Homework 4 Nadine Chancay

Solidity

To this contract

1. Add a variable to hold the address of the deployer of the contract
2. Update that variable with the deployer's address when the contract is deployed.
3. Write an external function to return
4. Address 0x000000000000000000000000000000000000dEaD if called by the deployer
5. The deployer's address otherwise

Bootcamp.sol

```
// SPDX-License-Identifier: None

pragma solidity 0.8.17;


contract BootcampContract {

    uint256 number;
    address deployer;
    constructor(){
       deployer = msg.sender;
    }


    function store(uint256 num) public {
        number = num;
    }


    function retrieve() external view returns (uint256){
        return number;
    }

    function getAddressDeployer() public view returns (address){
       if(msg.sender == deployer){
        return 0x000000000000000000000000000000000000dEaD;
       }
       else{
           return deployer;
       }
     }
}
```

DogCoin Contract:

1. In Remix, create a new file called DogCoin.sol .
2. Define the pragma compiler version to 0.8.18 .
3. Before the pragma version, add a license identifier
   // SPDX-License-Identifier: UNLICENSED .
4. Create a contract called DogCoin.
5. Create a variable to hold the total supply, with an initial amount of 2 million.
6. Make a public function that returns the total supply.
7. Make a public function that can increase the total supply in steps of 1000.
8. Declare an address variable called owner . this address will be allowed to change the total supply
9. Next, create a modifier which only allows an owner to execute certain functions.
10. Make your change total supply function public , but add your modifier so that only the owner can execute it.
11. Create a constructor to initialise the state of the contract and within the constructor, store the owner's address.
12. Create an event that emits the new value whenever the total supply changes. When the supply changes, emit this event.
13. In order to keep track of user balances, we need to associate a user's address with the balance that they have.
    a) What is the best data structure to hold this association ?
    b) Using your choice of data structure, set up a variable called balances to keep track of the number of tokens that a user has.
14. We want to allow the balances variable to be read from the contract, there are 2 ways to do this.
    What are those ways ?
    Use one of the ways to make your balances variable visible to users of the contract.
15. Now change the constructor, to give all of the total supply to the owner of the contract.
16. Now add a public function called transfer to allow a user to transfer their tokens to another address. This function should have 2 parameters :

Why do we not need the sender's address here ?
What would be the implication of having the sender's address as a parameter ?

17. Add an event to the transfer function to indicate that a transfer has taken place, it should log the amount and the recipient address.
18. We want to keep a record for each user's transfers. Create a struct called Payment that stores the transfer amount and the recipient's address.
19. We want to have a payments array for each user sending the payment. Create a mapping which returns an array of Payment structs when given this user's address.

DogCoin.sol

```
//SPDX-License-Identifier: UNLICENSED .
pragma solidity >=0.8.18;

contract DogCoin {

    //variables
    uint256 totalSupply;
    address owner;

    //modifiers
    modifier onlyOwner{
    if (msg.sender==owner){
        _;
    }
    }

    //constructors

    constructor() {
    owner = msg.sender;
    totalSupply= 2000000;
    balance[msg.sender] = totalSupply;
}

    //structs
    struct Payment {
        address recipientAddress;
        uint256 transferAmount;
    }

    //mappings
    mapping (address => uint256) balance;
    mapping(address => Payment[]) public payments;

    //events
    event SupplyChanges(uint256);
    event TransferNotification(uint256, address);



    //functions
    function increaseTotalSupply() public {
        totalSupply += 1000;
        emit SupplyChanges(totalSupply);
    }

    function getTotalSupply() public view returns (uint256){
        return totalSupply;
    }

    function transfer(address destination, uint256 amount) public {
        balance[destination] += amount;
        balance[msg.sender] -= amount;
        payments[msg.sender].push(Payment({recipientAddress: destination, transferAmount: amount}));
        emit TransferNotification(amount, destination);
    }

    function getPayments(address _address) public view returns (Payment[] memory) {
        return payments[_address];
    }

}
```