

MDSD in der Praxis

Ralph Winzinger, Senacor

Agenda 08.01.2014

- ⌚ Vorstellung R. Winzinger, Senacor & MDSD Projekte
- ⌚ DSLs textuell & grafisch
- ⌚ Übungsblock 1: getting started with Xtext grammars
- ⌚ Hello World
- ⌚ Service/Logic/Entity
- ⌚ Q&A, Diskussion

Agenda 15.01.2014

- ⌚ Übungsblock 2: some more Xtext and Xtend
- ⌚ Editor: Validierungen, Hilfestellungen, Formatierung, ...
- ⌚ Xtend: Sprachkonstrukte, (Code-)Generierung
- ⌚ Best Practices
- ⌚ Q&A, Diskussion

Ralph Winzinger & Senacor

RALPH WINZINGER
PRINCIPAL ARCHITECT

 SENACOR

Beratungsschwerpunkte

- Enterprise Architektur
- Legacy-Integration/-Migration
- Software-Engineering
- JEE, SOA, Webservices, SAP Integration, OSGi, Frontendtechnologien, Mobile



Branchenfokus

- Banking

Ausbildung und berufliche Erfahrung

- Über 10 Jahre Erfahrung in der Anwendungsentwicklung, technischen Projektleitung und Architekturberatung im Bankenumfeld
- Architekt, Senacor Technologies AG (seit 2000)
- Freiberufliche Tätigkeit im Bereich Mobile Computing (1996 – 2002)
- Diplom in Informatik, Univ. Erlangen-Nürnberg

Ralph Winzinger & sonst



Senacor & MDSD

Warum DSLs?

Wer hat schon mal mit  DSLs gearbeitet?

Wer wusste vorher, dass  mit einer DSL arbeitet?

DSLs sind vielleicht nicht überall,
aber durchaus weit verbreitet.

Warum DSLs?

```
internalOnly (true|false) "false">

<!ELEMENT characteristic EMPTY>
<!ATTLIST characteristic
  name CDATA #REQUIRED
  value CDATA #REQUIRED>
]>

<processmodel>
  <domain name="Baufi" id="1">

    <module name="Vorgang" id="1">
      <state name="n.a." id="1">
        <transition name="externBearbeiten" state="extern in Bearbeitung"/>
        <transition name="bearbeiten" state="inBearbeitung"/>
        <transition name="internerVKBearbeiten" state="interner VK bearbeitet"/>
      </state>

      <state name="extern in Bearbeitung" maps="vertrieb-inBearbeitung">
        <transition name="uebergeben" state="uebergeben"/>
        <transition name="interner VK in Bearbeitung" state="interner VK bearbeitet"/>
      </state>

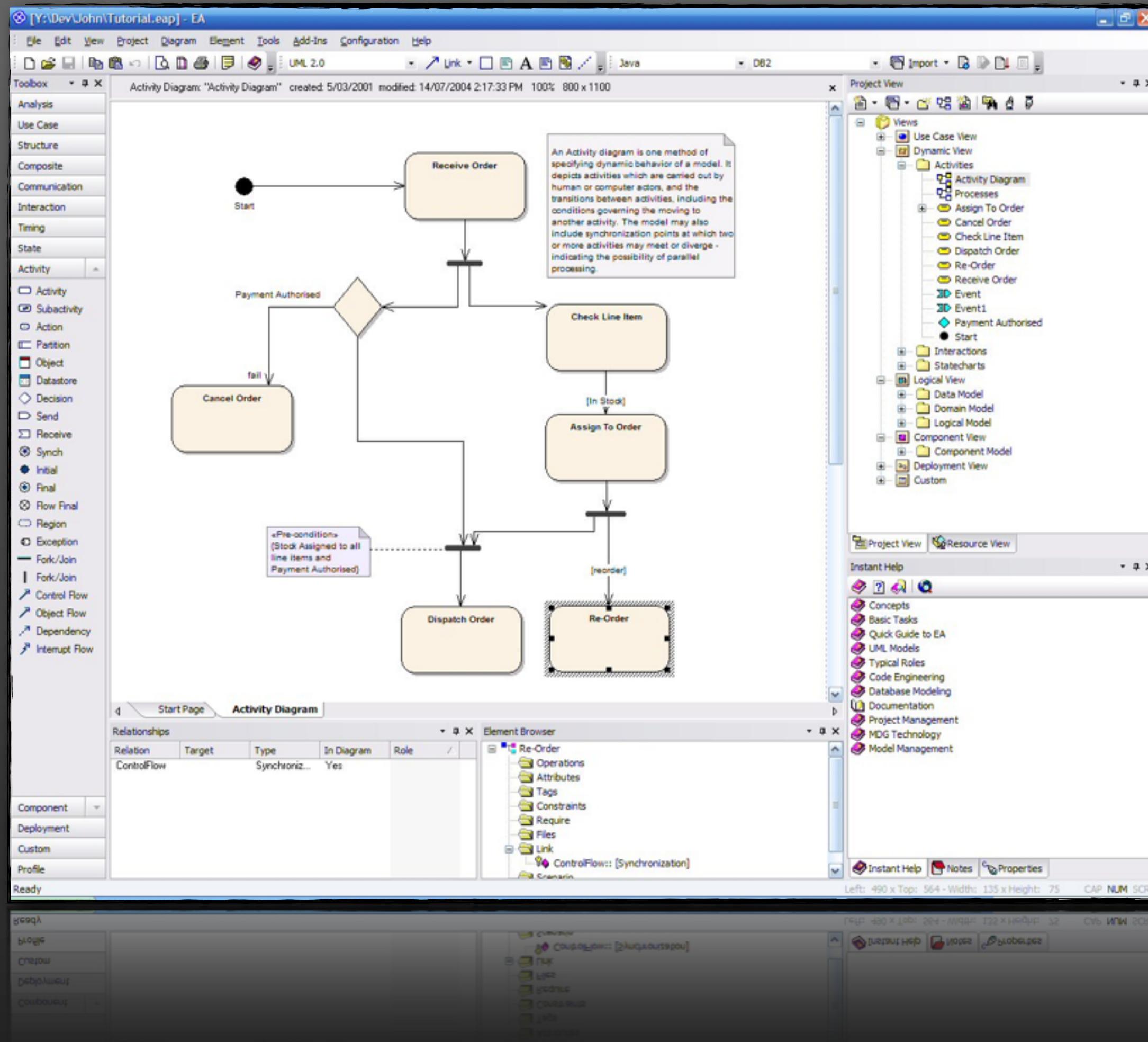
      <state name="uebergeben" id="3">
        <dependency module="PreScoring" state="vollstaendig"/>
        <transition name="bearbeiten" state="inBearbeitung"/>
      </state>

      <state name="frei" id="4">
        <transition name="bearbeiten" state="inBearbeitung"/>
        <transition name="zurueckstellen" state="zurueckgestellt"/>
      </state>
    </module>
  </processmodel>
```

⌚ Abstraktion von eingesetzter Technologie

⌚ Annäherung an fachliches Vokabular, gemeinsame Diskussionsbasis

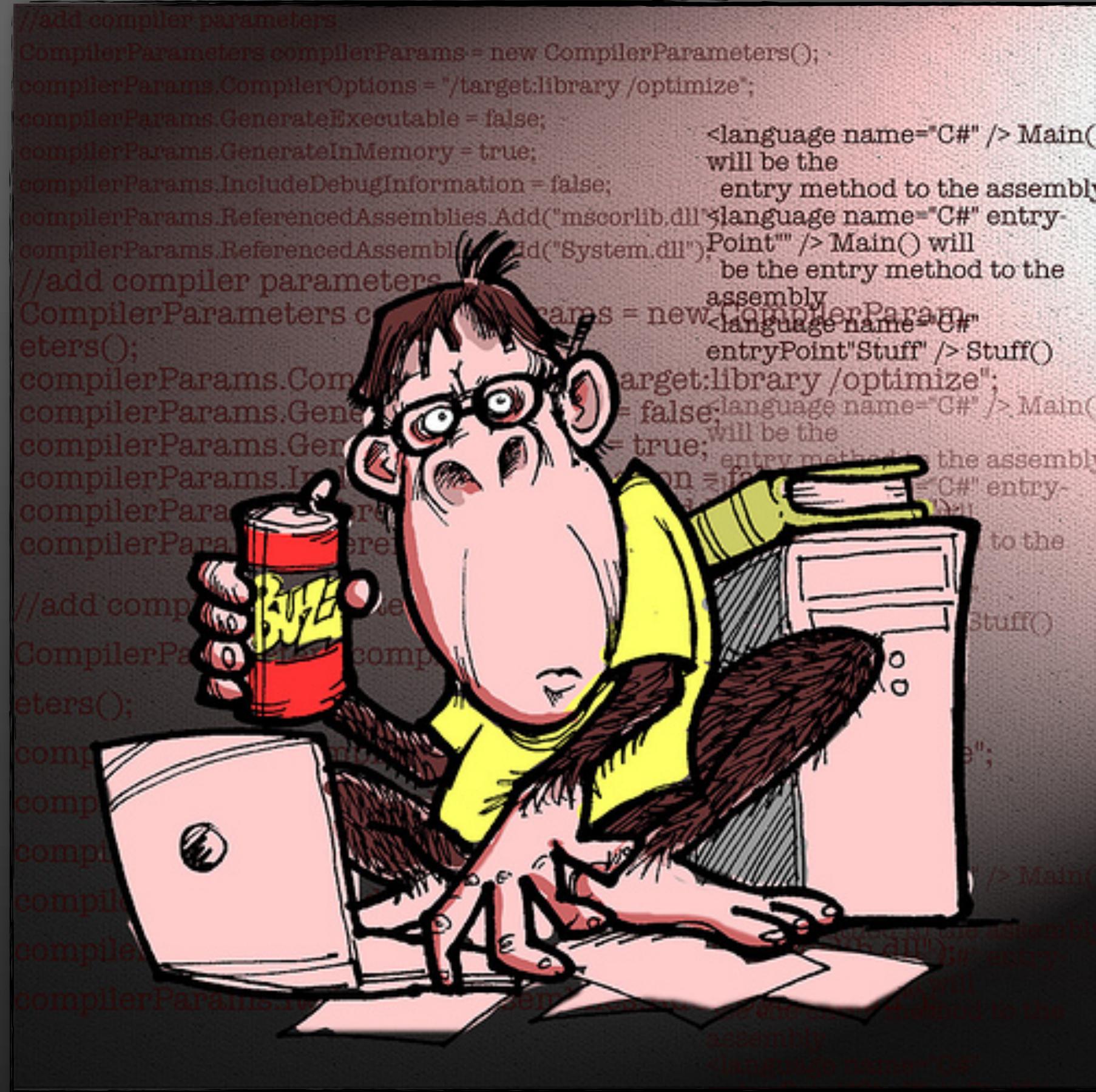
Warum DSLs?



