Lab 04:
Launching an ICO

Nick Zoghb

LAB OUTLINE

1. THE ERC20 STANDARD
2. BUILDING ON ERC20
3. EXTRA TOOLS
4. MIDTERM: ICO LAUNCH

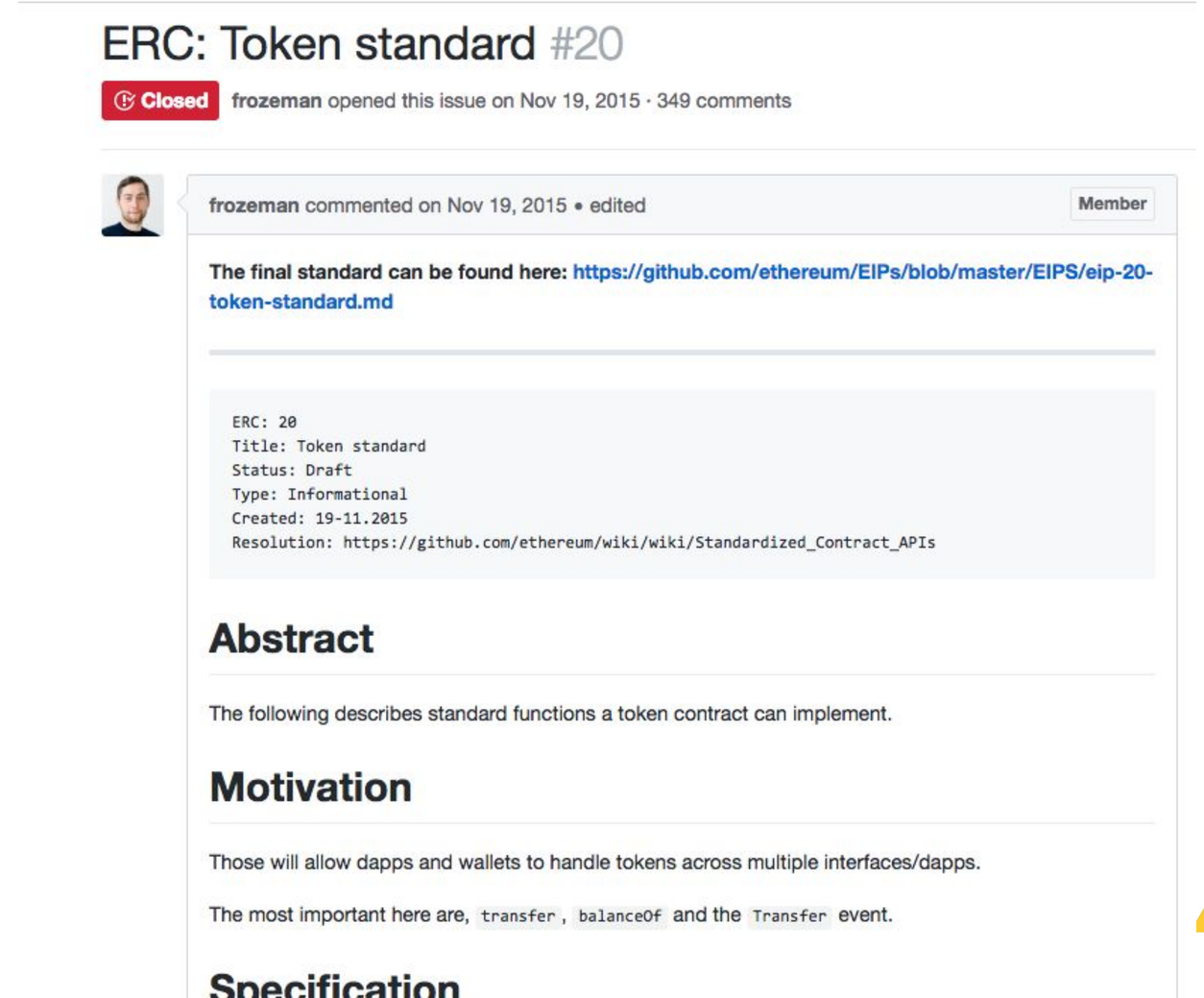
1 THE ERC20 STANDARD

ETHEREUM REQUEST FOR COMMENTS

ISSUE #20

- Comes under Ethereum Improvement Proposal
- Standardize token transfers
- Big problem: at least $400,000 lost

