Karlsruher Institut für Technologie

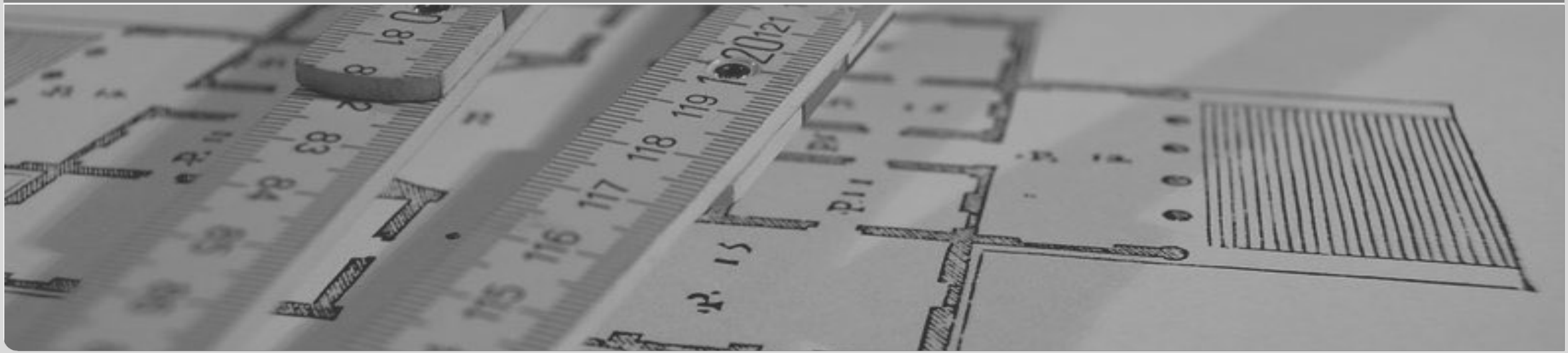
Java-Schnittstelle für die Instanziierung von EMF-Modellen

Praktikum: Werkzeuge für Agile Modellierung

Louisa Lambrecht

Betreuer: Yves Richard Kirschner

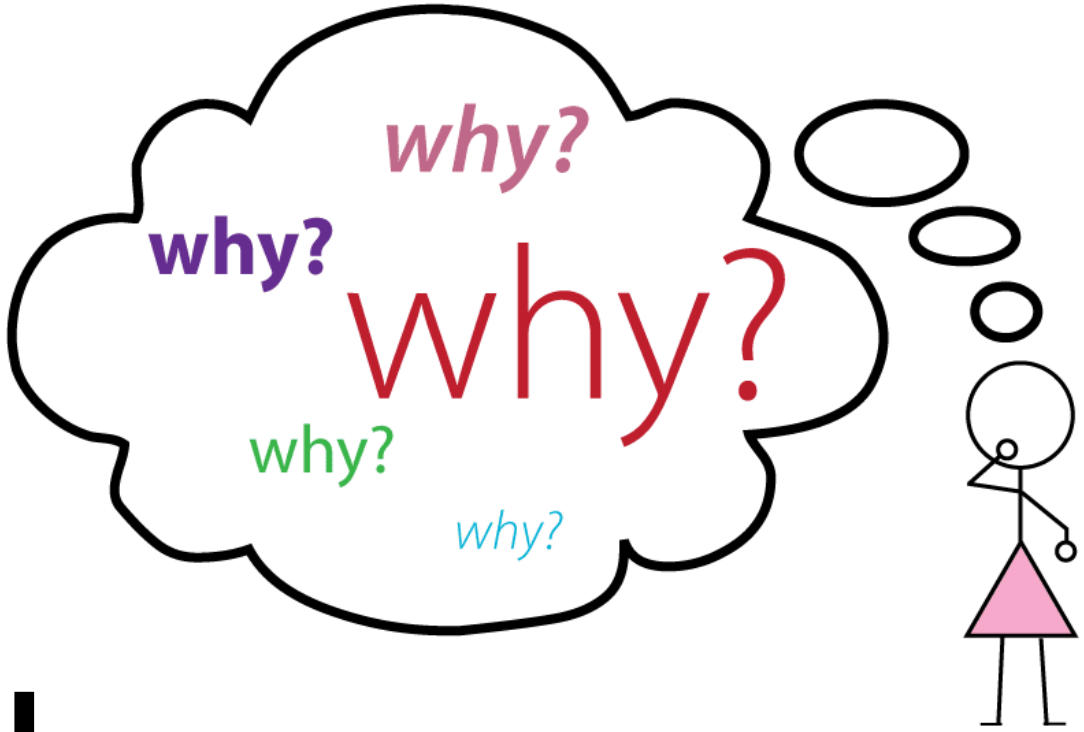
ARCHITECTURE-DRIVEN REQUIREMENTS ENGINEERING,
INSTITUT FÜR PROGRAMMSTRUKTUREN UND DATENORGANISATION, KIT-FAKULTÄT FÜR INFORMATIK



