pragma solidity >=0.7.0 <0.8.0;

contract BlockBank {

 receive() external payable { }

 fallback() external payable { }

 event DepositLog (address depositor, uint amountDeposited);

 event WithdrawLog (address withdrawer, uint amountWithdrew);

 event BorrowLog (address Borrower, uint amountBorrowed);

 mapping (address => bool) private existingAcct;

 mapping (address => uint) private depositBalance;

 mapping (address => uint) private loanBalance;

 address public bankOwner;

 address [10] public customerList;

 uint public accountCount;

 constructor() public payable {

 require(msg.value >= 2 ether, "At least 2 ether for initial funding of bank required");

 bankOwner = msg.sender;

 accountCount = 0;

 }

 //Use this function to open an account with the bank before user can deposit, withdraw, borrow or transfer

 function openAccount() public returns (address) {

 require(accountCount < 10, "Maximum of 10 accounts reached, unable to open new account");

 if(existingAcct[msg.sender] == false) {

 existingAcct[msg.sender] = true;

 customerList[accountCount] = msg.sender;

 accountCount++;

 }

 }

 //Use this function to deposit amount to the bank

 //Fund will move from customer address to bank contract address, bank to update deposit balance record

 //Message value must match with deposit amount declared, otherwise error message will be prompted

 function deposit(uint depositAmount) public payable returns (uint) {

 require(existingAcct[msg.sender] == true, "Please open an account first");

 require(msg.value == depositAmount, "Message value does not equal to deposit amount declared. Abort.");

 depositBalance[msg.sender] += depositAmount;

 emit DepositLog (msg.sender, depositAmount);

 return depositBalance[msg.sender];

 }

 //Use this function to withdraw amount from the bank

 //Fund will move from bank contract address to customer address, bank to update deposit balance record

 //Amount to be withdrew must be larger than deposit balance, otherwise error message will be prompted

 //Amount to be withdrew must be less than bank's remaining balance, otherwise error message will be prompted

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

 require(existingAcct[msg.sender] == true, "Please open an account first");

 require(withdrawAmount <= depositBalance[msg.sender], "Withdrawal amount cannot be more than deposit balance. Abort.");

 require(withdrawAmount <= getRemainingBankBalance());

 depositBalance[msg.sender] -= withdrawAmount;

 msg.sender.transfer(withdrawAmount);

 emit WithdrawLog (msg.sender, withdrawAmount);

 return depositBalance[msg.sender];

 }

 //Use this function to borrow from the bank

 //Fund will move from bank contract address to customer address, bank to update loan balance record

 //Bank must have at least 1 eth after lending out the declared amount, otherwise error message will be prompted

 //Loan balance recorded included a 5% interest

 //Loan amount declared must be at least 100 wei, and any amount declared will be rounded down to the nearest 100 wei

 function borrow(uint loanAmount) public returns (uint) {

 require(existingAcct[msg.sender] == true, "Please open an account first");

 require(getRemainingBankBalance() - loanAmount >= 1 ether, "Bank has insufficient balance, unable to lend");

 require(loanAmount > 100, "Loan amount declared must be at least 100 wei");

 loanAmount -= loanAmount % 100;

 loanBalance[msg.sender] += loanAmount \* 105/100;

 msg.sender.transfer(loanAmount);

 emit BorrowLog (msg.sender, loanAmount);

 return loanBalance[msg.sender];

 }

 //Use this function to repay borrowed money back to the bank

 //Fund will move from customer address to bank contract address, bank to update loan balance record

 //Repayment amount must be less than or equal to outstanding loan balance, otherwise error message will be prompted

 //Message value must match with repayment amount declared, otherwise error message will be prompted

 function repay(uint repayAmount) public payable returns (uint) {

 require(existingAcct[msg.sender] == true, "Please open an account first");

 require(repayAmount <= loanBalance[msg.sender], "Repayment amount is more than loan balance. Abort.");

 require(msg.value == repayAmount, "Message value does not equal to repayment amount declared. Abort.");

 loanBalance[msg.sender] -= repayAmount;

 return loanBalance[msg.sender];

