pragma solidity >=0.4.22 <0.6.0;

contract PayThem {

uint EtherRecived = 0; // count of ether in the contractor

mapping (address => uint) paidAmount; //like a dictionary to see if they have been paid

address [4] employees;

constructor () public payable{ //constructor

updatingEtherReceived();

}

function () external payable{ // fall back function.

updatingEtherReceived() ;

}

function updatingEtherReceived() internal {

EtherRecived += msg.value; // in the wallet for the contract

}

modifier canTakeOut() { // check to see if they are an employee

bool employed = false ;

for (uint i=0; i< employees.length; i++){

if (employees[i] == msg.sender){

employed=true;

}

}

require (employed); //modifier syntax

\_; //modifier syntax

}

function salary() canTakeOut() public { // we have now added the modifier

//to ensure that they are an employee

uint payGiven = EtherRecived/employees.length; // the amount of money per employee

uint payTaken = paidAmount [msg.sender]; // the pay (0 or amount) already given to // an employee

uint pay = payGiven-payTaken; // the amount paid this time

if (pay >0){ // only if there is something to pay send it to them

msg.sender.transfer(pay);

}

paidAmount[msg.sender] =payTaken + pay ; // the total amount they have been paid, //update for next time

}

}