Model Management
in Xtend
(second part)

Mathieu Acher
Maître de Conférences
mathieu.acher@irisa.fr

Material

http://mathieuacher.com/teaching/MDE/

Plan

- Model Management in a nutshell
  - Loading, serializing, transforming models
- Xtend
  - Java 10, cheatsheet
  - **Advanced features: extension methods, active annotations, template expressions**
  - Xtend: behing the magic (Xtext+MDE)
- Model Management + Xtend
  - Model transformations
  - @Aspect annotation
  - Xtend + Xtext (breathing life into DSLs)

Contract

- Practical foundations of model management
- Learning and understanding Java 10 (aka Xtend)
  - advanced features of a general GPL, implementation of a sophisticated language using MDE
- Model transformations
  - Model-to-Text
  - Model-to-Model
- Metaprogramming
  - Revisit annotations (e.g., as in JPA or many frameworks)
- DSLs and model management: all together (Xtext + Xtend)

Active Annotations

(a practical way to transform your data, programs, models)

Do You know Java Annotations ?

JUnit

@Override

@SuppressWarnings

google-guice

Guice (pronounced 'juice') is a lightweight dependency injection framework for Java 5 and above, brought to you by Google.

JUnit

```java
package com.vogella.junit.first;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ MyClassTest.class, MySecondClassTest.class })
public class AllTests {

}
```

```java
public class MyClassTest {

  @BeforeClass
  public static void testSetup() {
  }

  @AfterClass
  public static void testCleanup() {
    // Teardown for data used by the unit tests
  }

  @Test(expected = IllegalArgumentException.class)
  public void testExceptionIsThrown() {
    MyClass tester = new MyClass();
    tester.multiply(1000, 5);
  }

  @Test
  public void testMultiply() {
    MyClass tester = new MyClass();
    assertEquals("10 x 5 must be 50", 50, tester.multiply(10, 5));
  }
}
```

Annotations (Junit 4)

@Test public void method()	The `@Test` annotation identifies a method as a test method.
@Test (expected = Exception.class)	Fails, if the method does not throw the named exception.
@Test(timeout=100)	Fails, if the method takes longer than 100 milliseconds.
@Before public void method()	This method is executed before each test. It is used to can prepare the test environment (e.g. read input data, initialize the class).
@After public void method()	This method is executed after each test. It is used to cleanup the test environment (e.g. delete temporary data, restore defaults). It can also save memory by cleaning up expensive memory structures.
@BeforeClass public static void method()	This method is executed once, before the start of all tests. It is used to perform time intensive activities, for example to connect to a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.
@AfterClass public static void method()	This method is executed once, after all tests have been finished. It is used to perform clean-up activities, for example to disconnect from a database. Methods annotated with this annotation need to be defined as `static` to work with JUnit.

http://www.vogella.com/articles/JUnit/article.html#usingjunit_annotations

Java Annotations

```java
@XmlRootElement
public class Customer {

        String name;
        int age;
        int id;

        public String getName() {
                return name;
        }

        @XmlElement
        public void setName(String name) {
                this.name = name;
        }

        public int getAge() {
                return age;
        }

        @XmlElement
        public void setAge(int age) {
                this.age = age;
        }

        public int getId() {
                return id;
        }

        @XmlAttribute
        public void setId(int id) {
                this.id = id;
        }

}
```

```java
Customer customer = new Customer();
customer.setId(100);
customer.setName("mkyong");
customer.setAge(29);
```


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<customer id="100">
    <age>29</age>
    <name>mkyong</name>
</customer>
```

2.2.1. Marking a POJO as persistent entity

Every persistent POJO class is an entity and is declared using the @Entity annotation (at the class level):

```
@Entity
public class Flight implements Serializable {
    Long id;

    @Id
    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
}
```

@Entity declares the class as an entity (i.e. a persistent POJO class), @Id declares the identifier property of this entity. The other mapping declarations are implicit. The class Flight is mapped to the Flight table, using the column id as its primary key column.

```
@Entity
class MedicalHistory implements Serializable {
  @Id @OneToOne
  @JoinColumn(name = "person_id")
  Person patient;
}

@Entity
public class Person implements Serializable {
  @Id @GeneratedValue Integer id;
}
```

Javadoc
(old fashion, not real annotations)

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
       try {
           return getImage(new URL(url, name));
       } catch (MalformedURLException e) {
           return null;
       }
}
```

Disclaimer

- @AhaMoment
- @BossMadeMeDoIt
- @HandsOff
- @IAmAwesome
- @LegacySucks

Enforceable

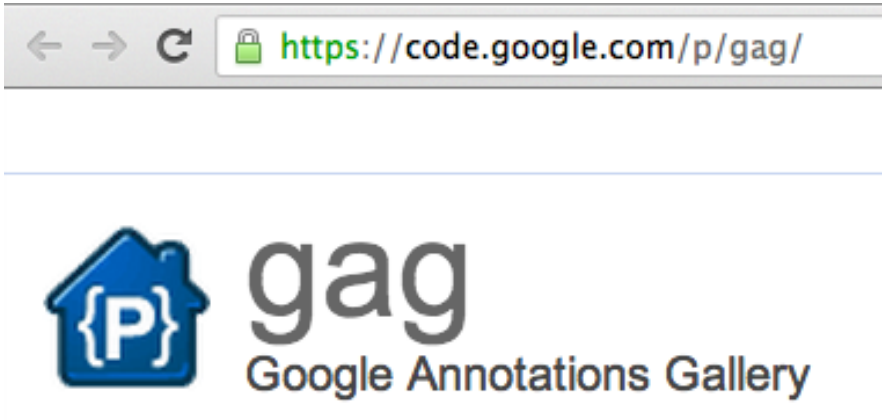
- @CantTouchThis
- @ImaLetYouFinishBut

Literary Verse (new subcategory)

- @Burma Shave
- @Clerihew
- @DoubleDactyl
- @Haiku (moved to this subcategory)
- @Limerick
- @Sonnet

Remarks

- @Fail
- @OhNoYouDidnt
- @RTFM
- @Win



The Google Annotations Gallery is an exciting new Java open source library that provides a rich set of annotations for developers to express themselves.
Do you find the standard Java annotations dry and lackluster? Have you ever resorted to leaving messages to fellow developers with the @Deprecated annotation? Wouldn't you rather leave a @LOL or @Facepalm instead?

Not only can you leave expressive remarks in your code, you can use these annotations to draw attention to your poetic endeavors. How many times have you written a palindromic or synecdochal line of code and wished you could annotate it for future readers to admire? Look no further than @Palindrome and @Synecdoche.

But wait, there's more. The Google Annotations Gallery comes complete with dynamic bytecode instrumentation. By using the gag-agent.jar Java agent, you can have your annotations behavior-enforced at runtime. For example, if you want to ensure that a method parameter is non-zero, try @ThisHadBetterNotBe(Property.ZERO). Want to completely inhibit a method's implementation? Try @Noop.

Annotations for…

- Documentation
  - Javadoc like
- Information to the Compiler
  - Supress warnings, error detections
- Generation
  - Code (Java, SQL, etc.)
  - Configuration files (e.g., XML-like)
- Runtime processing

⇒**Transformation of programs, datas, models**

⇒ **You can define your own**

Annotations: How does it work?

junit.sourceforge.net/javadoc/org/junit/Test.html

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS

SUMMARY: REQUIRED | OPTIONAL

org.junit

Annotation Type Test

```
@Retention(value=RUNTIME)
@Target(value=METHOD)
public @interface Test
```

The `Test` annotation tells JUnit that the `public void` method to which it
exceptions are thrown, the test is assumed to have succeeded.

A simple test looks like this:

```
public class Example {
   @Test
   public void method() {
      org.junit.Assert.assertTrue( new ArrayList().isEmpty
   }
}
```

The `Test` annotation supports two optional parameters. The first, `expecte`

```
   @Test(expected=IndexOutOfBoundsException.class) public
      new ArrayList<Object>().get(1);
   }
```

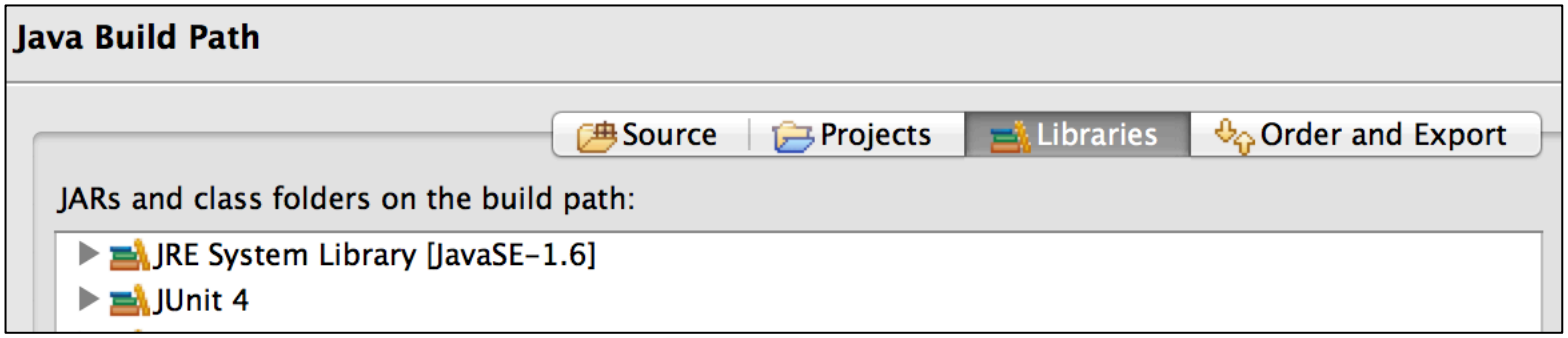
The second optional parameter, `timeout`, causes a test to fail if it takes lon

```
   @Test(timeout=100) public void infinity() {
      while(true);
   }
```

Annotations: How does it work?

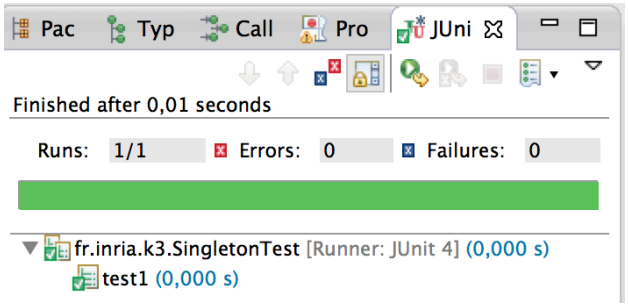
GitHub, Inc. [US] https://github.com/junit-team/junit/blob/master/src/main/java/org/junit/Test.java

```
60  @Retention(RetentionPolicy.RUNTIME)
61  @Target({ElementType.METHOD})
62  public @interface Test {
63
64      /**
65       * Default empty exception
66       */
67      static class None extends Throwable {
68          private static final long serialVersionUID = 1L;
69
70          private None() {
71          }
72      }
73
74      /**
75       * Optionally specify <code>expected</code>, a Throwable, to cause a
76       * and only if an exception of the specified class is thrown by the
77       */
78      Class<? extends Throwable> expected() default None.class;
79
80      /**
81       * Optionally specify <code>timeout</code> in milliseconds to cause
82       * takes longer than that number of milliseconds.
83       * <p>
84       * <b>THREAD SAFETY WARNING:</b> Test methods with a timeout paramet
85       * thread which runs the fixture's @Before and @After methods. This
86       * code that is not thread safe when compared to the same test metho
87       * <b>Consider using the {@link org.junit.rules.Timeout} rule instea
88       * same thread as the fixture's @Before and @After methods.
89       * </p>
90       */
91      long timeout() default 0L;
92  }
```

