USAGE CONTRACTS

KIM MENS
UNIVERSITÉ CATHOLIQUE DE LOUVAIN (UCL)

JOINT WORK WITH ANGELA LOZANO & ANDY KELLENS

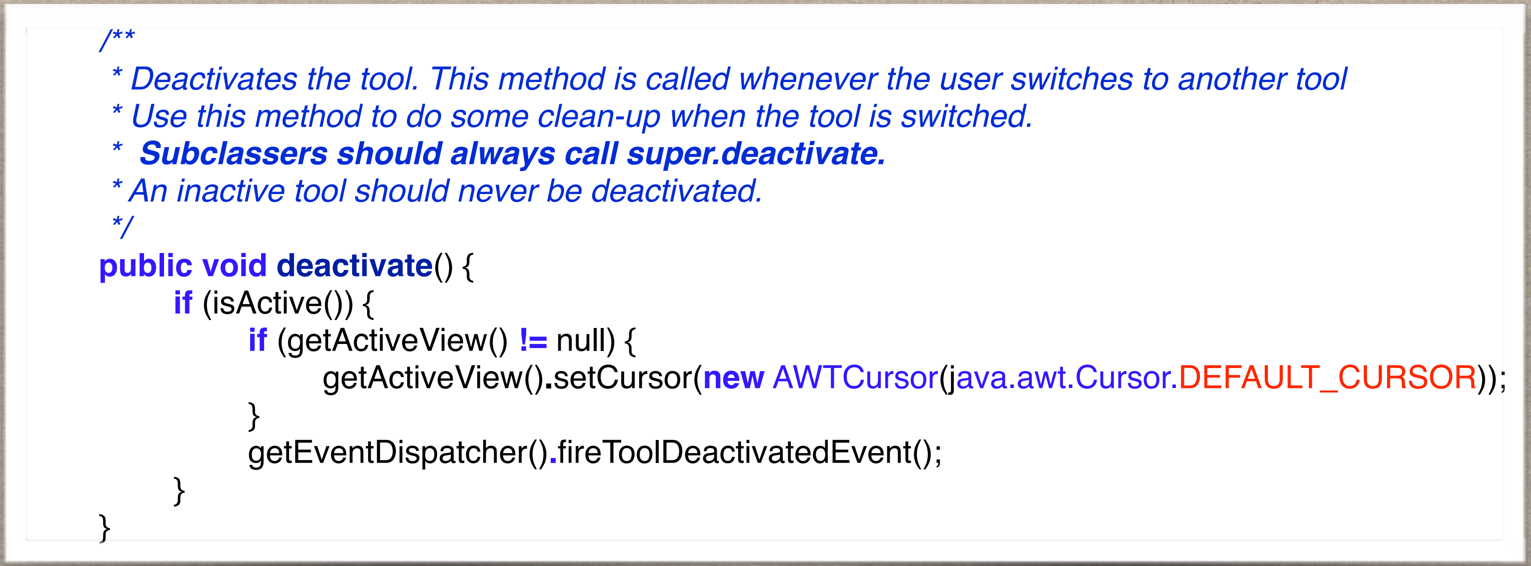
SATTOSE 2014 – L'AQUILA – 9-11.07.2014

SOME OF MY RESEARCH INTERESTS

- **Programming languages**
  - Context-Oriented Programming
  - Language interoperability between logic and OO
  - (Aspect-oriented programming ✝)
- **Tool support** for software development, maintenance and evolution
  - source code mining
  - source-code based recommendation tools
  - structural source-code regularities (e.g. usage contracts)

USAGE CONTRACTS : MOTIVATION

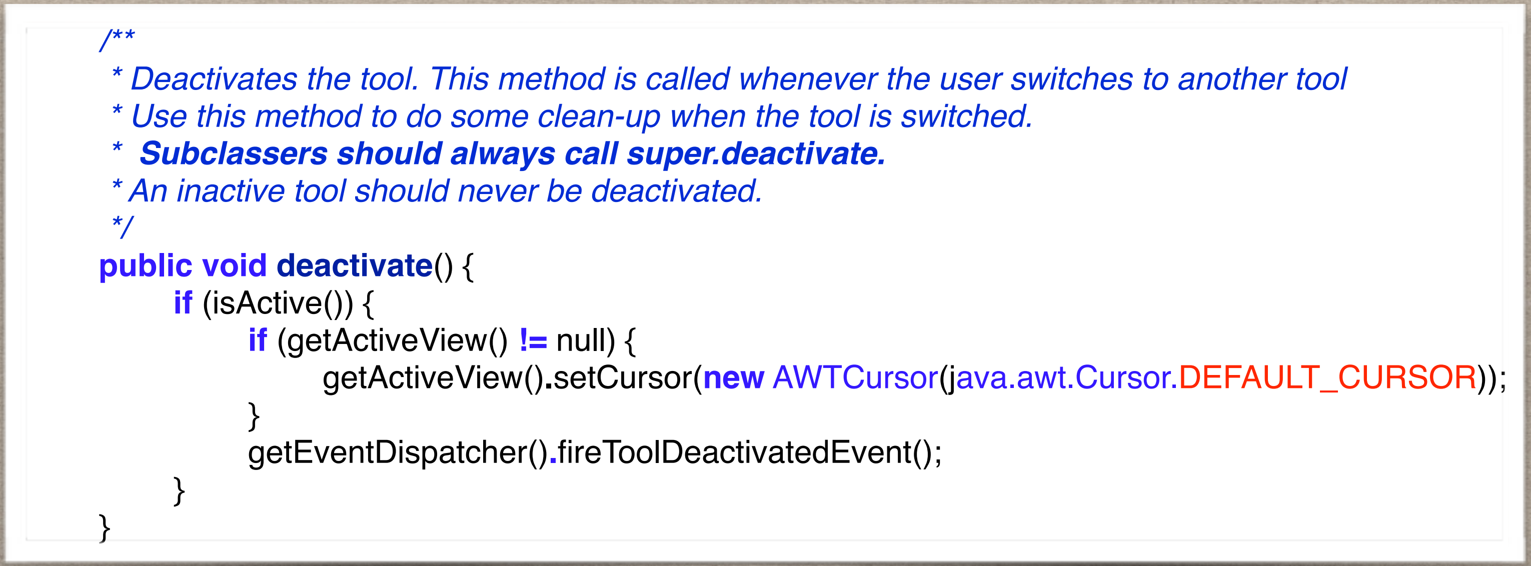
Often you find code comments like



```
/**
 * Deactivates the tool. This method is called whenever the user switches to another tool
 * Use this method to do some clean-up when the tool is switched.
 *  Subclassers should always call super.deactivate.
 * An inactive tool should never be deactivated.
 */
public void deactivate() {
        if (isActive()) {
                if (getActiveView() != null) {
                        getActiveView().setCursor(new AWTCursor(java.awt.Cursor.DEFAULT_CURSOR));
                }
                getEventDispatcher().fireToolDeactivatedEvent();
        }
}
```

