分析項目中的 Java 套件

這是一個 Python 腳本，它會遞歸分析目錄，找出所有在 `.java` 文件中使用的 Java 包，移除重複項，並列出項目中使用的所有唯一包。該腳本僅專注於 `.java` 文件，並處理可能有模組的項目，通過遍歷所有子目錄。

Python 腳本：

```
import os
import sys

def find_java_files(root_dir):
    """
    遞歸查找給定目錄及其子目錄中的所有 .java 文件。

    Args:
        root_dir (str): 開始搜索的根目錄。

    Yields:
        str: 每個 .java 文件的完整路徑。
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_package(import_statement):
    """
    從導入語句中提取包名。

    使用包名通常是小寫，而類名通常以大寫字母開頭的約定。對於通配符導入 (*)，包是 '*' 前面的部分。

    Args:
        import_statement (str): Java 文件中的導入語句行。

    Returns:
        str: 包名，如果未確定則為空字符串。
    """
    # 如果存在則移除 'import' 和 'static'
    parts = import_statement.split()
    if parts[0] == 'import':
```

```
        parts = parts[1:]
    if parts[0] == 'static':
        parts = parts[1:]

    # 連接剩餘部分並移除尾隨的 ';'
    import_path = ' '.join(parts).strip(';').strip()

    # 以 '.' 分割導入路徑
    identifiers = import_path.split('.')

    # 找到第一個標識符以大寫字母開頭或是 '*' 的索引
    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__':
    # 檢查命令行參數以獲取根目錄
    if len(sys.argv) < 2:
        print(" 使用方法: python script.py <root_directory>")
        sys.exit(1)

    root_dir = sys.argv[1]
    packages = set()

    # 遍歷目錄及其子目錄中的所有 .java 文件
    for java_file in find_java_files(root_dir):
        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:  # 只添加非空包名
                            packages.add(package)
```

```
        except Exception as e:
            print(f" 警告: 無法讀取文件 {java_file}: {e}")
            continue

    # 打印排序後的唯一包列表
    for package in sorted(packages):
        print(package)
```

如何使用腳本：

1. 將腳本保存到文件中，例如 `analyze_java_packages.py`。
2. 從命令行運行腳本，提供 Java 項目根目錄的路徑：

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

3. 腳本將輸出一個排序後的唯一包名列表，這些包名在 `.java` 文件中被導入。

腳本的功能：

- **查找 `.java` 文件：**
  - 使用 `os.walk()` 遞歸遍歷目錄及其子目錄。
  - 確定所有以 `.java` 結尾的文件。
- **提取包名：**
  - 逐行分析每個 `.java` 文件。
  - 確定以 `import` 開頭的行。
  - 從導入語句中提取包名，處理：
    - 普通導入（例如 `import java.util.List;`）
    - 通配符導入（例如 `import java.util.*;`）
    - 靜態導入（例如 `import static java.util.Collections.sort;`）
  - 使用基於 Java 命名約定的啟發式方法：
    - 包名通常是小寫（例如 `java.util`）。
    - 類名通常以大寫字母開頭（例如 `List`，`Collections`）。
    - 通配符導入以 `*` 結尾。
- **移除重複項：**
  - 將所有包名收集到 `set` 中，自動移除重複項。
- **輸出結果：**
  - 將 `set` 轉換為排序後的列表，並打印每個包名。

假設和限制：

- **導入語句：**
  - 假設導入語句在自己的行上，而不是嵌入在註釋或字符串中。
  - 處理普通和靜態導入，包括通配符導入。
- **命名約定：**
  - 依賴於包名是小寫，類名以大寫字母開頭的約定。
  - 可能無法正確處理非標準命名（例如大寫包名），但在實踐中這種情況很少見。
- **文件編碼：**
  - 假設 `.java` 文件以 UTF-8 編碼。如果需要，請調整 `encoding` 參數。
- **錯誤處理：**
  - 包含基本的錯誤處理，對於無法讀取的文件，打印警告並繼續。

範例：

如果您的 Java 項目有以下 `.java` 文件：

- **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.*;
```

運行腳本將輸出：

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

這列出了項目中導入的所有唯一包，按字母順序排序。

注意事項：

- 腳本專注於通過 `import` 語句導入的包，這些包代表項目使用的外部和內部包。
- 它不包括 `.java` 文件本身的包聲明（例如 `package com.mycompany.myapp;`），因為這些定義了文件屬於的包，而不是它們使用的包。
- 由於項目可能有模組，腳本遞歸分析所有子目錄，以確保處理所有 `.java` 文件。