Decentralized Voting (Lab)

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.25;

// This contract is a simple Decentralized Voting System
// It allows people to vote for proposals in a transparent and trustless manner

contract DecentralizedVoting {
    // Struct to represent a proposal
    struct Proposal {
        string name;         // Proposal name
        uint voteCount;      // Number of votes received
    }

    // Owner of the contract
    address public owner;

    // Mapping to track if an address has voted
    mapping(address => bool) public hasVoted;

    // Array of proposals
    Proposal[] public proposals;

    // Constructor to initialize the proposals
    // When the contract is deployed, the deployer (owner) sets up the proposals
    constructor(string[] memory proposalNames) {
        owner = msg.sender; // Set the contract deployer as the owner
        for (uint i = 0; i < proposalNames.length; i++) {
            // Create a new proposal for each name provided
            proposals.push(Proposal({
                name: proposalNames[i],
                voteCount: 0
            }));
        }
    }

    // Function to vote for a proposal
    // Anyone can call this function to vote for a proposal by providing its index
    function vote(uint proposalIndex) public {
        // Ensure the sender has not already voted
        require(!hasVoted[msg.sender], "You have already voted.");
        // Ensure the proposal index is valid
        require(proposalIndex < proposals.length, "Invalid proposal index.");

        // Mark the sender as having voted
        hasVoted[msg.sender] = true;

        // Increment the vote count of the selected proposal
        proposals[proposalIndex].voteCount++;
    }

    // Function to get all proposals
    // Returns an array of all the proposals with their names and vote counts
    function getProposals() public view returns (Proposal[] memory) {
        return proposals;
    }

    // Function to get the winning proposal
    // Finds the proposal with the most votes and returns its name and vote count
    function getWinner() public view returns (string memory winnerName, uint winnerVoteCount) {
        uint winningVoteCount = 0;
        uint winningIndex = 0;
        for (uint i = 0; i < proposals.length; i++) {
            if (proposals[i].voteCount > winningVoteCount) {
                winningVoteCount = proposals[i].voteCount;
                winningIndex = i;
            }
        }
        return (proposals[winningIndex].name, proposals[winningIndex].voteCount);
    }
}
```

Testing the Contract

Once the contract is deployed, let's test its functionality step-by-step.

1. **View Proposals**:
   - Scroll down to the deployed contract instance.
   - Click on **`getProposals`**.
   - This will return an array of the proposals with their respective names and current vote counts, which should all be `0` at the beginning.
2. **Vote for a Proposal**:
   - Switch the **Account** in Remix to simulate different users voting.
   - Enter the **index** of the proposal you want to vote for (e.g., `0` for "Proposal A").
   - Click on **`vote`**.
   - You will need to change accounts and repeat this to cast additional votes.
3. **Check Voting Restrictions**:
   - If you try to vote again with the same account, you should see an error saying `"You have already voted."`.
   - This demonstrates the effectiveness of the `hasVoted` mapping in preventing multiple votes.
4. **Check Voting Results**:
   - Use **`getProposals`** again to see which proposals received votes.
   - The corresponding `voteCount` should have increased for the proposals voted for.
5. **Get the Winner**:
   - Click **`getWinner`** to see which proposal currently has the most votes.
   - The output will include the `winnerName` and `winnerVoteCount`.