USAGE CONTRACTS : MOTIVATION

Often you find code comments like

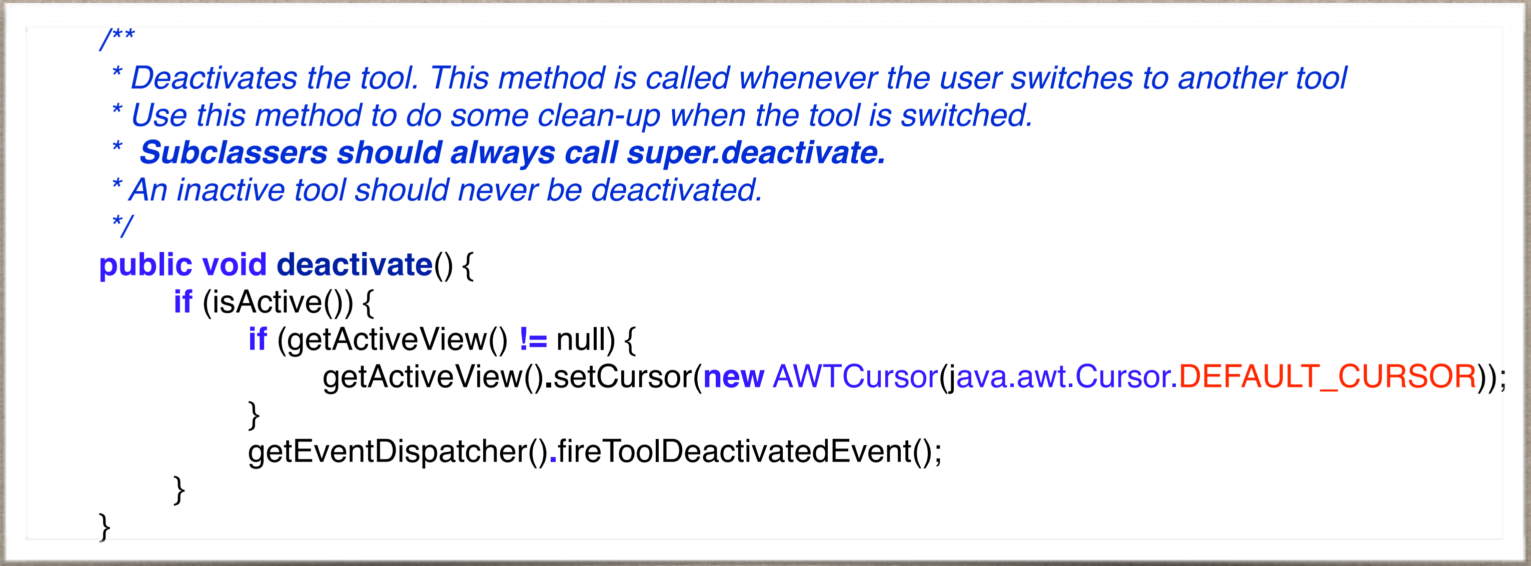


```
/**
 * Deactivates the tool. This method is called whenever the user switches to another tool
 * Use this method to do some clean-up when the tool is switched.
 *  Subclassers should always call super.deactivate.
 * An inactive tool should never be deactivated.
 */
public void deactivate() {
        if (isActive()) {
                if (getActiveView() != null) {
                        getActiveView().setCursor(new AWTCursor(java.awt.Cursor.DEFAULT_CURSOR));
                }
                getEventDispatcher().fireToolDeactivatedEvent();
        }
}
```

We studied JHotDraw for occurrences of "should, may, must, can(not), could, ought, have, has, need, require, …." and found 22 structural regularities like :

subclassers of this class	should	call …
this class	should not	do a supercall
…	must	implement …
	should (not)	override
methods in this class	…	only be called by …
this method		only be called internally
		be called after …

USAGE CONTRACTS : GOAL



```
/**
 * Deactivates the tool. This method is called whenever the user switches to another tool
 * Use this method to do some clean-up when the tool is switched.
 *  Subclassers should always call super.deactivate.
 * An inactive tool should never be deactivated.
 */
public void deactivate() {
        if (isActive()) {
                if (getActiveView() != null) {
                        getActiveView().setCursor(new AWTCursor(java.awt.Cursor.DEFAULT_CURSOR));
                }
                getEventDispatcher().fireToolDeactivatedEvent();
        }
}
```

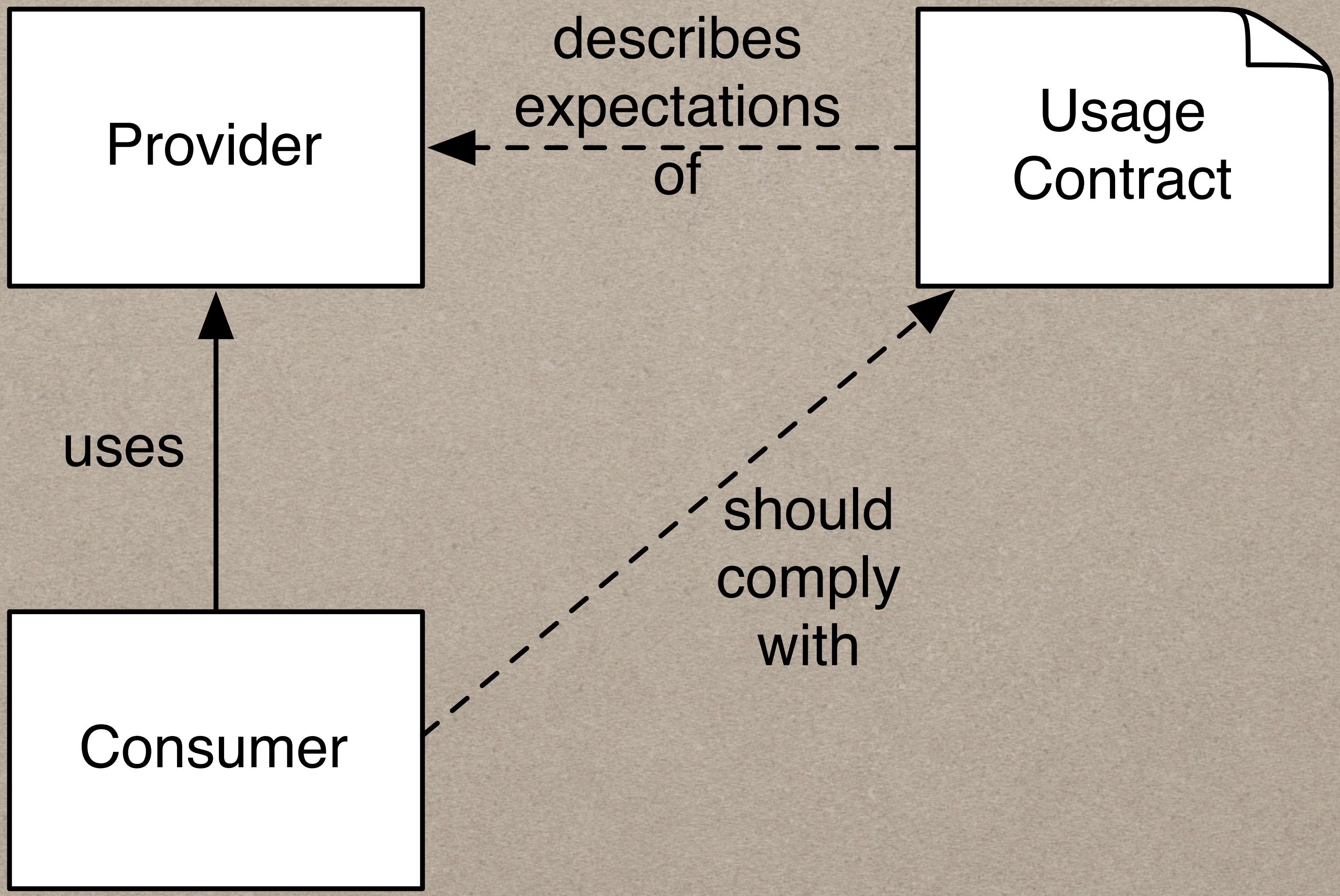
We want a tool that allows **encoding** such regularities **and** offering **immediate feedback on violations of** such **structural source-code regularities**

The tool should be **proactive** (violations reported 'on the fly' during coding)

