Querying, visualizing models

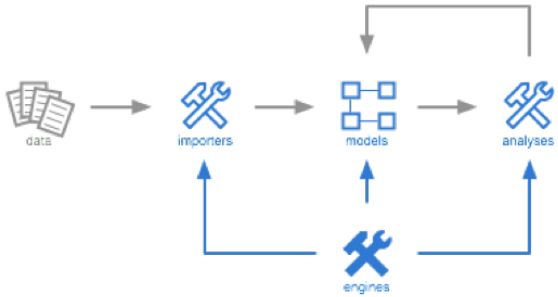
Overview

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

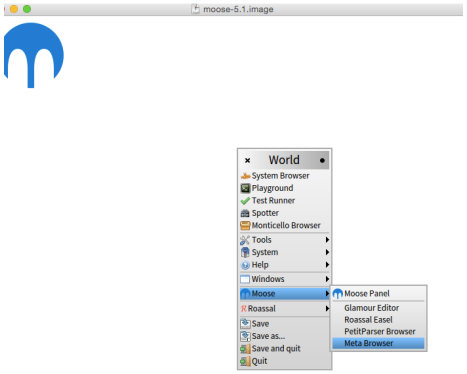
Moose

an extensible toolbox for software and data analysis

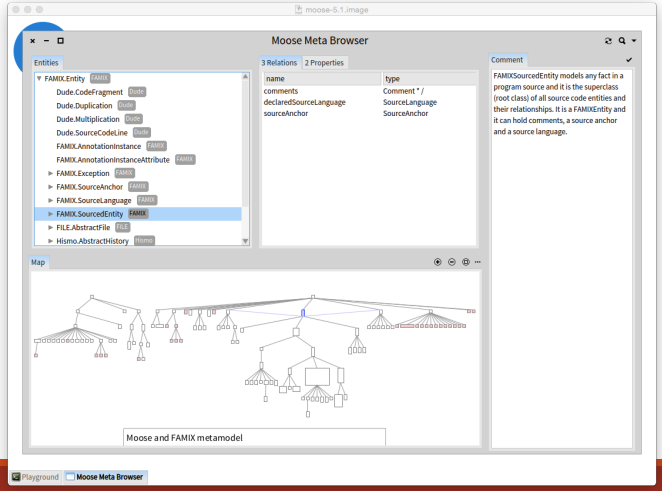
- Importers
- Model
- Visualizations
- Rules engines
- Etc…



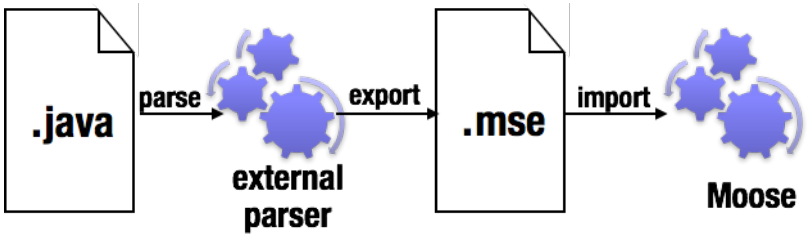
Meta browser



Meta browser

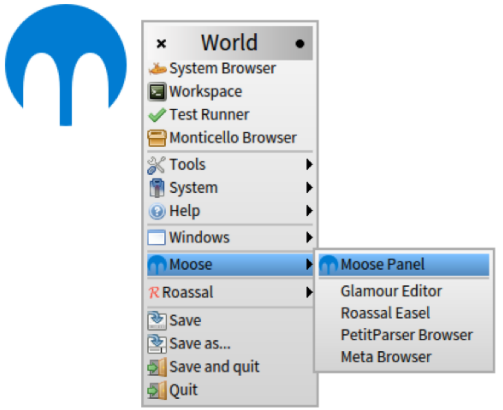


From Java code to Moose Model



Import du Projet (Source)

