

MODEL DRIVEN SOFTWARE DEVELOPMENT

A PRACTICAL APPROACH

JEPPE CRAMON - TIGERTEAM APS

DANSK IT'S JAVA COMPETENCE NETWORK - 24TH OF FEBRUARY 2009

WHAT IS MDSD?

It's about using models to:

- Simplify reality
 - A simplification of reality using a language with a well defined syntax and semantics
- Raise the level of abstraction
 - Using different models on different levels
- Improve communication
 - Different models for different audiences (Business, Analysis, Design, Implementation)
- Automate the development of software
 - Automate design patterns
 - Executable model transformations

TWO TYPES OF MODELS

Textual
and
Visual

TEXTUAL MODELS

DOMAIN SPECIFIC LANGUAGES (DSL)

External DSL's:

Cascading Style Sheets (CSS):

```
<style type="text/css">
  table {
    border-collapse: separate;
    empty-cells: show;
  }

  li {
    list-style-type: none;
    line-height: 150%;
    list-style-image:
      url(../images/arrowSmall.gif);
  }
</style>
```

Regular Expressions:

```
^4[0-9]{12}(?:[0-9]{3})?$
```

Internal DSL's:

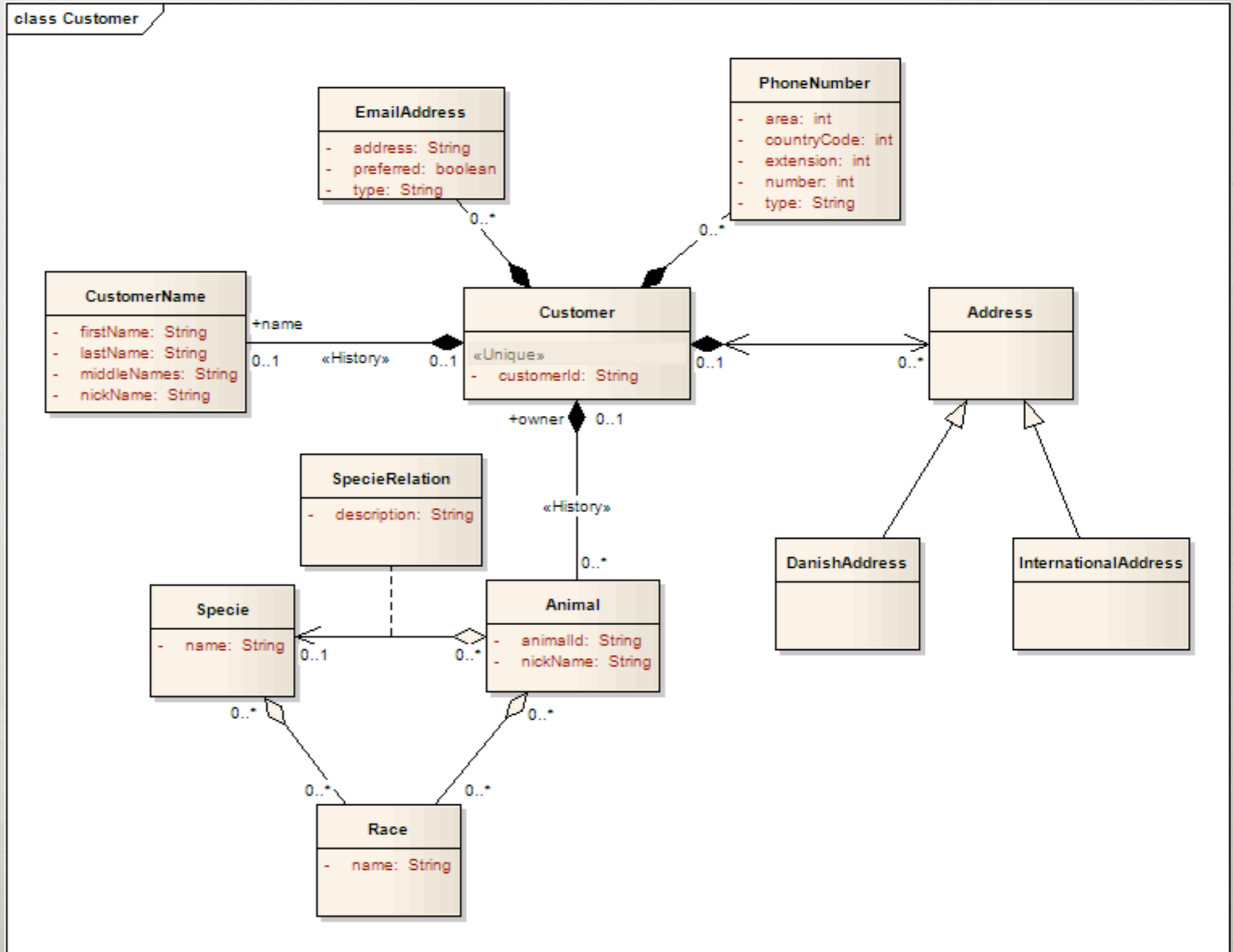
TigerView:

(JSP replacement that is specialized for working with the [Stripes](#) Web framework)

```
stripesForm(CustomerActionBean.class,
  errors(),
  stripesHidden(bind().getCustomer()),
  table(
    tbody(
      tr(
        td("Name:"),
        td(
          stripesTextInput(bind().getCustomer().getName())
        )
      ),
      tr(
        td("Birthday:"),
        td(
          stripesTextInput(bind().getCustomer().getBirthDay())
        )
      )
    )
  ),
  stripesSubmit(
    toActionBean(CustomerActionBean.class).save(),
    value("Gem")
  ),
  stripesSubmit(
    toActionBean(CustomerActionBean.class).cancel(),
    value("Cancel")
  )
);
```

