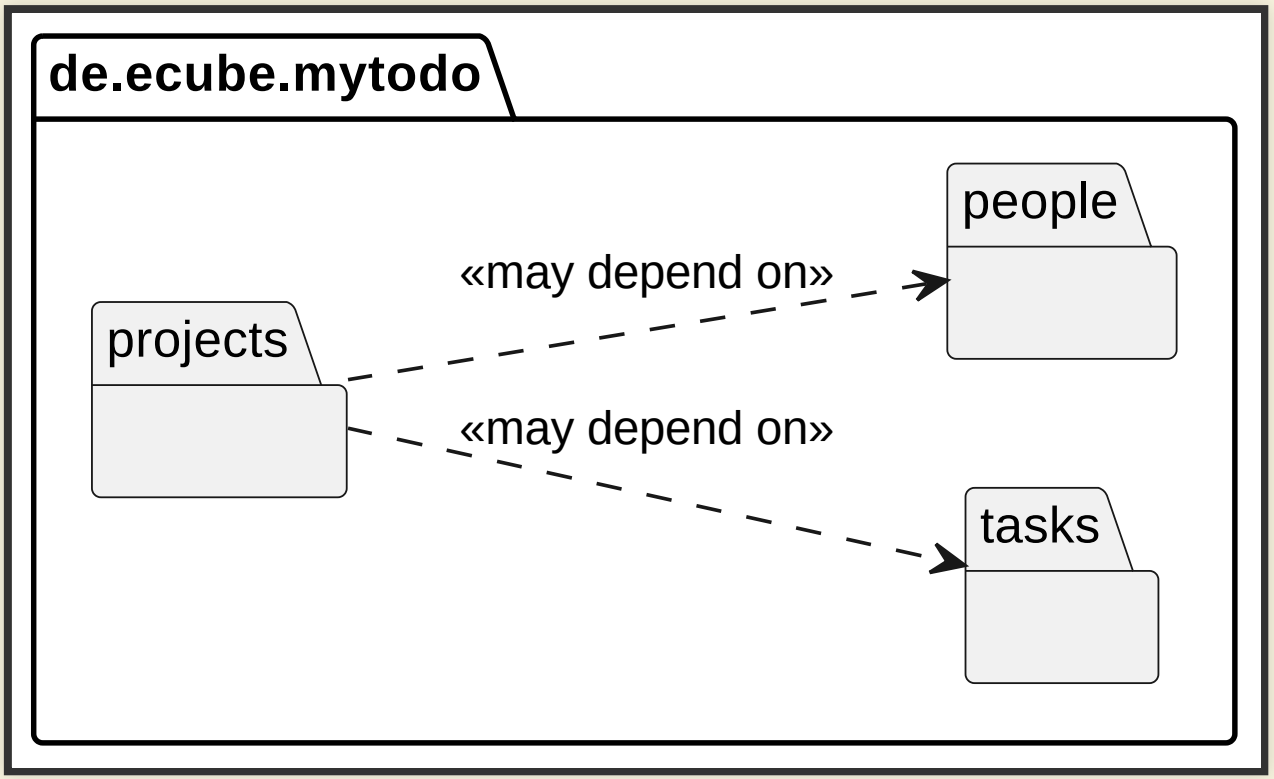



de.ecube.mytodo
adapter
rest
persistence
cli
application
domain
service
model

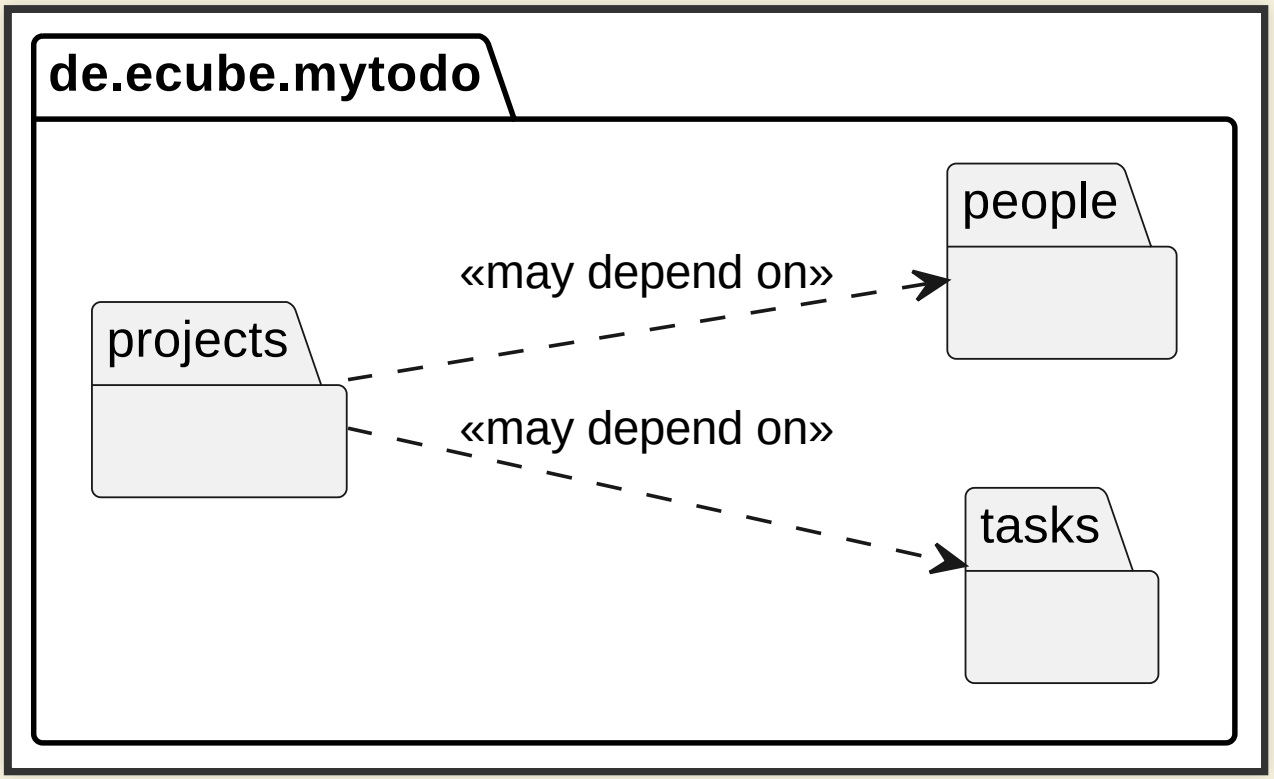



```
onionArchitecture()
    .domainModels("..domain.model..")
    .domainServices("..domain.service..")
    .applicationServices("..application..")
    .adapter("cli", "..adapter.cli..")
    .adapter("persistence", "..adapter.persistence..")
    .adapter("rest", "..adapter.rest..");
```

VERTIKALE SLICES



VERTIKALE SLICES




```
SlicesRuleDefinition.slices().matching("..mytodo.(*)..")
    .should().beFreeOfCycles()

SlicesRuleDefinition.slices().matching("..mytodo.(*)..")
    .should().notDependOnEachOther()
```

PLANTUML ALS EINGABESPRACHE

my-architecture.puml

```
@startuml
left to right direction

[Presentation] <<..presentation..>>
[Service] <<..service..>>
[Persistence] <<..persistence..>>

[Presentation] --> [Service]
[Service] --> [Persistence]
@startuml
```

my-architecture.puml

```
@startuml
left to right direction

[Presentation] <<..presentation..>>
