1. Dynamic Programming: Minimum Coin Change Problem
A vending machine needs to return the minimum number of coins as change. The machine should be optimized to return the least number of coins for a given amount.

Task:
Given a list of coin denominations and an amount, implement a function using DP to find the minimum number of coins required to make the amount.

CODE:

```
import java.util.Arrays;
public class CoinChange {
    public static int minCoins(int[] coins, int amount) {
        int max = amount + 1;
        int[] dp = new int[amount + 1];
        Arrays.fill(dp, max);
        dp[0] = 0;
        for (int i = 1; i <= amount; i++) {
            for (int coin : coins) {
                if (i >= coin) {
                    dp[i] = Math.min(dp[i], dp[i - coin] + 1);
                }
            }
        }
        return dp[amount] == max ? -1 : dp[amount];
    }
    public static void main(String[] args) {
        int[] coins = {1, 2, 5};
        int amount = 11;
        System.out.println(minCoins(coins, amount));
    }
}
```

OUTPUT:

3

2. You are building a weather monitoring system that records temperature readings. You need to find the K-th closest reading to a target temperature for accurate prediction adjustments.

Task:
Given a list of sensor readings and a target value, write a program to find the K-th closest element using the Quickselect algorithm.

CODE:

```
import java.util.Arrays;
public class KthClosestElement {
    public static int absDiff(int a, int b) {
        return Math.abs(a - b);
    }
    public static void sortReadings(int[] readings, int[] diffs, int n) {
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (diffs[j] > diffs[j + 1]) {
                    int temp = diffs[j];
                    diffs[j] = diffs[j + 1];
                    diffs[j + 1] = temp;
                    temp = readings[j];
                    readings[j] = readings[j + 1];
                    readings[j + 1] = temp;
                }
            }
        }
    }
    public static int findKthClosest(int[] readings, int target, int k) {
        int n = readings.length;
        int[] diffs = new int[n];
        for (int i = 0; i < n; i++) {
            diffs[i] = absDiff(readings[i], target);
        }
```

```
        sortReadings(readings, diffs, n);
        return readings[k - 1];
    }
    public static void main(String[] args) {
        int[] readings = {72, 75, 68, 80, 74};
        int target = 73, k = 2;
        System.out.println(findKthClosest(readings, target, k));
    }
}
```

OUTPUT:

74