Laboratoire 2 - Assistants de Refactoring

LOG530 – Réingénierie du Logiciel

Date de remise : 7 février 2024 à 23 h 59

Table des matières

1. Objectifs

Le travail à réaliser dans ce laboratoire vise à explorer :

- Les différents outils disponibles pour l'assistance au refactoring. En utilisant des outils appropriés, nous pouvons planifier à l'avance les activités de refactoring qui pourraient améliorer notre système logiciel.
- Les mauvaises odeurs (ou code smells en anglais) qui représentent des symptômes de problèmes de conception ou d'implémentation. Ces mauvaises odeurs nous donnent des indices sur où et comment refactoriser. Lors de la planification des activités de refactoring, gardez à l'esprit le patron « Keep it Simple » (livre Object-Oriented Reengineering Patterns OORP, p.37), et « Most valuable first » (OORP, p. 29).
- Les opérations de refactoring à appliquer : consulter les catalogues de refactoring, se familiariser avec le refactoring via les IDEs et les outils avancés.

2. Matériel et outils à utiliser

Outils de base

- IntelliJ IDEA (vous pouvez utiliser Eclipse, à votre discrétion, mais cela peut nécessiter des adaptations pour le projet que nous utilisons pendant les sessions de laboratoire)
- Le projet JPacman.
- CodeScene : pas d'installation requise, mais il nécessite un compte GitHub. L'intégration de cet outil avec GitHub lui permet de visualiser vos répertoires de code. La partie dette technique (Technical Debt) affiche les cibles de refactoring. Les codes biomarqueurs (Code Biomarkers) montrent une analyse plus détaillée des odeurs (code smells), mais ce n'est disponible que pour les abonnements payants.
- SonarQube : un outil/plateforme d'inspection continue de la qualité du code.
- Compte GitHub

Lectures de référence recommandées

- Chapitre 2 « Setting Direction » du livre Object-Oriented Reengineering Patterns (disponible en PDF sous l'onglet Références dans Moodle) : page 29 (« Most valuable first »), et page 37 (« Keep it Simple »).

Outils auxiliaires

Les outils auxiliaires **ne sont pas nécessaires** pour la réalisation de laboratoire, mais ils peuvent être utiles pour obtenir des informations supplémentaires (ou des alternatives) sur un projet. Vous pouvez les utiliser à votre discrétion.

- RefactoringMiner : un API/outil qui détecte l'historique des opérations de refactoring appliquées dans un projet (Java).
- PMD : un outil d'analyse du code source. Il permet de détecter certaines odeurs de code et mauvaises pratiques.
- SpotBugs : un outil de détection de bogues dans Java. Certaines de ses catégories de détection peuvent être considérées comme des odeurs de code (au lieu de bogues réels). Par conséquent, cet outil est utile pour trouver les odeurs de code et les mauvaises pratiques d'implémentation qui peuvent nécessiter du refactoring. Il s'agit de la continuation de l'outil FindBugs qui devient obsolete depuis 2015.
- CheckStyle : un outil de contrôle de code qui permet de vérifier le style du code source écrit en langage Java.
- JDeodorant : un plugin Eclipse qui détecte certaines odeurs de code et recommande des operations de refactoring appropriées pour les corriger. Il supporte les odeurs de code suivant : Type Checks, Feature Envy, God Class, Long Parameter List. JDeodorant a récemment une version de plugin spécifique pour IntelliJ nommée IntelliJDeodorant.
- JMove : un autre plugin Eclipse, spécialisé pour le refactoring move method uniquement. Par conséquent, c'est un bon outil pour détecter et corriger l'odeur de code Feature Envy.

