Análisis de Dependencias de Java

Para abordar su consulta, modificaré el script de Python proporcionado para generar un grafo de dependencias para clases de Java, donde las dependencias se agrupan por un nivel especificado de la jerarquía de paquetes. Por ejemplo, si `level=1`, los paquetes se agrupan en el nivel superior (por ejemplo, “org”), y si `level=2`, se agrupan en el segundo nivel (por ejemplo, “org.springframework”). Esto significa que antes de analizar las dependencias, acumulamos las clases en grupos basados en las primeras `level` partes de sus nombres de paquete, y el grafo mostrará las dependencias entre estos grupos en lugar de clases individuales.

A continuación se muestra el script modificado, seguido de una explicación de los cambios y cómo cumple con sus requisitos.

Script Modificado

```
import os
import sys
import re
from collections import defaultdict

def get_package(file_path):
    """
    Extrae el nombre del paquete de un archivo .java.

    Args:
        file_path (str): Ruta al archivo .java.

    Returns:
        str: El nombre del paquete, o None si no se encuentra.
    """
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            for line in f:
                match = re.search(r'^\s*package\s+([\w.]+);', line)
                if match:
                    return match.group(1)
    except Exception as e:
        print(f"Advertencia: No se pudo leer {file_path}: {e}")
    return None

def get_specific_imports(file_path):
```

```
    """
    Extrae importaciones de clases específicas de un archivo .java, excluyendo importaciones comodín.

    Args:
        file_path (str): Ruta al archivo .java.

    Returns:
        list: Lista de nombres de clases importadas completamente calificadas.
    """
    imports = []
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            for line in f:
                match = re.search(r'^\s*import\s+([\w.]+);', line)
                if match:
                    imp = match.group(1)
                    # Excluir importaciones comodín (por ejemplo, import java.util.*;)
                    if not imp.endswith('.*'):
                        imports.append(imp)
    except Exception as e:
        print(f"Advertencia: No se pudo leer {file_path}: {e}")
    return imports

def get_package_group(full_class_name, level):
    """
    Obtiene el grupo de paquetes basado en las primeras 'level' partes del nombre del paquete.

    Args:
        full_class_name (str): Nombre de clase completamente calificado (por ejemplo, "org.springframework.bo
        level (int): Número de niveles de paquete a incluir (por ejemplo, 1 para "org", 2 para "org.springfra

    Returns:
        str: El grupo de paquetes (por ejemplo, "org" o "org.springframework").
    """
    package = '.'.join(full_class_name.split('.')[:-1])  # Extraer paquete, excluyendo el nombre de la clase
    parts = package.split('.')
    if len(parts) <= level:
        return package  # Usar el paquete completo si tiene menos o igual partes que level
    else:
        return '.'.join(parts[:level])  # Usar las primeras 'level' partes
```

```
if __name__ == '__main__':
    # Verificar argumentos de línea de comandos: root_directory y level
    if len(sys.argv) != 3:
        print("Uso: python script.py <root_directory> <level>")
        sys.exit(1)

    root_dir = sys.argv[1]
    try:
        level = int(sys.argv[2])
        if level < 1:
            raise ValueError
    except ValueError:
        print("Error: level debe ser un entero positivo")
        sys.exit(1)

    all_classes = set()

    # Primera pasada: Recopilar todos los nombres de clases completamente calificadas en el proyecto
    for root, dirs, files in os.walk(root_dir):
        for file in files:
            if file.endswith('.java'):
                file_path = os.path.join(root, file)
                package = get_package(file_path)
                if package:
                    class_name = file.replace('.java', '')
                    full_class_name = f"{package}.{class_name}"
                    all_classes.add(full_class_name)

    # Almacenar dependencias entre grupos de paquetes
    group_dependencies = set()

    # Segunda pasada: Analizar dependencias basadas en grupos de paquetes
    for root, dirs, files in os.walk(root_dir):
        for file in files:
            if file.endswith('.java'):
                file_path = os.path.join(root, file)
                package = get_package(file_path)
                if package:
                    class_name = file.replace('.java', '')
```

```
                full_class_name = f"{package}.{class_name}"
                importer_group = get_package_group(full_class_name, level)
                imports = get_specific_imports(file_path)
                for imp in imports:
                    # Solo incluir dependencias en clases dentro del proyecto
                    # Excluir auto-dependencias
                    if imp in all_classes and imp != full_class_name:
                        imported_group = get_package_group(imp, level)
                        if imported_group != importer_group:
                            group_dependencies.add((importer_group, imported_group))

# Salida del grafo de dependencias en formato DOT
print('digraph G {')
for from_group, to_group in sorted(group_dependencies):
    print(f'  "{from_group}" -> "{to_group}";')
print('}')
```

