Context of this work

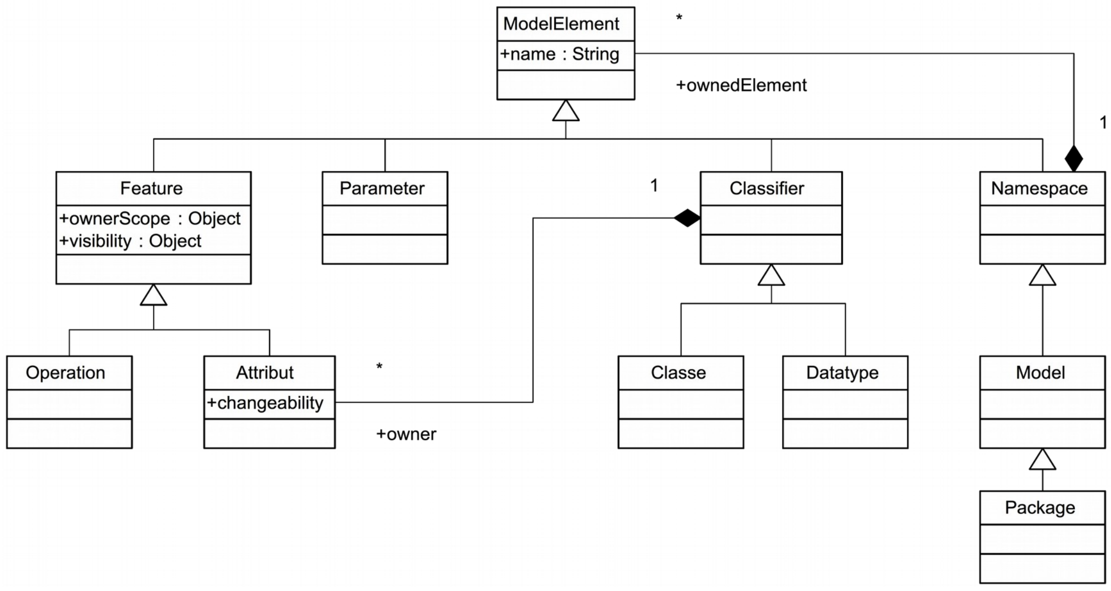
The present courseware has been elaborated in the context of the MODELWARE European IST FP6 project (http://www.modelware-ist.org/).

Co-funded by the European Commission, the MODELWARE project involves 19 partners from 8 European countries. MODELWARE aims to improve software productivity by capitalizing on techniques known as Model-Driven Development (MDD).

To achieve the goal of large-scale adoption of these MDD techniques, MODELWARE promotes the idea of a collaborative development of courseware dedicated to this domain.

1

2 Description of the Exercise

Short Name: UML2Java
Full Name: From UML Class Diagrams to Java Classes
Short Description: The UML to Java exercise requires a transformation of a UML model to a simplified Java model. The Java model holds the information for the creation of Java classes, especially what concerns the structure of these classes, namely the package reference, the attributes and the methods.
Tools: The required transformation may be implemented in any model transformation language. The description of the exercise in this document assumes that metamodels will be expressed in KM3 [2] and ATL [3] will be used for writing the transformation program. ATL engine can be downloaded from GMT web site [3].
Source Metamodels: The source metamodel is a simplified version of the UML metamodel standardized by OMG [1]. It is shown in Figure 1.  ModelElement +name : String * +ownedElement 1 Feature +ownerScope : Object +visibility : Object Parameter 1 Classifier Namespace Operation Attribut +changeability * +owner Classe Datatype Model Package **Figure 1 Simplified UML metamodel**
Target Metamodels: The target metamodel captures some characteristics of Java classes. A possible target metamodel of Java (see Figure 2) consists principally of JavaElements which all have a name. A JavaClass has Methods and Fields and belongs to a Package. Methods, Fields and JavaClasses are subclasses of Modifiers and therefore indicate whether they are public, static or final. JavaClasses and Methods declare with the isAbstract attribute whether they are abstract or not. PrimitiveTypes and JavaClasses are Types. A Method has a Type as return Type and parameters of certain Types. A Field has also a Type.