☞ Dokumentation

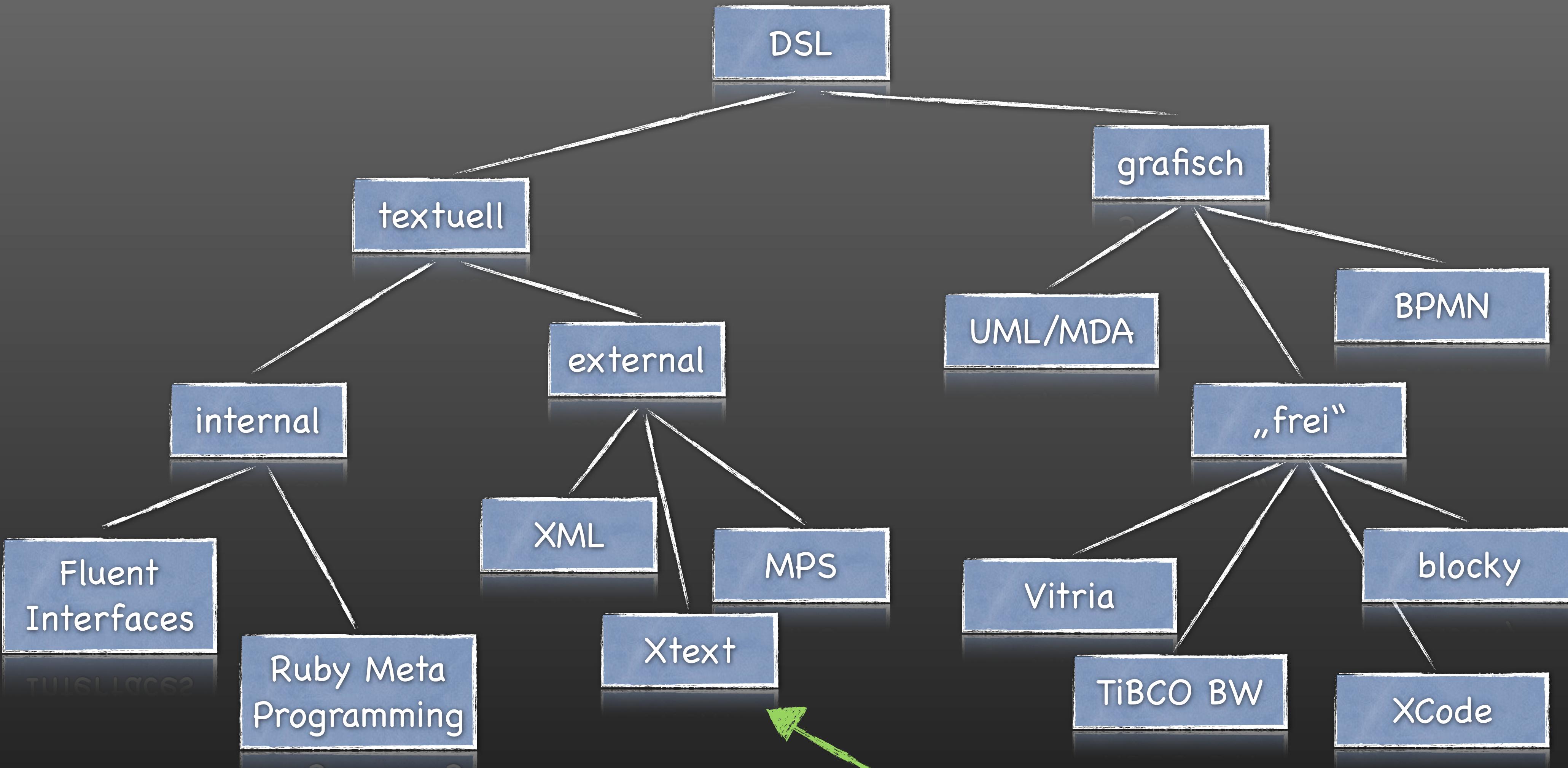
☞ Definition von
Sachverhalten mit
komplexen
Strukturen

Warum DSLs?



Dont
Repeat
Yourself

DSLs haben viele Gesichter



Welche DSL
und weshalb nun eigentlich?

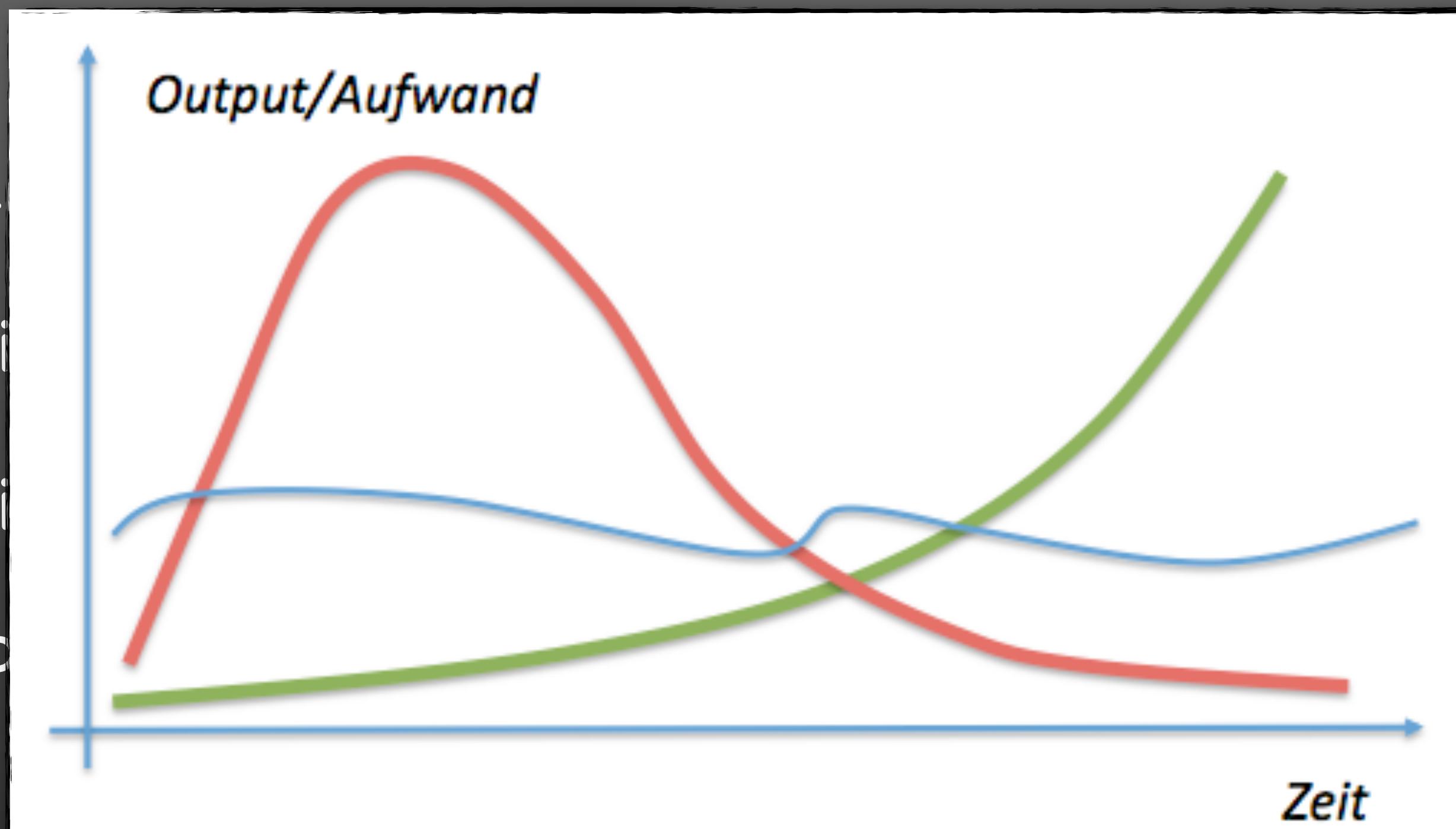
~~do obey „the law of the instrument“!~~

„do we have a problem here?“

- Lern- bzw. Effizienzkurve
- Problem von hinreichender Größe
- Problem von hinreichender Dauer
- Spricht Domain-/Projekt-/Sprachgröße für GPL oder DSL?

„do we have a problem here?“

- Lern- bzw. Effizienzproblem
- Problem von hinreichendem Output/Aufwand
- Spricht Domain-/Problem



561f

Grafisch oder Textuell?

- ⌚ Grafische DSL abstrahiert und vereinfacht - kann sie dem Problem gerecht werden?
- ⌚ Können komplexe Sachverhalte überhaupt sinnvoll dargestellt werden?
- ⌚ Können Lösungen effizient beschrieben werden?

Grafisch oder Textuell?

The image displays three distinct examples of programming interfaces:

- Left Panel (Scratch-like):** A graphical programming environment with a script editor. The script is a "repeat while [true]" loop containing a "move forward" block and several conditional blocks ("if not wall to the left", "else if wall ahead", etc.). Below the script is a "Move" block panel with settings for Port (B), Direction (Up), and Steering (C). On the left, there's a timeline showing waveforms for HCLK, HSEL, HADDR, HTRANS, HREADY, and HREADYOUT.
- Middle Panel (Block-based):** A graphical interface for a microcontroller. It features a grid workspace with various blocks (e.g., sensors, actuators, logic) and a toolbar on the right. A specific block, labeled "CB", is highlighted.
- Right Panel (Textual/Flowchart):** A process flow diagram using BPMN-like notation. It includes:
 - A central process box labeled "BusinessPartnerProcess".
 - An external input "FromSapBpTriggerQ" leading to the process.
 - An output "ToSapBpQ" from the process.
 - Four parallel gateway boxes labeled "FromSapBpQ1", "FromSapBpQ2", "FromSapBpQ3", and "FromSapBpQ4" receiving inputs from the process.
 - A final process box "BusinessPartnerStatusProcess" receiving an input "CompletionCallback" from one of the parallel gateways.

Grafisch oder Textuell?

The screenshot illustrates the trade-off between graphical user interfaces and textual domain-specific languages (DSLs) in software development.

Left Side (Graphical User Interface):

- A screenshot of a web-based application showing the **Nichtamtliches Inhaltsverzeichnis** (Non-governmental content index).
- The main heading is **§ 102 Ergänzende Leistungen**.
- Text content describes winter subsidies for workers, mentioning amounts like 2,50 Euro and 1,00 Euro.
- Navigation links include **zum Seitenanfang** and **Datenschutz**.

Right Side (Textual DSL Definition):

- A screenshot of the **DSL-Editor v31.3.0 - BA-DSL-Perspektive**.
- The project tree shows nodes like **Funktion_EGVd_Eingliederu**, **Funktion_EGv_Einglied**, and **Maske_ANZb_AnrechnungszeitBearbeiten**.
- The code editor displays a textual DSL definition for a mask (**Maske_ANZb_AnrechnungszeitBearbeiten**). The code includes:
 - A title: `funktion Funktion_ANZb_AnrechnungszeitBearbeiten titel "Anrechnungszeit bearbeiten - & #Person.anzeigeName"`
 - An initialization section: `maskenmuster eingabe initialisieren %ANZb_AnrechnungszeitBearbeiten_Initialisieren`
 - A description: `beschreibung "Diese Maske zeigt Informationen einer Anrechnungszeit zur Bearbeitung an"`
 - A help section: `praxishilfe "anrechnungszeitenauflisten"`
 - A conditional section: `abschnitt Abschnitt_Anrechnungszeit titel wenn #IstAnrechnungszeitAV = ja dann "Anrechnungszeit Arbeitsvermittlung" sonst "Anrechnungszeit Berufsberatung"`
 - Date ranges: `daten Startdatum #Anrechnungszeit.startdatum (beschriftung "Beginn MAZ-Zeitraum")` and `daten Endedatum #Anrechnungszeit.enddatum (beschriftung "Ende MAZ-Zeitraum")`
 - Motivation: `daten Meldungsgrund #Anrechnungszeit.meldungsgrund (beschriftung "Meldungsgrund" vorbelegung #Anrechnungszeit.meldungsgrund inhalt wenn #IstAnrechnungszeitAV=ja dann $BLA_AV sonst $BLA_BB status wenn #IstAnrechnungszeitAV=ja dann aktiv sonst anzeigen"`
- The bottom right shows a preview window for **Maske_ANZb_AnrechnungszeitBearbeiten.html** with fields for **Beginn MAZ-Zeitraum**, **Ende MAZ-Zeitraum**, and **Meldungsgrund**.

