Projet : Analyse Statique d'une Application Java avec une Interface Graphique

1. Structure générale du projet

L'architecture du projet suit une organisation claire et modulaire :

`com.hnguyen703.AnalyzerUI`

Point d'entrée de l'application, gère l'interface graphique avec **JavaFX**. JavaFX est utilisé pour créer une interface utilisateur interactive permettant de visualiser les résultats de l'analyse statique.

`com.hnguyen703.analyzer`

Contient la logique principale d'analyse du code :

- `Analyzer` : classe principale qui parcourt les fichiers sources Java et agrège les informations collectées.
- Utilise les classes *visitors* pour parcourir l'arbre syntaxique (AST).

`com.hnguyen703.analyzer.visitors`

Regroupe les visiteurs JDT responsables de l'extraction des informations :

- `TypeDeclarationVisitor` : détecte les classes et interfaces.
- `MethodDeclarationVisitor` : recense les méthodes, leurs signatures et leurs lignes de code.
- `FieldAccessVisitor` et `VariableDeclarationFragmentVisitor` : identifient les attributs.
- `MethodInvocationVisitor` : repère les appels de méthodes (pour le graphe d'appel).

`com.hnguyen703.ui`

Regroupe tous les éléments liés à l'interface graphique.

`ui.components`

- `StatisticsView` : affiche les statistiques calculées (nombre de classes, méthodes, moyennes, etc.).
- `CallGraphView` : visualise le graphe d'appel à l'aide de **GraphStream**.
- `SummaryPanel` : propose une vue synthétique du projet analysé.

`ui.models`

- `ClassStat`, `MethodStat` : structures de données stockant les résultats (nom, nombre de méthodes, lignes, attributs...).

`ui.utils`

- `FileUtils` : gère la lecture des fichiers et répertoires.
- `Constants` : centralise les chemins et paramètres.

`dossier.test`

Regroupe les classes de test :

- `MainTest`, `SampleTest`, `TestRunner` pour vérifier la fiabilité de l'analyse et de l'affichage.

---

2. Fonctionnement de l'analyse

1. **Lecture du projet cible** : `Analyzer` parcourt récursivement les fichiers `.java`.
2. **Visite de l'AST** : les visiteurs extraient les informations sur les classes, attributs, méthodes et appels.
3. **Calcul des statistiques** :
    - Nombre de classes, méthodes, packages.
    - Moyennes (méthodes/classe, lignes/méthode, attributs/classe).
    - Classes ayant le plus de méthodes ou d'attributs.
4. **Construction du graphe d'appel** :
    - Les relations méthode → méthode sont enregistrées par `MethodInvocationVisitor`.
    - `CallGraphView` affiche ces liens sous forme de graphe orienté.

---

3. Interface graphique

L'interface est construite autour de `AnalyzerUI`, composée de deux onglets principaux :

- **Onglet "Statistics"** → affiche les résultats numériques et les classes les plus importantes.
- **Onglet "Call Graph"** → montre visuellement les appels entre méthodes via **GraphStream**.

L'utilisateur peut charger un dossier Java, lancer l'analyse et visualiser instantanément :

- la complexité du projet,
- les zones fortement couplées,
- les classes principales de l'architecture.