Solidity Pt. 2

Portland State
Computer Science

Lessons 3-5

Libraries/OpenZeppelin, SafeMath

Time

Random number generation

Transfers

Tokens

Comments

Including libraries and contracts

- OpenZeppelin repository containing source code for implementing commonly used functions and base smart contracts
  - Can include base contract class to derive from
  - Can include only functions
- Included into contract via "`import`" statement in Solidity

Example: Ownership contract

- Common features for denoting and managing contract control
  - Set owner to creator of contract in constructor
  - Implement modifier that throws an error if owner is not the caller

```
contract Ownable {
    address private _owner;
    constructor() internal {
        _owner = msg.sender;
    }

    function owner() public view returns(address) {
         return _owner;
    }

    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    function isOwner() public view returns(bool) {
        return msg.sender == _owner;
    }
```

- `renounceOwnership` to remove owner with no replacement (functions with "`onlyOwner`" modifier can no longer be called)
  - e.g. to disable God mode ☺
- Transfer ownership to new owner
  - External `transferOwnership` call protected with `onlyOwner`
  - Internal `_transferOwnership` call not callable from outside

```
    function renounceOwnership() public onlyOwner
        _owner = address(0);
    }

    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        _owner = newOwner;
    }
}
```

Example: SafeMath library

- What happens here

```
uint8 number = 255;
number++;
```

- and here?

```
uint8 number = 0;
number--;
```

  - Same as C: integer overflow and underflow
  - DASP Top 10 D3 (Arithmetic issues)
- Motivates OpenZeppelin SafeMath library for preventing overflow and underflow
  - SafeMath library performs operations, but includes an `assert` to ensure no issues

But first ...

- Defining libraries similar to contracts
  - Done via `library` keyword

```
library SafeMath {
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }
  . . .
  . . .
}
```

- Include via the `using` keyword that associates library methods to a specific datatype
  - e.g. Library code used for datatype `uint256`

```
import "./safemath.sol";
using SafeMath for uint256;
uint256 a = 5;
uint256 b = a.add(3); // 5 + 3 = 8
uint256 c = a.mul(2); // 5 * 2 = 10
```

- Uses code in library to perform operations

```
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
  if (a == 0) { return 0; }
  uint256 c = a * b;
  assert(c / a == b);
  return c;
}
```

  - Note: first argument is implicit (`a`)
- What about `uint8`, `uint16`, `uint32`?
  - Must implement SafeMath operations per datatype

Time

- `now` keyword returns Unix timestamp of latest block (# of seconds since epoch 1/1/1970)
- Semantic issue
  - Looks like a variable, but is actually a *function* call
- Native time units of `seconds`, `minutes`, `hours`, `days`, `weeks`, and `years` part of Solidity
  - Unit conversion done by language similar to currency conversion

```
function updateTimestamp() public {
    lastUpdated = now;
}

function fiveMinutesHavePassed() public view returns (bool) {
    return (now >= (lastUpdated + 5 minutes));
}
```

Random numbers (or lack thereof)

- `keccak256` hash function
  - Avalanche effect results in random distribution of output

```
// Generate a random number between 1 and 100:
uint nonce = 0;
uint random1;
uint random2;

random1 = uint(keccak256(abi.encodePacked(now, msg.sender, nonce))) % 100;
nonce++;

random2 = uint(keccak256(abi.encodePacked(now, msg.sender, nonce))) % 100;
```

  - But, input often known to everyone or subject to manipulation
    - Who controls `now` variable (block timestamp)?
      - Miner
      - Can choose a value to his/her advantage
    - What if miner doesn't like the random number generated after mining?
      - Can keep mined block to him/herself
      - DASP Top 10 D7 (Front-running), D8 (Time manipulation)

- Agreeing on random numbers problematic
  - Secure-coin flipping (not possible, afaik)
  - Oracles off-chain?
    - https://ethereum.stackexchange.com/questions/191/how-can-i-securely-generate-a-random-number-in-my-smart-contract
- Contracts that rely upon random numbers vulnerable
  - DASP Top 10 D6 (Bad randomness)

