Querying, visualizing models

ANNE ETIEN

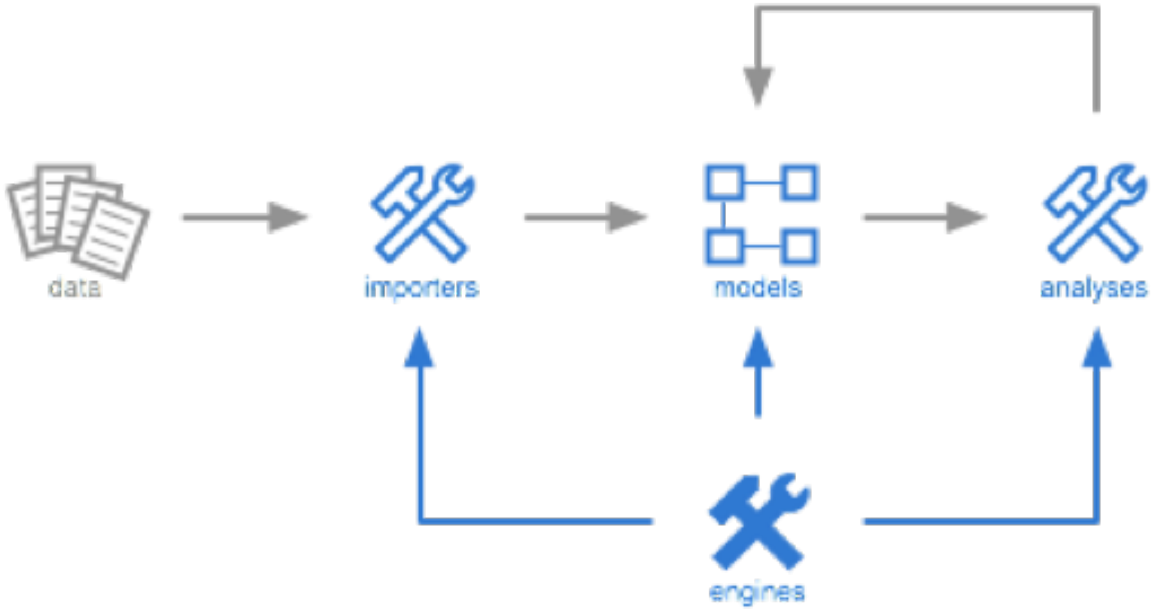
Overview

- FAMIX metamodel
- Access models
- Query
- Select
- Navigate

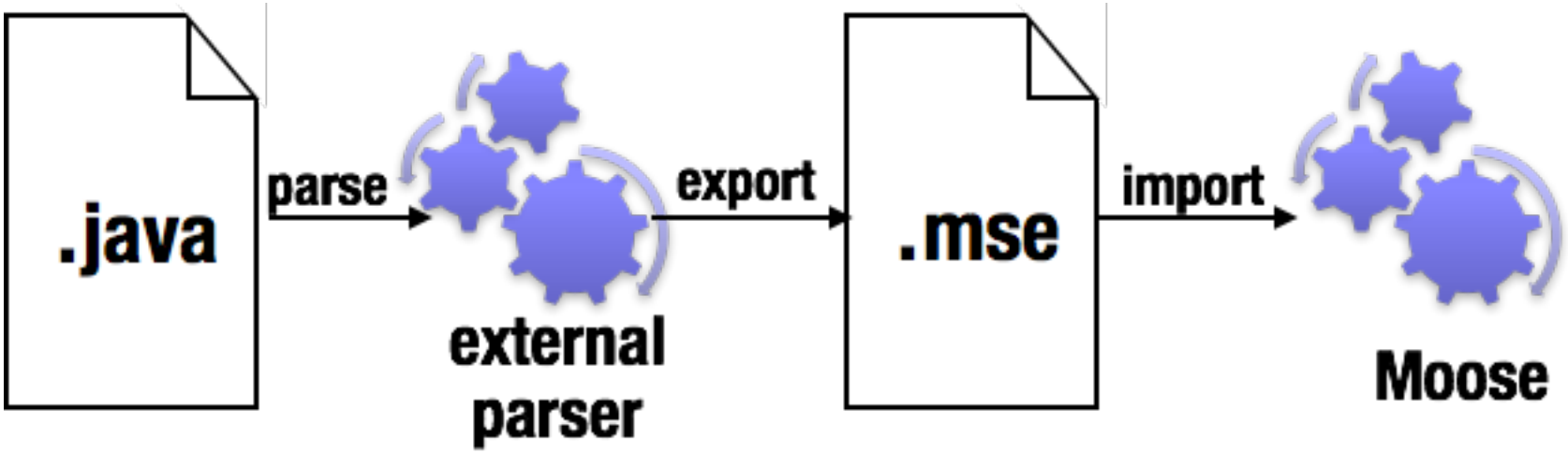
Moose

an extensible toolbox for software and data analysis

- Importers
- Model
- Visualizations
- Rules engines
- Etc…

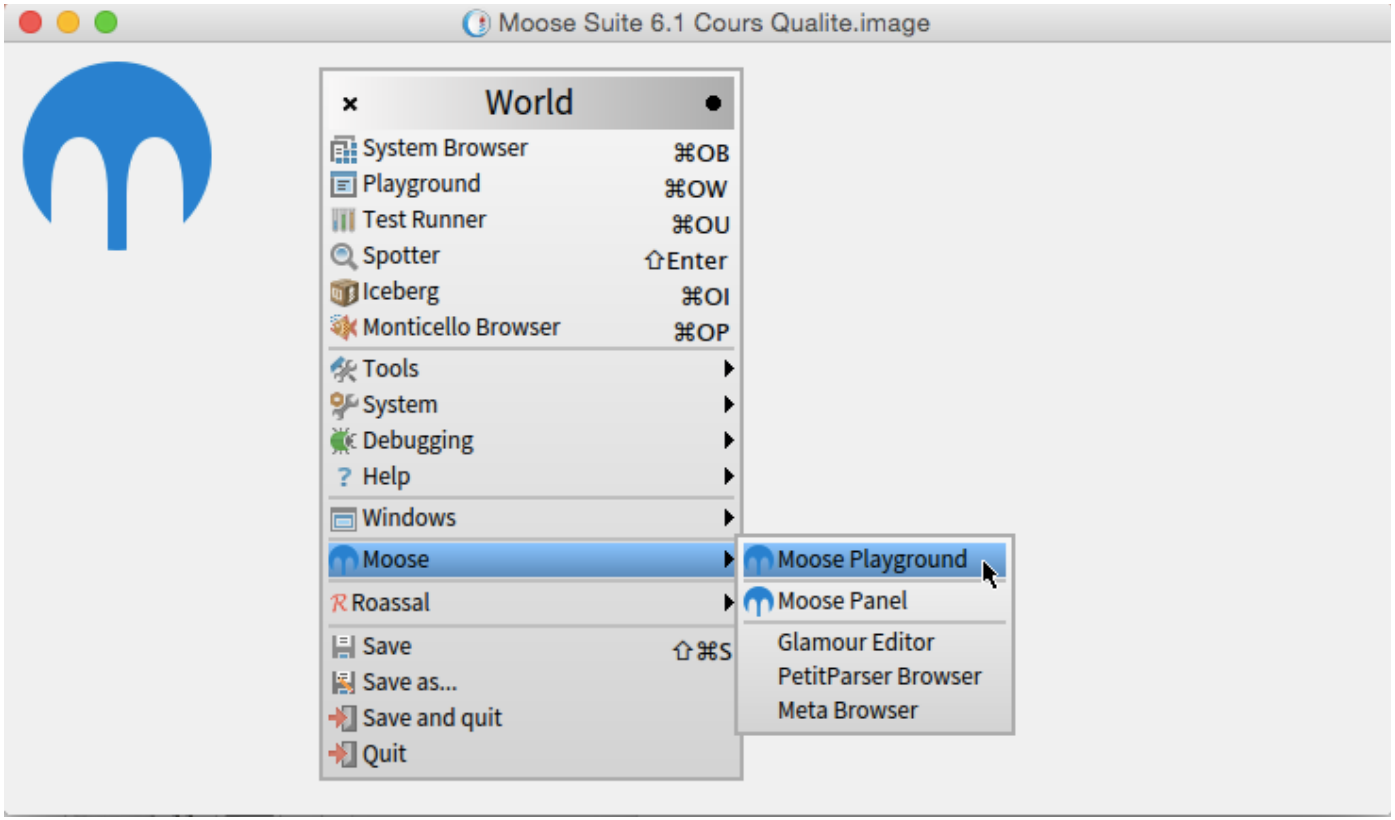


From Java code to Moose Model



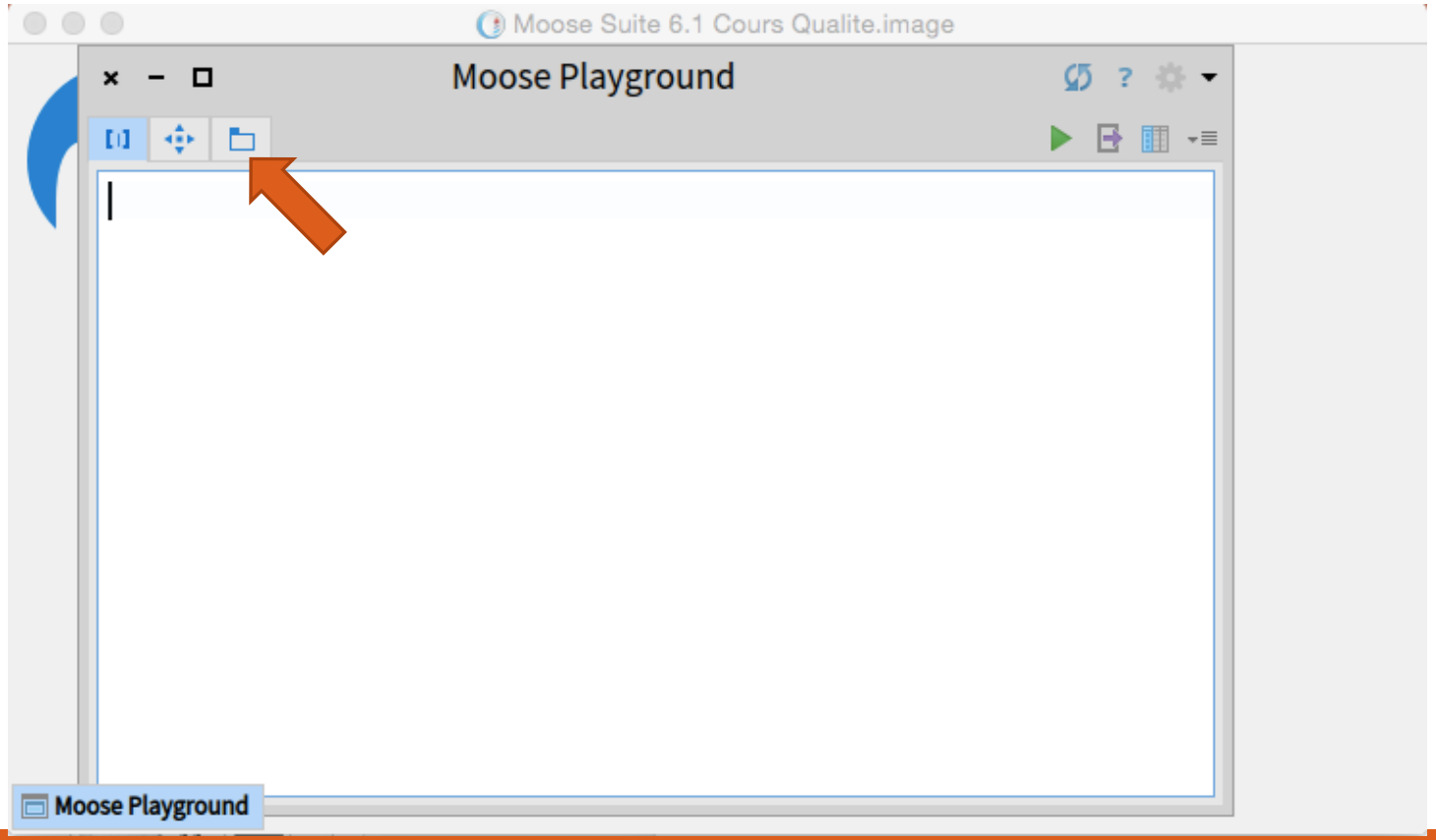
Import du Projet (Source)

