

Model Management

The Art Of Domain-Specific Languages
Let's Hack Our Own Languages!

Plan

- Model Management in a nutshell
 - Loading, serializing, transforming models: scenarios
 - Taxonomy
 - Model transformation in Xtend
- Xtend: a case study for GPL/DSL, MDE, and model transformation
 - Advanced features: extension methods, active annotations, template expressions
 - Xtend: behind the magic (Xtext+MDE)
 - Xtend + Xtext (breathing life into DSLs)
 - @Aspect annotation

Contract

- Practical foundations of model management
- Model transformations
 - Model-to-Text
 - Model-to-Model
 - Metaprogramming
- DSLs and model management: all together (Xtext + Xtend)

Model Transformations

Taxonomy + Examples

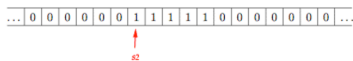
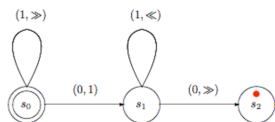
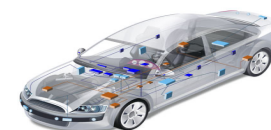
Abstraction Gap

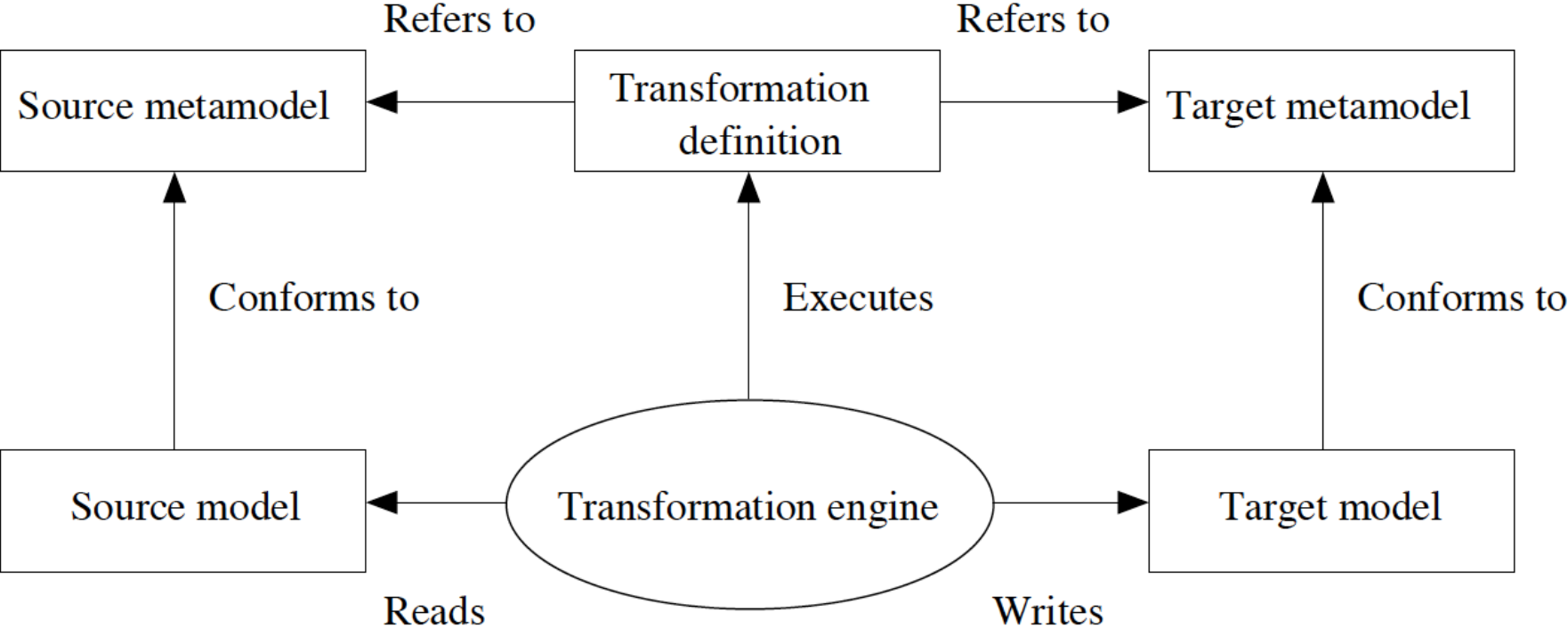
Transformation is the key

Problem space
domain-specific
language

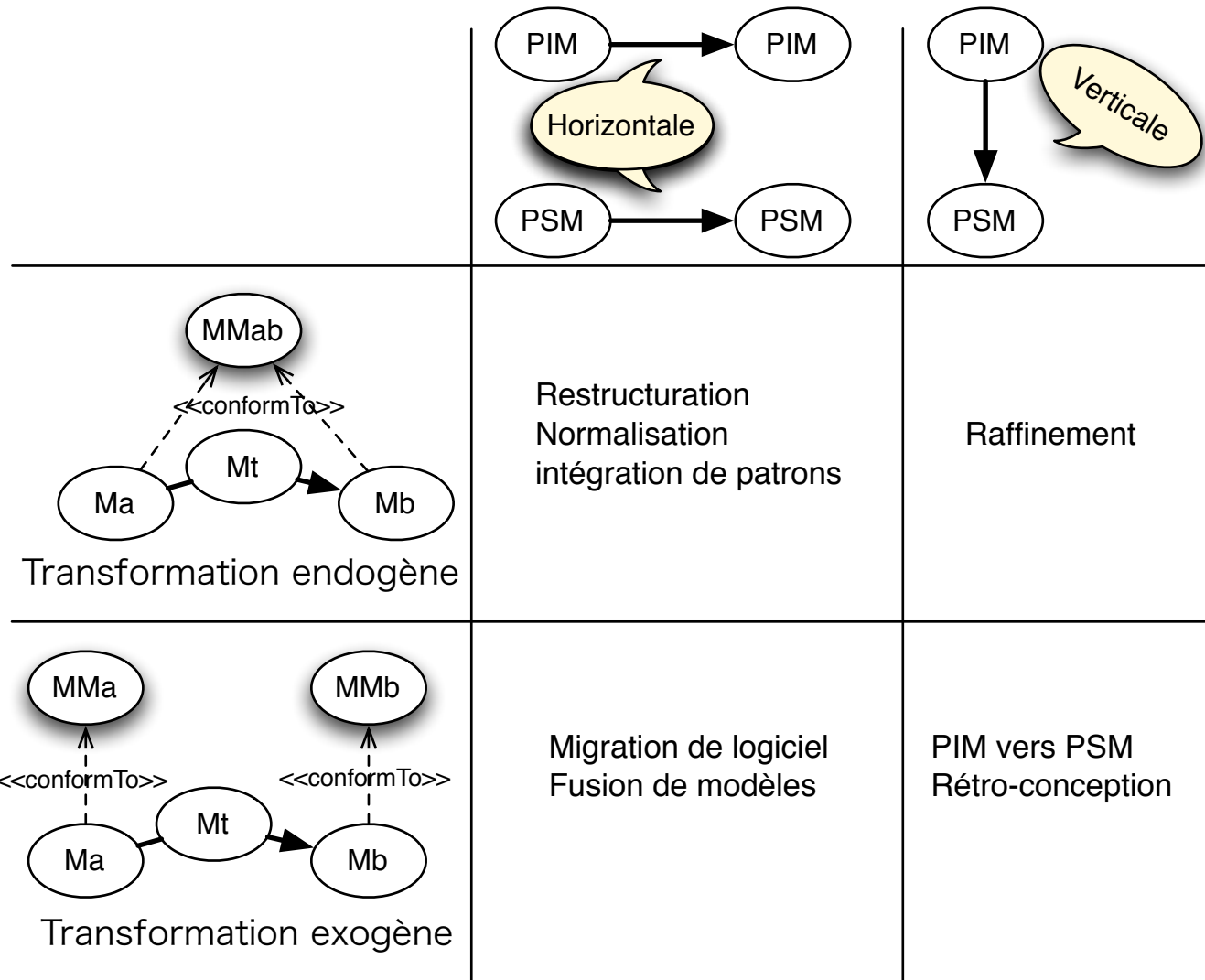
Transformation

Solution space
implementation
language





Model Transformation: Taxonomy

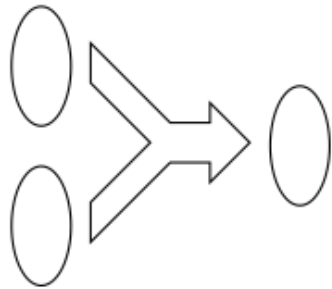


Overview of Generative Software Development

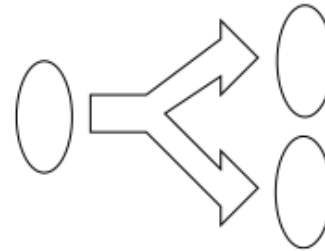
Krzysztof Czarnecki



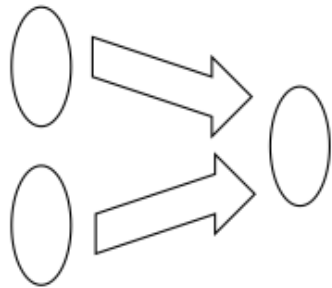
a. Chaining of mappings



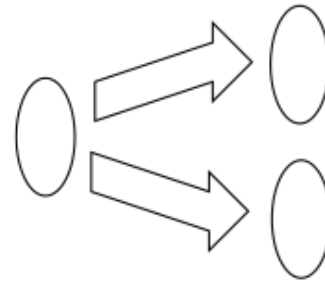
b. Multiple problem spaces



c. Multiple solution spaces



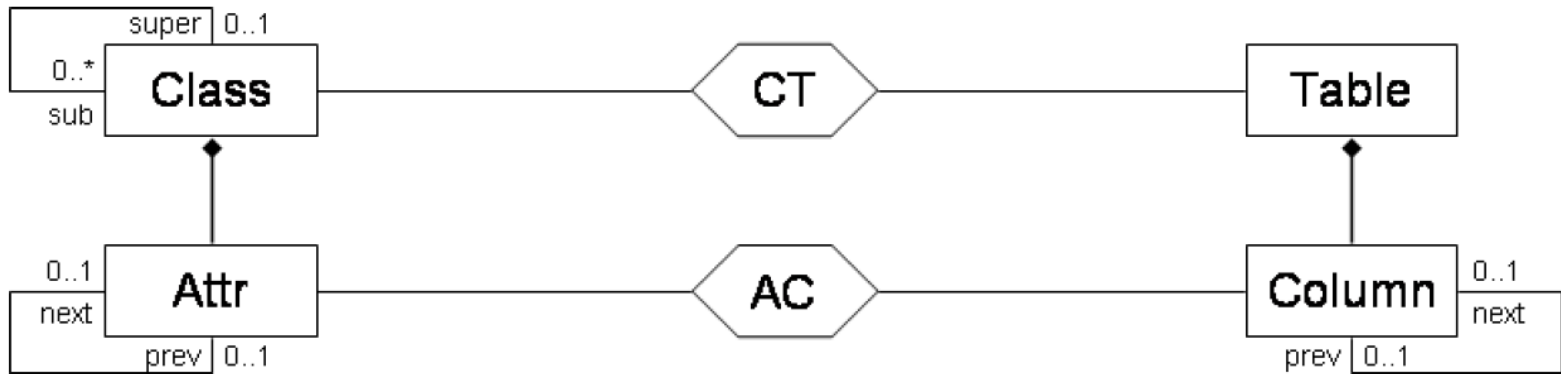
d. Alternative problem spaces



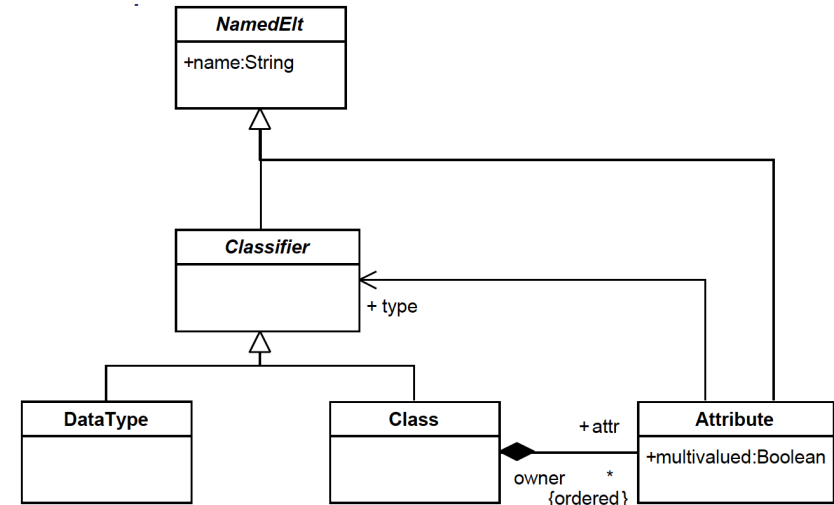
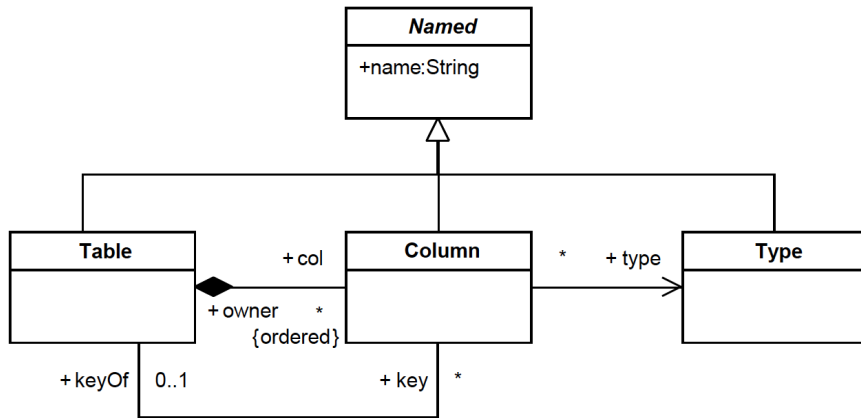
e. Alternative solution spaces

Andy Schürr, Felix Klar “15 Years of Triple Graph Grammars.” ICGT 2008

(declarative; bi-directional; model-to-model)



ATL (<http://www.eclipse.org/atl/atlTransformations/>)



```
rule Class2Table {
```

```
  from
```

```
    c : Class!Class
```

```
  to
```

```
    t : Relational!Table
```

```
}
```

-- source pattern

-- target pattern

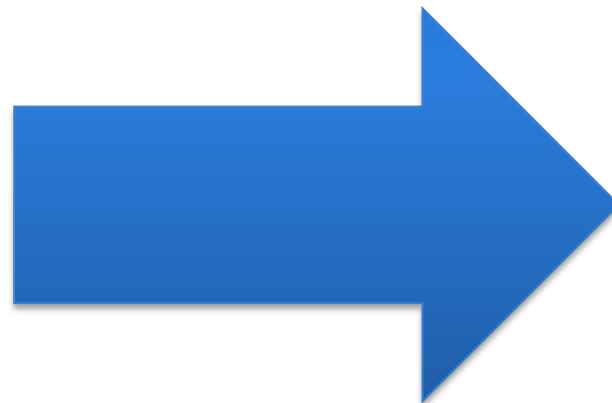
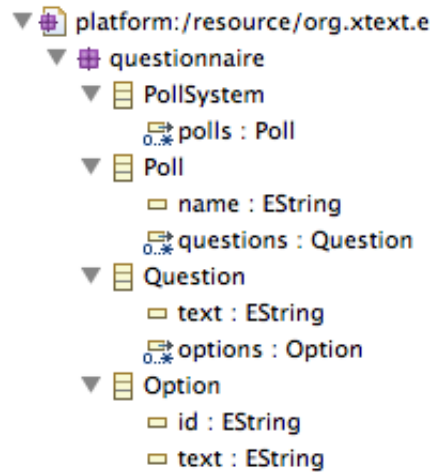
Quizz Time

Characterize the following model transformations

Endogeneous? Exogeneous?

Vertical? Horizontal?

Model-to-text? Model-to-Model?



```

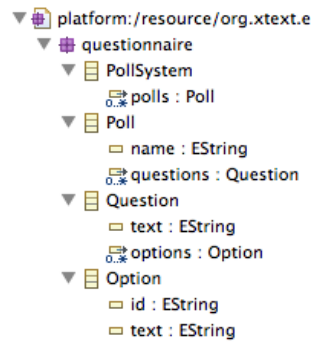
foo1.q
PollSystem {
  Poll poll1 {
    Question A {
      "What is A ?"
      options
        b : "B"
        c : "C"
        d : "D"
    }
  }
  Poll poll2 {
    Question D {
      "What is D ?"
