# Analyzing Java Packages

Based on your query, you want a script where, if the `level` parameter is 0 or not provided, it uses the entire package name (all parts of it), essentially treating `level = 0` as "infinite"or "use all parts,"instead of limiting it to a specific number of parts (like only the first part when `level = 1`). Below, I'll explain how this can be implemented in a Python script that analyzes Java files to extract package names, and I'll provide a complete solution tailored to your request.

### Understanding the Requirement

- **When `level = 0` or not provided**: Use the full package name as extracted from Java `import` statements (e.g., `java.util` or `com.google.common.eventbus`), without truncating it.
- **When `level > 0`**: Truncate the package name to the first `level` parts (e.g., for `com.google.common.eventbus`, `level = 2` would give `com.google`).
- The script should still process Java files, extract package names from `import` statements, and handle edge cases appropriately.

### Solution Approach

1. **Argument Parsing**:
   - If only the root directory is provided (e.g., `python script.py /path/to/dir`), set `level = 0`, meaning use the full package name.
   - If `level` is provided (e.g., `python script.py /path/to/dir 2`), use it to truncate the package name, ensuring it's a non-negative integer.
   - Exit with an error if the arguments are invalid.
2. **Package Extraction**:
   - Extract the package name from `import` statements (e.g., `import java.util.ArrayList;` yields `java.util`).
   - Use Java naming conventions: packages are typically lowercase, and class names start with uppercase.
3. **Package Truncation Logic**:
   - If `level = 0`, add the full package name to the result set.
   - If `level > 0`, split the package name by dots (`.`) and take the first `level` parts.
4. **Logging and Output**:
   - Clearly indicate whether full package names or truncated ones are being used.
   - List all unique packages found.

**Implementation**

Here's the complete Python script that meets your requirements:

```
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):
        print(f"[INFO] Entering directory: {dirpath}")
        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.

    Assumes package names are lowercase and stops at class names (uppercase) or wildcards (*).

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

    Returns:
        str: The package name, or empty string if not determined.
    """
    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__':
    # Parse command-line arguments
    if len(sys.argv) == 2:
        root_dir = sys.argv[1]
        level = 0  # No level provided, use full package names
    elif len(sys.argv) == 3:
        root_dir = sys.argv[1]
        try:
            level = int(sys.argv[2])
            if level < 0:
                raise ValueError
        except ValueError:
            print("[ERROR] Level must be a non-negative integer.")
            sys.exit(1)
    else:
        print("Usage: python script.py <root_directory> [level]")
        sys.exit(1)

    # Verify the directory exists
    if not os.path.isdir(root_dir):
        print(f"[ERROR] The specified path is not a directory: {root_dir}")
        sys.exit(1)

    # Log the start of the analysis
    if level == 0:
        print(f"[INFO] Starting analysis of directory: {root_dir} with full package names.")
    else:
        print(f"[INFO] Starting analysis of directory: {root_dir} at level: {level}")

    # Initialize variables
```

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

# Process Java files
for java_file in find_java_files(root_dir):
    print(f"[INFO] Processing file: {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:
                        if level == 0:
                            # Use the full package name
                            packages.add(package)
                        else:
                            # Truncate to the first 'level' parts
                            parts = package.split('.')
                            truncated_package = '.'.join(parts[:level])
                            packages.add(truncated_package)
        total_files += 1
    except Exception as e:
        print(f"[ERROR] Could not read file {java_file}: {e}")
        error_files += 1
        continue

# Print summary
print(f"[INFO] Total Java files attempted: {total_files + error_files}")
print(f"[INFO] Successfully processed: {total_files}")
print(f"[INFO] Files with errors: {error_files}")
if level == 0:
    print(f"[INFO] Total unique full packages: {len(packages)}")
else:
    print(f"[INFO] Total unique packages at level {level}: {len(packages)}")

# Print results
if packages:
    if level == 0:
```

```
            print("[INFO] Analysis complete. Printing unique full packages:")
        else:
            print(f"[INFO] Analysis complete. Printing unique packages at level {level}:")
        for package in sorted(packages):
            print(package)
    else:
        print("[INFO] No packages found.")
```

### How It Works

- **Running the Script**:
  - `python script.py /path/to/java/project`: Analyzes all `.java` files in the directory and uses full package names (`level = 0`).
  - `python script.py /path/to/java/project 2`: Truncates package names to the first 2 parts (e.g., `com.google.common.eventbus` becomes `com.google`).
- **Example Output**: Suppose you have a Java file with:

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

  - **With `level = 0` (or no level provided)**:

```
[INFO] Starting analysis of directory: /path/to/java/project with full package names.
[INFO] Entering directory: /path/to/java/project
[INFO] Processing file: /path/to/java/project/MyFile.java
[INFO] Total Java files attempted: 1
[INFO] Successfully processed: 1
[INFO] Files with errors: 0
[INFO] Total unique full packages: 2
[INFO] Analysis complete. Printing unique full packages:
com.google.common.eventbus
java.util
```

  - **With `level = 2`**:

```
[INFO] Starting analysis of directory: /path/to/java/project at level: 2
[INFO] Entering directory: /path/to/java/project
[INFO] Processing file: /path/to/java/project/MyFile.java
[INFO] Total Java files attempted: 1
[INFO] Successfully processed: 1
[INFO] Files with errors: 0
[INFO] Total unique packages at level 2: 2
```

```
[INFO] Analysis complete. Printing unique packages at level 2:
com.google
java.util
```

- **Key Features**:
  - **Full Package Names**: When `level = 0` or not passed, it uses the entire package name as extracted (e.g., `java.util`, `com.google.common.eventbus`).
  - **Truncation**: When `level > 0`, it takes the first `level` parts.
  - **Error Handling**: Checks for valid directory and non-negative `level`.
  - **Uniqueness**: Stores packages in a `set` to avoid duplicates.

**Edge Cases Handled**

- **Missing Level**: Defaults to `level = 0`, using full package names.
- **Invalid Level**: Exits with an error if `level` is negative or not an integer.
- **Short Packages**: If a package has fewer parts than `level` (e.g., `java` with `level = 2`), it uses the full package (`java`).
- **Empty Imports**: Skips lines where no package is extracted.

This script fully satisfies your requirement: when `level = 0` or not provided, it uses all parts of the package name, effectively treating it as “infinite”in terms of parts used.