Transfers and withdrawals

- Smart contracts can send and receive Ether to/from wallets and other contracts
- Example: Owner of contract cashes out all $ from it
  - Specify address of recipient (e.g. `_owner`)
  - Then using built-in function `address()` and the special keyword `this` to specify current contract before accessing the attribute `balance` to get the amount of Ether the contract has
  - Before invoking built-in `transfer()` function in `address` to send funds to `_owner`.

```
contract GetPaid is Ownable {
    function withdraw() external onlyOwner {
        address _owner = owner();
        _owner.transfer(address(this).balance);
    }
}
```

- Example: Consignment store giving seller money after someone buys item

```
contract ConsignmentStore {
    uint commission = 0.001 ether;
    function buyItem(address itemOwner) external payable {
        ...
        itemOwner.transfer(msg.value - commission);
        ...
    }
}
```

  - What if `msg.value` is less than commission?
- Example: On-line store contract repays a sender if they've overpaid for an item

```
uint itemPrice = 0.01 ether;
msg.sender.transfer(msg.value - itemPrice);
```

  - What happens when `msg.value` is 0.001?
  - DASP Top 10 D3 (Arithmetic issues)
  - CTF exercise

Tokens

- Special contracts that track ownership stakes within it similar to corporate stocks
  - Each token with a pre-defined interface (e.g. standard set of functions) to enable exchanges
  - Many kinds of tokens, standardized via ERC (Ethereum Request for Comments)
  - Main tokens being used: ERC 721 and ERC 20

ERC 721 standard

- Unique (non-fungible), indivisible tokens suitable for single owner object ownership (http://erc721.org/)

```
import "./erc721.sol"

contract foo is ERC721 {
}
```

  - Supports standard calling interface

```
function balanceOf(address _owner) public view returns (uint256 _balance);
function ownerOf(uint256 _tokenId) public view returns (address _owner);
function transfer(address _to, uint256 _tokenId) public;
function approve(address _to, uint256 _tokenId) public;
function takeOwnership(uint256 _tokenId) public;
```

  - Supports standard events for web interface (will revisit with `web3.js`)

```
event Transfer(address indexed _from, address indexed _to,
               uint256 _tokenId);

event Approval(address indexed _owner, address indexed _approved,
               uint256 _tokenId);
```

ERC 20 tokens

- Interchangeable (fungible), divisible tokens suitable for being used as currency
  - Proposed by Fabian Vogelsteller 11/2015 to implement tradeable digital assets in an interoperable manner
    - An application written to interact with one ERC20 token can directly work with another ERC20 token
  - Commonly used for crowdfunding startup companies doing an initial coin offering to raise money (ICO) by issuing virtual shares
    - Examples
      - EOS, TON, Tezos, Filecoin (> $200 million each)
      - Polyswarm podcast for how it was used to raise money
    - Total number of ERC-20 token contracts
      - https://etherscan.io/tokens

ERC 20 token interface

```
contract ERC20 {
  // Get the total token supply in circulation
  function totalSupply() constant returns (totalSupply);

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

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

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

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

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

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

ERC 20 example token

```
pragma solidity ^0.4.17;

contract MyToken is ERC20 {
  mapping (address => mapping (address => uint256)) allowed;
  mapping (address => uint256) balances;

  function MyToken() {
    // There will be 5 million tokens
    totalSupply = 5 * (10 ** 6);
    // All initial tokens belong to the owner
    balances[msg.sender] = totalSupply;
  }

  // Gets the balance of the specified address.
  function balanceOf(address _owner) constant returns (uint256 balance) {
    // Return the balance of _owner
    return balances[_owner];
  }

  // Transfer tokens to a specified address
  function transfer(address _to, uint256 _value) returns (bool) {
    require(balances[msg.sender] >= _value);
    // Decrease the balance of the sender by _value
    // Then, increase the value of _to by _value
    Transfer(msg.sender, _to, _value);
    return true;
  }
```