      options
        e : "E"
        f : "F"
    }
  }
}

```

```

foo1.q  foo2.q
PollSystem {
  Poll poll1_poll {
    Question {
      "What is A ?"
      options
        b : "B"
        c : "C"
        d : "D"
    }
  }
  Poll poll2_poll {
    Question {
      "What is D ?"
      options
        e : "E"
        f : "F"
    }
  }
}

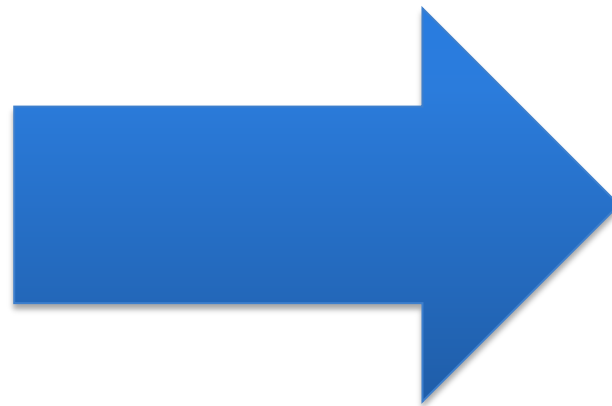
```



```

fool.q
PollSystem {
  Poll poll1 {
    Question A {
      "What is A ?"
      options
        b : "B"
        c : "C"
        d : "D"
    }
  }
  Poll poll2 {
    Question D {
      "What is D ?"
      options
        e : "E"
        f : "F"
    }
  }
}

```



poll1

What is A ?

- B
- C
- D

poll2

What is D ?

- E
- F

Model Transformation in Java/Xtend

Effective Model Management

- How to load/serialize a model?
- How to visit, analyze and transform models?
- You can do it in Java (EMF API)

- We arbitrarily choose  **xtend**
– Java 10, interesting « features »  **xtext**
– Integration within Eclipse ecosystem (incl. Xtext) and facilities to manage models
– An example of a sophisticated language

Before going into details of Xtend...

- Recap of the scenarios
 - Text-to-Model
 - Model(s)-to-Model transformation
 - Metamodels as a « bridge » between technologies
 - Model-to-Text
- The solution of some of the « scenarios »
 - Just to give an overview of Xtend capabilities
 - To give a more practical/concrete view of some of the previous scenarios


```
def loadPollSystem(URI uri) {
  new QuestionnaireStandaloneSetupGenerated().createInjectorAndDoEMFRegistration()
  var res = new ResourceSetImpl().getResource(uri, true);
  res.contents.get(0) as PollSystem
}
```

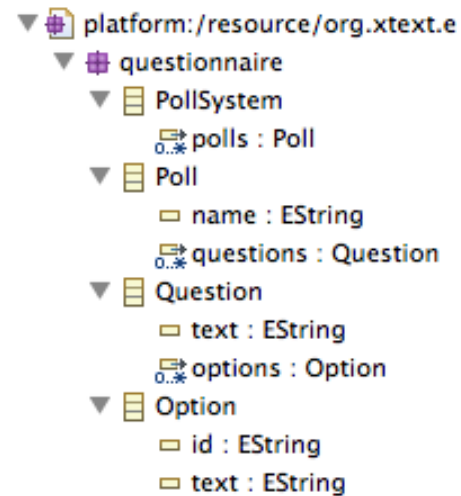
```
def savePollSystem(URI uri, PollSystem polls) {
  var Resource rs = new ResourceSetImpl().createResource(uri);
  rs.getContents.add(polls);
  rs.save(new HashMap());
}
```

```
@Test
def test1() {

  // loading
  var polls = loadPollSystem(URI.createURI("foo1.q"))
  assertNotNull(polls)
  assertEquals(2, polls.polls.size)

  // MODEL MANAGEMENT (ANALYSIS, TRANSFORMATION)
  polls.polls.forEach[p | p.name = p.name + "_poll"]

  // serializing
  savePollSystem(URI.createURI("foo2.q"), polls)
}
```



```

@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

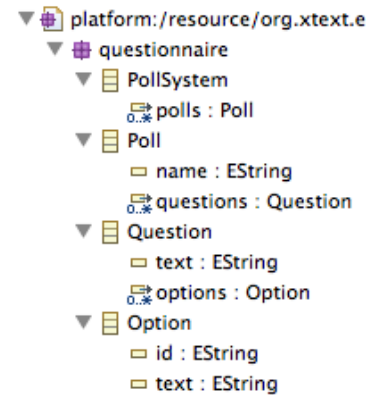
}

```

```

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>
'''

```



poll1

What is A ?

