Analysieren von Java-Paketen

Basierend auf Ihrer Anfrage möchten Sie ein Skript, bei dem, wenn der `level`-Parameter 0 ist oder nicht angegeben ist, der gesamte Paketname (alle Teile davon) verwendet wird, was `level = 0` als "unendlich" oder "verwende alle Teile"behandelt, anstatt es auf eine bestimmte Anzahl von Teilen zu beschränken (wie nur den ersten Teil, wenn `level = 1`). Unten werde ich erklären, wie dies in einem Python-Skript implementiert werden kann, das Java-Dateien analysiert, um Paketnamen zu extrahieren, und ich werde eine vollständige Lösung bereitstellen, die auf Ihre Anfrage zugeschnitten ist.

Verständnis der Anforderung

- **Wenn `level = 0` oder nicht angegeben**: Verwenden Sie den vollständigen Paketnamen, wie er aus Java `import`-Anweisungen extrahiert wird (z. B. `java.util` oder `com.google.common.eventbus`), ohne ihn zu kürzen.
- **Wenn `level > 0`**: Kürzen Sie den Paketnamen auf die ersten `level` Teile (z. B. für `com.google.common.eventbus`, `level = 2` würde `com.google` ergeben).
- Das Skript sollte weiterhin Java-Dateien verarbeiten, Paketnamen aus `import`-Anweisungen extrahieren und Randfälle angemessen behandeln.

Lösung

1. **Argument Parsing**:
   - Wenn nur das Stammverzeichnis angegeben ist (z. B. `python script.py /pfad/zu/verzeichnis`), setzen Sie `level = 0`, was bedeutet, dass der vollständige Paketname verwendet wird.
   - Wenn `level` angegeben ist (z. B. `python script.py /pfad/zu/verzeichnis 2`), verwenden Sie es, um den Paketnamen zu kürzen, und stellen Sie sicher, dass es eine nicht-negative ganze Zahl ist.
   - Beenden Sie mit einem Fehler, wenn die Argumente ungültig sind.
2. **Paket Extraktion**:
   - Extrahieren Sie den Paketnamen aus `import`-Anweisungen (z. B. `import java.util.ArrayList;` ergibt `java.util`).
   - Verwenden Sie Java-Benennungskonventionen: Pakete sind in der Regel Kleinbuchstaben, und Klassennamen beginnen mit Großbuchstaben.
3. **Paketkürzungslogik**:
   - Wenn `level = 0`, fügen Sie den vollständigen Paketnamen zur Ergebnismenge hinzu.
   - Wenn `level > 0`, teilen Sie den Paketnamen durch Punkte (`.`) und nehmen Sie die ersten `level` Teile.
4. **Protokollierung und Ausgabe**:
   - Geben Sie klar an, ob vollständige Paketnamen oder gekürzte verwendet werden.

- Listen Sie alle eindeutigen Pakete auf, die gefunden wurden.

Implementierung

Hier ist das vollständige Python-Skript, das Ihren Anforderungen entspricht:

```
import os
import sys
from collections import Counter

def find_java_files(root_dir):
    """
    Durchsucht rekursiv alle .java-Dateien im angegebenen Verzeichnis und seinen Unterverzeichnissen.

    Args:
        root_dir (str): Das Stammverzeichnis, von dem die Suche beginnen soll.

    Yields:
        str: Der vollständige Pfad zu jeder .java-Datei.
    """
    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):
    """
    Extrahiert den Paketnamen aus einer import-Anweisung.

    Verwendet die Konvention, dass Paketnamen Kleinbuchstaben sind, während Klassennamen mit Großbuchstaben b
    Behandelt Wildcard-Imports (*).

    Args:
        import_statement (str): Die import-Anweisungszeile aus einer Java-Datei.

    Returns:
        str: Der Paketname oder eine leere Zeichenkette, wenn nicht bestimmt.
    """
    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__':
    # Parsen der Befehlszeilenargumente
    if len(sys.argv) < 2 or len(sys.argv) > 4:
        print("Verwendung: python script.py <root_directory> [level] [--count]")
        sys.exit(1)

    root_dir = sys.argv[1]
    level = 0
    count = False

    if len(sys.argv) == 3:
        if sys.argv[2] == "--count":
            count = True
        elif sys.argv[2].isdigit():
            level = int(sys.argv[2])
        else:
            print(f"Ungültiges Argument: {sys.argv[2]}")
            sys.exit(1)
    elif len(sys.argv) == 4:
        if sys.argv[3] == "--count" and sys.argv[2].isdigit():
            level = int(sys.argv[2])
            count = True
        else:
            print(f"Ungültige Argumente: {sys.argv[2]} {sys.argv[3]}")
            sys.exit(1)
```

```
# Überprüfen, ob das Verzeichnis existiert
if not os.path.isdir(root_dir):
    print(f"[FEHLER] Der angegebene Pfad ist kein Verzeichnis: {root_dir}")
    sys.exit(1)

# Protokollieren des Analysebeginns
level_str = "Verwenden vollständiger Paketnamen" if level == 0 else f"auf Ebene {level}"
count_str = "mit Auftrittszahlen" if count else ""
print(f"[INFO] Analyse des Verzeichnisses beginnt: {root_dir} {level_str} {count_str}")

# Initialisieren von Variablen
package_counter = Counter()
total_files = 0
error_files = 0

# Verarbeiten der Java-Dateien
for java_file in find_java_files(root_dir):
    try:
        with open(java_file, 'r', encoding='utf-8') as f:
            file_packages = set()
            for line in f:
                line = line.strip()
                if line.startswith('import'):
                    package = extract_package(line)
                    if package:
                        if level > 0:
                            parts = package.split('.')
                            truncated_package = '.'.join(parts[:level])
                        else:
                            truncated_package = package
                        file_packages.add(truncated_package)
        for pkg in file_packages:
            package_counter[pkg] += 1
        total_files += 1
    except Exception as e:
        print(f"[FEHLER] Datei konnte nicht gelesen werden {java_file}: {e}")
        error_files += 1
        continue

# Ausgabe der Zusammenfassung
```

```
print(f"[INFO] Gesamtzahl der versuchten Java-Dateien: {total_files + error_files}")
print(f"[INFO] Erfolgreich verarbeitet: {total_files}")
print(f"[INFO] Dateien mit Fehlern: {error_files}")
if count:
    print(f"[INFO] Gesamtzahl eindeutiger Pakete mit Zählungen: {len(package_counter)}")
else:
    print(f"[INFO] Gesamtzahl eindeutiger Pakete: {len(package_counter)}")

# Ausgabe der Ergebnisse mit angemessener Sortierung
if package_counter:
    if count:
        print("[INFO] Analyse abgeschlossen. Drucken eindeutiger Pakete mit Zählungen (sortiert nach abneh
        # Sortieren nach abnehmender Zählung, dann nach Paketnamen aufsteigend
        for pkg, cnt in sorted(package_counter.items(), key=lambda x: (-x[1], x[0])):
            print(f"{pkg}: {cnt}")
    else:
        print("[INFO] Analyse abgeschlossen. Drucken eindeutiger Pakete (sortiert nach aufsteigendem Namen
        # Sortieren nach aufsteigendem Paketnamen
        for pkg in sorted(package_counter):
            print(pkg)
else:
    print("[INFO] Keine Pakete gefunden.")
```

