Bài 3

|  |
| --- |
| pragma solidity ^0.8.0;  contract VotingSystem {  struct Candidate {  string name;  uint256 voteCount;  }  mapping(address => bool) private voters;  Candidate[] private candidates;  event CandidateRegistered(string name);  event VoteCasted(address indexed voter, uint256 candidateIndex);  event VotingResult(string winner, uint256 voteCount);  function registerCandidate(string memory name) public {  require(bytes(name).length > 0, "Invalid candidate name");  candidates.push(Candidate(name, 0));  emit CandidateRegistered(name);  }  function castVote(uint256 candidateIndex) public {  require(candidateIndex < candidates.length, "Invalid candidate index");  require(!voters[msg.sender], "Already voted");  candidates[candidateIndex].voteCount++;  voters[msg.sender] = true;  emit VoteCasted(msg.sender, candidateIndex);  }  function getVotingResult() public view returns (string memory, uint256) {  require(candidates.length > 0, "No candidates registered");  uint256 maxVoteCount = 0;  string memory winner;  for (uint256 i = 0; i < candidates.length; i++) {  if (candidates[i].voteCount > maxVoteCount) {  maxVoteCount = candidates[i].voteCount;  winner = candidates[i].name;  }  }  return (winner, maxVoteCount);  }  function getCandidateCount() public view returns (uint256) {  return candidates.length;  }  function getCandidate(uint256 index) public view returns (string memory, uint256) {  require(index < candidates.length, "Invalid candidate index");  Candidate memory candidate = candidates[index];  return (candidate.name, candidate.voteCount);  }  } |

Bài 4

|  |
| --- |
| pragma solidity ^0.8.0;  contract InvestmentFund {  struct Investor {  uint256 balance;  uint256 lastProfitDistribution;  }  mapping(address => Investor) private investors;  uint256 private totalBalance;  uint256 private lastProfitDistribution;  event Deposit(address indexed investor, uint256 amount);  event Withdraw(address indexed investor, uint256 amount);  event ProfitDistributed(uint256 amount);  function deposit() public payable {  require(msg.value > 0, "Invalid amount");  investors[msg.sender].balance += msg.value;  totalBalance += msg.value;  emit Deposit(msg.sender, msg.value);  }  function withdraw(uint256 amount) public {  require(amount > 0, "Invalid amount");  require(amount <= investors[msg.sender].balance, "Insufficient balance");  investors[msg.sender].balance -= amount;  totalBalance -= amount;  payable(msg.sender).transfer(amount);  emit Withdraw(msg.sender, amount);  }  function getTotalBalance() public view returns (uint256) {  return totalBalance;  }  function distributeProfit(uint256 amount) public {  require(amount > 0, "Invalid amount");  require(amount <= address(this).balance, "Insufficient contract balance");  require(block.timestamp > lastProfitDistribution + 1 days, "Cannot distribute profit yet");  uint256 totalInvestedBalance = totalBalance;  uint256 sharePerBalance = amount / totalInvestedBalance;  for (address investorAddress : getAllInvestors()) {  uint256 investorBalance = investors[investorAddress].balance;  uint256 profit = investorBalance \* sharePerBalance;  investors[investorAddress].balance += profit;  }  totalBalance += amount;  lastProfitDistribution = block.timestamp;  emit ProfitDistributed(amount);  }  function getInvestorBalance(address investorAddress) public view returns (uint256) {  return investors[investorAddress].balance;  }  function getLastProfitDistribution() public view returns (uint256) {  return lastProfitDistribution;  }  function getAllInvestors() public view returns (address[] memory) {  address[] memory investorAddresses = new address[](totalInvestors());  uint256 index = 0;  for (address investorAddress : investors) {  investorAddresses[index] = investorAddress;  index++;  }  return investorAddresses;  }  function totalInvestors() public view returns (uint256) {  return investors.length;  }  } |

Bài 6

|  |
| --- |
| pragma solidity ^0.8.0;  contract InsurancePolicy {  struct Policy {  uint256 premium;  uint256 coverage;  bool claimed;  bool approved;  }  mapping(address => Policy) private policies;  uint256 private totalPremiums;  uint256 private totalClaims;  event PolicyPurchased(address indexed policyHolder, uint256 premium, uint256 coverage);  event ClaimRequested(address indexed policyHolder, uint256 amount);  event ClaimApproved(address indexed policyHolder, uint256 amount);  event ClaimRejected(address indexed policyHolder);  function purchasePolicy() public payable {  require(policies[msg.sender].premium == 0, "Policy already purchased");  uint256 premium = msg.value;  uint256 coverage = premium \* 2;  Policy memory policy = Policy(premium, coverage, false, false);  policies[msg.sender] = policy;  totalPremiums += premium;  emit PolicyPurchased(msg.sender, premium, coverage);  }  function requestClaim() public {  require(policies[msg.sender].premium > 0, "No policy purchased");  require(!policies[msg.sender].claimed, "Claim already requested");  policies[msg.sender].claimed = true;  emit ClaimRequested(msg.sender, policies[msg.sender].coverage);  }  function approveClaim(address policyHolder) public {  require(policies[policyHolder].claimed, "No claim requested");  require(!policies[policyHolder].approved, "Claim already approved");  policies[policyHolder].approved = true;  uint256 claimAmount = policies[policyHolder].coverage;  totalClaims += claimAmount;  payable(policyHolder).transfer(claimAmount);  emit ClaimApproved(policyHolder, claimAmount);  }  function rejectClaim(address policyHolder) public {  require(policies[policyHolder].claimed, "No claim requested");  require(!policies[policyHolder].approved, "Claim already approved");  policies[policyHolder].claimed = false;  emit ClaimRejected(policyHolder);  }  function getTotalPremiums() public view returns (uint256) {  return totalPremiums;  }  function getTotalClaims() public view returns (uint256) {  return totalClaims;  }  function getPolicy(address policyHolder) public view returns (uint256, uint256, bool, bool) {  Policy memory policy = policies[policyHolder];  return (policy.premium, policy.coverage, policy.claimed, policy.approved);  }  } |