Inhalt

- Motivation
- Grundlagen
  - Palladio Repository Model
  - Fluent Interfaces
- Palladio Fluent API Model Generator
- Ergebnis

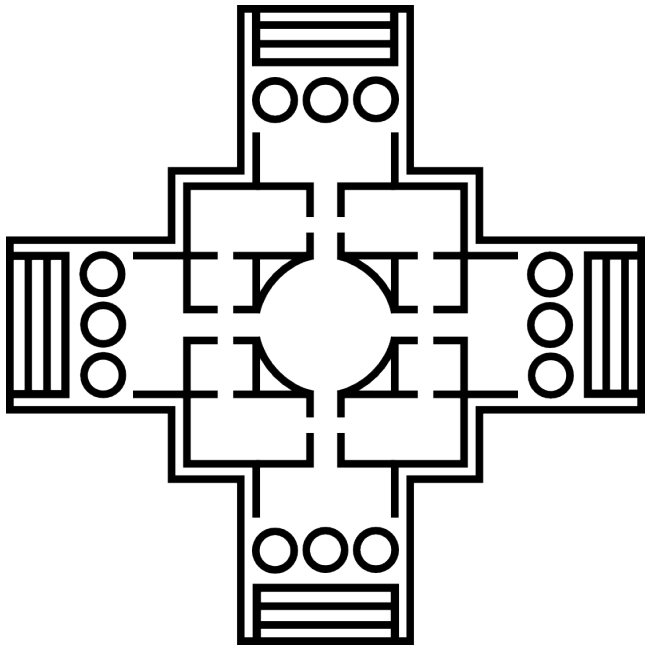
MOTIVATION



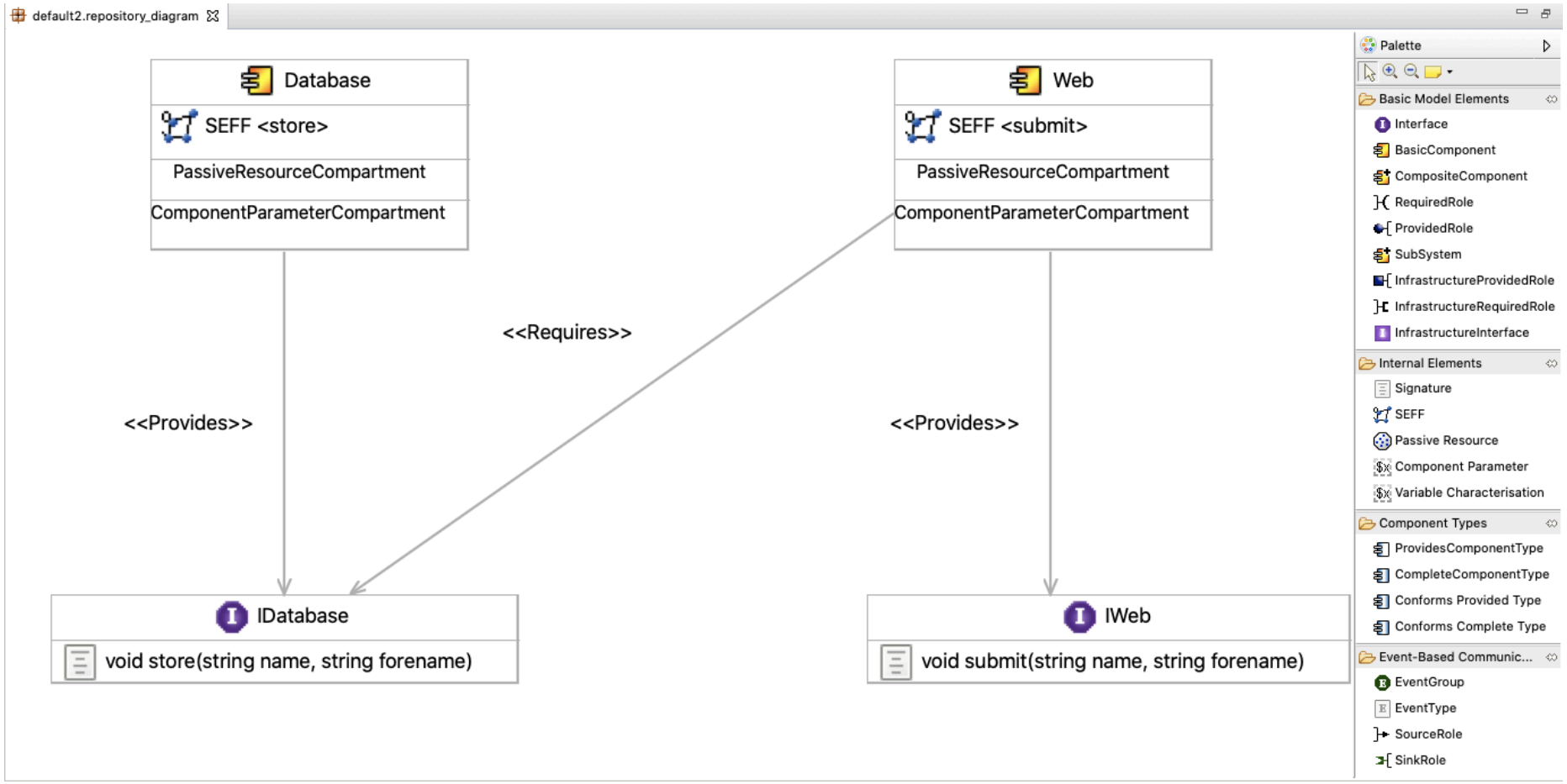
Motivation

- Graphische Editoren sind
  - für Einsteiger geeignet
  - anschaulich
  - bei großen Modellen schnell unübersichtlich
  - ermüdend durch das „Herumklicken“

- Komplexe Objekte programmatisch zu erzeugen ist
  - unübersichtlich
  - unverständlich
  - schreibintensiv
  - schneller