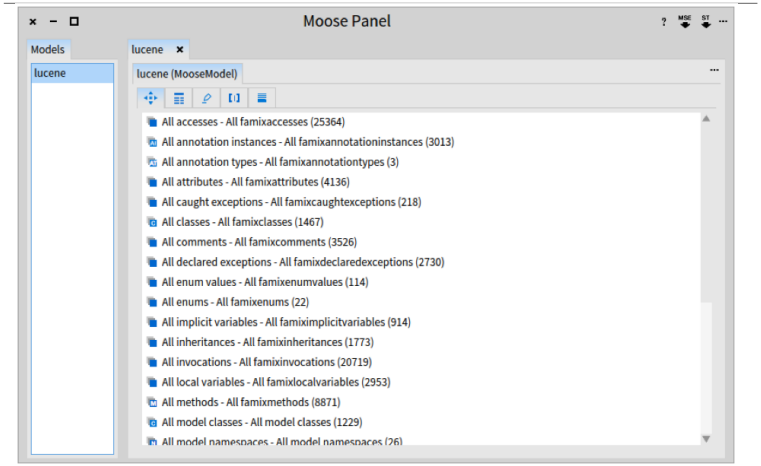
- Click on Moose desktop



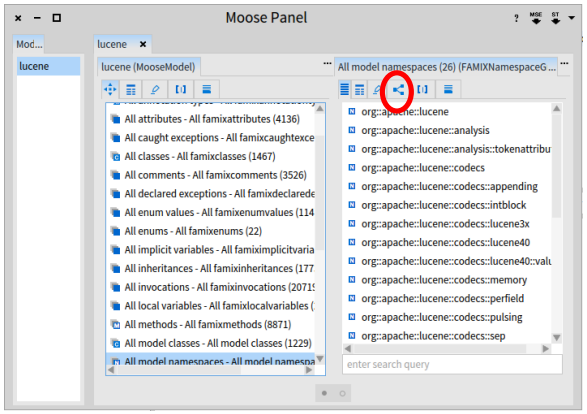
Import du Projet (Source)

- Import the model through the .mse

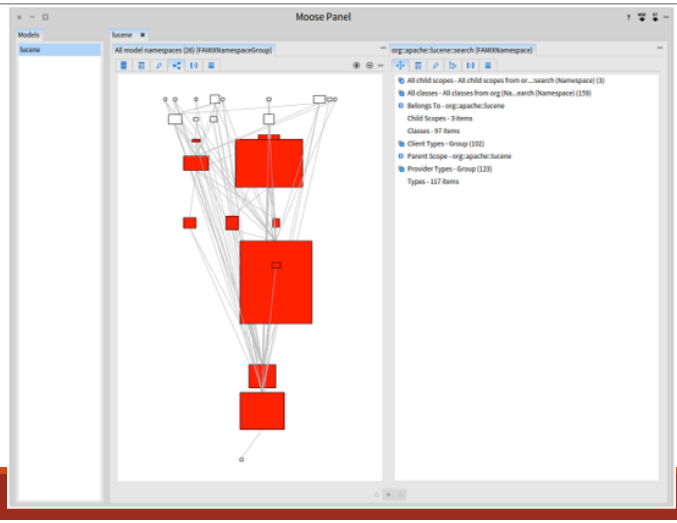
Import du Projet (Source)