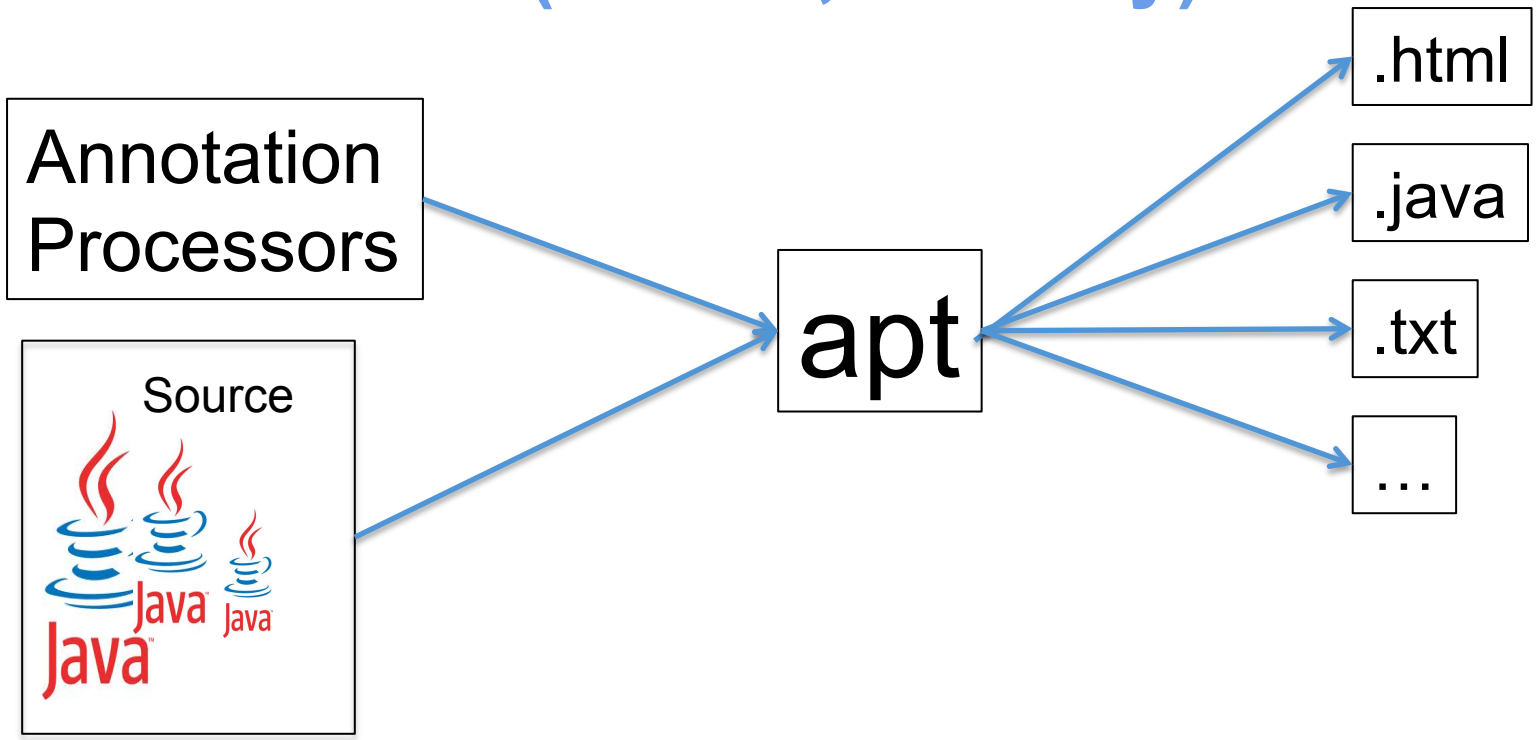



```java
package com.vogella.junit.first;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ MyClassTest.class, MySecondClassTest.class })
public class AllTests {

}
```

Transformation of Java code



Annotations and Transformations (Java 5, old way)



docs.oracle.com/javase/1.5.0/docs/guide/apt/GettingStarted.html

Java

Getting Started with the Annotation Processing Tool (`apt`)

What is `apt`?

The command-line utility `apt`, annotation processing tool, finds and executes *annotation processors* based on the annotations present in the set of specified source files being examined. The annotatio

Annotations and Transformations (Java 5, old way)

Annotation Processors

apt

```
/*
 * This class is used to run an annotation processor that lists class
 * names.  The functionality of the processor is analogous to the
 * ListClass doclet in the Doclet Overview.
 */
public class ListClassApf implements AnnotationProcessorFactory {
    // Process any set of annotations
    private static final Collection<String> supportedAnnotations
        = unmodifiableCollection(Arrays.asList("*"));

    // No supported options
    private static final Collection<String> supportedOptions = emptySet();

    public Collection<String> supportedAnnotationTypes() {
        return supportedAnnotations;
    }

    public Collection<String> supportedOptions() {
        return supportedOptions;
    }

    public AnnotationProcessor getProcessorFor(
            Set<AnnotationTypeDeclaration> atds,
            AnnotationProcessorEnvironment env) {
        return new ListClassAp(env);
    }

    private static class ListClassAp implements AnnotationProcessor {
        private final AnnotationProcessorEnvironment env;
        ListClassAp(AnnotationProcessorEnvironment env) {
            this.env = env;
        }

        public void process() {
            for (TypeDeclaration typeDecl : env.getSpecifiedTypeDeclarations())
                typeDecl.accept(getDeclarationScanner(new ListClassVisitor(),
                                                      NO_OP));
        }

        private static class ListClassVisitor extends SimpleDeclarationVisitor {
            public void visitClassDeclaration(ClassDeclaration d) {
                System.out.println(d.getQualifiedName());
            }
        }
    }
}
```

The `apt` Command Line

In addition to its own options, the `apt` tool accepts all of the command-line options accep

The `apt` specific options are:

`-s` *dir*
: Specify the directory root under which processor-generated source files will be pla

`-nocompile`
: Do not compile source files to class files.

`-print`
: Print out textual representation of specified types; perform no annotation processing

`-A`[*key*[`=`*val*]]
: Options to pass to annotation processors -- these are not interpreted by `apt` directly

`-factorypath` *path*
: Specify where to find annotation processor factories; if this option is used, the class

`-factory` *classname*
: Name of `AnnotationProcessorFactory` to use; bypasses default discovery proce

How `apt` shares some of `javac`'s options:

`-d` *dir*
: Specify where to place processor and javac generated class files

`-cp` *path* or `-classpath` *path*
: Specify where to find user class files and annotation processor factories. If `-facto`

There are a few `apt` hidden options that may be useful for debugging:

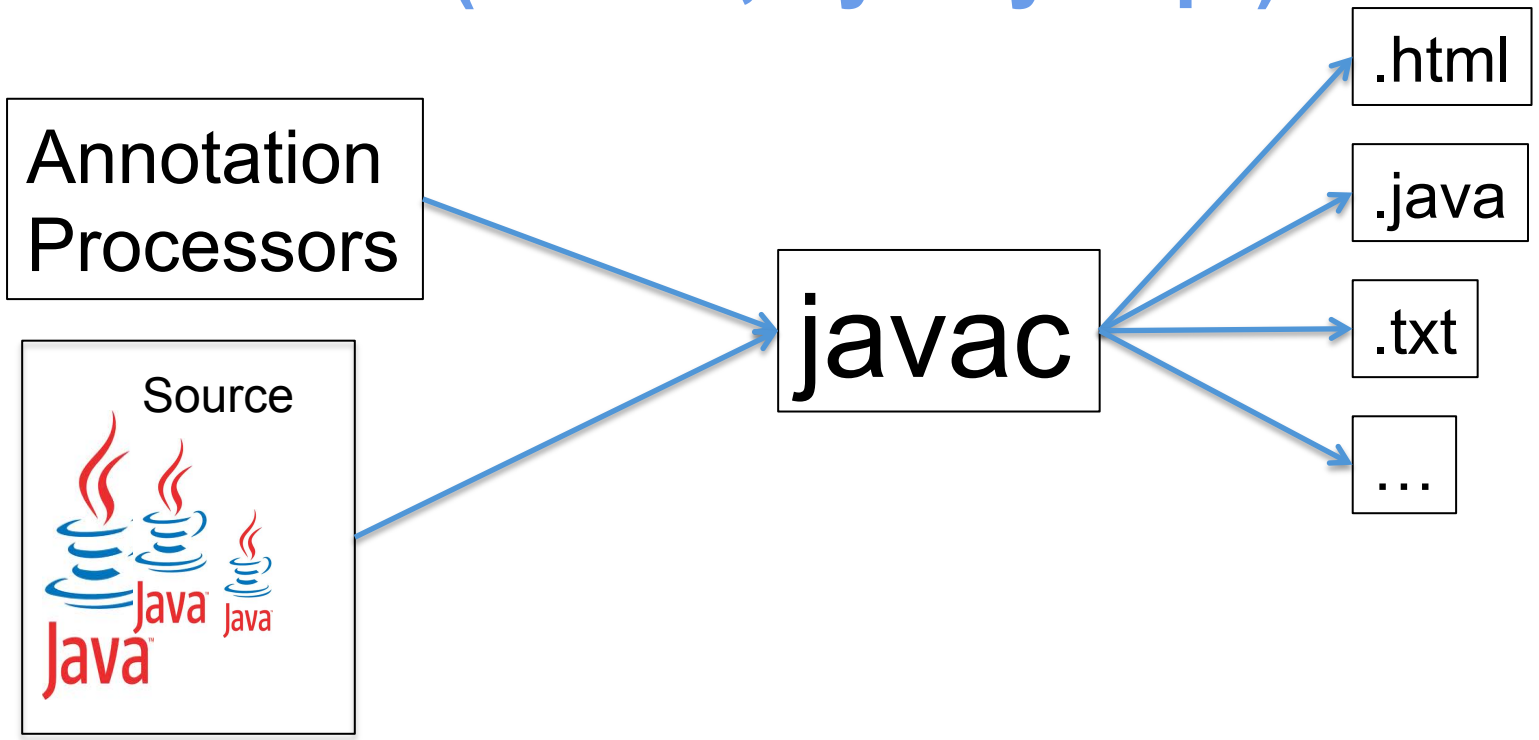
`-XListAnnotationTypes`
: List found annotation types