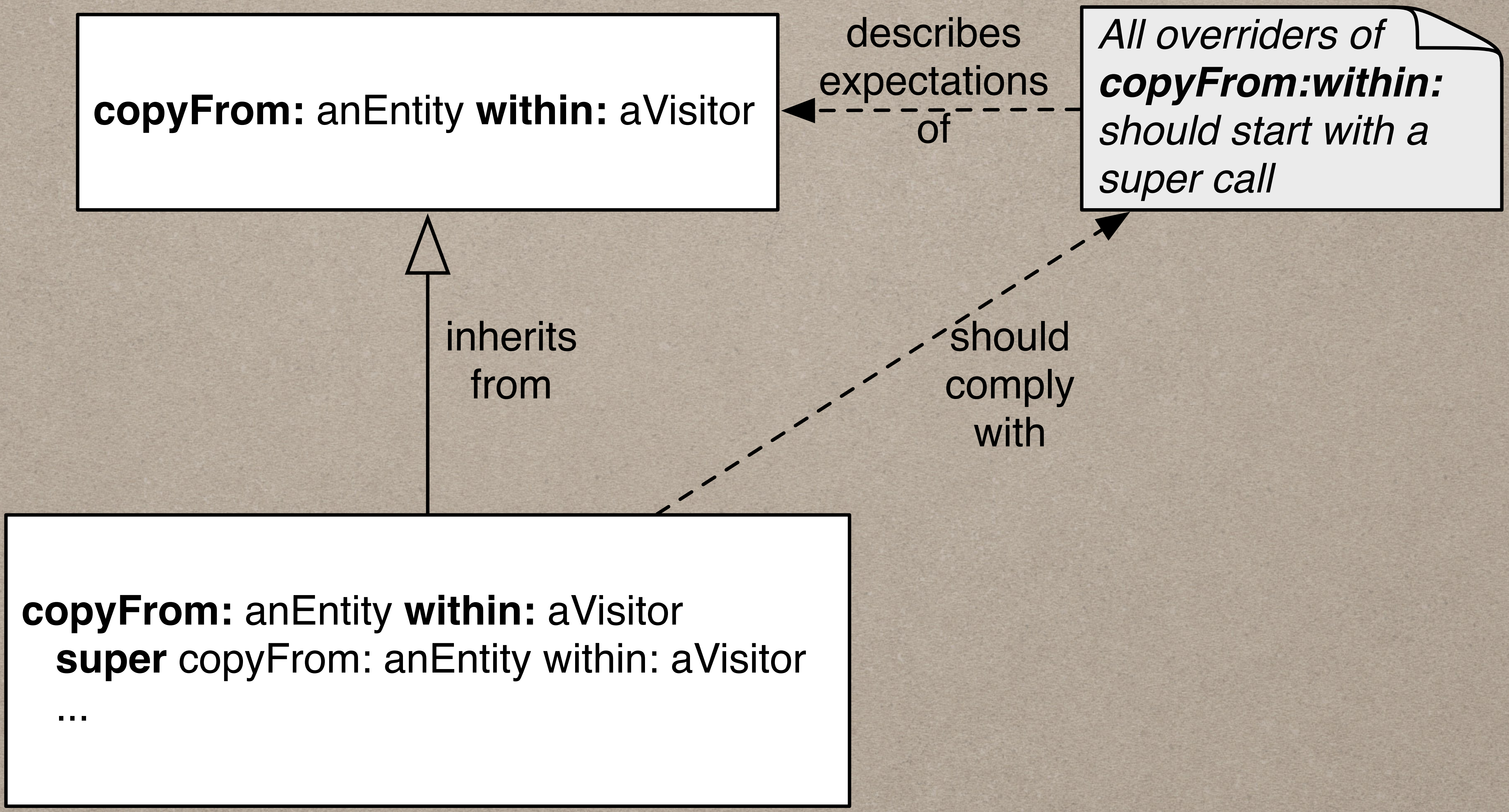
The tool should be "**developer-friendly**" (*like unit testing but for usage expectations*)

- desired regularities expressed in the same programming language
- tight integration with the integrated development environment
- not coercive

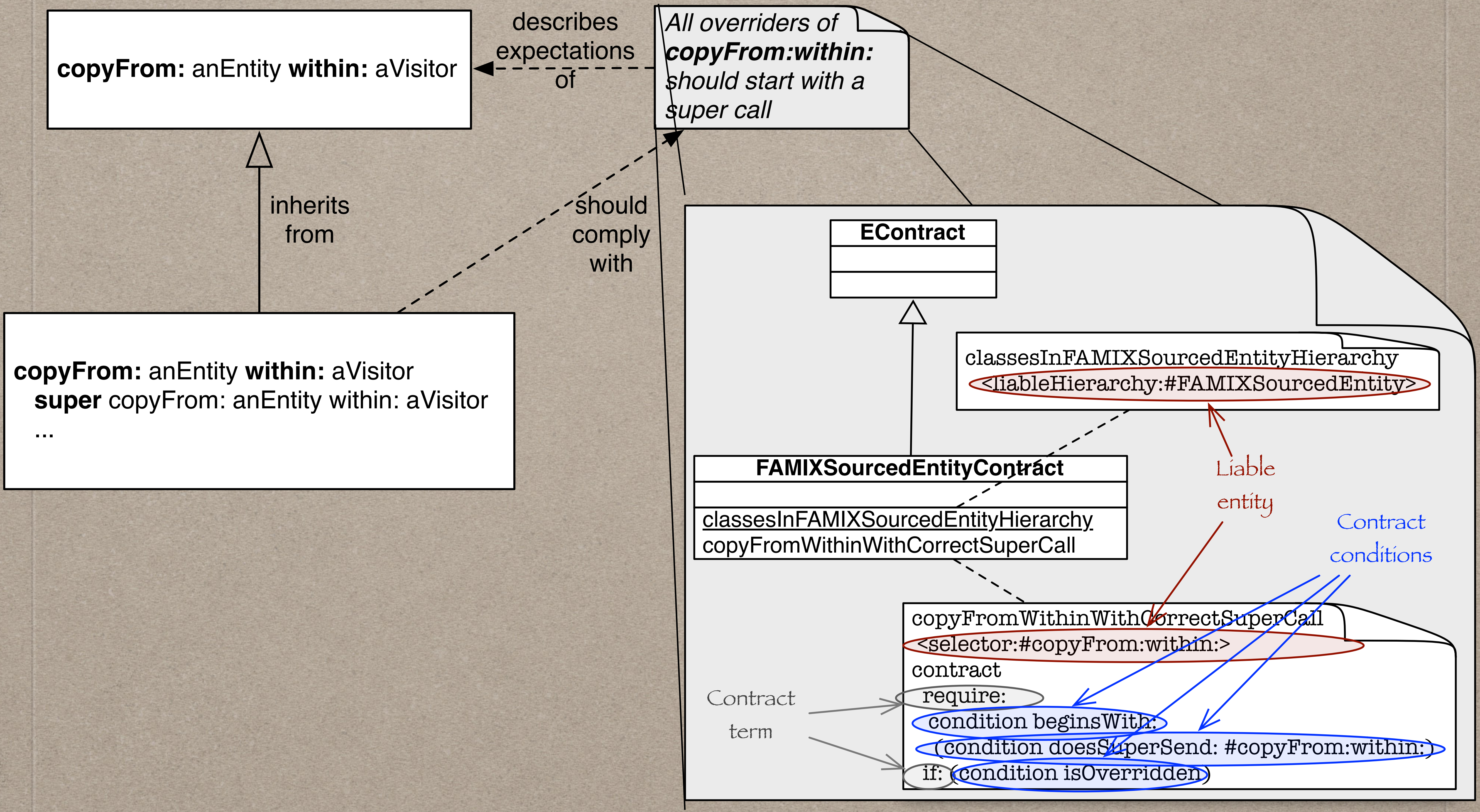
METAPHOR


Provider
describes expectations of
Usage Contract
uses
should comply with
Consumer

EXAMPLE


copyFrom: anEntity **within:** aVisitor
describes expectations of
All overriders of ***copyFrom:within:*** *should start with a super call*
inherits from
should comply with
copyFrom: anEntity **within:** aVisitor
super copyFrom: anEntity within: aVisitor
...