Tooling

- ⌚ Bei grafischen DSLs in der Regel vorgegeben, keine Anpassungsmöglichkeiten
- ⌚ Maturity
- ⌚ Benutzbarkeit (Zielgruppe!)
- ⌚ Integration in den Entwicklungsprozess
- ⌚ Refactoring

Collaboration & Versioning

- Sind Modellierer Teamplayer oder Einzelkämpfer?
- Diff & Merge, insbesondere bei grafischen DSLs
- Wird Versionierung angemessen unterstützt?
- Wird gewähltes Vorgehen unterstützt?

Modell- & Templatedefinition

- ☛ Top down, nicht auf der grünen Wiese
- ☛ mit dem Fachbereich DSL anhand tatsächlicher Problemstellungen erarbeiten
- ☛ Templates entwickeln, nachdem hinreichend viele Use-Cases prototypisch umgesetzt wurden und daraus generische Codeanteile extrahiert werden können

nicht zu kurz denken ...

- ⌚ Nicht nur primäre Artefakte modellieren/generieren!
- ⌚ Tests
- ⌚ Spezifikation, Dokumentation
- ⌚ Querschnittliche Aspekte
- ⌚ ...
- ⌚ Aber nur soweit sinnvoll!

<CODE>

Umgebung & Technologie



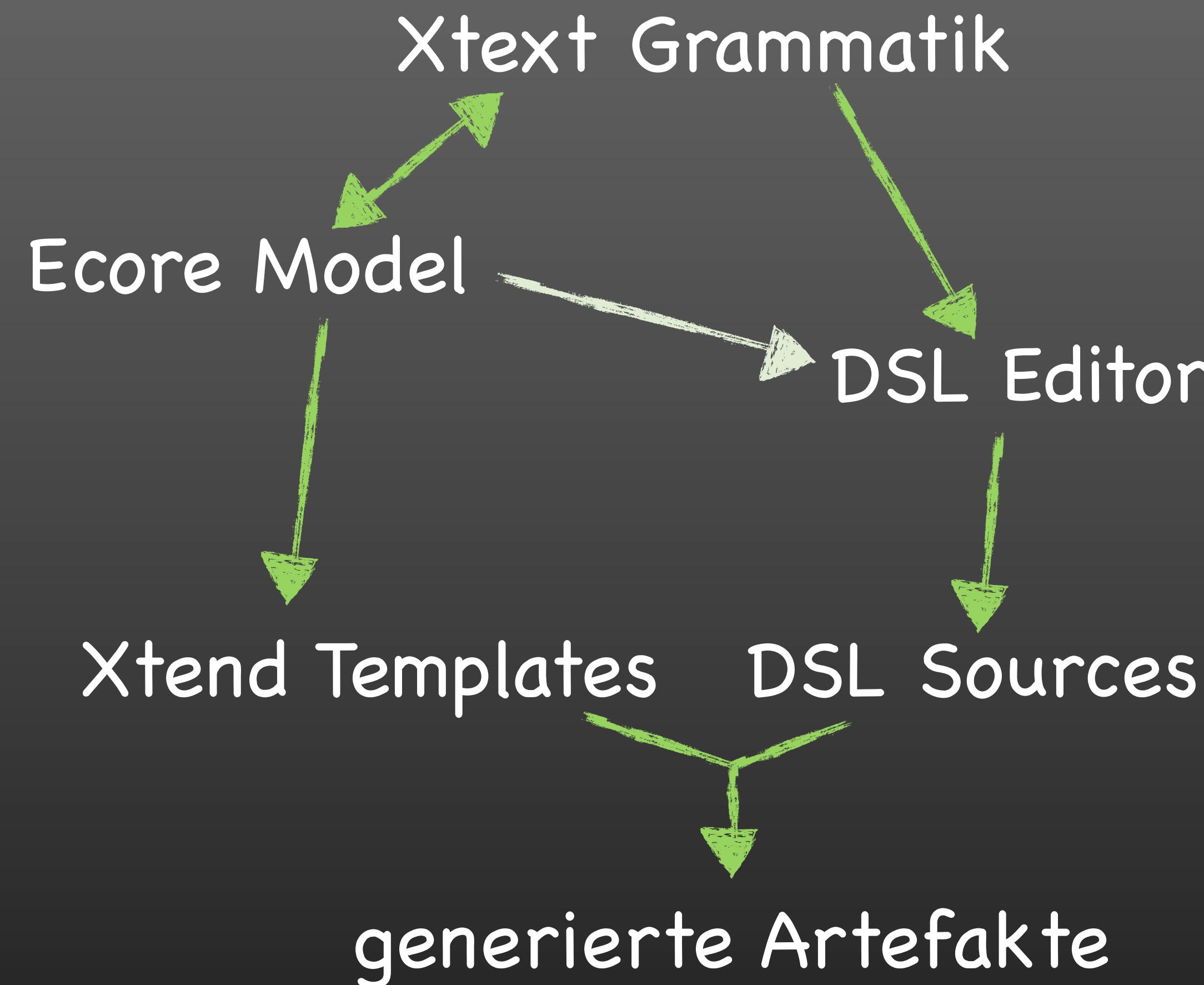
Umgebung & Technologie

- ⦿ Xtext & Xtend
- ⦿ Java basiert, Eclipse Projekte
- ⦿ Definition textueller DSL Grammatiken & Templating Engine
- ⦿ Automatisch generierter Editor für DSL Code
- ⦿ enge Integration mit weiteren Eclipse Projekten

Use Cases

- ⌚ Hello World
- ⌚ Service/Logic/Entity

Xtext & Xtend Komponenten



Xtext Basics

Grammar:

```
'grammar' name=GrammarID
('with' usedGrammars+=[Grammar|GrammarID] (',' usedGrammars+=[Grammar|GrammarID])* )?
(definesHiddenTokens?='hidden' '(' (hiddenTokens+=[AbstractRule] (',' hiddenTokens
    +=[AbstractRule])* )? ')')?
metamodelDeclarations+=AbstractMetamodelDeclaration*
(rules+=AbstractRule)+
;
```

GrammarID returns ecore::EString:

```
ID ('.' ID)*;
```

```
AbstractRule : ParserRule | TerminalRule | EnumRule;
```

```
AbstractMetamodelDeclaration :
GeneratedMetamodel | ReferencedMetamodel;
```

usw. ...

EBNF

Xtext Basics

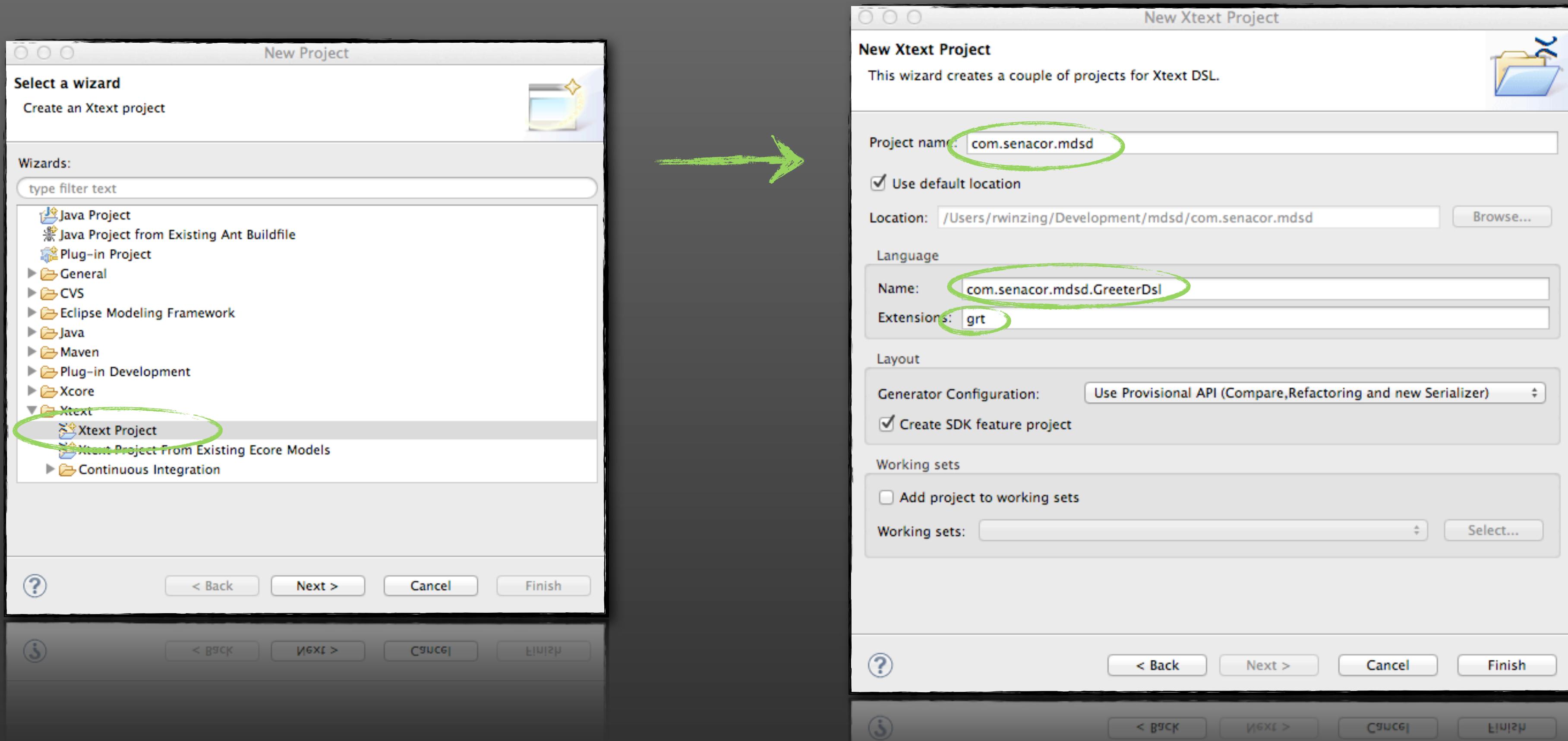
- ⌚ Rules
- ⌚ Terminal
- ⌚ DataType
- ⌚ Parser/Production/EObject
- ⌚ Keywords
- ⌚ Assignments
- ⌚ Referenzen

```
Model:  
greetings+=Greeting*;  
  
Greeting:  
'Hello' name=ID '!';
```

