Blockchain & Αποκεντρωμένες Εφαρμογές

2η Εργαστηριακή Δραστηριότητα

Μαυρίδης Ιωάννης, Φουληράς Παναγιώτης
Μάστορας Θεόδωρος
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

MULTIMEDIA SECURITY NETWORKING

Α. Τύποι και δομές δεδομένων της Solidity.
Τροποποιητές πρόσβασης, Public, Private κ.λπ.

Συνήθεις τύποι μεταβλητών

uint: **unsigned** int of 256 bits
int: integer **positive** and **negative** value accepted 256 bits
string: string of **characters**
bool: supports logic **true** and **false** value

Πολλαπλή ανάθεση
(age, gender) = getAgeGender();

```
contract HelloWorld {
    uint x;
    int y;
    bool z;
    address a; //20 bytes = 160bits  default 0x0000 ή address(0)

    uint256 b; //32 bytes
    uint128 c; //16 bytes
    uint64 d; //8 bytes
    uint32 e; //4 bytes
    uint16 f; //2 bytes
    uint8 g; //1 byte
    uint h; // alias for uint256
}
```

“implicit convertion” Δεν λειτουργεί από μεγάλα σε μικρά και από προσημασμένα σε απρόσημα. Ακόμη κι όταν λειτουργεί είναι πιθανά runtime errors

Τελεστές

```
+
-
/
*
%

&&
||
!
```

=
+=
-=
*=
/=
%=
**
++
--

mapping (hash table)

```
mapping (uint => string) phoneToName;

struct customer {
    uint idNum;
    string name;
    uint bidAmount;
}
mapping (address => customer) custData;
```

ΔΕΝ επιτρέπεται η χρήση mapping ως key (ούτε πίνακα... είναι reference)... (αλλά επιτρέπεται ως value)

ΔΕΝ επιτρέπεται αρχικοποίηση στη γραμμή της δήλωσης, αλλά το δημιουργείς άδειο και προσθέτεις ΜΕΛΗ στην πορεία

Στην πράξη όλα τα mappings είναι δηλωμένα **σφαιρικά**

Προσοχή!!! αν καλέσεις την get για key που δεν υπάρχει επιστρέφεται η default τιμή του τύπου της value (εδώ π.χ. θα δώσει false) Άρα ΔΕΝ ΜΠΟΡΟΥΜΕ ΠΡΑΓΜΑΤΙΚΑ να γνωρίζουμε ΑΝ κάποιο key υπάρχει. Για αυτό χρειάζεται έξτρα μεταβλητή (π.χ. πίνακας) εκτός κι αν γνωρίζουμε πως ποτέ η τιμή ΔΕΝ ΘΑ είναι η default.

```
contract HelloWorld {
    mapping(uint => int) map1;

    function x() public {
        mapping(uint => int) map = map1;
        map[0] = 8;
        map[1] = 5;
    }

    function getItem(uint key) public view returns (int) {
        return map1[key];
```

```
    }
}
```

mapping **ως παράμετρος** μιας συνάρτησης ΔΕΝ ΕΠΙΤΡΕΠΕΤΑΙ εκτός κι αν οριστεί ως **storage ΚΑΙ** η συνάρτηση ΔΕΝ είναι public αλλά αυστηρά internal καλείται ΜΟΝΟ από το ίδιο το συμβόλαιο ή απογόνους του

```
function x(mapping(uint => uint) storage map) internal {
    map[0] = 8;
}
```

Δήλωση συνάρτησης 1 / 2

```
function name(<parameters>)
    visibilityModifier
    accessModifiers
    modificationStatus
    returns (<returnParameters>)
```

Δήλωση συνάρτησης 2 / 2

```
function winningProposal()
  public
  validStage(Stage.Done)
  view
  returns (uint8 _winningProposal) {
  …
}
```

Είδη συναρτήσεων

- Constructor
- Fallback
- View
- Pure
- Public
- Private
- Internal
- External

```
contract HelloWorld {
    uint internal x = 0; // είναι ίδιο με uint x = 0;
    uint private y = 0; // δεν επιτρέπεται πρόσβαση σε απογόνους
    uint public z = 0; // δημιουργεί αυτόματα getter

    function x() external returns (uint) { // Μόνο από άλλο contract (function)
        return 1;
    }
}
```

