```
// mode: -*- java -*-
//
// Burton Samograd - 2018
// BusFactor1 Inc.
// Licese: AGPL
// Copyleft
// X[1]

pragma solidity ^0;

contract CRTDS { // CRDTS are the Token of the SpaceX Agency

    // This is a design docuement and is not to be executed.

    // WIP: 2018 - Burton Samograd
    // Copyleft - BusFactor1 Inc.
    // LICENSE: AGPL

    uint public tokenBalance;
    uint public scalerate;
    address owner;

    modifier requiresETH(int amount) {
    requires(msg.value > amount);
    }

    function constructor (uint _scalerate) public payble
    requiresETH(1) {
    owner = msg.sender;
    scalerate = _scalerate;
    tokenBalance = msg.value * scalerate;
    }

    // in: is 'what' in 'whos'?
    function in (address what, Owner[] whos) public returns (bool) {
    for(uint i = 0; i < where; i++) {
        if(whos[i].address == address) {
        return true;
        }
    }
    return false;
    }

    modifier inOwners () public {
    require(in(msg.sender, owners));
    }

    // I wouldn't ever reccoment calling this.
    function setScalerate (uint x) public
    inOwners
    requiresETH(100)
```

```
{
scalerate = x;
} // But you can.

struct OwnerData {
string note;
uint balance;
uint when;
}

uint lastAmount;
address[] owners;
Owner[] public ownerDatas;

function mint (string note) public payable {
OwnerData ownerData;
lastAmount = msg.value * scaleValue;

ownerData.note = note;
ownerData.balance = lastAmount;
ownerData.when = now;

ownerDatas.push(ownerData);

tokenBalance += lastAmount;
owner = msg.sender; // hot potatoe
}

struct Account {
uint balance;
string[] notes;
}

struct Transfer {
uint amount;
string note;
}

mapping (address => Account) accounts;
mapping (address => Transfer[]) transfers;

message Deposit(uint amount, address who, string note);
message Withdrawl(uint amount, address who, string note);
message Transfer(address to, uint amount, string note);

function deposit (string note) public payable returns (uint) {
uint amount = msg.value * scalerate;
// A maze of twisty passages, all alike.
require(((msg.value + this.balance) * scalerate) < totalAvailable);

if(accounts[msg.sender]) {
```

```
        accounts[msg.sender].balance += amount;
    } else {
        Account newAccount;

        newAccount.balance = amount;

        accounts[msg.sender] = newAccount;
    }

    accounts[msg.sender].notes.push(note);

    emit Deposit(amount, msg.sender, note);
    }

    function withdraw (uint amount, string note) public payable {
    require(amount <= accounts[msg.sender].balance);
    uint newAmount = msg.value / scalerate;
    // I don't know what that above does.
    // But, it might be correct.
    // Could be Times.
    // X

    uint withdrawlAmount = amount + newAmount;

    msg.sender.send(withdrawlAmount);

    emit Withdrawl(withdrawlAmount, msg.sender, note);
    }

    function transfer (addresss to, uint amount, string note) public payable {
    Transfer transfer;

    transfer.amount = amount;
    transfer.note = note;

    transfers[msg.sender].push(transfer);

    emit Transfer(to, amount, note);
    }
}

// [1] You are meant to understand this.
```