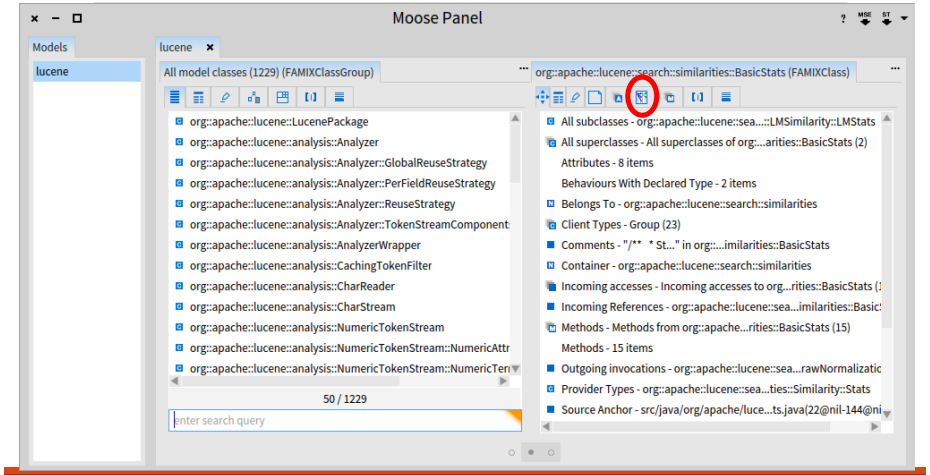
Links between packages: Cycle Map



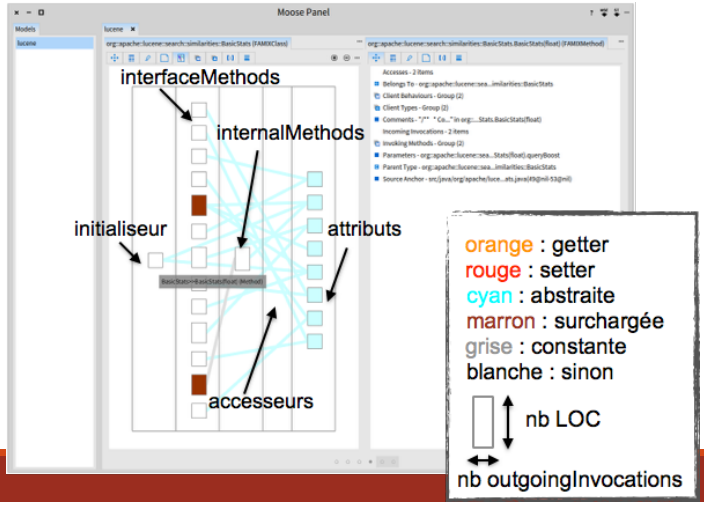
Links between packages: Cycle Map



Global view of a class



Global view of a class: Blueprint



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);` — Highlighting Java syntax
- `postman send mail recipient` — Removing Java syntax
- `postman send mail to recipient` — Adding small word to distinguish parameters
- `postman send: mail to: recipient` — 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

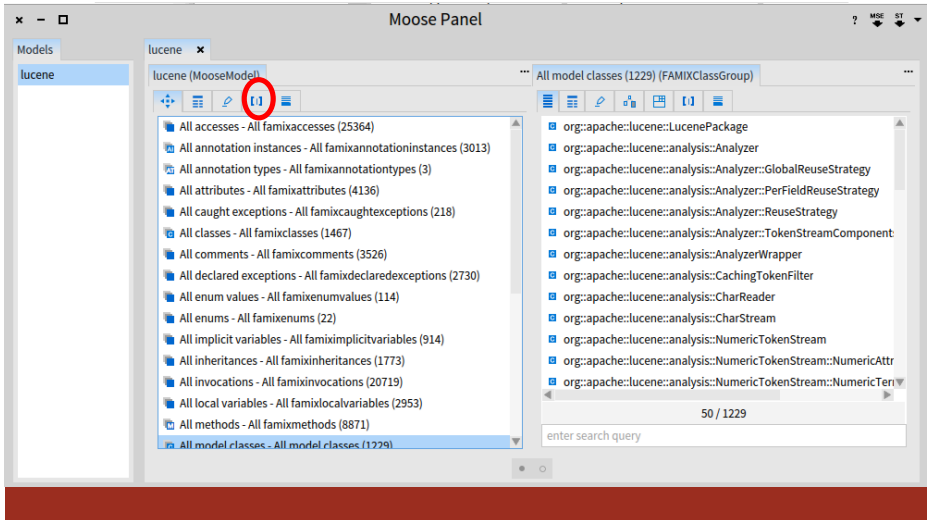
```
initialAnswer := fullName isEmptyOrNil

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

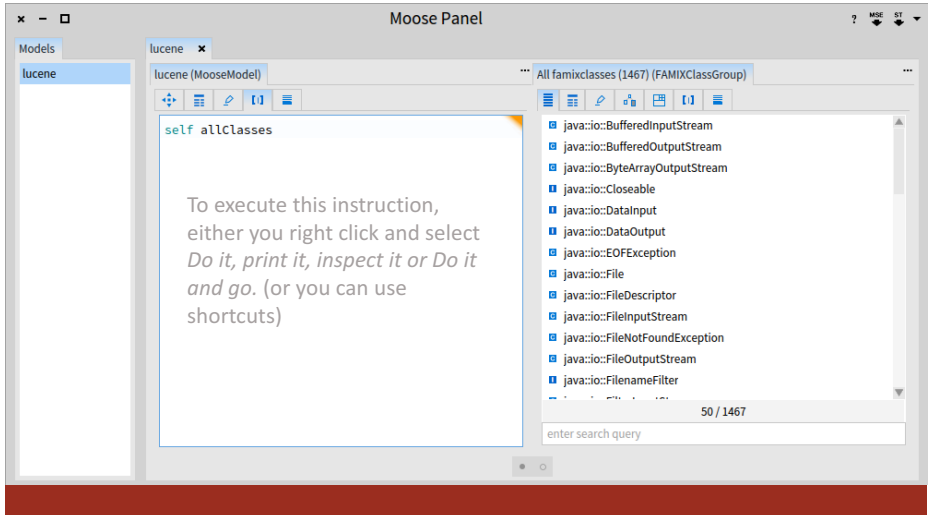
Getting a bit further

PERSONALIZED INFORMATION

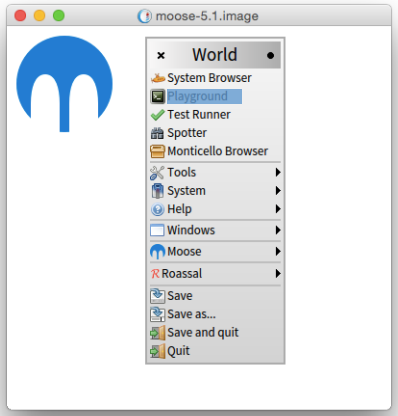
How many classes do we have in the model?



How many classes do we have in the model?



How many classes do we have in the model?



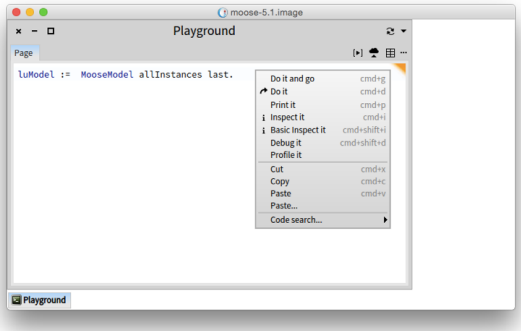
You can also use the playground

Click on Moose desktop and select Playground

How many classes do we have in the model?