- B
- C
- D

```

foo1.q
PollSystem {
  Poll poll1 {
    Question A {
      "What is A ?"
      options
        b : "B"
        c : "C"
        d : "D"
    }
  }
  Poll poll2 {
    Question D {
      "What is D ?"
      options
        e : "E"
        f : "F"
    }
  }
}

```



poll2

What is D ?

- E
- F

Plan

- Model Management in a nutshell
 - Loading, serializing, transforming models: scenarios
 - Taxonomy
 - Model transformation in Xtend
- Xtend: a case study for GPL/DSL, MDE, and model transformation
 - Advanced features: extension methods, active annotations, template expressions
 - Xtend: behind the magic (Xtext+MDE)
 - Xtend + Xtext (breathing life into DSLs)
 - @Aspect annotation

Contract

- Practical foundations of model management
- Model transformations
 - Model-to-Text
 - Model-to-Model
 - Metaprogramming
- DSLs and model management: all together (Xtext + Xtend)

Xtend

A pragmatic general-
purpose language
implemented using MDE
techniques

Hello World

```
HelloWorld.xtend ✖
1 package fr.inria.k3
2
3 class HelloWorld {
4
5     def static void main(String[] args) {
6         println("HW")
7     }
8
9 }
10 // HW
```

Semi-colon is optional (within class, methods, etc.)

```
1 package fr.inria.k3.fields
2
3 class MyFielder {
4
5     int count = 1
6     static boolean debug = false
7     var name = 'Foo' // type String is inferred
8     val UNIVERSAL_ANSWER = 42 // final field with inferred type int
9     // ...
10    public int count2 = 2 ;
11
12 }
13
```

Package Declaration

HelloWorld.xtend

```
1 package fr.inria.k3
2
3 class HelloWorld {
4
5     def static void main(String[] args) {
6         println("HW")
7     }
8
9 }
10 // HW
```

Semi-colon ‘;’ is optional

HelloWorld.xtend

HelloWorld.java

PackageExploder.xtend

```
1 package fr.inria.k3.^def
2
3 class PackageExploder {
4
5 }
```

‘^’ for avoiding keyword conflicts

Methods

By default:
visibility
conditions set
to public

```
1 package fr.inria.k3.methods
2
3 class FooMethod {
4
5     def foo1() {
6         "A"
7     }
8
9     def foo2() {
10        6 + 3
11    }
12
13    def private foo3() {
14        6 + 3
15    }
16
17    def public foo4() {
18        foo3() * 8
19    }
20 }
21
22 class FooMethodUses {
23
24     def fooUse() {
25         new FooMethod().foo1
26         new FooMethod().foo3()
27
28
29
30         new FooMethod().f
31
32
33     }
34
35 }
```

- foo1 : String - FooMethod.foo1()
- foo2 : int - FooMethod.foo2()
- foo4 : int - FooMethod.foo4()

```
HelloWorld.xtend
1 package fr.inria.k3
2
3 class HelloWorld {
4
5     def static void main(String[] args) {
6         println("HW")
7     }
8
9 }
10 // HW
```

Methods

```
1 package fr.inria.k3.methods
2
3 class FooMethod {
4
5     def foo1() {
6         "A"
7     }
8
9     def foo2() {
10        6 + 3
11    }
12
13    def private foo3() {
14        6 + 3
15    }
16
17    def public foo4() {
18        foo3() * 8
19    }
20 }
21
22 class FooMethodUses {
23
24     def fooUse() {
25         new FooMethod().foo1
26         new FooMethod().foo3()
27
28
29
30         new FooMethod().f
31
32
33     }
34
35 }
```

Type inference (return type)

Outline

- fr.inria.k3.methods
 - FooMethod
 - foo1() : String
 - foo2() : int
 - foo3() : int
 - foo4() : int
 - FooMethodUses
 - fooUse() : Object

Method Calling

You can omit parentheses

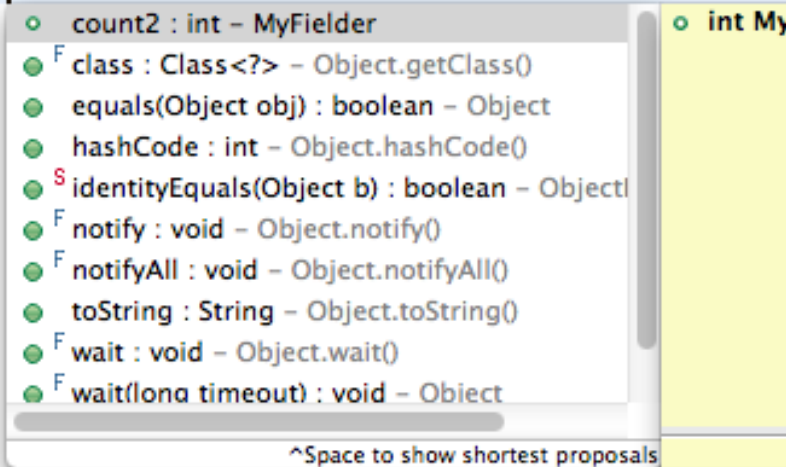
```
33 def fooUse2() {  
34     3.toString  
35     "4".length  
36     new FooMethod().foo1  
37     new FooMethod().foo1()  
38     new FooMethod().foo1 + 3.toString  
39  
40 }
```

Outline

- fr.inria.k3.methods
 - FooMethod
 - foo1() : String
 - foo2() : int
 - foo3() : int
 - foo4() : int
 - FooMethodUses
 - fooUse() : int
 - fooUse2() : String

Fields

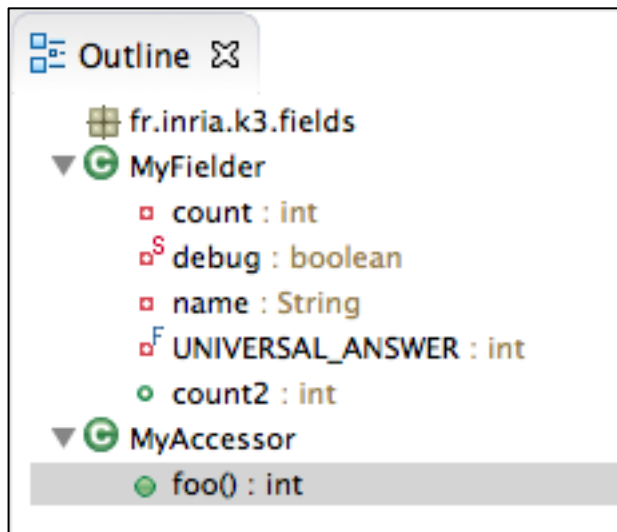
```
1 package fr.inria.k3.fields
2
3 class MyFielder {
4
5     int count = 1
6     static boolean debug = false
7     var name = 'Foo' // type String is inferred
8     val UNIVERSAL_ANSWER = 42 // final field with inferred type int
9     // ...
10    public int count2 = 2 ;
11
12 }
13
14 class MyAccessor {
15
16     def foo() {
17         new MyFielder().
18     }
19 }
20
```



**By default:
visibility
conditions set
to private**

Fields

```
1 package fr.inria.k3.fields
2
3 class MyFielder {
4
5     int count = 1
6     static boolean debug = false
7     var name = 'Foo' // type String is inferred
8     val UNIVERSAL_ANSWER = 42 // final field with inferred type int
9     // ...
10    public int count2 = 2 ;
11
12 }
13
```



primitive types of Java (int, boolean, etc) with autoboxing

var: type inference

val: constant, « final » in Java

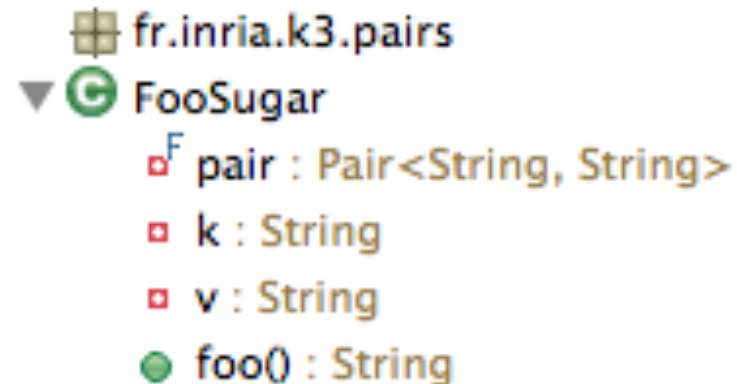
Static Methods (::)

```
1 package fr.inria.k3.stat
2
3 import java.util.Collections
4
5 class FooStati {
6
7
8     static var colors = newList(46, 76, 89, 53)
9
10
11     def static void main(String... args) {
12         println("B " + colors)
13         Collections::sort(colors)
14         println("A " + colors)
15
16         colors.add(45)
17         println("A " + colors)
18
19     }
20 }
```

```
B [46, 76, 89, 53]
A [46, 53, 76, 89]
A [46, 53, 76, 89, 45]
```

Pairs

```
1 package fr.inria.k3.pairs
2
3 class FooSugar {
4
5     // syntactic sugar
6     val pair = "spain" -> "italy"
7     var k = pair.key
8     var v = pair.value
9
10    def foo() {
11
12        println("key=" + k + " value=" + v)
13
14    }
15 }
```



Pairs

```
1 package fr.inria.k3.pairs
2
3 class FooSugar {
4
5
6     static val greetings = newHashMap(
7         "german" -> "Hallo",
8         "english" -> "Hello",
9         "french" -> "Bonjour"
10    )
11
12    def static main(String... args) {
13        greetings.forEach[key, value | println("HW in " + key + " : " + value)]
14    }
15 }
16 }
```

Outline

- fr.inria.k3.pairs
 - FooSugar
 - greetings : HashMap<String, String>
 - main(String[]) : void

Problems @ Javadoc Declaration Console

```
<terminated> FooSugar [Java Application] /Library/Java/JavaVirtualMa
HW in english : Hello
HW in french : Bonjour
HW in german : Hallo
```