- Click on Moose desktop



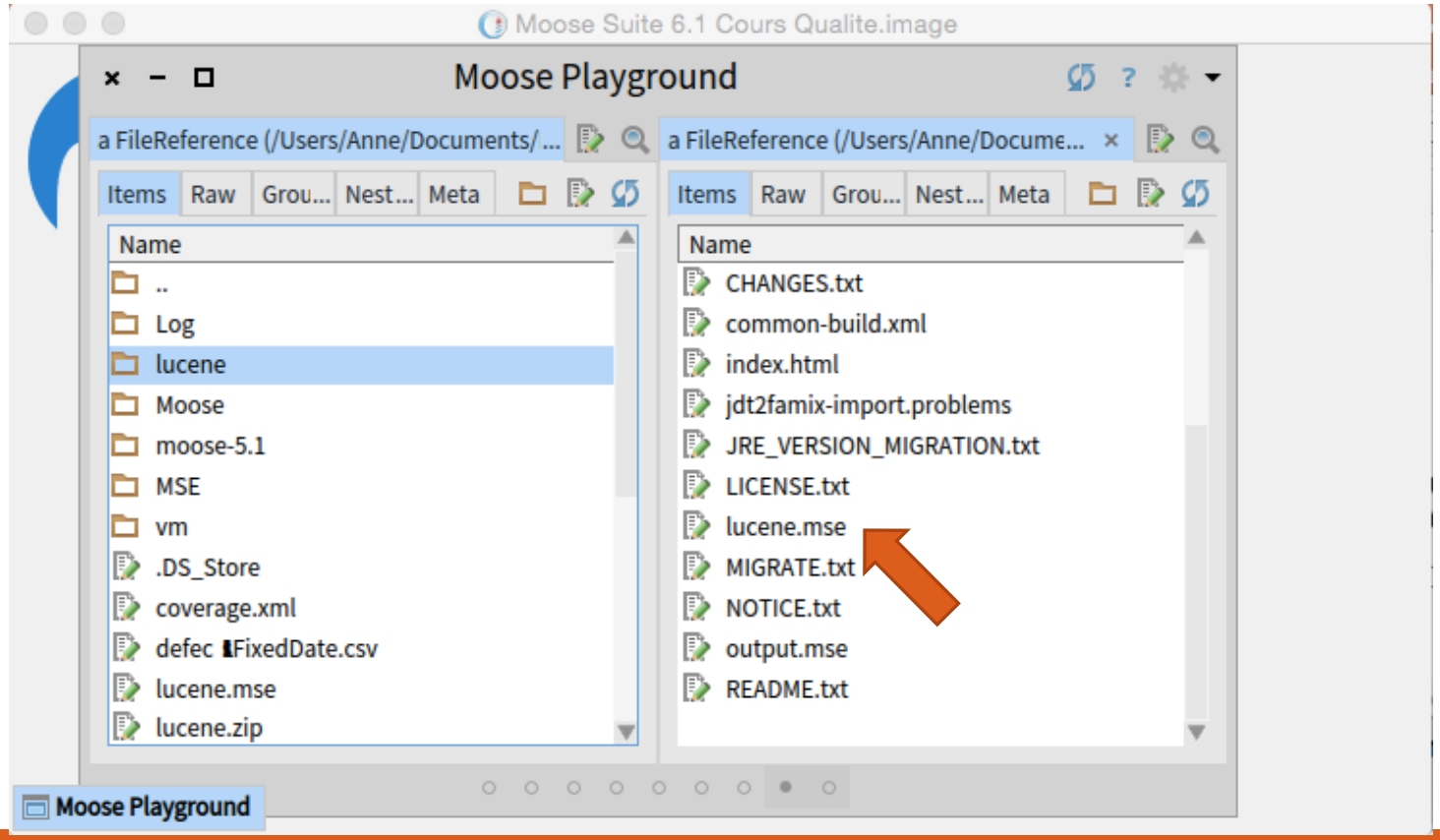
Import du Projet (Source)

- Import the model



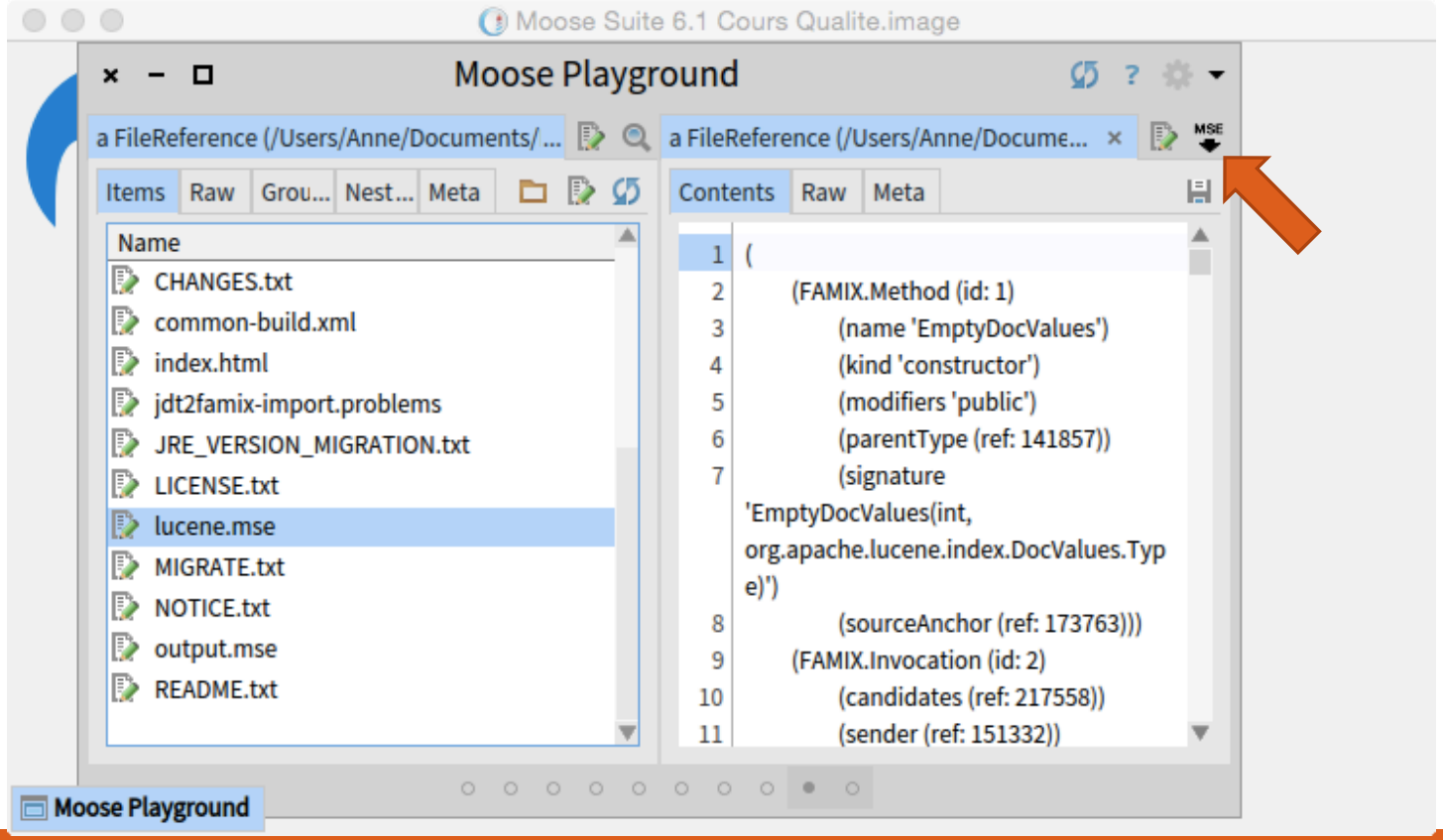
Import du Projet (Source)