```
luModel :=  MooseModel allInstances last.
```

To execute this instruction, either you right click and select *Do it, print it, inspect it or Do it and go.* (or you can use shortcuts)



How many classes do we have in the model?

```
luModel :=  MooseModel allInstances last.
luClasses :=  luModel allClasses.
luClasses size
```

How many classes do we have in the model?

```
luModel :=  MooseModel allInstances last.
luClasses :=  luModel allClasses.
luClasses size 1467
```

How many classes do we have in the model?

- How many classes are defined in lucene? (the others are stub)

How many classes do we have in the model?

- How many classes are defined in lucene? (the others are stub)

```
luClasses :=  luModel allModelClasses.
luClasses size
```

How many classes do we have in the model?

- How many classes are defined in lucene? (the others are stub)

```
luClasses :=  luModel allModelClasses.
luClasses size 1229
```

How many classes are « god » classes?

- (contain more than 50 methods)

```
luClasses :=  luModel allModelClasses.
(luClasses select: [ :each | each
numberOfMethods > 50 ]) size
```

How many classes are « god » classes?

- (contain more than 50 methods)

```
luClasses :=  luModel allModelClasses.
(luClasses select: [ :each | each
numberOfMethods > 50 ]) size 6
```

How many classes are « god » classes?

- (contain more than 500 lines of code)

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

How many classes are « god » classes?

- (contain more than 500 lines of code)

```
(luClasses select: [ :each | each
numberOfLinesOfCode > 500 ]) size 19
```

How many packages?

```
luModel allNamespaces
```

How many packages?

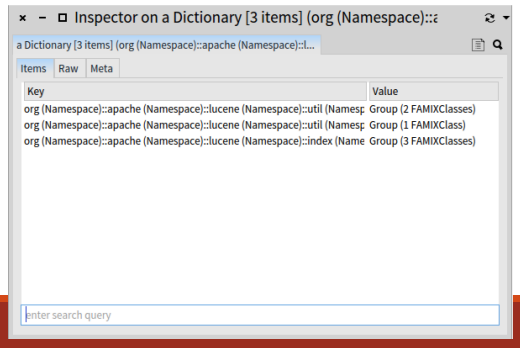
```
luModel allNamespaces
```

All famixnamespaces (47 FAMIXNamespaces)

Where are the largest classes located?

