分析项目中的 Java 包

这是一个 Python 脚本，它递归分析目录，找到所有在 `.java` 文件中使用的 Java 包，去除重复项，并列出项目中使用的所有唯一包。该脚本专注于 `.java` 文件，并通过遍历所有子目录来处理可能包含模块的项目。

Python 脚本：

```
import os
import sys

def find_java_files(root_dir):
    """
    递归查找给定目录及其子目录中的所有 .java 文件。

    参数：
        root_dir (str): 开始搜索的根目录。

    生成：
        str: 每个 .java 文件的完整路径。
    """
    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):
    """
    从导入语句中提取包名。

    使用约定，包名通常是小写的，而类名以大写字母开头。对于通配符导入 (*)，包是 '*' 之前的部分。

    参数：
        import_statement (str): 来自 Java 文件的导入语句行。

    返回：
        str: 包名，如果未确定则为空字符串。
    """
    # 如果存在则删除 'import' 和 'static'
    parts = import_statement.split()
    if parts[0] == 'import':
```

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

    # 连接剩余部分并删除尾部的 ';'
    import_path = ' '.join(parts).strip(';').strip()

    # 通过 '.' 分割导入路径
    identifiers = import_path.split('.')

    # 查找第一个标识符以大写字母开头或是 '*' 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__':
    # 检查命令行参数以获取根目录
    if len(sys.argv) < 2:
        print(" 用法: python script.py <root_directory>")
        sys.exit(1)

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

    # 遍历目录及其子目录中的所有 .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:  # 仅添加非空包名
                            packages.add(package)
```

```
        except Exception as e:
            print(f" 警告：无法读取文件 {java_file}: {e}")
            continue

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

如何使用脚本：

1. 将脚本保存到文件中，例如 `analyze_java_packages.py`。
2. 从命令行运行脚本，提供 Java 项目根目录的路径：

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

3. 脚本将输出一个排序后的唯一包名列表，这些包名在 `.java` 文件中被导入。

脚本功能：

- **查找 `.java` 文件：**
  - 使用 `os.walk()` 递归遍历目录及其子目录。
  - 识别所有以 `.java` 结尾的文件。
- **提取包名：**
  - 逐行分析每个 `.java` 文件。
  - 识别以 `import` 开头的行。
  - 从导入语句中提取包名，处理：
    - 普通导入（例如 `import java.util.List;`）
    - 通配符导入（例如 `import java.util.*;`）
    - 静态导入（例如 `import static java.util.Collections.sort;`）
  - 使用基于 Java 命名约定的启发式方法：
    - 包名通常是小写（例如 `java.util`）。
    - 类名通常以大写字母开头（例如 `List`、`Collections`）。
    - 通配符导入以 `*` 结尾。
- **去除重复：**
  - 将所有包名收集到 `set` 中以自动去除重复。
- **输出结果：**
  - 将 `set` 转换为排序后的列表并打印每个包名。

假设和限制：

- **导入语句：**
  - 假设导入语句在单独的行上，而不是嵌入在注释或字符串中。
  - 处理普通和静态导入，包括通配符导入。
- **命名约定：**
  - 依赖于包名为小写且类名以大写字母开头的约定。
  - 可能无法正确处理非标准命名（例如大写包名），但在实践中这种情况很少见。
- **文件编码：**
  - 假设 `.java` 文件以 UTF-8 编码。如果需要，请调整 `encoding` 参数。
- **错误处理：**
  - 包含基本的错误处理，对于不可读的文件打印警告并继续。

示例：

如果您的 Java 项目有以下 `.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.*;
```

运行脚本将输出：

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

这列出了项目中导入的所有唯一包，按字母顺序排序。

注意事项：

- 脚本专注于通过 `import` 语句导入的包，这些包代表项目使用的外部和内部包。
- 不包括 `.java` 文件本身的包声明（例如 `package com.mycompany.myapp;`），因为这些定义了文件所属的包，而不是它们使用的包。
- 由于项目可能包含模块，脚本递归分析所有子目录，以确保处理所有 `.java` 文件。