[Service] <<..service..>>
[Persistence] <<..persistence..>>

[Presentation] --> [Service]
[Service] --> [Persistence]
@startuml
```

```
URL myArchitecture = getClass().getResource("my-architecture.puml");

classes().should( adhereToPlantUmlDiagram(
    myArchitecture,
    consideringAllDependencies()
));
```

"ALLGEMEINGÜLTIGE" REGELN

- Verwendung von Logging statt `System.out` oder `System.err`
- Es wird keine generische `Exception` direkt geworfen
- Keine Verwendung von Field-Injection
- Klassen dürfen nicht von Eltern-Paketen abhängen

```
@ArchTest
ArchRule no_access_to_standard_streams =
    GeneralCodingRules.NO_CLASSES_SHOULD_ACCESS_STANDARD_STREAMS;

...

@ArchTest
ArchRule no_field_injection =
    GeneralCodingRules.NO_CLASSES_SHOULD_USE_FIELD_INJECTION;
```

ARCHITEKTUR VS. DESIGN

$$\text{Architektur} := \sum \text{wichtige Entscheidungen}$$

ARCHITEKTUR VS. DESIGN

$$\text{Architektur} := \sum \text{wichtige Entscheidungen}$$

Design ist der Rest

LANG API

Wir schreiben uns unsere Regeln einfach selbst!!!