```
luClasses :=  luModel allModelClasses.

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



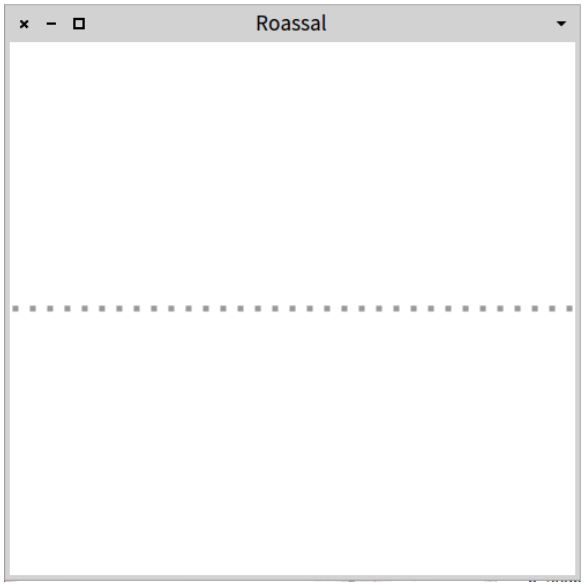
What are the classes with an annotation?

```
luModel allModelTypes select:
[ :t | t annotationInstances notEmpty ]
```

Visualization

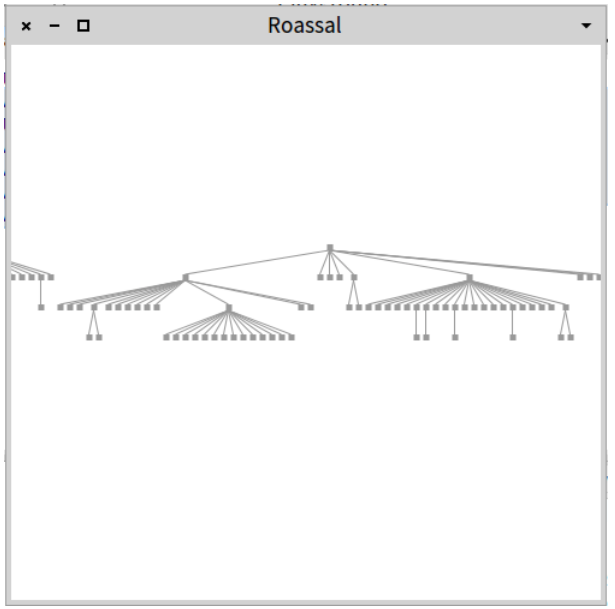
Classes

```
view := RTMondrian new.
luClasses := luModel allModelClasses.
view nodes: luClasses.
view open
```



Classes

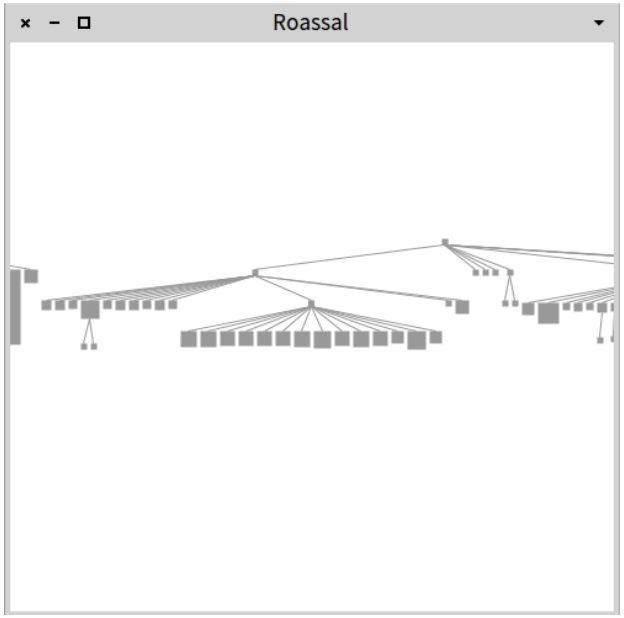
```
luModel :=  MooseModel allInstances last.
view := RTMondrian new.
luClasses := luModel allModelClasses.
view nodes: luClasses.
view edgesFrom: #superclass.
view treeLayout.
view open.
```