PALLADIO REPOSITORY MODEL



Erstellung einer Palladio Repository Model Instanz mit Hilfe des graphischen Editors

Ein Beispiel

Ein Design Prinzip

FLUENT INTERFACES

Fluent Interfaces

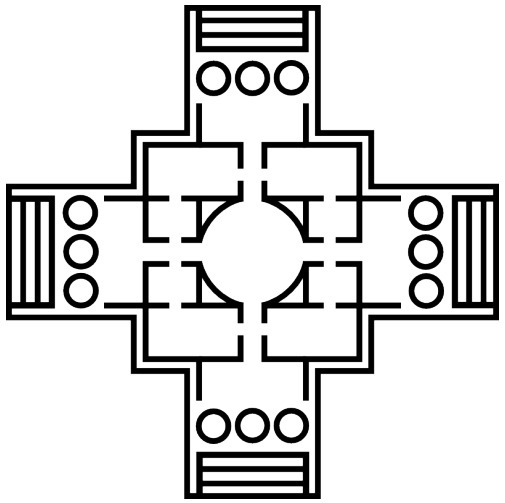
- Besonders nützlich, um Objekte zu erstellen und zu manipulieren
- Ziel: lesbarer, kompakter Code
- Methodenverkettung, die sich wie ein natürlich-sprachlicher Satz liest
- Gibt einen Rahmen vor und bietet eine natürliche Intuition für verfügbare Features
- Beispiele: Java Stream API, JMock