REGELAUFBAU

1. Einstiegspunkt
2. Predicates
3. Conditions

KLASSEN

Positive Aussage

```
classes()
    .that().haveSimpleNameContaining("Controller")
    .should().resideInAPackage("..adapter.rest..");
```

Negative Aussage

```
noClasses()
    .should().accessClassesThat().resideInAPackage("..internal..");
```

METHODEN

Positive Aussagen

```
methods()
    .that().areAnnotatedWith(GetMapping.class).and().arePublic()
    .should().haveRawReturnType(ResponseEntity.class);
```

Negative Aussagen

```
noMethods()
    .that().areDeclaredInClassesThat().haveNameMatching(".*Dao")
    .should().declareThrowableOfType(SQLException.class);
```

FELDER

Positive Aussagen

```
fields()
    .that().areAnnotatedWith(Id.class)
    .should().haveNameEndingWith("Id");
```

Negative Aussagen

```
noFields().should().bePublic();
```

WEITERE EINSTIEGSPUNKTE

`constructors()`

Konstruktoren

`codeUnits()`

ausführbare Code-Blöcke

`members()`

kombiniert Felder, Methoden und Konstruktoren

EIGENE PREDICATES UND CONDITIONS

Kombination vorhandener Konzepte

```
DescribedPredicate<JavaClass> serializableNamedFoo =
    simpleName("Foo").and(assignableTo(Serializable.class));
```

EIGENE PREDICATES UND CONDITIONS

Kombination vorhandener Konzepte

```
DescribedPredicate<JavaClass> serializableNamedFoo =
    simpleName("Foo").and(assignableTo(Serializable.class));
```

Selbst implementiert

```
DescribedPredicate<JavaClass> areSpecial = (1)
    new DescribedPredicate<JavaClass>("are special"){

    @Override
    public boolean apply(JavaClass input) {
        // Decide if class is special via some rather complex logic
        return Random.nextBoolean();
    }
};

classes().that(areSpecial).should()...; (2)
```

KEINE REGEL OHNE GRUND :D

```
classes().that(haveAFieldAnnotatedWithPayload)
    .should(onlyBeAccessedBySecuredMethods)
    .as("Payload may only be accessed in a secure way")
    .because("@Secured methods will be intercepted");
```

CORE API

IMPORTER

- Einen Paketbaum importieren

```
JavaClasses importedClasses = new ClassFileImporter()
    .importPackage("com.ecube.mytodo");
```

- Den gesamten ClassPath importieren

```
JavaClasses importedClasses = new ClassFileImporter()
    .importclassPath();
```

