Analyzing Java Packages in Project

Here's a Python script that analyzes a directory recursively, finds all Java packages used in `.java` files, removes duplicates, and lists all unique packages used in the project. The script focuses solely on `.java` files and handles projects that may have modules by traversing all subdirectories.

Python Script:

```
import os
import sys

def find_java_files(root_dir):
    """
    Recursively find all .java files in the given directory and its subdirectories.

    Args:
        root_dir (str): The root directory to start the search from.

    Yields:
        str: The full path to each .java file.
    """
    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):
    """
    Extract the package name from an import statement.

    Uses the convention that package names are typically lowercase, while class names
    start with uppercase letters. For wildcard imports (*), the package is the part before '*'.

    Args:
        import_statement (str): The import statement line from a Java file.

    Returns:
        str: The package name, or empty string if not determined.
    """
    # Remove 'import' and 'static' if present
```

```
    parts = import_statement.split()
    if parts[0] == 'import':
        parts = parts[1:]
    if parts[0] == 'static':
        parts = parts[1:]

    # Join remaining parts and remove trailing ';'
    import_path = ' '.join(parts).strip(';').strip()

    # Split the import path by '.'
    identifiers = import_path.split('.')

    # Find the index where the first identifier starts with uppercase or is '*'
    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__':
    # Check command-line argument for root directory
    if len(sys.argv) < 2:
        print("Usage: python script.py <root_directory>")
        sys.exit(1)

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

    # Traverse all .java files in the directory and subdirectories
    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:  # Only add non-empty package names
                        packages.add(package)
        except Exception as e:
            print(f"Warning: Could not read file {java_file}: {e}")
            continue

    # Print the sorted list of unique packages
    for package in sorted(packages):
        print(package)
```

How to Use the Script:

1. Save the script to a file, e.g., `analyze_java_packages.py`.
2. Run the script from the command line, providing the path to the root directory of the Java project:

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

3. The script will output a sorted list of unique package names that are imported in the `.java` files.

What the Script Does:

- **Finds `.java` Files:**
  - Uses `os.walk()` to recursively traverse the directory and its subdirectories.
  - Identifies all files ending with `.java`.
- **Extracts Package Names:**
  - Analyzes each `.java` file line by line.
  - Identifies lines starting with `import`.
  - Extracts package names from import statements, handling:
    - Regular imports (e.g., `import java.util.List;`)
    - Wildcard imports (e.g., `import java.util.*;`)
    - Static imports (e.g., `import static java.util.Collections.sort;`)
  - Uses a heuristic based on Java naming conventions:
    - Package names are typically lowercase (e.g., `java.util`).
    - Class names typically start with uppercase letters (e.g., `List`, `Collections`).
    - Wildcard imports end with `*`.
- **Removes Duplicates:**
  - Collects all package names in a `set` to automatically remove duplicates.
- **Outputs Results:**
  - Converts the set to a sorted list and prints each package name on a new line.

Assumptions and Limitations:

- **Import Statements:**
  - Assumes import statements are on their own lines and not embedded in comments or strings.
  - Handles both regular and static imports, including wildcard imports.
- **Naming Conventions:**
  - Relies on the convention that package names are lowercase and class names start with uppercase letters.
  - May not correctly handle non-standard naming (e.g., uppercase package names), but this is rare in practice.
- **File Encoding:**
  - Assumes `.java` files are encoded in UTF-8. Adjust the `encoding` parameter if necessary.
- **Error Handling:**
  - Includes basic error handling for unreadable files, printing a warning and continuing.

Example:

If your Java project has the following `.java` files:

- **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.*;
```

Running the script will output:

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

This lists all unique packages imported in the project, sorted alphabetically.

Notes:

- The script focuses on packages imported via `import` statements, which represent the external and internal packages the project uses.
- It does not include the package declarations of the `.java` files themselves (e.g., `package com.mycompany.myapp;`), as these define the packages the files belong to, not packages they use.
- Since the project may have modules, the script recursively analyzes all subdirectories to ensure all `.java` files are processed.