`-XListDeclarations`
: List specified and included declarations

`-XPrintAptRounds`
: Print information about initial and recursive `apt` rounds

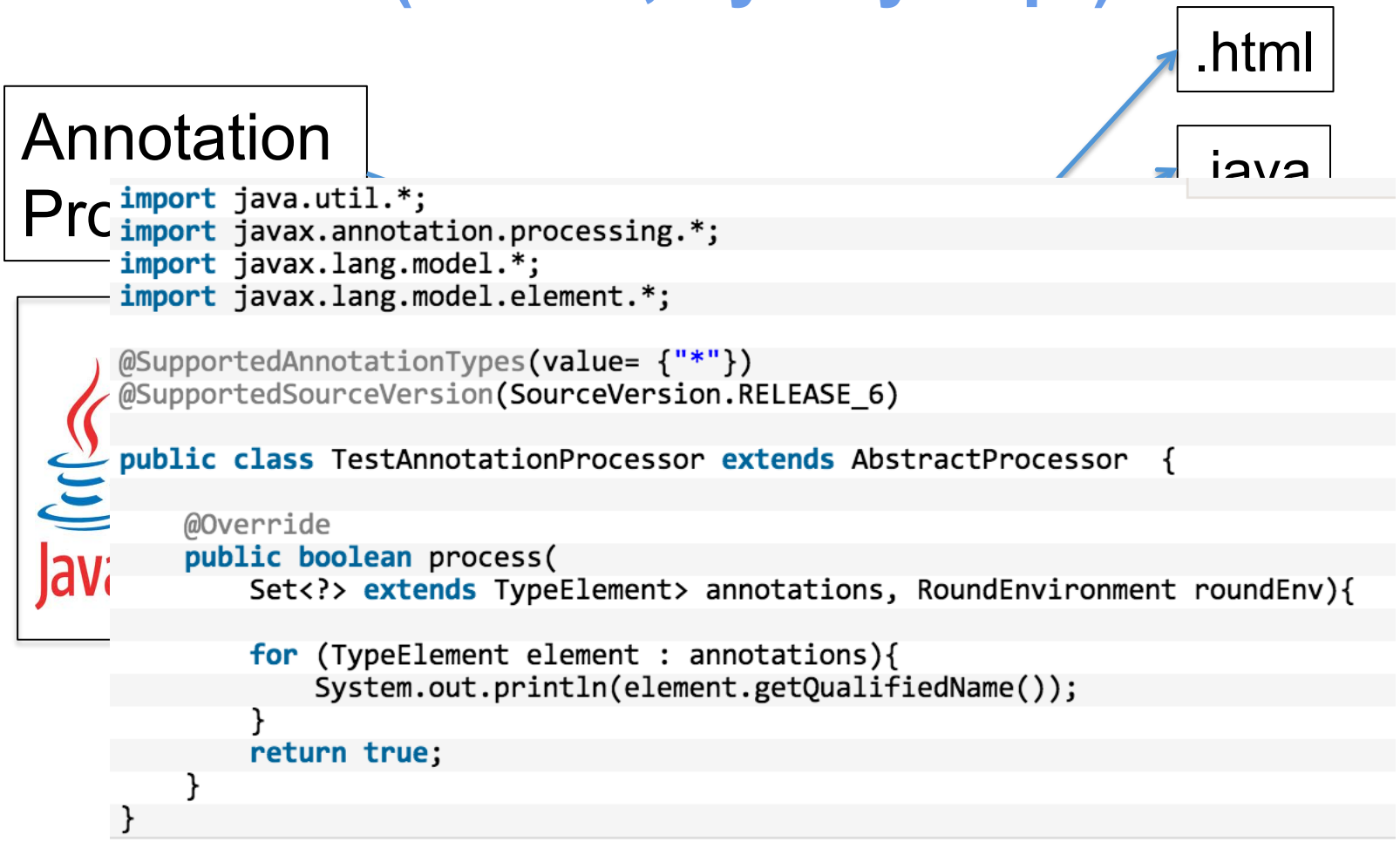
`-XPrintFactoryInfo`
: Print information about which annotations a factory is asked to process

Annotations and Transformations (Java 6, bye bye apt)



Integrated into the Java compiler (javac)
New API: Pluggable Annotation Processing

Annotations and Transformations (Java 6, bye bye apt)



javac –processor …

Alternative: Java Reflection

```java
import java.lang.annotation.Documented;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Documented
@Retention(RetentionPolicy.RUNTIME)
public @interface Todo {

  public enum Importance {
    MINEURE, IMPORTANT, MAJEUR, CRITIQUE
  };

  Importance importance() default Importance.MINEURE;

  String[] description();

  String assigneA();

  String dateAssignation();
}
```

```java
@Todo(importance = Importance.CRITIQUE,
      description = "Corriger le bug dans le calcul",
      assigneA = "JMD",
      dateAssignation = "11-11-2007")
public class TestInstrospectionAnnotation {

  public static void main(
      String[] args) {
    Todo todo = null;

    // traitement annotation sur la classe
    Class classe = TestInstrospectionAnnotation.class;
    todo = (Todo) classe.getAnnotation(Todo.class);
    if (todo != null) {
      System.out.println("classe "+classe.getName());
      System.out.println("  ["+todo.importance()+"]"+" ("+todo.assigneA()
        +" le "+todo.dateAssignation()+")");
      for(String desc : todo.description()) {
        System.out.println("      _ "+desc);
      }
    }

    // traitement annotation sur les méthodes de la classe
    for(Method m : TestInstrospectionAnnotation.class.getMethods()) {
      todo = (Todo) m.getAnnotation(Todo.class);
      if (todo != null) {
        System.out.println("methode "+m.getName());
        System.out.println("  ["+todo.importance()+"]"+" ("+todo.assigneA()
          +" le "+todo.dateAssignation()+")");
        for(String desc : todo.description()) {
          System.out.println("      _ "+desc);
        }
      }
    }
  }

  @Todo(importance = Importance.MAJEUR,
        description = "Implementer la methode",
        assigneA = "JMD",
        dateAssignation = "11-11-2007")
  public void methode1() {

  }

  @Todo(importance = Importance.MINEURE,
        description = {"Completer la methode", "Ameliorer les logs"},
        assigneA = "JMD",
        dateAssignation = "12-11-2007")
  public void methode2() {

  }

}
```

http://www.jmdoudoux.fr/java/dej/chap-annotations.htm#annotations-7

You can define your own annotations

- Specification
  - At the Class, Field, Method level
  - Annotations can be combined
  - Annotations can have parameters
- Transformation (compilation)
  - Introspection
  - Compiler (javac/apt) and definition of « processors »
- Widely used
  - Generation, verification, etc.

Back to Xtend

- Active Annotations
  - Facilities to specify Annotations and their treatment (API)
  - Seamless integration in the IDE
    - On-the-fly compilation to Java allows proper type checking and auto-completion

Example

```
package fr.inria.k3

@Singleton
class GUIWindow {


    int x ;
    int y ;

}
```

Example

```
package fr.inria.k3

@Singleton
class GUIWindow {


    int x ;
    int y ;

}
```

```
public final class GUIWindow {
  private GUIWindow() {
    // singleton
  }

  private int x;

  private int y;

  private final static GUIWindow INSTANCE = new GUIWindow();

  public static GUIWindow getINSTANCE() {
    return INSTANCE;
  }
}
```

```
package fr.inria.k3

@Singleton
class GUIWindow {


    int x ;
    int y ;

}
```

```
public final class GUIWindow {
  private GUIWindow() {
    // singleton
  }

  private int x;

  private int y;

  private final static GUIWindow INSTANCE = new GUIWindow();

  public static GUIWindow getINSTANCE() {
    return INSTANCE;
  }
}
```

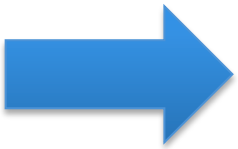
```
class SingletonProcessor extends AbstractClassProcessor {

    override doTransform(MutableClassDeclaration annotatedClass, extension TransformationContext context) {

        annotatedClass.final = true

        if (annotatedClass.declaredConstructors.size > 1)
            annotatedClass.addError("More then one constructor is defined")

        val constructor = annotatedClass.declaredConstructors.head
        if (constructor.parameters.size > 0)
            constructor.addError("Constructor has arguments")

        if (constructor.body == null) {

            // no constructor defined in the annotated class
            constructor.visibility = Visibility::PRIVATE
            constructor.body = ['''// singleton''']
        } else {
            if (constructor.visibility != Visibility::PRIVATE)
                constructor.addError("Constructor is not private")
        }



        annotatedClass.addField('INSTANCE') [
            visibility = Visibility::PRIVATE
            static = true
            final = true
            type = annotatedClass.newTypeReference
            initializer = [
                '''new «annotatedClass.simpleName»()'''
            ]
        ]

        annotatedClass.addMethod('getINSTANCE') [
            visibility = Visibility::PUBLIC
            static = true
            returnType = annotatedClass.newTypeReference
            body = [
                '''return INSTANCE;'''
            ]
        ]
    }
```

Example (2)

```
package fr.inria.k3

@Extract
class ExtractA {

}
```

```
package fr.inria.k3;

import fr.inria.k3.Extract;

@Extract
@SuppressWarnings("all")
public class ExtractA implements ExtractAInterface {
}
```

```
package fr.inria.k3

@Extract
class ExtractA {

}
```



```
package fr.inria.k3;

import fr.inria.k3.Extract;

@Extract
@SuppressWarnings("all")
public class ExtractA implements ExtractAInterface {
}
```

```
/**
 * Extracts an interface for all locally declared public methods.
 */
@Target(ElementType.TYPE)
@Active(ExtractProcessor)
annotation Extract {}

class ExtractProcessor extends AbstractClassProcessor {

    override doRegisterGlobals(ClassDeclaration annotatedClass, RegisterGlobalsContext context) {
        context.registerInterface(annotatedClass.interfaceName)
    }

    def getInterfaceName(ClassDeclaration annotatedClass) {
        annotatedClass.qualifiedName+"Interface"
    }

    override doTransform(MutableClassDeclaration annotatedClass, extension TransformationContext context) {
        val interfaceType = findInterface(annotatedClass.interfaceName)

        // add the interface to the list of implemented interfaces
        annotatedClass.implementedInterfaces = annotatedClass.implementedInterfaces + #[interfaceType.newTypeReference]

        // add the public methods to the interface
        for (method : annotatedClass.declaredMethods) {
            if (method.visibility == Visibility.PUBLIC) {
                interfaceType.addMethod(method.simpleName) [
                    docComment = method.docComment
                    returnType = method.returnType
                    for (p : method.parameters) {
                        addParameter(p.simpleName, p.type)
                    }
                    exceptions = method.exceptions
                ]
            }
        }
    }

}
```

Predefined Annotations

```
@Singleton
class SingletonA {

    @Property
    int a = 13 ;

    @Property
    int b ;

    @Property
    String c ;

}
```

```
@Singleton
@SuppressWarnings("all")
public final class SingletonA {
  private SingletonA() {
    // singleton
  }

  private int _a = 13;

  public int getA() {
    return this._a;
  }

  public void setA(final int a) {
    this._a = a;
  }

  private int _b;

  public int getB() {
    return this._b;
  }

  public void setB(final int b) {
    this._b = b;
  }

  private String _c;

  public String getC() {
    return this._c;
  }

  public void setC(final String c) {
    this._c = c;
  }

  private final static SingletonA INSTANCE = new SingletonA();

  public static SingletonA getINSTANCE() {
    return INSTANCE;
  }
}
```

Plan

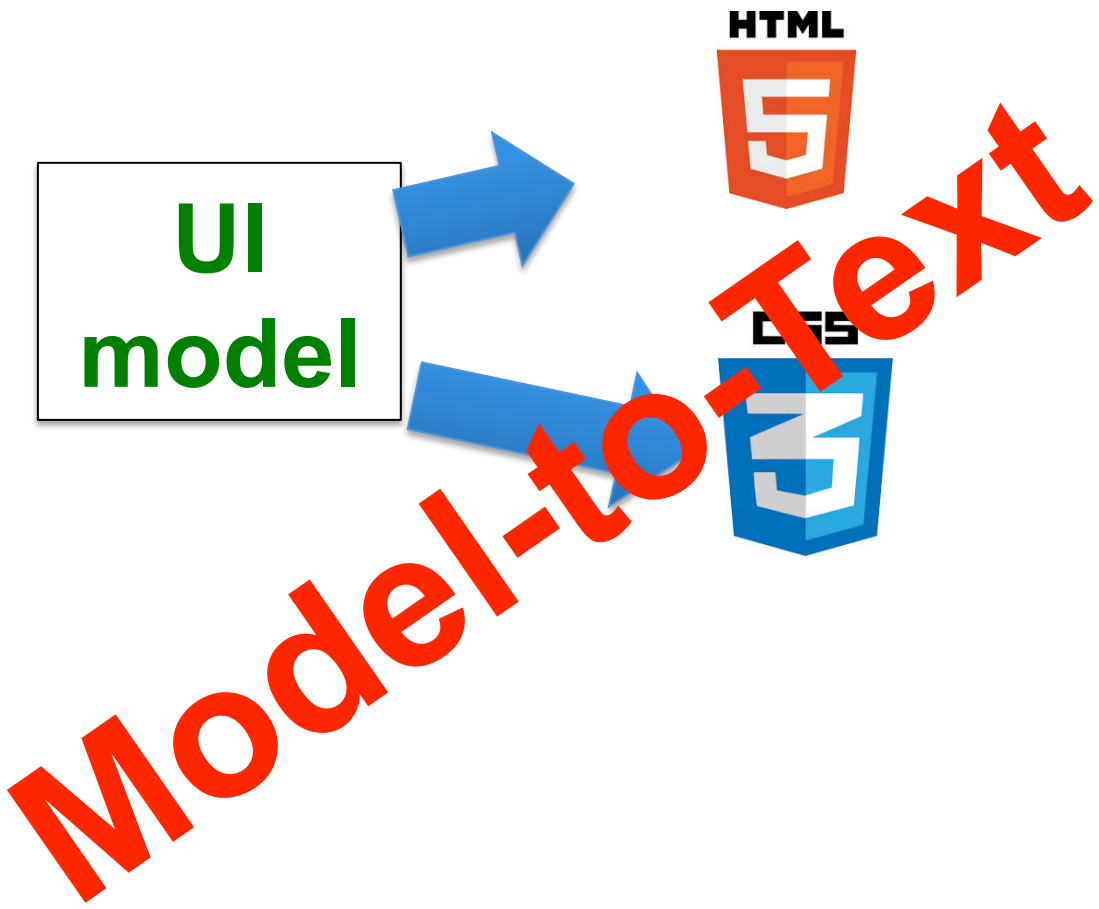
- Model Management in a nutshell
  - Loading, serializing, transforming models
- Xtend
  - Java 10, cheatsheet
  - Advanced features: extension methods, active annotations, template expressions
  - Xtend: behing the magic (Xtext+MDE)
- Model Management + Xtend
  - Model transformations
  - @Aspect annotation
  - Xtend + Xtext (breathing life into DSLs)