- Class-Dateien importieren

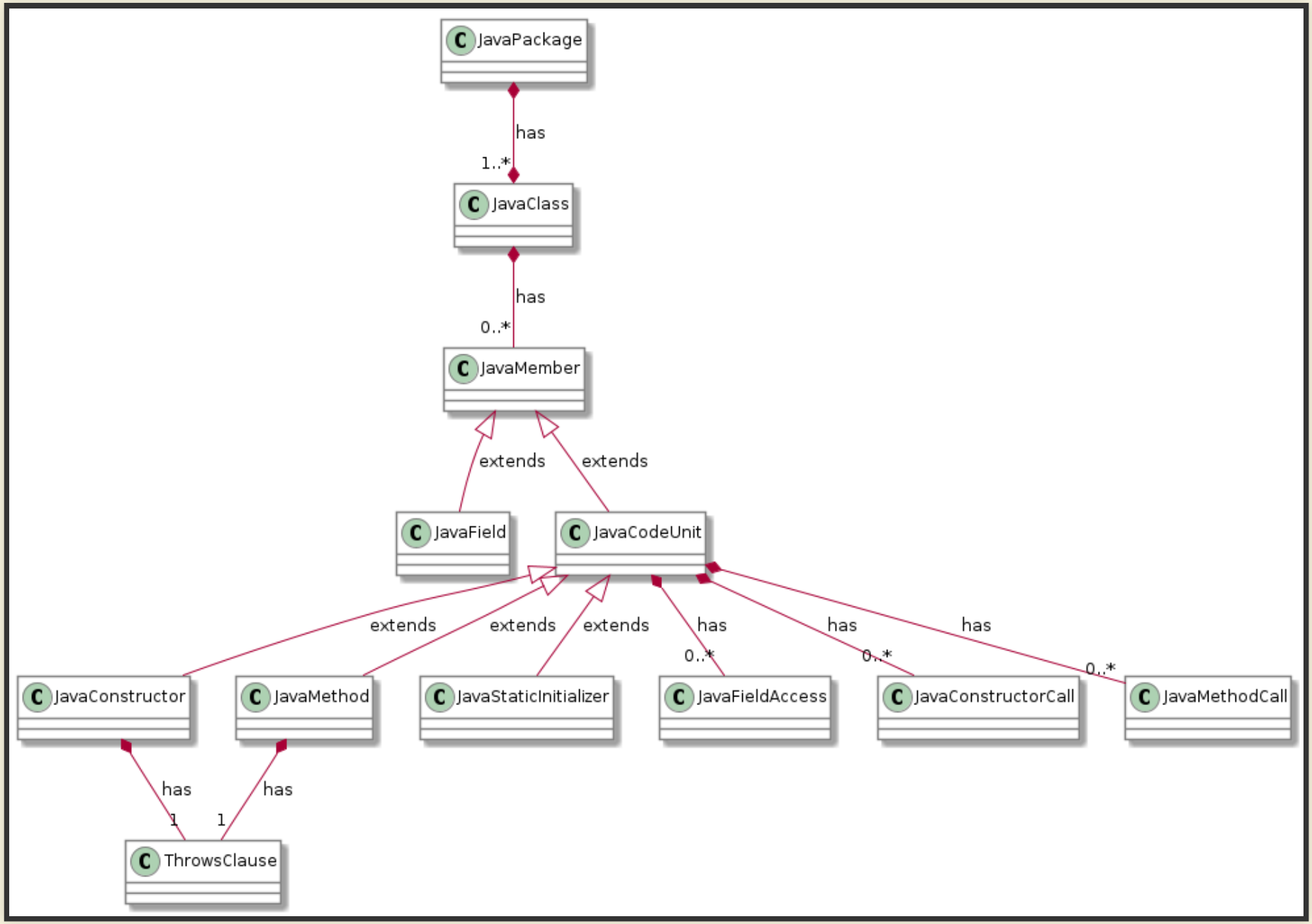
```
JavaClasses importedClasses = new ClassFileImporter()
    .importPath("../otherProject/target/classes");
```

IMPORTER OPTIONEN

```
JavaClasses importedClasses = new ClassFileImporter()
    .withImportOption(ImportOption.Predefined.DO_NOT_INCLUDE_JARS)
    .withImportOption(ImportOption.Predefined.DO_NOT_INCLUDE_TESTS)
    .importClasspath();
```

DOMÄNEN-MODELL

- Repräsentation des Java-Codes
- Nach dem Vorbild der Reflection-API


JavaPackage
has
1..*
JavaClass
has
0..*
JavaMember
extends
extends
JavaField
JavaCodeUnit
extends
extends
extends
has
has
has
0..*
0..*
0..*
JavaConstructor
JavaMethod
JavaStaticInitializer
JavaFieldAccess
JavaConstructorCall
JavaMethodCall
has
has
1
1
ThrowsClause

WEITERE FUNKTIONEN

VORGEHEN IM BESTANDSPROJEKT

2. Image by Thomas Ulrich from Pixabay

KONTINUIERLICHE VERBESSERUNGEN

Beginnt mit wenigen Regeln, die euch wichtig sind.

FROZEN RULES

Hilfe ich bin im LegacyLand gefangen. ALLES ist eine Violation.