 }

 //Use this function to get the current deposit balance of a customer

 function getDepositBalance() public view returns (uint) {

 return depositBalance[msg.sender];

 }

 //Use this function to get the current loan balance of a customer

 function getLoanBalance() public view returns (uint) {

 return loanBalance[msg.sender];

 }

 //Use this function to get the net balance of a customer

 //Net balance equals to deposit balance minus loan balance

 function balance() public view returns (int) {

 return int(depositBalance[msg.sender]) - int(loanBalance[msg.sender]);

 }

 //Use this function to get the total balance of the bank

 function getRemainingBankBalance() public view returns (uint) {

 return address(this).balance;

 }

 //Use this function to check if a customer is registered as a customer of the bank

 function verifiedAsCustomer() public view returns (bool) {

 return existingAcct[msg.sender];

 }

 //Use this function to check how many accounts has a net zero or positive balance

 //Bank can only be closed if there is no net negative balance account

 function PositiveBalanceCount() public returns (uint) {

 uint n=0;

 for (uint i=0; i<accountCount; i++) {

 if(depositBalance[customerList[i]] >= loanBalance[customerList[i]]) {

 n+=1;

 }

 }

 return n;

 }

 //Use this function to close the bank

 //Only bank owner can close the bank, otherwise error message will be prompted

 //Bank can only be closed if there is no outstanding negative balance account, otherwise error message will be prompted

 //All deposit balance net of loan balance will be returned to each customer

 //Any remaining fund after distribution of balances to customers will be returned to bank owner

 function closeBank() public {

 require(msg.sender == bankOwner, "Only bank owner can close the bank. Abort.");

 require(PositiveBalanceCount() == accountCount, "Outstanding negative balance(s) exist, unable to close bank");

 for (uint j=0; j<accountCount; j++) {

 address payable recipientCustomer = address(uint160(customerList[j]));

 recipientCustomer.transfer(depositBalance[customerList[j]] - loanBalance[customerList[j]]);

 depositBalance[customerList[j]] = 0;

 loanBalance[customerList[j]] = 0;

 }

 address payable recipientOwner = address(uint160(bankOwner));

 recipientOwner.transfer(address(this).balance);

 }

 //Use this function to check if the declared transferee is a customer of the recipient bank

 function checkTransfereeOnList (address checkAddress) public returns (bool onList) {

 return existingAcct[checkAddress];

 }

 //Use this function to update the deposit balance of the transferee in the recipient bank after the fund is successful transferred

 function updateFundReceived (address transfereeAddress, uint transferAmount) public {

 depositBalance[transfereeAddress] += transferAmount;

 }

 //Use this function to transfer fund to a transferee of another recipient bank

 //Transferee must have an account at the recipient bank, otherwise error message will be prompted

 //Transferor must have sufficient deposit balance in order for the declared amount of fund to be transferred, otherwise error message will be prompted

 //Amount to be transferred must be less than bank's remaining balance, otherwise error message will be prompted

 //Deposit balance records of the transferor in remitting bank and the transferee in recipient bank will be updated

 function fundTransfer(address payable toBankAddress, address transfereeAddress, uint transferAmount) public returns (bool transferSuccess) {

 BlockBank ReceivingBank;

 ReceivingBank = BlockBank(toBankAddress);

 require(transferAmount <= depositBalance[msg.sender], "Insufficient deposit balance to make transfer. Abort.");

 require(ReceivingBank.checkTransfereeOnList(transfereeAddress) == true, "Transferee does not have an account at the bank");

 require(transferAmount <= getRemainingBankBalance());

 toBankAddress.transfer(transferAmount);

 depositBalance[msg.sender] -= transferAmount;

 ReceivingBank.updateFundReceived(transfereeAddress, transferAmount);

 transferSuccess = true;

 }

}