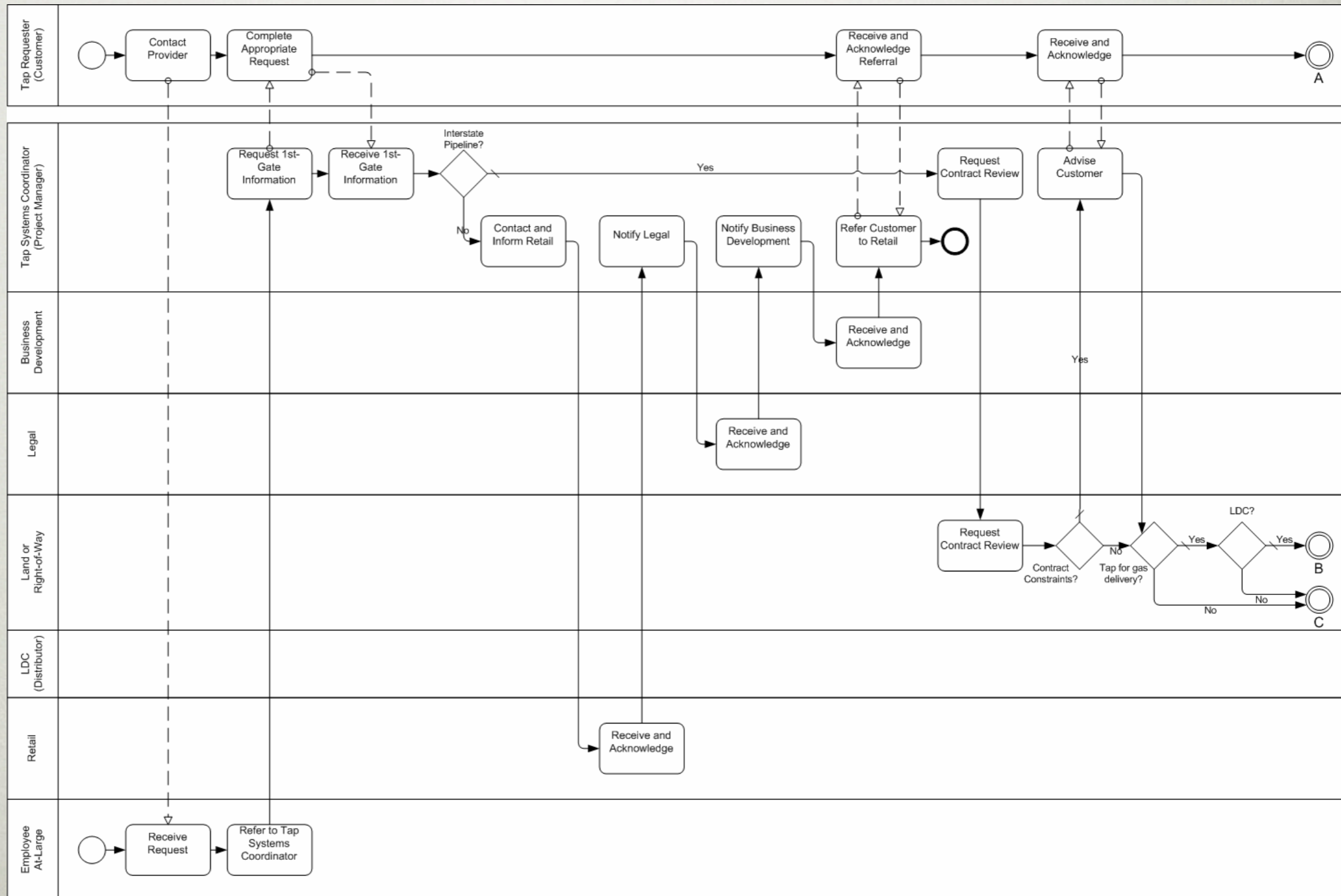
VISUAL MODELS

UML



VISUAL MODELS

BPMN



TEXTUAL VS. VISUAL MODELS

	Textual Model	Visual Model
Pro	<ul style="list-style-type: none">• Very Expressive• Allow configuration by exception• Good for sharing within teams	<ul style="list-style-type: none">• Good for visualizing relationships and flows• Can be made very expressive using proper visual models (rules out UML)• Very good for defining Domain models
Con	<ul style="list-style-type: none">• Poor at “visualizing” relationships and flows	<ul style="list-style-type: none">• Poor at sharing for teams (requires less granularity and more complexity for easier sharing)• UML is poor at creating domain languages (even with UML profiles) because of the limited visual styles and lack of language capabilities (OCL is a poor substitute)

BEHIND MODELS

META MODELS

- A Model / Language is typically defined using a **Meta Model** (aka. a Model of a Model)
- 2 different kinds of Meta Models
 - **Linguistic Meta Model**
 - Defines the **Abstract Syntax**
 - **Ontological Meta Model**
 - Defines the **Semantics**
- A Meta Model is defined using a **Meta-Meta Model** (*e.g. using MOF*)

ABSTRACT SYNTAX

Defines the Concepts of a language
and their relationship

E.g. A Student is an instance of an Entity

CONCRETE SYNTAX

- Defines the physical appearance of a language
- **For Textual languages:**
 - Defines how to form sentences
 - *E.g. Statement, Expression, Identifier, etc.*
- **For Visual/Graphical languages:**
 - Defines the graphical appearance of the language concepts and how they may be combined into a model

SEMANTICS

Describes the meaning of
A sentence in a textual model

or

An element in a visual model

E.g. Student is also an instance of Person

Can be:

- Defined using Meta Models
- Derived from runtime behavior

MODEL TRANSFORMATIONS

The process of transforming
from one Model (adhering to a Meta Model)
to another Model (adhering to another Meta Model)

E.g. from a UML model to a Java model

CODE GENERATION

The process of transforming a Model into source code in some target language (e.g. Java)

MDSD USING TIGERMDSD

TigerMDSD is a lean MDSD Java framework which supports:

- Parsing an UML model into an in-memory UML Model
- Transforming the UML model to one or more Models (e.g. a Java Model)
- Generating code for these Models to a target language using a Template language of choice

MDSD EXAMPLES:

- **SOA Example:**
 - UML Class diagram of a Service Model to WSDL documents and XML Schema
- **Java/JEE Example:**
 - UML Class diagram of a Domain Model to a JPA / Hibernate Entity Model