Funktionsweise

- **Ausführen des Skripts**:
  - `python script.py /pfad/zu/java/projekt`: Analysiert alle `.java`-Dateien im Verzeichnis und verwendet vollständige Paketnamen (`level = 0`).
  - `python script.py /pfad/zu/java/projekt 2`: Kürzt Paketnamen auf die ersten 2 Teile (z. B. `com.google.common.eventbus` wird `com.google`).
- **Beispielausgabe**: Angenommen, Sie haben eine Java-Datei mit:

```
import java.util.ArrayList;
import com.google.common.eventbus.EventBus;
```

  - **Mit `level = 0` (oder kein Level angegeben)**:

```
[INFO] Analyse des Verzeichnisses beginnt: /pfad/zu/java/projekt mit vollständigen Paketnamen.
[INFO] Betreten des Verzeichnisses: /pfad/zu/java/projekt
[INFO] Verarbeiten der Datei: /pfad/zu/java/projekt/MyFile.java
[INFO] Gesamtzahl der versuchten Java-Dateien: 1
```

```
[INFO] Erfolgreich verarbeitet: 1
[INFO] Dateien mit Fehlern: 0
[INFO] Gesamtzahl eindeutiger vollständiger Pakete: 2
[INFO] Analyse abgeschlossen. Drucken eindeutiger vollständiger Pakete:
com.google.common.eventbus
java.util
```

- **Mit `level = 2`**:

```
[INFO] Analyse des Verzeichnisses beginnt: /pfad/zu/java/projekt auf Ebene: 2
[INFO] Betreten des Verzeichnisses: /pfad/zu/java/projekt
[INFO] Verarbeiten der Datei: /pfad/zu/java/projekt/MyFile.java
[INFO] Gesamtzahl der versuchten Java-Dateien: 1
[INFO] Erfolgreich verarbeitet: 1
[INFO] Dateien mit Fehlern: 0
[INFO] Gesamtzahl eindeutiger Pakete auf Ebene 2: 2
[INFO] Analyse abgeschlossen. Drucken eindeutiger Pakete auf Ebene 2:
com.google
java.util
```

- **Wichtige Merkmale**:
  - **Vollständige Paketnamen**: Wenn `level = 0` oder nicht angegeben, wird der gesamte Paketname verwendet, wie er extrahiert wird (z. B. `java.util`, `com.google.common.eventbus`).
  - **Kürzung**: Wenn `level > 0`, wird der erste `level` Teile genommen.
  - **Fehlerbehandlung**: Überprüft auf gültiges Verzeichnis und nicht-negative `level`.
  - **Einzigartigkeit**: Speichert Pakete in einer `set`, um Duplikate zu vermeiden.

Randfälle

- **Fehlendes Level**: Standardmäßig `level = 0`, verwendet vollständige Paketnamen.
- **Ungültiges Level**: Beendet mit einem Fehler, wenn `level` negativ oder keine ganze Zahl ist.
- **Kurze Pakete**: Wenn ein Paket weniger Teile als `level` hat (z. B. `java` mit `level = 2`), wird das vollständige Paket (`java`) verwendet.
- **Leere Imports**: Überspringt Zeilen, aus denen kein Paket extrahiert wird.