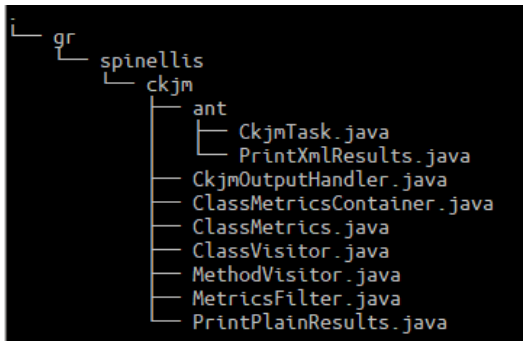
IFT3913 QUALITÉ DE LOGICIEL ET MÉTRIQUES – AUTOMNE 2022 – TRAVAIL PRATIQUE 1

Dans ce TP, vous allez chasser les *« classes divines »* («god classes»), c'est-à-dire des classes qui reconnaissent trop de choses ou font trop de choses. Les *classes divines* sont un exemple d'un anti-patron (Wikipédia). Dans ce qui suit, tous les programmes que je vous demande d'écrire sont supposés être exécutés à partir de la ligne de commande et sont supposés produire leur sortie sur la ligne de commande.

PARTIE 0 (15%)

Créez le programme **jls** qui prenne en entrée le chemin d'accès d'un dossier qui contient du code java potentiellement organisé en paquets (sous-dossiers, organisés selon les normes java) et produise en sortie en format CSV (« *comma separated values* », valeurs séparées par des virgules) les colonnes :

- `chemin du fichier`
- `nom du paquet`
- `nom de la classe`



Prenons comme exemple le dossier src du projet ckjm :

Votre **jls** doit produire 9 lignes sur la ligne de commande.

Pour le fichier `MethodVisitor.java` votre programme doit produire la ligne :

```
./gr/spinellis/ckjm/MethodVisitor.java, gr.spinellis.ckjm, MethodVisitor
```

PARTIE 1 (15%)

Créez le programme **nvloc** qui, étant donné un fichier source d'une classe java, calcule la métrique de taille NVLOC : nombre de lignes de code non-vides. Il doit juste sortir la valeur du NVLOC à la ligne de commandes.

PARTIE 2 (25%)

Créez le programme **lcsec** qui prend comme entrée le chemin d'un dossier, et un fichier CSV avec la sortie du **jls** pour le même dossier, et produise en sortie les mêmes données que dans le fichier, augmentés par la métrique de couplage CSEC («couplage simple entre classes», voir dessous) de chaque classe. Par exemple, si la classe MethodVisitor a un CSEC=3, la sortie du **lcsec** doit avoir, parmi autres, la ligne :

```
./gr/spinellis/ckjm/MethodVisitor.java, gr.spinellis.ckjm, MethodVisitor, 3
```

La métrique CSEC d'une classe c dans un ensemble de classes K est calculé comme suit :

$$CSEC(c) = \left| \{ d \in K-\{c\} \mid \textit{utilise(c,d)} \vee \textit{utilise(d,c)} \} \right|$$

Où le prédicat *utilise(c1,c2)* est vrai si le nom de c2 est *mentionné* dans le code de c1. Cela peut se produire pour diverses raisons : c1 déclare un objet du type c2, a une méthode avec un paramètre du type c2, fait appel à une méthode d'un objet du type c2, fait appel à une méthode statique de c2, accède à un attribut du c2, etc.

Pour **lcsec**, l'ensemble K est l'ensemble de classes dans son entrée. Ignorez toute autre classe.
NB : |K| pourrait être grand!

PARTIE 3 (20%)

Créez le programme **egon** (en l'honneur d'Egon Spengler) qui, étant donné un dossier avec du code java et un seuil, utilise **nvloc** et **lcsec** pour suggérer des classes qui sont suspectes d'être des *classes divines*.

Utilisez le seuil de l'entée pour l'heuristique suivante : une classe est suspecte si ses métriques NVLOC et CSEC sont tous les deux dans les *<seuil>*% supérieures de toutes les classes dans le dossier et sous-dossiers.

La suggestion à la sortie doit être composée par les lignes des classes suspectes sous format CSV, utilisant la même structure que **lcsec**, plus une colonne avec NVLOC. Par exemple, si MethodVisitor (NVLOC=109) et ClassVisitor (NVLOC=189) sont les seules suspectes pour ckjm, **egon** doit produire la sortie :

```
./gr/spinellis/ckjm/ClassVisitor.java, gr.spinellis.ckjm, ClassVisitor, 4, 189
./gr/spinellis/ckjm/MethodVisitor.java, gr.spinellis.ckjm, MethodVisitor, 3, 109
```

PARTIE 4 (25%)

Appliquez votre outil au code du https://github.com/jfree/jfreechart Téléchargez le code manuellement (votre outil n'a pas besoin de faire le téléchargement automatiquement). Vous devez soumettre des fichiers CSV avec la sortie d'**egon** pour le dossier https://github.com/jfree/jfreechart/tree/master/src/main/java pour des seuils de 1%, 5% et 10%.

À partir de vos résultats discutez brièvement l'utilité d'**egon** comme détecteur de problèmes de modularité : l'heuristique d'**egon,** semble-t-elle de vraiment trouver des classes divines? Comment pourrions-nous l'améliorer? Vous devez ajouter votre réponse dans un fichier de texte (voir précisions globales plus bas).

PRÉCISIONS GLOBALES

Travaillez en équipes de 2. Le TP est dû le **vendredi 30 septembre 23h59** via StudiUM. Aucun retard ne sera accepté. Vous pouvez utiliser **Python ou Java**.

Vous devez créer un **repositoire git** pour stocker votre code. Vous pouvez utiliser n'importe quel service gratuit comme Github, Bitbucket, et autres (quelques-uns vous permettent de créer des comptes académiques avec votre courriel @umontreal.ca). Utilisez le repositoire pour collaborer avec votre coéquipier. Nous allons examiner l'historique de votre repositoire pour nous assurer que tous les deux coéquipiers ont travaillé sur le TP et que votre code n'est pas plagié. Un historique de commit plausible devrait contenir de nombreux petits commit, chacun avec un message de commit approprié. **Faire juste quelques commit massives proche à la date limite pourrait entraîner une déduction considérable.**

Un membre de l'équipe doit soumettre un fichier ZIP contenant

a) un **fichier exécutable** de façon **autonome** (c.-à-d., incluant toutes les librairies que vous pourriez utiliser). N'ajoutez pas du code dans le zip! Votre repoisitoire doit être visible aux membres de l'équipe enseignante
b) pour la partie 4, votre ZIP doit contenir un sous-dossier appelé **PARTIE4**, contenant les CSV générés, ainsi qu'un fichier **reponse.txt** avec votre réponse.

c) un fichier **README.TXT** avec les noms des 2 membres de l'équipe en tête, le lien vers votre repositoire et avec la documentation de la façon de compiler, d'exécuter et d'utiliser votre code
d) tout autre information pertinente en format **TXT**.

L'autre membre doit soumettre juste le fichier README.TXT (les deux fichiers doivent être identiques).

Votre code doit être compilable et exécutable (même s'il peut être manque quelques fonctionnalités). **Code qui ne compile ou n'exécute pas sera accordé un 0**, donc assurez-vous d'empaqueter toutes les librairies nécessaires.

Manque de documentation en ce qui concerne la façon de compiler, d'exécuter **et d'utiliser** votre code, pourrait entraîner une déduction considérable.