Assignment 3

Team Members

Purvisha Patel

(B00912611)

Subject:

Software Development Concepts

Professor:

Mike McAllister

Overview

In this assignment, we will create a DependencyManager class that will accept data about how different files are interdependent and then document the status of all file types. For developing this, five different methods are used which are listed below.

- addClass
- buildOrder
- standaloneModules
- highUseClasses
- hasDependencyCycle

Files and external data

Java Files

1. DependencyManager.java

Methods

- **boolean addClass(String className)**
  - This method adds classes to the list and throws an exception.
  - Param className accepts className as input if ClassName cannot be represent as a class or if the list of dependents is useless.
  - If the className is stored in the list it will return true.
  - If the className is null, empty or blank, it returns false

- **boolean addClass(String className, Set dependencies)**
  - This function adds the classes and their dependencies to the list.
  - className and dependent classes are the inputs for param className.
  - If the className and dependent className were stored in the list, return true.
  - If the className is null, empty or blank, it returns False.

- **List<String> buildOrder ()**
  - This function obtain all of the class names.
  - Class names are stored in order so it could compile the objects, and when we comes to a class in the sequence, all of the classes that it depends on have already been buit.

- Topological Sort was used to retrieve the sequence of classes.

- **List<String>highUseClasses(int listMax)**
  - This class provides us “listMax” class in descending order of dependencies.
  - Classes that the most other classes directly depend upon.
  - @param listMax takes listMax int as an input.

- **boolean hasDependencyCycle ()**
  - This method is used to identify the cyclic dependency in the graph.
  - It will return true if there exists one or more dependency cycles and return.
  - False if there are no dependency cycles.

- **Set<Set<String>> standaloneModules()**
  - This method is used to identify group of class files that can be compiled into standalone executable
  - @return Return A set of class files can be compiled into a standalone executable program

- **Private Map<String, Integer> sortByValue(Map<String, Integer> unsortMap)**
  - This method is used to sort the map elements
  - UnsortedMap is a parameter which takes the unsorted map elements as input
  - @return It will retrun the sorted map elements

2. standaloneModules.java

- **Public void addEdge(String source, String destination)**
  - This method is responsible for creating the egde between the nodes
  - @param source and destination takes source and destination as input for creating the edge between them.

- **Private Set<String> DFSCheck(String v, HashMap<String, Boolean> visited, Set<String> finalList)**
  - This method search is root by depth first search.
  - It checks weather the node is visited or not.

- **Public Set<Set<String>> graphConnectedComponents()**
    - It checks and connects visitedNodes.
    - If the node is visited by using HashMap we store its key and value.

- **Public boolean isCyclicCheck(String i, HashMap<String, Boolean> visited, HashMap<String, Boolean> recursionStack)**
    - This method checks weather the nodes of the graph is visited and forms cycle or not.
    - It also checks recursion stack and visitedNodes are null or not.

- **Public boolean Cyclic()**

    - It checks whether the graph is dependent on the the other classes or not.
    - It returns true if the cyclic check is done and false if it is not cyclic.

3. Graph.java

- **Public Graph()**
    - It stores the visitedNodes into an ArrayList using HashMap

- **Public void addEdge(String src, String dest)**
    - This method adds Edge of nodes from source to destinaton.

- **Public void topoSort()**
    - This Topological sort is used to create a build order method

Data structures and their relations to each other

Data Structure Used in the Code

1. **LinkedList**
   - This data structure is used to store the stream of words that will be read from the file for the dictionary.

   Advantage of using LinkedList over other data structure

   - The main benefit of using LinkedList is that you can add or remove elements without having to reorganise the entire structure. LinkedList is more efficient than other data structures in terms of inserting and deleting items.
   - Another advantage is that we don't have to define size at the beginning.

Because LinkedList is a dynamic data structure, I used it to store the stream of words. It has the ability to expand and contract in real time. As a result, it does not require an initial size, unlike arrays, which do. Because number of words which we'll read from the file is not fix, so, LinkedList is an excellent choice. There is no memory waste, which is another benefit.

2. **Arraylist**
   - An array is used to implement an ArrayList. When the array fills up, the ArrayList class creates a new array with twice the capacity and copies all of the data over.
   - An ArrayList, or dynamically resizing array, combines the advantages of an array with the size capabilities of a list.

3. **Graph**
   - A graph is a data structure for storing connected data, such as social media platform’s network of people.
   - Vertices and edges make up a graph. The entity (for example, people) is represented by a vertex, and the relationship between entities are represented by an edge.

- Edges and vertices make up each graph (also called nodes). There is a relationship between each vertex and edge. The data is represented by the vertex, and the relationship between them is represented by the edge. A circle with a label on it denotes a vertex. A line connecting nodes is used to represent edges (vertices).

Key algorithms and design elements

- I have implemented the problem using topological sort and DFS.
- The topological sort method takes a directed network and outputs an array of nodes, with each node appearing before all of the nodes to which it refers.
- Depth First Search (DFS) is a graph traversal technique that begins by examining a source node (usually the root node) and then moves on to as many nodes as feasible before returning. Unlike breadth-first search, node exploration is inherently non-uniform.

Limitation/Drawbacks

- There's a limitation In "highUseClasses", when there is tie in counts, the output should be in alphabetical order, which is not done in my code. The classes that the most other classes directly depend upon should be Reported the top "listMax" classes, in decreasing order of dependencies and, when there is a tie in those counts, in alphabetic order of the class name.