3. Installation et préparation

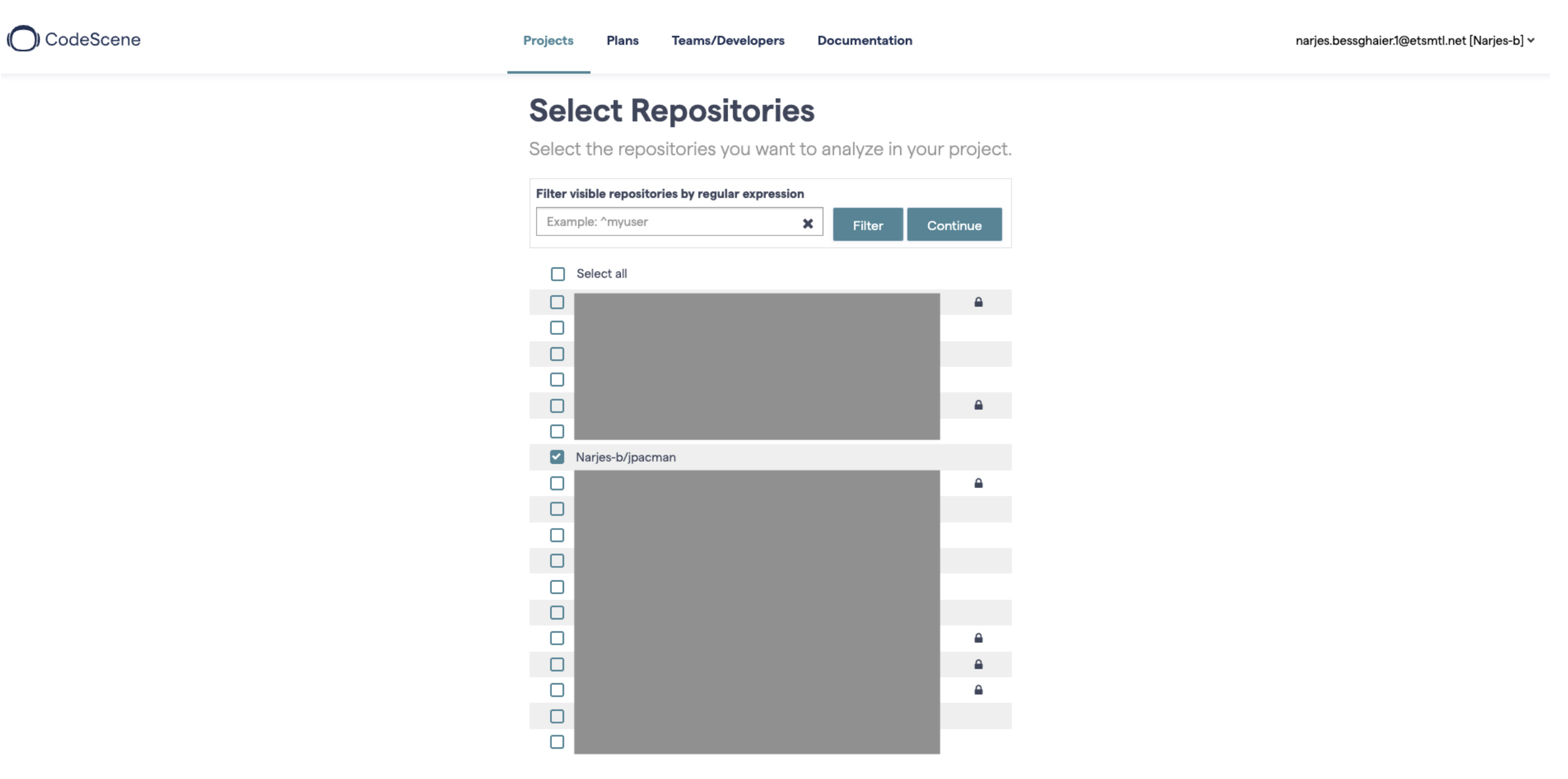
Pour faire ce laboratoire, préparer les outils de base indiqués dans la section 2 : Outils de base

CodeScene

Tout d'abord, assurez-vous de transférer (faire un "fork") du projet JPacman vers votre compte GitHub. Il est nécessaire de faire un fork du projet car la version gratuite de CodeScene voit que les projets de votre propre compte.

