A SIMPLE BANK

Implemented through a smart contract

[Presentation Plan]

- Intro to smart contracts
- Development Environment
- Describe functions of a simple bank:
  - Smart contract class
  - Keep track of deposits
  - Allow users to make deposits
  - Ensure deposits are legitimate
  - Generate interest through lending (Compound)
  - Allow users to make withdrawals
- Demo
- Challenges
- Possible Expansions
- Key takeaways

WHAT IS A SMART CONTRACT?

EXTERNALLY OWNED ACCOUNTS (EOA)

Users with public and private keys; can initiate transactions on the Ethereum blockchain

SMART CONTRACT

Like users, contracts have an address (public key), and can hold currency
However, it cannot initiate its own transactions, and transacts based on **coded logic**

DEVELOPMENT ENVIRONMENT

REMIX IDE

Open-Source Ethereum IDE used to compile solidity to execute smart contract

SOLIDITY

Solidity - Programming language designed for ETH

Playground for learning how to use the Ethereum network

DEVELOPMENT ENVIRONMENT

METAMASK

Cryptocurrency wallet for the Blockchain

Allows users to own their keys instead of relying on exchanges

Safe from hackers

DEVELOPMENT ENVIRONMENT

ROPSTEN TESTNET

Ethereum test network that allows testing before deployment on Mainnet

Uses rETH instead of real ETH

Available since November 2016

DEVELOPMENT ENVIRONMENT

METAMASK

Cryptocurrency wallet for the Blockchain. Allows users to own their keys instead of relying on exchanges

REMIX IDE

Integrated development environment for developing, deploying and administering smart contracts on the Ethereum blockchain

ROPSTEN TESTNET

Testing blockchain for developers; uses rETH instead of real ETH

IMPLEMENTING BANK FUNCTIONALITY

A simple bank has to be able to…

1. Have its own reserves
2. Keep track of deposits
3. Allow users to make deposits
4. Ensure deposits are legitimate (no double spending)
5. Allow users to make withdrawals
6. Generate interest through lending

Smart Contract

```
//a smart contract is easy to implement in Solidity. Like classes
in other languages, we can code functions specific to it.

contract SmartBankAccount {




}
```

1. Bank's own reserves

```
//smart contracts are accounts, and can store value. We have to
implement a variable within the contract to record it.

contract SmartBankAccount {

    uint totalContractBalance = 0;

    function addMoneyToContract() public payable {

        totalContractBalance += msg.value;

    }

}
```

2. Keep track of deposits

```
//implementing a “mapping” of user addresses to integer values,
or bank balances

    mapping(address ⇒ uint) balances;
```

3. Allow users to make deposits

```
//implementing a function to add money to the contract and record how much a user's balance has changed

    mapping(address ⇒ uint) balances;

    function addBalance() public payable {

        balances[msg.sender] += msg.value;

    }
```

A function that can be accessed by users (hence "public"), and that enables payment acceptable ("payable")

3. Allow users to make deposits

```
//implementing a function to add money to the contract and record how much a user's balance has changed

    mapping(address ⇒ uint) balances;

    function addBalance() public payable {

        balances[msg.sender] += msg.value;

    }
```

A function that can be accessed by users (hence "public"), and that enables payment acceptable ("payable")

obtain user's address through msg.sender, update our records by how much they sent (msg.value)

4. Ensure Deposits are Legit

//this is accomplished by the Blockchain.

//As discussed earlier in the semester, miners (or in the case of Ethereum, validators) ensure every new transaction is legitimate

- **Users** - who hold accounts and make/receive transactions.
- **Nodes** - who keep an updated copy of the ledger.
- **Miners** - who verify transactions on the blockchain.

