Project Description: Object-Oriented Programming with Graphs

1 Objective

The objective of this project is to explore object-oriented programming concepts and design patterns by implementing various types of graphs and related algorithms, while integrating database communication.

2 Project Components

2.1 Graph Classes

Define an abstract class `Graph` that serves as the base class for all graph types. Implement subclasses such as `DirectedGraph`, `WeightedGraph`, and `Tree` that extend the `Graph` class.

Abstract Class: Graph

- `vertices`: A list or set of vertices in the graph (List of index node 1 is represented by int "1").
- `edges`: A table of pairs of vertices representing the connections between vertices.

Undirected Graph (extends Graph)

None (inherits attributes from the `Graph` class).

Directed Graph (Digraph) (extends Graph)

- `directedEdges`: A list of ordered pairs of vertices indicating the direction of the edges.

Weighted Graph (extends Graph)

- `weights`: A dictionary or map containing the weight or cost associated with each edge.

Tree (extends Graph)

- `root`: The topmost node in the tree structure.
- `parent`: A list or map containing parent-child relationships.
- `children`: A list or map containing child-parent relationships.
- `depth`: A dictionary or map representing the depth of each node in the tree.

2.2 Algorithms

- Breadth-First Search (BFS): for all types
- Depth-First Search (DFS): for all types
- Dijkstra's Algorithm: implement this algorithm for undirected, directed, and weighted graphs only.

2.3 Adapter Pattern

- Create an adapter pattern that converts an adjacency list representation of a graph to an adjacency matrix representation, and vice versa.
- The adapter should provide methods for conversion and manipulation of both representations.

2.4 Singleton Pattern

- Implement a Singleton pattern to manage the database connection using Java Database Connectivity (JDBC) and PostgreSQL.
- Ensure that only one instance of the database connection is created throughout the application.
- Add this Maven dependency in `pom.xml` to establish the connectivity:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.7.2</version>
</dependency>
```

2.5 Database Communication

- Integrate database communication to save and retrieve graph data.
- Design database tables to store graph information efficiently.
- Implement an interface named `IGraphManaging` that includes methods for creating, updating, and deleting graphs. Then, create a class `GraphManaging` to represent the database and implement this interface along with its related methods.