*. Soit vous essayer codescene dans le cloud:

Donc connectez vous a votre compte Github. Une fois connecté, vous pouvez choisir de créer un nouveau projet ou CodeScene affichera tous les projets GitHub de votre compte. Sélectionnez JPacman et passez à la partie suivante jusqu'à ce qu'elle se termine (cela peut prendre un certain temps, selon le projet).



*. Soit essayez codescene sur votre machine:

1- Si git est installé sur votre machine, passez a l'étape 3, sinon, installez git;

2- Ajoutez git au PATH (variable d'environement);

3- Lancez le fichier JAR du CodeScene à partir de votre ligne de commandes ou terminal, en utilisant la commende: `java -jar path/to/file.jar`. Si vous trouvez un probleme de permission, executez la commande `sudo chmod 755 path/to/file.jar` et ressayer de nouveau;

4- Dirigez votre navigateur vers l'addresse: `localhost:3003`;

5- Connectez vous en utlisant le user name et licence key deja reçu par e-mail de la part du CodeScene;

6- Clonez le projet JPacman;

7- Dans le menu `Projects`, choisissez `Analyse local Git repositories on your hard drive` et ajoutez le chemin local vers le git du project JPacman.

Ouvrez le project JPacman (`new --> project from existing sources --> jpacman --> Gradle --> create`) sur IntelliJ; lancer le `build.gradle`. JPacman utilise Gradle comme gestionnaire de build/dépendances. Assurez-vous de pouvoir faire le build et l'exécuter avant de modifier le code source.

SonarQube

Téléchargez SonarQube (si vous ne l'avez pas déjà fait) et décompressez-le sur votre ordinateur.

NB. On va utiliser la version de *sonarqube-9.8* qui est compatible ***uniquement*** avec Oracle JRE 11 ou 17 ou OpenJDK 11 (On recommende l'utilisation de JRE 17). 1- Assurez-vous d'ajouter le jdk comme une variable d'environement pour qui'il soit visible par le system.

Guides: JDK_mac_installation , JDK_windows_installation

Nous devrons utiliser la ligne de commande pour démarrer l'outil SonarQube. Ouvrez un terminal (invite de commande sous Windows) et assurez-vous que vous vous trouvez actuellement dans le dossier SonarQube. Ensuite, vous devriez aller dans le dossier bin par exemple : `cd bin` Dans le dossier "bin" de SonarQube, vous trouverez des dossiers spécifiques pour chaque système d'exploitation. Accédez au dossier correspondant au système d'exploitation de votre machine. Par exemple, sur une machine exécutant Windows 64 bits, la commande sera : `cd windows-x86-64` Maintenant, vous pouvez exécuter SonarQube via la commande :