Classes

- But how are such classes?
  - Number of methods?

```
view shape rectangle size:
    [ :cls | cls numberOfMethods ].
view nodes: luClasses.
view edgesFrom: #superclass.
```



Classes

- Classes overriding a lot

```
model allModelTypes select: [ :t |
t numberOfMethodsOverriden >
  (t numberOfMethodsInherited / 3) ]
```

methods of all super classes recursively, that are not private

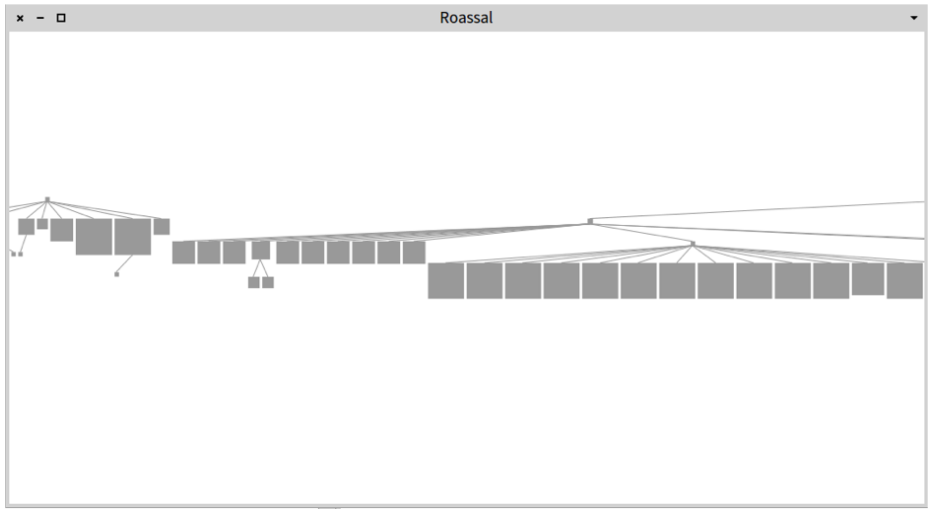
methods of the type that are also inherited

Classes

```
view := RTMondrian new.
luClasses := luModel allModelClasses.
view shape rectangle size: [ :cls |
      | inh |
      inh := 1 max: cls numberOfMethodsInherited.
      cls numberOfMethodsOverriden / inh * 100 ].
view nodes: luClasses.
view edgesFrom: #superclass.
view treeLayout.
view open·
```