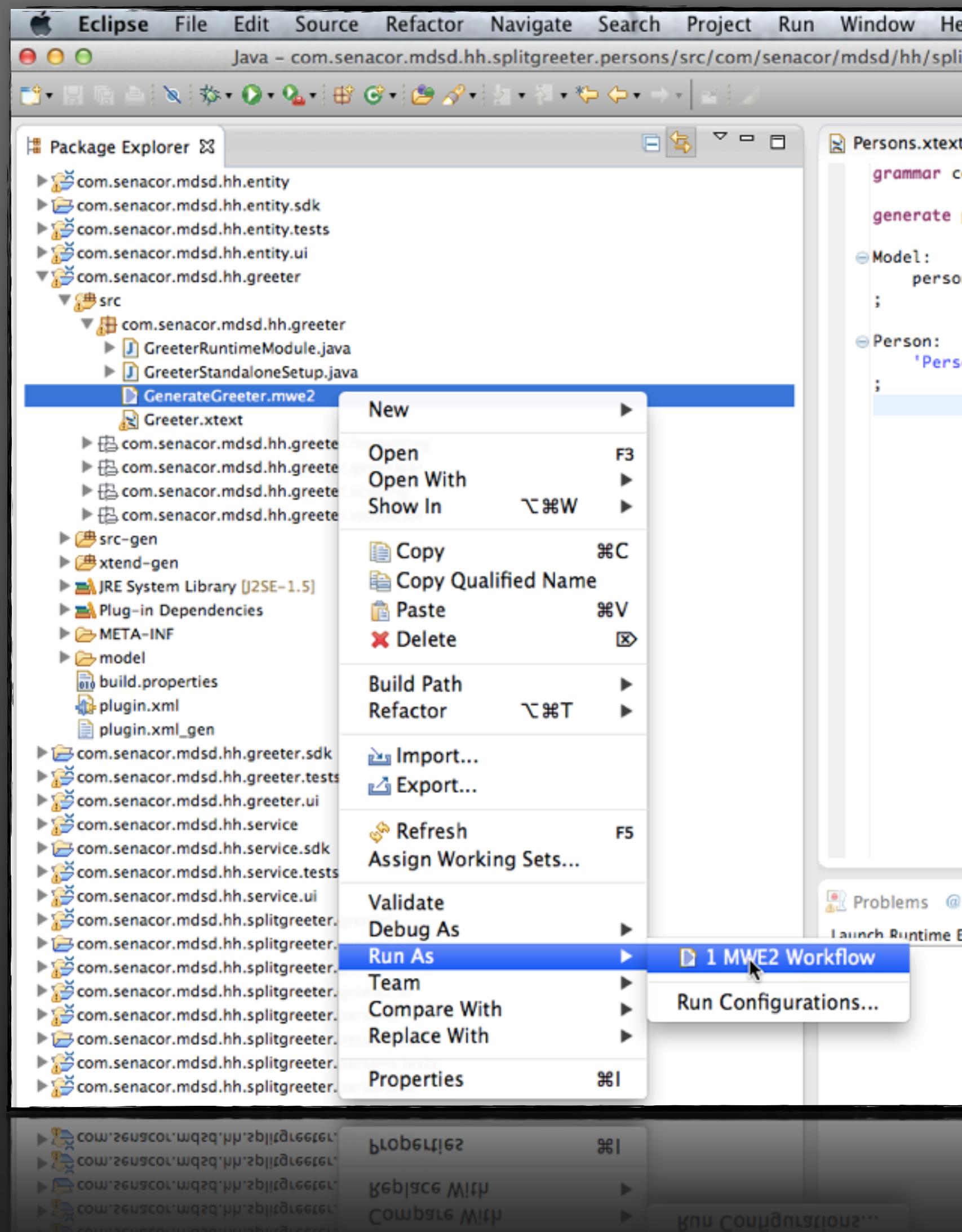
Hello, World! (1)

- ⌚ Projekt erzeugen
- ⌚ Editor testen
- ⌚ generierte Artefakte ansehen

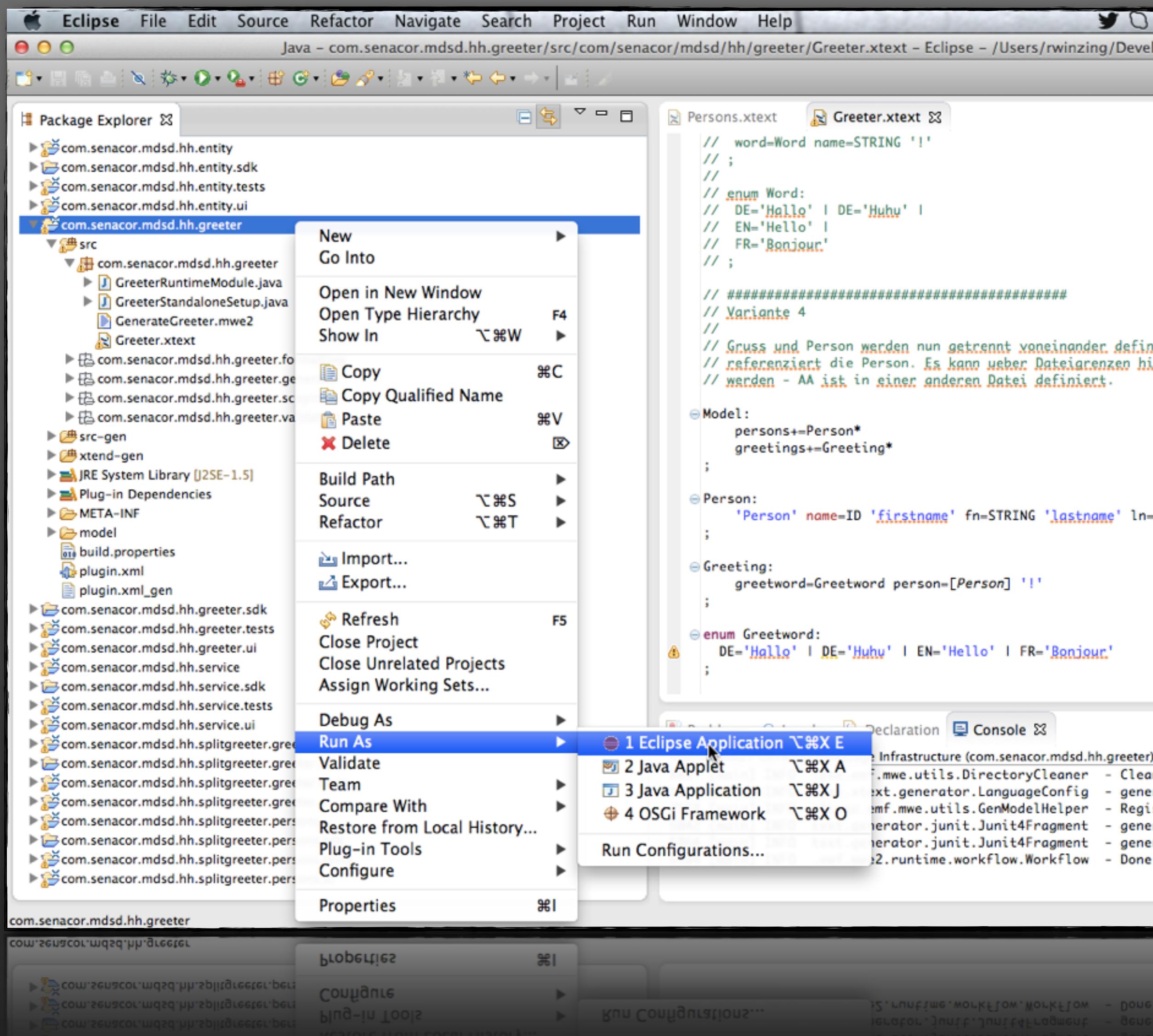
Ein erstes Xtext Projekt ...



... Artefakte generieren ...



... und Editor starten.



Hello, World! (2)

- etwas Struktur durch Klammern
- Name soll als String erfasst werden

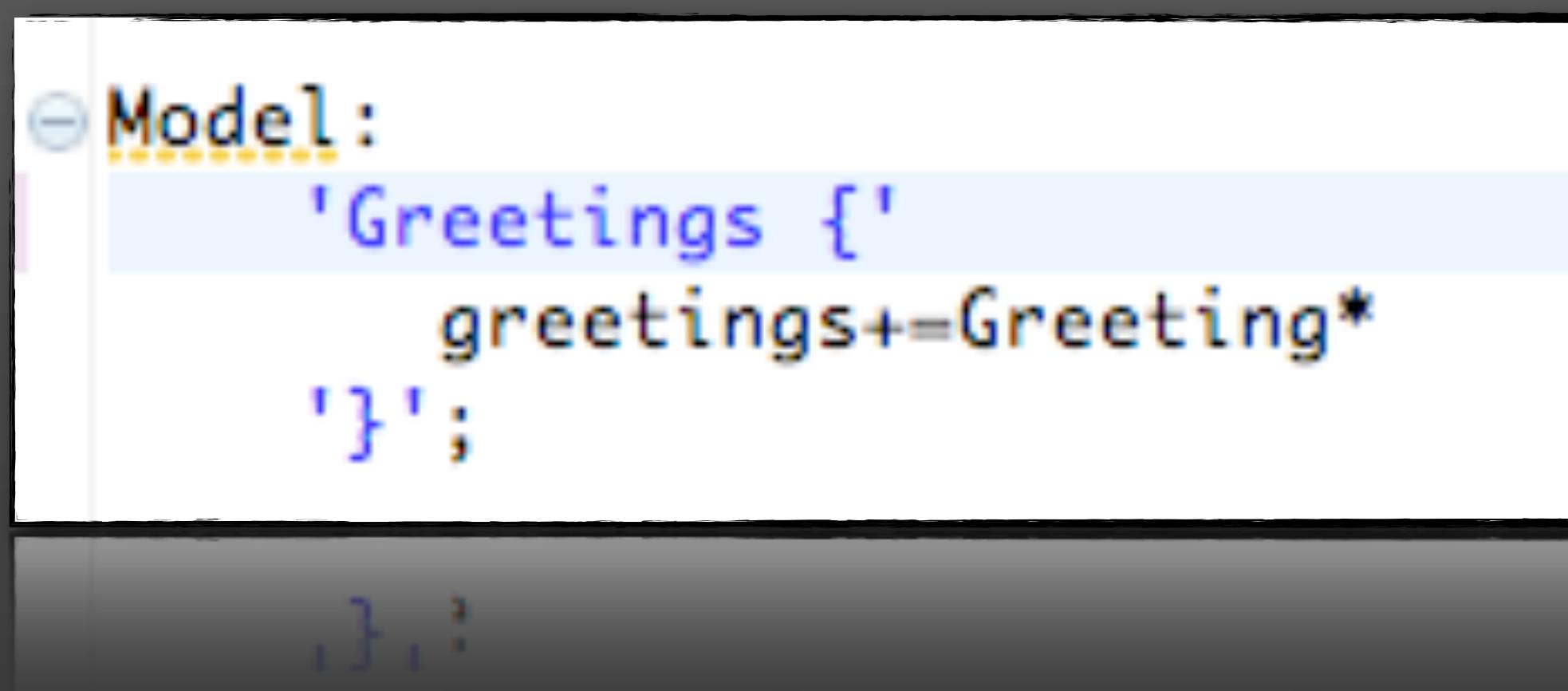


```
test.grt01 test.grt02
Greetings {
Hello World!
}
```

```
test.grt01 test.grt02
Greetings {
Hello "World"!
}
```

Hello, World! (2)

Wie wäre es mit



The screenshot shows a code editor window with a dark theme. A tooltip or callout box is overlaid on the screen, pointing to a section of the code. The code is as follows:

```
Model:
    'Greetings {
        greetings+=Greeting*
    }';
```

The word "Model" is highlighted in yellow. The code uses inconsistent indentation: the opening brace of the block is at the same level as the opening brace of the "Greetings" block, while the closing brace is at a higher level. This is a common mistake in Java where tabs and spaces are often confused.

