Visualisation de RCA - Benoît Verhaeghe

L'objectif de ce TP est d'analyser l'application RCA en utilisant **Moose** et **Mondrian** comme vu en cours. Pour cela, vous avez à votre disposition le code source de l'application RCA, un fichier *.mse* généré à partir de l'application, accès à un parseur d'Interface Graphique Swing pour Moose, un méta-modèle d'UI et Java pour Moose.

Ressources

RCA Image + mse + Source code

Pour commencer

1. Télécharger le Pharo Launcher
2. Désarchiver le zip
3. Lancer localement l'image téléchargé
4. Charger le *.mse* dans Moose (si vous avez pris l'image depuis les Ressources, il est possible que cela soit déjà fait).
5. Créer un nouveau paquetage nommé "RCA-Analyse" et une classe à l'intérieur dans laquelle vous pourrez enregistrer votre code.

Pour charger le *.mse* dans Moose exécuter le code suivant dans un playground :

```
mooseModel := FAMIXModel importFromMSEStream: './verveinej/rca.mse' asFileReference readStream.
mooseModel name: 'rca'.
mooseModel rootFolder: './rcaexplore'.
mooseModel install.
```

Recherche possible

Notre objectif est de trouver des éléments que l'on pourrait améliorer dans l'application RCA. Pour cela, nous pouvons effectuer des requêtes sur le modèle pour essayer de trouver des problèmes dans le code. Voici une liste d'idée à explorer.

Pour s'échauffer

- Diagramme de classes
- God classes et lazy classes
- Code mort (toutes les méthodes qui ne sont pas invoquées ?)
- La complexité cyclomatique des méthodes (il y a déjà un outil qui fait le calcul dans Moose)
- Méthodes dépréciées

Faire des analyses plus avancées

- Hiérarchie de paquetage avec pour chaque paquetage
  - Nombre de classes en largeur
  - Nombre de méthodes en hauteur
- Hiérarchie de paquetage avec les classes à l'intérieur des paquetages dans la visualisation
  - La taille de chaque classe correspondra à son nombre de ligne de code
- Calcul de l'adhérence entre le projet RCA et les autres frameworks

Quelques questions pour cette dernière partie :

- Quelles sont les classes les plus importantes (en termes de ligne de code ? de complexité ?) ?
  - Est-ce que l'on si attend ?
- Que pensez-vous de l'adhérence entre *cern::colt* et RCA ?
- Si demain (imaginons) les développeurs décident d'abandonner colt, pouvez-vous leurs indiquer où le framework est utilisé, et la complexité de supprimer colt pour chacun des endroits détectés ?

Pour finir

- Ouvrez Iceberg dans pharo (Ctrl+O, Ctrl+I)
- Ajouter un repository (add)
- Dans "Clone From github.com" entrer comme information
  - Owner name : **badetitou**
  - Project name : **OOAnalysis**
- Valider, et charger le paquetage dans votre image Pharo.

Vous devriez maintenant avoir un paquetage *OOAnalysis* dans votre image avec au moins la classe *OOCriticsVisu*. Cette classe contient de nombreuses méthodes pour effectuer des requêtes sur le modèle.
