```
import java.util.Arrays;

// Class to represent the weighted directed graph
class Graph {
  private int[][] adjacencyMatrix;
  private int numNodes;

  public Graph(int numNodes) {
    this.numNodes = numNodes;
    this.adjacencyMatrix = new int[numNodes][numNodes];

    // Initialize the adjacency matrix with infinity for non-adjacent nodes
    for (int i = 0; i < numNodes; i++) {
      Arrays.fill(adjacencyMatrix[i], Integer.MAX_VALUE);
    }

    // Initialize the diagonal with 0 (distance to itself is 0)
    for (int i = 0; i < numNodes; i++) {
      adjacencyMatrix[i][i] = 0;
    }
  }

  // Function to add an edge with weight to the graph
  public void addEdge(int from, int to, int weight) {
    adjacencyMatrix[from][to] = weight;
  }

  // Floyd-Warshall algorithm to find the shortest paths between all pairs of
  // nodes
  public void floydWarshall() {
    for (int k = 0; k < numNodes; k++) {
      for (int i = 0; i < numNodes; i++) {
        for (int j = 0; j < numNodes; j++) {
          if (adjacencyMatrix[i][k] != Integer.MAX_VALUE &&
              adjacencyMatrix[k][j] != Integer.MAX_VALUE &&
              adjacencyMatrix[i][k] + adjacencyMatrix[k][j] < adjacencyMatrix[i][j]) {
            adjacencyMatrix[i][j] = adjacencyMatrix[i][k] + adjacencyMatrix[k][j];
          }
        }
      }
    }
  }

  // Function to get the shortest path from node 'from' to node 'to'
  public int getShortestPath(int from, int to) {
    return adjacencyMatrix[from][to];
  }
}

public class project_6 {
  public static void main(String[] args) {
    // Create a graph with a given number of nodes
    Graph graph = new Graph(5);
```

```
    // Add edges with weights to the graph
    graph.addEdge(0, 1, 2);
    graph.addEdge(1, 2, 3);
    graph.addEdge(2, 3, 1);
    graph.addEdge(3, 4, 4);
    graph.addEdge(0, 4, 5);

    // Apply Floyd-Warshall algorithm to find shortest paths
    graph.floydWarshall();

    // Get the shortest path between two nodes
    int shortestPath = graph.getShortestPath(0, 4);

    // Print the result
    System.out.println("Shortest path between nodes 0 and 4: " + shortestPath);
  }
}
```