- Import the model by selecting the mse file

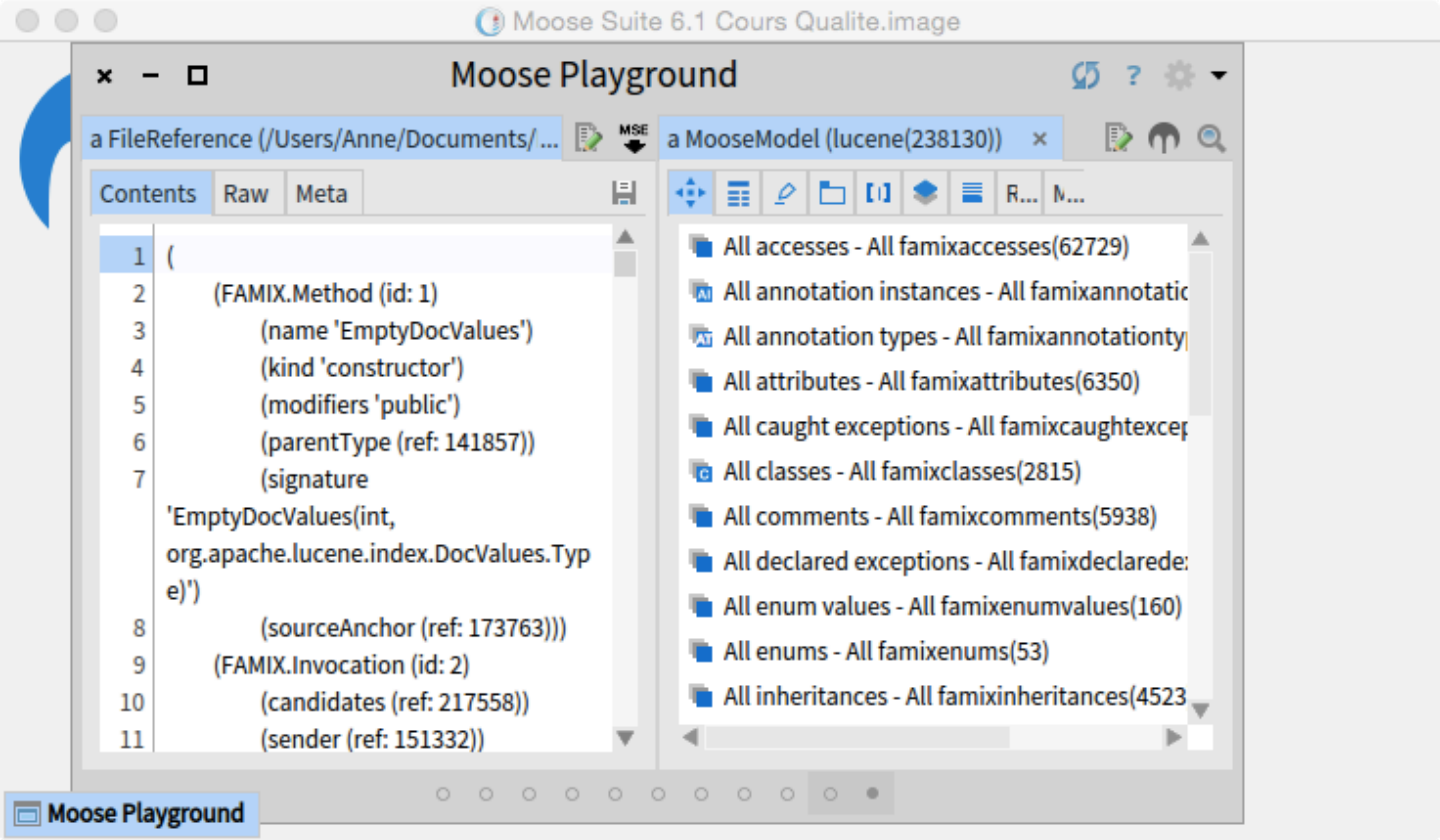


Import du Projet (Source)

- Import the model by clicking on the MSE button



Import du Projet (Source)



Pharo

Less is More

- No constructors
- No types declaration
- No interfaces
- No packages/private/protected
- No parametrized types
- No boxing/unboxing
- And really powerful

A Pure OO World

Only objects!

- mouse, booleans, arrays, numbers, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, stream…

3 kinds of messages

- Unary messages
  - Message without argument

```
5 factorial
Transcript cr
```

- Binary messages
  - Message with only one argument and is named by one or more symbol characters

```
3 + 4
```

- Keywords messages
  - Message with one or more arguments that are inserted in the message name

```
2 between: 0 and: 5
Transcript show: 'hello world'
```

Precedence

- Parentheses > unary > binary > keyword and finally from left to right.
- `(10 between: 1 and: 2 + 4 * 3) not`
- Here, the messages `+` and `*` are sent first then `between:and:` is sent, and finally `not`.
- The rule suffers no exception: operators are just binary messages with no notion of mathematical precedence, so
  - `2 + 4 * 3` reads left-to-right and gives 18,not 14!

From Java to Pharo

- `postman.send(mail,recipient);`
- `postman.send(mail,recipient);`
- `postman send mail recipient`
- `postman send mail to recipient`
- `postman send: mail to: recipient`

Highlighting Java syntax

Removing Java syntax

Adding small word to distinguish parameters

Pharo message with two parameters

Pharo Syntax

- Six reserved words only
  - `nil` the undefined object
  - `true, false` boolean objects
  - `self` the receiver of the current message (equivalent to `this` in Java)
  - `super` the receiver, in the superclass context
  - `thisContext` the current invocation on the call stack

Pharo Syntax

- Reserved punctuation characters
  - `"comment"`
  - `'string'`
  - `#symbol` unique string
  - `$a` the character a
  - `12 2r1100 16rC` twelve (decimal, binary, hexadecimal)
  - `3.14 1.2e3` floating-point numbers
  - `.` expression separator (period)
  - `;` message cascade (semicolon)
  - `:=` assignment
  - `^` return a result from a method (caret)
  - `[ :p | expr ]` code block with a parameter
  - `| foo bar |` declaration of two temporary variables
  - `#(abc 123)` literal array with the symbol #abc and the number 123
  - `{foo. 3 + 2}` dynamic array built from 2 expressions

Conditionals: ifTrue:ifFalse:

- Booleans are objects
- Conditional are messages sent to booleans or block

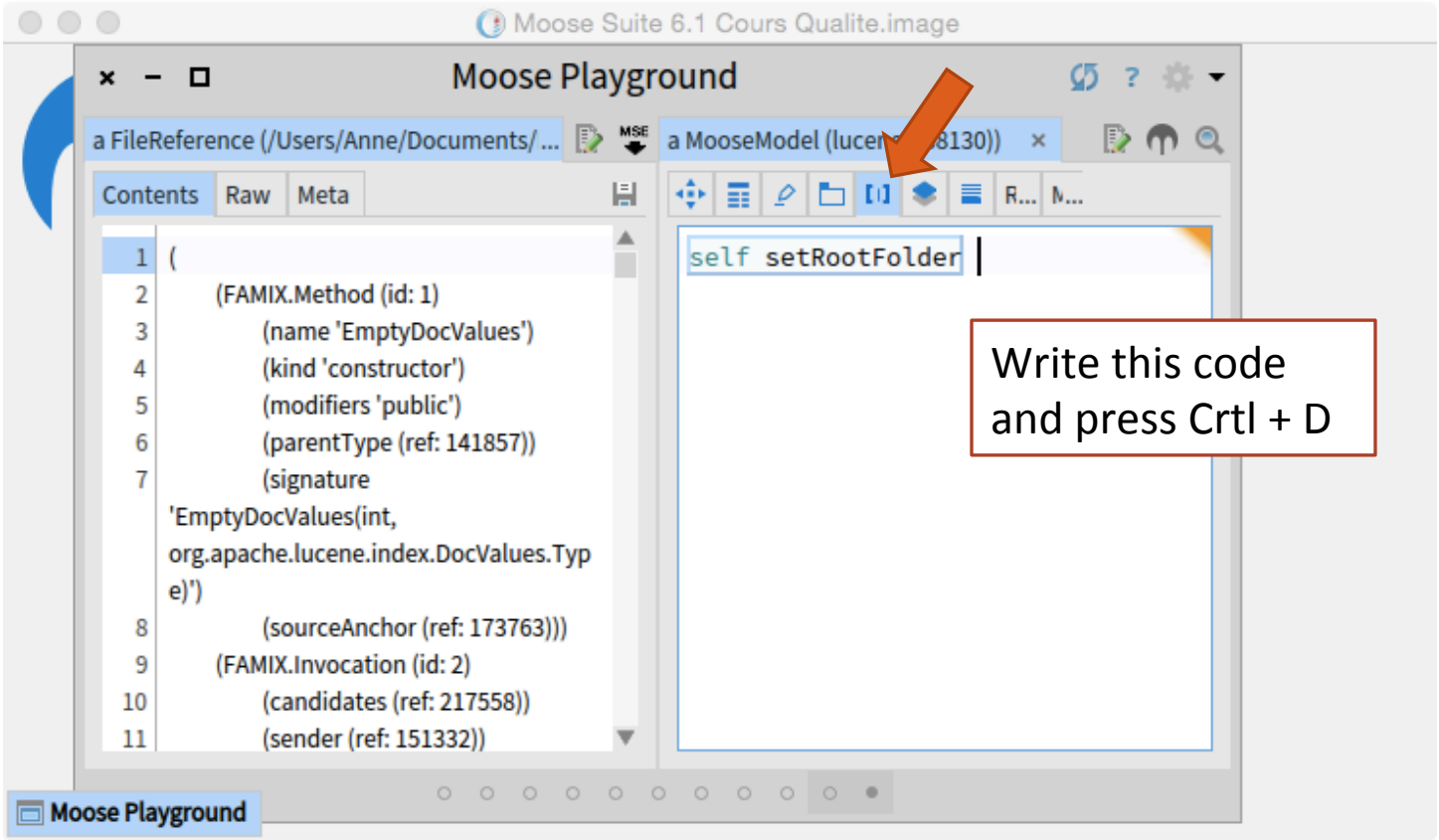
```
initialAnswer := fullName isEmptyOrNil

      ifTrue: ['FirstnameLastname' translated]
      ifFalse: [fullName].
```

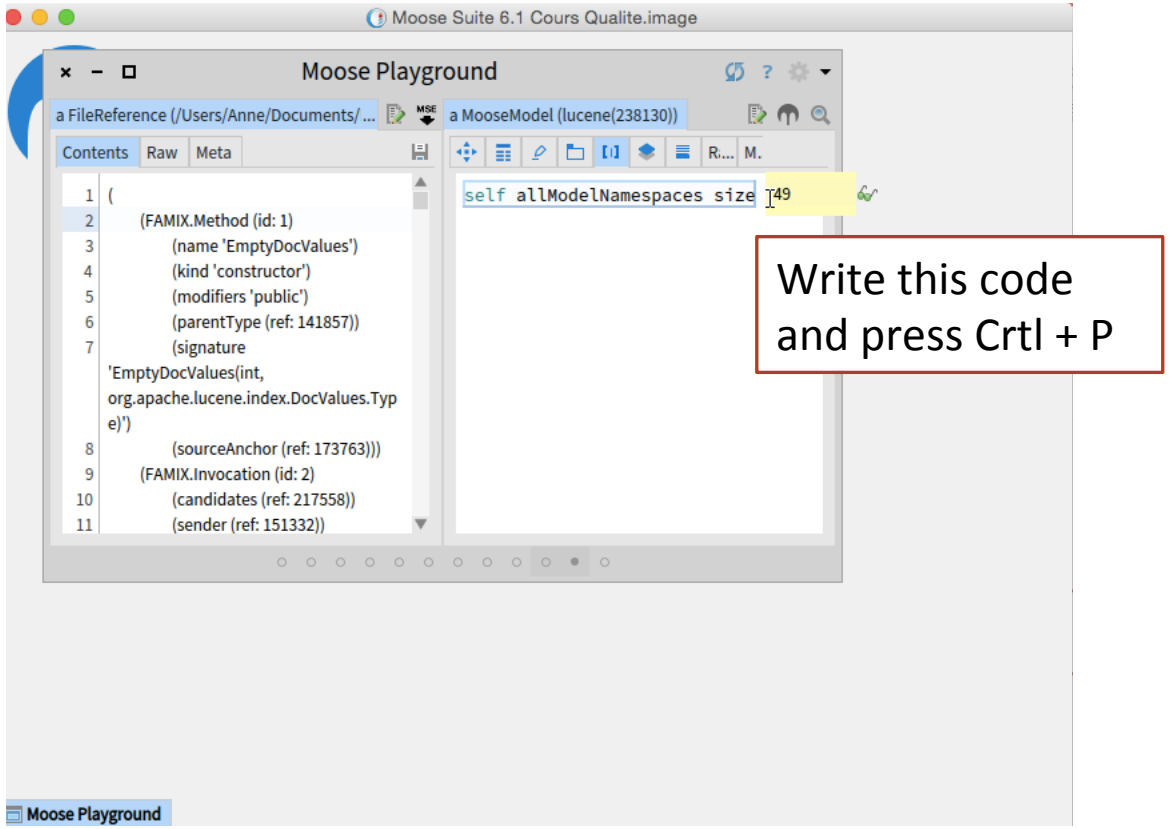
Getting a bit further

PERSONALIZED INFORMATION

Import du Projet (Source)