```
  // Transfer tokens from one address to another
  function transferFrom(address _from, address _to, uint256 _value) returns (bool) {
    var _allowance = allowed[_from][msg.sender];
    // Make sure the function does not get executed if _allowance is lower than _value
    // Make sure the balance of _from is larger than _value
    balances[_to] = balances[_to] + _value; balances[_from] = balances[_from] - _value;
    allowed[_from][msg.sender] = _allowance - _value;
    Transfer(_from, _to, _value);
    return true;
  }

  // Function to check the amount of tokens that an owner allowed to a spender
  function allowance(address _owner, address _spender) constant returns (uint256 remain) {
    return allowed[_owner][_spender];
  }

  // Approve the passed address to spend the specified amount of tokens on behalf
  // of msg.sender
  function approve(address _spender, uint256 _value) returns (bool) {
    // https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
    require((_value == 0) || (allowed[msg.sender][_spender] == 0));
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value); return true;
  }
}
```

Comments

- Similar to Javascript
  - Single-line comments via //
  - Multi-line comments via /* */
  - natspec standard similar to docstring for documenting function parameters (@param) and return values (@return)
  - Ensure your code is properly commented using natspec

```
/// @title A contract for basic math operations
/// @author H4XF13LD
/// @notice For now, this contract just adds a multiply function
contract Math {
  /// @notice Multiplies 2 numbers together
  /// @param x the first uint.
  /// @param y the second uint.
  /// @return z the product of (x * y)
  /// @dev This function does not currently check for overflows
  function multiply(uint x, uint y) returns (uint z) {
    // This is just a normal comment, and won't get picked up by natspec
    z = x * y;
  }
}
```

Example contracts

Fundraiser

```
contract Fundraiser {
    address public owner;
    uint256 public target;        // target fundraising value
    uint256 public endTime;       // time that fundraiser ends
    Contributor[] contributors;   // list of contributors
    struct Contributor {
        address userAddress;
        uint contribution;
    }
    constructor(uint _target, uint duration) public payable {
        owner = msg.sender;
        target = _target;
        endTime = now + duration;
    }
    function contribute() public payable {
        // require that fundraiser hasn't ended yet
        require(now < endTime);
        // add to list of contributors
        contributors.push(Contributor(msg.sender, msg.value));
    }
```

```
    function collect() public {
        //once target has been reached, owner can collect funds
        require(address(this).balance >= target);
        require(msg.sender == owner);
        selfdestruct(owner);
    }

    function refund() public {
        // If goal not met on time, anyone can trigger refund()
        require(now > endTime);
        require(address(this).balance < target);
        // refund all contributors
        for (uint i; i<contributors.length; i++) {
contributors[i].userAddress.transfer(contributors[i].contribution)
;
        }
    }

    function balance() public view returns(uint) {
        return address(this).balance;
    }
}
```

Lesson 6

web3.js

web3.js

- Web3.js running within browser interfaces with wallet (e.g. Metamask) to send transactions to blockchain and receive event callbacks from it

web3.js details

- web3 provider variable used to specify node to interact with (e.g. Infura)
  - Include in a NodeJS backend application that needs to interact with blockchain
  - Include in frontend application that needs to interact with blockchain
  - Session can be over a web socket

```
const web3 = new Web3(
            new Web3.providers.WebsocketProvider("wss://mainnet.infura.io/ws")
           );
```

  - Session can be over HTTPS

```
const Web3 = require("web3")
const web3 = new Web3(
         new Web3.providers.HttpProvider("https://mainnet.infura.io/..."))
```

  - web3 methods can then be used in application

```
web3.eth.getBalance("0x5A0b54D5dc17e0AadC383d2db43B0a0D3E029c4c",
                    function(err, result) {
                       if (err) { console.log(err) }
                       else {
                        console.log(web3.utils.fromWei(result, "ether"))
                       }
                    })
```

- Communication to/from full node done via JSON-RPC
  - akin to a REST API

```
CryptoZombies.methods.createRandomZombie("Vitalik").send({
          from: "0xb60e8dd61c5d32be8058bb8eb970870f07233155",
          gas: "3000000"})
```