?

SPACES ARE EVIL!

Hello, World! (3)

- Alternative Grußtexte
- Eine zusätzliche Rule



```
Greetings {
    Hello "World"!
    Hallo "Welt"!
}
```

Hello, World! (4)

- ⌚ Grußworte als Enum
- ⌚ Kommentare



```
Greetings {
    Hello "World"!
    Hallo /* comment */ "Welt"!
}
```

}

Hello, World! (5)

- Personen & Grüße
- Cross-References



The screenshot shows a text editor interface with two tabs at the top: "test.grt01" and "*test.grt02". The content of "test.grt01" is as follows:

```
Persons {
    rw firstname Ralph lastname Winzinger
}

Greetings {
    Hallo rw!
}
```

The word "Hallo" in the "Greetings" block is highlighted with a light blue selection bar underneath it. The status bar at the bottom right of the editor window displays the text "HOTDO_LM1".

Hello, World! (5)

```
grammar com.senacor.mdsd.greeter02.GreeterDsl02 with org.eclipse.xtext.common.Terminals
```

```
generate greeterDsl02 "http://www.senacor.com/mdsd/greeter02/GreeterDsl02"
```

Model:

```
'Persons' '{'  
    persons+=Person*  
'}'
```

```
'Greetings' '{'  
    greetings+=Greeting*  
'}';
```

Person:

```
name=ID 'firstname' firstname=ID 'lastname' lastname=ID gender=( 'w' | 'm' );
```

Greeting:

```
word=Greetword person=[Person] '!' ;
```

enum Greetword:

```
DE='Hallo' | EN='Hello' | FR='Bonjour' ;
```

```
DE='Hallo' | EN='Hello' | FR='Bonjour' ;
```

enum Greetword:

Hello, World! (6)

❶ Mixins - Grammatiken importieren

```
service.xtext ❷ Greeter.xtext ❸ com.senacor.mdsd.hh.splitgreeter.greeter ❹ persons.xtext
❶ Create grammar com.senacor.mdsd.hh.splitgreeter.greeter with org.eclipse.xtext.common.Terminals

❷ import "http://www.senacor.com/mdsd/hh/splitgreeter/persons/Persons" as Persons

❸ generate greeter "http://www.senacor.com/mdsd/hh/splitgreeter/greeter/Greeter"

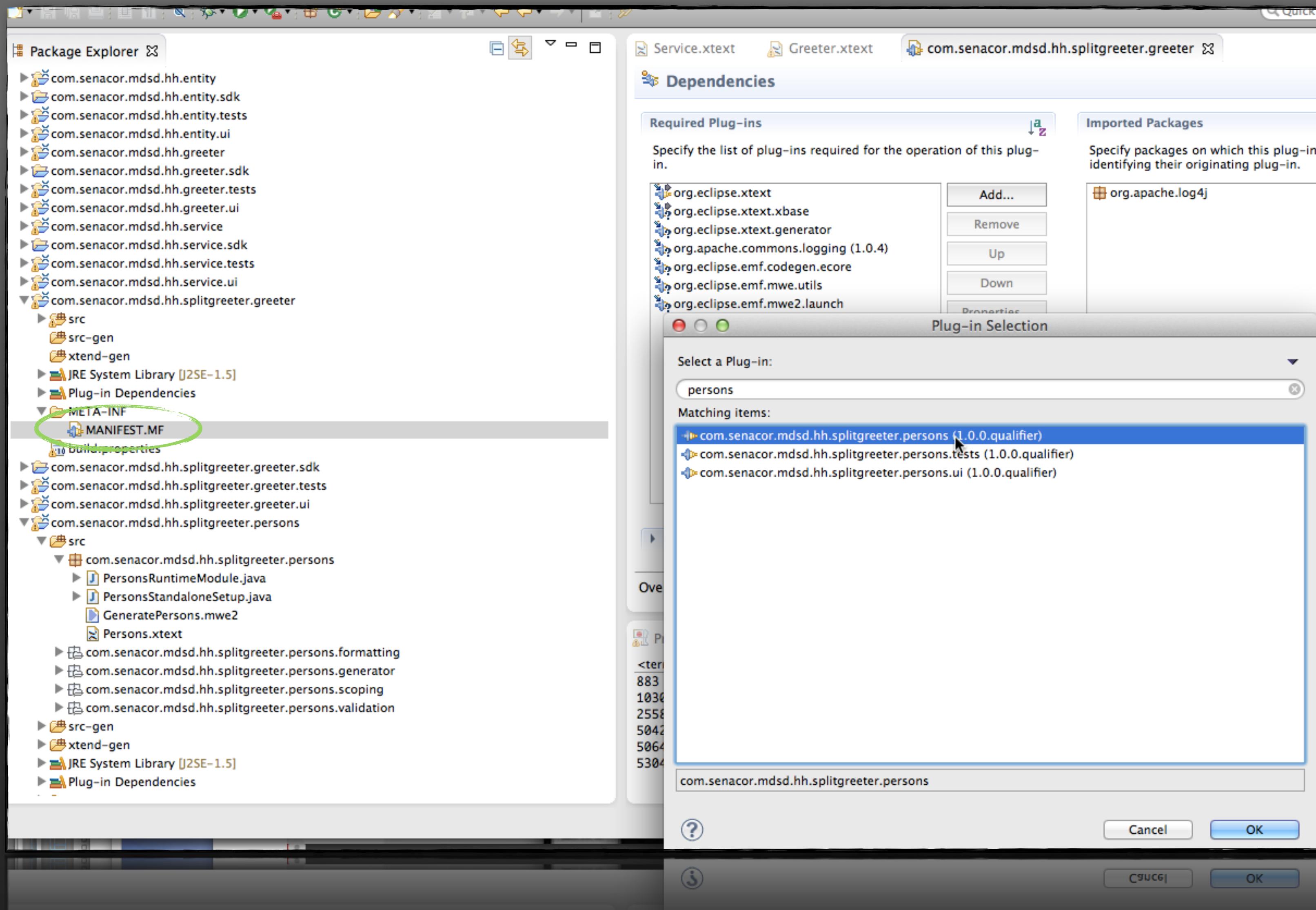
❹ Model:
    greetings+=Greeting*
;

❺ Greeting:
    greetword=Greetword person=[Persons::Person] !*
;

❻ enum Greetword:
    DE='Hallo' | DE='Huhu' | EN='Hello' | FR='Bonjour'
;

❾
;
DE='Hello', | DE='Huhu', | EN='Hello', | FR='Bonjour',
```

Hello, World! (6)



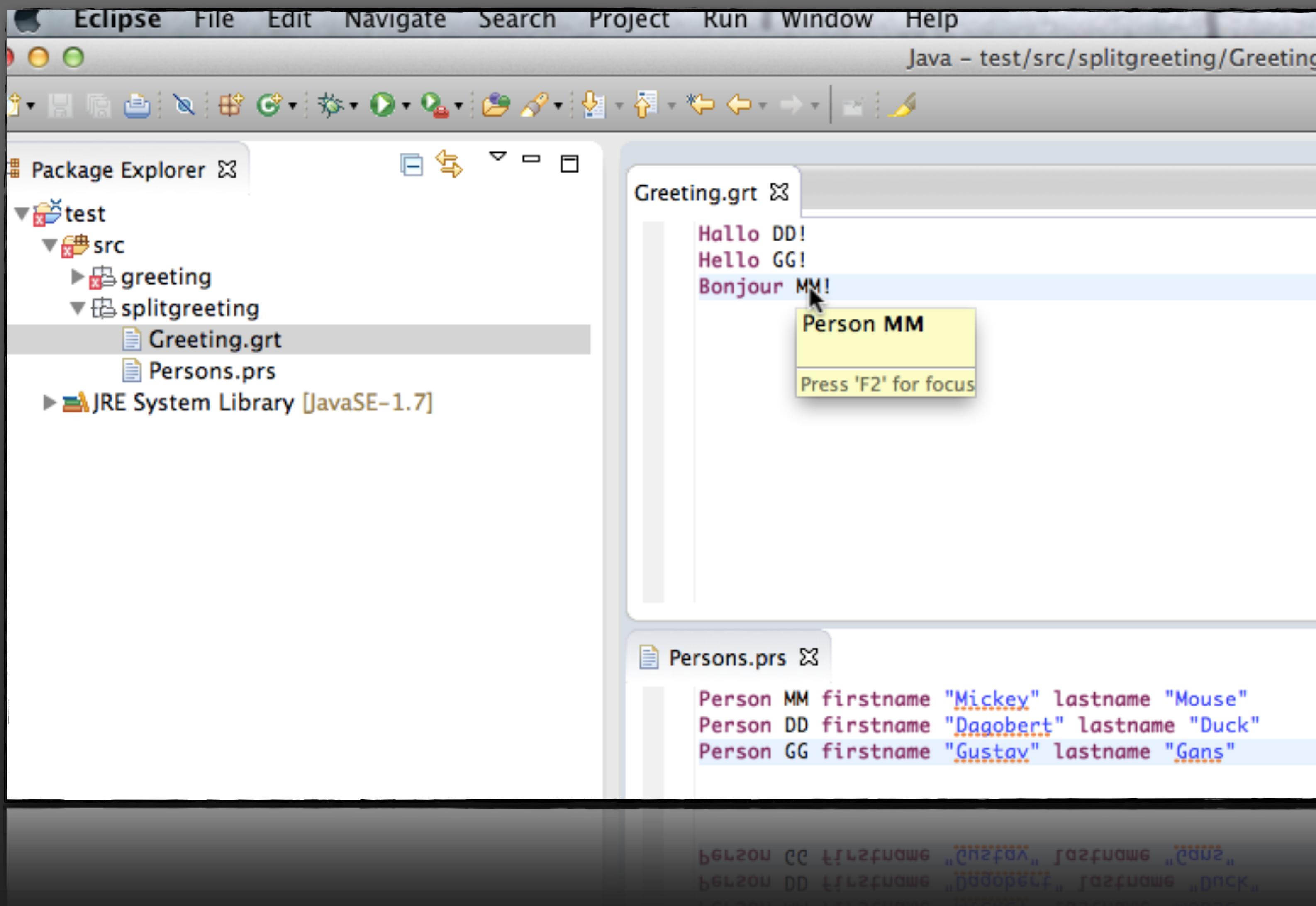
Hello, World! (6)

```
Workflow {
    bean = StandaloneSetup {
        scanClassPath = true
        platformUri = "${runtimeProject}..."
        // The following two lines can be removed, if Xbase is not used.
        // registerGeneratedEPackage = "org.eclipse.xtext.xbase.XbasePackage"
        // registerGenModelFile = "platform:/resource/org.eclipse.xtext.xbase/model/Xbase.genmodel"

        // register splitgreeter persons
        registerGeneratedEPackage = "com.senacor.mdsd.hh.splitgreeter.persons.persons.PersonsPackage"
        registerGenModelFile = "platform:/resource/com.senacor.mdsd.hh.splitgreeter.persons/model/generated/Persons.genmodel"
    }
}
```

java -jar C:\Users\Senacor\Downloads\splitgreeter-0.0.1-SNAPSHOT.jar

Hello, World! (6)



