**BT: Practical 3 (bank)**

pragma solidity ^0.5.8;

contract SimpleBank {

uint8 private clientCount;

mapping (address => uint) private balances;

address public owner;

// Log the event about a deposit being made by an address and its amount

event LogDepositMade(address indexed accountAddress, uint amount);

// Constructor is "payable" so it can receive the initial funding of 30,

// required to reward the first 3 clients

constructor() public payable {

require(msg.value == 30 ether, "30 ether initial funding required");

/\* Set the owner to the creator of this contract \*/

owner = msg.sender;

clientCount = 0;

}

/// @notice Enroll a customer with the bank,

/// giving the first 3 of them 10 ether as reward

/// @return The balance of the user after enrolling

function enroll() public returns (uint) {

if (clientCount < 3) {

clientCount++;

balances[msg.sender] = 10 ether;

}

return balances[msg.sender];

}

/// @notice Deposit ether into bank, requires method is "payable"

/// @return The balance of the user after the deposit is made

function deposit() public payable returns (uint) {

balances[msg.sender] += msg.value;

emit LogDepositMade(msg.sender, msg.value);

return balances[msg.sender];

}

/// @notice Withdraw ether from bank

/// @return The balance remaining for the user

function withdraw(uint withdrawAmount) public returns (uint remainingBal) {

// Check enough balance available, otherwise just return balance

if (withdrawAmount <= balances[msg.sender]) {

balances[msg.sender] -= withdrawAmount;

msg.sender.transfer(withdrawAmount);

}

return balances[msg.sender];

}

/// @notice Just reads balance of the account requesting, so "constant"

/// @return The balance of the user

function balance() public view returns (uint) {

return balances[msg.sender];

}

/// @return The balance of the Simple Bank contract

function depositsBalance() public view returns (uint) {

return address(this).balance;

}

}

**BT Practical 3 (Bank)**

pragma solidity ^0.4.24;

contract Bank{

int balance;

constructor() public {

balance=0;

}

function withdraw(int amount) public{

if (balance>amount)

{

balance = balance - amount;

}

}

function bal() public view returns(int) {

return balance;

}

function despo(int amount) public{

balance = balance+amount;

}

}

**BR Practical 4 (Student dataset)**

pragma solidity ^0.4.24;

contract StudentRegister{

/\*\*

\* @dev owner is a state variable

\*/

address public owner;

/\*\*

\* @dev mapping address as key to struct student with mapping name students

\*/

mapping (address=>student)students;

/\*\*

\* @dev assigning the contract deployer as the owner

\*/

constructor() public {

owner=msg.sender;

}

/\*\*

\* @dev a modifier onlyOwner is created to limit the access to function register to contract deployer

\*/

modifier onlyOwner {

require(msg.sender==owner);

\_;

}

/\*\*

\* @dev a struct student is defined

\*/

struct student{

address studentId;

string name;

string course;

uint256 mark1;

uint256 mark2;

uint256 mark3;

uint256 totalMarks;

uint256 percentage;

bool isExist;

}

/\*\*

\* @notice function to register studentid,name,course and marks

\* @param studentId is student's ethereum address

\* @param name student's name

\* @param course student's course

\* @param mark1 student's mark's

\*/

function register(address studentId,string memory name,string memory course,uint256 mark1,uint256 mark2,uint256 mark3) public onlyOwner {

/\*\*

\*@dev require statment to block multiple entry

\*/

require(students[studentId].isExist==false,"ha.. ha... Fraud Not Possible,student details already registered and cannot be altered");

uint256 totalMarks;

uint256 percentage;

/\*\*

\* @dev calculating totalMarks and percentage

\*/

totalMarks=(mark1+mark2+mark3);

percentage=(totalMarks/3);

/\*\*

\* @dev assigning the student details to a key (studentId)

\*/

students[studentId]=student(studentId,name,course,mark1,mark2,mark3,totalMarks,percentage,true);

}

/\*\*

\* @notice function to get the details of a student when studentId is given

\*/

function getStudentDetails(address studentId) public view returns (address,string memory,string memory,uint256,uint256){

/\*\*

\* @dev returning studentId,name,course,totalMarks and percentage of student to corresponding key

\*/

return(students[studentId].studentId,students[studentId].name,students[studentId].course,students[studentId].totalMarks,students[studentId].percentage);

}

}