Όταν επιστρέφει πολλαπλές τιμές χρειάζεται παρένθεση (και όλη η εντολή σε μία γραμμή αντίθετα από python)

```
function x() public returns (uint, uint) {
    return (1, 1);
}
```

Οι view δεν τροποποιούν τη storage. Μόνο μια view επιστρέφει τιμές ορατές στο Remix. Μια view επιτρέπεται να καλεί μόνο view ή pure.

```
function x() public view returns (uint) {
    return a;
}
```

Η pure δεν προσπελάζει ΚΑΝ τη storage ΚΑΙ καλεί ΜΟΝΟ pure. Οι pure δεν δημιουργούν συμβόλαια, δεν πυροδοτούν συμβάντα, δεν καλούν την selfdestruct, δεν καλούν low level functions, δεν στέλνουν λεφτά, δεν είναι payable.

Κλάση address

<address>.balance

(uint256) το υπόλοιπο της διεύθυνσης σε Wei

<address>.transfer(uint256 amount)

μεταφέρει στη διεύθυνση το ποσό που δίνεται ως παράμετρος (σε Wei)

Το αντικείμενο msg

```
address adr = msg.sender
uint amt = msg.value
```

Σφαιρικές μεταβλητές: this, msg, block, tx, gasleft()

address(this).balance τα λεφτά του συμβολαίου

msg.sender η διεύθυνση πορτοφολιού αυτού που καλεί το συμβόλαιο

msg.data όλα τα data των ορισμάτων της κλήσης

msg.sig τα τέσσερα πρώτα bytes των data (τα πρώτα γράμματα του ονόματος της συνάρτησης που καλείται)

msg.value τα λεφτά που μεταφέρει στο συμβόλαιο αυτός που το καλεί

block.number αριθμός του block στο οποίο έχει δημοσιευτεί και εκτελείται το τρέχον συμβόλαιο.

block.chainid το chain ID της blockchain

block.gaslimit το ανώτατο όριο gas που μπορεί να καταναλωθεί στο block (από όλες τις συναλλαγές σ' αυτό)

block.difficulty η δυσκολία του proof-of-work που απαιτείται για τη δημιουργία του τρέχοντος block.

block.timestamp η χρονική στιγμή κατά την οποία δημιουργήθηκε το block σε δευτερόλεπτα από την 1η Ιανουαρίου 1970 (UNIX timestamp)

block.coinbase η διεύθυνση του πορτοφολιού του miner που εξόρυξε (και αμείφθηκε για) το block του συμβολαίου

tx.origin η διεύθυνση του αρχικού αποστολέα της μεταφοράς χρημάτων - η χρήση της αποφεύγεται επειδή μπορεί να επιφυλάσσει προβλήματα ασφαλείας

Αποστολή και λήψη χρημάτων

Special συναρτήσεις... **χωρίς τη λέξη function**... **receive()** ... εκτελείται όποτε κάποιος στέλνει λεφτά στο συμβόλαιο!!! Λειτουργεί σαν συμβάν!!! Όταν δεν υπάρχει msg.data (όχι κλήση) ή όταν δεν καλείται συνάρτηση payable ΔΕΝ μπορεί το συμβόλαιο να δεχτεί λεφτά. Για να τα δεχτεί πυροδοτεί την receive() η οποία είναι πάντα payable. Επίσης, μετά από κάθε κλήση που αποδίδει λεφτά σε κάποια payable πυροδοτείται και πάλι η receive.

Αν δεν υπάρχει receive() ούτε και κλήση σε payable τότε πυροδοτείται η **fallback()** (επίσης payable) η οποία κανονικά πρέπει να κάνει revert την μεταφορά και να την ακυρώσει (δεν είναι πάντα δυνατό π.χ. μετά από selfdestruct).

```
contract PaymentContract {
    // Η receive καλείται όταν το συμβόλαιο λαμβάνει χρήματα
    // χωρίς να καλείται μια συγκεκριμένη συνάρτηση.
    receive() external payable {
        // Κάτι που μπορείτε να κάνετε κατά την λήψη χρημάτων
        received += msg.value;
    }

    // Η fallback καλείται όταν η συνάρτηση που κλήθηκε δεν ήταν payable
    // ή δεν υπάρχει η συνάρτηση ή η receive.
    fallback() external payable {
       Returned += msg.value;
    }
}
```

Η call είναι καλύτερη, γιατί οι άλλες έχουν μόνο 2300 gas.

