Moose meta-modeling infrastructure

Pavel Krivanek, Inria Nord Europe, RMoD

Moose

- platform for software and data analysis
- language independent meta-modeling toolkit

A platform for the development of
exploratory environments
for any language

FAMIX

- the basic meta-model for Moose
- language independent
- needs to cover several programming paradigms
- uniforms representation of procedural and object-oriented languages

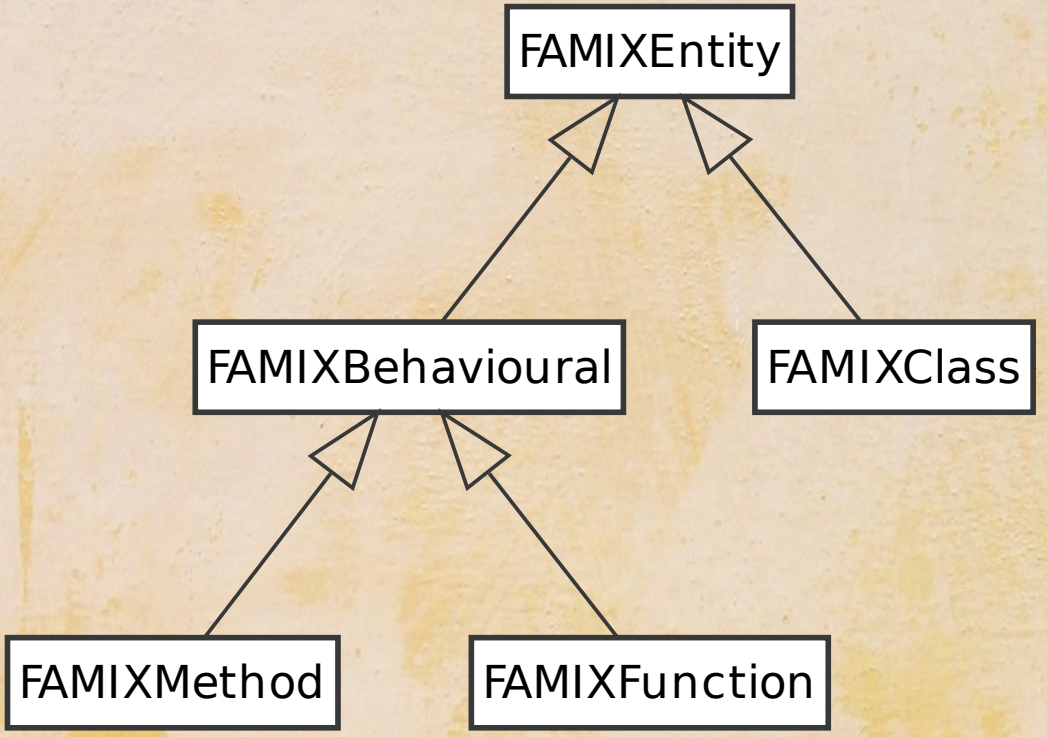
FAMIX issues

- impossible to uniform all languages
- one language implementation can have different meta-models
- SQL? DSLs? Structured data?
- AST-level meta-models?
- need of custom extensions
- multiple inheritance

