# THE MUSEUM TOUR ORGANIZER

```
import java.util.*;
class Exhibit {
    String name;
    int id;
    public Exhibit(String name, int id) {
        this.name = name;
        this.id = id;
    }

    @Override
    public String toString() {
        return name + " (ID: " + id + ")";
    }
}
```

```
public class MuseumTour {
    private int[][] adjacencyMatrix;
    private List<Exhibit> exhibits;

    public MuseumTour(int numberOfExhibits) {
        adjacencyMatrix = new int[numberOfExhibits][numberOfExhibits];
        exhibits = new ArrayList<>();
        for (int i = 0; i < numberOfExhibits; i++) {
            Arrays.fill(adjacencyMatrix[i], Integer.MAX_VALUE / 2); // Initialize distances as "infinity"
            adjacencyMatrix[i][i] = 0; // Distance to self is 0
        }
    }

    public void addExhibit(String name, int id) {
        exhibits.add(new Exhibit(name, id));
    }

    public void setDistance(int fromId, int toId, int distance) {
        adjacencyMatrix[fromId][toId] = distance;
        adjacencyMatrix[toId][fromId] = distance; // Assuming the graph is undirected
    }
```

```
public void displayExhibits() {
    System.out.println("Exhibits in the Museum:");
    for (Exhibit exhibit : exhibits) {
        System.out.println(exhibit);
    }
}

public void displayAdjacencyMatrix() {
    System.out.println("Adjacency Matrix:");
    for (int[] row : adjacencyMatrix) {
        System.out.println(Arrays.toString(row));
    }
}

private int[] dijkstra(int startId) {
    int n = exhibits.size();
    int[] distances = new int[n];
    boolean[] visited = new boolean[n];
    Arrays.fill(distances, Integer.MAX_VALUE / 2);
    distances[startId] = 0;

    PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
    pq.offer(new int[]{startId, 0});

    while (!pq.isEmpty()) {
```

```
            int[] current = pq.poll();
            int currentNode = current[0];
            int currentDistance = current[1];

            if (visited[currentNode]) continue;
            visited[currentNode] = true;

            for (int neighbor = 0; neighbor < n; neighbor++) {
                if (adjacencyMatrix[currentNode][neighbor] < Integer.MAX_VALUE / 2) { // Valid edge
                    int newDist = currentDistance + adjacencyMatrix[currentNode][neighbor];
                    if (newDist < distances[neighbor]) {
                        distances[neighbor] = newDist;
                        pq.offer(new int[]{neighbor, newDist});
                    }
                }
            }
        }
        return distances;
    }
```

```
public void findShortestTour(int startId, List<Integer> desiredIds) {
    // Step 1: Precompute all-pairs shortest paths using Dijkstra
    int n = exhibits.size();
    int[][] allPairsDistances = new int[n][n];
    for (int i = 0; i < n; i++) {
        allPairsDistances[i] = dijkstra(i);
    }

    // Step 2: Use a greedy algorithm to approximate the shortest tour
    boolean[] visited = new boolean[n];
    List<String> tourSequence = new ArrayList<>();
    int current = startId;
    int totalDistance = 0;

    tourSequence.add(exhibits.get(current).name);
    visited[current] = true;

    while (!desiredIds.isEmpty()) {
        int nearest = -1;
        int minDistance = Integer.MAX_VALUE;

        // Find the nearest unvisited exhibit
        for (int id : desiredIds) {
            if (!visited[id] && allPairsDistances[current][id] < minDistance) {
                nearest = id;
```

```
                minDistance = allPairsDistances[current][id];
            }
        }

        // Move to the nearest exhibit
        if (nearest != -1) {
            totalDistance += minDistance;
            current = nearest;
            visited[current] = true;
            tourSequence.add(exhibits.get(current).name);
            desiredIds.remove((Integer) current);
        } else {
            break; // No reachable exhibits left
        }
    }

    // Step 3: Return to the starting point
    totalDistance += allPairsDistances[current][startId];
    tourSequence.add(exhibits.get(startId).name);

    // Output the results
    System.out.println("Shortest Tour Sequence: " + tourSequence);
    System.out.println("Total Tour Time: " + totalDistance + " minutes");
}
```

```
//main method
    public static void main(String[] args) {
        // Example: Create a museum layout
        MuseumTour museum = new MuseumTour(5);

        museum.addExhibit("Monet's Water Lilies", 0);
        museum.addExhibit("Michelangelo's Pieta", 1);
        museum.addExhibit("Egyptian Sarcophagi", 2);
        museum.addExhibit("Picasso's Guernica", 3);
        museum.addExhibit("Van Gogh's Sunflowers", 4);

        museum.setDistance(0, 1, 5);
        museum.setDistance(0, 2, 10);
        museum.setDistance(1, 3, 15);
        museum.setDistance(2, 4, 20);
        museum.setDistance(3, 4, 25);

        museum.displayExhibits();
        museum.displayAdjacencyMatrix();

        // Find shortest path for a tour starting at exhibit 0 and visiting exhibits 1, 3, and 4
        museum.findShortestTour(0, new ArrayList<>(Arrays.asList(1, 3, 4)));
    }
}
```