Immutable data structure

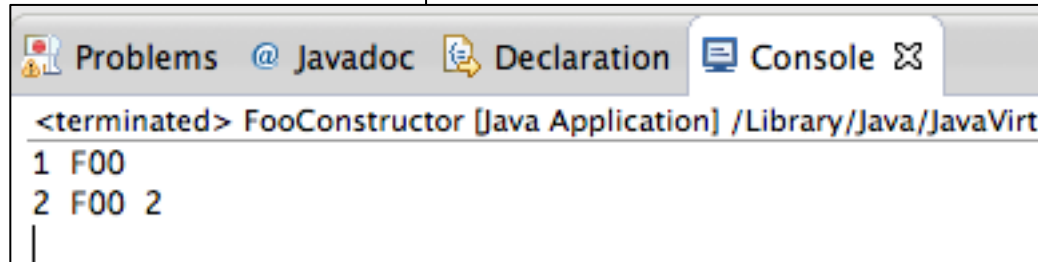
```
1 package fr.inria.k3.stat
2
3 import java.util.Collections
4
5 class FooStati {
6
7
8     static var colors = #[46, 76, 89, 53] // newList(46, 76, 89, 53)
9
10
11     def static void main(String... args) {
12         println("B " + colors)
13         Collections::sort(colors)
14         println("A " + colors)
15
16         colors.add(45)
17         println("A " + colors)
18     }
19 }
20 }
```

```
<terminated> FooStati [Java Application] /Library/Java/JavaVirtualMachines/jdk1.7.0_13.jdk/Contents/
B [46, 76, 89, 53]
Exception in thread "main" java.lang.UnsupportedOperationException
    at java.util.Collections$UnmodifiableList$1.set(Collections.java:1244)
    at java.util.Collections.sort(Collections.java:159)
    at fr.inria.k3.stat.FooStati.main(FooStati.java:15)
```

Constructor

Default visibility: public

```
1 package fr.inria.k3.classes
2
3 class FooConstructor {
4
5     var String l
6
7     new() {
8         this("F00")
9     }
10
11     new (String v) {
12         l = v
13     }
14
15
16     override toString() {
17         l
18     }
19
20     def static void main (String... args) {
21         println("1 " + new FooConstructor())
22         println("2 " + new FooConstructor("F00 2"))
23     }
24 }
```



```
<terminated> FooConstructor [Java Application] /Library/Java/JavaVirt
1 F00
2 F00 2
|
```

override keyword:
mandatory

Cast and Type

```
1 package fr.inria.k3.types
2
3 class FooTypes {
4
5     static val Object obj = "a string"
6     // static val String s = obj
7     static val String s = obj as String // cast
8
9     def static void main(String... args) {
10         println(typeof(String) + "") // String.class
11         println("\t" + s + "\n")
12
13     }
14 }
```

Extension Methods...

« ... allow to add new methods to existing types without modifying them. »

```
def removeVowels (String s){  
    s.replaceAll("[aeiouAEIOU]", "")  
}
```

We can call this method either like in Java:

```
removeVowels("Hello")
```

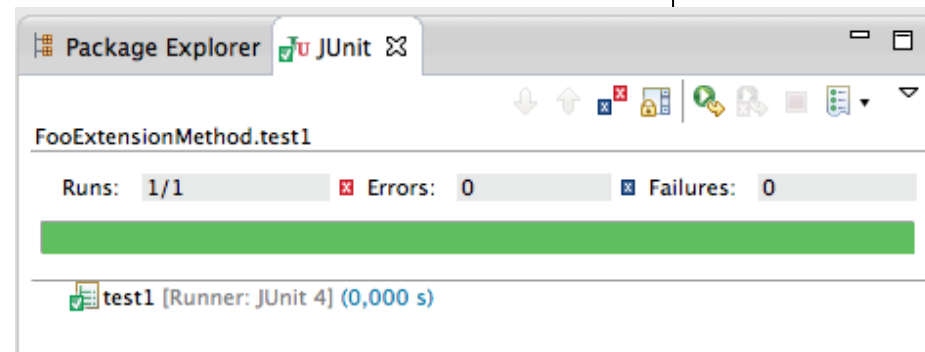
or as an extension method of String:

```
"Hello".removeVowels
```

The first parameter of a method can either be passed in after opening the parentheses or before the method call

Extension Method (local)

```
1 package fr.inria.k3.methods
2
3 import org.junit.Test
4 import static org.junit.Assert.*
5
6 class FooExtensionMethod {
7
8
9     def removeVowels (String s){
10         s.replaceAll("[aeiouAEIOU]", "")
11     }
12
13     def foo() {
14         "HelloWorld".removeVowels
15     }
16
17     @Test
18     def test1() {
19         assertEquals("HllWrld", new FooExtensionMethod().foo)
20     }
21 }
```



Extension Method (library)

```
1 package fr.inria.k3.methods
2
3 import org.junit.Test
4
5 import static org.junit.Assert.*
6
7 class FooExtensionLibrary {
8
9     var listOfStrings = #["A", "b", "c", "D", "e"]
10
11     def foo() {
12         var String h = "hello".toFirstUpper // calls StringExtensions.toFirstUpper(String)
13         listOfStrings.map[ toUpperCase ] // calls ListExtensions.<T, R>map(List<T> list, Function<? super T, ? extends R> mapFunction)
14     }
15
16
17     @Test
18     def test1() {
19         assertEquals("C", new FooExtensionLibrary().foo.get(2))
20     }
21 }
```

```
/**
 * Returns a list that performs the given {@code transformation} for each element of {@code original} when
 * requested. The mapping is done lazily. That is, subsequent iterations of the elements in the list will
 * repeatedly apply the transformation. The returned list is a transformed view of {@code original}; changes to
 * {@code original} will be reflected in the returned list and vice versa (e.g. invocations of {@link List#remove(int)}).
 *
 *
 * @param original
 *     the original list. May not be null.
 * @param transformation
 *     the transformation. May not be null.
 * @return a list that effectively contains the results of the transformation. Never null.
 */
@Pure
public static <T, R> List<R> map(List<T> original, Function1<? super T, ? extends R> transformation) {
    return Lists.transform(original, new FunctionDelegate<T, R>(transformation));
}
```

```
public class ListExtensions {
```

Extension Method (library)

```
1 package fr.inria.k3.methods
2
3 import org.junit.Test
4
5 import static org.junit.Assert.*
6
7 class FooExtensionLibrary {
8
9     var listOfStrings = #["A", "b", "c", "D", "e"]
10
11     def foo() {
12         var String h = "hello".toFirstUpper // calls StringExtensions.toFirstUpper(String)
13         listOfStrings.map[ toUpperCase ] // calls ListExtensions.<T, R>map(List<T> list, Function<? super T, ? extends R> mapFunction)
14     }
15
16
17     @Test
18     def test1() {
19         assertEquals("C", new FooExtensionLibrary().foo.get(2))
20     }
21 }
```

```
/**
 * Returns the {@link String} {@code s} with an {@link Character#isUpperCase(char) upper case} first character. This
 * function is null-safe.
 *
 * @param s
 *         the string that should get an upper case first character. May be null.
 * @return the {@link String} {@code s} with an upper case first character or null if the input
 *         {@link String} {@code s} was null.
 */
@Pure
public static String toFirstUpper(String s) {
    if (s == null || s.length() == 0)
        return s;
    if (Character.isUpperCase(s.charAt(0)))
        return s;
    if (s.length() == 1)
        return s.toUpperCase();
    return s.substring(0, 1).toUpperCase() + s.substring(1);
}
```

```
public class StringExtensions {
```

Lambda Expression (Java 8 will support it)

Anonymous classes can be found everywhere in Java code...

```
1. // Java Code!  
2. final JTextField textField = new JTextField();  
3. textField.addActionListener(new ActionListener() {  
4.     @Override  
5.     public void actionPerformed(ActionEvent e) {  
6.         textField.setText("Something happened!");  
7.     }  
8. });
```

... And have always been the poor-man's replacement for lambda expressions in Java.

Lambda Expression (Xtend answer)

Anonymous classes can be found everywhere in Java code...

```
1. // Java Code!  
2. final JTextField textField = new JTextField();  
3. textField.addActionListener(new ActionListener() {  
4.     @Override  
5.     public void actionPerformed(ActionEvent e) {  
6.         textField.setText("Something happened!");  
7.     }  
8. });
```

No need to
specify the
type for e

```
1.     textField.addActionListener([ e |  
2.         textField.setText = "Something happened!"  
3.     ])
```

You can even
ommit e

```
1.     textField.addActionListener([  
2.         textField.setText = "Something happened!"  
3.     ])
```

Lambda Expression

(Xtend answer, more impressive examples)

```
1. Collections.sort(someStrings) [ a, b |  
2.   a.length - b.length  
3. ]
```

Java 8: `shapes.forEach(s -> { s.setColor(RED); });`

Xtend: `shapes.forEach[color = RED]`

Java 8: `shapes.stream()
 .filter(s -> s.getColor() == BLUE)
 .forEach(s -> { s.setColor(RED); });`

Xtend: `shapes.stream
 .filter[color == BLUE]
 .forEach[color = RED]`

Lambda Expression

(Xtend answer, more impressive examples)

```
class FooLambda {  
  
    val l = [String s | s.length]  
    var (String)=>int l2 = [it.length] // it is a keyword for referring to the first parameter  
    var (String)=>int l3 = [length] // we can even omit it or the first parameter  
  
    @Test  
    def test1() {  
        assertEquals(l.apply("RRRR"), l2.apply("PPPP"))  
        assertEquals(l2.apply("RRRR"), l3.apply("PPPP"))  
    }  
}
```

```
▼ FooLambda  
  F l : Function1<String, Integer>  
  l2 : (String)=>int  
  l3 : (String)=>int  
  ● test1() : void
```

Templates

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

Templates (2)

```
@Test
def test2() {

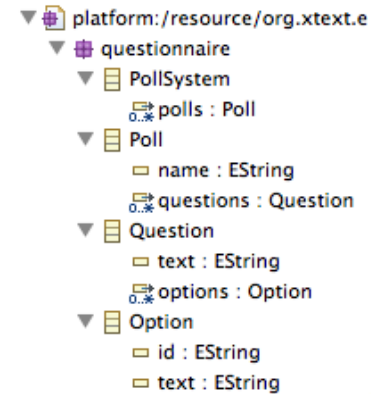
// loading
var pollS = loadPollSystem(URI.createURI("foo1.q"))

// 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>
'''
```



poll1

What is A ?

- B
- C
- D

poll2

What is D ?