FROZEN RULES

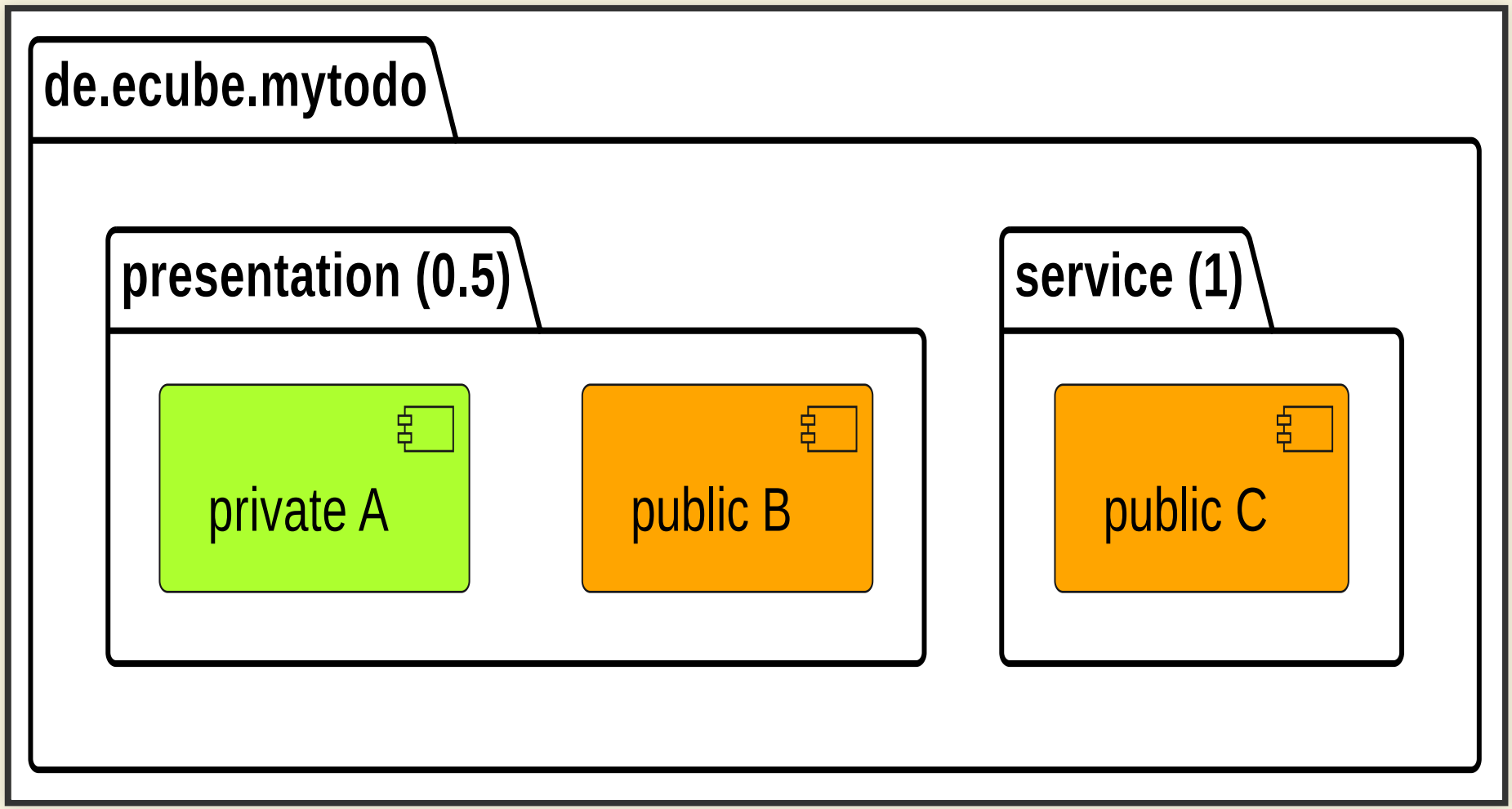
Hilfe ich bin im LegacyLand gefangen. ALLES ist eine Violation.

```
ArchRule rule = FreezingArchRule.freeze(
    classes().should().bePerfect()
);
```

IGNORIEREN VON BESTIMMTEN VIOLATIONS

SOFTWARE-ARCHITEKTUR-METRIKEN

u.a. Visibility Metrics von Herbert Dowalil



ARCHITEKTUR-ÜBERWACHUNG

- Die Maschine sagt "Nein!"
- "Der Gerät wird nie müde"

TIPPS ZUM START

- Das ist kein Alleingang!
- Startet im Team!
- Just Enough Design statt Big Design Up Front

ANDERE SPRACHEN?

- ArchUnitNET für C# und weitere
- TSArch für TypeScript

BONUSRUNDE!