Contract

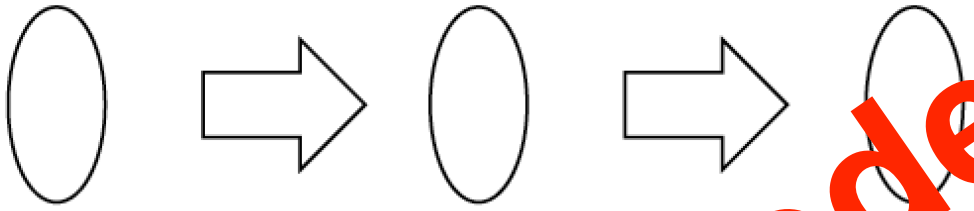
- Practical foundations of model management
- Learning and understanding Java 10 (aka Xtend)
  - advanced features of a general GPL, implementation of a sophisticated language using MDE
- Model transformations
  - Model-to-Text
  - Model-to-Model
- Metaprogramming
  - Revisit annotations (e.g., as in JPA or many frameworks)
- DSLs and model management: all together (Xtext + Xtend)

Model Transformation

M2T
M2M


UI
model
HTML
CSS
Model-to-Text

One step/stage transformation
hardly the case



a. Chaining of mappings

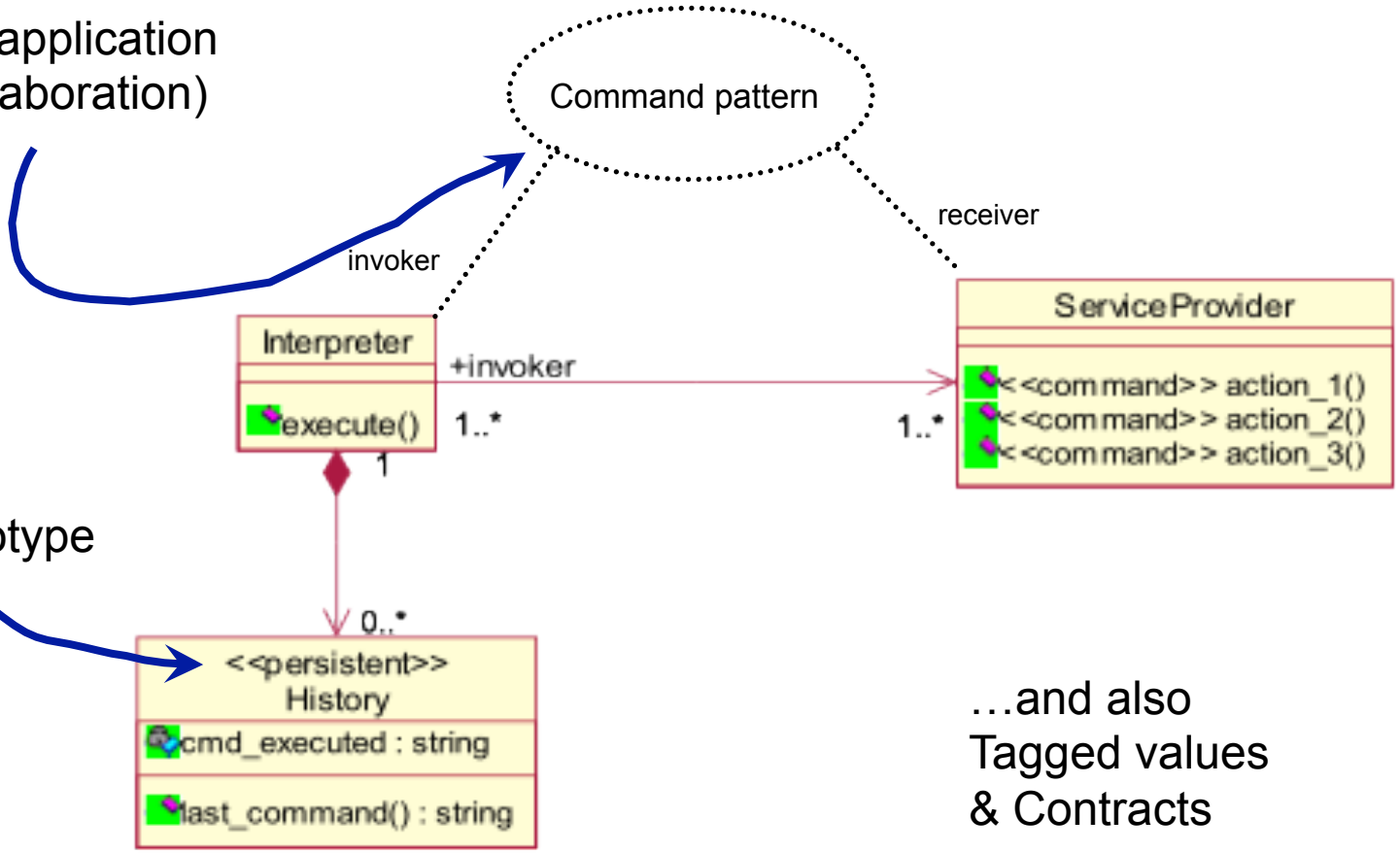
Model-to-Model

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```

UI model

Embedding implicit semantics into a model

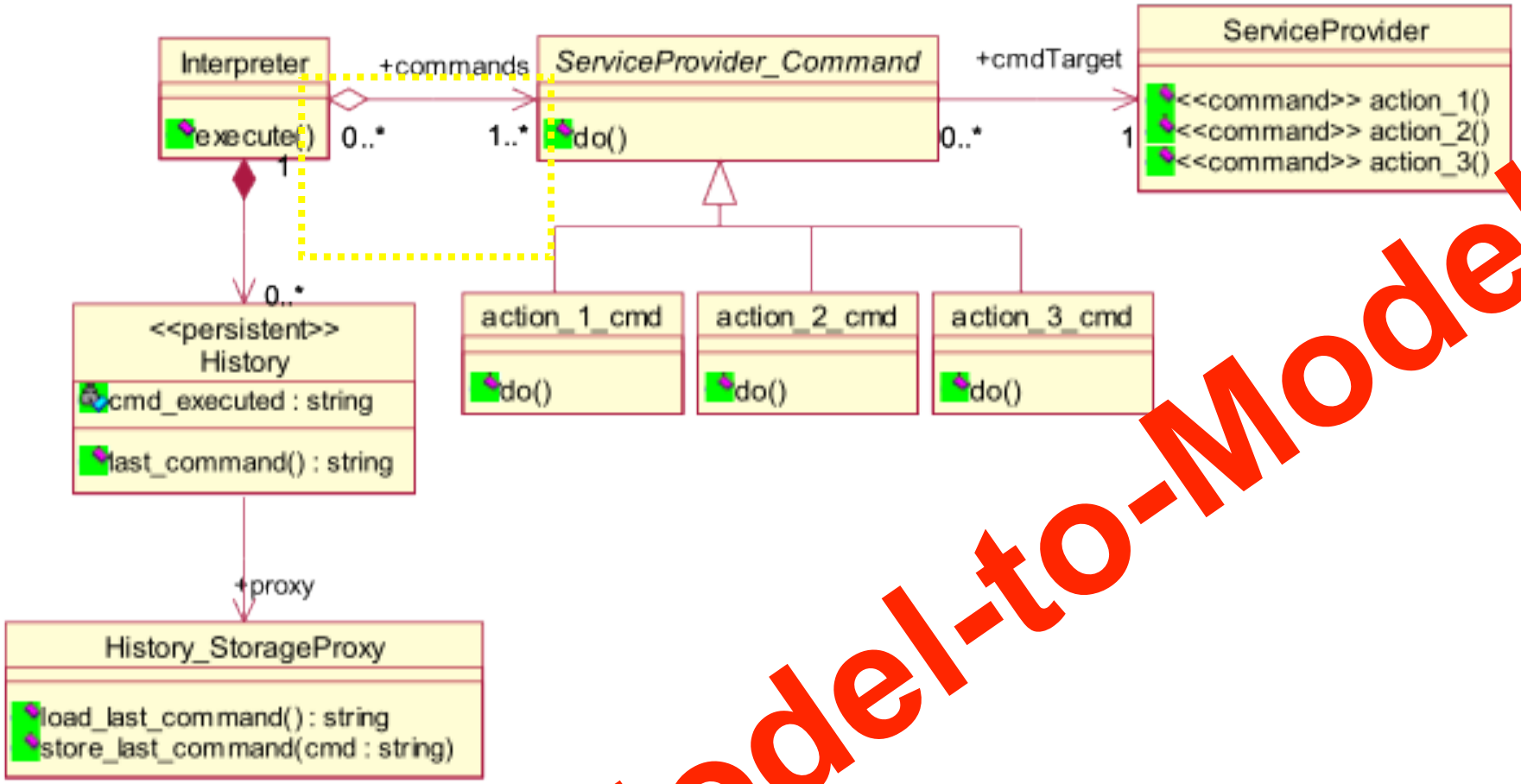
Design pattern application
(parametric collaboration)



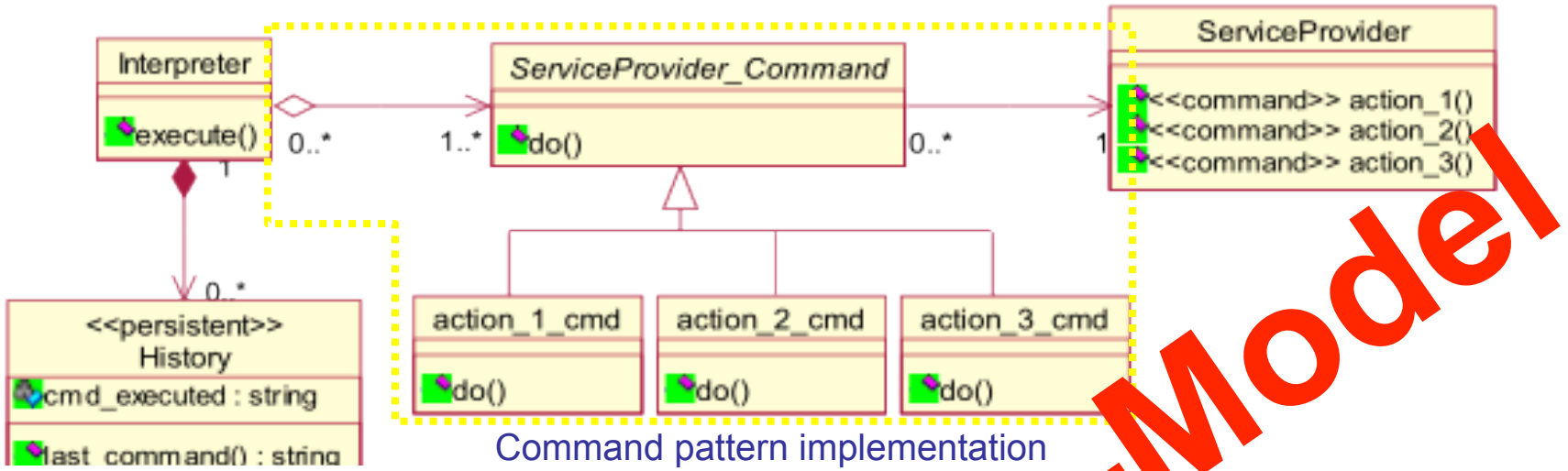
Element stereotype

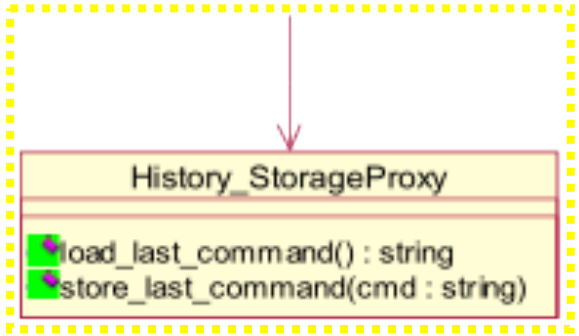
…and also
Tagged values
& Contracts

…and the result we want...