Original ERC20 issue, implementation example



ERC: Token standard #20

Closed frozeman opened this issue on Nov 19, 2015 · 349 comments

frozeman commented on Nov 19, 2015 • edited Member

The final standard can be found here: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md

```
ERC: 20
Title: Token standard
Status: Draft
Type: Informational
Created: 19-11.2015
Resolution: https://github.com/ethereum/wiki/wiki/Standardized_Contract_APIs
```

Abstract

The following describes standard functions a token contract can implement.

Motivation

Those will allow dapps and wallets to handle tokens across multiple interfaces/dapps.

The most important here are, `transfer`, `balanceOf` and the `Transfer` event.

Specification

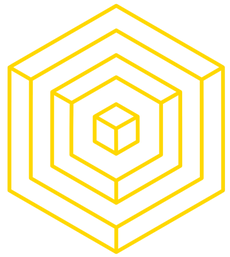
AUTHOR: NICK ZOGHB

ERC20 INTERFACE

WHY DIDN'T THEY THINK OF THIS EARLIER

The ERC-20 defines a common list of rules for all Ethereum tokens to follow, meaning that this particular token empowers developers of all types to accurately predict how new tokens will function within the larger Ethereum system.

```
contract ERC20Interface {
   function totalSupply() constant returns (uint256 totalSupply);
   function balanceOf(address _owner) constant returns (uint256 balance);
   function transfer(address _to, uint256 _value) returns (bool success);
   function transferFrom(address _from, address _to, uint256 _value) returns (bool success);
   function approve(address _spender, uint256 _value) returns (bool success);
   function allowance(address _owner, address _spender) constant returns (uint256 remaining);
   event Transfer(address indexed _from, address indexed _to, uint256 _value);
   event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```



```
contract ERC20Interface {
    // Get the total token supply
    function totalSupply() constant returns (uint256 totalSupply);

    // Get the account balance of another account with address _owner
    function balanceOf(address _owner) constant returns (uint256 balance);

    // Send _value amount of tokens to address _to
    function transfer(address _to, uint256 _value) returns (bool success);

    // Send _value amount of tokens from address _from to address _to
    function transferFrom(address _from, address _to, uint256 _value) returns (bool success);

    // Allow _spender to withdraw from your account, multiple times, up to the _value amount.
    // If this function is called again it overwrites the current allowance with _value.
    // this function is required for some DEX functionality
    function approve(address _spender, uint256 _value) returns (bool success);

    // Returns the amount which _spender is still allowed to withdraw from _owner
    function allowance(address _owner, address _spender) constant returns (uint256 remaining);

    // Triggered when tokens are transferred.
    event Transfer(address indexed _from, address indexed _to, uint256 _value);

    // Triggered whenever approve(address _spender, uint256 _value) is called.
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
}
```

ERC20 EXTRA FRAGMENTS

NECESSARY ADDITIONS

```
contract TokenContractFragment {
  // Constructor
  function TokenContractFragment(_uint amount) {
    // critical code to execute on deployment
  }
  // Balances for each account
  mapping(address => uint256) balances;
  // Owner of account approves the transfer of an amount to another account
  mapping(address => mapping (address => uint256)) allowed;
}
```

APPROVE EXAMPLE

A QUICK RUNTHROUGH

If 0x11111 (balance = 100) wants to authorize 0x22222 (balance = 200) to transfer some tokens to 0x22222, 0x11111 will execute the function:

```
tokenContract.approve(0x22222, 30)
```

The approve data structure will now contain the following information:

```
tokenContract.allowed[0x11111][0x22222] = 30
```

If 0x22222 wants to later transfer some tokens from 0x11111 to itself, 0x22222 executes the transferFrom(...) function:

```
tokenContract.transferFrom(0x11111, 20)
```

The balances data structure will be altered to contain the following information:

```
tokenContract.balances[0x11111] = 70
tokenContract.balances[0x22222] = 220
```

(Source)

APPROVE EXAMPLE

A QUICK RUNTHROUGH

And the approve data structure will now contain the following information:

```
tokenContract.allowed[0x11111][0x22222] = 10
```

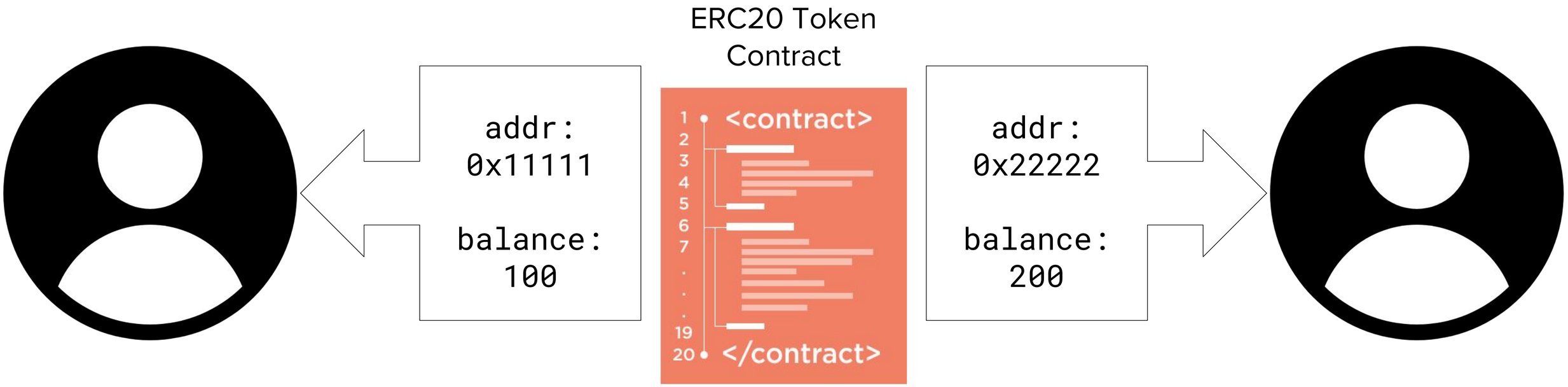
0x22222 can still spend 10 tokens from 0x11111.

The `balanceOf(...)` function will now return the following values:

```
tokenContract.balanceOf(0x11111) // will return 70
tokenContract.balanceOf(0x22222) // will return 230
```

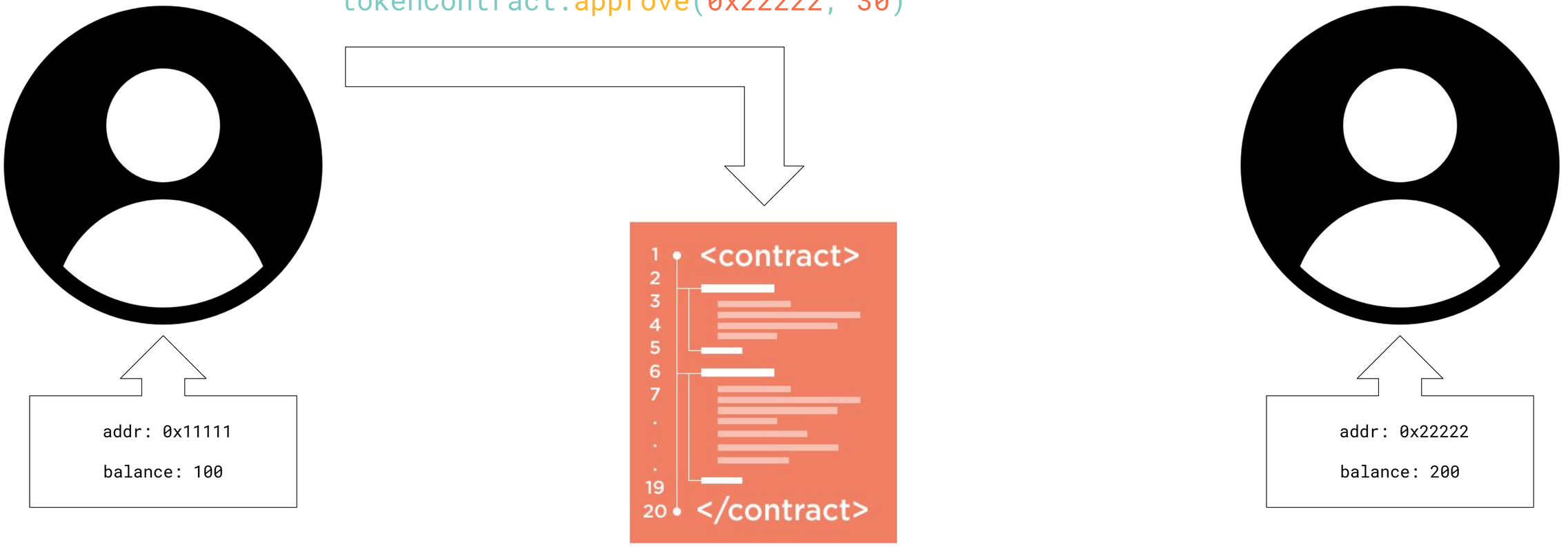
APPROVE EXAMPLE

A QUICK RUNTHROUGH



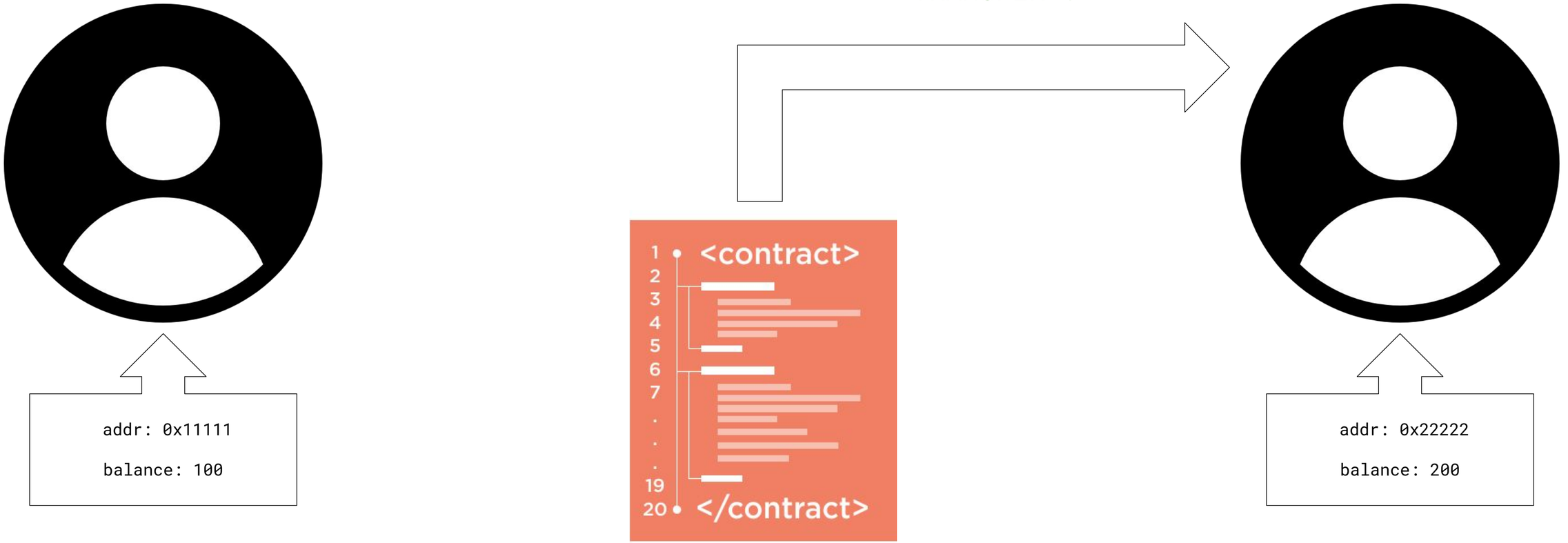
APPROVE EXAMPLE

A QUICK RUNTHROUGH



APPROVE EXAMPLE