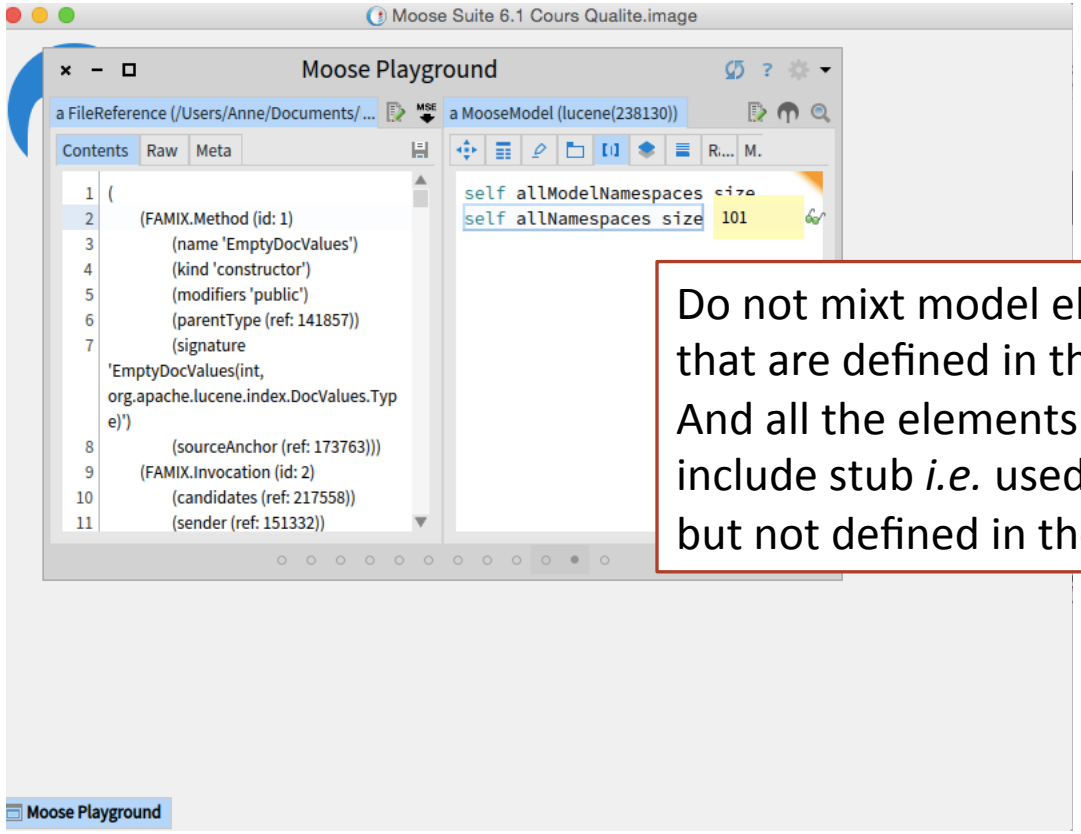


How many packages do we have in the model?



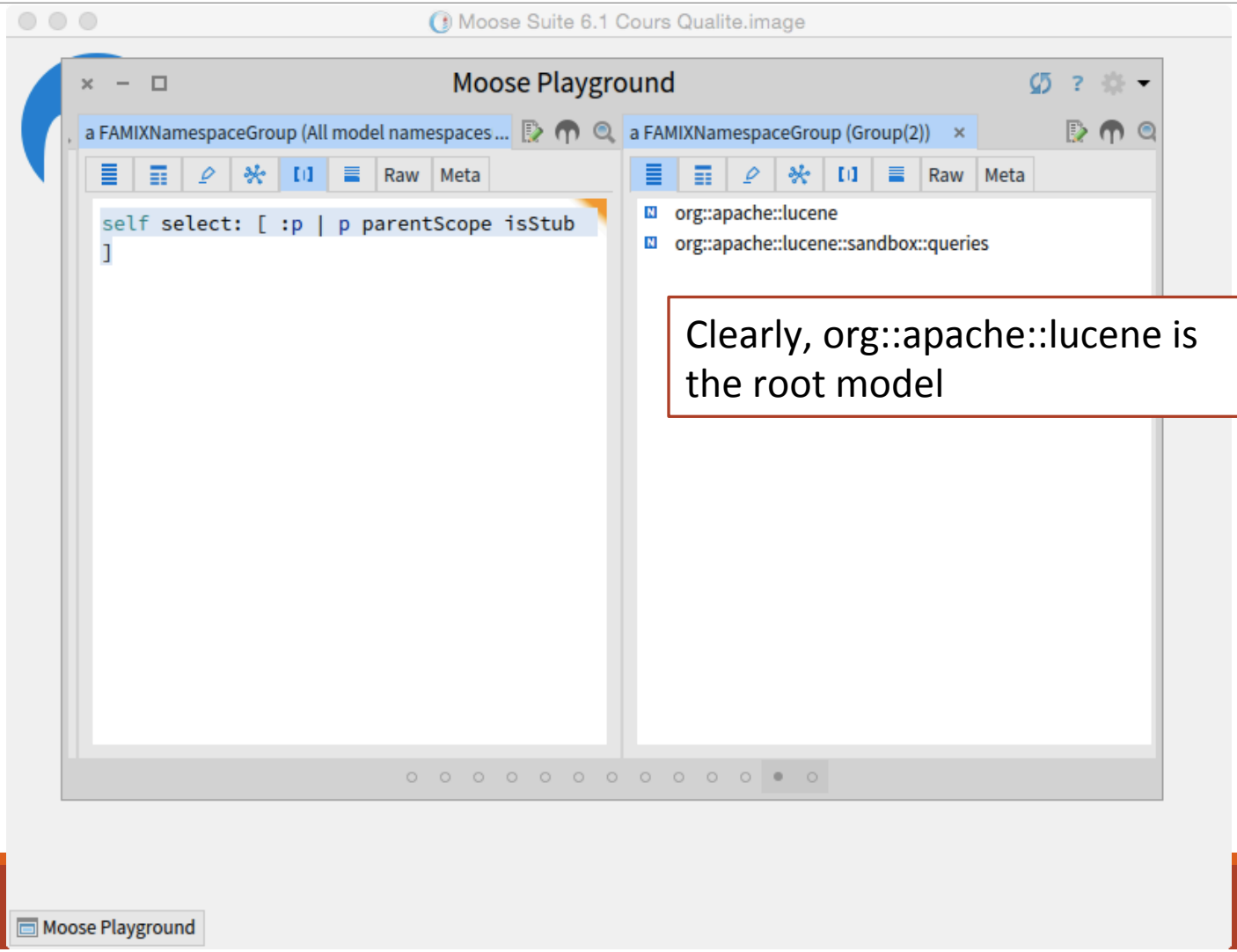
Write this code and press Crtl + P

How many packages do we have in the model?



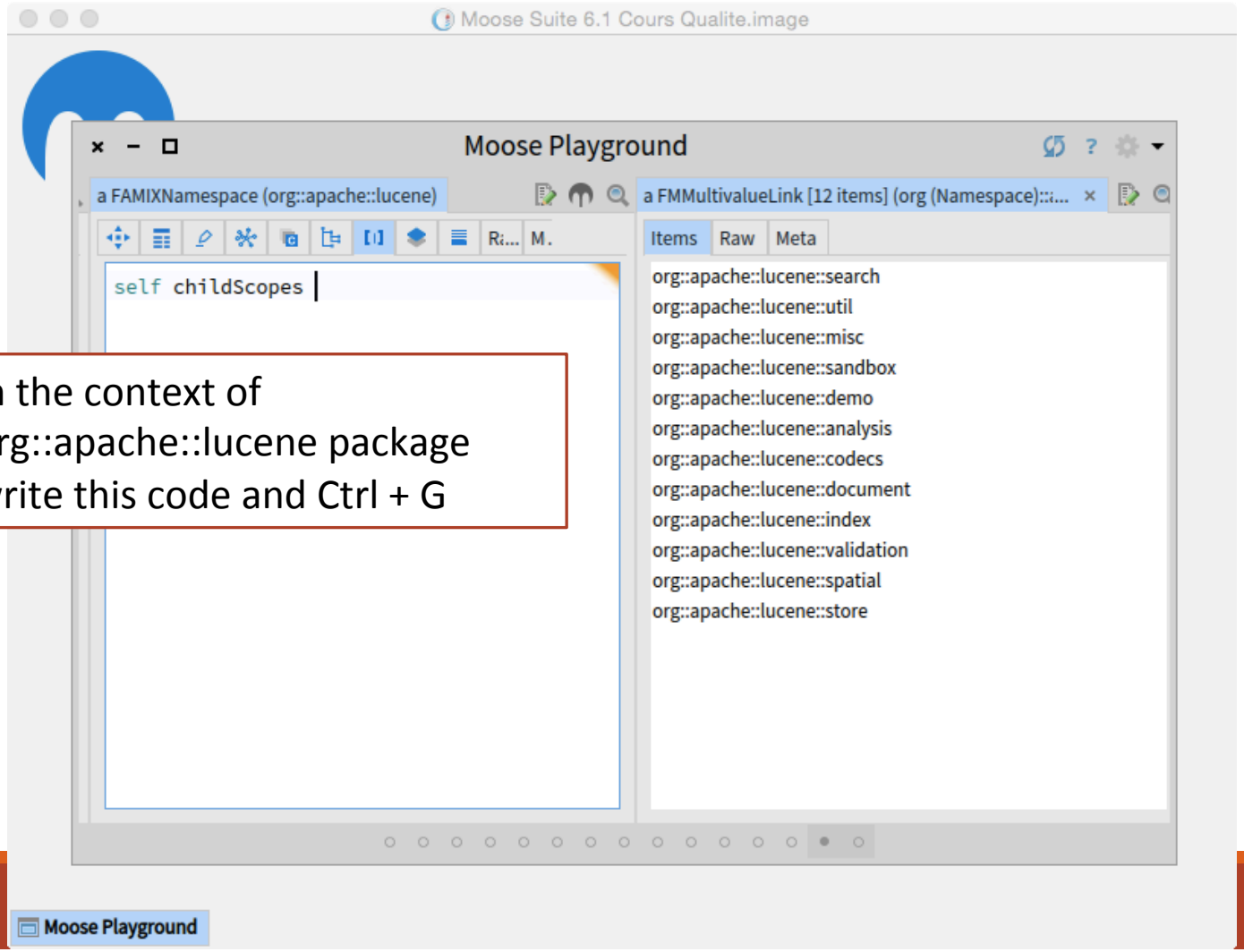
Do not mixt model elements that are defined in the model
And all the elements that also include stub *i.e.* used elements but not defined in the model

How many root packages do we have in the model?