- E
- F

```
foo1.q
PollSystem {
  Poll poll1 {
    Question A {
      "What is A ?"
      options
      b : "B"
      c : "C"
      d : "D"
    }
  }
  Poll poll2 {
    Question D {
      "What is D ?"
      options
      e : "E"
      f : "F"
    }
  }
}
```

Templates (3)

- You already experiment with web templating engines (JSP, Scala templates in Play!, Symfony templates, etc.)

```
<h1>Exemple de page JSP</h1>
<!-- Impression de variables -->
<p>Au moment de l'exécution de ce script, nous sommes le <%= date %>.</p>
<p>Cette page a été affichée <%= nombreVisites %> fois !</p>
</body>
</html>
```

- Alternatives exist in the modeling world
 - Multiple pre-defined and customizables generators



- Xtend: seamless integration into a general purpose language


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
```

Xtend to Java (2)

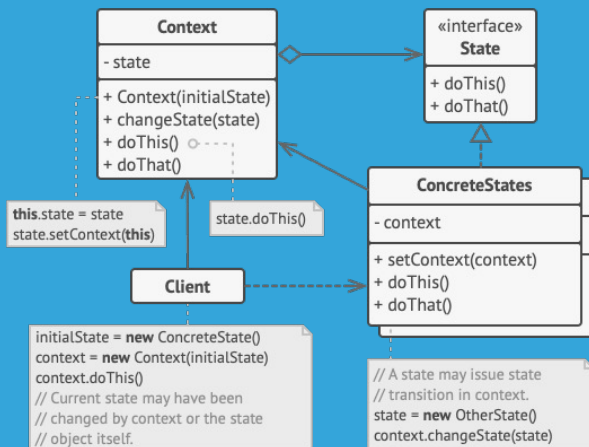
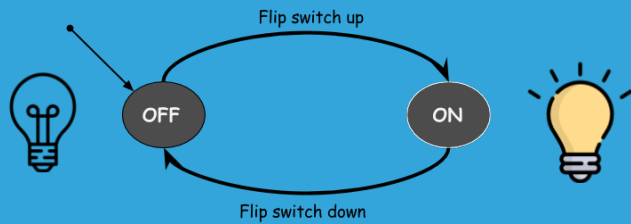
more after

```
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 org.eclipse.xtext.xbase.lib;  
import com.google.common.annotations.GwtCompatible;  
/**  
 * Utilities to print information to the console.  
 *  
 * @author Sven Efftinge - Initial contribution and API  
 */  
@GwtCompatible public class InputOutput {  
    /**  
     * Prints a newline to standard out, by delegating directly to System.out.println()  
     * @since 2.3  
     */  
    public static void println() {  
        System.out.println();  
    }  
    /**  
     * Prints the given {@code object} to System.out and terminate the line. Useful to log partial  
     * expressions to trap errors, e.g. the following is possible: println(1 + println(2)) + 3  
     *  
     * @param object  
     *         the to-be-printed object  
     * @return the printed object.  
     */  
    public static <T> T println(T object) {  
        System.out.println(object);  
        return object;  
    }  
}
```


Part 3: define a compiler from your language to Java



KEEP
CALM

AND

DO IT
YOURSELF

Xtend/Xtext

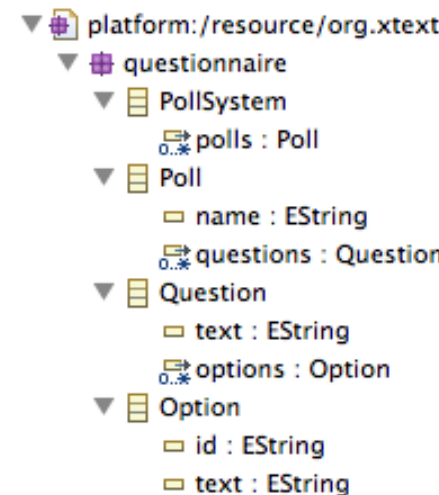
Back to our scenarios



```
def loadPollSystem(URI uri) {  
    new QuestionnaireStandaloneSetupGenerated().createInjectorAndDoEMFRegistration()  
    var res = new ResourceSetImpl().getResource(uri, true);  
    res.contents.get(0) as PollSystem  
}
```

```
def savePollSystem(URI uri, PollSystem polls) {  
    var Resource rs = new ResourceSetImpl().createResource(uri);  
    rs.getContents().add(polls);  
    rs.save(new HashMap());  
}
```

```
@Test  
def test1() {  
  
    // loading  
    var polls = loadPollSystem(URI.createURI("foo1.q"))  
    assertNotNull(polls)  
    assertEquals(2, polls.polls.size)  
  
    // MODEL MANAGEMENT (ANALYSIS, TRANSFORMATION)  
    polls.polls.forEach[p | p.name = p.name + "_poll"]  
  
    // serializing  
    savePollSystem(URI.createURI("foo2.q"), polls)  
}
```



Templates

```
1 package fr.inria.k3.templates
2
3 import org.junit.Test
4 import static org.junit.Assert.*
5
6 class FooTempl {
7
8     def someHTML(String content) '''<html><body>«content»</body></html>'''
9
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

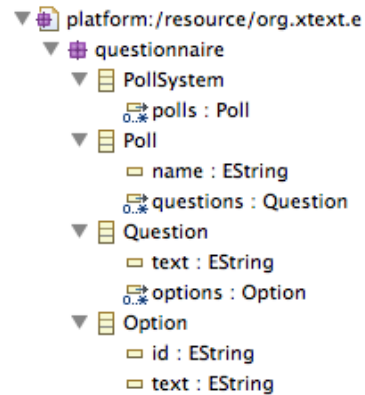
}

```

```

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>
'''

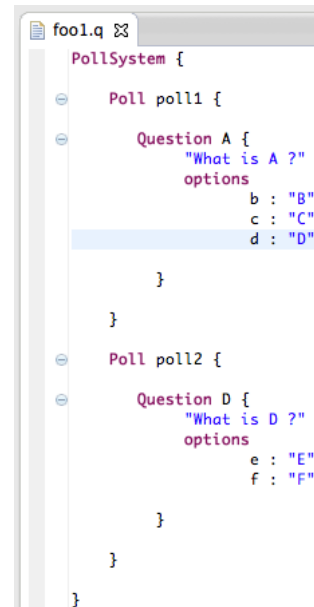
```



poll1

What is A ?

- B
- C
- D



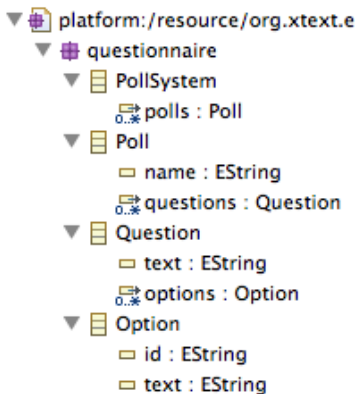
poll2

What is D ?

- E
- F

Facilities to create objects in a programmatic way

xtext



Ecore Model

EPackage
EClass
EAttribute
EReference

Code generation



Java Code

Package
Class
Attribute
Reference

```
@Test
```

```
def test2() {
```

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

Xtend is
implemented using
MDE principles

```
package fr.inria.k3
```

```
class FooA {
```

```
  //
```

```
}
```

Model

platform:/resource/FooActiveAnnotation/src/fr/inria/k3/FooA.xtend

- ▼ Xtend File fr.inria.k3
 - ▼ Xtend Class FooA
 - ◆ Xtend Type Declaration
 - ▼ Jvm Generic Type FooA
 - ◆ Jvm Unknown Type Reference java.lang.Object
 - ◆ Jvm Constructor FooA

<https://github.com/eclipse/xtend/blob/master/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/xtend/xbase/Xbase" as xbase
import "http://www.eclipse.org/xtend/xbase/Xtype" as xtype
import "http://www.eclipse.org/xtend/xbase/XAnnotations" as annotations
import "http://www.eclipse.org/xtend/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)*
    '}'
  )
;
```

The logo for 'xtend' features the word 'xtend' in a bold, black, sans-serif font. The 'x' is stylized with a blue-to-purple gradient and a curved, ribbon-like shape that loops around the top and bottom of the letter, extending towards the 'e'.

```
public class XtendCompiler extends XbaseCompiler {
```

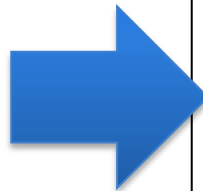
```
@Override
```

```
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;
    }
}
```

Xtend

Advanced features
(active annotation)

Model transformation in depth

Active Annotations

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

Do You know Java Annotations ?



JUnit



@Override

@SuppressWarnings



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

JUnit

```
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 {

}
```

```
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)

<code>@Test</code> <code>public void</code> <code>method()</code>	The <code>@Test</code> annotation identifies a method as a test method.
<code>@Test (expected =</code> <code>Exception.class)</code>	Fails, if the method does not throw the named exception.
<code>@Test(timeout=100)</code>	Fails, if the method takes longer than 100 milliseconds.
<code>@Before</code> <code>public void</code> <code>method()</code>	This method is executed before each test. It is used to can prepare the test environment (e.g. read input data, initialize the class).
<code>@After</code> <code>public void</code> <code>method()</code>	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.
<code>@BeforeClass</code> <code>public static void</code> <code>method()</code>	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 <code>static</code> to work with JUnit.
<code>@AfterClass</code> <code>public static void</code> <code>method()</code>	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 <code>static</code> to work with JUnit.

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


```
@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;
    }
```



JAXB

Java

Annotations

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



```
<?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
 - Suppress 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 is applied, if no 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, `expected`

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

The second optional parameter, `timeout`, causes a test to fail if it takes longer than the specified time.

```
@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 parameter
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 method
87      * <b>Consider using the {@link org.junit.rules.Timeout} rule instead
88      * same thread as the fixture's @Before and @After methods.
89      * </p>
90      */
91     long timeout() default 0L;
92 }
```