EXAMPLE


describes expectations of
All overriders of ***copyFrom:within:*** *should start with a super call*
inherits from
should comply with
EContract
FAMIXSourcedEntityContract
classesInFAMIXSourcedEntityHierarchy
copyFromWithinWithCorrectSuperCall
Liable entity
Contract conditions
Contract term

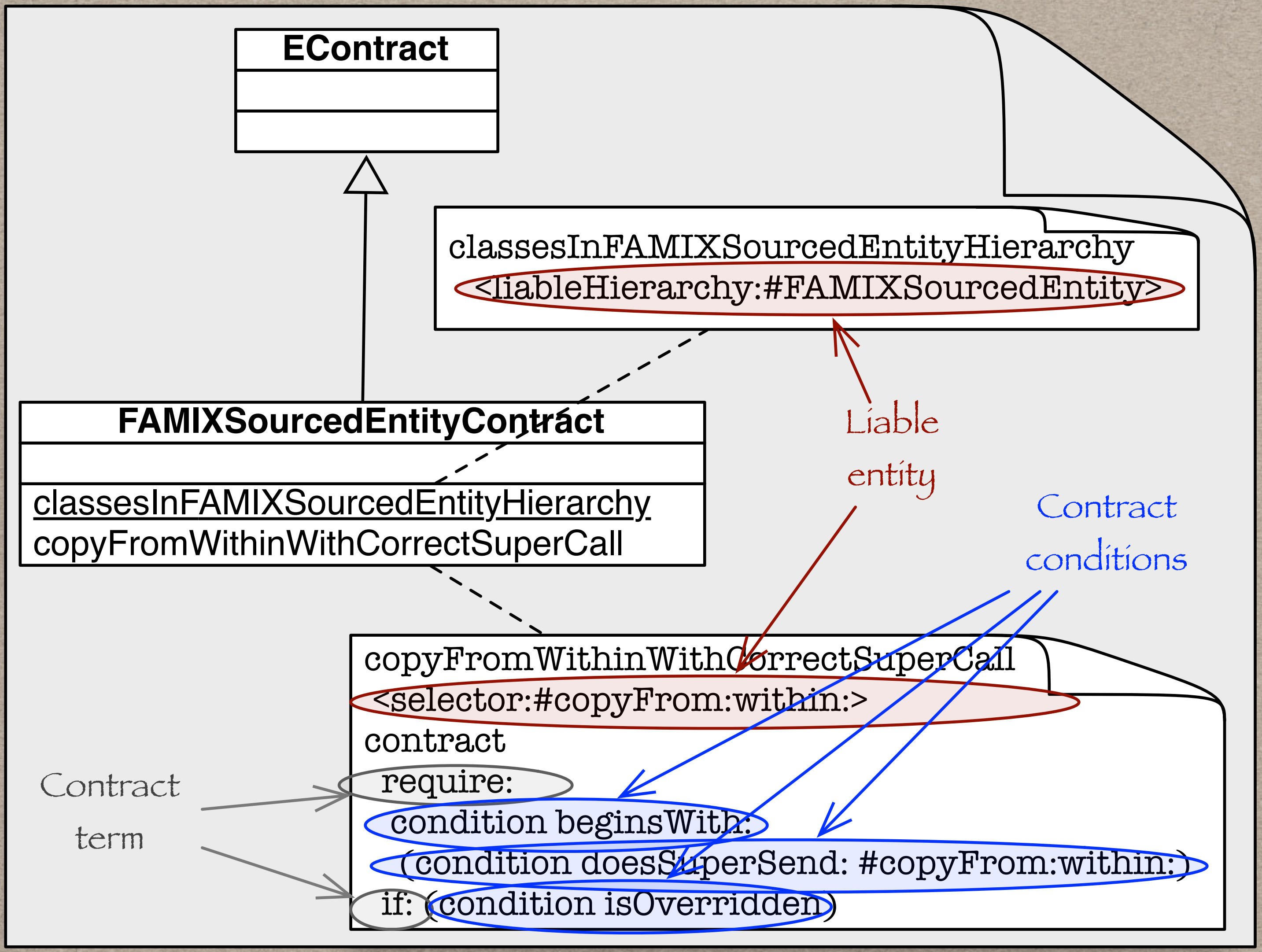
```
copyFrom: anEntity within: aVisitor
```

```
copyFrom: anEntity within: aVisitor
   super copyFrom: anEntity within: aVisitor
   ...
```

```
classesInFAMIXSourcedEntityHierarchy
 <liableHierarchy:#FAMIXSourcedEntity>
```

```
copyFromWithinWithCorrectSuperCall
 <selector:#copyFrom:within:>
contract
 require:
  condition beginsWith:
   (condition doesSuperSend: #copyFrom:within:)
 if: (condition isOverridden)
```

UCONTRACTS : THE LANGUAGE



EContract
classesInFAMIXSourcedEntityHierarchy
<liableHierarchy:#FAMIXSourcedEntity>
FAMIXSourcedEntityContract
classesInFAMIXSourcedEntityHierarchy
copyFromWithinWithCorrectSuperCall
Liable entity
Contract conditions
Contract term
copyFromWithinWithCorrectSuperCall
<selector:#copyFrom:within:>
contract
require:
condition beginsWith:
(condition doesSuperSend: #copyFrom:within:)
if: (condition isOverridden)

Liable classes

- **liableClass**: regExp / **exceptClass**: regExp
- **liableHierarchy**: className / **exceptHierarchy**: className
- **liablePackage**: regExp / **exceptPackage**: regExp

Liable methods

- **selector**: regExp / **exceptSelector**: regExp
- **protocol**: regExp / **exceptProtocol**: regExp
- / **exceptClass**: className **selector**: selector

Contract terms

- **require**: condition
- **suggest**: condition
- require: condition **if:** anotherCondition
- suggest: condition **if:** anotherCondition

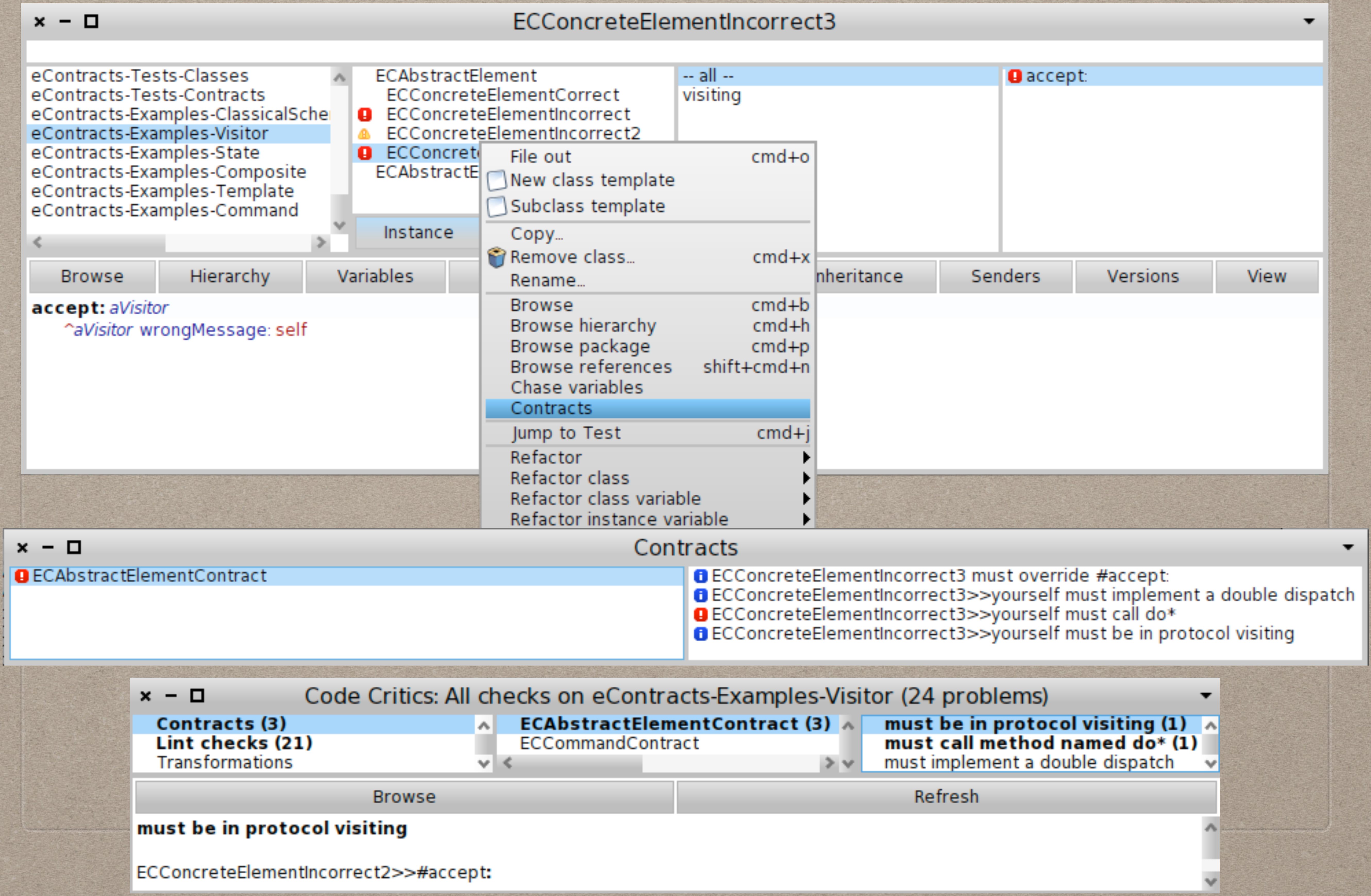
Contract conditions

- **assigns**: regExp
- **calls**: regExp
- **references**: regExp
- **returns**: expression
- **doesSuperSend**: regExp
- **doesSelfSend**: regExp
- **inProtocol**: regExp
- **isOverridden**: selector
- isOverridden
- **isImplemented**: selector
- **custom**: visitor