Explicación de los Cambios

1. **Argumentos de Línea de Comandos**:
   - **Original**: `python script.py <root_directory>`
   - **Modificado**: `python script.py <root_directory> <level>`
   - Se añadió soporte para un segundo argumento, `level`, que especifica el nivel de la jerarquía de paquetes. El script verifica que se proporcionen exactamente dos argumentos y que `level` sea un entero positivo.
2. **Nueva Función:** `get_package_group`:
   - Se añadió una función para calcular el grupo de paquetes para una clase basada en el `level` especificado.
   - Para un nombre de clase completamente calificado (por ejemplo, “org.springframework.boot.App”), extrae el paquete (“org.springframework.boot”), lo divide en partes (“org”, “springframework” , “boot”) y toma las primeras `level` partes:
     - Si `level=1`: Devuelve “org”.
     - Si `level=2`: Devuelve “org.springframework”.
     - Si el paquete tiene menos partes que `level` (por ejemplo, “com.example”con `level=3`), devuelve el paquete completo (“com.example”).
3. **Agrupación de Dependencias**:
   - **Original**: Usó `defaultdict(set)` para almacenar dependencias entre clases individuales.
   - **Modificado**: Usa un `set` (`group_dependencies`) para almacenar aristas dirigidas entre grupos de paquetes como tuplas `(from_group, to_group)`.

- Para cada clase:
  - Calcula su grupo de paquetes (`importer_group`) usando `get_package_group`.
  - Para cada importación específica que esté dentro del proyecto (`imp in all_classes`) y no sea la clase misma (`imp != full_class_name`):
    * Calcula el grupo de paquetes de la clase importada (`imported_group`).
    * Si los grupos difieren (`imported_group != importer_group`), añade una arista a `group_dependencies`.
- El `set` asegura la unicidad, por lo que múltiples dependencias entre los mismos grupos resultan en una sola arista.

4. **Salida DOT**:
   - **Original**: Imprimió aristas entre clases individuales (por ejemplo, “org.springframework.boot.App” -> “org.apache.commons.IOUtils”).
   - **Modificado**: Imprime aristas entre grupos de paquetes (por ejemplo, “org.springframework”-> “org.apache”para `level=2`).
   - Las aristas se ordenan para una salida consistente.

Cómo Cumple con Sus Requisitos

- **Soporte para Niveles**: El script ahora acepta un parámetro `level` para agrupar paquetes antes de analizar dependencias.
- **Nivel = 1**: Agrupa todas las clases por su paquete de nivel superior (por ejemplo, “org”). Por ejemplo, “org.springframework.boot.App”y “org.apache.commons.IOUtils”ambas pertenecen al grupo “org”, por lo que las importaciones entre ellas dentro de “org”no se muestran como aristas.
- **Nivel = 2**: Agrupa clases por los primeros dos niveles de paquete (por ejemplo, “org.springframework”). Por ejemplo, una importación de “org.springframework.boot.App”a “org.apache.commons.IOUtils” crea una arista de “org.springframework”a “org.apache”.
- **Acumular Paquetes Antes del Análisis de Dependencias**: El script determina el grupo de paquetes de cada clase basado en `level` antes de analizar sus importaciones, asegurando que las dependencias sean entre grupos, no clases individuales.
- **Ejemplo de Cumplimiento**: Para su ejemplo de arista:
  - Original: `"org.springframework.boot.web.servlet.server.Session" -> "org.springframework.boot.convert.D`
  - Con `level=2`: Ambas clases están en “org.springframework”, por lo que no se añade ninguna arista (mismo grupo).
  - Con `level=3`: “org.springframework.boot.web”-> “org.springframework.boot.convert”, añadiendo una arista entre estos grupos distintos.

Ejemplos de Uso

- **Comando**: `python script.py /ruta/a/proyecto 1`

  - Grupos: “org”, “com”, etc.
  - La salida podría incluir: `"org" -> "com"` si una clase en un paquete “org”importa una clase en un paquete “com”.

- **Comando**: `python script.py /ruta/a/proyecto 2`
  - Grupos: “org.springframework”, “org.apache”, etc.
  - La salida podría incluir: `"org.springframework" -> "org.apache"`.

Notas

- El script mantiene el comportamiento original de solo incluir dependencias intraproyecto (importaciones dentro de `all_classes`) y excluir importaciones comodín y auto-dependencias.
- Las clases sin una declaración de paquete se omiten, consistente con el script original.
- La salida es un grafo dirigido en formato DOT, visible con herramientas como Graphviz.

Este script modificado debería satisfacer plenamente su solicitud de agrupar dependencias por nivel de jerarquía de paquetes.