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- ▶ JRE System Library [JavaSE-1.6]
- ▶ JUnit 4



```
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



JUnit 4

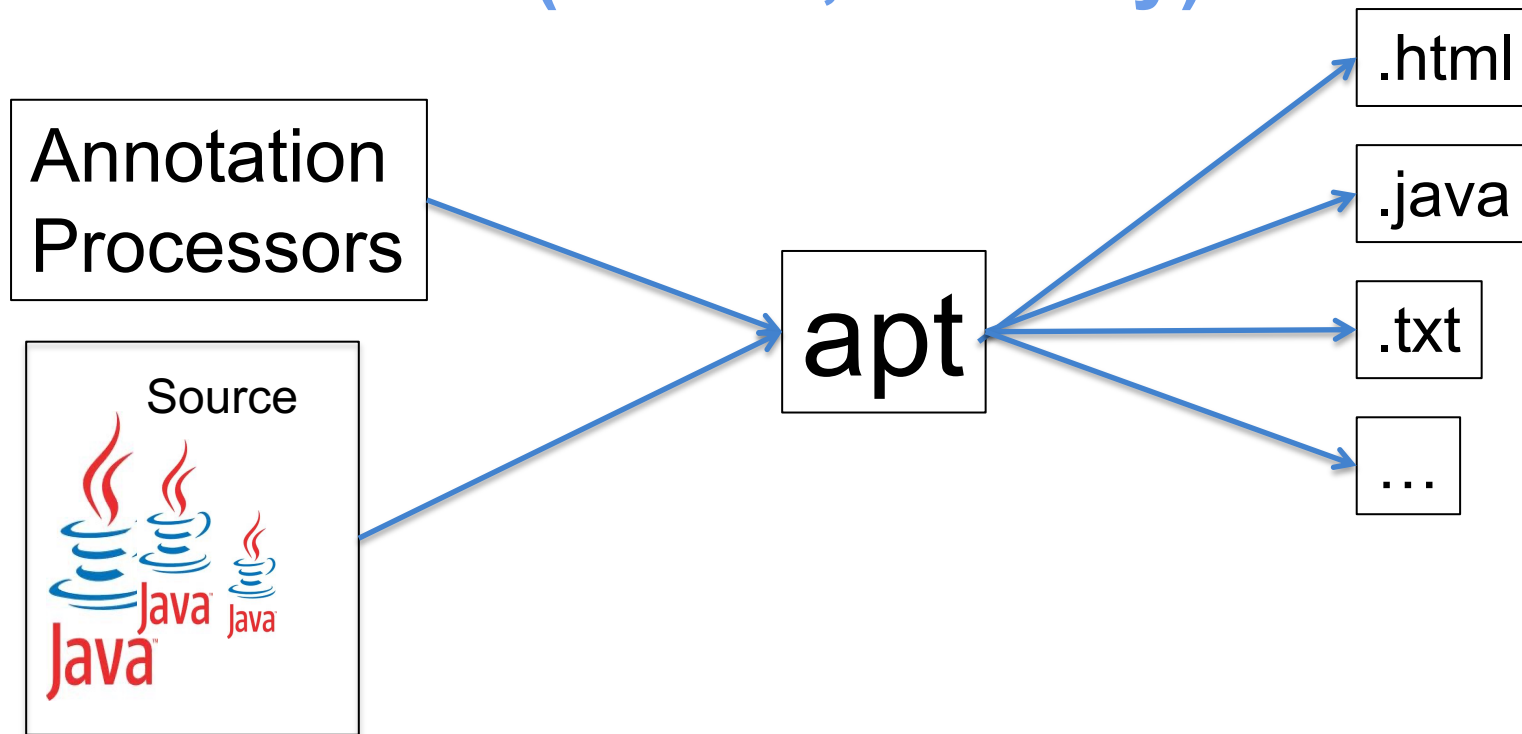
Finished after 0,01 seconds

Runs: 1/1 Errors: 0 Failures: 0

fr.inria.k3.SingletonTest [Runner: JUnit 4] (0,000 s)

test1 (0,000 s)

Annotations and Transformations (Java 5, old way)



← → ↻ docs.oracle.com/javase/1.5.0/docs/guide/apt/GettingStarted.html



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 annotation

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 accepted by javac.

The apt specific options are:

- s *dir*
Specify the directory root under which processor-generated source files will be placed.
- 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 classpath is ignored.
- factory *classname*
Name of AnnotationProcessorFactory to use; bypasses default discovery procedure.

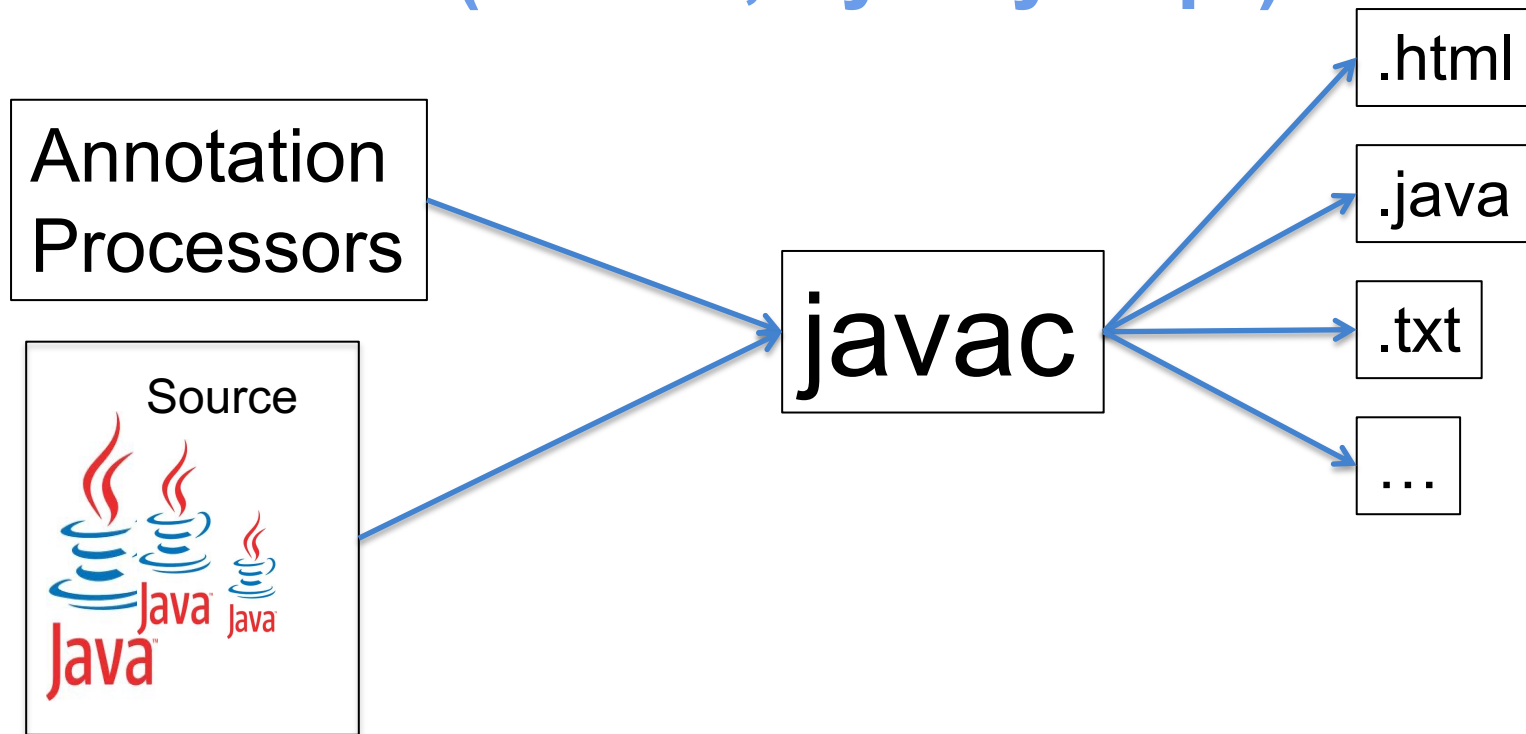
How apt shares some of javac's options:

- 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 -factory is used, this option is ignored.

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)

Annotation

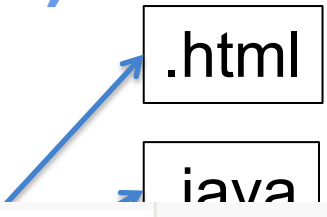
```
Proc import java.util.*;
import javax.annotation.processing.*;
import javax.lang.model.*;
import javax.lang.model.element.*;

@SupportedAnnotationTypes(value= {"*"})
@SupportedSourceVersion(SourceVersion.RELEASE_6)

public class TestAnnotationProcessor extends AbstractProcessor {

    @Override
    public boolean process(
        Set<?> extends TypeElement> annotations, RoundEnvironment roundEnv){

        for (TypeElement element : annotations){
            System.out.println(element.getQualifiedName());
        }
        return true;
    }
}
```



The diagram illustrates the transformation process. A blue arrow points from a box labeled ".java" to a box labeled ".html". The arrow originates from the ".java" box and points towards the ".html" box, indicating the output of the annotation processing.

javac -processor ...

Alternative: Java Reflection

```
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 dateAssignment();
}
```

<http://www.jmdoudoux.fr/java/dej/ch-ap-annotations.htm#annotations-7>

```
@Todo(importance = Importance.CRITIQUE,
      description = "Corriger le bug dans le calcul",
      assigneA = "JMD",
      dateAssignment = "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.dateAssignment()+")");
            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.dateAssignment()+")");
                for(String desc : todo.description()) {
                    System.out.println("    _ "+desc);
                }
            }
        }
    }

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

    }

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

    }
}
```

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 than 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;  
    }  
}
```