- **and**: cond1 with: cond2
- **or**: cond1 with: cond2
- **not**: cond

- **beginsWith**: cond
- **endsWith**: cond
- does: cond1 **after**: cond2
- does: cond1 **before**: cond2

UCONTRACTS : THE TOOL



ECConcreteElementIncorrect3
eContracts-Tests-Classes
eContracts-Tests-Contracts
eContracts-Examples-ClassicalSche
eContracts-Examples-Visitor
eContracts-Examples-State
eContracts-Examples-Composite
eContracts-Examples-Template
eContracts-Examples-Command
ECAbstractElement
ECConcreteElementCorrect
ECConcreteElementIncorrect
ECConcreteElementIncorrect2
ECConcrete
ECAbstractE
Instance
-- all --
visiting
accept:
File out cmd+o
New class template
Subclass template
Copy...
Remove class... cmd+x
Rename...
Browse cmd+b
Browse hierarchy cmd+h
Browse package cmd+p
Browse references shift+cmd+n
Chase variables
Contracts
Jump to Test cmd+j
Refactor
Refactor class
Refactor class variable
Refactor instance variable
Browse
Hierarchy
Variables
nheritance
Senders
Versions
View
Contracts
ECAbstractElementContract
Code Critics: All checks on eContracts-Examples-Visitor (24 problems)
Contracts (3)
Lint checks (21)
Transformations
ECAbstractElementContract (3)
ECCommandContract
must be in protocol visiting (1)
must call method named do* (1)
must implement a double dispatch
Browse
Refresh

```
accept: aVisitor
    ^aVisitor wrongMessage: self
```

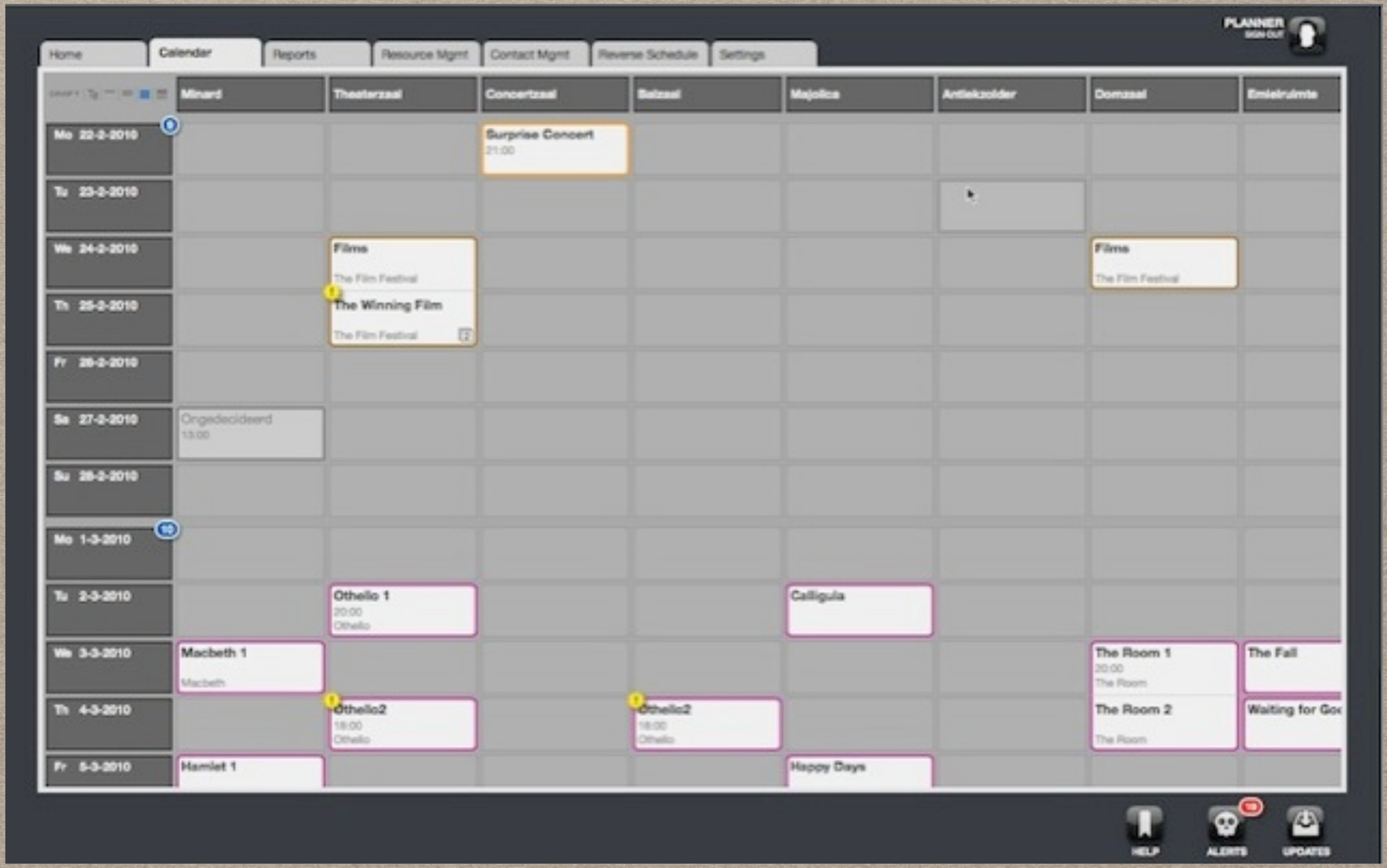
- ECConcreteElementIncorrect3 must override #accept:
- ECConcreteElementIncorrect3>>yourself must implement a double dispatch
- ECConcreteElementIncorrect3>>yourself must call do*
- ECConcreteElementIncorrect3>>yourself must be in protocol visiting

must be in protocol visiting

ECConcreteElementIncorrect2>>#accept:

VALIDATION ON AN INDUSTRIAL CASE

- An interactive web application for event & resource planning
  - developed in Pharo Smalltalk
  - uses the Seaside web development framework.
- Medium-sized
  - Packages: 45
  - Classes: 827
  - Methods: 11777
  - LOCs: 94151