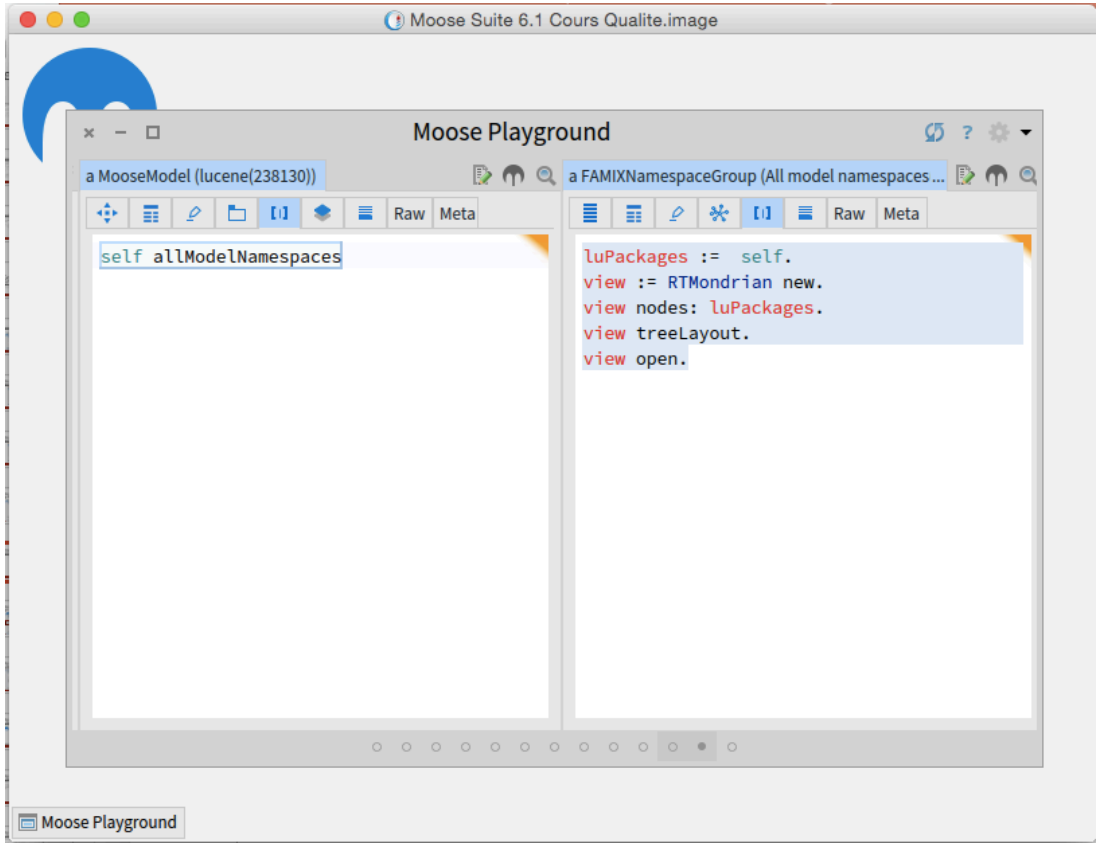
Clearly, org::apache::lucene is the root model

Which packages does the root package contain?

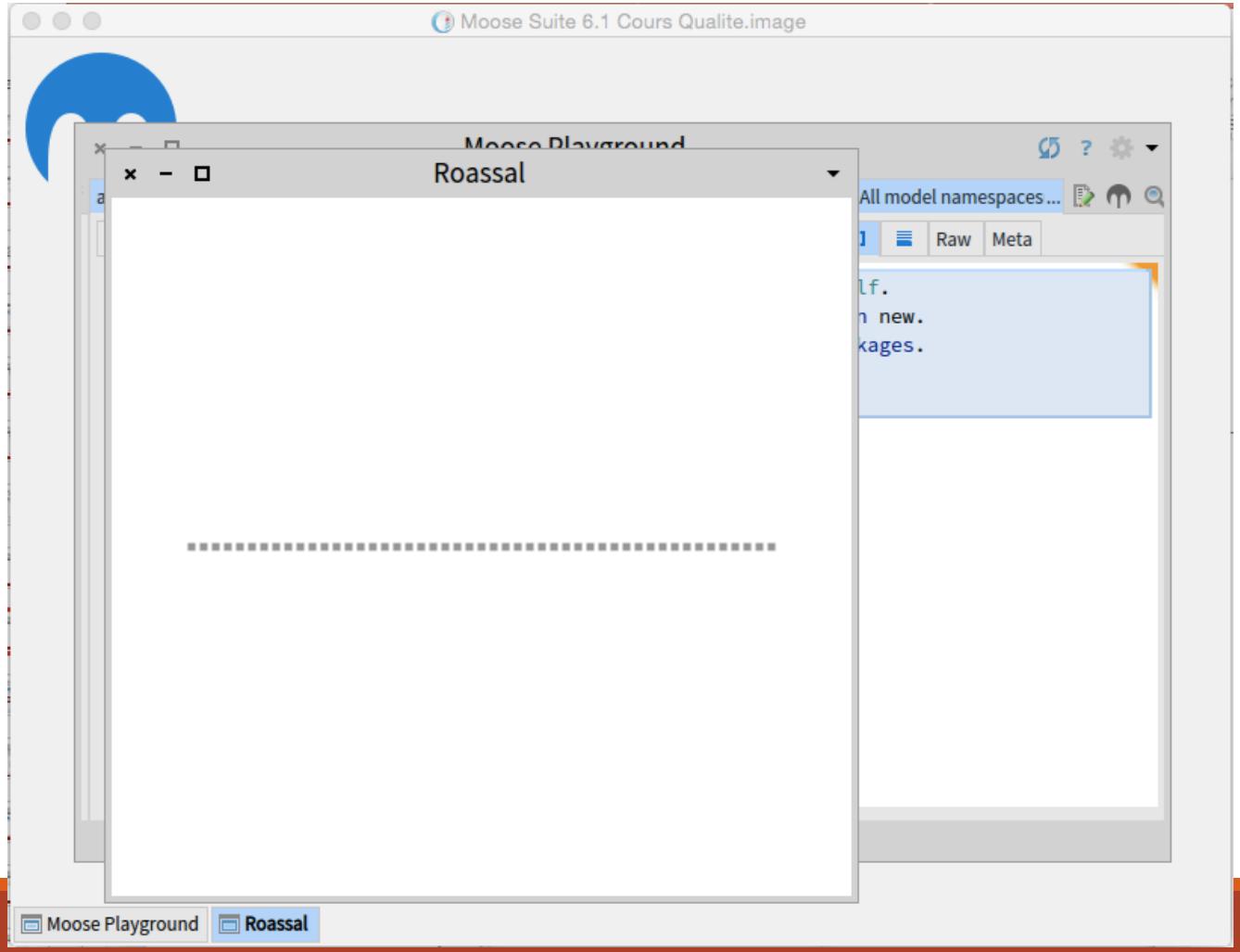


In the context of org::apache::lucene package write this code and Ctrl + G

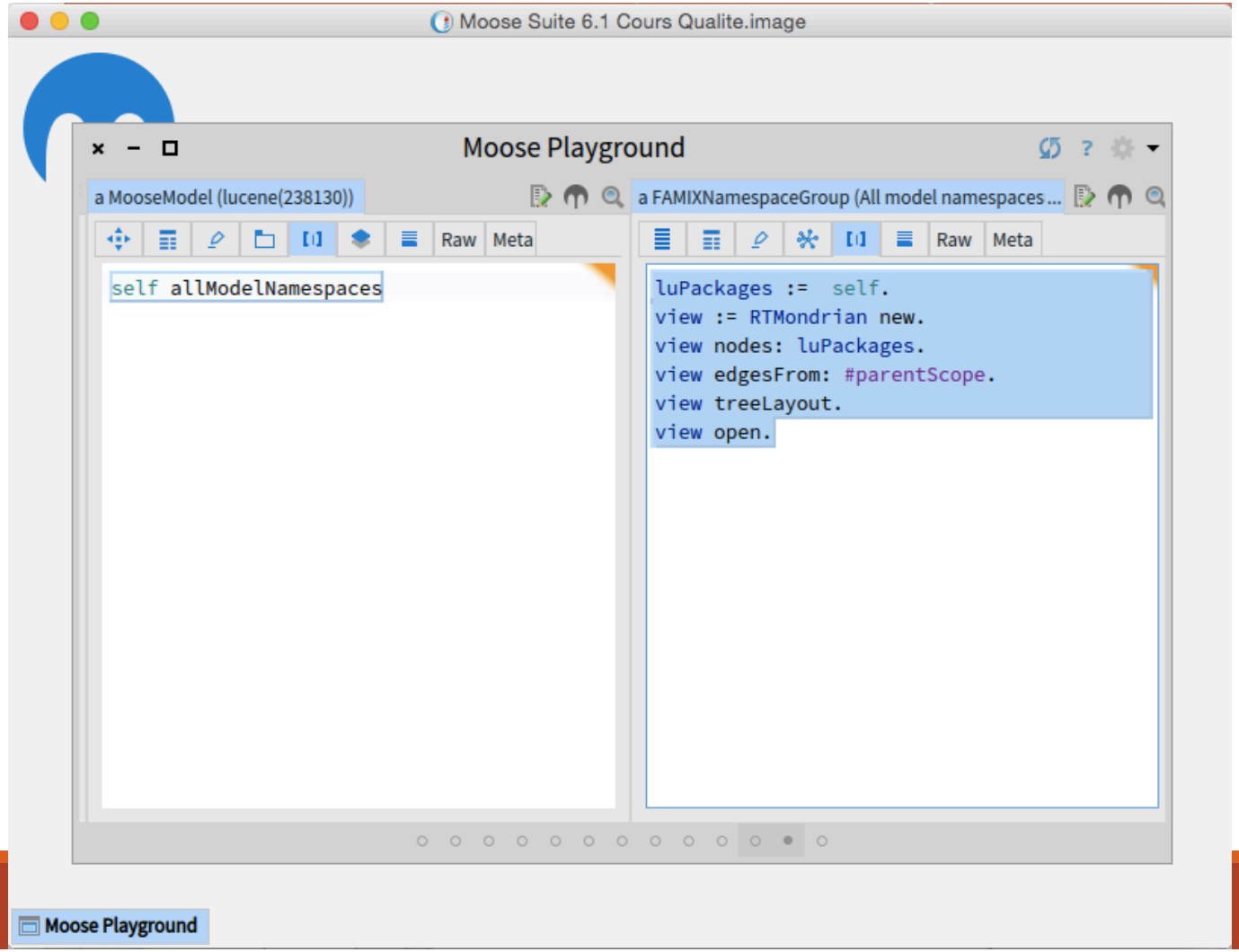
What is the package hierarchy?