A QUICK RUNTHROUGH

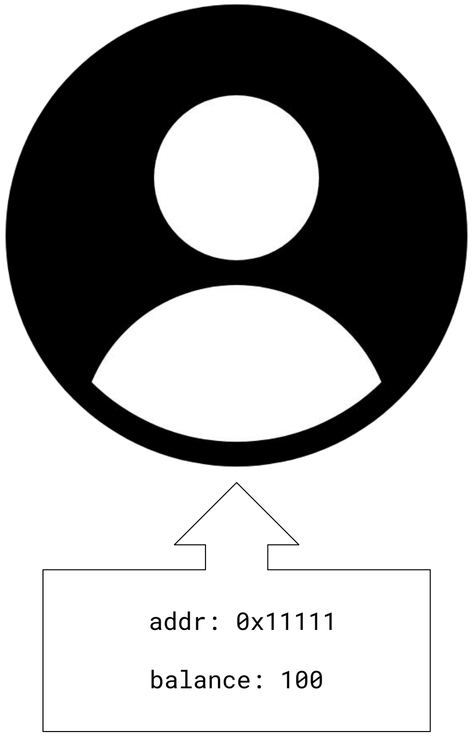


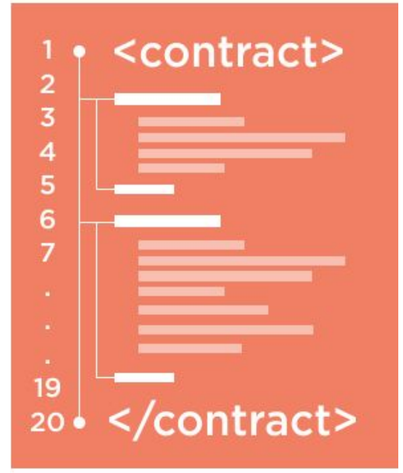
APPROVE EXAMPLE

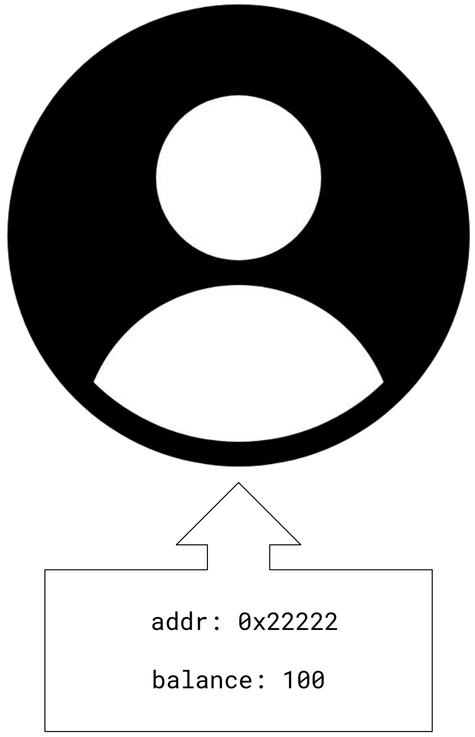
A QUICK RUNTHROUGH

30

tokenContract.allowed[0x11111][0x22222]







APPROVE EXAMPLE

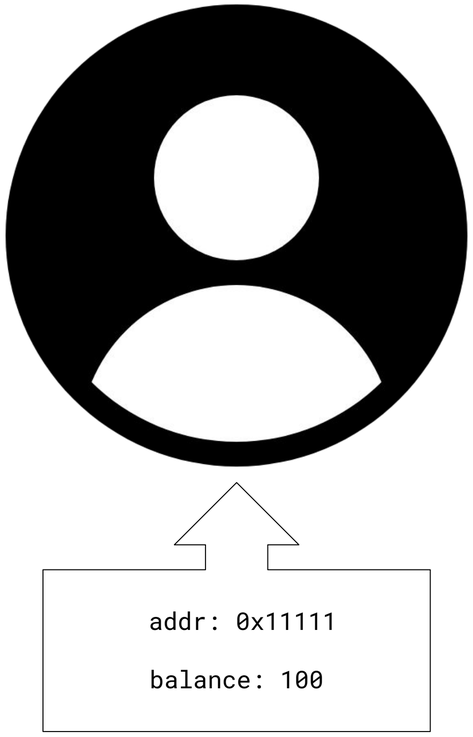