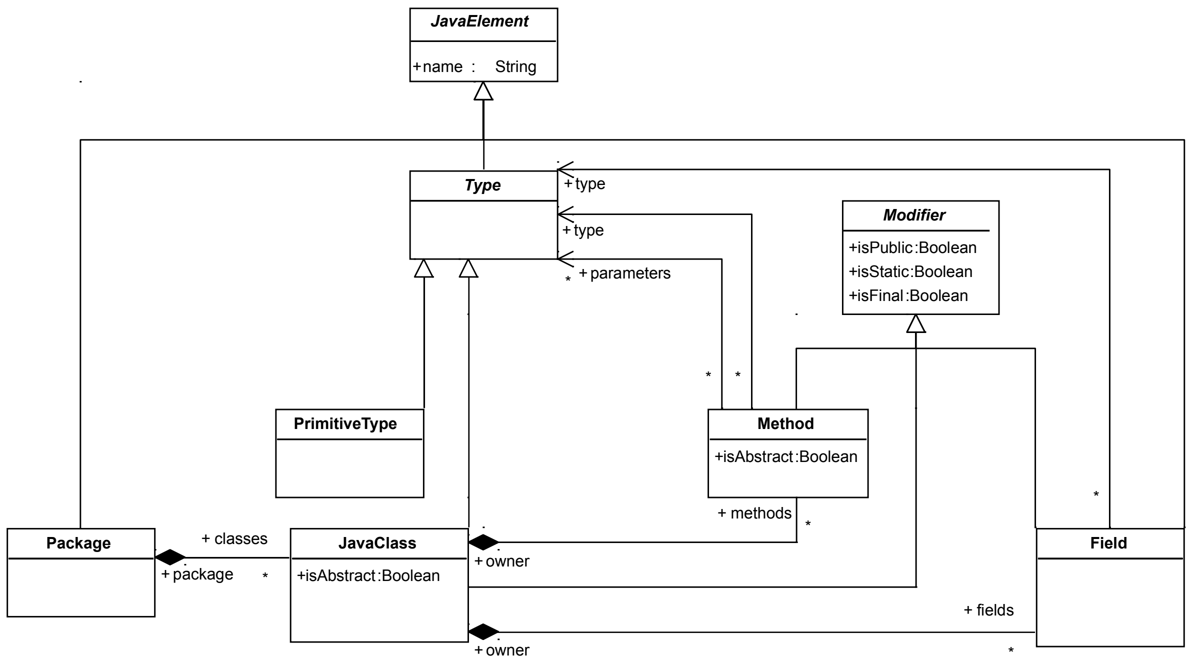


Figure 2 Simplified Java metamodel

Tasks:

1. Express the source and target metamodels in the KM3 language.
2. Write a transformation program that transforms UML models conforming to the source metamodel to Java models conforming to the target metamodel. The informal description of the transformation rules is the following:

Rule 1. For each UML Package instance, a Java Package instance has to be created.

- Their names have to correspond. However, in contrast to UML Packages which hold simple names, the Java Package name contains the full path information. The path separation is a point “.”.

Rule 2. For each UML Class instance, a JavaClass instance has to be created.

- Their names have to correspond.
- The Package reference has to correspond.
- The Modifiers have to correspond.

Rule 3. For each UML DataType instance, a Java PrimitiveType instance has to be created.

- Their names have to correspond.
- The Package reference has to correspond.

Rule 4. For each UML Attribute instance, a Java Field instance has to be created.

- Their names have to correspond.
- Their Types have to correspond.
- The Modifiers have to correspond.
- The Classes have to correspond.

Rule 5. For each UML Operation instance, a Java Method instance has to be created.

- Their names have to correspond.
- Their Types have to correspond.
- The Modifiers have to correspond.
- The Classes have to correspond.

3. Test the transformation program with a sample input UML model.

Guidelines: First try to write the transformation program by yourself. If you have difficulties then you can consult the solution given further in this document. Remember that there is no single right solution. It is always useful to compare your solution with other solutions.

3 An Example Solution

Example ATL Code:

This ATL code for the transformation of a UML to Java consists of several functions and rules. Among the functions, it is important to mention getExtendedName which recursively explores the namespace to concatenate a full path name. Concerning the rules, there are remarks necessary respective the rule O2M. This rule shows how to access sets via OCL expressions. For simplicity of implementation, the (return) type of a Java Method is the first parameter of an UML Operation. Some minor details, such as modifiers, are not yet fully implemented.

```
module UML2JAVA;
create OUT : JAVA from IN : UML;

helper context UML!ModelElement def: isPublic() : Boolean =
      self.visibility = #vk_public;

helper context UML!Feature def: isStatic() : Boolean =
      self.ownerScope = #sk_static;

helper context UML!Attribute def: isFinal() : Boolean =
      self.changeability = #ck_frozen;

helper context UML!Namespace def: getExtendedName() : String =
      if self.namespace.oclIsUndefined() then
            ''
      else if self.namespace.oclIsKindOf(UML!Model) then
            ''
      else
            self.namespace.getExtendedName() + '.'
      endif endif + self.name;

rule P2P {
      from e : UML!Package (e.oclIsTypeOf(UML!Package))
      to out : JAVA!Package (
            name <- e.getExtendedName()
      )
}

rule C2C {
      from e : UML!Class
      to out : JAVA!JavaClass (
            name <- e.name,
            isAbstract <- e.isAbstract,
            isPublic <- e.isPublic(),
            package <- e.namespace
```

```
	)
}

rule D2P {
	from e : UML!DataType
	to out : JAVA!PrimitiveType (
		name <- e.name,
		package <- e.namespace
	)
}

rule A2F {
	from e : UML!Attribute
	to out : JAVA!Field (
		name <- e.name,
		isStatic <- e.isStatic(),
		isPublic <- e.isPublic(),
		isFinal <- e.isFinal(),
		owner <- e.owner,
		type <- e.type
	)
}

rule O2M {
	from e : UML!Operation
	to out : JAVA!Method (
		name <- e.name,
		isStatic <- e.isStatic(),
		isPublic <- e.isPublic(),
		owner <- e.owner,
		type <- e.parameter->select(x|x.kind=#pdk_return)->
			asSequence()->first().type,
		parameters <- e.parameter->select(x|x.kind<>#pdk_return)->
		asSequence()
	)
}

rule P2F {
	from e : UML!Parameter (e.kind <> #pdk_return)
	to out : JAVA!FeatureParameter (
		name <- e.name,
		type <- e.type
	)
}
```

References

[1] OMG Unified Modeling Language (UML), version 1.4 (formal/03-03-01), 2002, http://www.omg.org/technology/documents/formal/uml.htm

[2] KM3 User Manual. http://eclipse.org/gmt

[3] The ATL Development Tools. http://eclipse.org/gmt