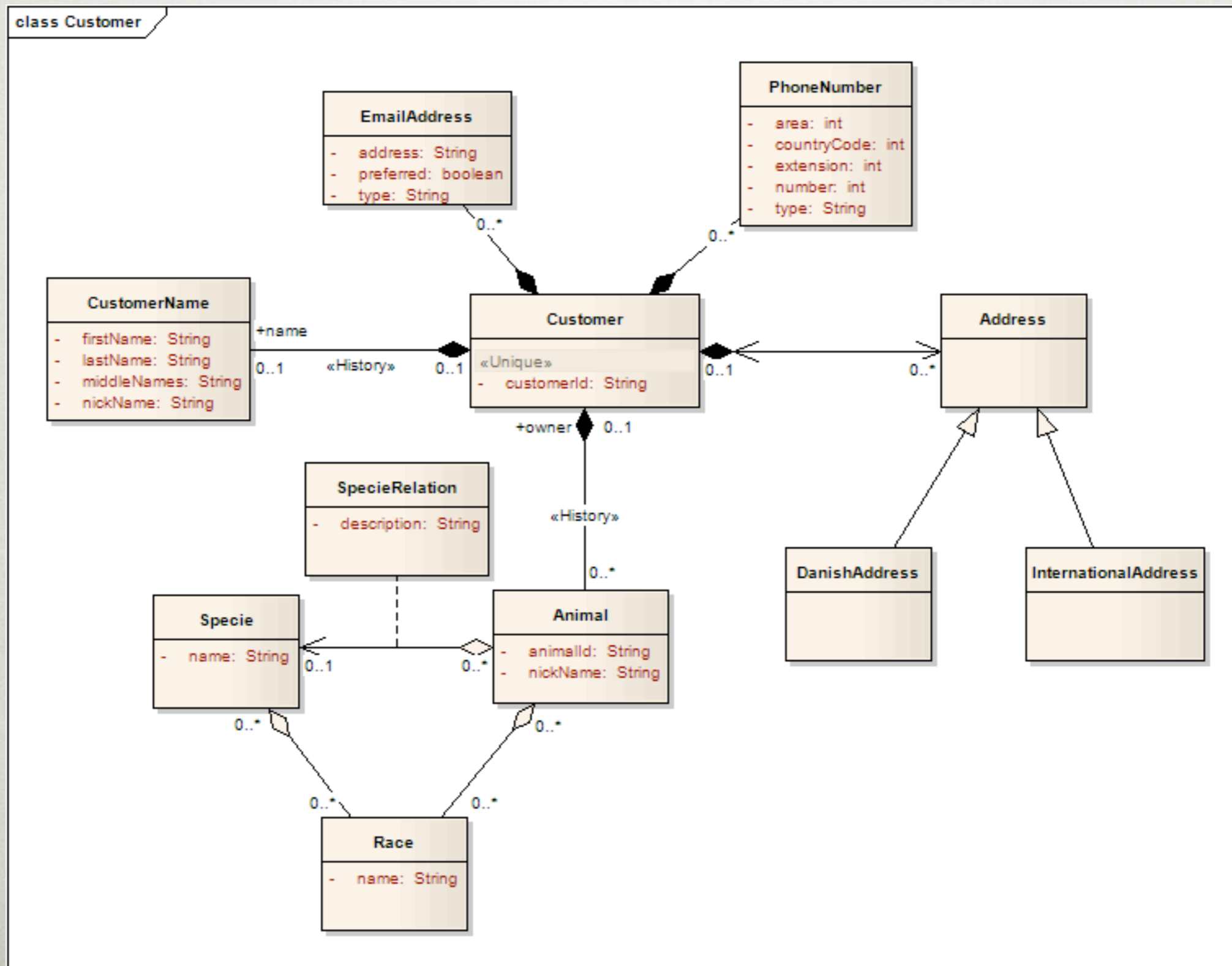
EXAMPLE:

JPA ENTITY MODEL

Process:

1. **Model** the domain entities using a UML Class Diagram
2. **Export** the UML model to XMI
3. **Parse** the XMI model into an in-memory UML model
4. **Transform** the in-memory UML Model to an in-memory Java Model
5. **Generate Java code** from the in-memory Java