Διάφοροι τρόποι πληρωμής:

```
bool done = payable(msg.sender).send(address(this).balance)
payable(msg.sender).transfer(address(this).balance)
payable(msg.sender).call{value: address(this).balance}("");
(bool done, bytes memory data) = payable(msg.sender).call{value: address(this).balance}("");
(bool done,) = payable(msg.sender).call{value: address(this).balance}("");
```

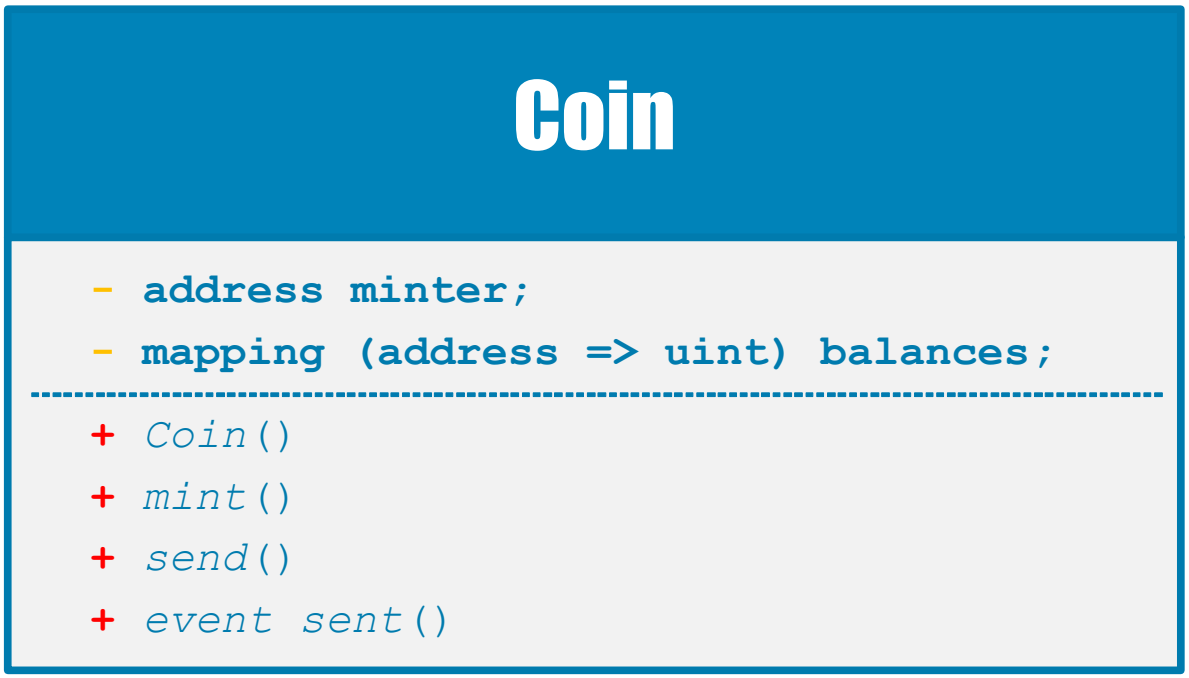
Αν δεν φτάνουν τα λεφτά η call επιστρέφει false

```
(bool done,) = payable(msg.sender).call{value: 1 ether}("");
```

Κληρονομικότητα

```
contract StandardPolicies {...}
contract NYPolicies is StandardPolicies {
    // plus other policies...
}
```

Β. Συμβάντα στο παράδειγμα minter.sol.



1. Επιστρέψτε στον "File explorer" του Remix και δημιουργήστε ένα νέο αρχείο solidity με το όνομα "Minter.sol".

