Hands-on: Begginers to Solidity

CBSoft 2018

Blockchain & Smart Contracts Crash Course

Henrique Rocha (henrique.rocha@gmail.com)

https://github.com/hscrocha/bc-tutorial-2018

https://hscrocha.github.io/bc-tutorial-2018/

Introduction

- Solidity is the main language for Smart Contracts in the Ethereum Platform.

- High-level language inspired by C++, Javascript, Python.

- Solidity is statically typed

- Solidity supports inheritance (multiple), user defined types (structs, enums), “aspects” (modifier).

Remix

- Remix is a web-browser IDE for Solidity
- “The best way to try out Solidity right now is using Remix” (Solidity Documentation)

https://remix.ethereum.org/

Remix

```
1   pragma solidity ^0.4.24;
2   /**
3    * @title My First Contract
4    * @author Henrique
5    */
6   contract MyFirstContract {
7       string name;
8
9       function setName(string _name){
10          name = _name;
11      }
12
13      function getName() returns (string) {
14          return name;
15      }
16
17  } //end of contract
```

Primitive types

- bool
- int / uint … int8 / uint8 … int256 / uint256
- string : dynamically-sized UTF-8-encoded string.
- byte / bytes / bytes1… bytes32 : a single byte or a byte array
- address : an Ethereum address (20bytes), references an account or contract. Every address has the member **balance**
- fixed / ufixed … fixed0x256 : fixed point numbers.

Predefined Modifiers

- Visibility Modifiers: private, public, internal, external
- Mutability Modifiers: view, pure
- Cryptocurrency Modifiers: payable

Functions

- The “rules” enforced by a contract is defined by its functions.
- It always starts with the “function” keyword.
- It can return more than one value (or none at all).
    - Caution: not every type can be returned (e.g., structs)
- Solidity supports function overloading
    - Carefull that some types resolve as the same parameter (e.g., contract ~ address)

My First Contract (revised)

```
1   pragma solidity ^0.4.24;
2   /**
3    * @title Simple Contract
4    * @author Henrique
5    */
6   contract SimpleContract {
7       uint public data = 10;
8       string private name;
9
10      function setName(string _name) public{
11          name = _name;
12      }
13
14      function getName() public view returns (string) {
15          return name;
16      }
17
18      function get() public view returns (string, uint){
19          return (name,data);
20      }
21  } //end of contract
```

Constructor

- Contracts can have only 1 construtor
- A constructor can have parameters
- Visibility: *public* or *internal*

```
constructor(address a, address b) public {
    //do something...
}
```

Pre-defined Variables

- msg : reference the current message call
  - msg.sender : address; the account that initiated the call
  - msg.value : uint; the amount of Wei sent
- block : reference the current block
  - block.timestamp or now : uint, timestamp

Exceptions

- The way to handle error in Solidity is by raising exceptions.
- An exception undo all changes made in the current execution (propagating to the call and sub-call).
  - require (condition, msg) : raises an exception if the condition is false, returning the msg to the caller.
  - revert (msg) : raises an exception and returns the msg
  - assert(condition) : raises an exception if the condition is false (used for internal checks)

Custom Modifiers

- Modifiers amend the semantics of a function.
    - Usually used for checking conditions and raising exceptions.
    - Similar to a limited aspect.

```
modifier checkBalance(uint amount){
    require(address(this).balance >= amount,
        "Insuficient funds for this operation.");
    _;
}
```

Restricted Access Contract

```
1  pragma solidity^0.4.24;
2  /**
3   * @title Restricted Access Example
4   * @author Henrique
5   */
6  contract Restricted {
7      address private owner;
8      uint private data;
9
10     constructor() public{
11         owner = msg.sender;
12     }
13
14     modifier onlyOwner(){
15         require(owner == msg.sender, "Only owner can call this function");
16         _;
17     }
18
19     function setData(uint d) public onlyOwner{
20         data = d;
21     }
22
23     function getData() public view returns (uint) {⟺}
26 } //end of contract
```

Dealing with Money (Ether)

- One the major advantages of Smart Contracts is dealing with cryptocurrency (always in Wei)

- We need to use specific syntax to handle Ether

  - payable : any function that receives ether must have this modifier

  - <address>.send(uint amount) : sends money to that address (returns false if failed)

  - <address>.transfer(uint amount) : sends money to that address (raises an exception if failed)

  - Remember the **msg.value** and <**address**>**.balance** members

Wallet Contract (v.1)

```
1  pragma solidity^0.4.24;
2  /**
3   * @title Wallet Contract Example v1
4   * @author Henrique
5   **/
6  contract MyWallet{
7      address private owner;
8      uint8 constant private version = 1;
9
10     constructor() public {
11         owner = msg.sender;
12     }
13
14     modifier onlyOwner(){⟺}
18     modifier checkBalance(uint amount){⟺}
22
23     function pay(address receiver, uint amount) public onlyOwner checkBalance(amount) {
24         receiver.transfer( amount );
25     }
26
27     function deposit() public payable {⟺}
29
30     function withdraw(uint amount) public onlyOwner checkBalance(amount) {⟺}
33 } //end of contract
```

Events

- Event are used for two important reasons in Solidity

  - **Logging:** every event (and its parameters) are stored in the transaction log. The information in the event log is more easy to find and access.

  - **Client Notification:** client applications using the Smart Contract can track the events and react to them outside the blockchain.

Wallet Contract (v.2)

```
 6  contract MyWallet{
 7      address private owner;
 8      uint8 constant private version = 2;
 9
10      event PayEvent(address receiver, uint amount);
11      event DepositEvent(address sender, uint amount);
12
13      constructor() public {...}
16
17      modifier onlyOwner(){...}
21      modifier checkBalance(uint amount){...}
25
26      function pay(address receiver, uint amount) public onlyOwner checkBalance(amount) {
27          receiver.transfer( amount );
28          emit PayEvent(receiver, amount);
29      }
30
31      function deposit() public payable {...}
34
35      function withdraw(uint amount) public onlyOwner checkBalance(amount) {...}
38  } // end of contract
```

Fallback Function

- Un-Named function that is executed on a call if no other function match the identifier.

- This function is also executed whenever the contract received Ether without data (in such case, the fallback must have the **payable** modifier)

- The fallback function cannot have arguments and cannot return anything.

    - The fallback should rely only on basic logging to receive Ether from send and transfer calls.

Wallet Contract (v.3)

```
2  /**
3   * @title Wallet Contract Example v3
4   * @author Henrique
5   **/
6  contract MyWallet{
7      address private owner;
8      uint8 constant private version = 3;
9
10     event PayEvent(address receiver, uint amount);
11     event DepositEvent(address sender, uint amount);
12
13     constructor() public {⟺}
16
17     modifier onlyOwner(){⟺}
21     modifier checkBalance(uint amount){⟺}
25
26     function pay(address receiver, uint amount) public onlyOwner checkBalance(amount) {⟺}
30
31     function withdraw(uint amount) public onlyOwner checkBalance(amount) {⟺}
34
35     function() public payable { //fallback
36         emit DepositEvent(msg.sender, msg.value);
37     }
38 } // end of contract
```

Complex Types

- Struct
- Enums
- Array
- Mapping
- Contract

End of Begginers to Solidity…

Let’s move to Solidity Patterns.