MODEL THE ENTITIES USING A UML CLASS



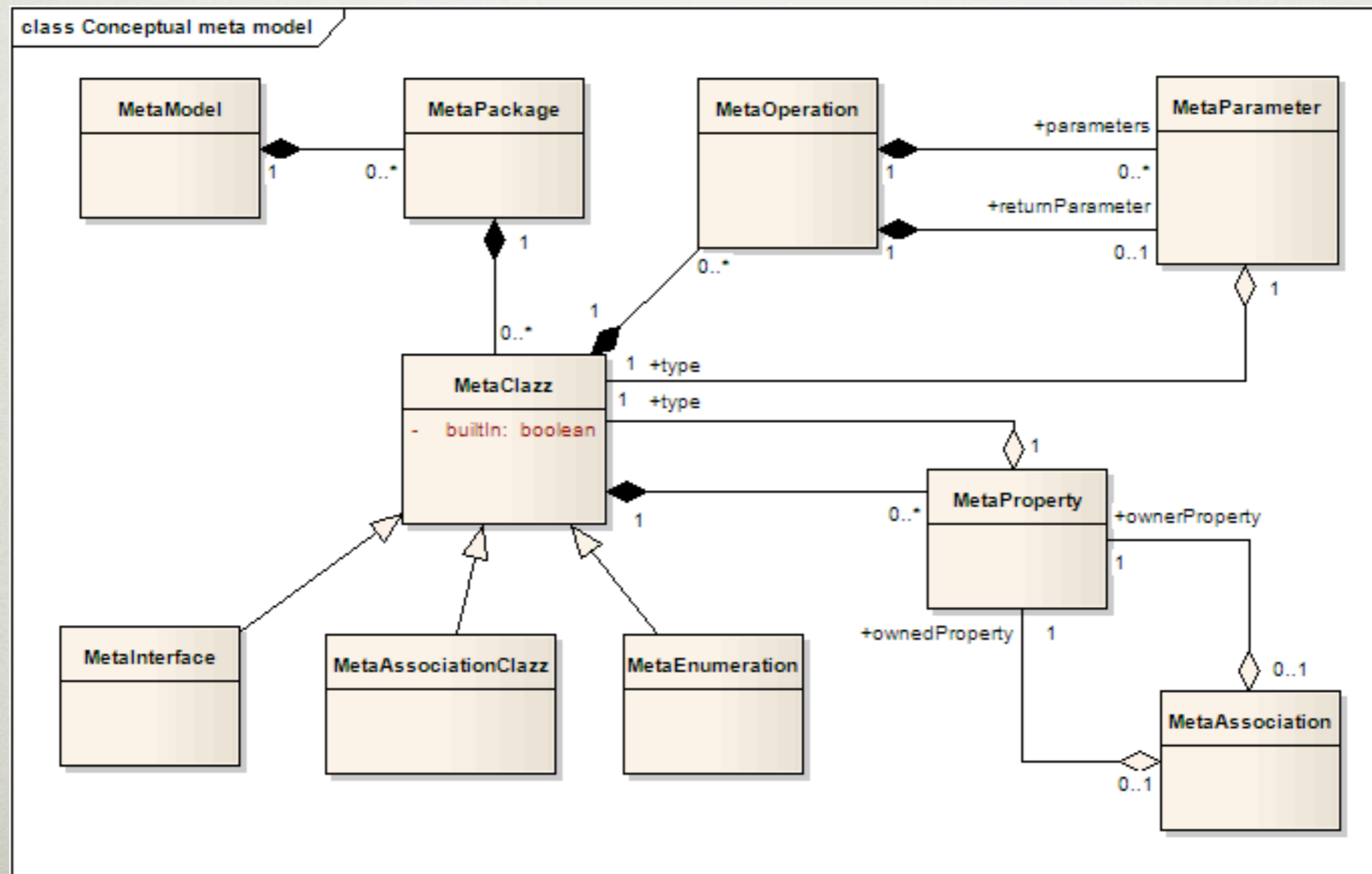
EXPORT THE UML MODEL TO XMI

TigerMDSD currently supports:

- **MagicDraw 15.x and 16.x**
 - In MagicDraw the .MDZIP model file contains the XMI file, so there's no need for exporting here.
- **Enterprise Architect 7.x**
 - In Enterprise Architect you have to Export using the "Export Package" dialog
(Project > Import/Export > Export Package to XMI)

PARSE THE XMI MODEL INTO AN IN-MEMORY UML MODEL

```
XmiReader reader = new EAXmiReader();  
XmiReader reader = new MagicDrawXmiReader();  
MetaModel metaModel = reader.read(this.getClass().getResource("/model.xml"));
```



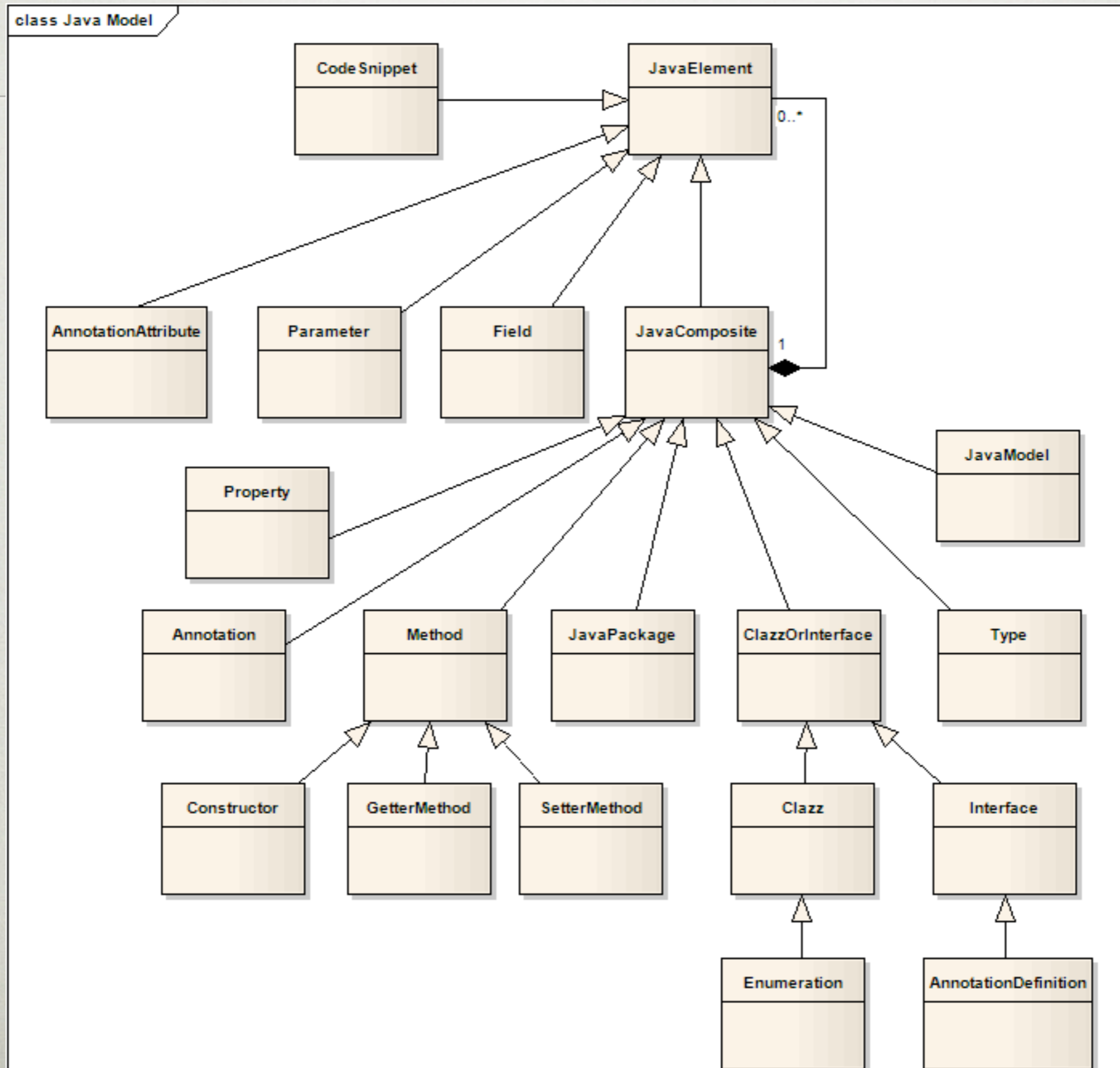
TRANSFORM THE IN-MEMORY UML MODEL TO AN IN-MEMORY JAVA MODEL

In TigerMDSD the specialized `JavaGenerator` performs the UML Model to Java Model transformation according to these rules:

Meta Type	Resulting Java Model
MetaPackage	JavaPackage
MetaClazz	Clazz
MetaAssociationClazz	Clazz
MetaEnumeration	Enumeration
MetaInterface	Interface
MetaProperty	Property (Composed of a Field, GetterMethod and SetterMethod)
MetaOperation	Method

```
JavaGenerator javaGenerator = new JavaGenerator();  
List<ClazzOrInterface> allGeneratedClazzes = javaGenerator.execute(metaModel, codeWriter);
```

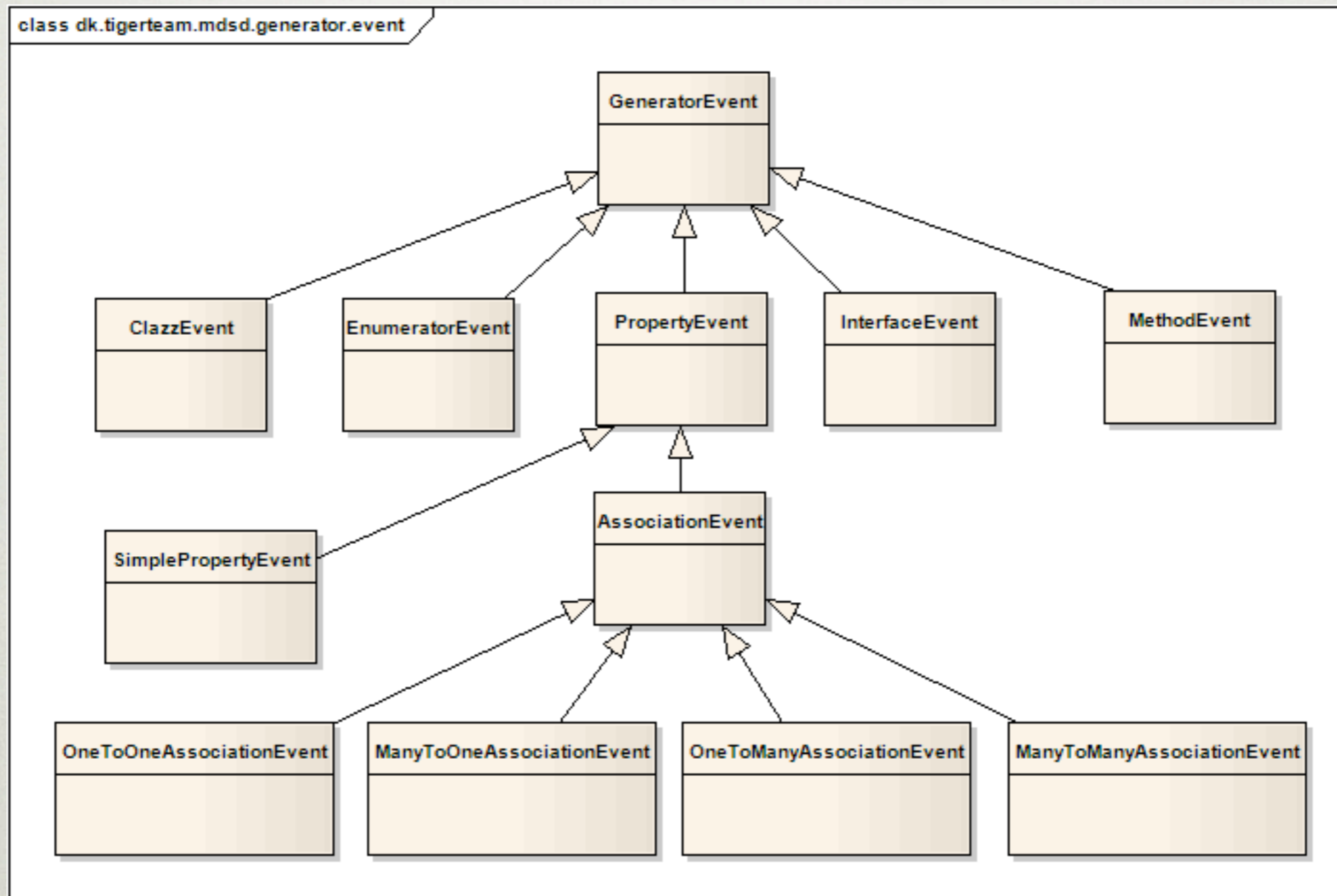
TRANSFORM TO A JAVA MODEL



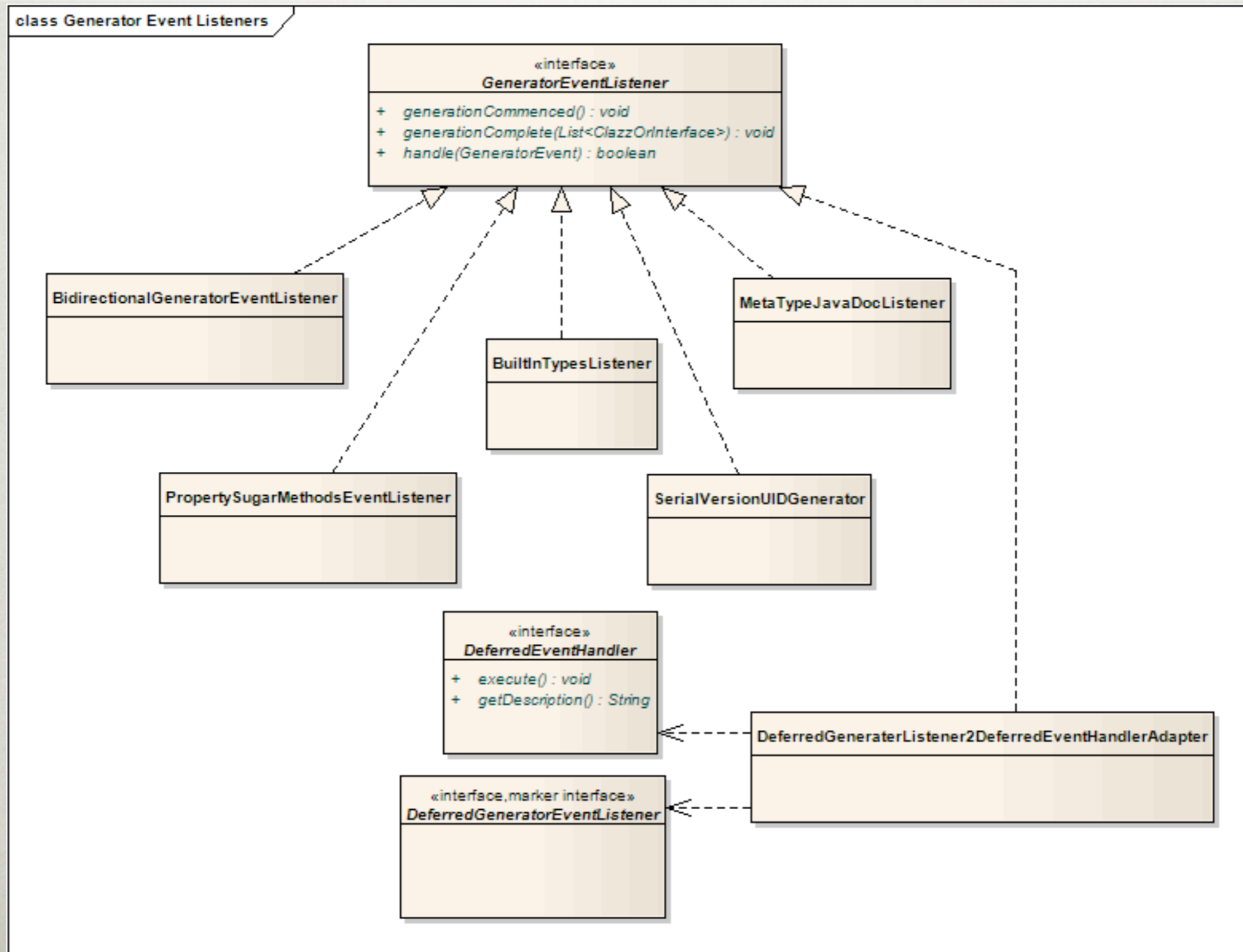
TRANSFORMATION EXTENSION

The JavaGenerator employs an event driven generator pattern, which through `GeneratorEventListeners` allows easy specialization of the core Transformation.

GENERATOR EVENTS



GENERATOR EVENT LISTENERS



JPA ANNOTATION SPECIFICATIONS LIST AND EXTENS

If an association has stereotype <<NotNull>> then this Listener applies the Hibernate Validator @NotNull attribute

Handles everything related to @Column, @JoinColumn, @JoinTable,

Makes all OneToMany and ManyToMany associations ordered by

Adds a (or reuses an existing) Hibernate @Table annotation with a "comment" attribute that contains in

Helper Listener which will allow OneToOne, ManyToOne, OneToMany and ManyToMany property Getter

Allows customization of Hibernate Batch-Size og SubSelect Fetch optimizations through Tagged Values on Properties

(Note: F named f

General purpose JPA field based annotation handler

- Supports:
- All common JPA mappings: OneToOne, OneToMany, ManyToOne, ManyToMany
 - Mapped super classes
 - Varying Inheritance Strategy (used Inheritance strategy into consideration)
 - Cardinality
 - Eager (lazy (eager/lazy))
 - C

Generates a standard JPA file with all Entity classes

Generates a Persistence.xml Pro

Generates a complete and automated **Integration test**, which test the persistence of each Entity while adhering to cardinality constraints (required and optional properties), which are read at runtime through the **RuntimeMetaData** provided through the **RuntimeMetaDataGenerator** (not shown)

override for Hibernate)

OrmXmlGenerator

Pers

GENERATOR EVENT LISTENER EXAMPLE

```