```
{
  "jsonrpc":"2.0",
  "method":"eth_sendTransaction",
  "params":[
       {"from":"0xb60e8dd61c5d32be8058bb8eb970870f07233155",
        "to":"0xd46e8dd67c5d32be8058bb8eb970870f07244567",
        "gas":"0x76c0",
        "gasPrice":"0x9184e72a000",
        "value":"0x9184e72a",
        "data":"0xd46e8dd67c5d32be8d46e...8eb970870f07244502445675"
       }
   ],
   "id":1
}
```

Metamask & web3.js

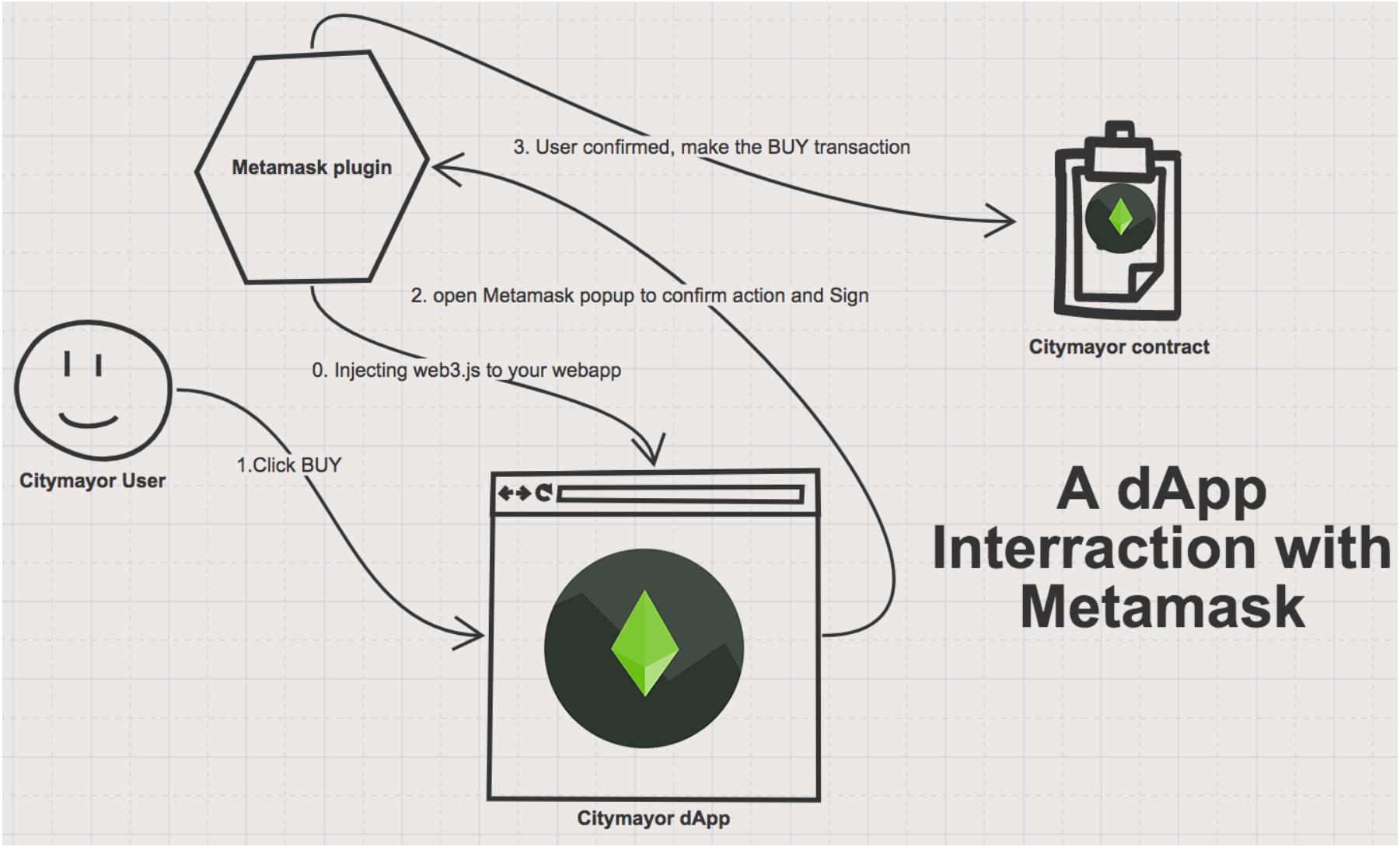
- Metamask
  - Browser extension for managing Ethereum accounts and private keys
  - Injects itself into web pages to set `web3.currentProvider` to itself
    - Allows client browser to interact with DApps on blockchain directly
    - Include in single-page web applications that interact with blockchain without web server intermediary

