TD : Compréhension Logiciel

L'objectif de cet exercice est de vous forcer à naviguer dans un code inconnu. Le but ultime sera de retrouver une partie de la spécification d'origine du programme.

Exercice 1

1. Récupérer le code source sur Moodle et importez le dans un nouveau projet sur Eclipse
2. Localisez le point d'entré du programme (la fonction *main*())
   2.1. Commentez sur la pertinence de l'emplacement du *main*()
   2.2. Selon vous, quel est l'intérêt des deux bloques *if* présent dans *main*()?
   2.3. Quel est le problème ?
3. Essayez d'exécuter le programme.
4. Écrivez un paragraphe (max 5) expliquant selon vous l'intention initiale des développeurs.
5. Proposez comment modifier le *main*() afin de résoudre ses problèmes.

Exercice 2

Dans cette exercice nous allons voir comment générer le diagramme UML d'un programme à partir de son code source.

1. Installation de ObjectAid
   1.1. Dans Eclipse, installez ObjectAid en utilisant *Install new software* et en utilisant ce lien http://www.objectaid.com/update/current. (Video tutoriel)
   1.2. Sélectionnez *ObjectAid Diagram Add-on* et poursuivez l'installation.
   1.3. Une fois terminé, selectionner le projet de l'exercice 1 dans Eclipse. $Cliquedroit > New > Other... > ObjectAidUMLDiagram > ObjectAidClassDiagram > Next$. Entrez le nom du projet et validez.
   1.4. Ouvez le fichier $< name > .ucls$ et glissez/déposez les sources de l'application dans la page. (voir tutoriel)
2. En étudiant le diagramme UML, observez l'architecture objet du programme et commentez là. Quels sont selon vous les points forts et les points faibles de cette architecture concernant la maintenabilité?

3. Donnez un exemple de design pattern (patron de conception) qui permettrait d'améliorer la maintenabilité du code source. Décrivez en quelques lignes comment l'implémenter dans ce programme.

4. Estimez, selon votre expérience, la difficulté de la tâche si vous deviez implémenter le design pattern. Vous devez pour cela considérer que vous êtes seul(e) à réaliser cette implémentation.

TP : Maintenabilité et Documentations

l'objectif du TP est de créer la documentation d'une application afin de facilité sa maintenance. Cette application est destinée à évaluer la maintenabilité du code source d'un logiciel. Elle prend en entré les paramètres de métadonnées tel que LoC, coupage inter classe, nombre de développeurs, les dépendances de bibliothèques, nombre de caractéristiques, le ratio bug/-fix etc. L'application calcule en sortie un score permettant aux développeurs de déterminer la maintenabilité du code.

Pour cela, vous aller d'abord commencer par imagier toutes les métriques possibles pouvant être utilisées pour évaluer de la maintenabilité (voir cours) (donnez en au moins 4). Décrivez aussi comment ses métriques sont acquise par l'application. Cette description doit être suffisamment précise pour faciliter la compréhension logiciel.

Proposez une architecture logicielle pour concevoir cette application. Attention toutefois car cette architecture doit être la plus maintenabilité possible. A vous donc de trouver les design pattern adaptés. Vous pouvez proposer cette application dans le paradigme que vous voulez (orienté objet, service, composant, etc). Vous justifierez votre choix de paradigme par rapport au critère de maintenabilité.

Le rendu du TP est un rapport PDF de 2 à 3 pages et doit inclure :

- une description générale de l'application
- un diagramme d'architecture décrivant une vue générale du programme.
- un diagramme UML des classes (simplifié).
- un design pattern (aux choix) et justifiez votre choix. Donnez également le diagramme UML du design pattern selon votre application.
- la liste des métriques et leurs emplacements dans l'architecture
- des indications sur les points d'évolution possible de l'application.