public class HibernateDeleteOrphanListener extends BaseJpaGeneratorEventListener {
    @Override
    protected boolean handleOneToManyOwnerOfAssociation(OneToManyAssociationEvent event) {
        if (isDeleteOrphanCandidate(event)) {
            event.getProperty().getField().addAnnotations(new Annotation(Cascade.class).addAnnotationAttribute("value", CascadeType.DELETE_ORPHAN));
            // Remove the Set collection method since it allows for situations where Hibernate fails, if you overwrite its internal
            // collection wrapper
            event.getProperty().setSetterMethod(null);
        }
        return true;
    }

    protected boolean isDeleteOrphanCandidate(OneToManyAssociationEvent event) {
        if (event.getMetaProperty().isOwnerOfAssociation() && !event.getMetaProperty().getAssociation().isBidirectional()
            && !event.getMetaProperty().getAssociation().isSelfReferencing()) {
            // Check the clazz of the opposite property to see what kind of associations it has
            for (MetaProperty subMetaProperty : event.getMetaProperty().getType().getProperties()) {
                if (subMetaProperty.isPartInAnAssociation()) {
                    if (subMetaProperty.isOwnerOfAssociation()) {
                        // One-To-One are always good for owning associations, so is Many-to-One, but if we meet a ManyToMany or a OneToMany we ignore them
                        // as a candidate for delete orphan
                        if (subMetaProperty.getAssociationType() == AssociationType.ManyToMany
                            || subMetaProperty.getAssociationType() == AssociationType.OneToMany) {
                            return false;
                        }
                    } else if (subMetaProperty.getAssociation().isBidirectional()){
                        // The type of the our sub property is not an owning association and we have
                        // a java association in both directions (bidirectional), which hibernate doesn't handle
                        return false;
                    }
                }
            }
        }
        return true;
    }
    return false;
}
}
```

