# Multi-tenant decentralized loan contract for Blockchain Global Master M9 End Project

**This project is on my github**

## Summary

- Lenders can deposit any amount above 1 ether and earn annual 10% interest.
- Borrower can borrow any amount at a time with annual 20% interest.

## Details

DecentralizedLoanContract.sol

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0 <0.9.0;

/**
 * @title Money Lender
 * @dev Multi-tenant decentralized loan contract. This is a experimental
contract, so we don't collect any transaction fee.
 * Lenders can deposit any amount above 1 ether and earn annual 10%
interest.
 * Borrower can borrow any amount at a time with annual 20% interest.
 */
contract DecentralizedLoanContract {
    // Total avaialble amount of this contract
    uint public totalFund = 0;

    // borrower's annual laon interest(20%) converted to daily basis
    //  Don't forget to divide by 100000 at calculation using this
variable.
    uint8 public BORROWER_DAILY_INTEREST = 55; // 20% / 365 = 0.0547% =
0.055%(round up) = 0.00055 = 55 / 100000.
    // lender's annual deposit interest(10%) converted to daily basis
    // Don't forget to divide by 100000 at calculation using this
variable.
    uint8 public LENDER_DAILY_INTEREST = 27; // 10% / 365 = 0.0273% =
0.027%(round down) = 0.00027 = 27 / 100000.

    // NOTE: Minimum deposit length is 30 days
    uint public MIN_DEPOSIT_LENGTH = 30 days;
    // NOTE: Minimum deposit amount is 30 days
    uint public MIN_DEPOSIT_AMOUNT = 1 ether;

    // NOTE: Maximum one-time loan amount for each borrower is 1 ether
    uint public MAX_LOAN_AMOUNT = 1 ether;
    // NOTE: Minimum loan length is 1 day
    uint32 public MIN_LENDING_LENGTH = 1 days;

    uint32 public A_DAY_IN_SEC = 86400; // 1 day = 86400 sec (=60 sec * 60
sec * 24 hour)

    struct Borrower {
        uint amount;
        uint time;
    }
    mapping (address => Borrower) public borrowers;

    struct Lender {
        uint deposit;
        uint time;
    }
    mapping (address => Lender) public lenders;

    /// Invalid repayment amount. Needed `required` but sent `amount`
    /// @param sent sent amount.
    /// @param required amount to send.
    error InvalidAmount (uint256 sent, uint256 required);


    /**
     * @dev deposit function for lenders
     */
    function deposit() public payable {
        require(msg.value >= MIN_DEPOSIT_AMOUNT, "1 ether is minimal
deposit amout.");
        totalFund += msg.value;
        lenders[msg.sender] = Lender({
            deposit: msg.value,
            time: block.timestamp
        });
    }

    /**
     * @dev withdraw function for lenders to withdraw all her deposit with
earned interest
     */
    function withdraw() public payable {
        uint amountWithInterest = calculateDepositAmountWithInterest();
        require(totalFund > amountWithInterest, "Not enough fund is
available at the moment. Please try later.");

        totalFund -= amountWithInterest;
        lenders[msg.sender].deposit = 0;
        payable(msg.sender).transfer(amountWithInterest);
    }

    /**
     * @dev calculate total return. A user can call this to know his total
amount including deposit interest.
     */
    function calculateDepositAmountWithInterest() public view returns
(uint) {
        uint amount = lenders[msg.sender].deposit;
        require(amount > 0, "Your deposit was not found!");

        uint timeElapsedInSec = block.timestamp -
lenders[msg.sender].time;
        require(timeElapsedInSec >= MIN_DEPOSIT_LENGTH, "At least 30 days
have to pass since you deposited money.");

        uint16 elapsedDays = uint16(timeElapsedInSec / A_DAY_IN_SEC);
        uint interest = (amount * elapsedDays * LENDER_DAILY_INTEREST) /
100000; // calculate earned interest based on deposit amount and length
        uint amountWithInterest = amount + interest;
        return amountWithInterest;
    }

    /**
     * @dev borrow function for borrower. A borrower cannot borrow more
than one time before settling existing loan.
     * @param amount to borrow
     */
    function borrow(uint amount) public payable {
        require(amount <= MAX_LOAN_AMOUNT, "Max loan amount for a customer
is 1 ether");
        require(amount <= totalFund, "Not enough found is available");

        uint existingAmount = borrowers[msg.sender].amount;
        require(existingAmount == 0, "You have to settle your existing
loan first.");

        borrowers[msg.sender] = Borrower({
            amount: amount,
            time: block.timestamp
        });
        payable(msg.sender).transfer(amount);
    }

    /**
     * @dev replayment function to pay back all amount she borrowed
     */
    function repay() public payable {
        uint amountWithInterest = calculateLoanAmountWithInterest();
        if (msg.value < amountWithInterest) {
            revert InvalidAmount({
               sent: msg.value,
               required: amountWithInterest
            });
        }

        borrowers[msg.sender].amount = 0;
        totalFund += msg.value;
    }

    /**
     * @dev calculate total repayment amount. A user can call this to know
his total repayment amount.
```

```
     */
    function calculateLoanAmountWithInterest() public view returns (uint)
{
        uint amount = borrowers[msg.sender].amount;
        require(amount > 0, "You have not borrowed any money.");

        uint timeElapsedInSec = block.timestamp -
lenders[msg.sender].time;
        timeElapsedInSec = MIN_LENDING_LENGTH * 1;
        require(timeElapsedInSec >= MIN_LENDING_LENGTH, "Minimum lending
time is 1 day");

        uint16 elapsedDays = uint16(timeElapsedInSec / A_DAY_IN_SEC);
        uint interest = (amount * elapsedDays * BORROWER_DAILY_INTEREST) /
100000; // calculate loan interest based on loan amount and length
        uint amountWithInterest = amount + interest;
        return amountWithInterest;
    }
}
```