Agenda 15.01.2014

- ⌚ Recap
- ⌚ Übungsblock 2: some more Xtext and Xtend
 - ⌚ Editor: Validierungen, Hilfestellungen, Formatierung, ...
 - ⌚ Xtend: Sprachkonstrukte, (Code-)Generierung
 - ⌚ Build: Integration & Automatisierung
- ⌚ Best Practices
- ⌚ Q&A, Diskussion

Recap

Model:

```
'Greetings' '{'  
    greetings+=Greeting*  
'}';
```

Greeting:

```
('Hello' | 'Hallo') name=ID '!'  
;
```

?:

```
(,HELLO, | ,HALLO,) name=ID , i,
```

Recap

Model:

```
'Greetings' '{'  
    greetings+=Greeting*  
'}';
```

Greeting:

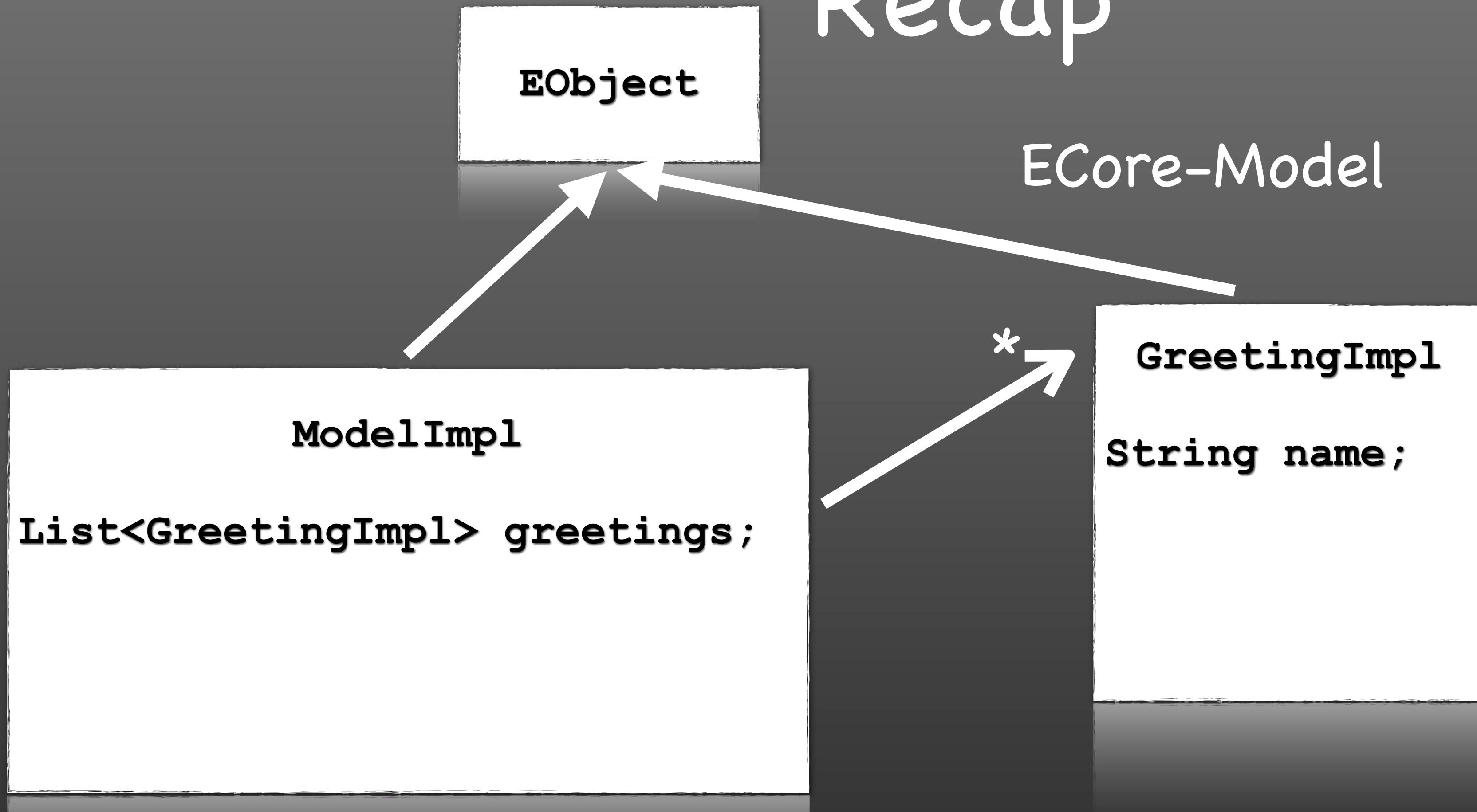
```
('Hello'|'Hallo') name=ID '!'  
;
```

:

```
Greetings {  
    Hello Ralph!  
    Hallo Hamburg!  
}
```

```
public class GreetingImpl extends EClass  
{  
    protected String name = NAME_EDEFAULT;  
  
    protected GreetingImpl() {  
        super();  
    }  
  
    protected EClass eStaticClass() {  
        return GreeterDsl02Package.Literals.GREETING;  
    }  
  
    public String getName() {  
        return name;  
    }  
}  
}  
}  
}  
Default name?
```

Recap



Recap

EObject

ECore-Model

ModelImpl

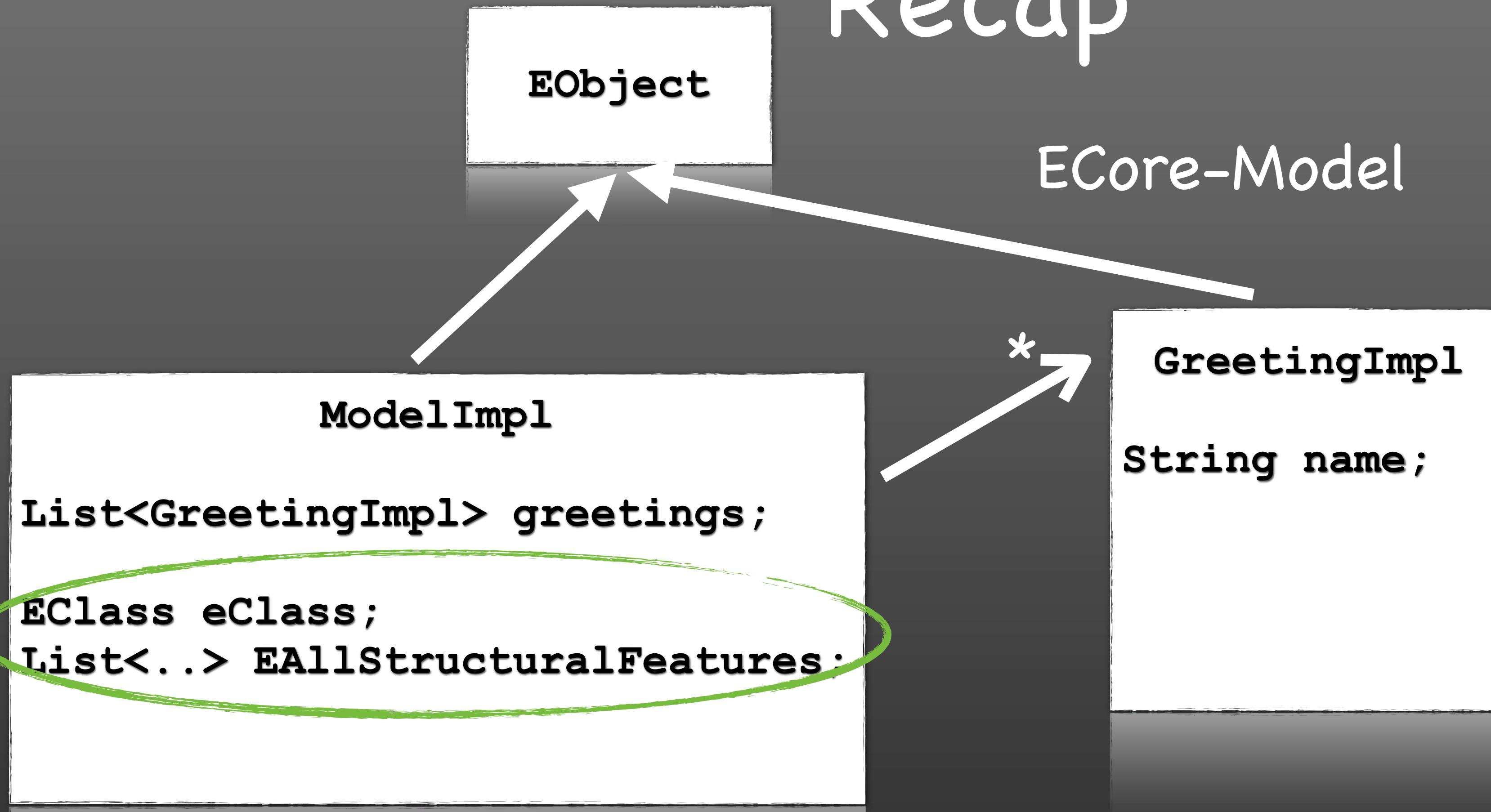
```
List<GreetingImpl> greetings;
```

EClass eClass;