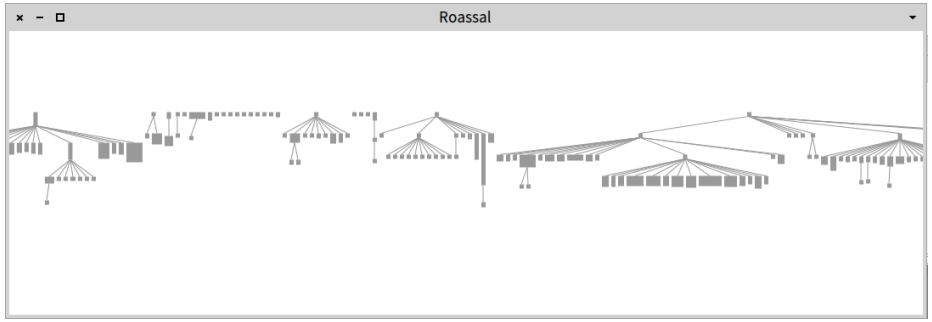
To avoid dividing by 0

Percentage of overriding methods



Classes

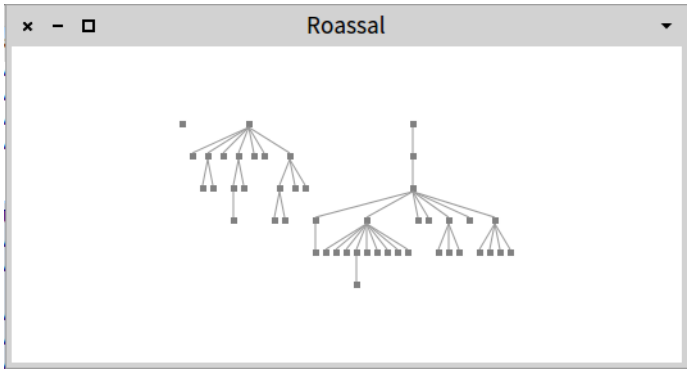
```
view := RTMondrian new.
luClasses := luModel allModelClasses.
view shape rectangle
        width: #numberOfAttributes ;
        height: #numberOfMethods.
view nodes: luClasses.
view edgesFrom: #superclass.
view treeLayout.
view open·
```



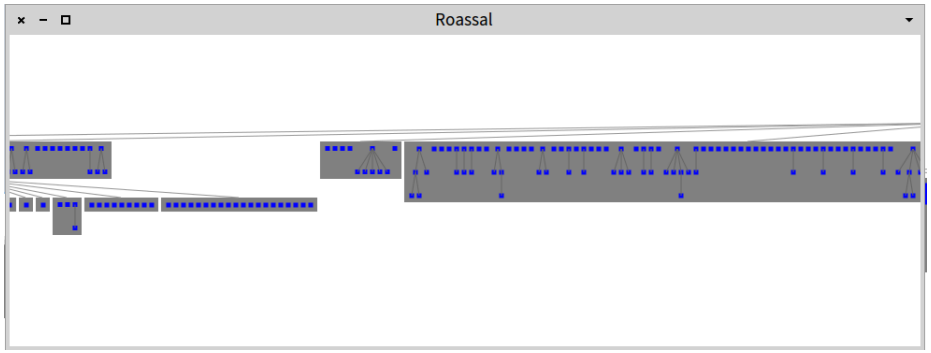
Namespace

- Where are the abstract roots or hierarchy / packages?

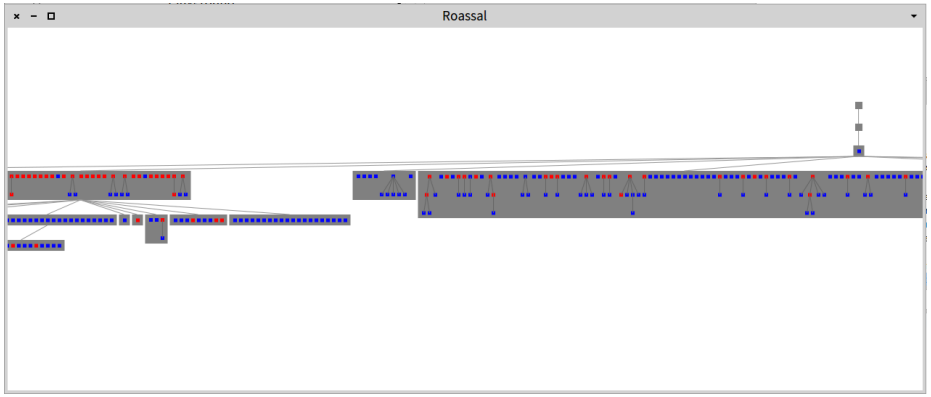
```
luModel :=  MooseModel allInstances last.
view := RTMondrian new.
view shape rectangle
     fillColor: Color gray.
view nodes: luModel allNamespaces.
view edgesFrom: #belongsTo.
view treeLayout.
view open.
```



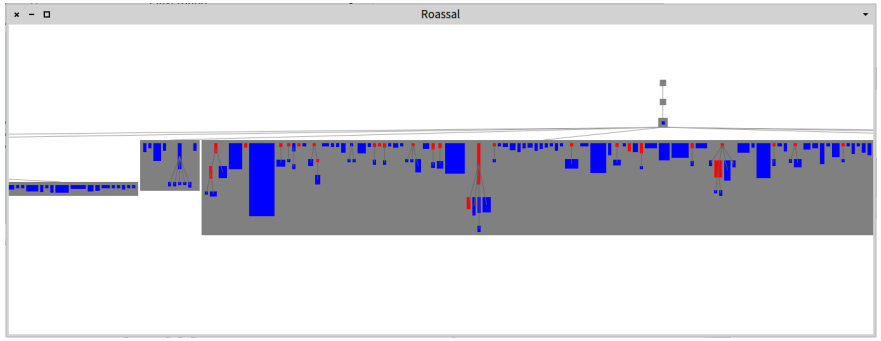

```
view := RTMondrian new.
view shape rectangle fillColor: Color white.
view nodes: luModel allNamespaces
   forEach: [:aNamespace |
      view shape rectangle
         color: Color blue.
      view nodes: aNamespace classes.
      view edgesFrom: #superclass.
      view treeLayout ].
view edgesFrom: #belongsTo. "[:each | each belongsTo]"
view treeLayout.
view open
```



