

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

Dans ce TP vous allez mesurer la sous-documentation du code java. En autres termes, votre programme va trouver les cas où les développeurs ont créé du code compliqué sans un niveau de documentation adéquat.

PARTIE 0 (10%)

Écrivez du code lisible et documenté. Écrivez du javadoc pour toutes les classes et les méthodes publiques. Stockez les paramètres de configuration dans un fichier externe, évitez les valeurs magiques, respectez les règles de codage et de nommage.

PARTIE 1 (30%)

Écrivez un programme qui, étant donné fichier source d'une classe java ou un dossier contenant un paquet java, calcule les métriques :

- classe_LOC : nombre de lignes de code d'une classe
- paquet_LOC : nombre de lignes de code d'un paquet (*java package*) -- la somme des LOC de ses classes
- classe_CLOC : nombre de lignes de code d'une classe qui contiennent des commentaires
- paquet_CLOC : nombre de lignes de code d'un paquet qui contiennent des commentaires
- paquet_DC : densité de commentaires pour une classe : $\text{classe_DC} = \text{classe_CLOC} / \text{classe_LOC}$
- paquet_DC : densité de commentaires pour un paquet : $\text{paquet_DC} = \text{paquet_CLOC} / \text{paquet_LOC}$

Précisions :

- 0) Les précisions au-dessous concernent le parsing du code java. **Cependant le TP n'est pas un exercice sur le parsing du code mais un exercice de mesure de la complexité.** Donc ne vous perdez pas dans les détails de parsing du code java. Il est acceptable que votre programme traite le code java **presque correctement**, dans une marge d'erreur, avec du code potentiellement mal formé. Les métriques au-dessus mesurent la taille du code, ce qui est par définition une mesure imprécise de complexité.
- 1) Vous devez traiter tous les trois types de commentaires de Java (`//`, `/* */`, `/** */`). Vous devez aussi traiter correctement le cas des commentaires imbriqués. Vous pouvez faire l'hypothèse raisonnable que javadoc se trouve là où il est censé se trouver : juste avant la définition d'une classe, énumération, interface, méthode ou variable,
- 2) Les lignes de commentaires (incluant les javadoc) doivent être incluses dans le calcul des métriques LOC. Les lignes vides ne doivent pas être prises en compte
- 3) Une ligne qui contient à la fois du code et de commentaires (voir exemples au-dessous) compte à la fois pour LOC et CLOC. Exemples :
 - `i++; //increment`
 - `for (String element : theArray) /* guaranteed to contain 2 elements */ {`

PARTIE 2 (20%)

Faites que votre code 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 deux fichiers au format CSV (« *comma separated values* », valeurs séparées par des virgules).

- 1) Fichier classes.csv
 - La première ligne doit être : chemin, class, classe_LOC, classe_CLOC, classe_DC
 - Les lignes suivantes doivent afficher les données appropriées pour chaque classe dans n'importe quel paquet.
- 2) Fichier paquets.csv
 - La première ligne doit être : chemin, paquet, paquet_LOC, paquet_CLOC, paquet_DC
 - Les lignes suivantes doivent afficher les données appropriées pour chaque paquet.

Précisions :

- 0) Contrairement au précision 0 du partie 1, vous devez être minutieux quant au format de la sortie de votre programme. Les fichiers CSV doivent être traitables automatiquement par des autres outils (pensez de l'interopérabilité de votre outil).
- 1) Votre code doit être capable de traiter le dossier récursivement. Par exemple, si li dossier contient un projet java, il devrait produire des résultats pour tous ses paquets et les fichiers java indépendamment de l'endroit où ils se trouvent dans la hiérarchie des dossiers du projet. Traitez les interfaces, les énumérations et les classes abstraites comme des classes.

PARTIE 3 (30%)

Ajoutez de fonctionnalité afin que votre code calcule les métriques :

- WMC : « *Weighted Methods per Class* », pour chaque classe. C'est la somme pondérée des complexités cyclomatiques de McCabe de toutes les méthodes d'une classe. Si toutes les méthodes d'une classe sont de complexité 1, elle est égale au nombre de méthodes.
- WCP : « *Weighted Classes per Package* », pour chaque paquet. C'est la somme des WMC de toutes les classes d'un paquet et les WCP de ses sous-paquets.
- classe_BC : degré selon lequel une classe est bien commentée $\text{classe_BC} = \text{classe_DC} / \text{WMC}$
- paquet_BC : degré selon lequel un paquet est bien commentée $\text{paquet_BC} = \text{paquet_DC} / \text{WCP}$

Les résultats doivent être ajoutés aux fichiers CSV classes.csv et paquets.csv. Aux premières lignes de chaque fichier, utilisez les titres WMC, classe_BC et WCP, paquet_BC respectivement.

PARTIE 4 (10%)

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 les fichiers CSV générés par votre code. À partir de vos observations, identifiez les 3 classes et les 3 paquets les moins bien commentés et proposez des améliorations. Décrivez votre solution dans un fichier de texte (voir précisions globales plus bas).

BONUS 10%

Utilisez votre outil pour essayer d'améliorer le niveau de documentation dans un projet java open source sur Github (par exemple : <https://github.com/topics/java?o=desc&s=forks>). Pour qu'un bonus de 10% soit accordé, vous devez me montrer les résultats de l'application de votre outil au projet, le(s) *pull request(s)* soumis (que vous devez avoir soumis avant l'échéance du TP), et je dois être convaincu que cela a été fait de bonne foi et sérieusement.

Un bonus de chocolat vous sera accordé si un de vos *pull request* est accepté par les développeurs du projet. Pour que le bonus soit accordé, vous devez me convaincre qu'un de vos *pull request* a été réellement accepté par les développeurs du projet. Par exemple, ce n'est pas le cas que vous ayez accepté votre propre *pull request* ou que vous soyez ami avec le responsable du projet et que votre ami ait accepté le *pull request* dans le but du bonus du cours 😊. Je considérerai les *pull requests* qui sont acceptés en tout temps avant le 2022-04-15.

PRÉCISIONS GLOBALES

Travaillez en équipes de 2. Le TP est dû le **vendredi 11 février 23h59** via StudiUM. Aucun retard ne sera accepté. Vous pouvez utiliser n'importe quel langage **du JVM** (ex. Java, Scala, Jython, Kotlin, Groovy, ...).

Vous devez créer un **répositoire 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 répositoire pour collaborer avec votre coéquipier. Nous allons examiner l'historique de votre répositoire 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 JAR exécutable** de façon **autonome** (c.-à-d., incluant toutes les librairies que vous pourriez utiliser). N'ajoutez pas du code dans le zip! Votre répositoire 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 par votre programme, ainsi qu'un fichier **solution.txt** avec votre réponse.
- c) un fichier **README.TXT** avec les noms des 2 membres de l'équipe en tête, le lien vers votre répositoire et avec la documentation de la façon de compiler, d'exécuter et d'utiliser votre code
- d) toute 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 manqué 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.

Vous êtes encouragés d'utiliser des outils automatisés pour l'amélioration de votre code, comme SpotBugs (<https://spotbugs.github.io/>), checkstyle (<https://checkstyle.sourceforge.io/>), PMD (<https://pmd.github.io/>), JDeodorant (<https://github.com/tsantalis/JDeodorant>), lint4j (<http://www.jutils.com/>). L'utilisation de ces outils n'est pas une précondition pour le TP, c'est juste une suggestion.