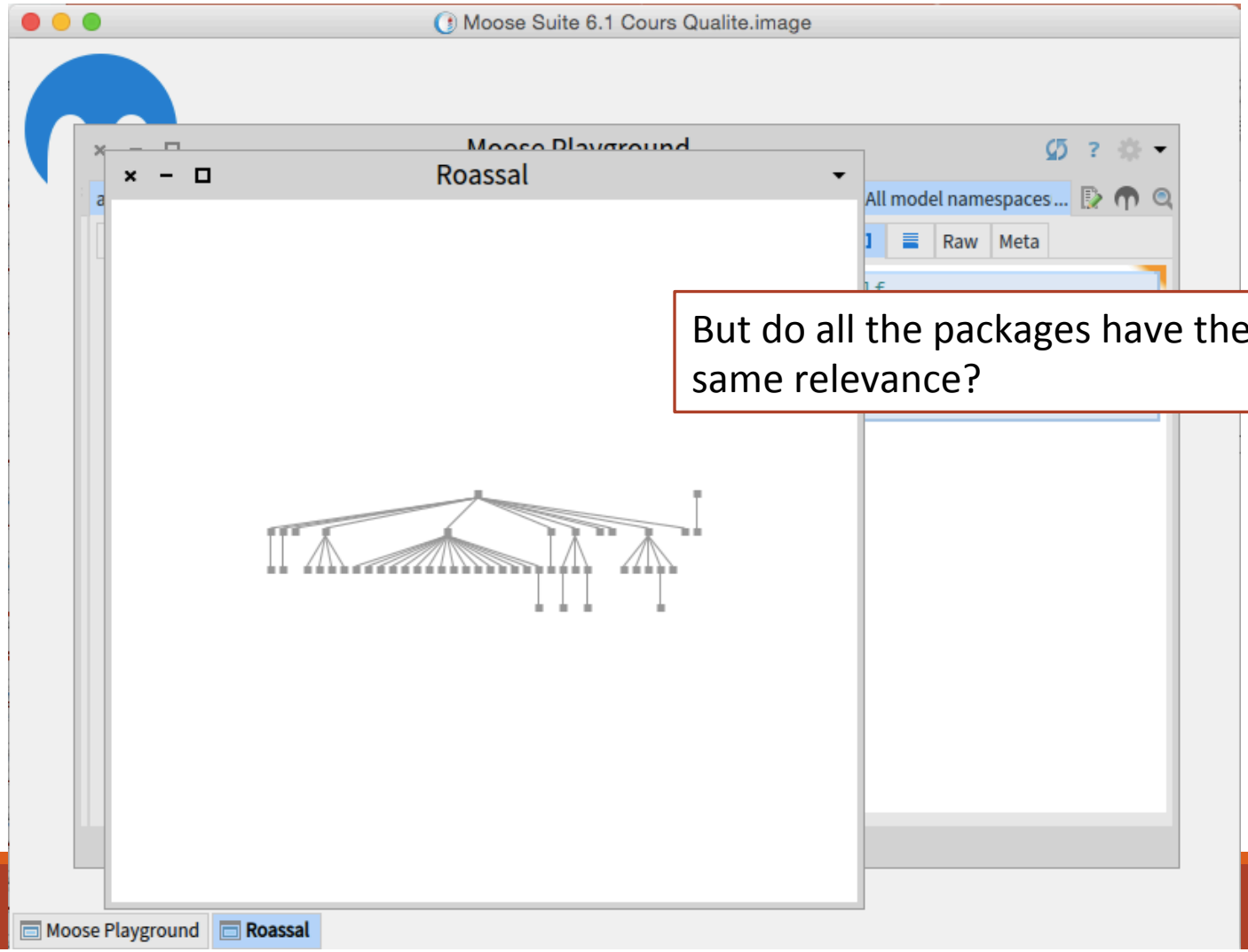
What is the package hierarchy?



What is the package hierarchy?

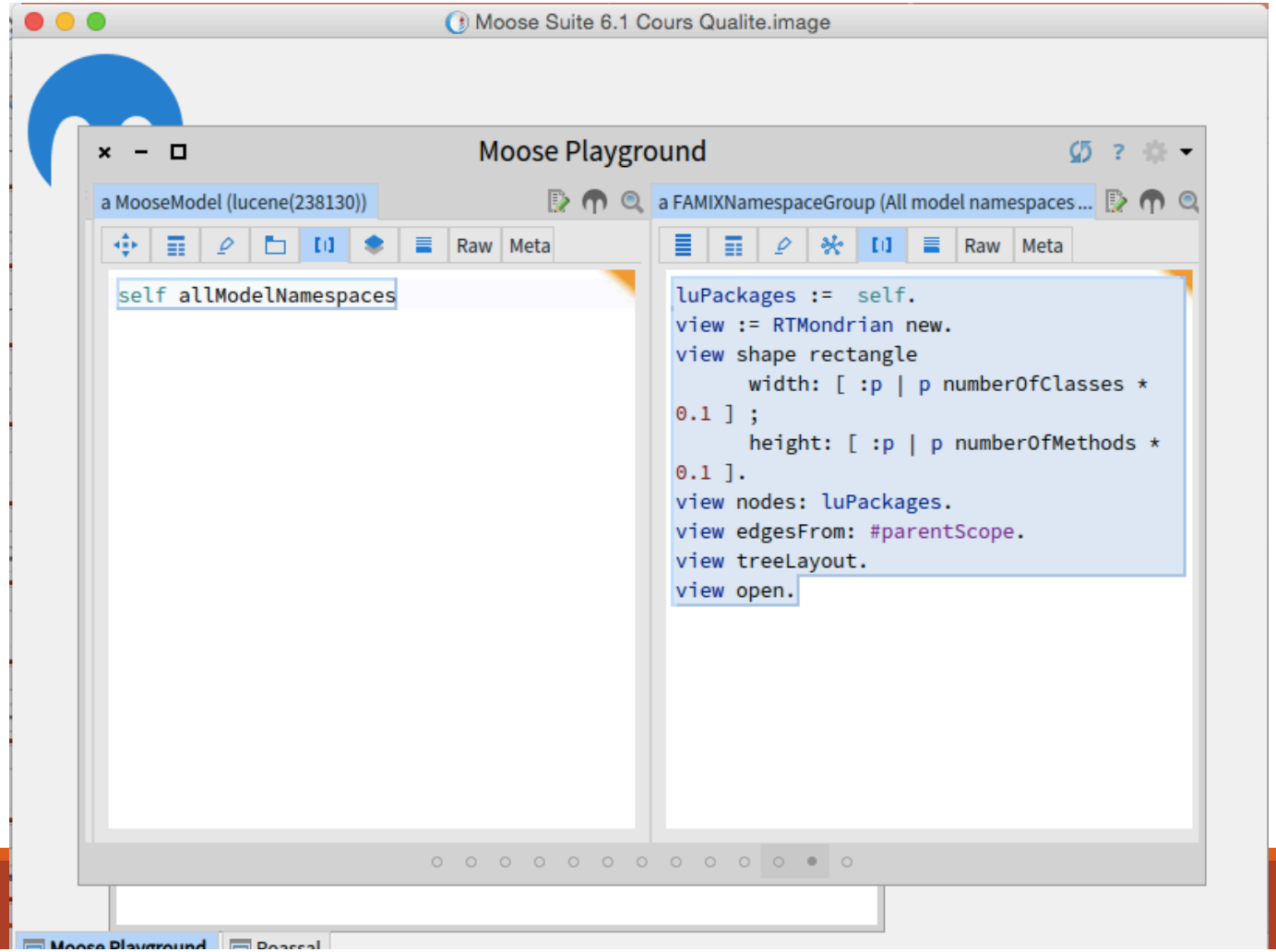


What is the package hierarchy?

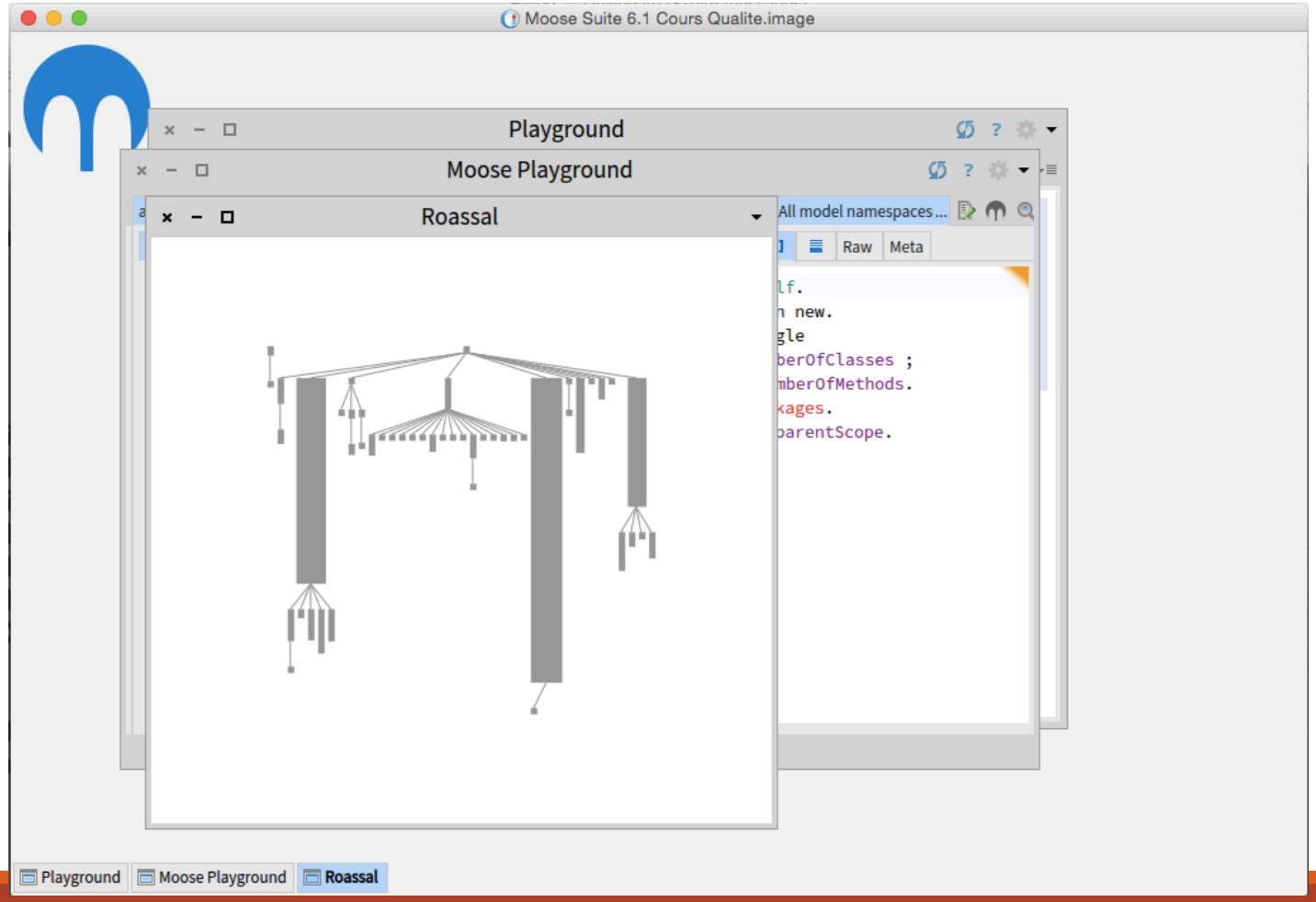


But do all the packages have the same relevance?