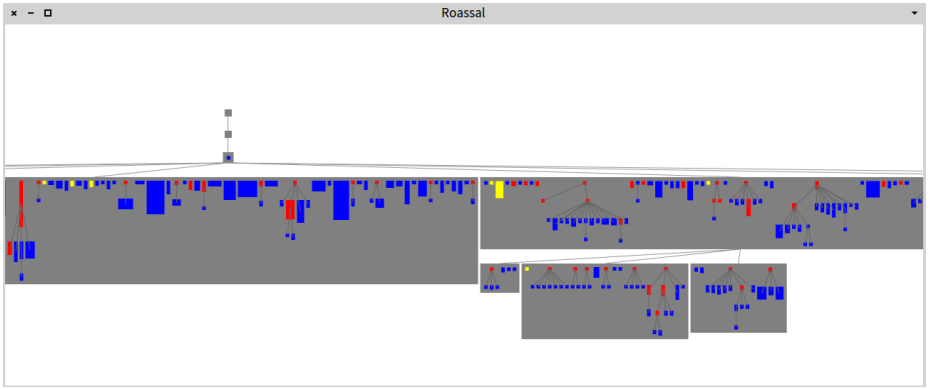

```
view := RTMondrian new.
view shape rectangle fillColor: Color white.
view nodes: luModel allNamespaces
   forEach: [:aNamespace |
      view shape rectangle
         color: [:cl | cl isAbstract
            ifTrue: [Color red]
            ifFalse: [Color blue] ].
      view nodes: aNamespace classes.
      view edgesFrom: #superclass.
      view treeLayout ].
view edgesFrom: #belongsTo. "[:each | each belongsTo]"
view treeLayout.
view open
```




```
view := RTMondrian new.
view shape rectangle fillColor: Color white.
view nodes: luModel allNamespaces
   forEach: [:aNamespace |
      view shape rectangle
         color: [:cl | cl isAbstract
            ifTrue: [Color red]
            ifFalse: [Color blue] ] ;
         width: [ :cl | cl numberOfAttributes ] ;
         height: #numberOfMethods.
      view nodes: aNamespace classes.
      view edgesFrom: #superclass.
      view treeLayout ].
view edgesFrom: #belongsTo. "[:each | each belongsTo]"
view treeLayout.
view open
```




```
view := RTMondrian new.
view shape rectangle fillColor: Color white.
view nodes: luModel allNamespaces
   forEach: [:aNamespace |
      view shape rectangle
         color: [:cl | cl isInterface
            ifTrue: [Color yellow]
            ifFalse: [ cl isAbstract
                  ifTrue: [Color red]
                  ifFalse: [Color blue] ] ] ;
         width: [ :cl | cl numberOfAttributes ] ;
         height: #numberOfMethods.
      view nodes: aNamespace classes.
      view edgesFrom: #superclass.
      view treeLayout ].
view edgesFrom: #belongsTo. "[:each | each belongsTo]"
view treeLayout.
view open
```



An alternative visualization

```
view := RTMondrian new.
luClasses := luModel allModelClasses.
allClasses := luClasses select: [:c | c numberOfLinesOfCode >
150].
view shape rectangle
       width: #numberOfAttributes ;
       height: [:c | c numberOfMethods * 0.5 ] ;
       fillColor: [:c ||loc|
              loc := c numberOfLinesOfCode.
              loc < 500
                     ifTrue: [ Color green ]
                     ifFalse: [loc < 1000
                            ifTrue: [ Color yellow ]
                            ifFalse: [ Color red ]]].
view nodes: allClasses.
view layout grid.
view open
```

An alternative visualization