Ethereum and Solidity - The Complete Developer's Guide

Section 3. Advanced Smart Contracts

Training project "The Lottery Contract"

Lottery Contract

Variables

Name	Purpose
manager	Address of person who created the contract
players	Array of addresses of people who have entered

Functions

Name	Purpose
enter	Enters a player into the lottery
pickWinner	Randomly picks a winner and sends them the ether

Workflow

- Players send money to the contract
- Manager tells the contract to randomly select the winner
  - ❓ Q: Is it possible to do this by timer?
  - ❓ Q: Is it possible to do this by external - non contract event?
- Contract selects the winner and send him all the money, and restarts

Contract

Basic Types

Basic Types

Name	Notes	Examples
string	Sequence of characters	"Hi there!" / "Chocolate"
bool	Boolean value	true / false
int	Integer, positive or negative. Has no decimal	0 / -30000 / 59158
uint	'Unsigned' integer, positive number. Has no decimal	0 / 30000 / 999910
fixed/ufixed	'Fixed' point number. Number with a decimal after it	20.001 / -42.4242 / 3.14
address	Has methods tied to it for sending money	0x19ba41b9c8c8cae139c8d

int

Integer Ranges		
Name	Lower Bound	Upper Bound
int8	-128	+27
int16	-32,768	32,767
int32	-2,147,483,648	2,147,483,647
...	...	...
int256	Really, really negative	Really, really big

int ... int256

uint (is similar)

Unsigned Integer Ranges		
Name	Lower Bound	Upper Bound
uint8	0	255
uint16	0	65,535
uint32	0	4,294,967,295
...	...	...
uint256	0	Really, really big

uint ... uint256

Solidity Variable types

Message Global Variable

The 'msg' Global Variable	
Property Name	Property Name
msg.data	'Data' field from the call or transaction that invoked the current function
msg.gas	Amount of gas the current function invocation has available
msg.sender	Address of the account that started the current function invocation
msg.value	Amount of ether (in wei) that was sent along with the function invocation

Each function call has this 'msg' variable available

Reference Types

Reference Types		
Name	Notes	Examples
fixed array	Array that contains a single type of element. Has an unchanging length.	int[3] --> [1, 2, 3] / bool[2] --> [true, false]
dynamic array	Array that contains a single type of element. Can change in size over time	int[] --> [1,2,3] / bool[] --> [true, false]
mapping	Collection of key value pairs. Think of Javascript objects, Ruby hashes, or Python dictionary. All keys must be of the same type, and all values must be of the same type	mapping(string => string) / mapping(int => bool)
struct	Collection of key value pairs that can have different types.	struct Car { string make; string model; uint value; }

- public array
  - Generates public function with index param - to retrieve array's element by index
  - If need to return array then need to explicitly create a public function

```
function getPlayers() public view returns (address[]) {
    return players;
}
```

- Gotcha with nested rays

```
const myArray = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

Solidity World	ABIJS/Web3 World
Nested Dynamic Arrays	Nested Dynamic Arrays
	No Dice

⚠️ We cannot transfer nested array of strings to JS!

- string in Solidity is represented as dynamic array

⚠️ We cannot transfer array of strings to JS!

Validation (with Require statements)

- require(msg.value > 0.01 ether)
  - To ensure that > 0.01 ether was sent to the transaction Enter()
- Gotha with require()
  - ⚠️ Error message returned is not informative

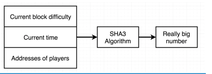
```
address public manager;
address[] public players;

function Lottery() public {
    manager = msg.sender;
}

function enter() public payable {
    require(msg.value > .01 ether);
    players.push(msg.sender);
}
```

Remix Debugger

- A tool to debug the contract
- 👍 Remix Debugger can be used to see where error happened and step back/step forward

Generate random number

- ⚠️ There is no Random Generator in Solidity!
- We can emulate it - pseudorandom



  - ⚠️ But such RND can be cheated!
- Other global variables
  - block
    - block.difficulty
  - now