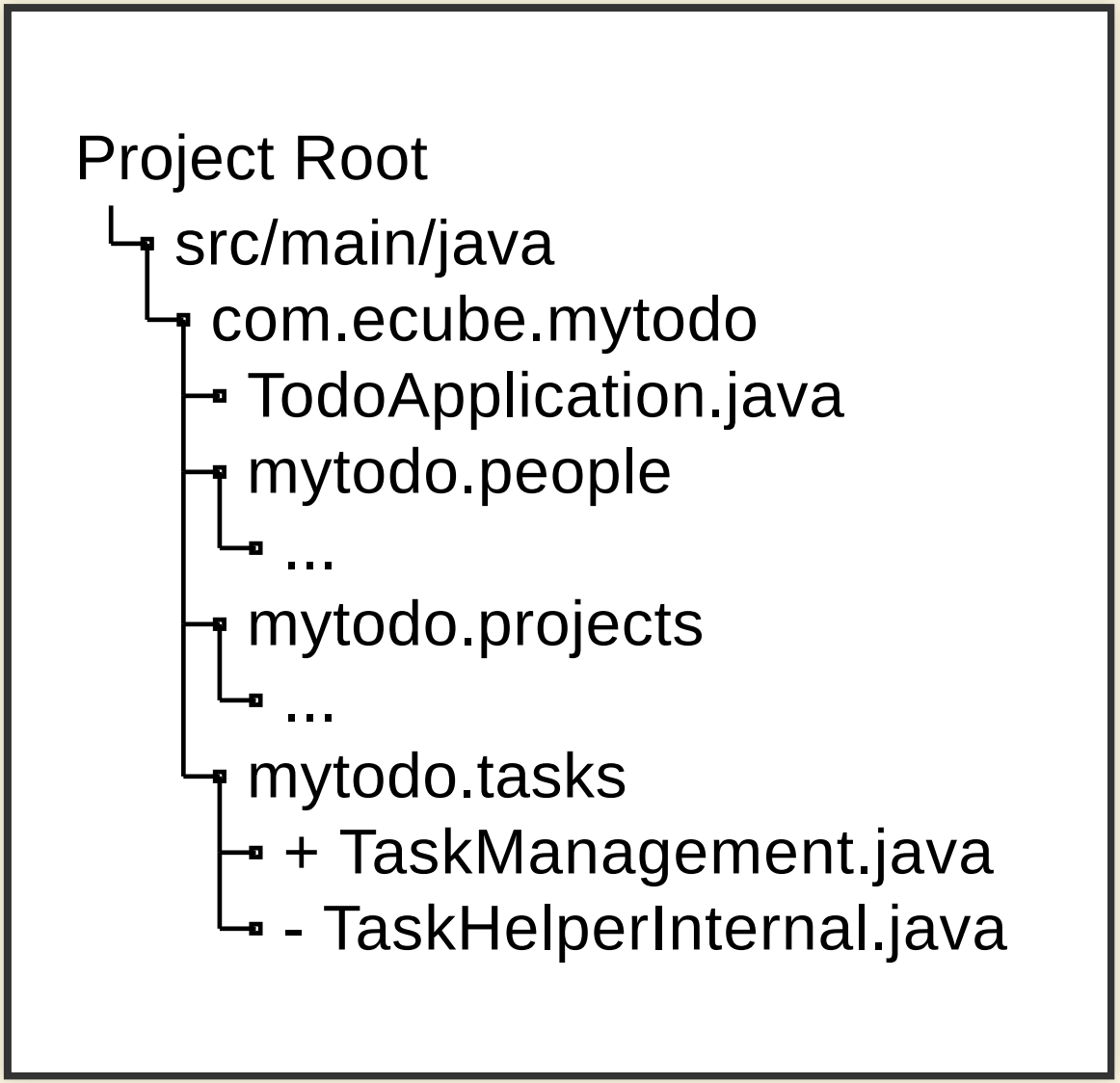
JMOLECULES

```
import org.jmolecules.ddd.annotation.*;

@Entity
class BankAccount { /* ... */ }

@ValueObject
class IBAN { /* ... */ }

@ValueObject
record Currency { /* ... */ }

@Repository
class Accounts { /* ... */ }
```

