□□□□ □□□□□□□□□ □□ □□□□□□□ का विश्लेषण

यह एक □□□□□□ स्क्रिप्ट है जो एक डायरेक्टरी को रिकर्सिव रूप से विश्लेषित करता है, सभी `.java` फाइलों में उपयोग किए गए सभी □□□□ पैकेजों को ढूंढता है, डुप्लिकेट्स को हटाता है, और प्रोजेक्ट में उपयोग किए गए सभी अननिक पैकेजों की सूची बनाता है। स्क्रिप्ट केवल `.java` फाइलों पर ध्यान केंद्रित करता है और मॉड्यूल हो सकते हैं, सभी सबडायरेक्टरी को ट्रैवर्स करने के द्वारा प्रोजेक्टों को संभालता है।

□□□□□□ **स्क्रिप्ट:**

```
import os
import sys

def find_java_files(root_dir):
    """
                            .java

    Args:
        root_dir (str):

    Yields:
        str:     .java
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        print(f"[INFO]                 : {dirpath}")
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_package(import_statement):
    """



                                                                    (*)


    Args:
        import_statement (str):   Java

    Returns:
        str:     ,
    """
    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__':
    #
    if len(sys.argv) < 2:
        print("   : python script.py <root_directory>")
        sys.exit(1)

    root_dir = sys.argv[1]

    #
    if not os.path.isdir(root_dir):
        print(f"[ERROR]                  : {root_dir}")
        sys.exit(1)

    #
    print(f"[INFO]                  : {root_dir}")

    #
    packages = set()
    total_files = 0
    error_files = 0

    # Java
    for java_file in find_java_files(root_dir):
        print(f"[INFO]                : {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]     {java_file}            : {e}")
            error_files += 1
            continue

    #
    print(f"[INFO]    Java          : {total_files + error_files}")
    print(f"[INFO]                 : {total_files}")
    print(f"[INFO]            : {error_files}")
    print(f"[INFO]              : {len(packages)}")

    #
    if packages:
        print("[INFO]                              :")
        for package in sorted(packages):
            print(package)
    else:
        print("[INFO]             ")
```

स्क्रिप्ट का उपयोग कैसे करें:

1. स्क्रिप्ट को एक फाइल में सेभ करें, उदाहरण के लिए, `analyze_java_packages.py`।
2. कमांड लाइन से स्क्रिप्ट चलाएं, □□□□ प्रोजेक्ट के मूल डायरेक्टरी के पथ प्रदान करते हुए:

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

3. स्क्रिप्ट `.java` फाइलों में इम्पोर्ट किए गए अननिक पैकेज नामों की सॉर्टेड सूची को आउटपुट करेगा।

स्क्रिप्ट क्या करता है:

- **.□□□□ फाइलों को ढूंढता है:**
  - `os.walk()` का उपयोग करके डायरेक्टरी और उसके सबडायरेक्टरी को रिकर्सिव रूप से ट्रैवर्स करता है।

- सभी `.java` से समाप्त होने वाले फाइलों को पहचानता है।

 पैकेज नाम निकालता है:

- प्रत्येक `.java` फाइल को लाइन द्वारा लाइन विश्लेषित करता है।
- `import` से शुरू होने वाले लाइनों को पहचानता है।
- इम्पोर्ट स्टेटमेंट से पैकेज नाम निकालता है, संभालता है:
  * सामान्य इम्पोर्ट (उदाहरण के लिए, `import java.util.List;`)
  * वाइल्डकार्ड इम्पोर्ट (उदाहरण के लिए, `import java.util.*;`)
  * स्टैटिक इम्पोर्ट (उदाहरण के लिए, `import static java.util.Collections.sort;`)
- □□□□ नामकरण परंपराओं पर आधारित एक हीयूरिस्टिक का उपयोग करता है:
  * पैकेज नाम आमतौर पर लोअरकेस होते हैं (उदाहरण के लिए, `java.util`).
  * क्लास नाम आमतौर पर अप्परकेस लेटर से शुरू होते हैं (उदाहरण के लिए, `List`, `Collections`).
  * वाइल्डकार्ड इम्पोर्ट `*` से समाप्त होते हैं।

 डुप्लिकेट्स को हटाता है:

- सभी पैकेज नामों को एक `set` में इकट्ठा करता है ताकि डुप्लिकेट्स को स्वचालित रूप से हटाया जा सके।

 नतिजे आउटपुट करता है:

- सेट को एक सॉर्टेड सूची में बदलता है और प्रत्येक पैकेज नाम को नए लाइन पर प्रिंट करता है।

अनुमान और सीमाएं:

 इम्पोर्ट स्टेटमेंट:

- इम्पोर्ट स्टेटमेंट को अपने लाइन पर और टिप्पणियों या स्ट्रिंग्स में नहीं समाहित होने की अनुमान लगाता है।
- दोनों सामान्य और स्टैटिक इम्पोर्ट्स को संभालता है, जिसमें वाइल्डकार्ड इम्पोर्ट शामिल हैं।

 नामकरण परंपराएं:

- पैकेज नाम लोअरकेस होते हैं और क्लास नाम अप्परकेस लेटर से शुरू होते हैं, इस पर निर्भर करता है।
- नॉन-स्टैंडर्ड नामकरण (उदाहरण के लिए, अप्परकेस पैकेज नाम) को सही ढंग से संभालने में असमर्थ हो सकता है, लेकिन यह प्रैक्टिस में दुर्लभ है।

 फाइल एन्कोडिंग:

- `.java` फाइलों को □□□-8 में एन्कोड किया गया मानता है। आवश्यकता के अनुसार `encoding` पैरामीटर को समायोजित करें।

 गलती संभालना:

- पढ़ने में असमर्थ फाइलों के लिए बुनियादी गलती संभालना शामिल है, एक चेतावनी प्रिंट करता है और जारी रहता है।

उदाहरण:

अगर आपका □□□□ प्रोजेक्ट निम्नलिखित `.java` फाइलों को रखता है:

 □□□□1.□□□□:

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

- **□□□□2.□□□□:**

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

स्क्रिप्ट चलाने पर आउटपुट होगा:

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

यह प्रोजेक्ट में इम्पोर्ट किए गए सभी अननिक पैकेजों की सूची को अल्फाबेटिक क्रम में सॉर्ट करता है।

नोट्स:

- स्क्रिप्ट केवल `import` स्टेटमेंट्स के माध्यम से इम्पोर्ट किए गए पैकेजों पर ध्यान केंद्रित करता है, जो प्रोजेक्ट द्वारा उपयोग किए गए बाहरी और आंतरिक पैकेजों को दर्शाता है।
- यह `.java` फाइलों के पैकेज घोषणाओं को शामिल नहीं करता है (उदाहरण के लिए, `package com.mycompany.myapp;`), क्योंकि ये पैकेजों को दर्शाते हैं जिनमें फाइलों का समावेश होता है, न कि वे पैकेज जो वे उपयोग करते हैं।
- क्योंकि प्रोजेक्ट में मॉड्यूल हो सकते हैं, स्क्रिप्ट सभी सबडायरेक्टरी को रिकर्सिव रूप से विश्लेषित करता है ताकि सभी `.java` फाइलों को प्रोसेस किया जा सके।