TASK 8.1

```
import java.util.*;
class FlightFinder {
    static class Flight {
        int city, cost, stops;
        Flight(int city, int cost, int stops) {
            this.city = city;
            this.cost = cost;
            this.stops = stops;
        }
    }
    public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {
        Map<Integer, List<int[]>> graph = new HashMap<>();
        for (int[] flight : flights) {
            graph.computeIfAbsent(flight[0], x -> new ArrayList<>()).add(new int[]{flight[1],
flight[2]});
        }
        PriorityQueue<Flight> pq = new PriorityQueue<>(Comparator.comparingInt(f ->
f.cost));
        pq.offer(new Flight(src, 0, 0));
        Map<Integer, Integer> visited = new HashMap<>();
        while (!pq.isEmpty()) {
            Flight current = pq.poll();
            if (current.city == dst) {
                return current.cost;
            }
            if (visited.containsKey(current.city) && visited.get(current.city) <= current.stops) {
                continue;
            }
            visited.put(current.city, current.stops);
```

```
            if (current.stops <= k) {
                List<int[]> neighbors = graph.getOrDefault(current.city, new ArrayList<>());
                for (int[] neighbor : neighbors) {
                    int nextCity = neighbor[0];
                    int price = neighbor[1];
                    pq.offer(new Flight(nextCity, current.cost + price, current.stops + 1));
                }
            }
        }
        return -1;
    }
    public static void main(String[] args) {
        FlightFinder solution = new FlightFinder();
        int n = 4;
        int[][] flights = {
            {0, 1, 100},
            {1, 2, 100},
            {2, 3, 100},
            {0, 3, 500}
        };
        int src = 0, dst = 3, k = 1;
        int result = solution.findCheapestPrice(n, flights, src, dst, k);
        System.out.println("Cheapest price: " + result);
    }
}
```

TASK 8.2 - Minimum cost to connect

Program:

```
import java.util.Arrays;
public class MinCostConnectTwoGroups {
    public int connectTwoGroups(int[][] cost) {
        int size1 = cost.length;
        int size2 = cost[0].length;
        int[] minCostToSecondGroup = new int[size2];
        Arrays.fill(minCostToSecondGroup, Integer.MAX_VALUE);
        for (int i = 0; i < size1; i++) {
            for (int j = 0; j < size2; j++) {
                minCostToSecondGroup[j] = Math.min(minCostToSecondGroup[j], cost[i][j]);
            }
        }
        int[][] dp = new int[size1 + 1][1 << size2];
        for (int[] row : dp) {
            Arrays.fill(row, Integer.MAX_VALUE);
        }
        dp[0][0] = 0; // Starting with no points connected
        for (int i = 0; i < size1; i++) {
            for (int mask = 0; mask < (1 << size2); mask++) {
                if (dp[i][mask] == Integer.MAX_VALUE) continue;
                for (int j = 0; j < size2; j++) {
                    int newMask = mask | (1 << j);
                    dp[i + 1][newMask] = Math.min(
                        dp[i + 1][newMask],
                        dp[i][mask] + cost[i][j]
                    );
```

```
            }
        }
    }
    int result = Integer.MAX_VALUE;
    for (int mask = 0; mask < (1 << size2); mask++) {
        if (dp[size1][mask] == Integer.MAX_VALUE) continue;
        int totalCost = dp[size1][mask];
        for (int j = 0; j < size2; j++) {
            if ((mask & (1 << j)) == 0) { // Point j in group 2 is not connected
                totalCost += minCostToSecondGroup[j];
            }
        }
        result = Math.min(result, totalCost);
    }
    return result;
}
public static void main(String[] args) {
    MinCostConnectTwoGroups solver = new MinCostConnectTwoGroups();
    int[][] cost = {
        {15, 96},
        {36, 2}
    };
    int result = solver.connectTwoGroups(cost);
    System.out.println("Minimum cost to connect two groups: " + result);
}
}
```

Task 8.3 Decode string

```
import java.util.Scanner;
import java.util.Stack;
public class DecodeString {
    public String decodeString(String s) {
        Stack<Integer> countStack = new Stack<>();
        Stack<StringBuilder> stringStack = new Stack<>();
        StringBuilder currentString = new StringBuilder();
        int k = 0;
        for (char ch : s.toCharArray()) {
            if (Character.isDigit(ch)) {
                k = k * 10 + (ch - '0');
            } else if (ch == '[') {
                countStack.push(k);
                stringStack.push(currentString);
                currentString = new StringBuilder(); // Reset for a new substring
                k = 0;
            } else if (ch == ']') {
                int repeatCount = countStack.pop();
                StringBuilder decodedString = stringStack.pop();
                for (int i = 0; i < repeatCount; i++) {
                    decodedString.append(currentString);
                }
                currentString = decodedString;
            } else {
                currentString.append(ch);
            }
```

```
        }
        return currentString.toString();
    }

    // Main method for user input
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        DecodeString decoder = new DecodeString();
        System.out.print("Enter the encoded string: ");
        String encodedInput = scanner.nextLine();
        String result = decoder.decodeString(encodedInput);
        System.out.println("Decoded string: " + result);
        scanner.close();
    }
}
```