プロジェクトの Java パッケージの分析

ここには、ディレクトリを再帰的に分析し、`.java` ファイルで使用されているすべての Java パッケージを見つけ、重複を削除し、プロジェクトで使用されているすべての一意のパッケージをリスト表示する Python スクリプトがあります。このスクリプトは、`.java` ファイルにのみ焦点を当て、モジュールを持つプロジェクトを処理するためにすべてのサブディレクトリをトラバースします。

Python スクリプト:

```
import os
import sys

def find_java_files(root_dir):
    """
    指定されたディレクトリおよびそのサブディレクトリ内のすべての .java ファイルを再帰的に検索します。

    引数:
        root_dir (str): 検索を開始するルートディレクトリ。

    返却:
        str: 各 .java ファイルのフルパス。
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        print(f"[INFO] ディレクトリに入る: {dirpath}")
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_package(import_statement):
    """
    インポートステートメントからパッケージ名を抽出します。

    パッケージ名は小文字、クラス名は大文字で始まるという規約を使用します。ワイルドカードインポート (*) を

    引数:
        import_statement (str): Java ファイルからのインポートステートメント行。

    返却:
        str: パッケージ名、または決定できない場合は空文字列。
    """
```

```
    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__':
    # ディレクトリが提供されているか確認
    if len(sys.argv) < 2:
        print(" 使用法: python script.py <root_directory>")
        sys.exit(1)

    root_dir = sys.argv[1]

    # ディレクトリが存在するか確認
    if not os.path.isdir(root_dir):
        print(f"[ERROR] 指定されたパスはディレクトリではありません: {root_dir}")
        sys.exit(1)

    # 分析の開始をログに記録
    print(f"[INFO] ディレクトリの分析を開始: {root_dir}")

    # 変数の初期化
    packages = set()
    total_files = 0
    error_files = 0

    # Java ファイルを処理
    for java_file in find_java_files(root_dir):
        print(f"[INFO] ファイルを処理中: {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] ファイル {java_file} を読み取れません: {e}")
            error_files += 1
            continue

    # サマリを表示
    print(f"[INFO] 試行された Java ファイルの合計: {total_files + error_files}")
    print(f"[INFO] 成功して処理された: {total_files}")
    print(f"[INFO] エラーが発生したファイル: {error_files}")
    print(f"[INFO] 見つかった一意のパッケージの合計: {len(packages)}")

    # 結果を表示
    if packages:
        print("[INFO] 分析が完了しました。一意のパッケージを表示:")
        for package in sorted(packages):
            print(package)
    else:
        print("[INFO] パッケージが見つかりませんでした。")
```

スクリプトの使用方法:

1. スクリプトをファイルに保存します。例えば、`analyze_java_packages.py`。
2. コマンドラインからスクリプトを実行し、Java プロジェクトのルートディレクトリのパスを提供します:

   ```
   python analyze_java_packages.py /path/to/your/java/project
   ```

3. スクリプトは、`.java` ファイルでインポートされている一意のパッケージ名のソートされたリストを出力します。

スクリプトの機能:

- **`.java` ファイルの検索:**

  - `os.walk()` を使用してディレクトリとそのサブディレクトリを再帰的にトラバースします。
  - `.java` で終わるすべてのファイルを特定します。

- **パッケージ名の抽出:**
  - 各 `.java` ファイルを行ごとに分析します。
  - `import` で始まる行を特定します。
  - インポートステートメントからパッケージ名を抽出し、以下を処理します:
    * 通常のインポート (例: `import java.util.List;`)
    * ワイルドカードインポート (例: `import java.util.*;`)
    * 静的インポート (例: `import static java.util.Collections.sort;`)
  - Java の命名規約に基づくヒューリスティックを使用します:
    * パッケージ名は通常小文字 (例: `java.util`)。
    * クラス名は通常大文字で始まります (例: `List`, `Collections`)。
    * ワイルドカードインポートは `*` で終わります。

- **重複の削除:**
  - すべてのパッケージ名を `set` に収集して、自動的に重複を削除します。

- **結果の出力:**
  - `set` をソートされたリストに変換し、各パッケージ名を新しい行に表示します。

仮定と制限:

- **インポートステートメント:**
  - インポートステートメントがコメントや文字列に埋め込まれていないことを前提としています。
  - 通常のインポートと静的インポート、ワイルドカードインポートを処理します。

- **命名規約:**
  - パッケージ名は小文字、クラス名は大文字で始まるという規約に依存します。
  - 非標準的な命名 (例: 大文字のパッケージ名) を正しく処理できない場合がありますが、これは実践上稀です。

- **ファイルのエンコーディング:**
  - `.java` ファイルが UTF-8 でエンコードされていることを前提としています。必要に応じて `encoding` パラメータを調整してください。

- **エラー処理:**
  - 読み取れないファイルに対する基本的なエラー処理を含み、警告を表示して続行します。

例:

Java プロジェクトに以下の `.java` ファイルがある場合:

- **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.*;
```

スクリプトを実行すると、以下のように出力されます:

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

これは、プロジェクトでインポートされているすべての一意のパッケージをアルファベット順にリスト表示します。

注意点:

- スクリプトは、`import` ステートメントを介してインポートされる外部および内部パッケージに焦点を当てています。
- `.java` ファイル自身のパッケージ宣言 (例: `package com.mycompany.myapp;`) は含まれません。これらはファイルが属するパッケージを定義するためです。
- プロジェクトがモジュールを持つ場合、スクリプトはすべてのサブディレクトリを再帰的に分析して、すべての `.java` ファイルが処理されるようにします。