A QUICK RUNTHROUGH

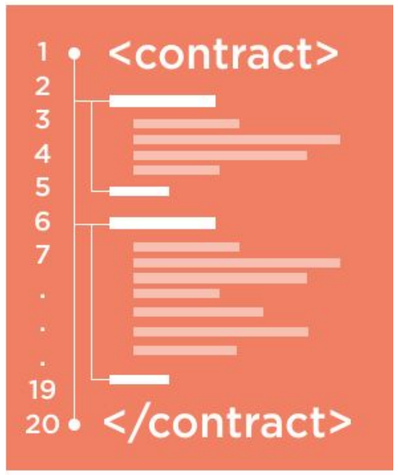
APPROVE EXAMPLE

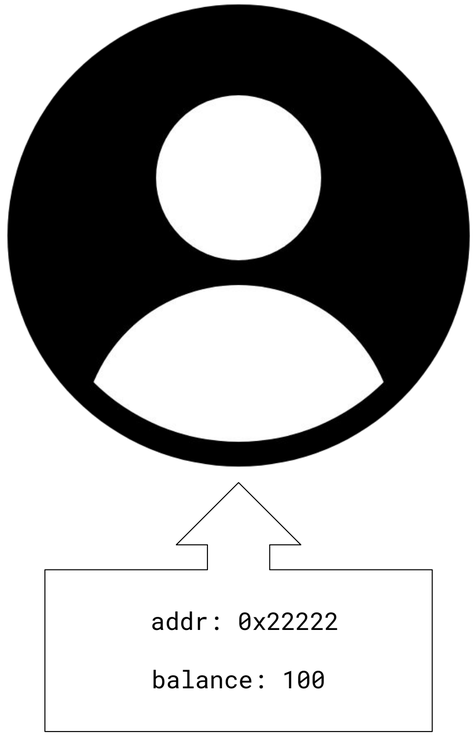
A QUICK RUNTHROUGH

10

tokenContract.allowed[0x11111][0x22222]







2 BUILDING ON ERC20

ERC20 HUMAN STANDARD

HUMAN READABILITY IS IMPORTANT

