Famix

- MétaModel composable pour la modélisation logicielle (ou autre)
- Modéliser n'importe quel langage de programmation (Java, Pharo, Fortran, Ada, C/C++, TypeScript, Cobol, 4D)
- Tous les outils Moose doivent fonctionner sur tous les modèles de tous les languages
- Basé sur les traits (méta-modèle **composable**)

Famix

- Essentiellement un modèle de dépendances
  - Note : aussi quelques métriques
- Entités du programme : Package, Classe, Méthodes, Variables, ...
- Dépendances entre elles :
  - Héritage `FamixJavaInheritance`
  - Accès (à une variable) `FamixJavaAccess`
  - Invocations (de méthodes) `FamixJavaInvocation`
  - Références (à un type) `FamixJavaReference`

Exercice Famix

- Navigation dans un modèle
- Trouver les sous-classes de `BLLazyServerPojo`
  - En Pharo (méthodes sur les collections, accesseurs)
  - Dans l'*Inspecteur* (inspectez le modèle)
  - Avec l'API *Moose Query* (v. ci-dessous)

MooseQuery

- API de requêtage uniforme sur les entités Famix
- Basé sur la description du méta-modèle (le méta-méta-modèle : *Fame*)
- Indépendant du méta-modèle (donc du langage modélisé)
- https://moosequery.ferlicot.fr/

Cheat sheet -- MooseQuery, parents/enfants

- `#children` (récursif `#allChildren`), ex: Package -> Package -> Class
- `#parents` (récursif `#allParents`), ex: Method -> Class -> Package
- Scopes : Cherche les ascendants ou descendants ayant un type donné
  - peut "sauter" des niveaux: `methodeA atScope: FamixJavaPackage`
  - `#atScope: <Type>`, recherche ascendante (récursif `#allAtScope:`)
  - `#toScope: <Type>`, recherche descendante (récursif `#allToScope:`)

Cheat sheet -- MooseQuery, "voisins"

- `#queryAllIncoming` / `#queryAllOutgoing` retournent toutes les *associations* (FamixJavaInheritance, FamixJavaInvocation, ...)
  - rajouter `#opposites` pour avoir les entités au bout des associations
  - ex: `packageX queryAllIncoming opposites`
- `#query: <#in/#out> with: <association>`
  - ex: `methodA query: #in with: FamixJavaInvocation`
- composition de requêtes (tous les packages dépendant de *packageX*):
  `packageX queryAllIncoming opposites atScope: FamixJavaPackage`

Famix Java -- Exercice

- Rechercher les attributs des principales entités
- `FamixJavaPackage`
- `FamixJavaClass`, `FamixJavaInterface`, `FamixJavaEnum`, …
- `FamixJavaMethod`
- `FamixJavaAttribute`, `FamixJavaParameter`, `FamixJavaLocalVariable`, `FamixJavaEnumValue`, …
- `FamixJavaComment`, `FamixJavaAnnotationInstance`

Les traits Famix

- Les principales entités (package, classe, ...) sont nommées
  - `FamixTNamedEntity` (propriété `#name` de type ... ?)
- Elles sont localisées dans un fichier source
  - `FamixTSourceEntity` (propriété `sourceAnchor` de type ... ?)
  - `FamixJavaSourceAnchor` (note : en lien avec le `rootFolder` du modèle)

UML

- génération d'un script PlantUML pour repreśenter un méta-modèle Famix

```
FamixMMUMLDocumentor new
  beWithStub ;
  model: FamixJavaModel ;
  generatePlantUMLModel.
```

- importer le script dans plantuml.com
- pour avoir tous les traits Famix, faire la commande ci-dessus sur le méta-modèle `FamixModel`

UML

- pour avoir tous les traits Famix, générer le UML :

```
FamixMMUMLDocumentor new
  model: FamixModel ;
  generatePlantUMLModel.
```

- *Blog post* : https://modularmoose.org/2021/06/04/plantUML-for-metamodel.html
- *Blog post* : https://modularmoose.org/2021/07/19/automatic-metamodel-documentation-generation.html

Création de méta-modèle

- *Blog post* https://modularmoose.org/2021/02/15/Coasters.html
- Conseils
  - Inspirer vous d'un méta-modèle existant (ex: FamixJava)
  - Utiliser au maximum les Traits Famix existants
  - Ne commencer que quand le méta-modèle est clair et complet
- créer un méta-modèle est une *longue* tâche, après 20 ans, FamixJava n'est toujours pas complètement fini

VerveineJ

- Création d'un modèle FamixJava

  ```
  docker run -v "/local/source/dir":/src -v "/local/lib/dir":/dependency
  ghcr.io/evref-bl/verveinej:v3.0.7 -format json -o projet.json .
  ```

  - Produit un fichier `projet.json` contenant le modèle Famix du projet
- Charger le projet dans Moose (outil *ModelBrowser*)
  - Attention au `rootFolder`

FAST

- Famix est essentiellement un modèle de dépendances
- Pour des analyses plus poussées (ou migration), il faut un AST complet
- FASTJava (Famix-AST)
  - *Carrefour* permet la création, au vol, de l'AST d'une `FamixJavaMethod`
  - À condition d'avoir accès au code source de la méthode
  - https://github.com/moosetechnology/Carrefour

Carrefour

- Charger *Carrefour* dans l'image (/!\ )
- Choisir une `FamixJavaMethod` (non stub)
- Vérifier qu'on a accès à son code source (*Inspector*, onglet "SourceText")
- Appeler `generateFastAndBind` sur cette méthode
- Inspecter le contenu de `fast` sur la méthode (onglets : "Tree" et "SourceText")
- *Blog post* https://modularmoose.org/2022/06/30/carrefour.html