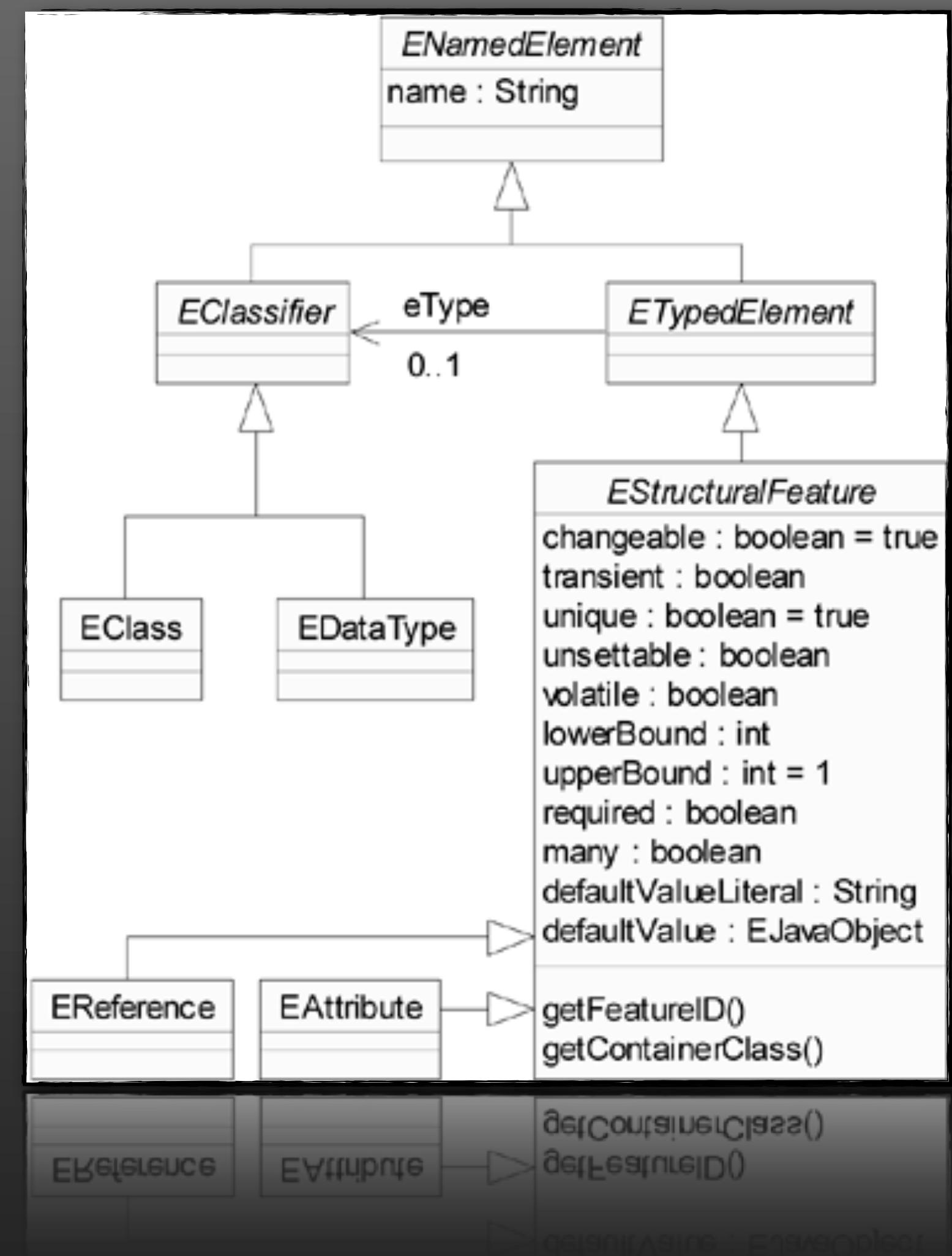
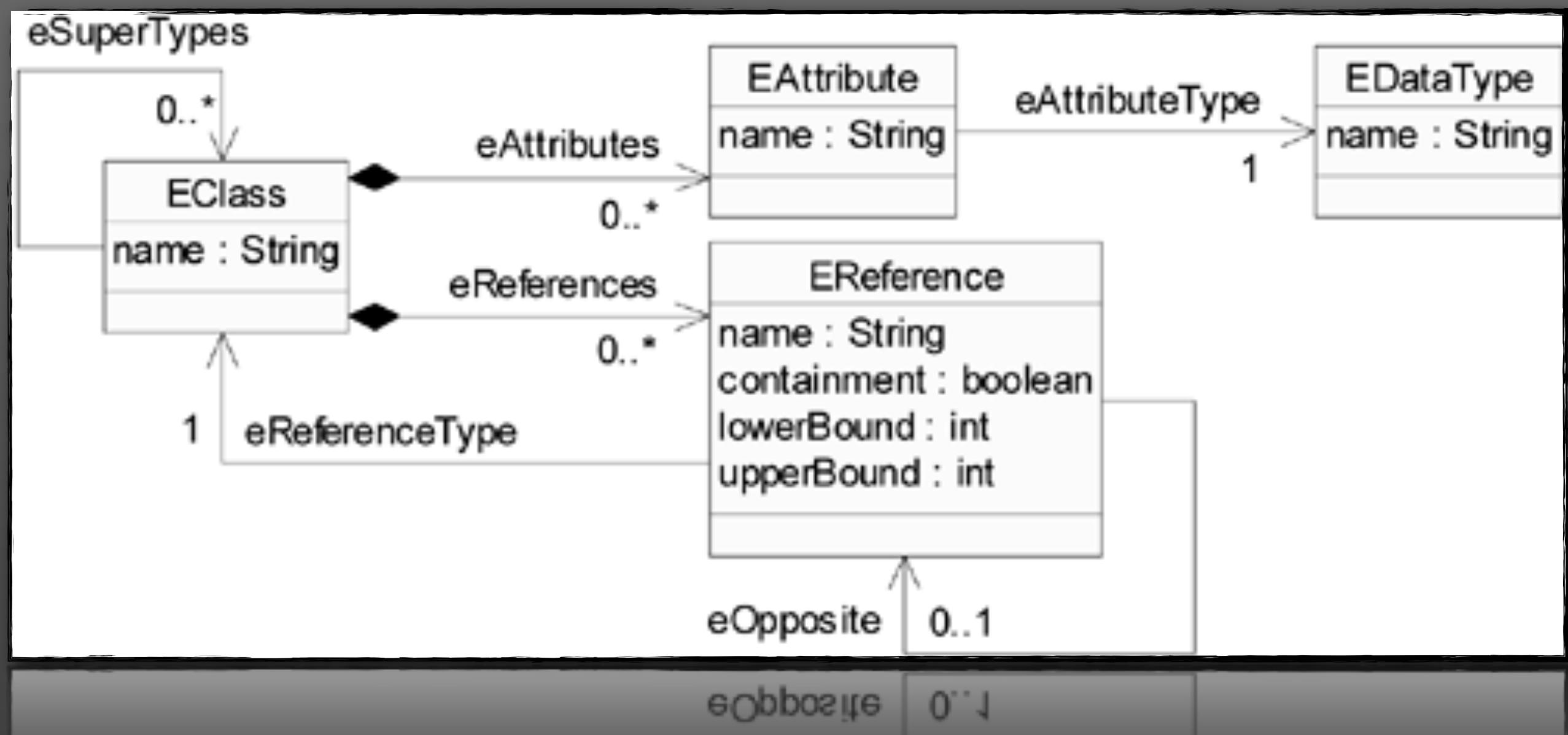
```
List<..> EAllStructuralFeatures;
```

GreetingImpl

String name;



ecore Model



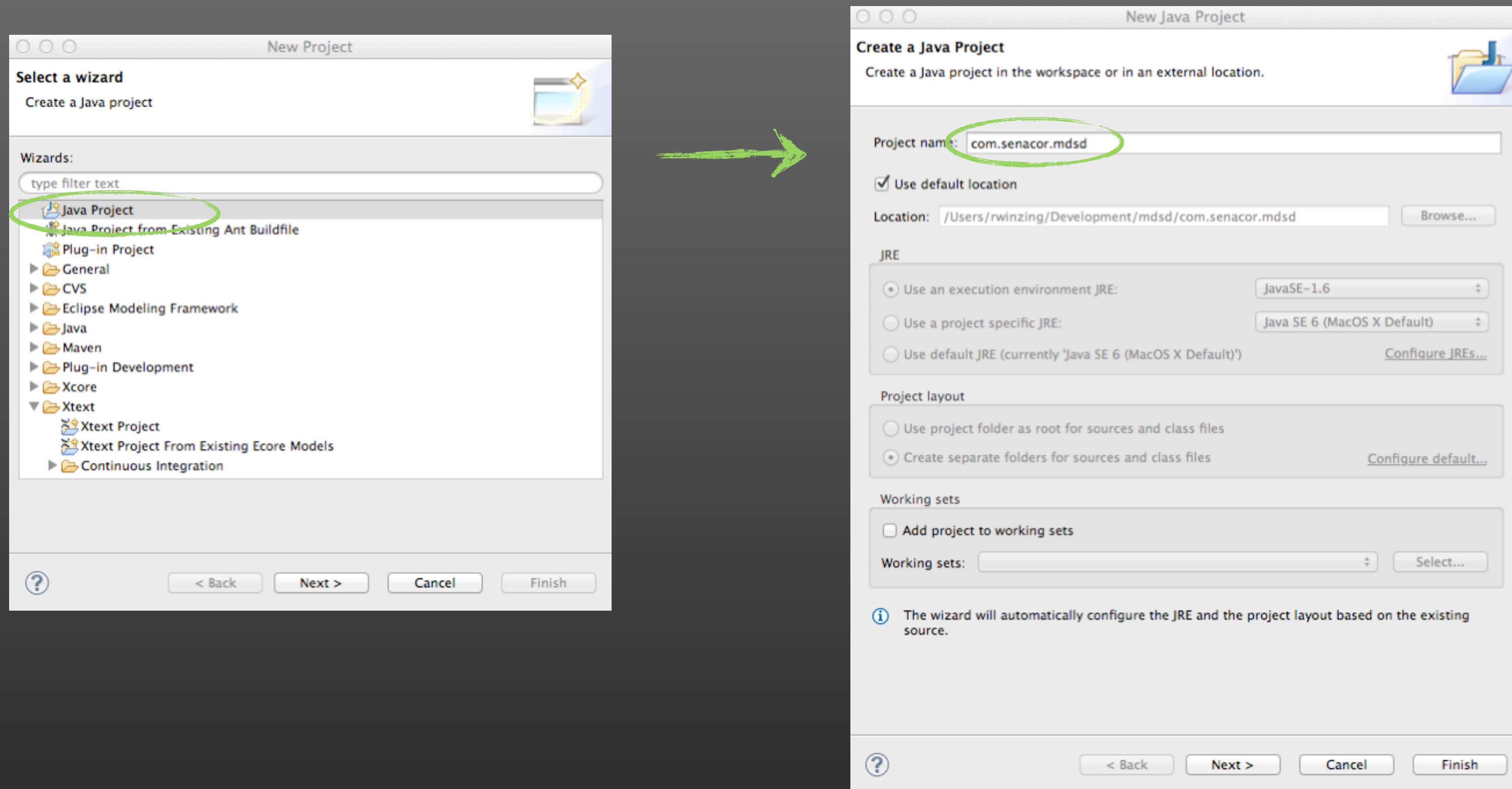
Xtend Basics

- Java Erweiterung (kompiliert wieder zu Java-Quellcode)
- Type extensions
- Type inference
- Keywords def, dispatch, val & var
- Lambdas
- Templates

Hello, World! (wieder mal)