```
window.addEventListener('load', function() {
  if (typeof web3 !== 'undefined') {
    web3js = new Web3(web3.currentProvider);
  } else {
    // Prompt to install Metamask
  }
  ...
)
```

  - Then, get user account using provider

```
var userAccount = web3.eth.accounts[0]
```

Example


Metamask plugin
3. User confirmed, make the BUY transaction
Citymayor contract
2. open Metamask popup to confirm action and Sign
0. Injecting web3.js to your webapp
Citymayor User
1.Click BUY
Citymayor dApp
A dApp Interraction with Metamask

Interacting with contracts in web3.js

- Need address to interact with to be set in DApp's JavaScript code

```
address myContractAddress = 0x06012c8cf97BEaDab38...
```

- Need ABI (application binary interface) to format calls to the contract
  - Compiled and stored so that clients can interact with it appropriately
  - From Lesson 6, Chapter 3
    - Include ABI via `<script>` include in `<head>` for `cryptozombies_abi.js`

cryptozombies_abi.js

```
var myABI = [
  { "constant": false,
    "inputs": [
      { "name": "_to",
        "type": "address"
      },
      { "name": "_tokenId",
        "type": "uint256"
      }
    ],
    "name": "approve",
    "outputs": [],
    "payable": false
  },
  {
    "constant": false,
    "inputs": [
      { "name": "_zombieId",
        "type": "uint256"
      }
    ],
    "name": "levelUp",
    "outputs": [],
    "payable": true,
    "stateMutability": "payable",
    "type": "function"
  },
  ...
```

```
address myContractAddress = 0x06012c8cf97BEaDab38...
// Instantiate myContract
var myContract = new web3js.eth.Contract(myABI, myContractAddress);
```

web3 `call` vs. `send`

- Invoke methods in ABI
- `call()`
  - Used to invoke view and pure functions in ABI
  - In CryptoZombies contract `Zombie[] public zombies;`
    - Public list of zombies automatically has a *gettr* function associated with it
  - From Javascript, can use below call to retrieve
    - `cryptoZombies.methods.zombies(id).call()`
  - Only runs on local node so no gas required and wallet will not be prompted for funds
  - Returns a JSON object

```
{
  "name": "H4XF13LD MORRIS'S COOLER OLDER BROTHER",
  "dna": "1337133713371337",
  "level": "9999",
  "winCount": "999999999",
  "lossCount": "0"
}
```

- `send()`
  - Used to create a transaction and send to blockchain
  - Requires user to pay gas to execute so wallet will be prompted for funds via pop-up
  - Similar to `call()`, but must include a sending (`from`) address

```
cryptoZombies.methods.createRandomZombie(name).send({
    from: userAccount
})
```

    - web3 provider (Metamask) automatically signs transaction when approved by user
    - Wallet address initializes `msg.sender` in transaction sent to contract
  - Significant delay before transaction committed to blockchain so requires the use of asynchronous JavaScript handling

- Example: calling payable functions via `send()`

```
function levelUp(uint _zombieId) external payable {
    require(msg.value == levelUpFee);
    zombies[_zombieId].level++;
}
```

  - In JavaScript, denomination units are in `wei` (not Ether)
    - Function to do conversion supplied in web3.js
    - `10^18 wei =1 ether`

```
cryptoZombies.methods.levelUp(zombieId).send({
      from: userAccount,
      value: web3js.utils.toWei("0.001", "ether")
    })
```