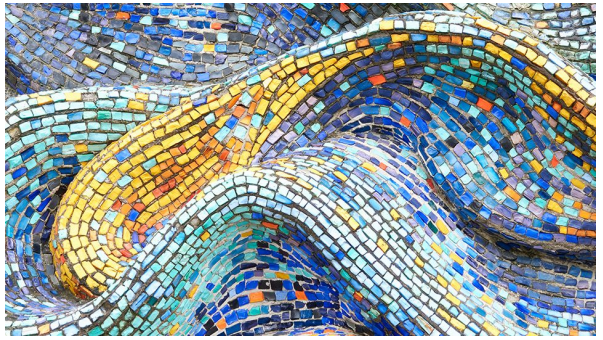
Fluent Interfaces: Beispiel

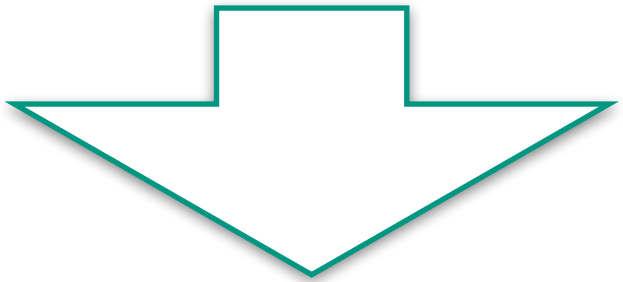
```
private void makeNormal(Customer customer) {
    Order o1 = new Order();
    customer.addOrder(o1);
    OrderLine line1 = new OrderLine(6, Product.find("TAL"));
    o1.addLine(line1);
    OrderLine line2 = new OrderLine(5, Product.find("HPK"));
    o1.addLine(line2);
    OrderLine line3 = new OrderLine(3, Product.find("LGV"));
    o1.addLine(line3);
    line2.setSkippable(true);
    o1.setRush(true);
}
```

```
private void makeFluent(Customer customer) {
    customer.newOrder()
            .with(6, "TAL")
            .with(5, „HPK“).skippable()
            .with(3, „LGV")
            .priorityRush();
```

Quelle: https://martinfowler.com/bliki/FluentInterface.html (Martin Fowler)



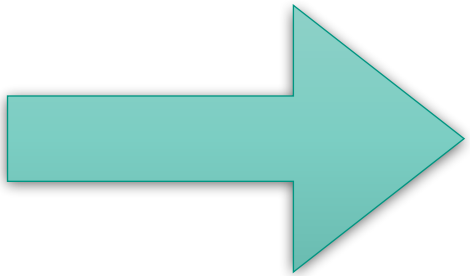




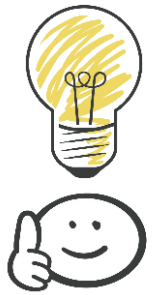
PALLADIO FLUENT API MODEL GENERATOR

Herangehensweise

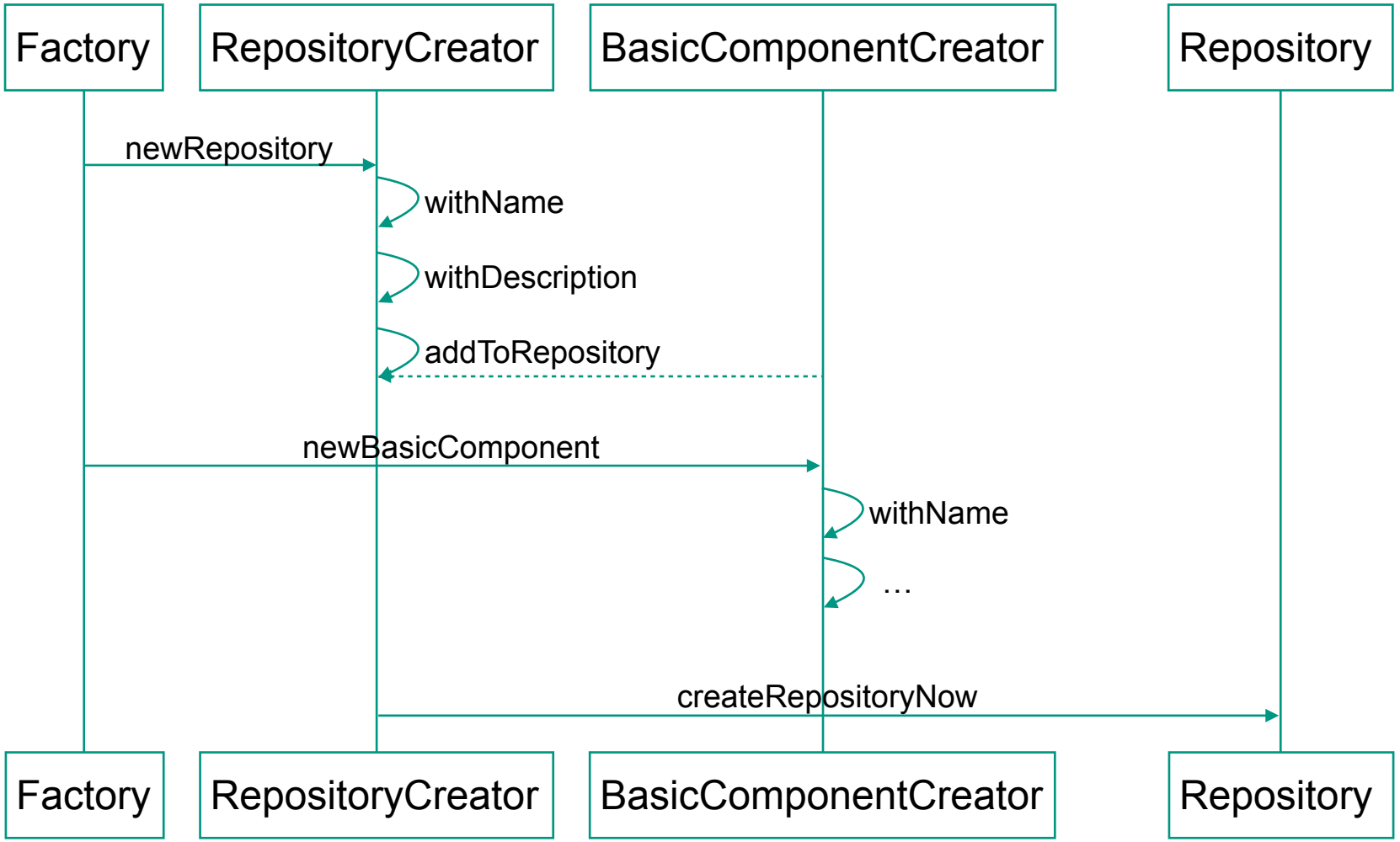
“Create a new repository model. Add to this repository a basic component with name ‘Database‘.“



```
185        FluentRepositoryFactory create = new FluentRepositoryFactory();
186
187        Repository repository = create.newRepository()
188                .addToRepository(create.newBasicComponent()
189                        .withName("Database"))
190                .createRepositoryNow();
```