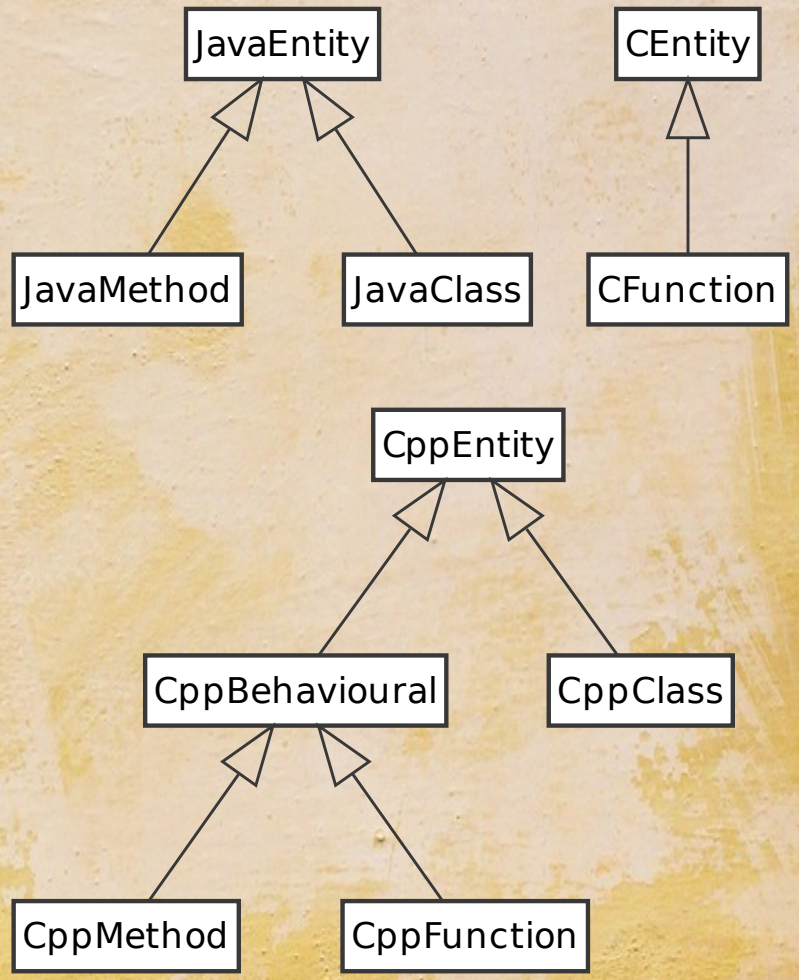
new Famix

Standalone meta-models

Old Famix:



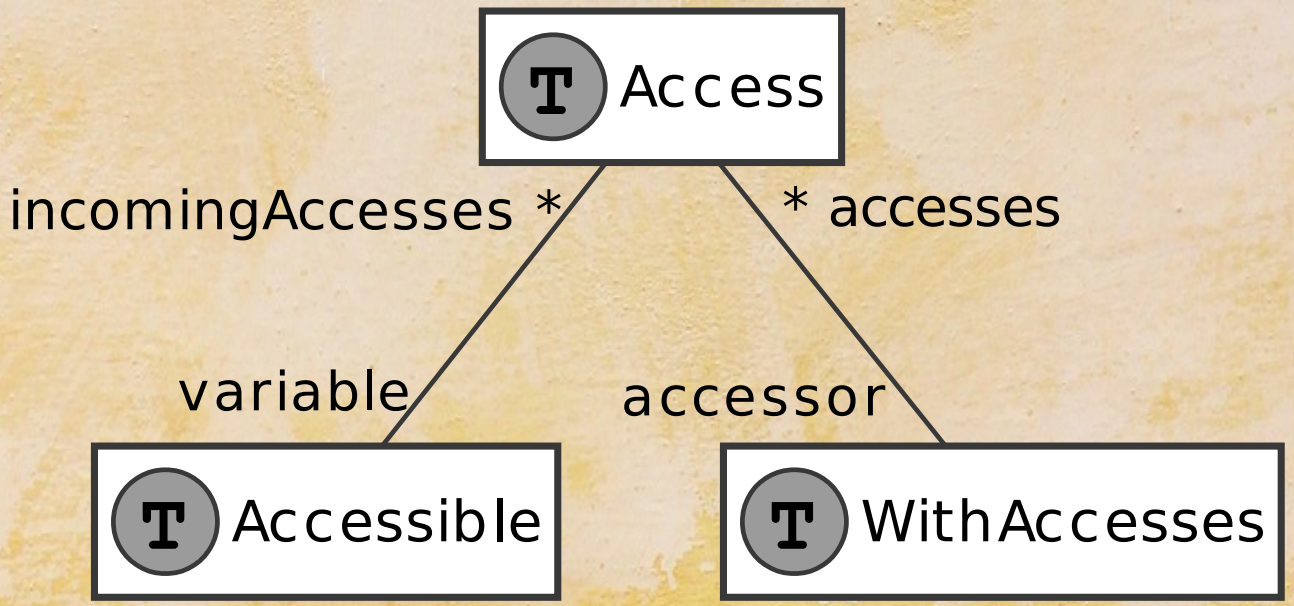
New Famix:



new Famix

Meta-model composition

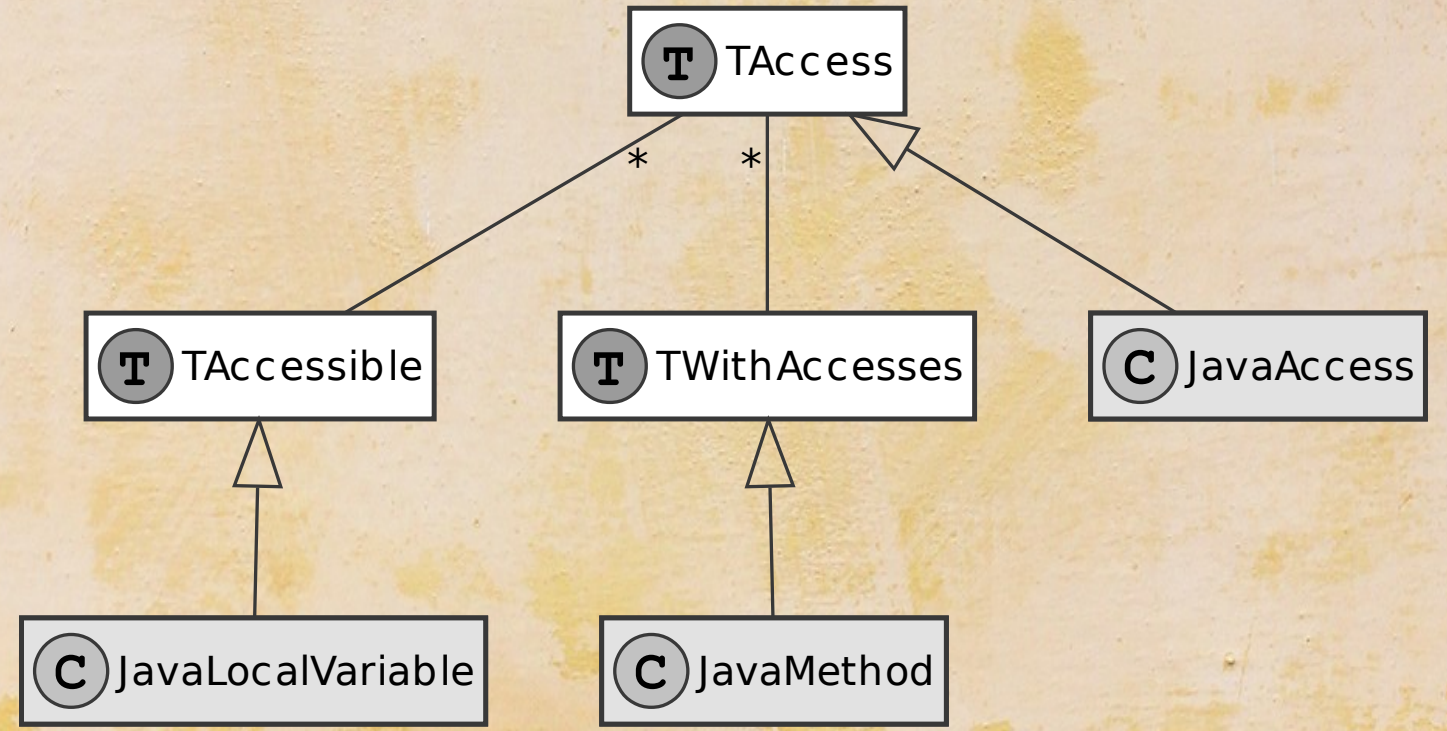
- Famix = collection of trait groups
- relations
- useful messages, metrics, tools support...