INDUSTRIAL VALIDATION : SET-UP OF THE EXPERIMENT

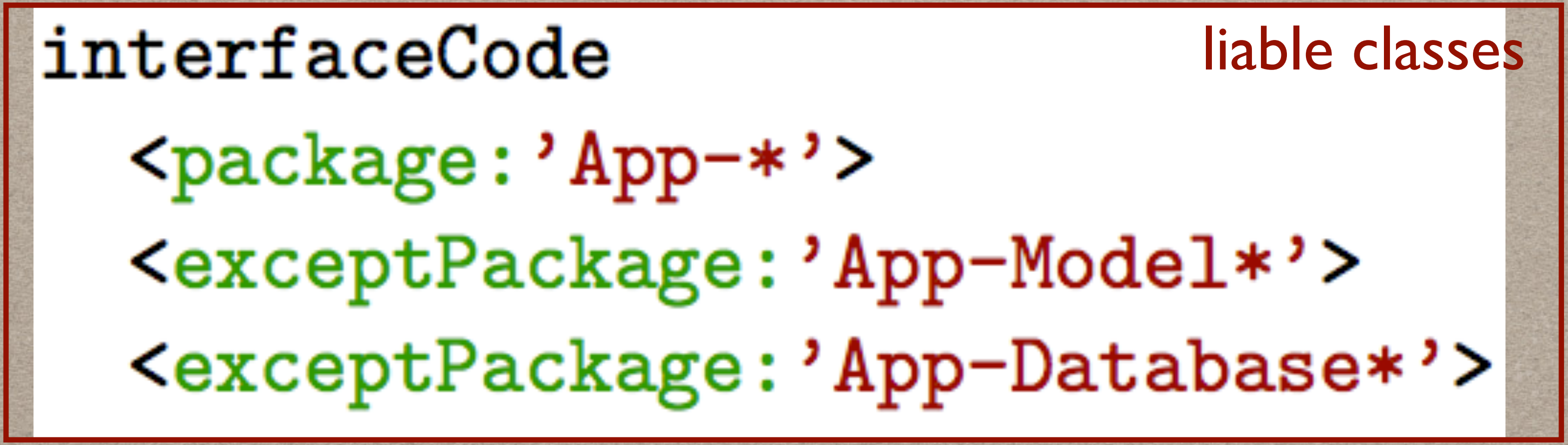
- Qualitative assessment
- Ideally we would have liked the tool to be used directly by the developers, but instead we had to perform an offline experiment.
- Together with the developers, during 2 days we defined 13 contracts documenting important regularities in their framework
- We checked all contracts in December and reported all contract breaches to the developers
- 3 months later, we reverified compliance of the code against the same contracts

INDUSTRIAL VALIDATION : ABOUT THE CONTRACTS

- contracts related to the **model** of the web application
  - for 3/5 of them violations were found
  - 214 liable classes, 88 violations
- contracts related to the classes dealing with **persistency**
  - for 2/2 of them violations where found
  - 75 liable classes, 2 violations found
- contracts about how the **UI** is constructed with the Seaside framework
  - for 4/6 of them violations where found
  - 598 liable classes, 8 violations found

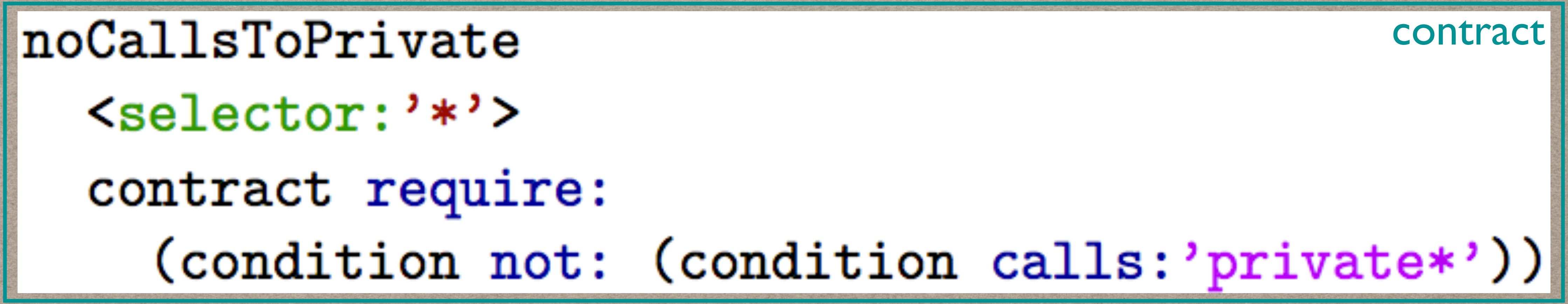
INDUSTRIAL VALIDATION : EXAMPLE OF A CONTRACT

Private methods should not be called directly



liable classes

```
interfaceCode
  <package:'App-*'>
  <exceptPackage:'App-Model*'>
  <exceptPackage:'App-Database*'>
```



contract

```
noCallsToPrivate
  <selector:'*'>
  contract require:
    (condition not: (condition calls:'private*'))
```

INDUSTRIAL VALIDATION : EXAMPLE OF A CONTRACT

In domain classes, state changes must mark model objects as dirty so that they can be re-rendered

liable classes

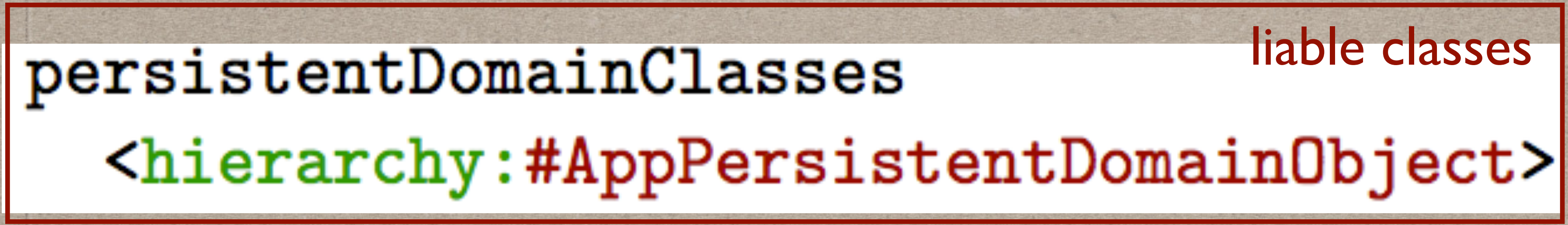
```
domainClasses
  <hierarchy:#AppDomainObject>
```

contract

```
dirtyFlag
  <selector:'*'>
  contract
    require: (condition calls: #markAsChanged:)
    if: (condition assigns: '*')
```

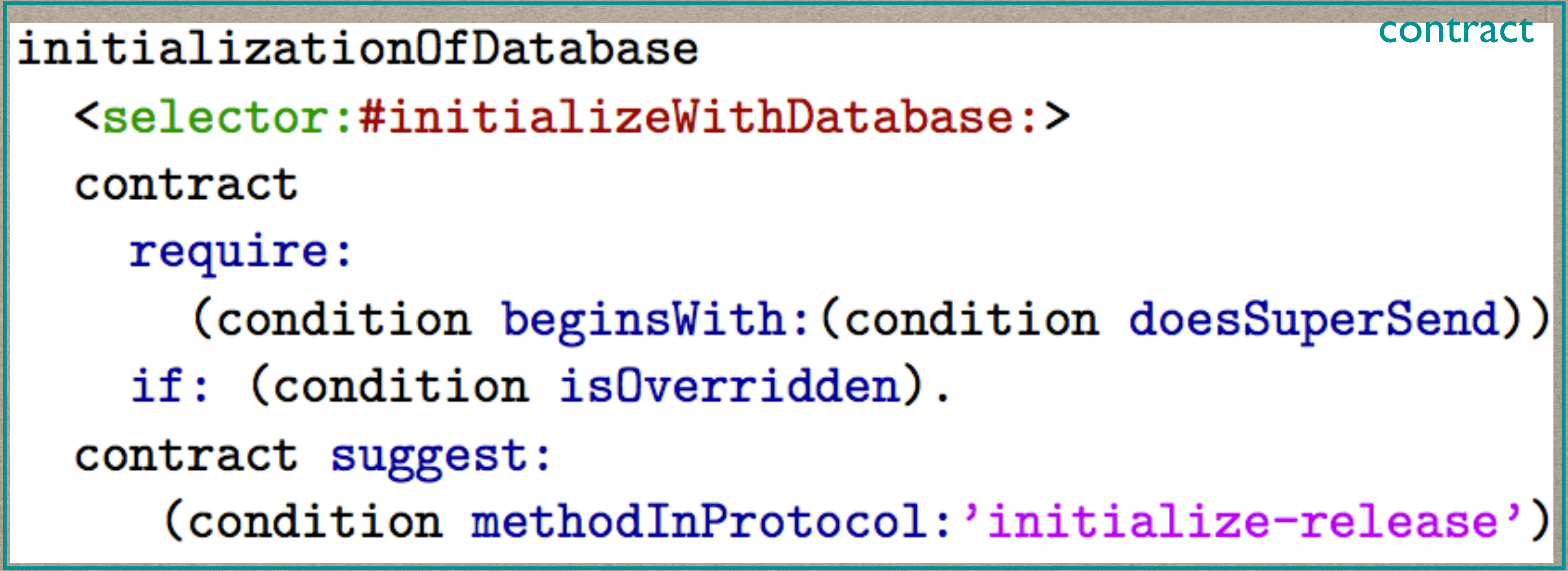
INDUSTRIAL VALIDATION : EXAMPLE OF A CONTRACT