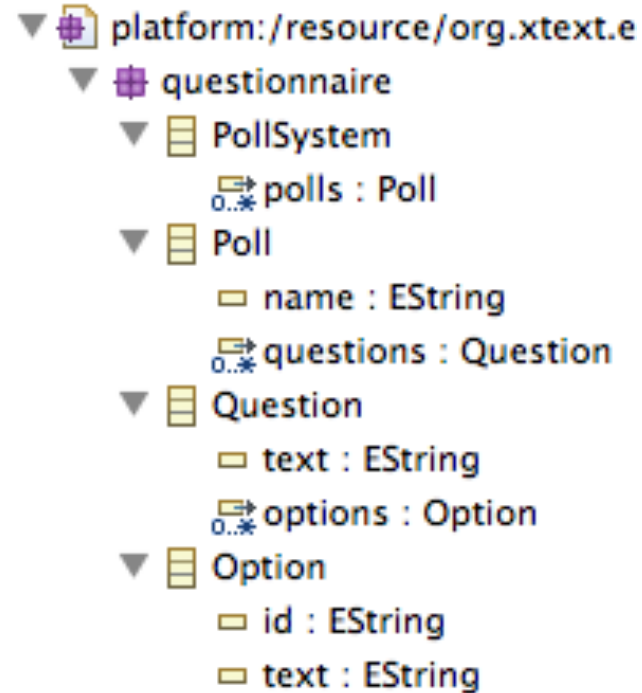
Visitors, EMF, and Xtend

(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"
      }
    }
  }
}

```



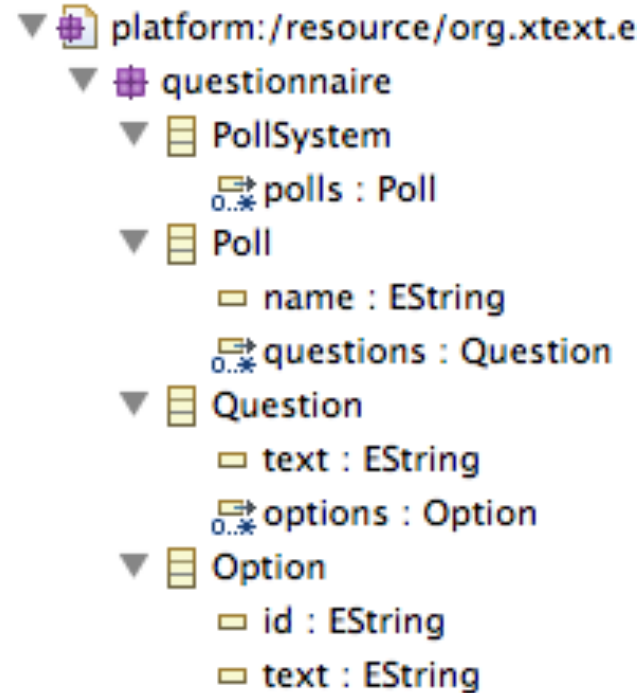
We already give examples of transformation, defined over the metamodel...

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)

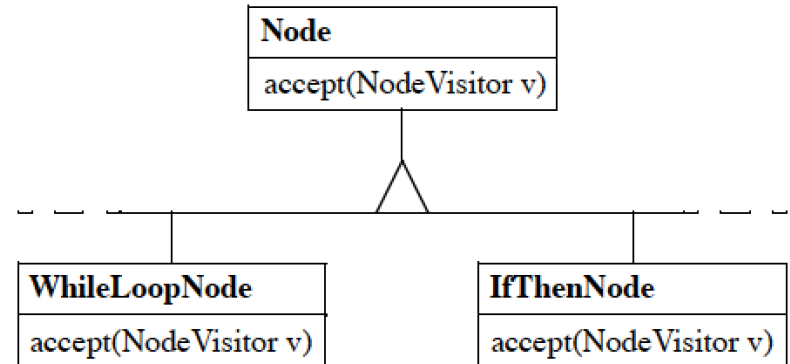
Possible solution: a series of casts (lots of if-statements and traversal loops)

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);
}

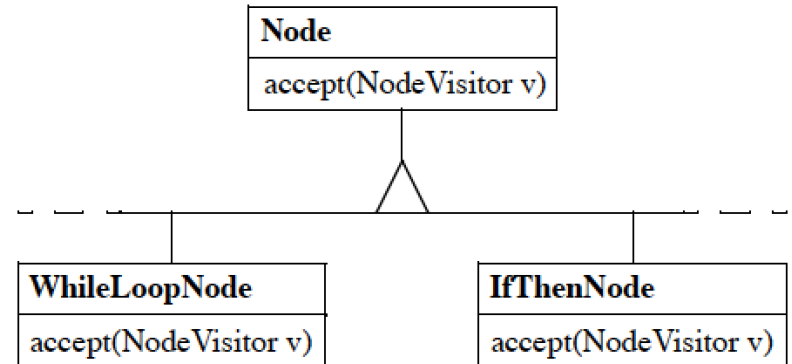
public class TypeCheckingVisitor extends NodeVisitor {
    /* ... */
    public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
    public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

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);
}

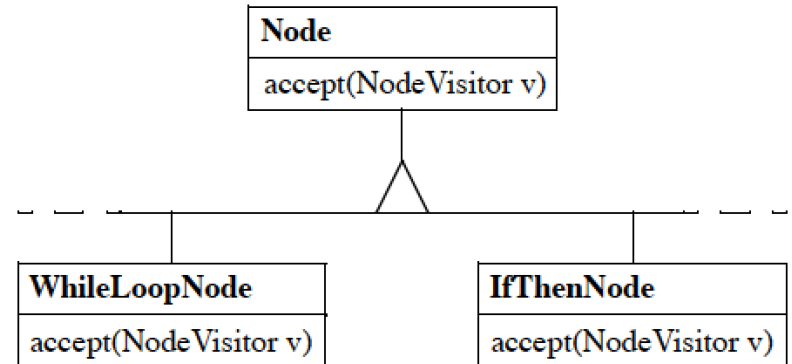
public class TypeCheckingVisitor extends NodeVisitor {
    /* ... */
    public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
    public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

#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);
}

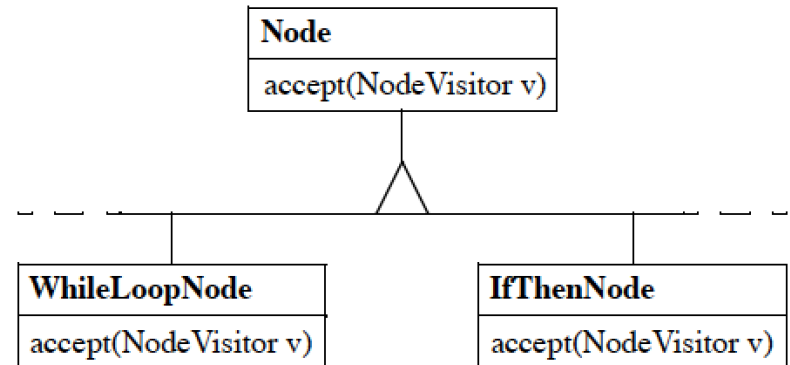
public class TypeCheckingVisitor extends NodeVisitor {
    /* ... */
    public void visitWhileLoop(WhileLoopNode n) { n.getCondition().accept(this); /* ... */ }
    public void visitIfThen(IfThenNode n) { /* ... */ }
}
```

#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)

- org.xtext.example.questionnaire
 - src
 - org.xtext.example.mydsl
 - QuestionnaireRuntimeModule.java
 - QuestionnaireStandaloneSetup.java
 - GenerateQuestionnaire.mwe2
 - Questionnaire.xtext

- org.xtext.example.mydsl.formatting
- org.xtext.example.mydsl.generator
- org.xtext.example.mydsl.scoping
- org.xtext.example.mydsl.validation

- src-gen
 - org.xtext.example.mydsl
 - org.xtext.example.mydsl.parser.antlr
 - org.xtext.example.mydsl.parser.antlr.internal

- org.xtext.example.mydsl.questionnaire
 - Option.java
 - Poll.java
 - PollSystem.java
 - Question.java
 - QuestionnaireFactory.java
 - QuestionnairePackage.java

- org.xtext.example.mydsl.questionnaire.impl
- org.xtext.example.mydsl.questionnaire.util
- org.xtext.example.mydsl.serializer
- org.xtext.example.mydsl.services
- org.xtext.example.mydsl.validation

- xtend-gen
- JRE System Library [J2SE-1.5]
- Plug-in Dependencies
- META-INF

- model
 - generated
 - Questionnaire.ecore
 - Questionnaire.genmodel

```
grammar org.xtext.example.mydsl.Questionnaire with org.eclipse.xtext.common.Terminals
generate questionnaire "http://www.xtext.org/example/mydsl/Questionnaire"

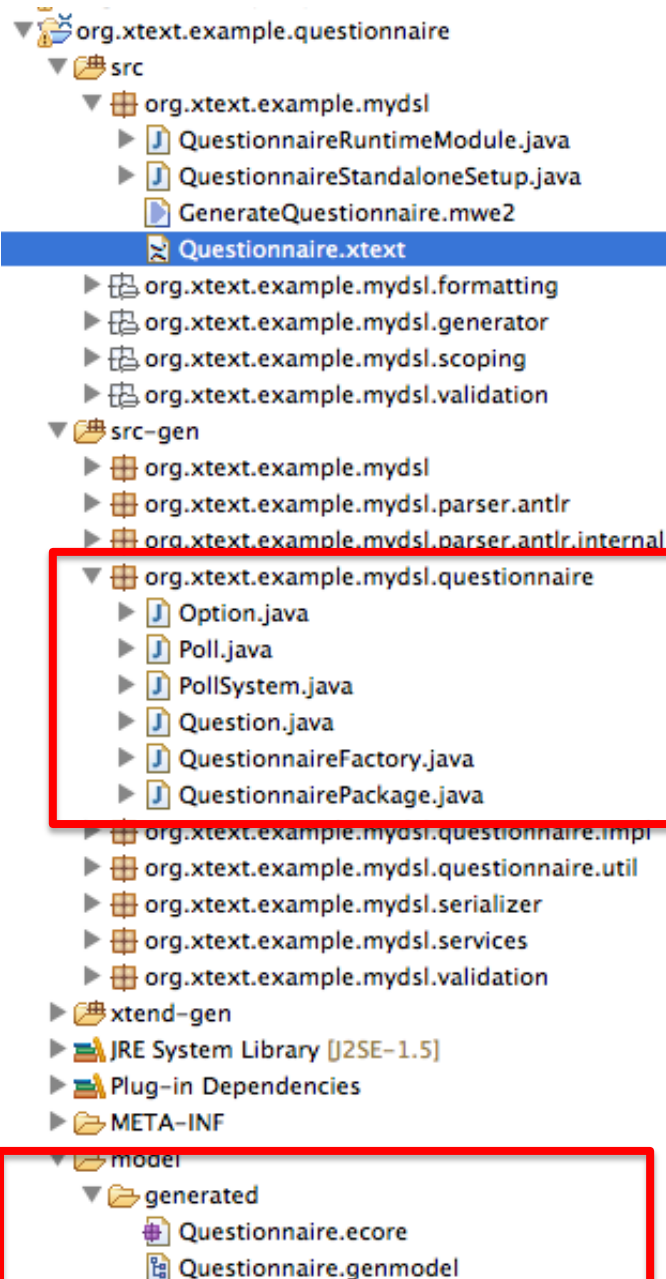
PollSystem:
    'PollSystem' '{' polls+=Poll+ '}';

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 mo
     * </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);
}
```

