12.1 Problem: Best Time to Buy and Sell Stock with Transaction Fee
Code:

```
public class StockProfitWithFee {
   public static int maxProfit(int[] prices, int fee) {
      if (prices == null || prices.length < 2) {
         return 0;
      }
      int n = prices.length;
      int[] hold = new int[n]; // Max profit if holding a stock
      int[] cash = new int[n]; // Max profit if not holding a stock
      hold[0] = -prices[0]; // Buy the first stock
      cash[0] = 0; // No profit without transactions
      for (int i = 1; i < n; i++) {
         hold[i] = Math.max(hold[i - 1], cash[i - 1] - prices[i]);
         cash[i] = Math.max(cash[i - 1], hold[i - 1] + prices[i] - fee);
      }
      return cash[n - 1];
   }
   public static void main(String[] args) {
      int[] prices = {1, 3, 2, 8, 4, 9};
      int fee = 2;
      System.out.println("Maximum Profit: " + maxProfit(prices, fee));
   }
}
```

12.2 Problem: Minimum Number of Taps to Open to Water a Garden

Code:

```
import java.util.Arrays;
public class MinimumTapsToWaterGarden {
   public static int minTaps(int n, int[] ranges) {
      int[] maxReach = new int[n + 1];
      for (int i = 0; i <= n; i++) {
         int left = Math.max(0, i - ranges[i]);
         int right = Math.min(n, i + ranges[i]);
         maxReach[left] = Math.max(maxReach[left], right);
      }
      int taps = 0, end = 0, farthest = 0;
      for (int i = 0; i <= n; i++) {
         if (i > farthest) {
            return -1; // Cannot water the whole garden
         }
         if (i > end) {
            taps++;
            end = farthest;
         }
         farthest = Math.max(farthest, maxReach[i]);
      }
      return taps;
   }
   public static void main(String[] args) {
      System.out.println("Minimum Taps: " + minTaps(n, ranges));
   }
}
```

12.3 Problem: Water Bottles

Code

```
public class WaterBottlesExchange
{
 public static int numWaterBottles(int numBottles, int numExchange) {
      int totalDrunk = 0;
      int emptyBottles = 0;
      while (numBottles > 0) {
         totalDrunk += numBottles; // Drink all full bottles
         emptyBottles += numBottles; // Add to empty bottles
         numBottles = emptyBottles / numExchange; // Exchange empty bottles for new full ones
         emptyBottles %= numExchange; // Remaining empty bottles after exchange
      }
      return totalDrunk;
   }
   public static void main(String[] args) {
      System.out.println("Total Bottles Drunk: " + numWaterBottles(numBottles, numExchange));
   }
}
```