[Windows] `StartSonar.bat`
[Linux / Mac] `./sonar.sh console`

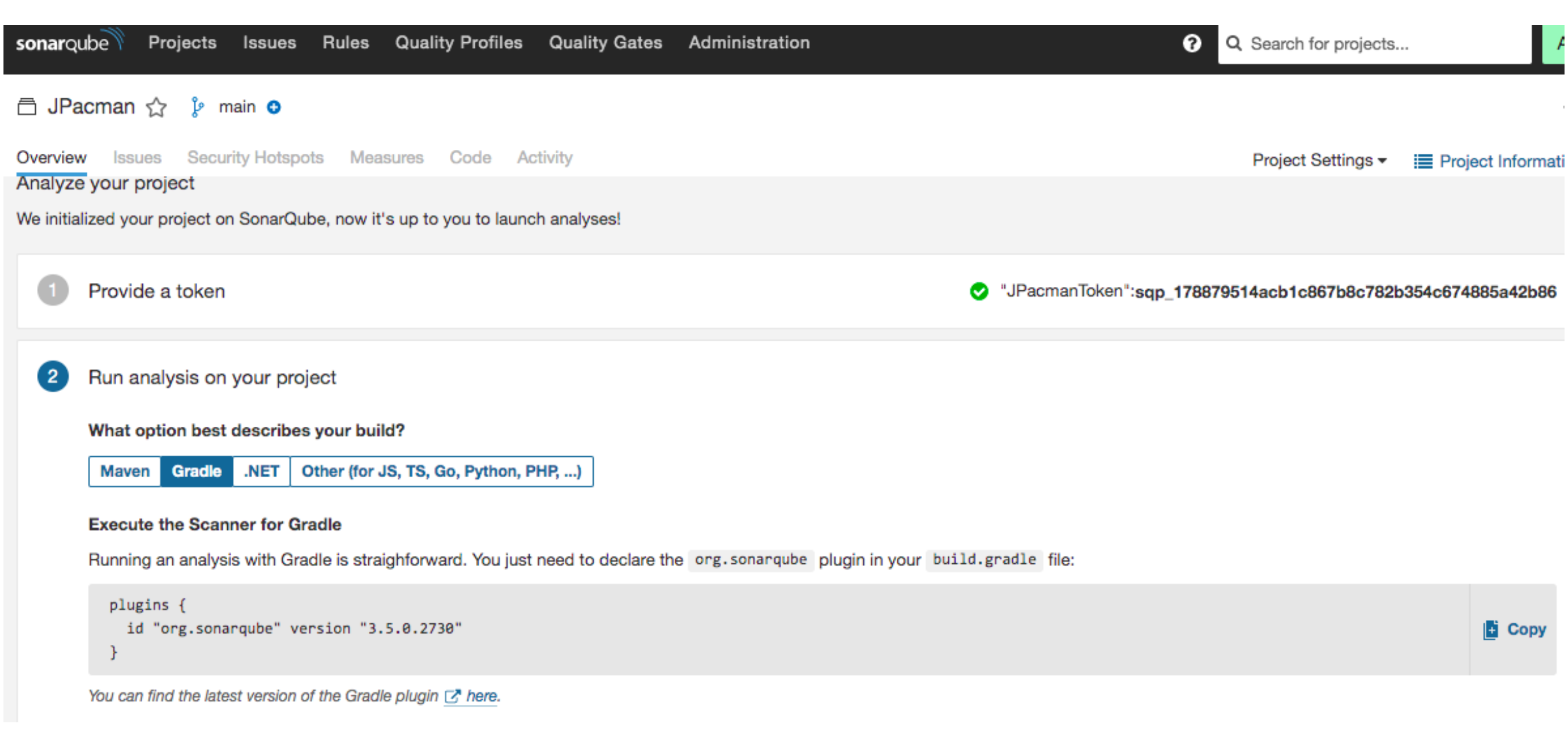
Si vous rencontrez des problèmes, veuillez consulter la documentation SonarQube). Si tout s'est bien déroulé, alors vous devriez voir dans votre terminal un message comme `<date> INFO yyy [ xxxx ] SonarQube is up`. Ouvrez maintenant votre navigateur et saisissez l'adresse suivante:

`http://localhost:9000`

Cliquez sur `Log In`. L'identifiant et le mot de passe est "admin" (vous pouvez changer plus tard le mot de passe si vous le souhaitez). Après la connexion, vous pouvez cliquer sur `Create New Project`. Pour le Projet `Key` et `Display`, tapez "JPacman" (si vous changez le nom, vous devrez adapter d'autres commandes plus tard, car ils utilisent la clé pour identifier le projet).

Maintenant, vous verrez la page `Analyze your Project`. Dans la première étape, pour le nom de token, tapez `jpacmantoken`. Cliquez sur `generate` puis `continue`.

Si tout allait bien, alors vous devriez voir l'étape 2. Sélectionnez `Gradle`. Vous pouvez suivre les instructions données mais pour cette session de laboratoire, vous pouvez simplement continuer à suivre ici. La première chose à faire est donc de mettre à jour votre fichier Gradle de construction (si vous avez récupéré la dernière version du référentiel, ce n'est pas nécessaire).