Πληκτρολογήστε ή αντιγράψτε το παρακάτω πρόγραμμα Solidity:

link

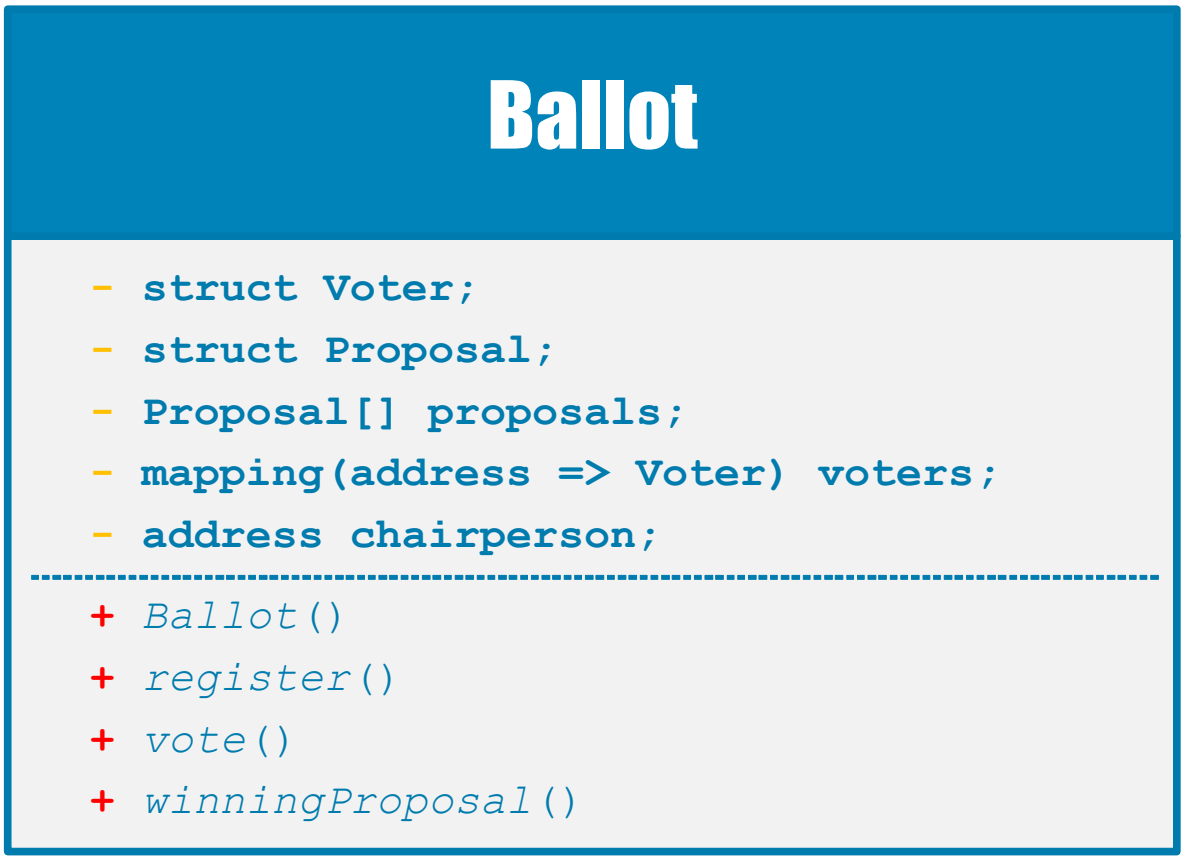
```
pragma solidity >=0.5.9;

contract Coin {
    // The keyword "public" makes those variables
    // readable from outside.
    address public minter;
    mapping (address => uint) public balances;

    // Events allow light clients to react on
    // changes efficiently.
    event Sent(address from, address to, uint amount);

    // This is the constructor whose code is
    // run only when the contract is created.
    constructor() {
        minter = msg.sender;
    }

    function mint(address receiver, uint amount) public  {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }

    function send(address receiver, uint amount) public {
        if (balances[msg.sender] < amount) return;
```

```
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        emit Sent(msg.sender, receiver, amount);
    }
}
```

Ένα υπεραπλουστευμένο token. Ο ιδιοκτήτης του συμβολαίου **δημιουργεί** νομίσματα (tokens) τα οποία «χαρίζει» **αυθαίρετα** σε όποιον θέλει… όποτε θέλει… καλώντας τη μέθοδο **mint**. Όποιος έχει νομίσματα (tokens), τα στέλνει όποτε και όπου θέλει καλώντας τη μέθοδο **send**. Το συμβόλαιο κρατάει στο mapping **balances ΟΛΕΣ** τις πληροφορίες ιδιοκτησίας νομισμάτων. Κάθε φορά που αποστέλλονται νομίσματα πυροδοτείται (καταγράφεται) το συμβάν **Sent**.