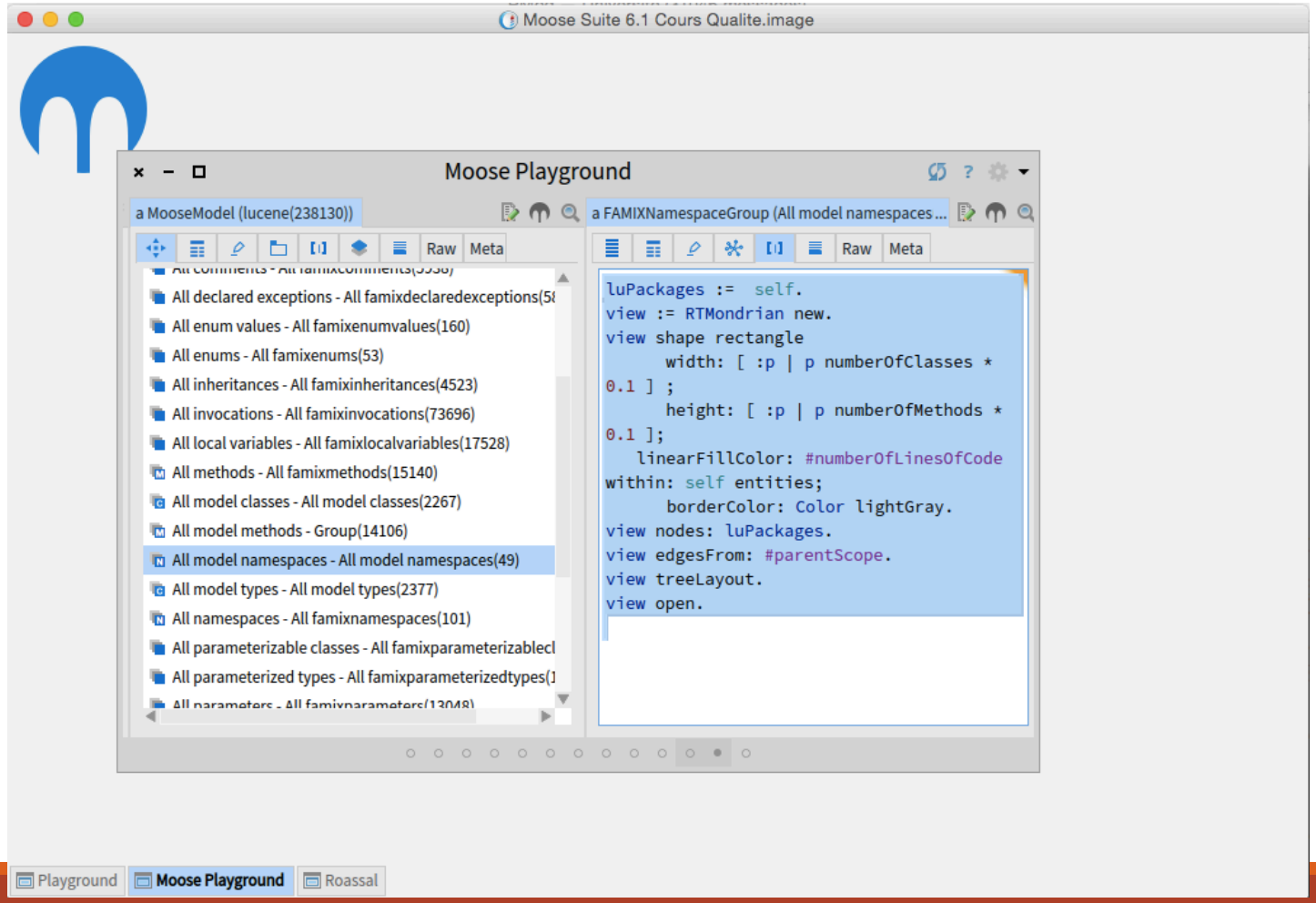
What is the package hierarchy?



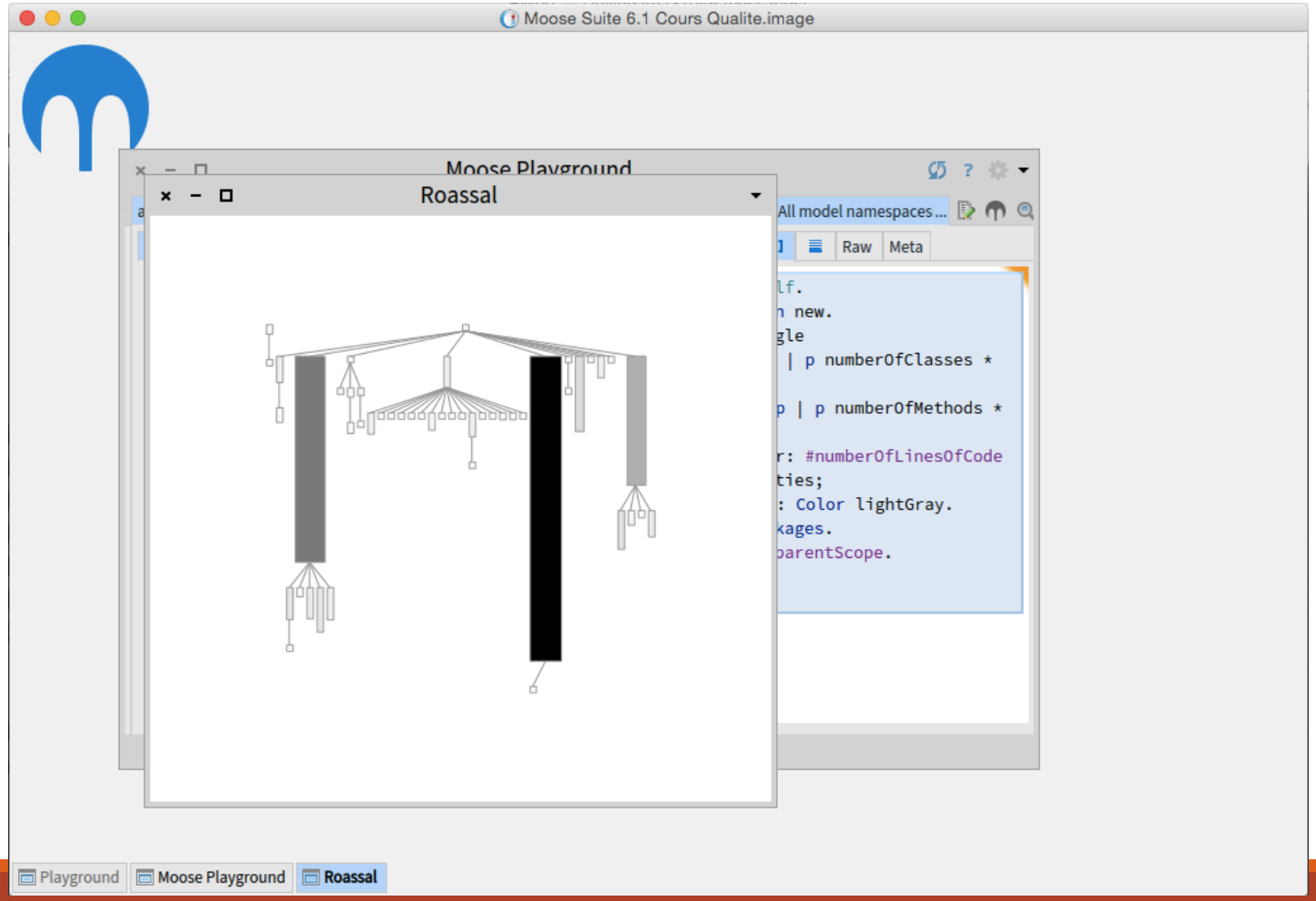
What is the package hierarchy?



What is the package hierarchy?



What is the package hierarchy?



Focus on external library

- What are the used external library?

```
externalLib := self allNamespaces select:
[:n | n isStub and: [ n classes
notEmpty ] ].
```

- What is the adherence with the systems?

```
self allNamespaces select: [:n | n isStub
and: [ n classes notEmpty ] ] thenCollect:
[ :n | n queryIncomingDependencies ]
```

Focus on classes

- Within the model context:
  - `self allClasses size` **2815**
  - `self allModelClasses size` **2267**
    - classes defined in the model vs stub
- Ordered descending classes by number of methods

```
self allModelClasses asOrderedCollection sorted:
[ :a :b | a numberOfMethods > b numberOfMethods ]
```

- and to see the values:

```
(self allModelClasses asOrderedCollection sorted:
[ :a :b | a numberOfMethods > b
numberOfMethods ]) collect: #numberOfMethods.
```

Focus on classes

- Mean number of methods per class:
  - `(self allModelClasses collect: [ :c | c numberOfMethods ]) asOrderedCollection average asFloat.` **6**
- Median number of methods per class:
  - `((self allModelClasses collect: [ :t | t numberOfMethods ]) asOrderedCollection sort: [ :a :b | a <b ]) median.` **4**

Compute the fraction

Focus on classes

- Mean number of lines of code per class:
  - `(self allModelClasses collect: [ :c | c numberOfLinesOfCode ]) asOrderedCollection average asFloat.` **77.2**
- Median number of lines of code per class
  - `((self allModelClasses collect: [ :t | t numberOfLinesOfCode ]) asOrderedCollection sort: [ :a :b | a <b ]) median.` **31**

Focus on classes – more detail

- Not anonymous classes without method:
  - `(self allModelClasses select: [ :c | c isAnonymousClass not and: [ c numberOfMethods = 0 ] ] )`
    - 5 classes to analyse
- Not anonymous classes without line of code:
  - `(self allModelClasses select: [ :c | c isAnonymousClass not and: [ c numberOfLinesOfCode = 0 ] ] )`
    - 5 classes to analyse
  - The same classes? **Yes**
    - Relevance of these classes