new Famix

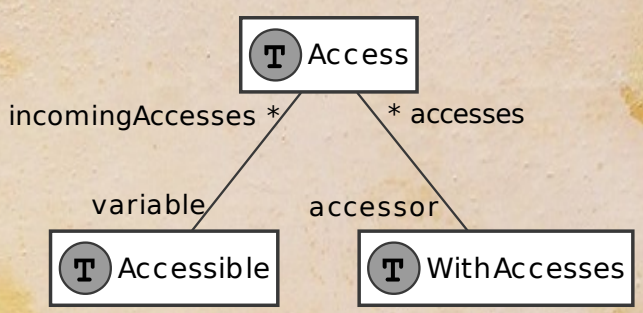
Meta-model composition

- meta-model entities use common traits



new Famix
Relations

- realized using slots
- automatic relation management




```
Trait named: #TAccess
        slots: {
            #accessor => FMOne type: #TWithAccesses opposite: #accesses.
            #variable => FMOne type: #TAccessible opposite: #incomingAccesses }

Trait named: #TAccessible
        slots: {
            #incomingAccesses => FMMany type: #TAccess opposite: #variable }

Trait named: #TWithAccesses
        slots: {
            #accesses => FMMany type: #TAccess opposite: #accessor }
```

new Famix

Stateful traits

- New feature of Pharo 7
- by Pablo Tesone
- separate ESUG 2018 talk – do not miss it!
- traits can provide instance variables
- image-side enhancement, no extra VM support
- loadable library (kernel without traits support)
- realized using custom Metaclasses

new Famix

How to create meta-models?

By hand?

FAMIXSourcedEntity subclass: #FAMIXNamedEntity
uses: FamixTInvocationsReceiver + FamixTNamed + FamixTPackageable + FamixTPossibleStub + FamixTWithAnnotationInstances + FamixTWithModifiers + TDependencyQueries + TEntityMetaLevelDependency...

- complex class definitions
- need to write some manual methods
  - annotations for FAME
  - accessors
  - additional containment definitions for Moose-Query
  - testing methods...

new Famix

Meta-model builder

- Smalltalk-based DSL for meta-models description

```
behaviouralEntity := builder newClassNamed: #BehaviouralEntity.
function := builder newClassNamed: #Function.

behaviouralEntity <|-- function.
function --|> #TFunction

(namedEntity property: #name type: #String)
    comment: 'This is a comment'.
```

new Famix

Meta-model builder - relations

```
class -* method.

((tAccess property: #accessor)
    comment: 'Behavioural entity making the access to the variable';
    source)
        *-
((tWithAccesses property: #accesses)
    comment: 'Accesses to variables made by this behaviour').
```

- new Famix meta-model is described using DSL

new Famix

Meta-model generators

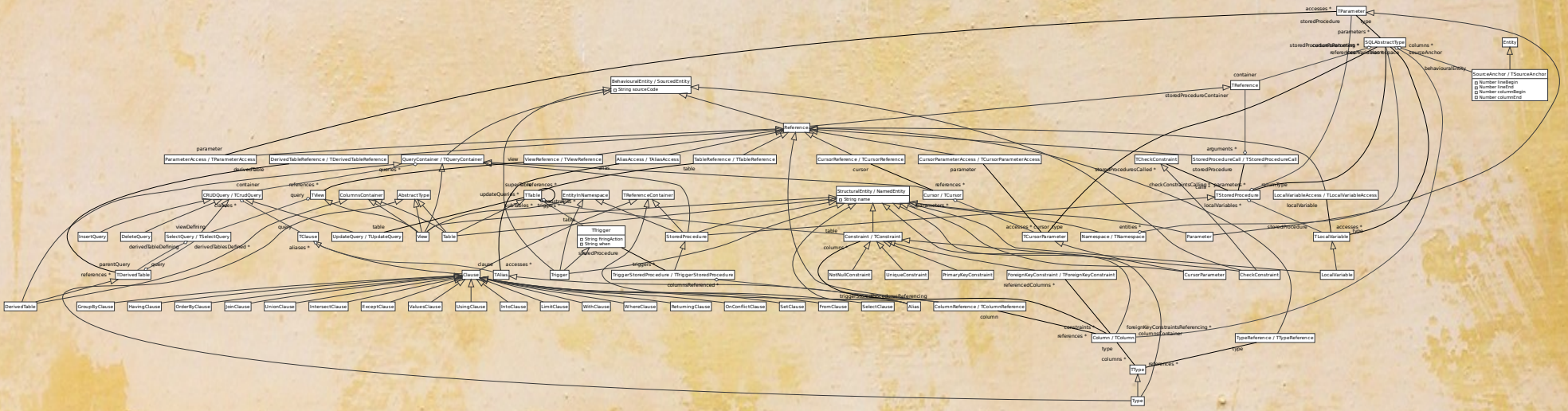
- Most meta-models share the same basic infrastructure
  - entity, namedEntity, sourceLanguage, association, sourced entity, comment...
- own meta-model generator as subclass of a predefined one
- FamixBasicInfrastructureGenerator
- FamixFileBasedLanguageGenerator
  - + file, fileAnchor, folder...

