pragma solidity ^0.4.0;

contract HelloWorld {

uint public age;

string public name;

address public owner;

function HelloWorld() {

owner = msg.sender;

}

modifier onlyOwner(address \_account)

{

require(msg.sender == \_account,"Only Owner is authorized!!!" );

\_; // Do not forget the "\_;"! It will be replaced by the actual function body when the modifier is used.

}

function setValue(uint \_age, string \_name) onlyOwner(owner) {

//require(msg.sender==owner);

age = \_age;

name = \_name;

}

function getAge() returns(uint) {

return age;

}

function getName() returns(string) {

return name;

}

}

pragma solidity ^0.4.0;

contract HelloWorld {

struct Person {

uint age;

string name;

}

address public owner;

Person public persons;

function HelloWorld() {

owner = msg.sender;

}

modifier onlyOwner(address \_account)

{

require(msg.sender == \_account,"Only Owner is authorized!!!" );

\_; // Do not forget the "\_;"! It will be replaced by the actual function body when the modifier is used.

}

function setValue(uint \_age, string \_name) onlyOwner(owner) {

//require(msg.sender==owner);

persons.age = \_age;

persons.name = \_name;

}

function getAge() returns(uint) {

return persons.age;

}

function getName() returns(string) {

return persons.name;

}

}

pragma solidity ^0.4.0;

contract Ballot {

struct Voter {

uint weight;

bool voted;

uint8 vote;

address delegate;

string name;

}

struct Proposal {

uint voteCount;

}

address public chairperson;

mapping(address => Voter) public voters;

Proposal[] public proposals;

uint public numofVoters;

/// Create a new ballot with $(\_numProposals) different proposals.

function Ballot(uint8 \_numProposals) public {

chairperson = msg.sender;

voters[chairperson].weight = 1;

proposals.length = \_numProposals;

numofVoters = 0;

}

/// Give $(toVoter) the right to vote on this ballot.

/// May only be called by $(chairperson).

function giveRightToVote(address toVoter,string \_name) public {

require(chairperson==msg.sender,"Only chairperson can add voter!!!");

if (voters[toVoter].voted) return;

voters[toVoter].weight = 1;

voters[toVoter].name = \_name;

numofVoters= numofVoters + 1;

}

/// Give a single vote to proposal $(toProposal).

function vote(uint8 toProposal) public {

Voter storage sender = voters[msg.sender];

if (sender.voted || toProposal >= proposals.length) return;

sender.voted = true;

sender.vote = toProposal;

proposals[toProposal].voteCount += sender.weight;

}

function getVoterName(address \_addr) public constant returns(string) {

return voters[\_addr].name;

}

function winningProposal() public constant returns (uint8 \_winningProposal) {

uint256 winningVoteCount = 0;

for (uint8 prop = 0; prop < proposals.length; prop++)

if (proposals[prop].voteCount > winningVoteCount) {

winningVoteCount = proposals[prop].voteCount;

\_winningProposal = prop;

}

}

}



* 1. Copy genesis.json in your preferred location. e.g. C:\BlockChain\. Copy from https://github.com/SSPgg/BlockchainWork/blob/master/genesis.json
  2. Create a new folder ‘Chain1’ in your selected location, e.g. C:\BlockChain\Chain1\
  3. Navigate to the Chain1 folder in DOS prompt
  4. To initialize blockchain use: ***geth --datadir "Chain1" init "genesis.json"***
  5. **geth --datadir "Chain1" account new** :- Another way to create a new account before invoking the client.
  6. ***geth --datadir "Chain1" console 2> console.log*** : To start the chain and console in same window, this way log the messages in a log file.

# **Transfer ETH in geth:**

1. In geth console to unlock an account for longer time use the below web3.personal.unlockAccount("acct#", "pass", 10000000)
2. Transfer Ether from one account to another in console
   1. ***var a = personal.listAccounts*** 🡪 Assign the list of addresses to a variable.
   2. ***eth.getBalance(a[1])*** 🡪 Check the balance of the receiver account
   3. ***eth.sendTransaction({from:a[0],to:a[1],value:5000000000000000000})*** 🡪 Transfer
   4. ***Miner.start()*** to confirm the transaction 🡪 Start mining
   5. ***txpool.status*** or ***eth.pendingTransactions***🡪 To check if any pending transactions
   6. ***miner.stop()*** to stop mining 🡪 Stop mining if nothing pending to confirm
   7. ***eth.getBalance(a[1])*** 🡪 Check the balance
   * Create 2 folders Node1 & Node2. E.g. under C:\BlockChain folder and initiate them with the genesis file.

C:\BlockChain>***geth --datadir=.\Node1 init genesis.json***

C:\BlockChain>***geth --datadir=.\Node2 init genesis.json***

* + Start the 2 nodes in separate DOS shell

***geth --identity "Node1" --networkid 42 --datadir "./Node1" --nodiscover --rpc --rpcport "8042" --port "30303" --unlock 0 --password ./pass.sec --ipcpath "./Node1/geth.ipc" console 2>.\console1.log***

***geth --identity "Node2" --networkid 42 --datadir "./Node2" --nodiscover --rpc --rpcport "8043" --port "30304" --unlock 0 --password ./pass.sec --ipcpath "./Node2/geth.ipc" console 2>.\console2.log***

* + Make sure the nodes have coinbase account and some ETH balance with them.
  + ***admin.peers* 🡪** Show if and peers has been added
  + ***admin.nodeInfo.enode*** 🡪 Get the enode detail of the node.
  + ***admin.addPeer("enode")*** 🡪 To add another node as peer
  + ***eth.blockNumber*** 🡪 if connected show same numbers in both nodes
  + ***eth.getBlock(eth.blockNumber).miner*** 🡪 show the miner of the particular block
  + ***eth.getBalance(<addr>)*** 🡪 will show the ETH for a particular account
  + We can also use static method to add peers.
    - Create a 'static-nodes.json' file listing both enode and place them in both node folders

File content as example :-> <https://github.com/SSPgg/BlockchainWork/blob/master/static-nodes.json>

Replace [::] with the IP address. To get local IP address do ipconfig in DOS prompt.

* + - Run the 2 nodes one after another to be added as peer.
  + Send some Ether from one acct to another acct present in other node.
    - Check Balance and unlock accts.
    - ***web3.fromWei(eth.getBalance(eth.accounts[1]))*** 🡪 Get balance in for of Ether and not Wei.
    - ***eth.sendTransaction({from: eth.coinbase, to: "hard acct",value:web3.toWei(10,"ether")})***
    - Mine from any node to confirm the transaction.
    - ***eth.sendTransaction({from: eth.accounts[1], to: "hard acct",value:web3.toWei(3,"ether")})***