JAVAGENERATOR EXAMPLE

CONFIGURED USING JAVA

```
JavaGenerator javaGenerator = new JavaGenerator();
javaGenerator.addEventListeners(new BuiltInTypesListener() {
    @Override
    protected void resolveBuiltInTypes(Type type) {
        if (type.getName().equalsIgnoreCase("DateTime")) {
            type.setWrappedJavaClass(DateTime.class);
        } else {
            super.resolveBuiltInTypes(type);
        }
    }
});
javaGenerator.addEventListeners(new JPAGeneratorEventListener()
    .setShouldMakeBaseClassesMappedSuperClassesIfPossible(true)
    .setShouldGeneratePresentFieldInEmbeddables(false)
    .setDefaultToLazyFetchingForAllAssociations(true)
);
javaGenerator.addEventListeners(new JPANamedTablesAndColumnsListener());
javaGenerator.addEventListeners(new BidirectionalGeneratorEventListener());
javaGenerator.addEventListeners(new HibernateAssociationUnproxyListener());
javaGenerator.addEventListeners(new SerialVersionUIDGeneratorListener());
javaGenerator.addEventListeners(new PropertySugarMethodsEventListener());
javaGenerator.addEventListeners(new MetaTypeJavaDocListener());
javaGenerator.addEventListeners(new HibernateIndexingListener());
javaGenerator.addEventListeners(new HibernateDeleteOrphanListener());
javaGenerator.addEventListeners(new HibernateValidatorNotNullListener());
javaGenerator.setCreateExtensionClasses(true);
```

JAVAGENERATOR EXAMPLE CONFIGURED USING YAML

Example of a Simple Java POJO (Plain Old Java Object) generation from a MagicDraw diagram:

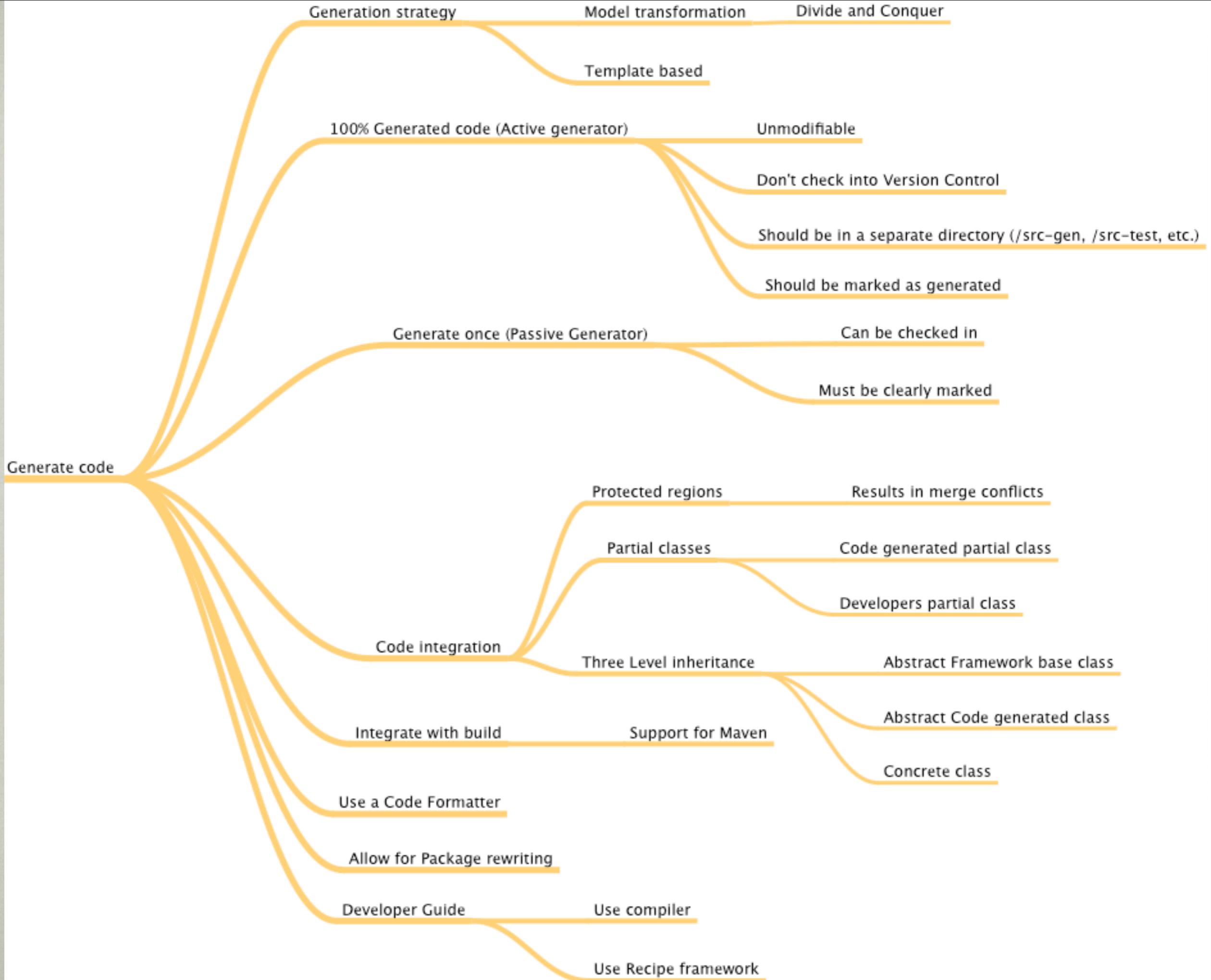
```
# Setup
xmiModelPath: model/DomainModel.mdzip
umlTool: MagicDraw
basePackageName: dk.tigerteam.portal.model
generateExtensionClasses: false
sourceCodeFormatting:
  implementation: dk.tigerteam.mdsd.generator.JalopyFormattingCodeWriter
  excludeFormattingForPackages:
    - dk.tigerteam.portal.model.meta

# Paths
generateBaseClassesToPath: src/main/generated
generateInterfacesToPath: src/main/generated

mapUmlPropertyTypes:
  - name: DateTime
    javaType: org.joda.time.DateTime

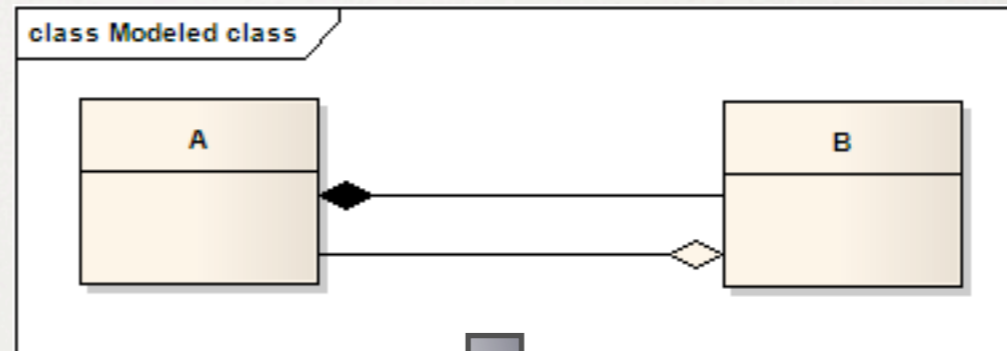
# Extensions
eventListeners:
  - dk.tigerteam.mdsd.generator.bidirectional.BidirectionalGeneratorEventListener
  - dk.tigerteam.mdsd.generator.extension.SerialVersionUIDGeneratorListener
  - dk.tigerteam.mdsd.generator.extension.PropertySugarMethodsEventListener
  - dk.tigerteam.mdsd.generator.extension.MetaTypeJavaDocListener
  - dk.tigerteam.mdsd.generator.extension.SerializablePojoListener

# Create Runtime Meta Model
createRuntimeMetaModel:
  packageName: dk.tigerteam.portal.model.meta
```