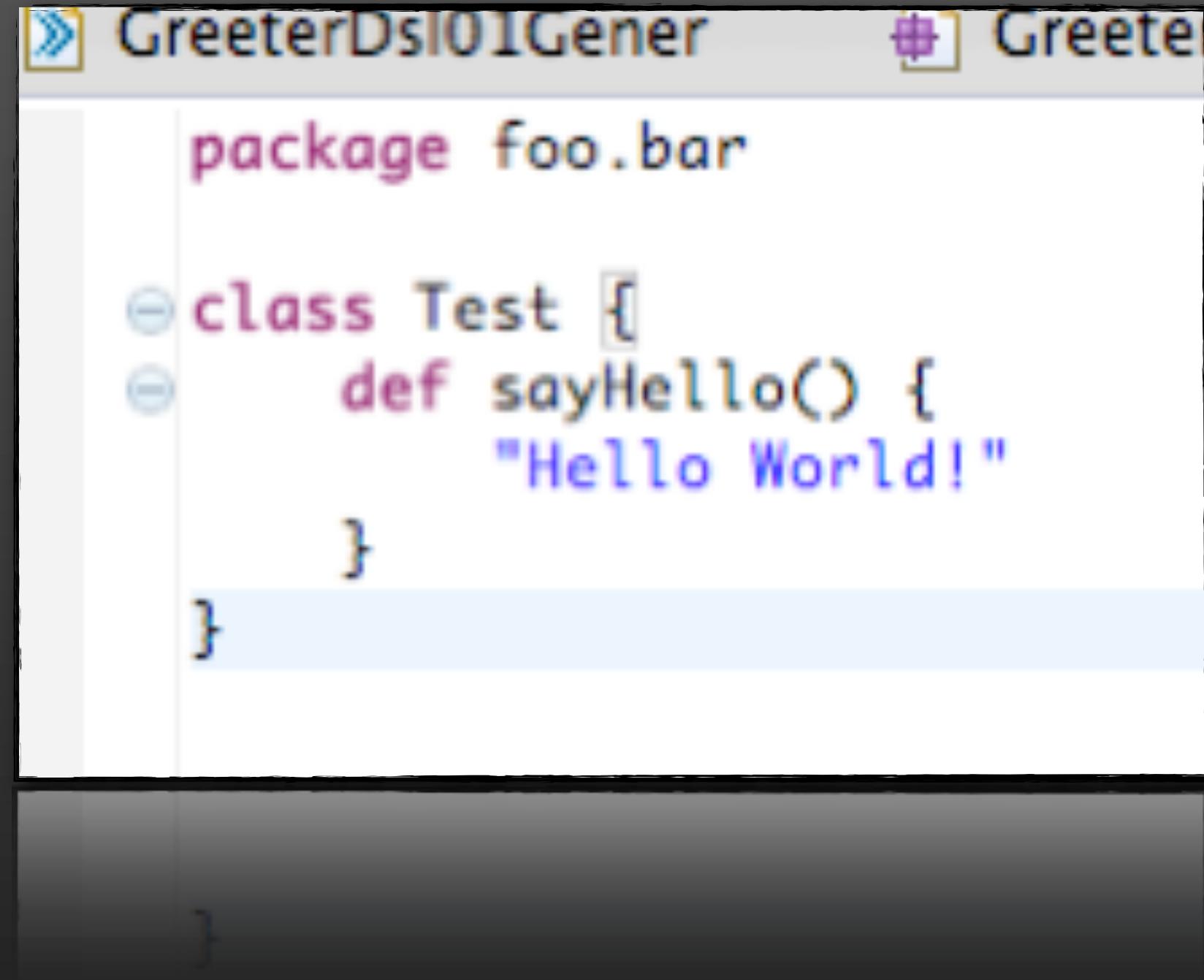
- ❶ Projekt erzeugen
- ❷ Xtend Klasse anlegen & speichern
- ❸ generierte Artefakte ansehen

Ein erstes Xtend Projekt ...



def, var & val

- „def“ deklariert eine Methode
- „var“ und „val“ deklarieren Variablen

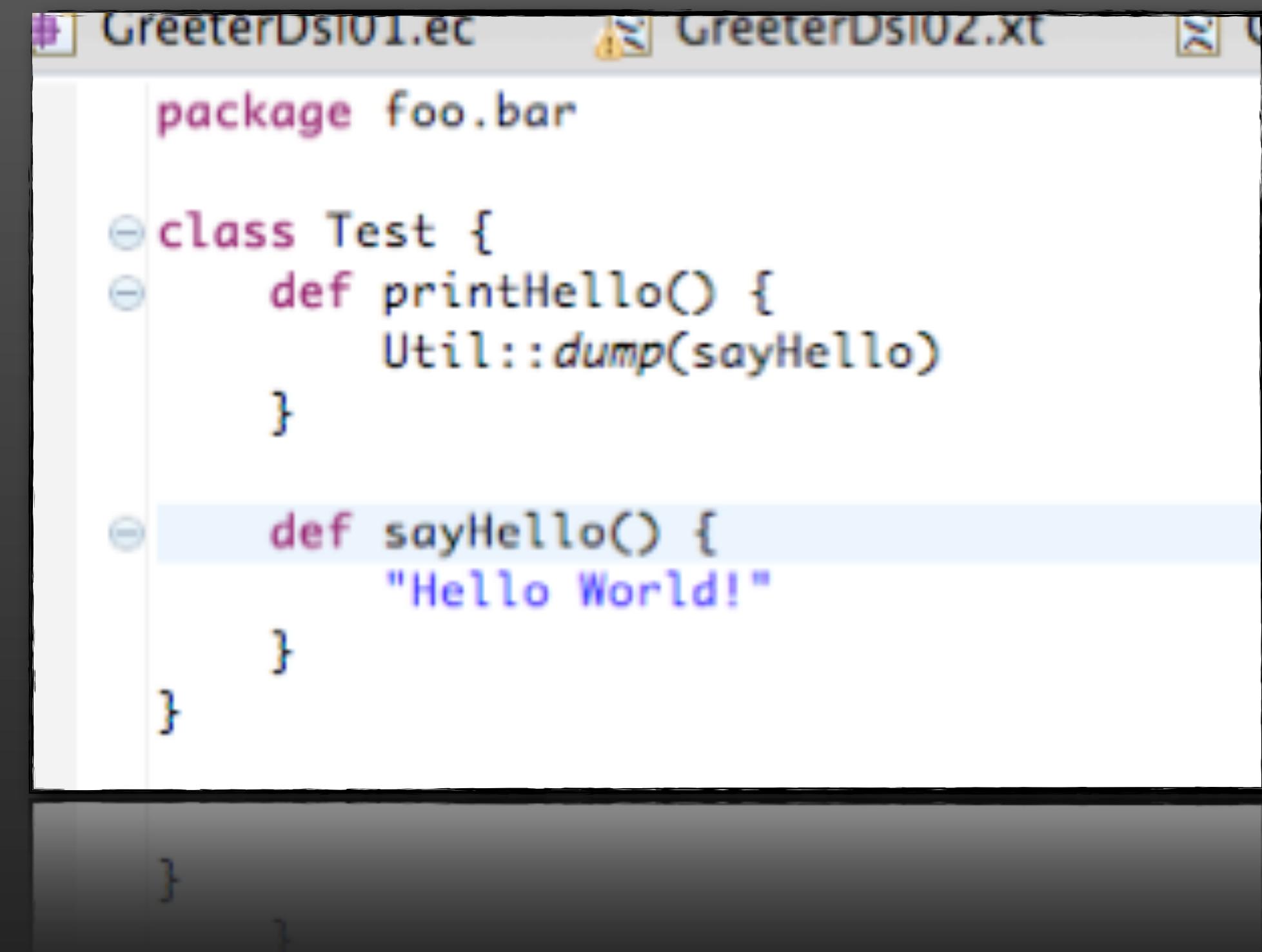


```
GreeterDSL01Generator GreeterDSL01Generator
package foo.bar

class Test {
    def sayHello() =
        "Hello World!"
}
```

Transparenter Zugriff zwischen Java & Xtend

@ „::“ für Zugriff auf statische Elemente



```
# GreeterDSL01.ec      GreeterDSL02.xt

package foo.bar

class Test {
    def printHello() {
        Util:::dump(sayHello)
    }

    def sayHello() {
        "Hello World!"
    }
}
```

Weshalb eigentlich „Xtend“?

foo(a, b) kann man schreiben als a.foo(b)

```
def dumpToConsole(String s) {  
    System.out.println(s);  
}
```

dumpToConsole(„hello, world!“);

```
„hel  
    for (gw: resource.allContents.toIterable.filter(typeof(GW))) {  
        generateFile(gw, fsa);  
    }  
}
```

Templates

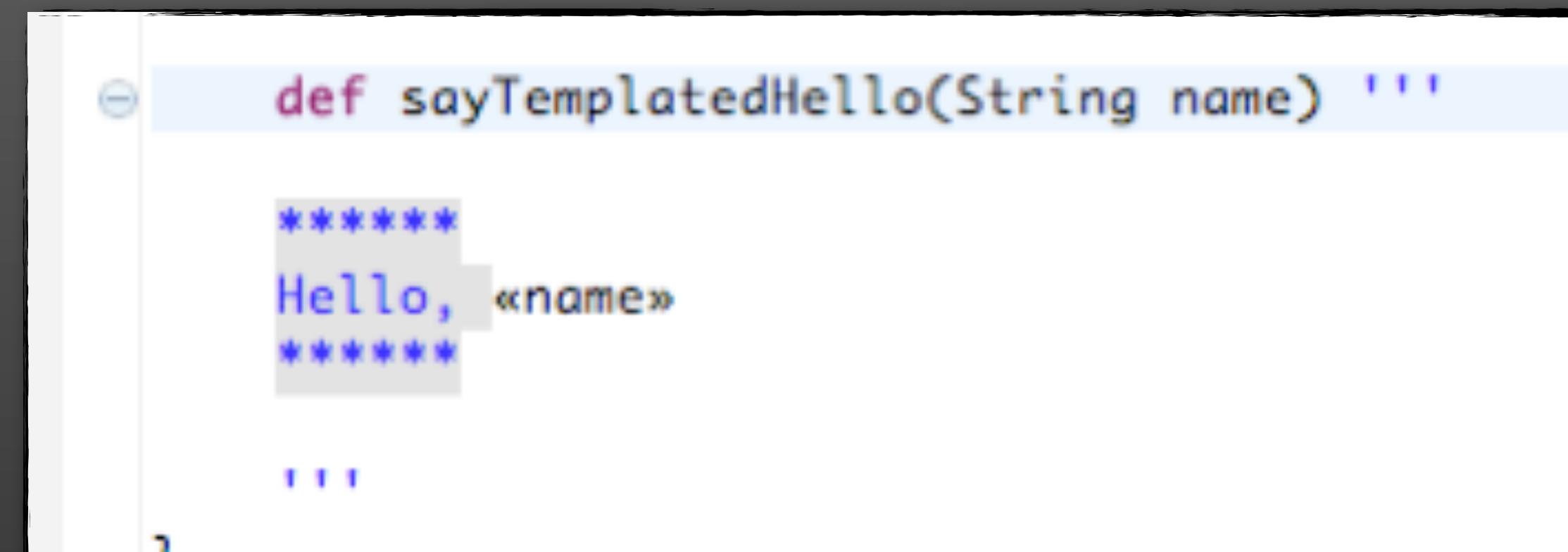
- Triple-Quotes leiten ein Template ein

- String-Ausdruck

- IF / ENDIF

- FOR / ENDFOR

- BEFORE / SEPARATOR / AFTER



A screenshot of a code editor showing a Python template. The code is:

```
def sayTemplatedHello(String name) ***  
    ****  
    Hello, «name»  
    ****  
    ...  
]
```

The string "Hello, «name»" is highlighted in blue, indicating it is a placeholder or variable in the template.

putting stuff together ...

- xtext und xtend verbinden
- und ein wenig generieren

Generation Gap

- Generierter und manuell implementierter Code müssen coexistieren können
- „Protected Regions“ reserviert Bereiche im generierten Code - leider fehleranfällig
- „Generation Gap“ verwendet OO-Mechanismen. Manueller Code wird in Subklassen verlagert, generiert werden nur (abstrakte) Oberklassen

<CODE>