Dans l'étape 2, vous aurez un code generé:

`id "org.sonarqube" version "X.Y.Z"`

Ouvrez votre IntelliJ avec le code source de JPacman, et dans le dossier racine du projet, vous trouverez le fichier `build.gradle`. Ouvrez le fichier et ajoutez le code generé après la ligne 7 (à l'intérieur de l'élément `plugins`) :

Vous pouvez maintenant utiliser le terminal à l'intérieur d'IntelliJ pour la commande donnée par SonarQube, placez-les tous sur la même ligne (adaptez la commande en conséquence pour Windows) :

```
gradlew sonarqube -Dsonar.projectKey=JPacman -Dsonar.host.url=http://localhost:9000
-Dsonar.login==<generated token code>
```

NB. Sur Mac, utilisez `./gradlew` au lieu de `gradlew`.

Si vous rencontrez des problèmes avec la commande ci-dessus, essayez plutôt celle-ci :

```
./gradlew sonarqube -Dsonar.projectKey=JPacman -Dsonar.host.url=http://localhost:9000
-Dsonar.login=admin -Dsonar.password=admin
```

NB. Assurez-vous d'avoir la bonne version de java dans `sourceCompatibility` et `targetCompatibility` dans le fichier `gradle.properties` du projet JPacman.

Si tout va bien, la page sur SonarQube devrait être mise à jour dans un certain temps avec les informations du code source de JPacman.

4. Travail à réaliser

Partie 1 : JPacman avec CodeScene

Pour la première tâche, nous utiliserons CodeScene pour identifier quels artefacts ont besoin de réusinage. Vous devriez déjà avoir JPacman dans votre CodeScene tel qu'expliqué dans la Section 3 (Installation et préparation).

Cliquez dans le menu "Code" sur le côté gauche, puis cliquez sur "Hotspots" et "Code health" pour visualiser l'état du code. Choisissez une classe et regardez sa `X-ray'', ` code review'' pour voir les candidats de refactoring recommandés.

Cliquez maintenant dans le sous-menu "Hotspot Code Health" (sous "Code") sur le côté gauche. Cela montre une liste des fichiers recommandés pour le refactoring. Sélectionnez un des fichiers dans la liste, puis cliquez sur le bouton "X-ray" ou "Run X-Ray" pour voir plus de détails sur la complexité au niveau des méthodes (l'exécution peut prendre de quelques à plusieurs minutes avant que l'outil affiche de résultat de l'analyse, dépendamment de la taille et la complexité du fichier analysé). Regarder, les couleurs pour les méthodes si existent.

Questions:

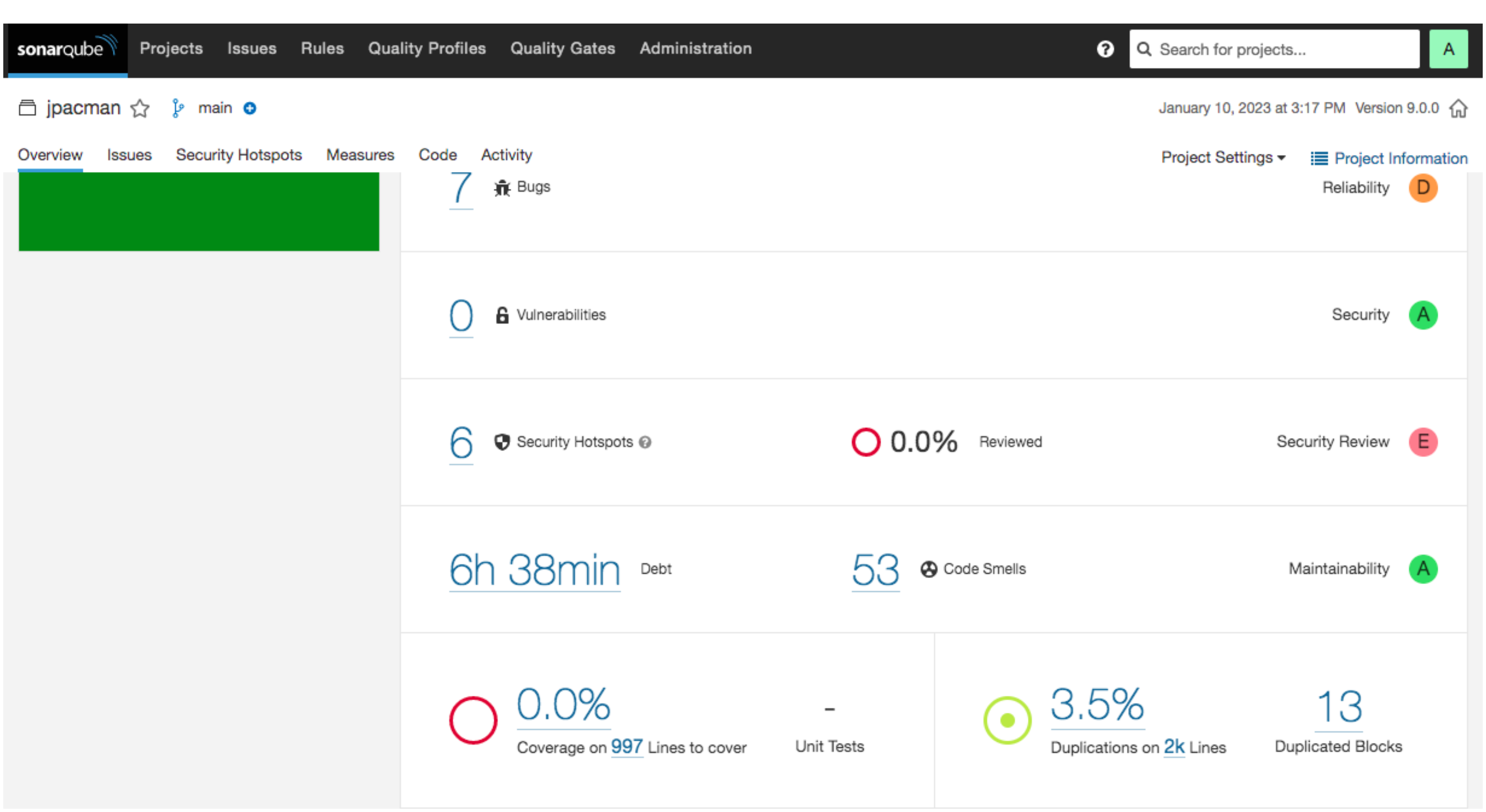
1. CodeScene vous a-t-il aidé à identifier les cibles possibles pour le refactoring ? Si oui, donner la liste d'artéfacts (classes et méthodes) qui ont besoin de refactoring ?

2. CodeScene vous a-t-il donné des recommandations ou des indices sur la façon de refactoriser les cibles proposées ? Comment serait-il possible de procéder avec le refactoring ?

Partie 2 : JPacman avec SonarQube

Pour la deuxième partie, nous allons utiliser un outil plus complexe et dédié pour trouver des cibles de refactoring. Pour la deuxième tâche, nous utiliserons SonarQube pour trouver des cibles de refactoring. Vous devriez déjà avoir SonarQube avec le projet qu'expliqué dans la Section 3 (Installation et préparation).

Si l'installation, c'est déroulé correctement, voici à quoi ressemblera la page de SonarQube une fois l'analyse terminée sur JPacman.



Cliquez sur "Code Smells" et analysez les odeurs détectées. Vous pouvez également voir que SonarQube donne aussi des explications sur les odeurs ("Why this is an issue ?").

Questions:

1. Prenez une capture d'écran de l'écran d'accueil avec le nom du projet.

2. Donnez la liste des artéfacts (classes et méthodes) qui ont besoin de refactoring.

3. Parmi CodeScene et SonarQube, lequel est plus utile pour trouver des cibles de refactoring ?

4. Parmi CodeScene et SonarQube, lequel fournit le meilleur raisonnement / explication sur les cibles de refactoring possibles ?