How To:
Automatic Model Transformations





Persitence implementation

Model-to-Model

Model-to-Text (M2T) vs. Model-to-Model (M2M)

- M2T Transformations
  - Should be limited to syntactic level transcoding
- M2M Transformations
  - To handle more complex, semantic driven transformations

M2T Approaches

- For generating: code, XML, HTML, doc.
  - Visitor-Based Approaches:
    - Some visitor mechanisms to traverse the internal representation of a model and write code to a text stream
    - Iterators, Write ()
      - **e.g., Processors (Annotations)**
  - Template-Based Approaches
    - A template consists of the target text containing slices of meta-code to access information from the source and to perform text selection and iterative expansion
    - The structure of a template resembles closely the text to be generated
    - Textual templates are independent of the target language and simplify the generation of any textual artefacts

Classification of M2M Transformation Techniques

1. General purpose programming languages
   - Java/C#...
2. Generic transformation tools
   - Graph transformations, XSLT…
3. CASE tools scripting languages
   - Objecteering, Rose…
4. Dedicated model transformation tools
   - OMG QVT style
5. Meta-modeling tools
   - Metacase, Xactium, Kermeta…

- Processors of Annotations can also be used

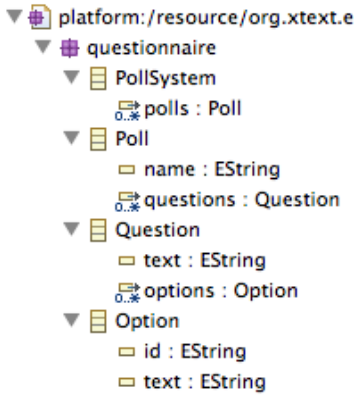
Transformations

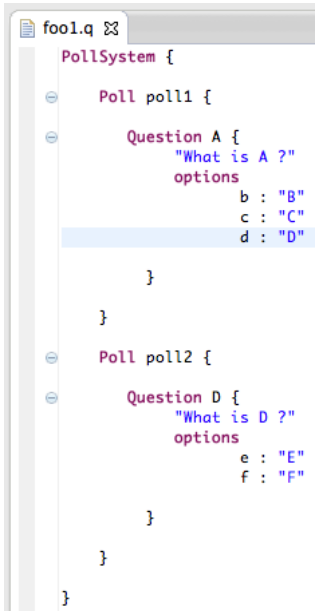
in Xtend

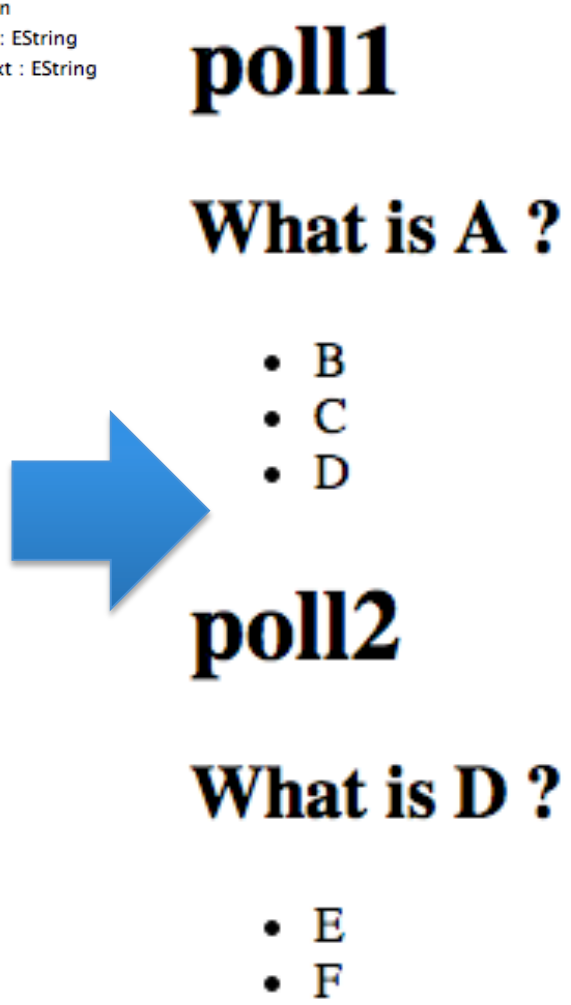
Templates (1)

```
1  package fr.inria.k3.templates
2
3  import org.junit.Test
4  import static org.junit.Assert.*
5
6  class FooTempl {
7
8
9      def someHTML(String content) '''<html><body>«content»</body></html>'''
10
11
12     @Test
13     def test1() {
14         assertEquals("<html><body>HW</body></html>", someHTML('HW').toString)
15     }
16
17 }
```

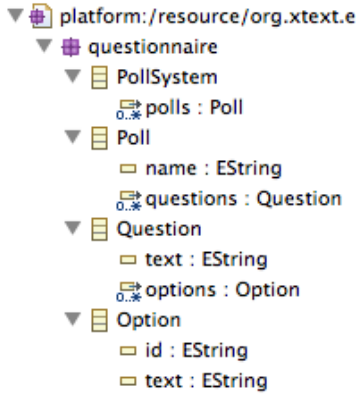
```
@Test
def test2() {

// loading
var pollS = loadPollSystem(URI.createURI("foo1.q"))

// MODEL MANAGEMENT (ANALYSIS, TRANSFORMATION)
var html = toPolls(pollS.polls)
assertNotNull(html)

// serializing (note: we could type check the HTML
// with Xtext by specifying the grammar for instance)
val fw = new FileWriter("foo1.html")
fw.write(html.toString)
fw.close


}

def toPolls(List<Poll> polls) '''
  <html>
    <body>
      «FOR p : polls»
        «IF p.name != null»
          <h1>«p.name»</h1>
        «ENDIF»
        «FOR q : p.questions»
        <p>
            <h2>«q.text»</h2>
            <ul>
             «FOR o : q.options»
             <li>«o.text»</li>
             «ENDFOR»
            </ul>
        </p>
        «ENDFOR»
      «ENDFOR»
    </body>
  </html>
'''
```





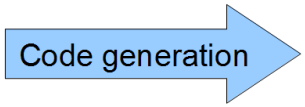


Facilities to create objects in a programmatic way



Ecore Model

Java Code



EPackage
EClass
EAttribute
EReference

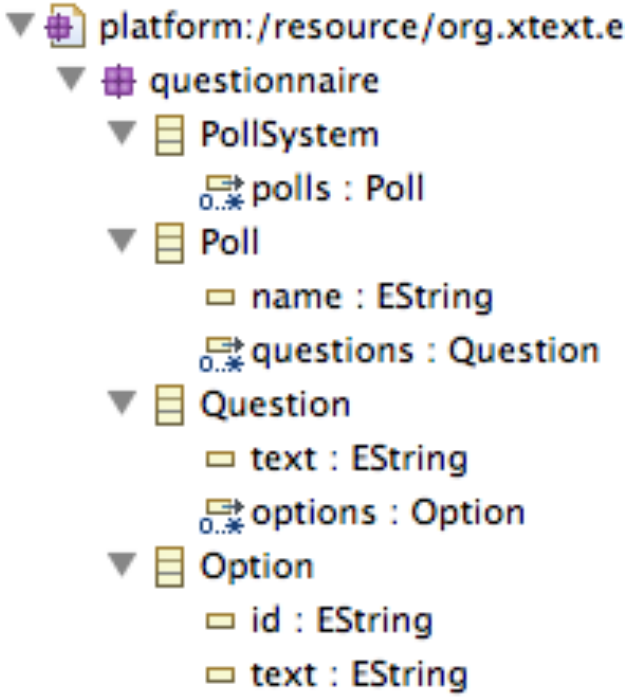
Package
Class
Attribute
Reference

```
@Test
def test2() {

    var pollSystem = QuestionnaireFactory.eINSTANCE.createPollSystem ;
    var p1 = QuestionnaireFactory.eINSTANCE.createPoll() ;
    p1.setName("p1");
    pollSystem.polls.add(p1)
    //
```

Visitors, EMF, and Xtend

(key to M2M or M2T:
iterate
over the model)

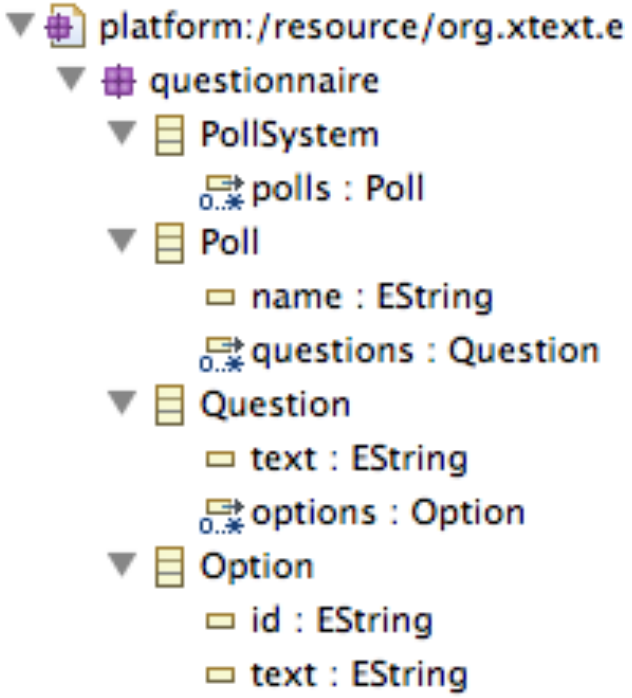
```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



We already give examples of transformation, defined over the metamodel…

Common point: the need to visit the model (graph)

```
PollSystem {
    Poll Quality {
        Question q1 {
            "Value the user experience"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
        Question q2 {
            "Value the layout"
            options {
                A : "It was not easy to locate elements"
                B : "I didn't realize"
                C : "It was easy to locate elements"
            }
        }
    }
    Poll Performance {
        Question q1 {
            "Value the time response"
            options {
                A : "Bad"
                B : "Fair"
                C : "Good"
            }
        }
    }
}
```



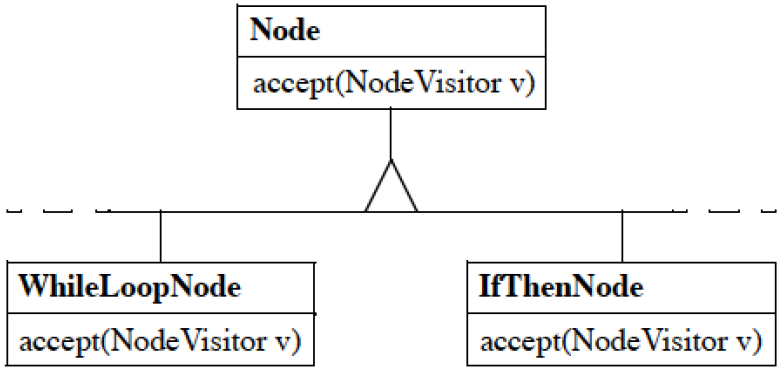
Visit the model (graph)

Possible solution: a series of casts (lots of if-statements and traversal loops)

Visitor Pattern

separating an algorithm from an object structure on which it operates