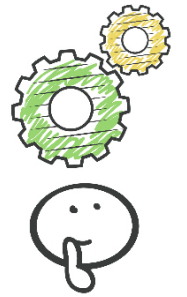


Fluent Interface Grammatik



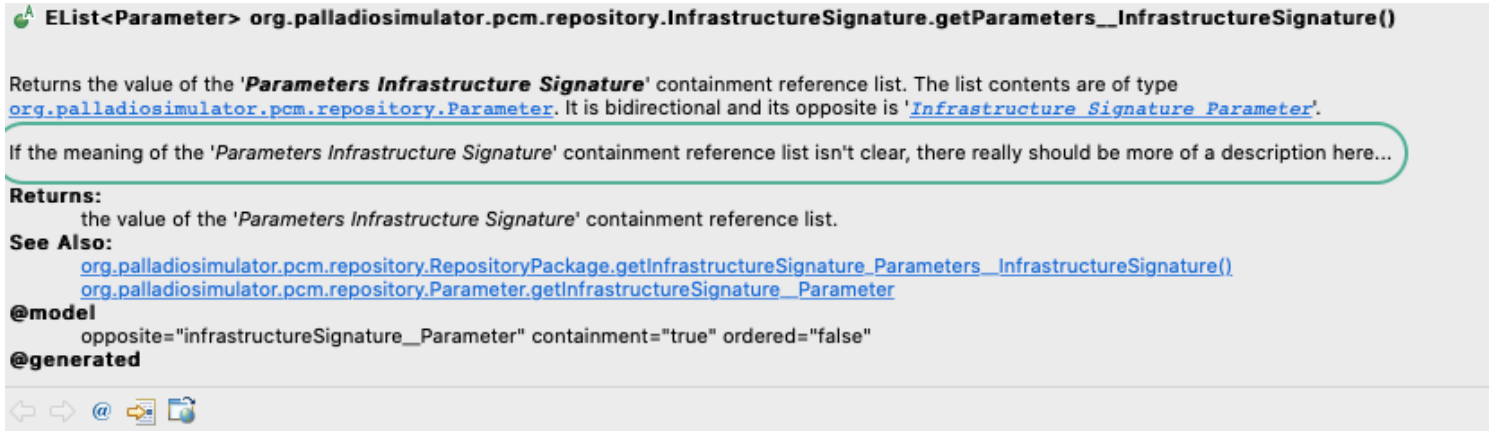
Methodenverkettung:
a().b().c()

Argument:
a(x)



Herausforderungen

- Verständnis des Meta-Models



- Design Entscheidungen
  - Erstellen der Modell-Elemente: Factory vs. Methodenverkettung
  - Reihenfolge der Methodenverkettung: strikt vs. frei
  - Methoden dürfen gesetzte Werte überschreiben