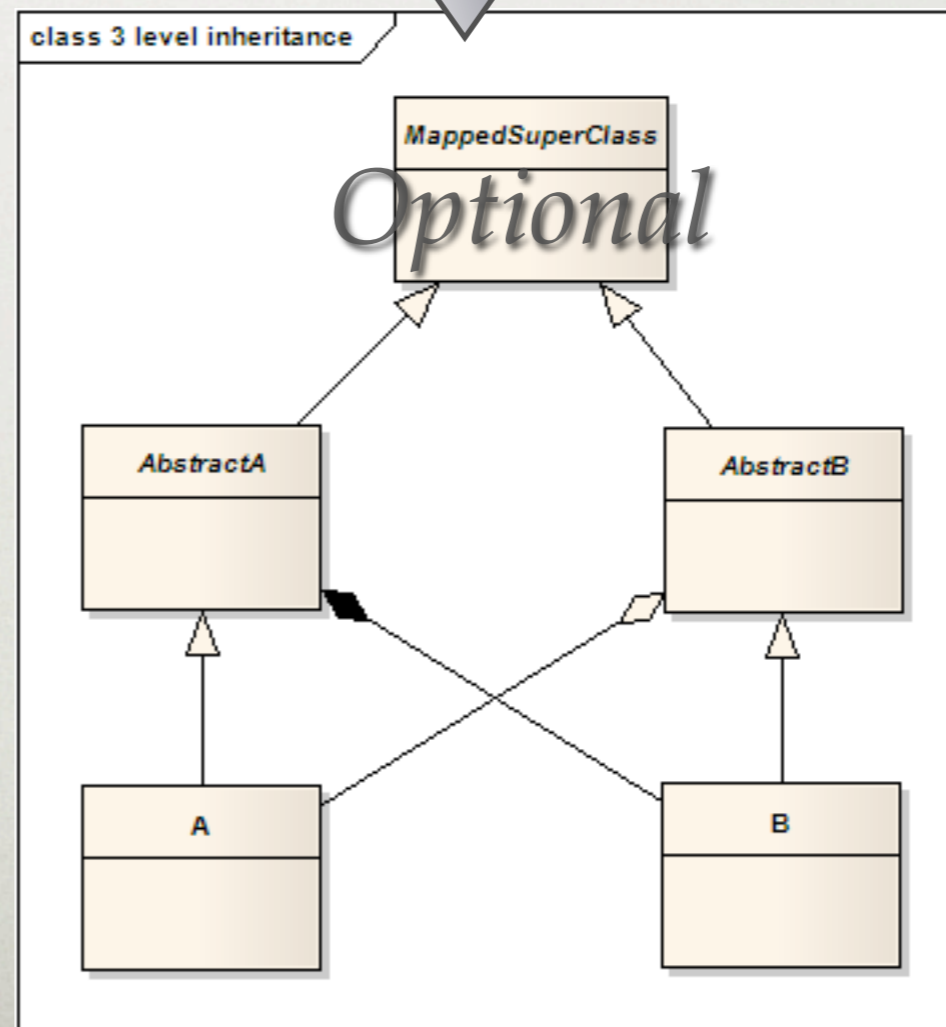


3 LEVEL INHERITANCE AKA. EXTENSION CLASSES

What we model:



What we generate:



CODE GENERATION WITH FORMATTING AND PACKAGE REWRITING

```
String[] excludeFormattingForPackages = {"dk.tigerteam.example.model"};
String generateClazzesAndBaseClazzesToDirectory = "src/main/generated";
String generateExtensionClazzesToDirectory = "src/main/generated-extensions";
String generateTestClazzesToDirectory = "src/test/generated";
String generateInterfacesToDirectory = "src/main/generated";

XmiReader reader = new EAXmiReader();
MetaModel metaModel = reader.read(this.getClass().getResource("/model.xml"));

JavaCodeWriter codeWriter = new JalopyFormattingCodeWriter(generateClazzesAndBaseClazzesToDirectory,
                                                         generateExtensionClazzesToDirectory,
                                                         generateTestClazzesToDirectory,
                                                         generateInterfacesToDirectory,
                                                         new PackageRewriter() {
                                                             public String rewrite(String fqcn) {
                                                                 if (fqcn.startsWith("model")) {
                                                                     fqcn = "dk.tigerteam.example." + fqcn;
                                                                 }
                                                                 return fqcn;
                                                             }
                                                         }, excludeFormattingForPackages);

JavaGenerator javaGenerator = new JavaGenerator();
javaGenerator.addEventListeners(new BidirectionalGeneratorEventListener());
...
javaGenerator.addEventListeners(new HibernateValidatorNotNullListener());
javaGenerator.setCreateExtensionClazzes(true);

List<ClazzOrInterface> allGeneratedClazzes = javaGenerator.execute(metaModel, codeWriter);

OrmXmlGenerator ormXmlGenerator = new OrmXmlGenerator();
Document ormXmlDocument = ormXmlGenerator.generate(allGeneratedClazzes);
JDomWriter.write(ormXmlDocument, generateClazzesAndBaseClazzesToDirectory + File.separator + "META-INF", "orm.xml");
```

MORE LIVE CODE EXAMPLES

THANK YOU FOR YOUR
ATTENTION

JEPPE@TIGERTEAM.DK