```
public class WhileLoopNode extends Node {
  protected Node condition, body;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitWhileLoop(this);
  }
}
public class IfThenNode extends Node {
  protected Node condition, thenBranch;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitIfThen(this);
  }
}
```

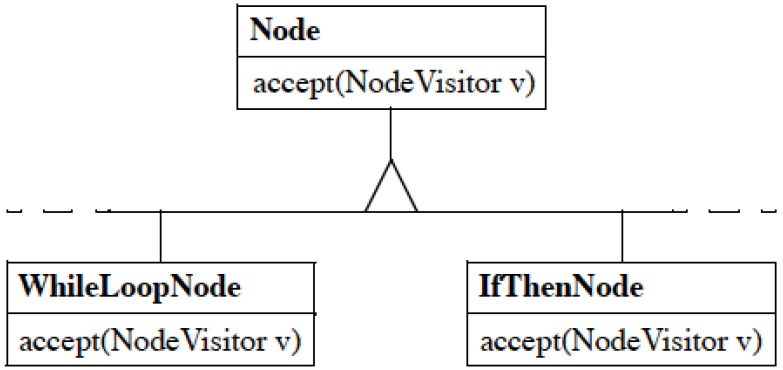



```
public abstract class NodeVisitor {
  /* ... */
  public abstract void visitWhileLoop(WhileLoopNode n);
  public abstract void visitIfThen(IfThenNode n);
}
public class TypeCheckingVisitor extends NodeVisitor {
  /* ... */
  public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
  public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

new operations can be added modularly, without needing to edit any of the Node subclasses: the programmer simply defines a new NodeVisitor subclass containing methods for visiting each class in the Node hierarchy.

Visitor Pattern (problems)

```
public class WhileLoopNode extends Node {
  protected Node condition, body;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitWhileLoop(this);
  }
}
public class IfThenNode extends Node {
  protected Node condition, thenBranch;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitIfThen(this);
  }
}
```

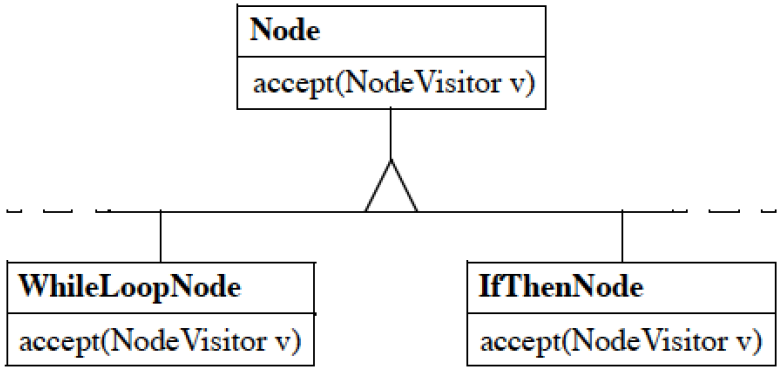



```
public abstract class NodeVisitor {
  /* ... */
  public abstract void visitWhileLoop(WhileLoopNode n);
  public abstract void visitIfThen(IfThenNode n);
}
public class TypeCheckingVisitor extends NodeVisitor {
  /* ... */
  public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
  public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

#1 stylized double-dispatching code is tedious to write and prone to error.

Visitor Pattern (problems)

```
public class WhileLoopNode extends Node {
  protected Node condition, body;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitWhileLoop(this);
  }
}
public class IfThenNode extends Node {
  protected Node condition, thenBranch;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitIfThen(this);
  }
}
```

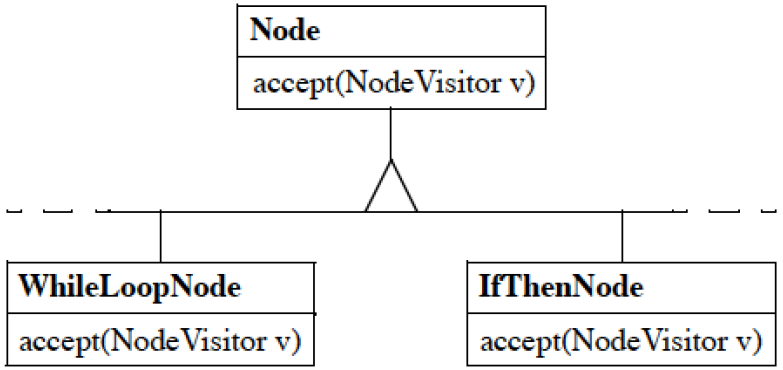



```
public abstract class NodeVisitor {
  /* ... */
  public abstract void visitWhileLoop(WhileLoopNode n);
  public abstract void visitIfThen(IfThenNode n);
}
public class TypeCheckingVisitor extends NodeVisitor {
  /* ... */
  public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
  public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

#2 the need for the Visitor pattern must be anticipated ahead of time, when the Node class is first implemented

Visitor Pattern (problems)

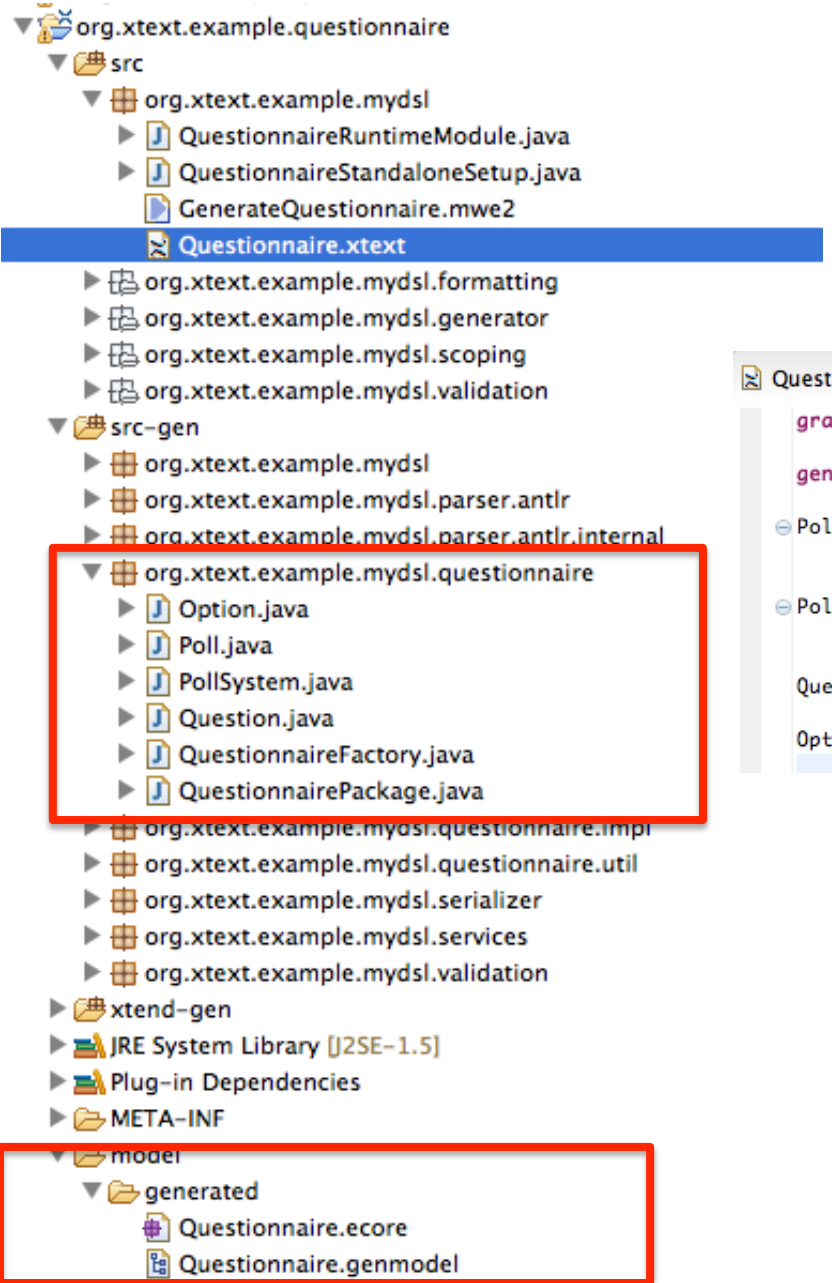
```
public class WhileLoopNode extends Node {
  protected Node condition, body;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitWhileLoop(this);
  }
}
public class IfThenNode extends Node {
  protected Node condition, thenBranch;
  /* ... */
  public void accept(NodeVisitor v) {
    v.visitIfThen(this);
  }
}
```




```
public abstract class NodeVisitor {
  /* ... */
  public abstract void visitWhileLoop(WhileLoopNode n);
  public abstract void visitIfThen(IfThenNode n);
}
public class TypeCheckingVisitor extends NodeVisitor {
  /* ... */
  public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
  public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

#3 class hierarchy evolution (e.g., new Node subclass) forces us to rewrite NodeVisitor

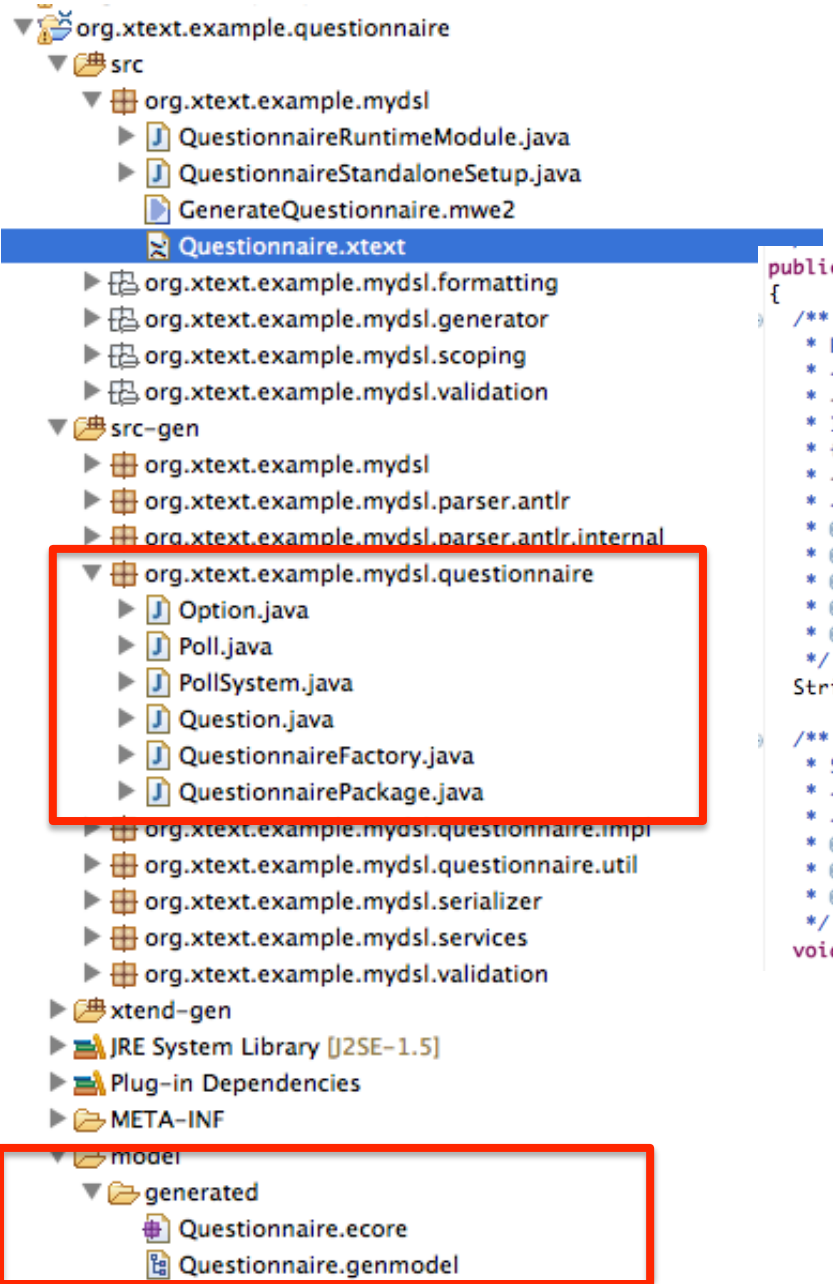
Visitor Pattern (impact of the problem)



Questionnaire.xtext