```
contract TokenContractFragment {
  // Human standard token modification
  string public constant name = "Token Name";
  string public constant symbol = "SYM";
  uint8 public constant decimals = 18;  // 18 is the most common number of decimal places
}
```

ERC20 (ADVANCED TOKEN)

NOT ALL TOKENS ARE THE SAME

```
/// @notice Create `mintedAmount` tokens and send it to `target`
/// @param target Address to receive the tokens
/// @param mintedAmount the amount of tokens it will receive
function mintToken(address target, uint256 mintedAmount) onlyOwner public {
    balanceOf[target] += mintedAmount;
    totalSupply += mintedAmount;
    Transfer(0, this, mintedAmount);
    Transfer(this, target, mintedAmount);
}
```

ERC20 (ADVANCED TOKEN)

NOT ALL TOKENS ARE THE SAME

```
    /// @notice `freeze? Prevent | Allow` `target` from sending & receiving tokens
    /// @param target Address to be frozen
    /// @param freeze either to freeze it or not
    function freezeAccount(address target, bool freeze) onlyOwner public {
        frozenAccount[target] = freeze;
        FrozenFunds(target, freeze);
    }

/// @notice Allow users to buy tokens for `newBuyPrice` eth and sell tokens for
`newSellPrice` eth
    /// @param newSellPrice Price the users can sell to the contract
    /// @param newBuyPrice Price users can buy from the contract
    function setPrices(uint256 newSellPrice, uint256 newBuyPrice) onlyOwner public {
        sellPrice = newSellPrice;
        buyPrice = newBuyPrice;
    }
```

ERC20 (ADVANCED TOKEN)

NOT ALL TOKENS ARE THE SAME

```
/// @notice Buy tokens from contract by sending ether
    function buy() payable public {
        uint amount = msg.value / buyPrice;               // calculates the amount
        _transfer(this, msg.sender, amount);              // makes the transfers
    }

    /// @notice Sell `amount` tokens to contract
    /// @param amount amount of tokens to be sold
    function sell(uint256 amount) public {
        require(this.balance >= amount * sellPrice);      // checks if the contract has
                                                          // enough ether to buy
        _transfer(msg.sender, this, amount);              // makes the transfers
        msg.sender.transfer(amount * sellPrice);
    // sends ether to the seller. It's important to do this last to avoid recursion attacks
    }
}
```

ERC223

WAIT, ERC20 DIDN'T FIX THIS?

Created: 2017-03-05, aimed to fix major flaws.

- Impossibility of handling incoming tx in receiver contract
- Tokens could be sent to contract that is not designed to work with tokens and *potentially could be lost*
- Token-transactions should match Ethereum *ideology of uniformity*. When a user needs to transfer their funds, they must always perform transfer. Uniform across sending to an externally owned account

Original ERC223 issue, implementation example

totalSupply

```
function totalSupply() constant returns (uint256 totalSupply)
```

Get the total token supply

name

```
function name() constant returns (string _name)
```

Get the name of token

symbol

```
function symbol() constant returns (bytes32 _symbol)
```

Get the symbol of token

decimals

```
function decimals() constant returns (uint8 _decimals)
```

Get decimals of token

balanceOf

```
function balanceOf(address _owner) constant returns (uint256 balance)
```

Get the account balance of another account with address _owner

ERC223

WAIT, ERC20 DIDN'T FIX THIS?

How much ERC20 tokens was lost (as of 27 Dec, 2017):

- QTUM, **$1,204,273** lost. watch on Etherscan
- EOS, **$1,015,131** lost. watch on Etherscan
- GNT, **$249,627** lost. watch on Etherscan
- STORJ, **$217,477** lost. watch on Etherscan
- Tronix , **$201,232** lost. watch on Etherscan
- DGD, **$151,826** lost. watch on Etherscan
- OMG, **$149,941** lost. watch on Etherscan
- STORJ, **$102,560** lost. watch on Etherscan

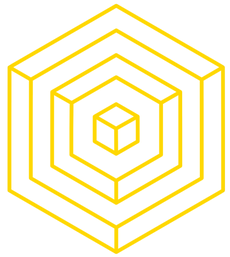
```
function transfer(address _to,
  uint _value,
  bytes _data) returns (bool) {...}

function tokenFallback(
  address _from,
  uint _value,
  bytes _data) {...}
```

ERC721

MORE LIKE ASSETS THAN TOKENS

Created: 2017-09-20, the “non-fungible” token model.

Original ERC721 issue, implementation example



```
contract ERC721 {

  // ERC20 compatible functions

  function name() constant returns (string name);
  function symbol() constant returns (string symbol);
  function totalSupply() constant returns (uint256 totalSupply);
  function balanceOf(address _owner) constant returns (uint balance);

  // Functions that define ownership

  function ownerOf(uint256 _tokenId) constant returns (address owner);
  function approve(address _to, uint256 _tokenId);
  function takeOwnership(uint256 _tokenId);
  function transfer(address _to, uint256 _tokenId);
  function tokenOfOwnerByIndex(address _owner, uint256 _index) constant returns (uint tokenId);

  // Token metadata

  function tokenMetadata(uint256 _tokenId) constant returns (string infoUrl);

  // Events

  event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
  event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);

}
```

ERC827

SIMILAR TO ERC223

Created: 2018-01-11, extends ERC20 functionality.