new Famix

PL/pgSQL meta-model

- Julien Delplanque
- multiple inheritances

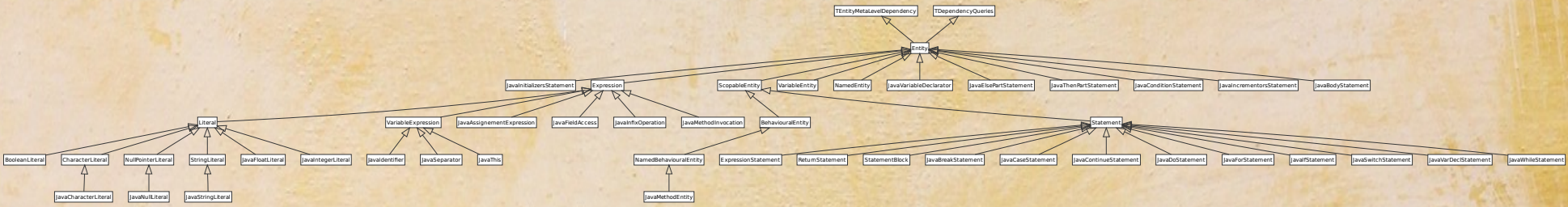
  e.g. Table: ColumnsContainer, AbstractType, StructuralEntity



new Famix

FAST

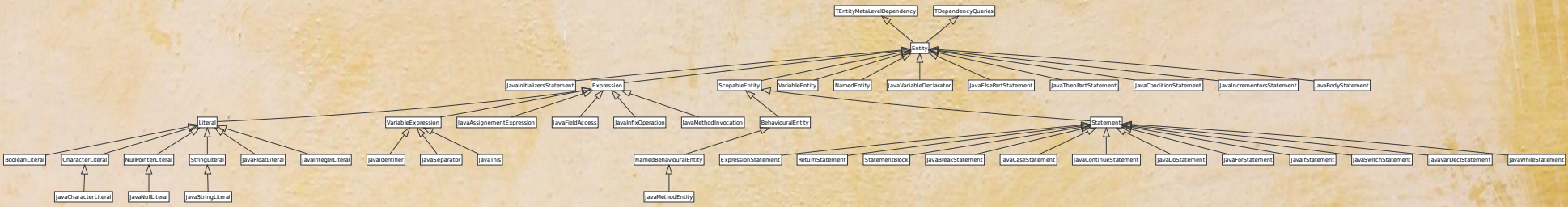
- Benoît Verhaeghe
- AST-level metamodel
- Java



new Famix

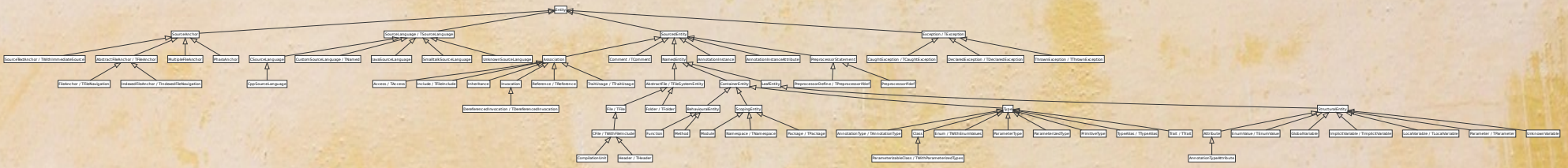
FAST

- Benoît Verhaeghe
- AST-level metamodel
- Java



new Famix

Old FAMIX compatibility metamodel



new Famix

DEMO

new Famix

Generator problem

When you generate code, how to enable manual code modifications or extensions and still be able to modify definition of the generator and support re-generation of it?

new Famix

Ring 2

- Generators to not generate code directly but create a Ring model
- this model is then applied on the image with respect to its current content
- BONUS:
  - fast and easy-to-write tests
  - browse the model with Calypso

new Famix

Generated methods

tAccess withTesting.