Focus on classes - « god » classes

- Instead of putting an arbitrary value, you can play with the previous collection and see how many methods has the 10th, 50th or 100th biggest classes.

```
((self allModelClasses asOrderedCollection
   sorted: [ :a :b |
     a numberOfMethods > b numberOfMethods
  ]) collect: #numberOfMethods) at: 10
```

Focus on classes - « god » classes

- (contain more than 50 methods)

```
(self allModelClasses select: [ :each | each
numberOfMethods > 50 ]) size
```

6

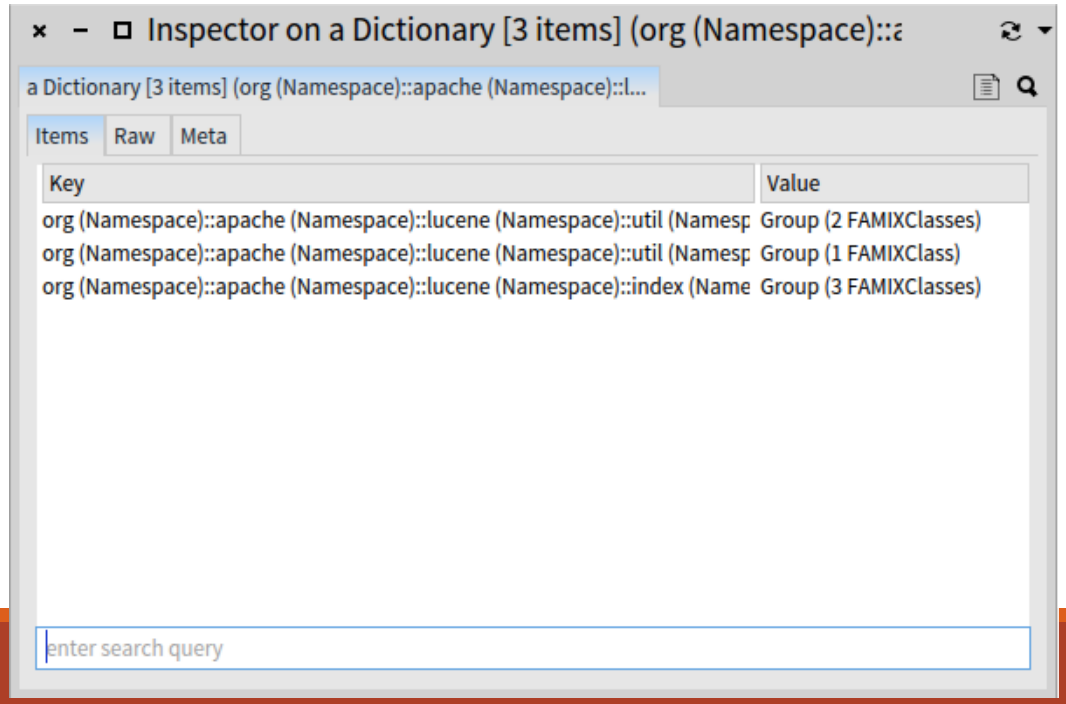
- Inspect the collection to see its elements (Ctrl + I or Ctrl + G)

```
(self allModelClasses select: [ :each | each
numberOfMethods > 50 ])
```

Focus on classes - « god » classes

- Where are the largest classes located?

```
(self allModelClasses select: [ :each | each
numberOfMethods > 50 ])
groupedBy: #belongsTo
```



Focus on classes - « god » classes

- Another definition (contain more than 500 lines of code)

```
(luClasses select: [ :each | each
numberOfLinesOfCode > 500 ]) size
```

19

Focus on classes – annotations

- What are the classes with an annotation?

```
self allModelClasses select:
[ :t | t annotationInstances notEmpty ]
```

What are these annotations?

```
((self allModelClasses select:
[ :t | t annotationInstances notEmpty ])
collect: #annotationTypes)
asOrderedCollection flattened asBag
```

Gives the number of occurrences for each different value

Focus on deprecated classes

- What are the deprecated classes?

```
deprecatedClasses := (self
allModelClasses select: [ :t | t
annotationInstances notEmpty

and: [ t annotationTypes

    anySatisfy: [ :a | a name =
            'Deprecated' ] ] ])
```

Focus on deprecated classes

- Are these deprecated classes referenced?

```
deprecatedClasses select: [ :c | c
incomingReferences notEmpty ]
```

None

- Are their methods invoked?

```
deprecatedClasses select: [:c | c
queryAllIncomingInvocations notEmpty]
```

Yes for all

Focus on methods

- Within the model context:
  - `self allMethods size` **8871**
  - `self allModelMethods size` **8263**
    - Methods defined in the model vs stub
- Ordered descending methods by number of lines of code

```
self allModelMethods asOrderedCollection
sorted: [ :a :b | a numberOfLinesOfCode > b
numberOfLinesOfCode ]
```

- and to see the values:

```
(self allModelMethods asOrderedCollection
sorted: [ :a :b | a numberOfLinesOfCode > b
numberOfLinesOfCode ]) collect:
#numberOfLinesOfCode.
```

Focus on methods – Compléxité cyclomatique

- Ordered descending methods by number of lines of code

```
self allModelMethods asOrderedCollection
sorted: [ :a :b | a cyclomaticComplexity >
b cyclomaticComplexity ].
```

- and to see the values:

```
(self allModelMethods asOrderedCollection
sorted: [ :a :b | a cyclomaticComplexity >
b cyclomaticComplexity ].) collect:
#numberOfLinesOfCode.
```

Max = 83!!!
170 methods have a cc ≥ 10

Focus on methods – annotations

- What are the methods with an annotation?

```
self allModelMethods select: [ :m | m
annotationInstances notEmpty ]
```

- What are these annotations?

```
((self allModelMethods select: [ :m | m
annotationInstances notEmpty ])collect:
#annotationTypes) asOrderedCollection
flattened asBag.
```

Focus on deprecated methods

- What are the deprecated methods?

```
deprecatedMethods := (self allModelMethods
select: [ :m | m annotationInstances
notEmpty
and: [ m annotationTypes
     anySatisfy: [ :a | a name =
              'Deprecated' ] ] ]).
```

- Are those methods invoked?

```
deprecatedMethods select: [:m | m
incomingInvocations notEmpty].
```

Yes for some

Focus on test methods

- What are the tests methods?

```
testMethods := (self allModelMethods
select: [ :m | m annotationInstances
notEmpty
and: [ m annotationTypes
     anySatisfy: [ :a | a name =
              'Test' ] ] ]).
```

86 methods to analyse

- Test methods not including assertion

```
testMethods reject: [ :m | m
invokedMethods anySatisfy: [ :n | n name
beginsWith: 'assert' ] ]
```

Focus on methods - comments

- Which methods do not get comments?

```
self allModelMethods select: [ :m | m
comments isEmpty ]
```

- Which methods without Test in its name or its class name without comment

6910 methods

```
self allModelMethods select: [ :m | (m
mooseName includesSubstring: 'Test')
not and: [m comments isEmpty ]].
```

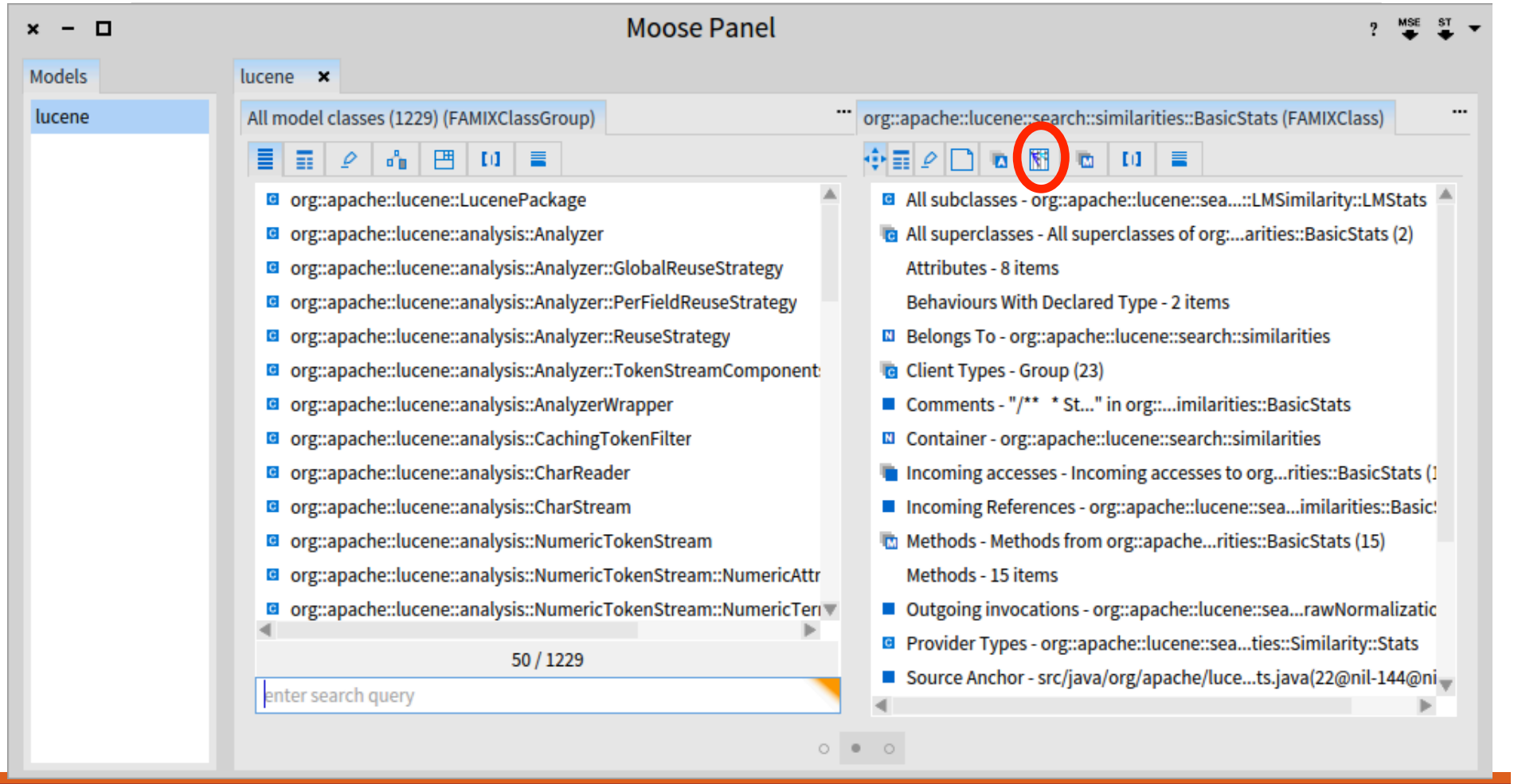
Focus on methods – main

- Identifying the `main`() method

```
self allModelMethods select: [ :m | (m
name = 'main')].
```

14 methods!!
What are the starting point?

Global view of a class



Global view of a class: Blueprint