2. Πειραματιστείτε με το συμβόλαιο αφού το μεταγλωττίσετε και το δημοσιεύσετε στην τοπική “Remix VM (Cancun)”.

C. Χρονικά στάδια και modifiers στο παράδειγμα Ballot.sol.



1. Επιστρέψτε στον "File explorer" του Remix και δημιουργήστε ένα νέο αρχείο solidity με το όνομα "Ballot.sol".

link

```
pragma solidity >=0.5.9;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
       // address delegate;
    }
    struct Proposal {
        uint voteCount; // could add other data about proposal
    }

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;


    /// Create a new ballot with $(_numProposals) different proposals.
    constructor(uint8 _numProposals) {
        chairperson = msg.sender;
        voters[chairperson].weight = 2;
```

```
        proposals.length = _numProposals;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function register(address toVoter) public {
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
        voters[toVoter].voted = false;
    }

    /// 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 winningProposal() public view 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;
            }
    }
}
```

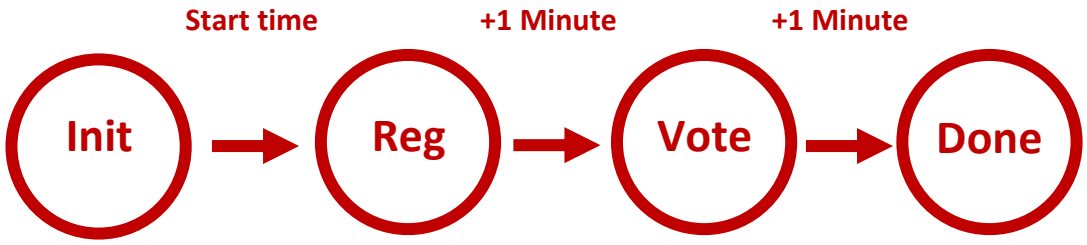
Ο δημιουργός του συμβολαίου είναι ο πρόεδρος ενός συμβουλίου το οποίο πρέπει να ψηφίσει μια πρόταση. Η ψήφος του προέδρου έχει διπλάσια βαρύτητα. Τα μέλη δηλώνονται/εγγράφονται από τον πρόεδρο (μόνο όσοι είναι καταχωρισμένοι έχουν δικαίωμα ψήφου). Επιτρέπεται να ψηφίσει κάποιον μόνο μία φορά. (*Στην παραπάνω **αρχική** εκδοχή υπάρχουν πολλά λειτουργικά κενά*)

2. Πειραματιστείτε με το συμβόλαιο αφού το μεταγλωττίσετε και το δημοσιεύσετε στην τοπική “ Remix VM (Cancun)”.

---

3. Time – Μετάβαση καταστάσεων

```
if (now >= creationTime + 1 day) stage = Stage.RegDone;

if (now <= voteStartTime + 60 minutes) ...να επιτρέπεται η ψηφοφορία
```




```
pragma solidity ^0.5.9;
contract StateTransV2 {

    enum Stage {Init, Reg, Vote, Done}
    Stage public stage;
    uint startTime;
    uint public timeNow;

    constructor() public {
        stage = Stage.Init;
        startTime = now;
    }

    //Assuming the Stage change has to be enacted APPROX every 1 mintute
    //timeNow variable is defined for underatanding the process, you can simply use
    //"now" Solidity defined variable
    //Of course, time duration for the Stages may depend on your application
    //1 minutes is set to illustrate the working
    function advanceState () public  {
        timeNow = now;
        if (timeNow > (startTime + 1 minutes)) {
        startTime = timeNow;
        if (stage == Stage.Init) {stage = Stage.Reg; return;}
        if (stage == Stage.Reg) {stage = Stage.Vote; return;}
        if (stage == Stage.Vote) {stage = Stage.Done; return;}
        return;
        }
    }
}
```