Bonus : Outils de maintenance intéressant

1. **SonarQube** Outils de gestion de la qualité du code.
   - https://fr.wikipedia.org/wiki/SonarQube
   - https://www.sonarsource.com/
   - https://linsolas.developpez.com/articles/java/qualite/sonar/?page=page$_1$
   - http://www.methodsandtools.com/tools/tools.php?sonar
   - https://www.baeldung.com/sonar-qube
   - https://www.sourceallies.com/2010/02/sonar-code-quality-analysis-tool/
2. **FindBugs** Son but est de trouver des bugs dans les programmes Java en identifiant des patterns reconnus comme étant des bugs.
   - http://findbugs.sourceforge.net/
   - https://fr.wikipedia.org/wiki/FindBugs
   - https://www.baeldung.com/intro-to-findbugs
   - https://www.commentcamarche.net/faq/19006-installation-et-utilisation-du-plugin-findbugs-d-elipse
   - https://github.com/findbugsproject/findbugs
3. **Soot** Soot est un framework pour la manipulation et l'optimization de Bytecode. Il s'utilise sur le langage intermédiaire de Java
   - http://sable.github.io/soot/
   - https://www.sable.mcgill.ca/soot/tutorial/pldi03/tutorial.pdf
   - https://en.wikipedia.org/wiki/Soot$_(software)$
   - https://github.com/Sable/soot/wiki/Tutorials
   - https://courses.cs.washington.edu/courses/cse501/01wi/project/sable-thesis.pdf
4. **Gumtree** Logiciel dit de *diff* qui met en avant les différences de codes entre deux versions d'un logiciel. Utilise l'AST pour être plus efficace.
   - https://github.com/GumTreeDiff/gumtree
   - https://www.labri.fr/perso/falleri/perso/tools/gumtree/
   - https://hal.archives-ouvertes.fr/hal-01054552/document
   - http://courses.cs.vt.edu/cs6704/spring17/slides$_by_students/CS6704_gumtree_Kijin_AN_Feb15.pdf$
   - https://www.openhub.net/p/gumtree

5. **MicroART** Il permet de représenter l'architecture global d'une architecture micro-service

   - https://github.com/microart/microART-Tool
   - https://fr.slideshare.net/paolodifrancesco/microart-a-software-architecture-recovery-tool-for-maintaining-microservicebased-systems
   - https://www.researchgate.net/publication/317927398_MicroART_A_Software_Architecture_Rec Based_Systems

6. **CodeCity** & **JSCity** Il permet de visualisation d'architecture logiciel. Représente l'architecture sous la forme d'une ville 3D.

   - https://wettel.github.io/codecity.html
   - https://wettel.github.io/download/Wettel08b-wasdett.pdf
   - https://www.researchgate.net/publication/221555855_CodeCity_3D_visualization_of$large-scale_software$/link/02bfe513998dce533f000000/download
   - https://marketplace.eclipse.org/content/codecity
   - https://fr.slideshare.net/esug/codecity-esug2008
   - https://github.com/ASERG-UFMG/JSCity/wiki/JSCITY
   - https://bjoernkw.com/2016/11/27/jscity-code-complexity-visualization-for-javascript-codebases/

7. **But4Reuse** Il permet de faire l'extraction d'une Ligne de Produit Logiciel.

   - https://but4reuse.github.io/
   - https://github.com/but4reuse/but4reuse/wiki
   - https://hal.sorbonne-universite.fr/hal-01531890/document

8. **SonarGraph** (**SonarJ**) Sonargraph-Explorer is a simple but powerful static analysis tool with a focus on metrics and dependency visualization Il s'agit d'un outil simple mais puissant d'analyse statique avec un attention particulière sur les métriques et la visualisation de dépendances

   - https://www.hello2morrow.com/products/sonargraph
   - https://www.hello2morrow.com/products/sonargraph/explorer

9. **KDiff3** équivalent Gumtree, mais intègre un *merger* de code.

   - http://kdiff3.sourceforge.net/
   - https://www.fosshub.com/KDiff3.html

10. **JArchitect** Permet d'extraire divers métrique sur du code Java
    - https://www.jarchitect.com/
    - https://en.wikipedia.org/wiki/JArchitect
11. **Checkstyle** Il s'agit d'un outil d'aide au développement de code écrit en Java, qui permet de vérifier si le code respecte les standards de style de programmation.
    - https://checkstyle.sourceforge.io/
    - https://github.com/checkstyle/checkstyle
    - https://www.baeldung.com/checkstyle-java