```
contract ERC827 is ERC20 {
  function approve( address _spender, uint256 _value, bytes _data) public returns (bool);
  function transfer( address _to, uint256 _value, bytes _data ) public returns (bool);
  function transferFrom( address _from, address _to, uint256 _value, bytes _data ) public
    returns (bool);
}
```

Original ERC827 issue, implementation example

MINIME TOKENS

KAWAII... !!

MINIME TOKENS

KAWAII... !!

Clone other tokens. Why though?

- Generating a voting token that is burned when you vote
- Generating a discount "coupon" that is redeemed when you use it
- Generating a token to enable the token holders to collect daily, monthly or yearly payments.
- Generating a token to limit participation in a token sale or similar event to holders of a specific token.
- Generating token that allows a central party complete control to transfer/generate/destroy tokens at will.

...Lots of other applications including all the applications the standard ERC 20 token can be used for.

MINIME TOKENS

KAWAII... !!

```
function createCloneToken(
    string _cloneTokenName,
    uint8 _cloneDecimalUnits,
    string _cloneTokenSymbol,
    uint _snapshotBlock,
    bool _isConstant
    ) returns(address) {
       …
  }
```

3 EXTRA TOOLS

OPENZEPPELIN

WHERE WOULD WE BE WITHOUT IT

openzeppelin

bash

```
[~]$ npm install zeppelin-solidity
Installing [============] 100%
> zeppelin-solidity@1.2.0 install
> scripts/install.sh
[~]$ vim ExampleToken.sol
[~]$ truffle console
> var myToken = ExampleToken.deployed();
> myToken.totalSupply()
100000000000000000000
> myToken.transfer(...)
true
```

vim ExampleToken.sol

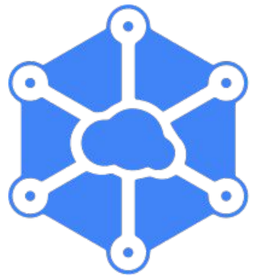
```
pragma solidity ^0.4.11;
import "zeppelin-solidity/contracts/token/StandardToken.sol"

contract ExampleToken is StandardToken {
 string public name = "ExampleToken";
 string public symbol = "EGT";
 uint public decimals = 18;
 uint public INITIAL_SUPPLY = 10000 * (10 ** decimals);

 function ExampleToken() {
  totalSupply = INITIAL_SUPPLY;
  balances[msg.sender] = INITIAL_SUPPLY;
 }
}
```

OPENZEPPELIN

WHERE WOULD WE BE WITHOUT IT

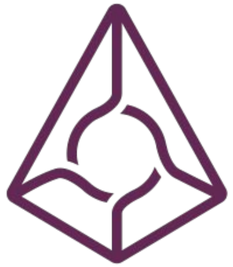
ARAGON

TRUFFLE



STORJ

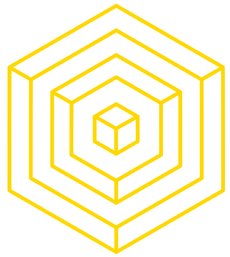
civic



augur

COSMOS

4 MIDTERM



github.com/Blockchain-for-Developers/sp18-midterm-p1

README.md

MIDTERM: Launching an ICO

Welcome to the Blockchain for Developers DeCal's midterm assignment. You'll be designing your own ICO.

Midterm Instructions

Your task is to complete both `Token.sol` and `Crowdsale.sol` such that they each function as described below. `Queue.sol` is also a necessary data structure contract. Notice that `Token.sol` inherits from `contracts/interfaces/ERC20Interface.sol`.

Implement all three contracts and test them in order to demonstrate your understanding of external calls, timestamping, testing, and everything else learned so far!

All work should be done in *contracts/Crowdsale.sol*, *contracts/Queue.sol* and *contracts/Token.sol*. Do not make changes to any other files. The exception to this is for testing: add as many test files as you need, of course

SEE YOU NEXT TIME

Web3 and Smart Contract Architecture

hi