```
grammar org.xtext.example.mydsl.Questionnaire with org.eclipse.xtext.common.Terminals

generate questionnaire "http://www.xtext.org/example/mydsl/Questionnaire"

PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';

Poll:
    'Poll' name=ID '{' questions+=Question+ '}';

Question : 'Question' ID? '{' text=STRING 'options' options+=Option+ '}';

Option : id=ID ':' text=STRING ;
```

Visitor Pattern (impact of the problem)




```
public interface Question extends EObject
{
  /**
   * Returns the value of the
   * <!-- begin-user-doc -->
   * <p>
   * If the meaning of the '<er
   * there really should be mor
   * </p>
   * <!-- end-user-doc -->
   * @return the value of the '
   * @see #setText(String)
   * @see org.xtext.example.mydsl.questionnaire.QuestionnairePackage#getQuestion_Text()
   * @model
   * @generated
   */
  String getText();

  /**
   * Sets the value of the '{@link org.xtext.example.mydsl.questionnaire.Question#getText <em>
   * <!-- begin-user-doc -->
   * <!-- end-user-doc -->
   * @param value the new value of the '<em>Text</em>' attribute.
   * @see #getText()
   * @generated
   */
  void setText(String value);
```

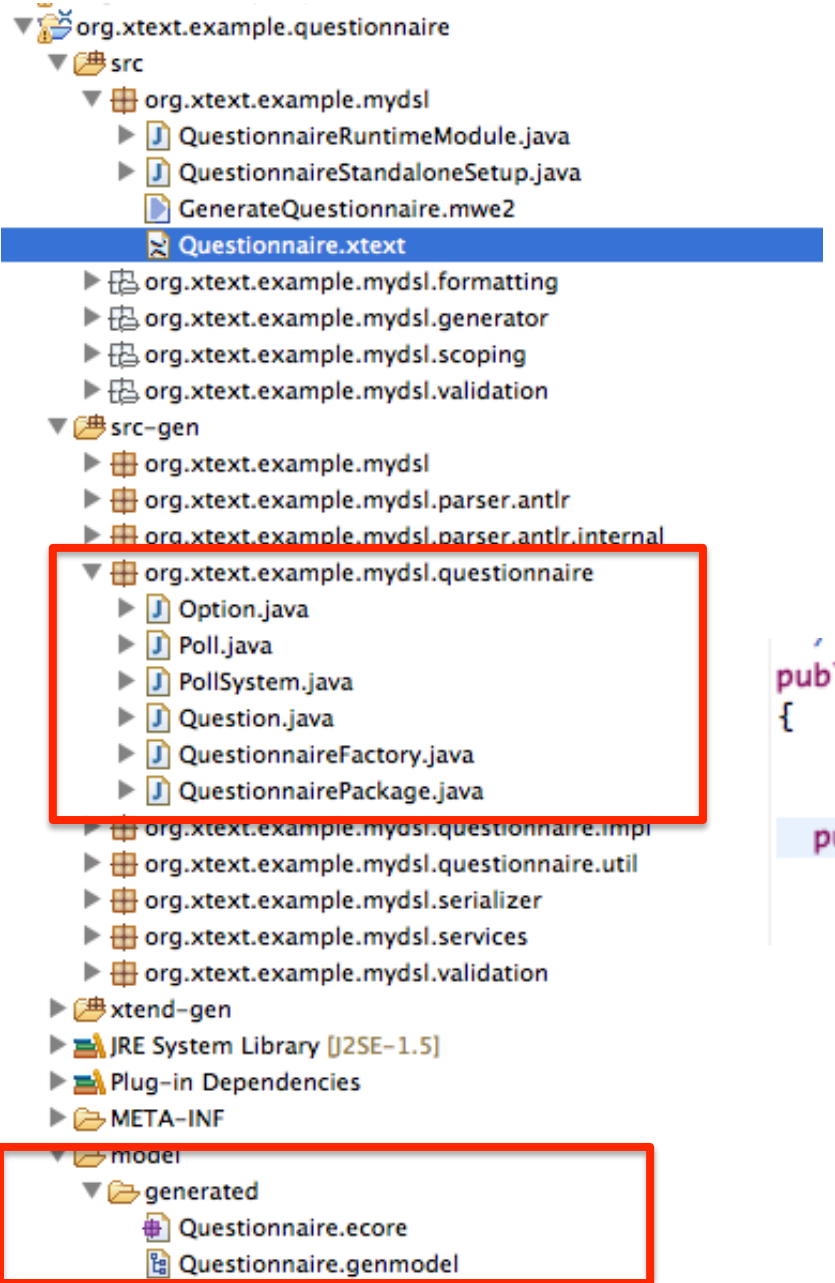
Outline

- org.xtext.example.mydsl.questionnaire
- Question
  - getText() : String
  - setText(String) : void
  - getOptions() : EList<Option>

No accept method

- platform:/resource/org.xtext.examp
  - questionnaire
    - PollSystem
      - polls : Poll
    - Poll
      - name : EString
      - questions : Question
    - Question
      - text : EString
      - options : Option
    - Option
      - id : EString
      - text : EString

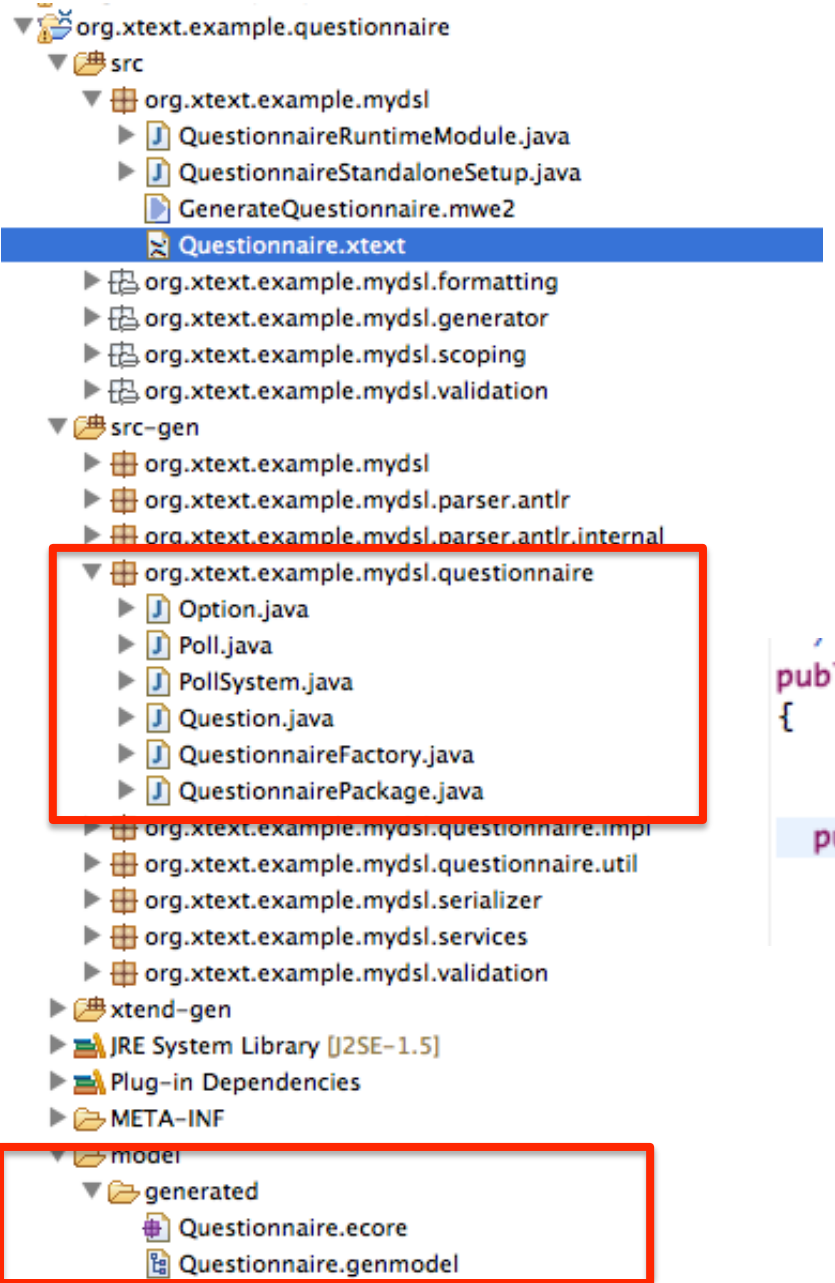
Visitor Pattern (impact of the problem)



Handcrafted code?

```
public interface Question extends EObject
{


  public void accept(QuestionnaireVisitor visi) ;
```

Visitor Pattern (impact of the problem)



⇒ **Manual**

⇒ **Some classes are not concerned by the visit…**

```
public interface Question extends EObject
{


  public void accept(QuestionnaireVisitor visi) ;
```

⇒ **If Xtext Grammar changes, you can restart again**

Visitor Pattern (requirements)

#1 stylized double-dispatching code is tedious to write and prone to error.

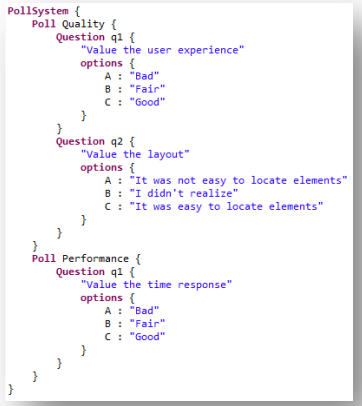
Automation

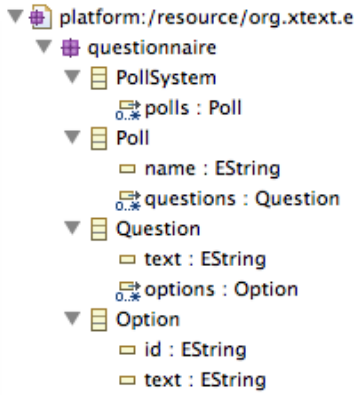
#2 the need for the Visitor pattern must be anticipated ahead of time, when the Node class is first implemented

No accept method
Violation of open/close principle: no way

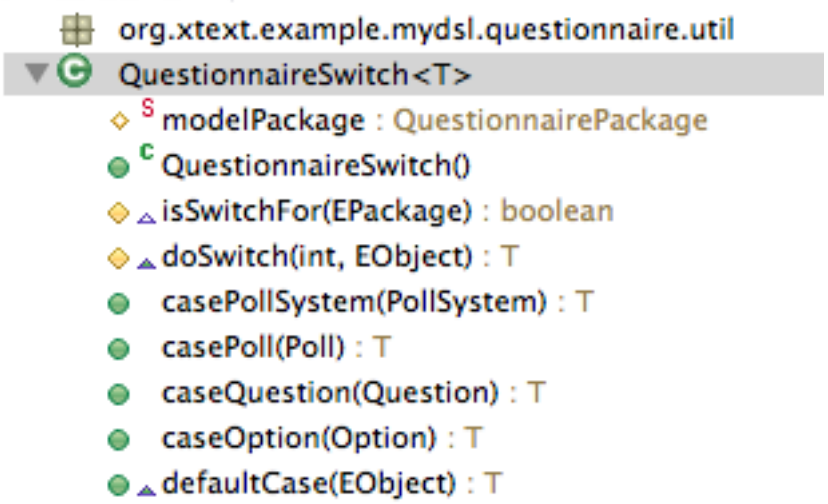
#3 class hierarchy evolution (e.g., new Node subclass) forces us to (completely) rewrite NodeVisitor

Automation

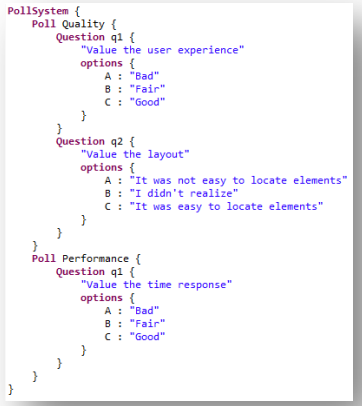


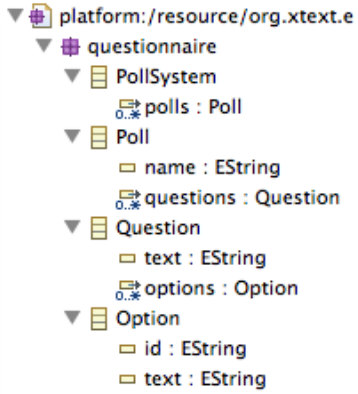


Possible solution (1): « *Switch » generated by… EMF