ERGEBNIS

Aufbau der FluentRepositoryFactory

- Die Factory erstellt
  - das Repository
  - Hauptelemente: Components, Interfaces, Data Types, Failure Types
  - komplexe interne Elemente: SEFFs, Signatures, Variable Usages

- Die Factory ermöglicht den Zugriff auf alle bereits erstellten Entitäten

Palladio Fluent API Model Generator

- Fluent API
  - zur Erstellung von Palladio Repository Objekten
  - mit nur einer Schnittstelle/Factory
  - mit einem klaren Rahmen der möglichen Eigenschaften durch die entwickelte Grammatik
    - ➔ auch für Einsteiger geeignet
- Vollständige Dokumentation (JavaDoc)
- Umfassende Projektbeschreibung und Einführung mit Beispielen (Readme)

```java
public static void readmeExampleBackend() {
    // Factory
    RepositoryFactory repoFact = RepositoryFactory.eINSTANCE;
    // Repository
    Repository repository = repoFact.createRepository();

    // Database component
    BasicComponent databaseComponent = repoFact.createBasicComponent();
    databaseComponent.setEntityName("Database");

    // IDatabase interface
    OperationInterface databaseInterface = repoFact.createOperationInterface();
    databaseInterface.setEntityName("IDatabase");

    // Signature store
    OperationSignature store = repoFact.createOperationSignature();
    store.setEntityName("store");
    // with parameters forename, name
    Parameter forename = repoFact.createParameter();
    forename.setParameterName("forename");
    forename.setDataType__Parameter(null); // referencing the imported data types poses another problem
    Parameter name = repoFact.createParameter();
    name.setParameterName("forename");
    name.setDataType__Parameter(null);

    // Providing connection from Database component to IDatabase interface
    OperationProvidedRole dbProvIDb = repoFact.createOperationProvidedRole();
    dbProvIDb.setProvidedInterface__OperationProvidedRole(databaseInterface);
    dbProvIDb.setProvidingEntity_ProvidedRole(databaseComponent);

    // Seff for Database component on service store
    ResourceDemandingSEFF storeSeff = SeffFactory.eINSTANCE.createResourceDemandingSEFF();
    storeSeff.setDescribedService__SEFF(store);
    databaseComponent.getServiceEffectSpecifications__BasicComponent().add(storeSeff);

    // Adding component + interfaces to the repository
    repository.getComponents__Repository().add(databaseComponent);
    repository.getInterfaces__Repository().add(databaseInterface);

    // Web component
    BasicComponent webComponent = repoFact.createBasicComponent();
    databaseComponent.setEntityName("Web");

    OperationInterface webInterface = repoFact.createOperationInterface();
    databaseInterface.setEntityName("IWeb");

    OperationSignature submit = repoFact.createOperationSignature();
    submit.setEntityName("submit");
    // with parameters forename, name
    Parameter forename2 = repoFact.createParameter();
    forename2.setParameterName("forename");
    forename2.setDataType__Parameter(null);
    Parameter name2 = repoFact.createParameter();
    name2.setParameterName("forename");
    name2.setDataType__Parameter(null);

    OperationProvidedRole webProvIweb = repoFact.createOperationProvidedRole();
```

```java
public static void readmeExampleFluentAPI() {
    // Factory
    FluentRepositoryFactory create = new FluentRepositoryFactory();

    Repository repository = create.newRepository()
            // Database
            .addToRepository(create.newOperationInterface().withName("IDatabase")
                    .withOperationSignature(create.newOperationSignature()
                            .withName("store")
                            .withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
                            .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
            .addToRepository(create.newBasicComponent().withName("Database")
                    .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("store")))
                    .provides(create.fetchOfOperationInterface("IDatabase")))
            // Web
            .addToRepository(create.newOperationInterface().withName("IWeb")
                    .withOperationSignature(create.newOperationSignature()
                    .withName("submit").withParameter("forename", Primitive.STRING, ParameterModifier.NONE)
                    .withParameter("name", Primitive.STRING, ParameterModifier.NONE)))
            .addToRepository(create.newBasicComponent().withName("Web")
                    .withServiceEffectSpecification(create.newSeff().onSignature(create.fetchOfSignature("submit")))
                    .provides(create.fetchOfOperationInterface("IWeb"))
                    .requires(create.fetchOfOperationInterface("IDatabase")))
            .createRepositoryNow();

    saveRepository(repository, "./", "fluentAPIExample.repository", false);
}
```

Palladio Fluent API Model Generator

Beispiel: Web und Database Komponenten mit ihren Schnittstellen

FRAGEN?