4. To Ballot με stages

link

```
pragma solidity ^0.5.9;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
    }
    struct Proposal {
        uint voteCount;
    }
    enum Stage {Init, Reg, Vote, Done}
    Stage public stage = Stage.Init;

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;


    uint startTime;

    /// Create a new ballot with $(_numProposals) different proposals.
    constructor(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 2; // weight is 2 for testing purposes
        proposals.length = _numProposals;
        stage = Stage.Reg;
        startTime = now;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function register(address toVoter) public {
        if (stage != Stage.Reg) {return;}
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
        voters[toVoter].voted = false;
        if (now > (startTime + 10 seconds)) {stage = Stage.Vote; startTime = now;}
    }

    /// Give a single vote to proposal $(toProposal).
    function vote(uint8 toProposal) public {
        if (stage != Stage.Vote) {return;}
        Voter storage sender = voters[msg.sender];
        if (sender.voted || toProposal >= proposals.length) return;
        sender.voted = true;
        sender.vote = toProposal;
```

```
        proposals[toProposal].voteCount += sender.weight;
        if (now > (startTime+ 10 seconds)) {stage = Stage.Done;}

    }

    function winningProposal() public view returns (uint8 _winningProposal) {
        if (stage != Stage.Done) {return _winningProposal;}
        uint256 winningVoteCount = 0;
        for (uint8 prop = 0; prop < proposals.length; prop++)
            if (proposals[prop].voteCount > winningVoteCount) {
                winningVoteCount = proposals[prop].voteCount;
                _winningProposal = prop;
            }

    }
}
```

Modifiers – require / revert

Ποιος καλεί

Ποια συνάρτηση

Πότε κ.λπ.

Παράδειγμα - Χωρίς modifier:

```
function pickWinner() public {
    require(msg.sender == manager);

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

Παράδειγμα - Με 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);
    _;
}
```

Revert & Error

link

```
// Errors allow you to provide information about
// why an operation failed. They are returned
// to the caller of the function.
error InsufficientBalance(uint requested, uint available);

// Sends an amount of existing coins
// from any caller to an address
function send(address receiver, uint amount) public {
    if (amount > balances[msg.sender])
         revert InsufficientBalance({
            requested: amount,
            available: balances[msg.sender]
        });

    balances[msg.sender] -= amount;
    balances[receiver] += amount;
    emit Sent(msg.sender, receiver, amount);
}
```

5. To Ballot με modifiers

link

```
pragma solidity ^0.5.9;
contract Ballot {

    struct Voter {
        uint weight;
        bool voted;
        uint8 vote;
        //address delegate;
    }
    struct Proposal {
        uint voteCount;
    }
    enum Stage {Init, Reg, Vote, Done}
    Stage public stage = Stage.Init;

    address chairperson;
    mapping(address => Voter) voters;
    Proposal[] proposals;

    event votingCompleted();

    uint startTime;
    //modifiers
    modifier validStage(Stage reqStage)
    { require(stage == reqStage);
      _;
    }


    /// Create a new ballot with $(_numProposals) different proposals.
    constructor(uint8 _numProposals) public  {
        chairperson = msg.sender;
        voters[chairperson].weight = 2; // weight is 2 for testing purposes
        proposals.length = _numProposals;
        stage = Stage.Reg;
        startTime = now;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function register(address toVoter) public validStage(Stage.Reg) {
        //if (stage != Stage.Reg) {return;}
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
        voters[toVoter].voted = false;
        if (now > (startTime+ 30 seconds)) {stage = Stage.Vote; }
    }
```

```
    /// Give a single vote to proposal $(toProposal).
    function vote(uint8 toProposal) public validStage(Stage.Vote)  {
       // if (stage != Stage.Vote) {return;}
        Voter storage sender = voters[msg.sender];
        if (sender.voted || toProposal >= proposals.length) return;
        sender.voted = true;
        sender.vote = toProposal;
        proposals[toProposal].voteCount += sender.weight;
        if (now > (startTime+ 30 seconds)) {stage = Stage.Done; emit votingCompleted();}

    }

    function winningProposal() public validStage(Stage.Done) view
                                            returns (uint8 _winningProposal) {
       //if(stage != Stage.Done) {return;}
        uint256 winningVoteCount = 0;
        for (uint8 prop = 0; prop < proposals.length; prop++)
            if (proposals[prop].voteCount > winningVoteCount) {
                winningVoteCount = proposals[prop].voteCount;
                _winningProposal = prop;
            }
       assert (winningVoteCount > 0);

    }
}
```