5. Existe-t-il des artefacts communs entre SonarQube (`Code Smells`) et CodeScene (`Refactoring targets`) ? Si oui, quels sont-ils ?

Partie 3 : Refactoring stratégique avec JPacman

Pour le cours de réingénierie de logiciel, nous valorisons le concept de *refactoring stratégique*, qui consiste à refactoriser avec un objectif. Les outils peuvent aider à identifier les artefacts dont les odeurs pourraient entraîner des problèmes potentiels. Cependant, seul le développeur peut vraiment identifier les artefacts nécessaires pour un objectif spécifique. Faisons cela pour JPacman.

Comme vous l'avez probablement remarqué, il manque à JPacman de nombreuses fonctionnalités présentes dans le jeu Pacman original. Maintenant, l'idée est de commencer à refactoriser le code pour prendre en charge l'une de ces fonctionnalités manquantes : plus de vies (Extra Lives). Dans le Pacman original, le joueur commence avec 3 vies. Dans JPacman, il n'y a qu'une seule vie et en cas de perte : game over. Si vous regardez le code (`/jpacman/src/main/java/nl/tudelft/jpacman/level/Player.java`), il n'y a pas d'attribut pour stocker le nombre de vies qu'il reste au joueur.

Regardez les artefacts de JPacman et planifiez-la ou les refactorings nécessaires pour faciliter l'ajout de plusieurs vies. Vous pouvez commencer par les refactorings les plus simples pour ne pas altérer le code ou procédez de façon raisonnable selon le patron « The Most Valuable First ». Il n'y a pas de mauvaise direction ici, faites ce que vous pensez être le plus facile ou le plus logique pour vous. Veuillez noter que vous n'avez qu'à "**planifier**" les refactorings pour ce laboratoire. (L'application des refactorings est pour la prochaine partie)

NB : La planification est ce qui le plus important pour la partie 3.
NB : Veuillez garder en tête que vous êtes dans la «Partie3 : Refactoring **stratégique** » lorsque vous répondez aux questions ci-dessous.

Questions:

1. Quelles est votre stratégie pour planifier ce refactoring ?

2. Pourquoi considérez-vous ces refactorings comme importants pour votre objectif ?

3. Les outils précédents (CodeScene ou SonarQube) ont-ils identifié les cibles de refactoring que vous jugiez nécessaires pour atteindre cet objectif ?

4. Préférez-vous refactoriser uniquement pour améliorer la qualité du code ou refactoriser avec un objectif ? Pourquoi ?

Partie 4 : Application du refactoring stratégique

Pour cette partie, vous appliquez les refactorings planifiés (dans la Partie 3) sur le code source. N'oubliez pas de vous assurer que vos refactorings ne brisent pas le code.

5. Conditions de réalisation

Le travail est à effectuer en équipes de 3 étudiants au maximum.

6. Aide et discussions

Vous êtes encouragés à discuter du laboratoire et à poser vos questions en utilisant le forum créé à cette fin sur Moodle ou sur Discord. Les membres de chaque équipe sont encouragés à utiliser les channels privés (textuel et vocal) créés pour leur équipe sur Discord pour discuter et travailler en équipe sur les différentes activités du laboratoire.

7. Remise

Le travail doit être remis électroniquement sur Moodle au plus tard le **7 février à 23 h 59**. Vous devrez remettre une archive `zip` ou `tar.gz` contenant tous les fichiers, ainsi qu'un fichier texte indiquant le nom de tous les membres de l'équipe ayant contribué à la réalisation du travail si le nom des membres n'est pas clairement indiqué dans le rapport. Une seule remise électronique est nécessaire par équipe. Remettez aussi individuellement le tableau de contribution tel vu dans le laboratoire précédent. Pour faciliter la correction, vous devez nommer votre dossier parent de la façon suivante :

`LOG530H2024-LabXX-EquipeYY-CodePermanent1_CodePermanent2_CodePermanent3`

Et votre rapport ainsi :

```
EquipeYY-Rapport.pdf
```