- automatic testing methods (e.g. #isAccess) in the root entities of the meta-model
- annotations...
- containment navigation utility methods

  belongsTo / belongsTo:

  - primary container
  - used by (Moose-Query)

new Famix

Importing contex

- cherry picking on models during import
- dependencies aware
- automatic

```
builder generateImportingContext: true.

user: context importMethod
importer: context shouldImportMethod

customization: #requires:
```

new Famix
Extensions

Problem: We want provide some utility methods or metrics that depend the meta-model structure

example:

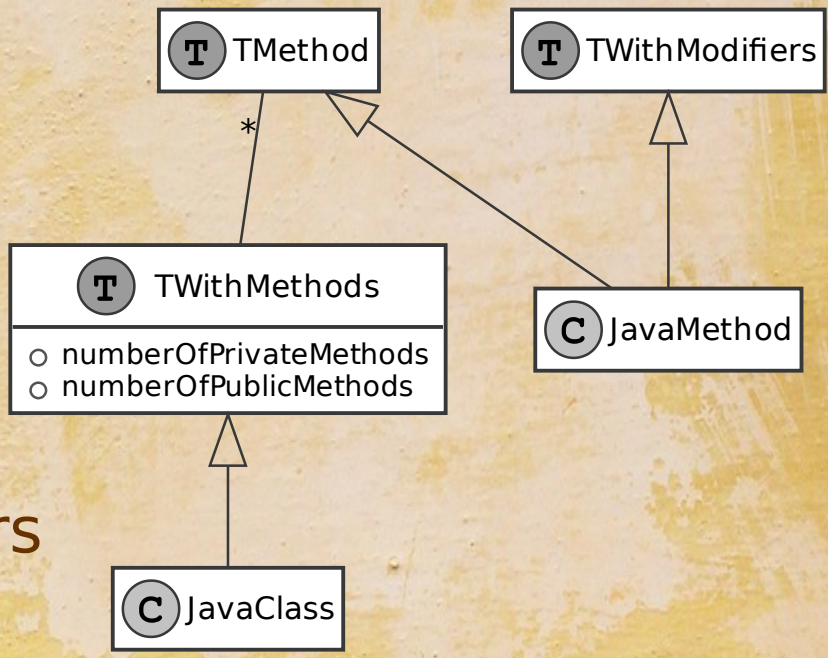
numberOfPrivateMethods

- on entity that owns methods

  TWithMethods

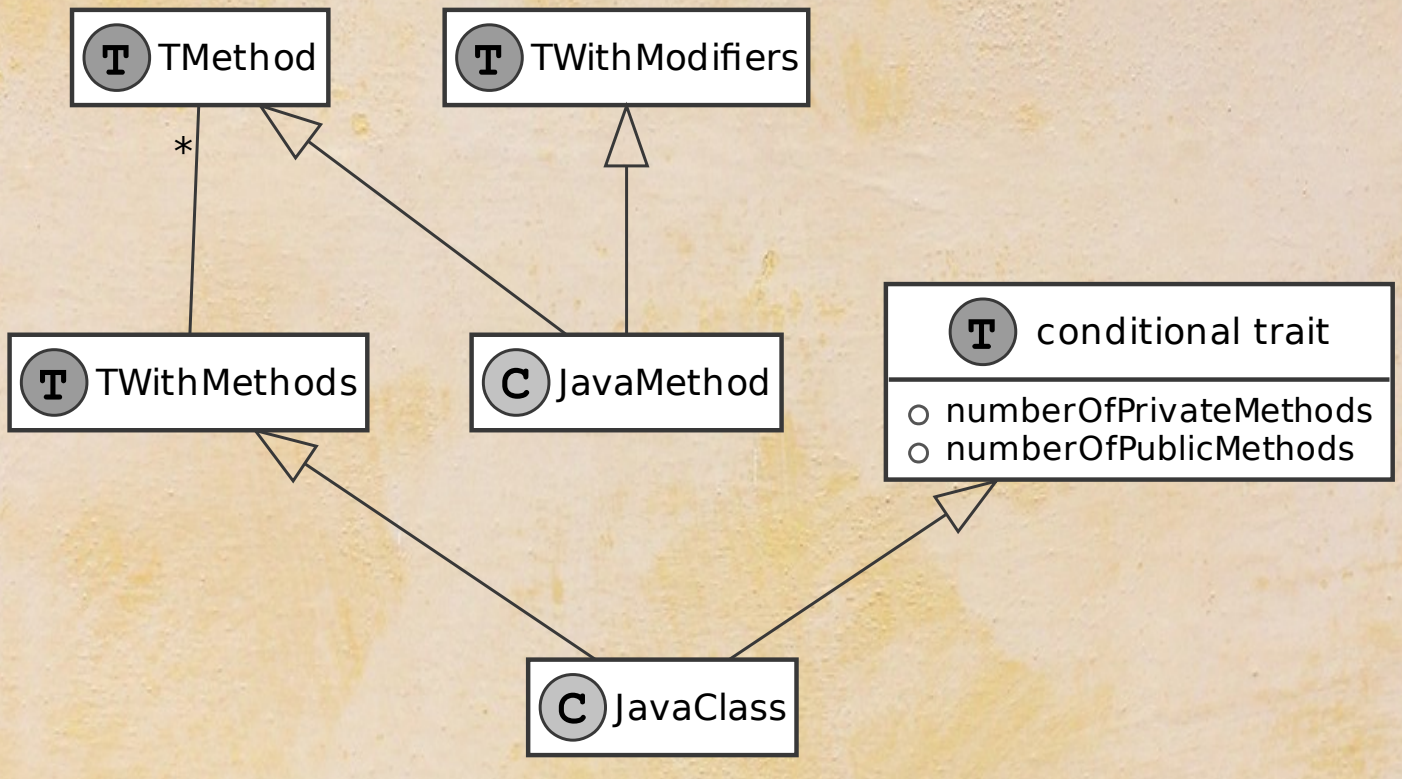
- methods must support modifiers

  TWithModifiers



new Famix

Conditional traits



- traits that are automatically used by entities that pass a condition