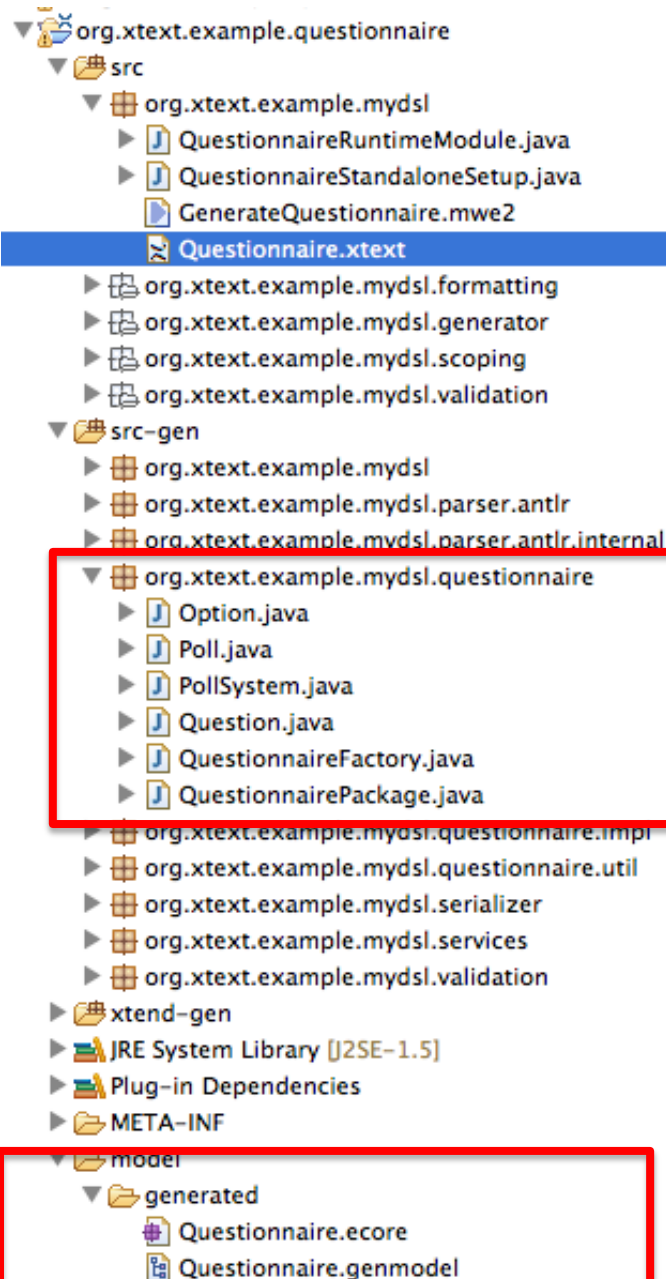
No accept method

The image shows the Eclipse IDE project tree for the `org.xtext.example.questionnaire` project. The tree is organized into several packages and source files:

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

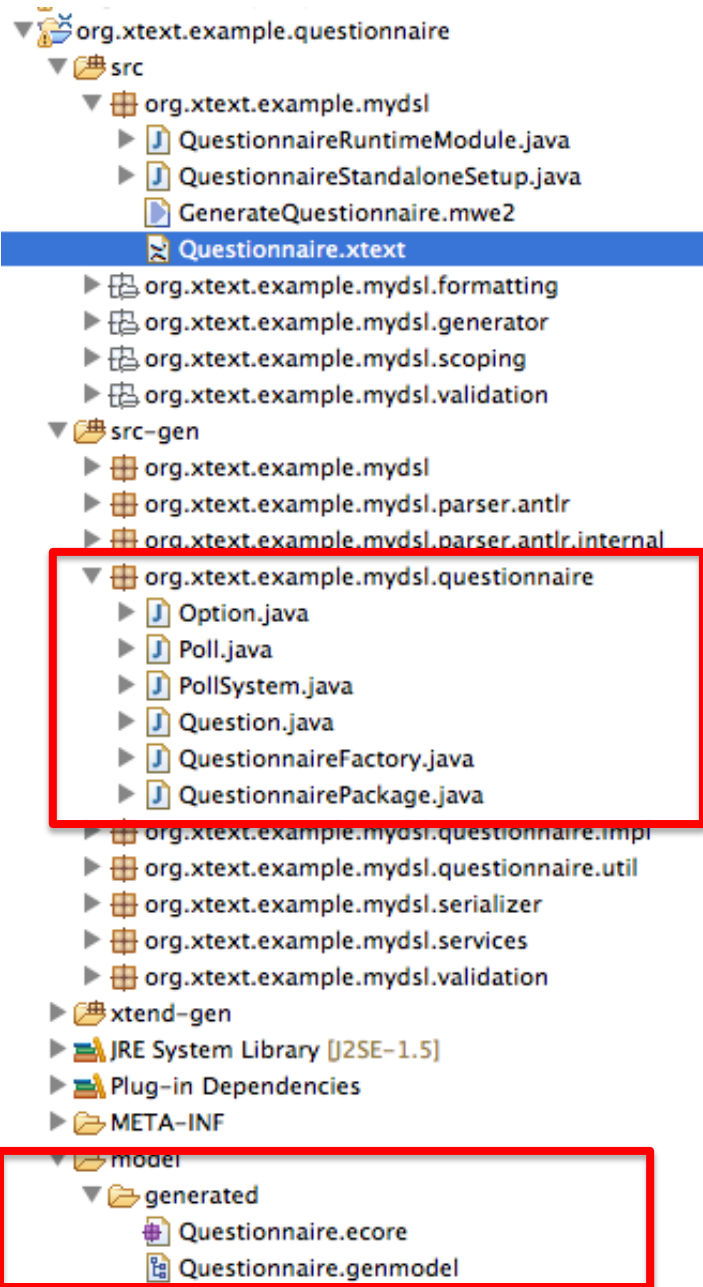
Visitor Pattern (impact of the problem)

⇒ **Manual**

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

```
public interface Question extends EObject  
{  
  
    public void accept(QuestionnaireVisitor vis);  
}
```

⇒ **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

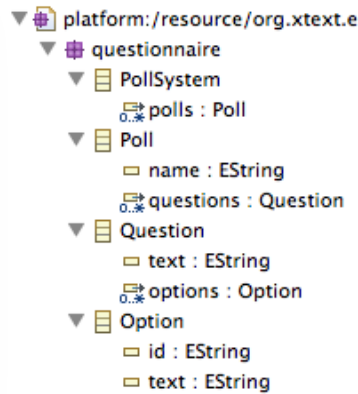
#3 class hierarchy evolution (e.g., new Node subclass) forces us to (completely) rewrite NodeVisitor

Automation

```

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"
      }
    }
  }
}

```



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

org.xtext.example.mydsl.questionnaire.util

QuestionnaireSwitch<T>

- ◆ S modelPackage : QuestionnairePackage
- C QuestionnaireSwitch()
- ◆ ▲ isSwitchFor(EPackage) : boolean
- ◆ ▲ doSwitch(int, EObject) : T
- casePollSystem(PollSystem) : T
- casePoll(Poll) : T
- caseQuestion(Question) : T
- caseOption(Option) : T
- ▲ defaultCase(EObject) : T

```

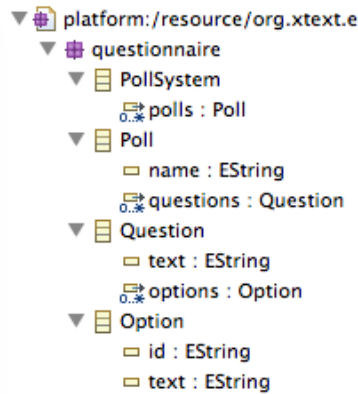
/**
 * The switch that delegates to the <code>createXXX</code> methods.
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
protected QuestionnaireSwitch<Adapter> modelSwitch =
  new QuestionnaireSwitch<Adapter>()
  {
    @Override
    public Adapter casePollSystem(PollSystem object)
    {
      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();
    }
  };

```

```

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"
      }
    }
  }
}

```



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, Li  
    val Map<MutableMethodDeclaration, Set<MutableMethodDeclaration>> dispatchmethod = new HashMap<MutableMethodDeclarati  
    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 className = clazz.annotations.findFirst[getValue('className') != null].getValue('className')  
        //addError(clazz, className.class.toString)
```

```
        //var simpleNameF = className.eClass.EAllStructuralFeatures.findFirst[name == "simpleName"]  
        //val className = className.eGet(simpleNameF) as String  
        val className = className.class.getMethod("getSimpleName").invoke(className) as String  
        //var identF = className.eClass.getEAllStructuralFeatures().findFirst[name == "identifier"]  
        //val identifier = className.eGet(identF) as String  
        val identifier = className.class.getMethod("getIdentifier").invoke(className) 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/k3-al/fr.inria.diverse.k3.al.annotationprocessor/src/main/java/fr/inria/diverse/k3/al/annotationprocessor/Aspect.xtend>

<http://www.eclipse.org/xtend/documentation.html>

<http://jnario.org/org/jnario/jnario/documentation/20FactsAboutXtendSpec.html>

<http://blog.efftinge.de/2012/12/java-8-vs-xtend.html>

<http://eclipsesource.com/blogs/tutorials/emf-tutorial/>

The logo for Xtend, featuring a stylized 'X' composed of three overlapping arrow-like shapes pointing right, followed by the word 'tend' in a bold, lowercase, sans-serif font.

#1 Model Transformations

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

#2 Xtend

(A general purpose language with advanced features; an illustration on how to transform models in practice; Xtend written in Xtext, using MDE principles)

#3 All together

Grammar, Metamodel,
models/specifications, DSL/GPL,
model transformations, meta-
programming

For breathing life
into models!

References

- Krzysztof Czarnecki, Simon Helsen: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3): 621-646 (2006)
- Krzysztof Czarnecki and Lurich Eisenecker “Generative Programming: Methods, Tools, and Applications”
- Pierre-Alain Muller, Franck Fleurey and Jean-Marc Jézéquel “Weaving Executability into Object-Oriented Meta-Languages” MODELS’05
- Pierre-Alain Muller , Frédéric Fondement, Franck Fleurey, Michel Hassenforder, Rémi Schnekenburger, Sébastien Gérard, Jean-Marc Jézéquel “Model-driven analysis and synthesis of textual concrete syntax” SoSyM’08
- Sven Efftinge, Moritz Eysholdt, Jan Köhnlein, Sebastian Zarnekow, Robert von Massow, Wilhelm Hasselbring, and Michael Hanus. Xbase: Implementing domain-specific languages for java. GPCE ’12

References

- M. Eysholdt and H. Behrens. “Xtext: Implement your language faster than the quick and dirty way.” In OOPSLA '10
- Curtis Clifton, Gary T. Leavens, Craig Chambers, and Todd Millstein. “MultiJava: modular open classes and symmetric multiple dispatch for Java” OOPSLA'00
- Andy Schürr, Felix Klar “15 Years of Triple Graph Grammars.” ICGT 2008
- Mark Hills, Paul Klint, Tijs van der Storm, Jurgen J. Vinju “A Case of Visitor versus Interpreter Pattern.” TOOLS'11
- Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais and Jean-Marc Jézéquel “Melange: A Meta-language for Modular and Reusable Development of DSLs “ SLE'15