```
/**
 * The switch that delegates to the <code>createXXX</code> methods.
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
protected QuestionnaireSwitch<Adapter> modelSwitch =
  new QuestionnaireSwitch<Adapter>()
  {
    @Override
    public Adapter casePollSystem(PollSystem object)
    {
      return createPollSystemAdapter();
    }
    @Override
    public Adapter casePoll(Poll object)
    {
      return createPollAdapter();
    }
    @Override
    public Adapter caseQuestion(Question object)
    {
      return createQuestionAdapter();
    }
    @Override
    public Adapter caseOption(Option object)
    {
      return createOptionAdapter();
    }
    @Override
    public Adapter defaultCase(EObject object)
    {
      return createEObjectAdapter();
    }
  };
```





Possible solution (2): Extension Methods of Xtend

```
def foo(PollSystem sys, Context c) {
    // treatment
}
```

```
pollSystem.foo (new Context)
```

Context (classical with the Visitor)

Can be seen as a way
to avoid a (very) long list of
parameters and record
the « state » of the visit

@Aspect

(Active Annotations
for implementing Visitors)

```
class A {

    def boolean testReplacement() {
        return false
    }
}
```

Weaving methods

AspectA can handle a context in a proper way

```
@Aspect(className=typeof(A))
abstract class AspectA {

    def String foo() {
        return "A"
    }

    abstract def String foofoo()

}
```

```
@Test
def void testA() {
    val l = new A
    l.foofoo
}
```

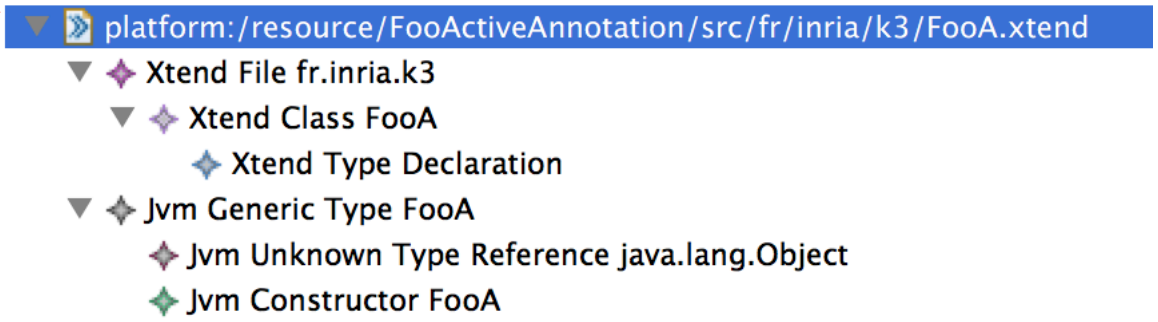
```
override def doTransform(List<? extends MutableClassDeclaration> classes, extension TransformationContext context) {


    //Method name_parameterlengths,
    val Map<MutableClassDeclaration, List<MutableClassDeclaration>> superclass = new HashMap<MutableClassDeclaration, Lis
    val Map<MutableMethodDeclaration, Set<MutableMethodDeclaration>> dispatchmethod = new HashMap<MutableMethodDeclaratic
    init_superclass(classes, context, superclass)
    init_dispatchmethod(superclass, dispatchmethod, context)



    for (clazz : classes) {


        //var List<String> inheritList1 = new ArrayList<String>() //sortByClassInheritance(clazz)


        var List<MutableClassDeclaration> listRes = sortByClassInheritance(clazz, classes,context)
        val List<String> inheritList = new ArrayList<String>()
        listRes.forEach[c|  inheritList.add(c.simpleName)]
        listResMap.put(clazz,listRes)
        //sortByClassInheritance(clazz, inheritList1,context)

        /*val StringBuffer log  = new StringBuffer
        log.append("before ")
        inheritList.forEach[ s | log.append(" " + s)]
        log.append("\n after ")
        inheritList1.forEach[ s | log.append(" " + s)]
        */
        //clazz.addError(log .toString)

        var classNam = clazz.annotations.findFirst[getValue('className') != null].getValue('className')
        //addError(clazz, classNam.class.toString)

        //var simpleNameF = classNam.eClass.EAllStructuralFeatures.findFirst[name == "simpleName"]
        //val className = classNam.eGet(simpleNameF) as String
        val className = classNam.class.getMethod("getSimpleName").invoke(classNam) as String
        //var identF = classNam.eClass.getEAllStructuralFeatures().findFirst[name == "identifier"]
        //val identifier = classNam.eGet(identF) as String
        val identifier = classNam.class.getMethod("getIdentifier").invoke(classNam) as String
        val Map<MutableMethodDeclaration, String> bodies = new HashMap<MutableMethodDeclaration, String>()

        //clazz.addError(className)
        //MOVE non static fields
        fields_processing(context, clazz, className, identifier, bodies)

        //Transform method to static
        methods_processing(clazz, context, identifier, bodies, dispatchmethod, inheritList, className)

        aspectContextMaker(context, clazz, className, identifier)

    }

}
```

https://github.com/diverse-project/k3/blob/master/core/k3/src/main/java/fr/inria/triskell/k3/Aspect.xtend

Xtend is implemented using MDE principles

```
package fr.inria.k3

class FooA {
    //
}
```

Model



http://git.eclipse.org/c/tmf/org.eclipse.xtext.git/tree/plugins/org.eclipse.xtend.core/src/org/eclipse/xtend/core/Xtend.xtext

```
grammar org.eclipse.xtend.core.Xtend with org.eclipse.xtext.xbase.annotations.XbaseWithAnnotations

import "http://www.eclipse.org/xtend"
import "http://www.eclipse.org/xtext/xbase/Xbase" as xbase
import "http://www.eclipse.org/xtext/xbase/Xtype" as xtype
import "http://www.eclipse.org/Xtext/Xbase/XAnnotations" as annotations
import "http://www.eclipse.org/xtext/common/JavaVMTypes" as types

File returns XtendFile :
    ('package' package=QualifiedName ';'?)?
        importSection=XImportSection?
        (xtendTypes+=Type)*
;

Type returns XtendTypeDeclaration :
        {XtendTypeDeclaration} annotations+=XAnnotation*
        (
                {XtendClass.annotationInfo = current}
                modifiers+=CommonModifier*
                'class' name=ValidID ('<' typeParameters+=JvmTypeParameter (',' typeParameters+=JvmTypeParameter)* '>')?
                ("extends" extends=JvmParameterizedTypeReference)?
                ('implements' implements+=JvmParameterizedTypeReference (',' implements+=JvmParameterizedTypeReference)*)?'{'
                   (members+=Member)*
                '}'
         |
                {XtendInterface.annotationInfo = current}
                modifiers+=CommonModifier*
                'interface' name=ValidID ('<' typeParameters+=JvmTypeParameter (',' typeParameters+=JvmTypeParameter)* '>')?
                ('extends' extends+=JvmParameterizedTypeReference (',' extends+=JvmParameterizedTypeReference)*)?'{'
                   (members+=Member)*
                '}'
         |
                {XtendEnum.annotationInfo = current}
                modifiers+=CommonModifier*
                'enum' name=ValidID '{'
                   (members+=XtendEnumLiteral (',' members+=XtendEnumLiteral)*)? ';'?
                '}'
         |
                {XtendAnnotationType.annotationInfo = current}
                modifiers+=CommonModifier*
                'annotation' name=ValidID '{'
                   (members+=AnnotationField)*
                '}'
        )
;
```

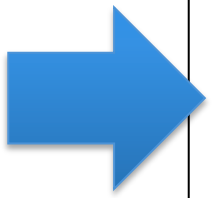
```
public class XtendCompiler extends XbaseCompiler {


@Override
public void acceptForLoop(JvmFormalParameter parameter, @Nullable XExpression expression) {
        currentAppendable = null;
        super.acceptForLoop(parameter, expression);
        if (expression == null)
                throw new IllegalArgumentException("expression may not be null");
        RichStringForLoop forLoop = (RichStringForLoop) expression.eContainer();
        forStack.add(forLoop);
        appendable.newLine();
        pushAppendable(forLoop);
        appendable.append("{").increaseIndentation();

        ITreeAppendable debugAppendable = appendable.trace(forLoop, true);
        internalToJavaStatement(expression, debugAppendable, true);
        String variableName = null;
        if (forLoop.getBefore() != null || forLoop.getSeparator() != null || forLoop.getAfter() != null) {
                variableName = debugAppendable.declareSyntheticVariable(forLoop, "_hasElements");
                debugAppendable.newLine();
                debugAppendable.append("boolean ");
                debugAppendable.append(variableName);
                debugAppendable.append(" = false;");
        }
        debugAppendable.newLine();
        debugAppendable.append("for(final ");
        JvmTypeReference paramType = getTypeProvider().getTypeForIdentifiable(parameter);
        serialize(paramType, parameter, debugAppendable);
        debugAppendable.append(" ");
        String loopParam = debugAppendable.declareVariable(parameter, parameter.getName());
        debugAppendable.append(loopParam);
        debugAppendable.append(" : ");
        internalToJavaExpression(expression, debugAppendable);
        debugAppendable.append(") {").increaseIndentation();
}
```

Xtend to Java

HelloWorld.xtend | HelloWorld.java

```
1  package fr.inria.k3;
2
3  import org.eclipse.xtext.xbase.lib.InputOutput;
4
5  @SuppressWarnings("all")
6  public class HelloWorld {
7    public static void main(final String[] args) {
8      InputOutput.<String>println("HW");
9    }
10 }
11
```

```
package fr.inria.k3

@Singleton
class GUIWindow {


    int x ;
    int y ;

}
```



```
public final class GUIWindow {
  private GUIWindow() {
    // singleton
  }

  private int x;

  private int y;

  private final static GUIWindow INSTANCE = new GUIWindow();

  public static GUIWindow getINSTANCE() {
    return INSTANCE;
  }
}
```

#1 Model Transformations

(importance, taxonomy, and some techniques -- templates, visitors, annotation processors)

#2 Xtend

(A general purpose language with advanced features and an illustration on how to transform models in practice)

http://docs.oracle.com/javase/6/docs/technotes/guides/language/annotations.html

http://docs.oracle.com/javase/tutorial/java/annotations/

http://techblog.troyweb.com/index.php/2012/05/switching-annotation-preprocessors-for-jpa-meta-model-generation-in-eclipse/

http://blog.jonasbandi.net/2011/09/using-jpa-metamodel-annotation.html

http://mojo.codehaus.org/apt-maven-plugin/