```
function random() private view returns (uint) {
    return uint(keccak256(block.difficulty, now, players));
}
```

Sending Ether

- Address variable has methods
  - transfer(amount)
    - Sends money from the current contract to the specified address
- this.balance is the total amount of money that contract currently has

```
function pickWinner() public {
    require(msg.sender == manager);
    uint index = random() % players.length;
    players[index].transfer(this.balance);
    players = new address[](0);
}
```

Function modifier

- When we want to require specific Caller
  - We can use require(msg.sender == manager)
    - e.g. to ensure that only manager can call PickWinner()
- Or we can use MODIFIER

```
function pickWinner() public restricted {
    uint index = random() % players.length;
    players[index].transfer(this.balance);
    players = new address[](0);
}

modifier restricted() {
    require(msg.sender == manager);
    _;
}
```

  - Underscore is a target where the function code will be placed!

Testing

Deploy contract locally

- Almost same as before - the only difference is
  - no parameter in constructor

ContractName.test.js

```
const assert = require("assert");
const ganache = require("ganache-cli");
const Web3 = require("web3");
const web3 = new Web3(ganache.provider());

const { interface, bytecode } = require("../compile");

let lottery;
let accounts;

beforeEach( fn: async () => {
  accounts = await web3.eth.getAccounts();

  lottery = await new web3.eth.Contract(JSON.parse(interface))
    .deploy( options: { data: bytecode })  ContractSendMethod
    .send( options: { from: accounts[0], gas: "1000000" });
});
```

Asserting the deployment

```
describe( title: "Lottery Contract", fn: () => {
  it( title: "deploys a contract", fn: () => {
    assert.ok(lottery.options.address);
  });
```

Asserting entering the contract

```
  it( title: "allows one account to enter", fn: async () => {
    await lottery.methods.enter().send({
      from: accounts[0],
      value: web3.utils.toWei( val: "0.02",  unit: "ether"),
    });

    const players = await lottery.methods.getPlayers().call({
      from: accounts[0],
    });

    assert.equal(accounts[0], players[0]);
    assert.equal( actual: 1, players.length);
  });
```

Asserting multiple contract participants

```
  it( title: "allows multiple accounts to enter", fn: async () => {
    await lottery.methods.enter().send({
      from: accounts[0],
      value: web3.utils.toWei( val: "0.02",  unit: "ether"),
    });
    await lottery.methods.enter().send({
      from: accounts[1],
      value: web3.utils.toWei( val: "0.02",  unit: "ether"),
    });
    await lottery.methods.enter().send({
      from: accounts[2],
      value: web3.utils.toWei( val: "0.02",  unit: "ether"),
    });

    const players = await lottery.methods.getPlayers().call({
      from: accounts[0],
    });

    assert.equal(accounts[0], players[0]);
    assert.equal(accounts[1], players[1]);
    assert.equal(accounts[2], players[2]);
    assert.equal( actual: 3, players.length);
  });
```

Asserting function call validation with try-catch

```
  it( title: "requires a minimum amount of ether to enter", fn: async () => {
    try {
      await lottery.methods.enter().send({
        from: accounts[0],
        value: 0,
      });
      assert( value: false);
    } catch (err) {
      assert(err);
    }
  });
```

```
  it( title: "only manager can call pickWinner", fn: async () => {
    try {
      await lottery.methods.pickWinner().send({
        from: accounts[1],
      });
      assert( value: false);
    } catch (err) {
      assert(err);
    }
  });
```

Asserting distribution of money

```
  it( title: "sends money to the winner and resets the players array", fn: async () => {
    await lottery.methods.enter().send({
      from: accounts[0],
      value: web3.utils.toWei( val: "2",  unit: "ether"),
    });

    const initialBalance = await web3.eth.getBalance(accounts[0]);
    await lottery.methods.pickWinner().send({ from: accounts[0] });
    const finalBalance = await web3.eth.getBalance(accounts[0]);
    const difference = finalBalance - initialBalance;

    assert( value: difference > web3.utils.toWei( val: "1.8",  unit: "ether"));
  });
```