Analyser les Packages Java dans un Projet

Voici un script Python qui analyse un répertoire de manière récursive, trouve tous les packages Java utilisés dans les fichiers `.java`, supprime les doublons et liste tous les packages uniques utilisés dans le projet. Le script se concentre uniquement sur les fichiers `.java` et gère les projets qui peuvent avoir des modules en parcourant tous les sous-répertoires.

Script Python :

```
import os
import sys

def find_java_files(root_dir):
    """
    Trouve de manière récursive tous les fichiers .java dans le répertoire donné et ses sous-répertoires.

    Args:
        root_dir (str): Le répertoire racine à partir duquel commencer la recherche.

    Yields:
        str: Le chemin complet de chaque fichier .java.
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        print(f"[INFO] Entrée dans le répertoire: {dirpath}")
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_package(import_statement):
    """
    Extrait le nom du package à partir d'une instruction d'importation.

    Utilise la convention que les noms de packages sont en minuscules, tandis que les noms de classes
    commencent par des lettres majuscules. Gère les importations sauvages (*).

    Args:
        import_statement (str): La ligne d'instruction d'importation d'un fichier Java.

    Returns:
        str: Le nom du package, ou une chaîne vide si non déterminé.
```

```
    """
    parts = import_statement.split()
    if parts[0] == 'import':
        parts = parts[1:]
    if parts[0] == 'static':
        parts = parts[1:]
    import_path = ' '.join(parts).strip(';').strip()
    identifiers = import_path.split('.')
    for i, ident in enumerate(identifiers):
        if ident == '*' or (ident and ident[0].isupper()):
            package_parts = identifiers[:i]
            break
    else:
        package_parts = []
    package = '.'.join(package_parts)
    return package

if __name__ == '__main__':
    # Vérifie si un répertoire est fourni
    if len(sys.argv) < 2:
        print("Utilisation: python script.py <répertoire_racine>")
        sys.exit(1)

    root_dir = sys.argv[1]

    # Vérifie que le répertoire existe
    if not os.path.isdir(root_dir):
        print(f"[ERROR] Le chemin spécifié n'est pas un répertoire: {root_dir}")
        sys.exit(1)

    # Log l'initialisation de l'analyse
    print(f"[INFO] Début de l'analyse du répertoire: {root_dir}")

    # Initialise les variables
    packages = set()
    total_files = 0
    error_files = 0

    # Traite les fichiers Java
    for java_file in find_java_files(root_dir):
```

```
        print(f"[INFO] Traitement du fichier: {java_file}")
        try:
            with open(java_file, 'r', encoding='utf-8') as f:
                for line in f:
                    line = line.strip()
                    if line.startswith('import'):
                        package = extract_package(line)
                        if package:
                            packages.add(package)
            total_files += 1
        except Exception as e:
            print(f"[ERROR] Impossible de lire le fichier {java_file}: {e}")
            error_files += 1
            continue

    # Affiche le résumé
    print(f"[INFO] Total des fichiers Java tentés: {total_files + error_files}")
    print(f"[INFO] Traités avec succès: {total_files}")
    print(f"[INFO] Fichiers avec erreurs: {error_files}")
    print(f"[INFO] Total des packages uniques trouvés: {len(packages)}")

    # Affiche les résultats
    if packages:
        print("[INFO] Analyse terminée. Affichage des packages uniques:")
        for package in sorted(packages):
            print(package)
    else:
        print("[INFO] Aucun package trouvé.")
```

Comment utiliser le script :

1. Enregistrez le script dans un fichier, par exemple `analyze_java_packages.py`.
2. Exécutez le script à partir de la ligne de commande, en fournissant le chemin vers le répertoire racine du projet Java :

   ```
   python analyze_java_packages.py /chemin/vers/votre/projet/java
   ```

3. Le script affichera une liste triée des noms de packages uniques qui sont importés dans les fichiers `.java`.

Ce que fait le script :

- **Trouve les fichiers `.java` :**
  - Utilise `os.walk()` pour parcourir de manière récursive le répertoire et ses sous-répertoires.
  - Identifie tous les fichiers se terminant par `.java`.
- **Extrait les noms de packages :**
  - Analyse chaque fichier `.java` ligne par ligne.
  - Identifie les lignes commençant par `import`.
  - Extrait les noms de packages des instructions d'importation, en gérant :
    - Les importations régulières (par exemple, `import java.util.List;`).
    - Les importations sauvages (par exemple, `import java.util.*;`).
    - Les importations statiques (par exemple, `import static java.util.Collections.sort;`).
  - Utilise une heuristique basée sur les conventions de nommage Java :
    - Les noms de packages sont généralement en minuscules (par exemple, `java.util`).
    - Les noms de classes commencent généralement par des lettres majuscules (par exemple, `List`, `Collections`).
    - Les importations sauvages se terminent par `*`.
- **Supprime les doublons :**
  - Collecte tous les noms de packages dans un `set` pour supprimer automatiquement les doublons.
- **Affiche les résultats :**
  - Convertit l'ensemble en une liste triée et affiche chaque nom de package sur une nouvelle ligne.

Hypothèses et limitations :

- **Instructions d'importation :**
  - Suppose que les instructions d'importation sont sur leurs propres lignes et ne sont pas intégrées dans des commentaires ou des chaînes.
  - Gère à la fois les importations régulières et statiques, y compris les importations sauvages.
- **Conventions de nommage :**
  - S'appuie sur la convention que les noms de packages sont en minuscules et que les noms de classes commencent par des lettres majuscules.
  - Peut ne pas gérer correctement le nommage non standard (par exemple, les noms de packages en majuscules), mais cela est rare en pratique.
- **Encodage des fichiers :**
  - Suppose que les fichiers `.java` sont encodés en UTF-8. Ajustez le paramètre `encoding` si nécessaire.
- **Gestion des erreurs :**

- Inclut une gestion de base des erreurs pour les fichiers illisibles, affichant un avertissement et continuant.

Exemple :

Si votre projet Java a les fichiers `.java` suivants :

- **File1.java :**

```
import java.util.List;
import java.util.*;
import java.lang.String;
```

- **File2.java :**

```
import java.util.Collections;
import static java.util.Collections.sort;
import com.example.myapp.*;
```

L'exécution du script affichera :

```
com.example.myapp
java.lang
java.util
```

Cela liste tous les packages uniques importés dans le projet, triés alphabétiquement.

Notes :

- Le script se concentre sur les packages importés via les instructions `import`, qui représentent les packages externes et internes que le projet utilise.
- Il ne comprend pas les déclarations de packages des fichiers `.java` eux-mêmes (par exemple, `package com.mycompany.myapp;`), car celles-ci définissent les packages auxquels les fichiers appartiennent, et non les packages qu'ils utilisent.
- Puisque le projet peut avoir des modules, le script analyse de manière récursive tous les sous-répertoires pour s'assurer que tous les fichiers `.java` sont traités.