Overridden initialisation methods should start with a super call (and be put in an appropriate protocol)



liable classes

```
persistentDomainClasses
   <hierarchy:#AppPersistentDomainObject>
```

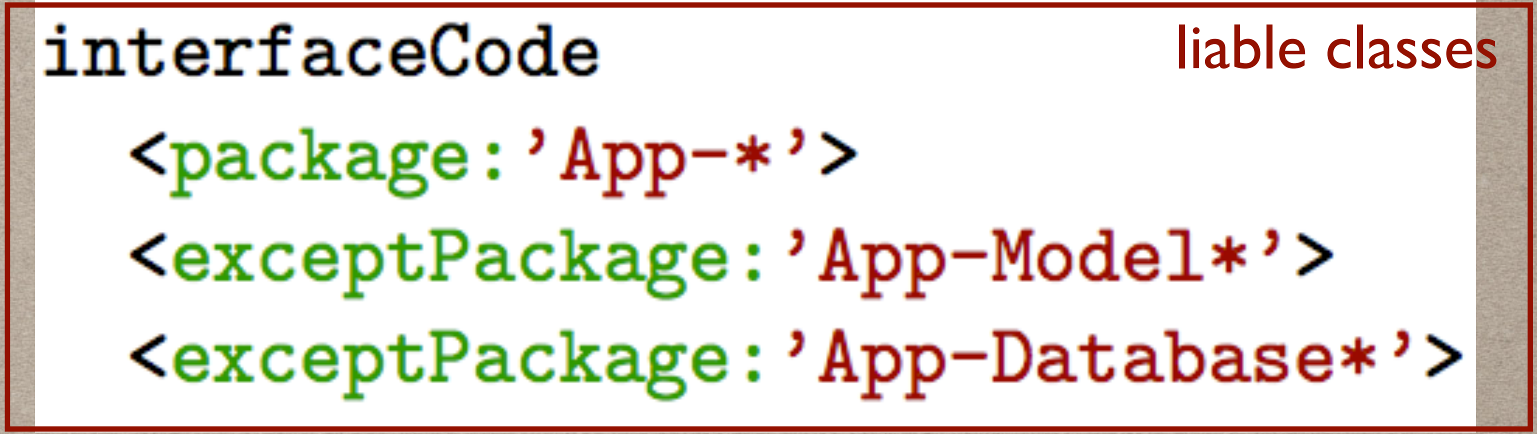


contract

```
initializationOfDatabase
  <selector:#initializeWithDatabase:>
  contract
    require:
      (condition beginsWith:(condition doesSuperSend))
    if: (condition isOverridden).
  contract suggest:
     (condition methodInProtocol:'initialize-release')
```

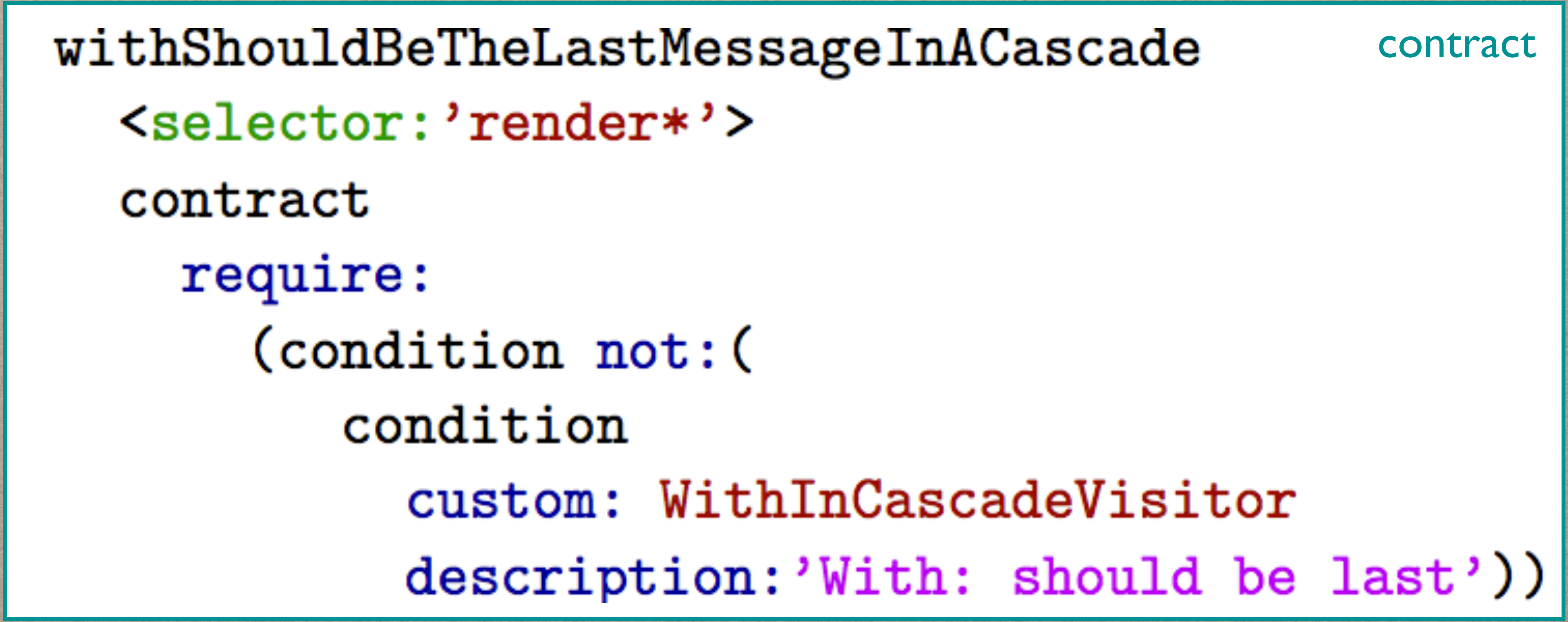
INDUSTRIAL VALIDATION : EXAMPLE OF A CONTRACT

Certain messages need to be sent at the end of a method cascade



liable classes

```
interfaceCode
  <package:'App-*'>
  <exceptPackage:'App-Model*'>
  <exceptPackage:'App-Database*'>
```

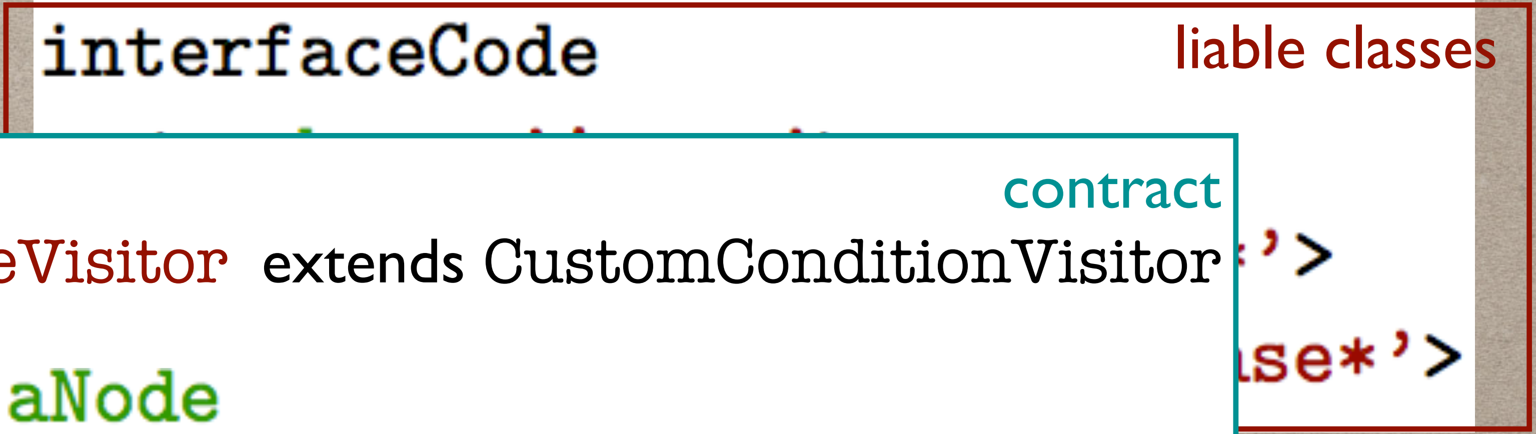


contract

```
withShouldBeTheLastMessageInACascade
  <selector:'render*'>
  contract
    require:
      (condition not:(
          condition
            custom: WithInCascadeVisitor
            description:'With: should be last'))
```

