CODE 1

```
#!/usr/bin/env python
# coding: utf-8

# Input from user for number of queens
N = int(input("Enter the number of queens: "))
print(f"Entered number of queens: {N}\n")

# Chessboard initialization (NxN matrix with all elements set to 0)
board = [[0] * N for _ in range(N)]

# Function to check if a position (i, j) is under attack by any other queen
def is_attack(i, j):
    # Check if there is a queen in the same row or column
    for k in range(N):
        if board[i][k] == 1 or board[k][j] == 1:
            return True

    # Check diagonals
    for k in range(N):
        for l in range(N):
            if (k + l == i + j) or (k - l == i - j):  # Checking if in diagonal
                if board[k][l] == 1:
                    return True
    return False

# Recursive function to solve the N-Queens problem
def N_queen(n):
    # If n is 0, all queens are placed, return True (solution found)
    if n == 0:
        return True

    # Try placing a queen in every position on the board
    for i in range(N):
        for j in range(N):
            # Check if we can place a queen here
            if not is_attack(i, j) and board[i][j] != 1:
                board[i][j] = 1  # Place the queen

                # Recursively try to place the remaining queens
                if N_queen(n - 1):
                    return True  # If a valid arrangement is found, return True
```

```
            # If placing the queen here does not lead to a solution, backtrack
            board[i][j] = 0

    return False

# Solve the N-Queens problem
if N_queen(N):
    # Output the solution
    print(f"Solution for {N}-Queens Problem:")
    for row in board:
        print(" ".join(str(x) for x in row))
else:
    print(f"No solution exists for {N}-Queens problem.")
```

CODE 2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Bank {
    // State variable to store the balance of the customer
    mapping(address => uint256) private balances;

    // Event to log deposits
    event Deposit(address indexed account, uint256 amount);

    // Event to log withdrawals
    event Withdrawal(address indexed account, uint256 amount);

    // Function to deposit money into the bank account
    function deposit() public payable {
        require(msg.value > 0, "Deposit amount must be greater than zero");

        // Increase the balance of the sender
        balances[msg.sender] += msg.value;

        // Emit the deposit event
        emit Deposit(msg.sender, msg.value);
    }

    // Function to withdraw money from the bank account
    function withdraw(uint256 amount) public {
        require(amount > 0, "Withdrawal amount must be greater than zero");
        require(balances[msg.sender] >= amount, "Insufficient balance");
```

```
        // Decrease the balance of the sender
        balances[msg.sender] -= amount;

        // Transfer the amount to the sender
        payable(msg.sender).transfer(amount);

        // Emit the withdrawal event
        emit Withdrawal(msg.sender, amount);
    }

    // Function to show the balance of the sender's account
    function showBalance() public view returns (uint256) {
        return balances[msg.sender];
    }
}
```