//the downsides of this are:

1. There are high transaction costs for writing directly on the Blockchain ($3.236 USD/tx)
2. Longer transaction times, or fewer transactions/second

5. Allow users to withdraw

Key Steps in Withdraw

Step 1 : Check request address and requested return amount

Step 2 : Currency transfer from bank to user

Step 3 : Update bank’s balance

{Full Withdrawal}

{Partial Withdrawal}

5. Allow users to withdraw

```
function FullWithdraw () public payable {

    address payable withdrawTo = payable(msg.sender);

    uint amount = getBalance(msg.sender);

    withdrawTo.transfer(amount);

    totalConBalance = totalConBalance - amount;

    balances[msg.sender] = 0;

    }
```

Step 1: Declaring address and requested amount

Step 2: Coin Transferring

Step 3: Updating **Total balance** and **users' personal balance**

5. Allow users to withdraw

```
function PartialWithdraw (uint AmountWant) public payable {
    address payable withdrawTo = payable(msg.sender);
    uint amount = AmountWant;
    withdrawTo.transfer(amount);
    totalConBalance = totalConBalance - amount;
    balances[msg.sender] = 0;
    }
```

Step 1: Getting request amount from input

Step 2: Coin Transferring

Step 3: Updating **Total balance** and users' personal balance

6. Generate Interest by Lending

```
function getBalance(address user) public view returns(uint) {
    uint principal = balances[user];
    uint timeElapsed = block.timestamp - depositTimestamps[user];
            //seconds
    return principal + uint((principal * 7 * timeElapsed)
    / (100 * 365 * 24 * 60 * 60)) + 1;
            //simple interest of 7%  per year
    }
```

6. Generate Interest by Lending

Generate Interest internally or externally?

```
function addMoneyToContract() public payable {

        totalConBalance += msg.value;

        // Initialize the bank with some coins so that it can
            return some interest together with the withdrawal

    }
```

6. Generate Interest by Lending

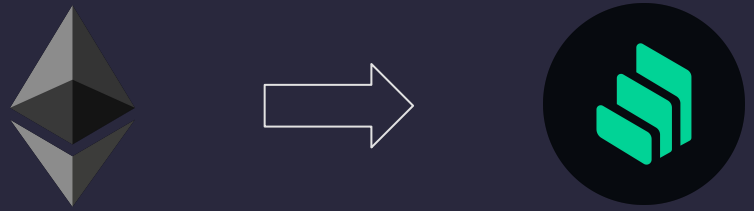
Generate Interest internally or externally?

//there are already smart contracts that function as networks for users to buy and sell from one another

//we “plug in” to one of these into our contract, known as **Compound**

we’ll collect ethers from our users and park it on compound. Then when they want to withdraw, we’ll take the interest from Compound and give it to the user.

6. Generate Interest by Lending



We pay ether (ETH)
and in return get
cETH at the current
exchange rate of
1 CETH = 0.0200 ETH

6. Generate Interest by Lending

We pay ether (ETH) and in return get cETH at the current exchange rate of 1 CETH = 0.0200 ETH

Some time later, we redeem cETH for ETH at a new exchange rate, e.g. 1 CETH = 0.0201 ETH

Finally, let’s demo

Pain Points

- **Getting the environment set-up**
  a. Getting enough test currency to work with
  b. Integrating with Compound contract
- **Getting used to Addresses**
  a. When a contract calls the function of another contract, it emits its own address
  b. Coding the correct address inputs (contract vs user) can be tricky

Pain Points

- **Coding without Decimals:** As Solidity doesn't support decimal places, everything is done in terms of wei. For context, 1 ETH is 10^18 wei. That's a lot of zeroes
- **Testing with Compound:**
  a. Each function call (transaction) takes 15-30s, sometimes more
  b. Documentation assumes we know *everything* about Solidity

Possible New Features

Multi-currency Wallet

Transfer between Accounts

Other Investment Vehicles

Key Takeaways

Accessible.

Smart contracts managing billions of dollars are easy to read and understand. Writing to ensure there are no loopholes and errors, however, is tough.

Value, codified.

Our smart contract essentially carries out the “middleman” work of a traditional bank, with lines of logic. Rather than costs of operations, we have transaction costs from computation.

/THANKS!

/DO YOU HAVE ANY QUESTIONS?

Catherine Li, Jeff Hu,
Jun Bin Ho, Raushan Khullar