BT-1

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.0 <0.9.0;
contract SimpleBank {
    struct client_account{
        int client_id;
        address client_address;
        uint client_balance_in_ether;
    }
    client_account[] clients;
    int clientCounter;
    address payable manager;
    modifier onlyManager() {
         require(msg.sender == manager, "Only manager can call this!");
        _;
    }
    modifier onlyClients() {
        bool isclient = false;
        for(uint i=0;i<clients.length;i++){
            if(clients[i].client_address == msg.sender){
                isclient = true;
                break;
            }
        }
        require(isclient, "Only clients can call this!");
        _;
    }
    constructor() {
        clientCounter = 0;
}
receive() external payable { }


function setManager(address managerAddress) public returns(string memory){
   manager = payable(managerAddress);
   return "";
}
function joinAsClient() public payable returns(string memory){
   clients.push(client_account(clientCounter++, msg.sender, address(msg.sender).balance));
   return "";
}
function deposit() public payable onlyClients{
   payable(address(this)).transfer(msg.value);
}
function withdraw(uint amount) public payable onlyClients{
   payable(msg.sender).transfer(amount * 1 ether);
}
function sendInterest() public payable onlyManager{
   for(uint i=0;i<clients.length;i++){
      address initialAddress = clients[i].client_address;
      payable(initialAddress).transfer(1 ether);
   }
}
function getContractBalance() public view returns(uint){
   return address(this).balance;
}
```

BT-2

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract Crud {
   struct User {
      uint id;
      string name;
   }
   User[] public users;
   uint public nextId = 0;
   function Create(string memory name) public {
      users.push(User(nextId, name));
      nextId++;
   }
   function Read(uint id) view public returns(uint, string memory) {
      for(uint i=0; i<users.length; i++) {
         if(users[i].id == id) {
            return(users[i].id, users[i].name);
         }
      }
   }

   function Update(uint id, string memory name) public {
      for(uint i=0; i<users.length; i++) {
         if(users[i].id == id) {
            users[i].name =name;
         }
      }
```

```
    }


    function Delete(uint id) public {
        delete users[id];
    }
 function find(uint id) view internal returns(uint) {
        for(uint i=0; i< users.length; i++) {
            if(users[i].id == id) {
                return i;
            }
        }
        // if user does not exist then revert back
        revert("User does not exist");
    }
}
```