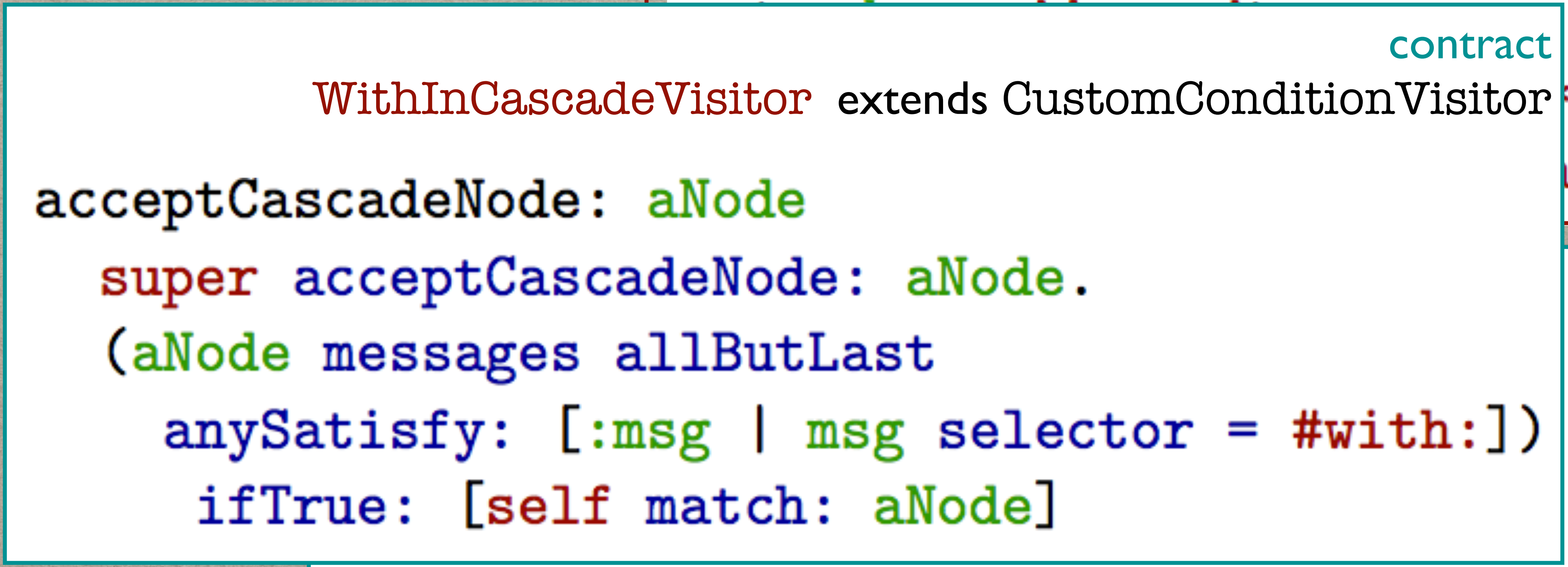
INDUSTRIAL VALIDATION : EXAMPLE OF A CONTRACT

Certain messages need to be sent at the end of a method cascade



liable classes

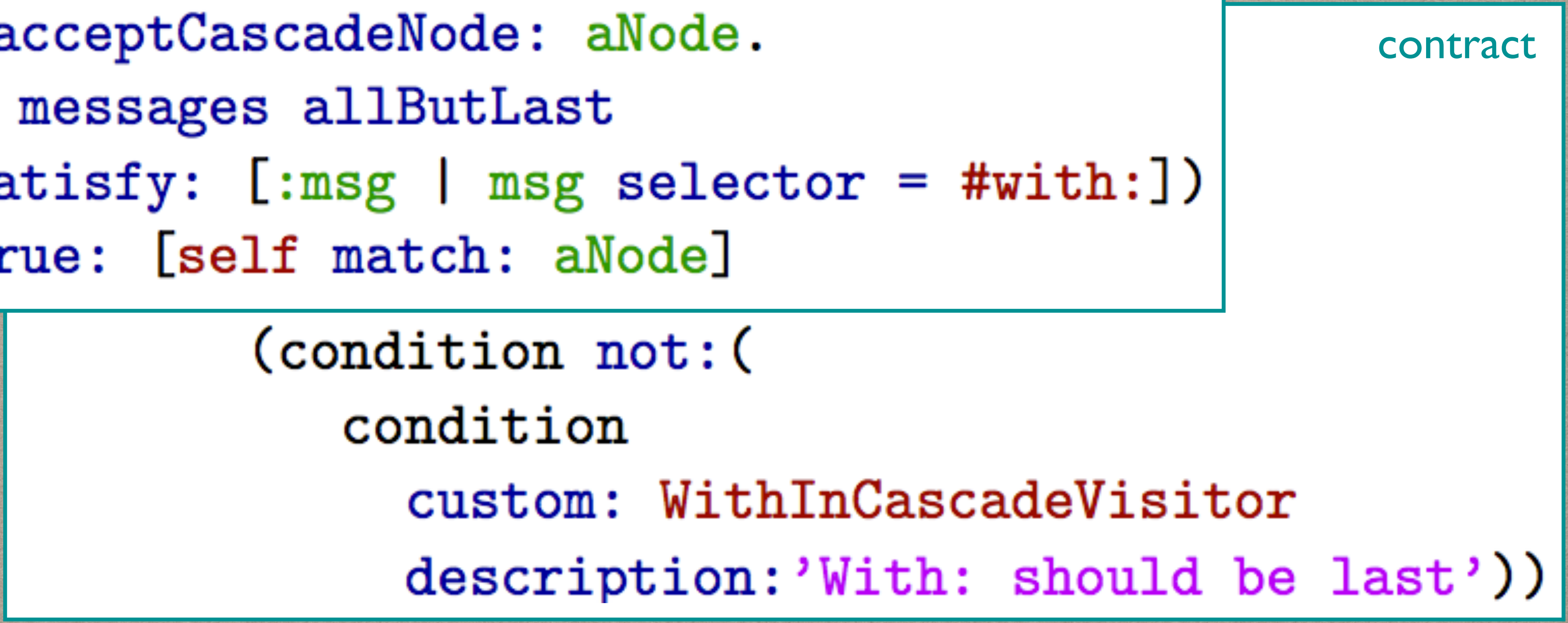
```
interfaceCode
                                                            '>
                                                          se*'>
```



contract

WithInCascadeVisitor extends CustomConditionVisitor

```
acceptCascadeNode: aNode
  super acceptCascadeNode: aNode.
  (aNode messages allButLast
    anySatisfy: [:msg | msg selector = #with:])
     ifTrue: [self match: aNode]
```



contract

```
        (condition not:(
           condition
             custom: WithInCascadeVisitor
             description:'With: should be last'))
```

INDUSTRIAL VALIDATION : RESULTS

Contract	Liable Methods	Exceptions	Errors December	Errors March
Private methods should not be called directly	7410	0	3	2
Marking dirty objects	333	5	7	2
Initialisation methods should start with super	44	0	1	0
Call ordering within method cascade	531	0	0	0

UCONTRACTS : CONCLUSION

- uContracts offer a simple unit-testing like way for letting programmers document and check conformance to structural source-code regularities
- using a "contract" metaphor
- focus on immediate feedback during development
- embedded DSL close to the programming language
- tight integration with the IDE
- Publication pending: A. Lozano, K. Mens, and A. Kellens, "Usage contracts: offering immediate feed- back on violations of structural source-code regularities". (submitted to SciCo)

FUTURE WORK

- More validation
- Improve / extend the DSL
- Port to most recent version of Pharo
- uContracts for other languages (e.g., Ruby)