new Famix

Conditional traits

- **Advantage:** we can provide clean meta-models with rich functionality and tools support without making meta-model generation more complex for the user
- **Disadvantage:** more traits in the class definition, a lot of small traits

What if we want to know number of private methods for a package?

- more complex conditions (roots, common superclasses)
- Moose-Query

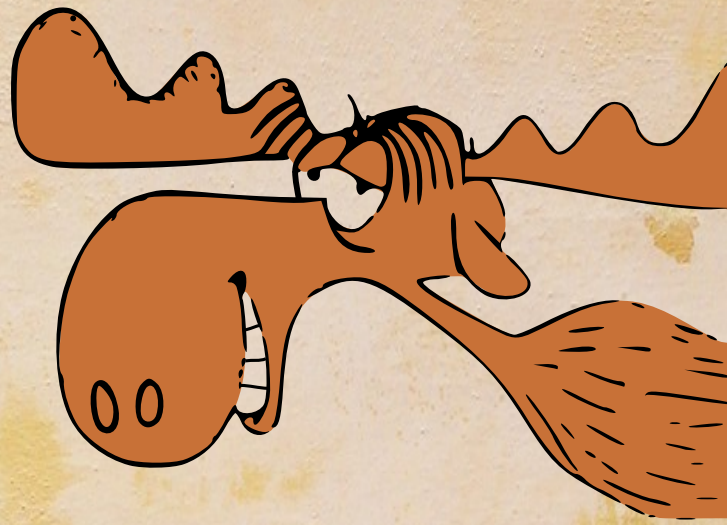
Git

- All Moose platform packages managed by Git
- Iceberg, Tonel
- Reproducible builds, CI

https://github.com/moosetechnology/Moose

https://github.com/pavel-krivanek/Moose

Summary



- Custom meta-models composition
- Groups of stateful traits
- Relations using slots
- DSL for meta-model description
- Basic language infrastructure generators
- Ring meta-model for code generation
- old FAMIX compatibility
- Importing contexts
- Conditional traits
- Git migration