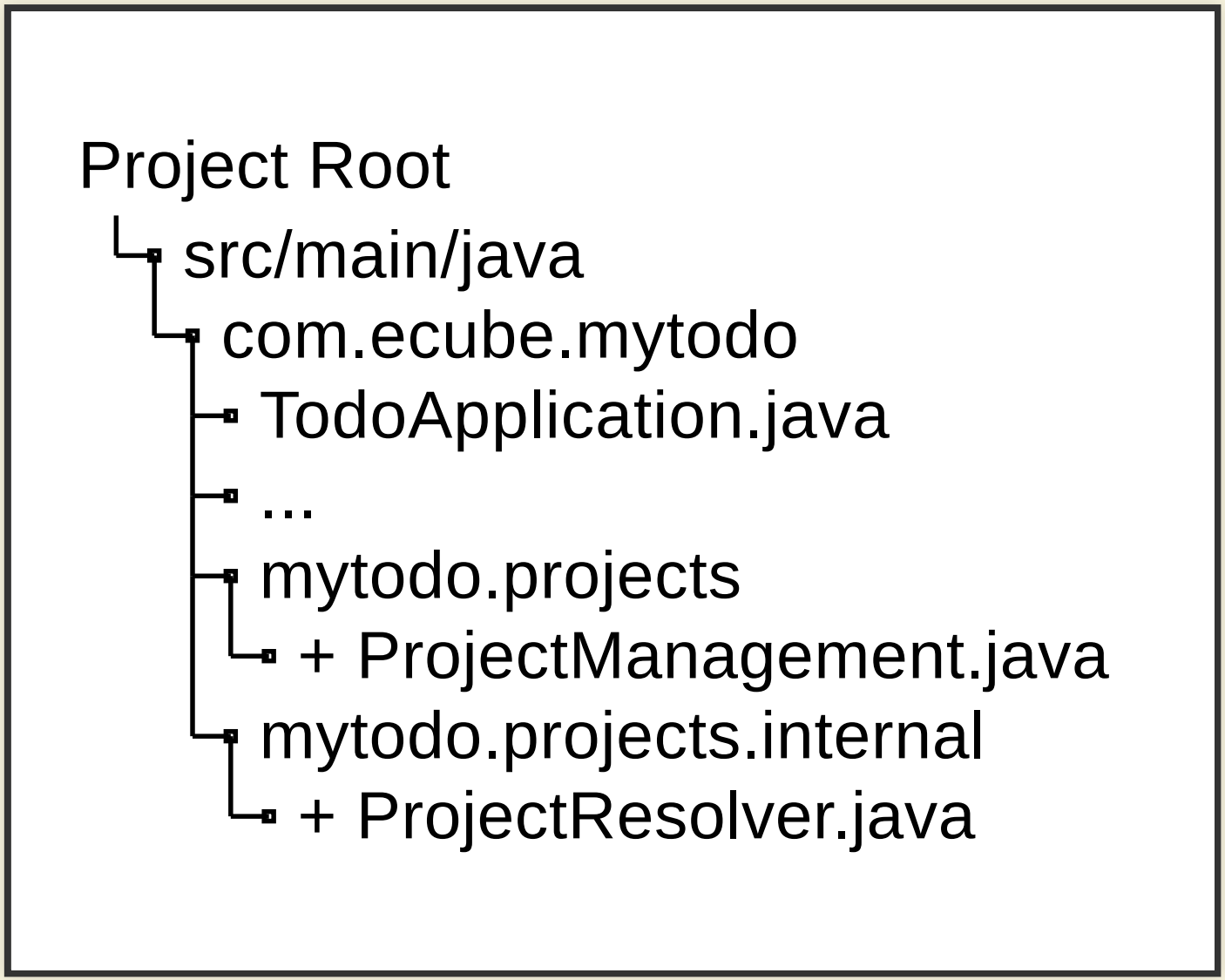
SCHICHTEN UND RINGE

```
import org.jmolecules.architecture.layered.*;

@DomainLayer
package org.acmebank.domain;

@ApplicationLayer
package org.acmebank.application;
```

SPRING MODULITH



KOMPLEXE MODULE



FRAGEN AN EUCH!

- Habt ihr schonmal eine ähnliche Situation wie zu Beginn geschildert erlebt?
- Wie seit ihr dem begegnet?