



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ**

---

**UNIVERSITY OF PIRAEUS**

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ  
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ  
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΛΙΚΗ ΕΡΓΑΣΙΑ ΜΑΘΗΜΑΤΟΣ  
<<ΤΕΧΝΟΛΟΓΙΕΣ BLOCKCHAIN ΚΑΙ ΕΦΑΡΜΟΓΕΣ>>

ΓΚΟΛΕΜΙ ΚΡΙΣΤΙΑΝ, Π18029

Για την εκπόνηση της εργασίας σύμφωνα με τις απαιτήσεις που περιγράφονται στην εκφώνηση, χρησιμοποιήθηκε η έκδοση 0.8.13 της γλώσσας προγραμματισμού solidity, στο προγραμματιστικό περιβάλλον Remix IDE ώστε να μπορούμε εύκολα να κάνουμε deploy και test το smart contract που υλοποιήθηκε. Για την ικανοποίηση όλων των απαιτήσεων δημιουργήθηκε ένα smart contract στο αρχείο Final\_p18029.sol. Παρακάτω, αναλύονται με βάση της απαιτήσεις της εργασίας οι συναρτήσεις που αναπτύχθηκαν και το γενικότερο σκεπτικό πίσω από αυτές.

2. Για το εν λόγω συμβόλαιο θα πρέπει να ορισθούν τρεις συγκεκριμένοι λογαριασμοί τύπου *Externally Owned Account (EOA)* μέσω των οποίων θα γίνεται η διαχείριση των χρηματικών ποσών που καταβάλλονται στο έξυπνο συμβόλαιο. Συγκεκριμένα, για να μεταβιβαστεί κάποιο ποσό, από το συνολικά διαθέσιμο του έξυπνου συμβολαίου, σε έναν λογαριασμό *Ethereum* θα πρέπει η συναλλαγή αυτή να εγκριθεί από δύο εκ των τριών προαναφερόμενων διαχειριστικών λογαριασμών.

Ορίζονται state variables για τον κάθε διαχειριστή του ηλεκτρονικού καταστήματος ως payable address και για κάθε έναν ορίζεται μεταβλητή τύπου boolean που αναπαριστά το αν αποδέχεται κάποια συναλλαγή ή όχι. Οι μεταβλητές αυτές αρχικοποιούνται ως false και μπορούν να μεταβληθούν σε true από τον κάθε διαχειριστή καλώντας την αντίστοιχη συνάρτηση admin\_acceptTransaction().

```
address payable admin1;
address payable admin2;
address payable admin3;
bool public admin1_acceptsTransaction = false;
bool public admin2_acceptsTransaction = false;
bool public admin3_acceptsTransaction = false;
```

```
// purchase acceptance functions for each administrator
function admin1_acceptTransaction() external {
    require(msg.sender == admin1, "Only admin1 can do that!");
    admin1_acceptsTransaction = true;
}

function admin2_acceptTransaction() external {
    require(msg.sender == admin2, "Only admin2 can do that!");
    admin2_acceptsTransaction = true;
}

function admin3_acceptTransaction() external {
    require(msg.sender == admin3, "Only admin3 can do that!");
    admin3_acceptsTransaction = true;
}
```

Σε περίπτωση που ο πελάτης του καταστήματος επιθυμεί αποζημίωση αφού έχει ολοκληρωθεί η συναλλαγή του, έχει δημιουργηθεί συνάρτηση που μπορεί να καλέσει μόνο κάποιος διαχειριστής ώστε να του κάνει μια μεταφορά token από το αποθεματικό του συμβολαίου. Όπως ζητείται από την εκφώνηση, λοιπόν, για να γίνει αυτό, όπως φαίνεται από το παρακάτω στιγμιότυπο, θα πρέπει τουλάχιστον δύο διαχειριστές να δέχονται τη συναλλαγή αυτή.

```
/*
 * function for the admins to return money to the client,
 * if the client asks to cancel the purchase after the payment has been completed
 */
function returnMoneyAfterCompleted(address payable client, uint tokensToReturn) external onlyAdmin {
    require(balances[address(this)] > tokensToReturn, "Not enough tokens to return to the client");

    if (msg.sender == admin1 && (admin2_acceptsTransaction == true || admin3_acceptsTransaction == true)) {
        balances[address(this)] -= tokensToReturn;
        balances[client] += tokensToReturn;
    }
    else if (msg.sender == admin2 && (admin1_acceptsTransaction == true || admin3_acceptsTransaction == true)) {
        balances[address(this)] -= tokensToReturn;
        balances[client] += tokensToReturn;
    }
    else if (msg.sender == admin3 && (admin1_acceptsTransaction == true || admin2_acceptsTransaction == true)) {
        balances[address(this)] -= tokensToReturn;
        balances[client] += tokensToReturn;
    }

    // reset the variables after the transaction is completed
    admin1_acceptsTransaction = false;
    admin2_acceptsTransaction = false;
    admin3_acceptsTransaction = false;
}
```

Παρακάτω φαίνεται η διαδικασία που αναλύθηκε παραπάνω. Ο admin3 προσπαθεί να εκτελέσει μια μεταφορά χρημάτων σε μια συγκεκριμένη διεύθυνση. Πριν ο admin3 καλέσει τη συνάρτηση `returnMoneyAfterCompleted`, καλούμε από τον λογαριασμό του admin1 τη συνάρτηση `admin1AcceptTransaction` ώστε να μεταφέρει πώς αποδέχεται τη συναλλαγή. Άρα, `admin1_acceptsTransaction = true`, ενώ `admin2_acceptsTransaction = false` και εννοείται πώς και ο admin3 δέχεται τη συναλλαγή αφού ο ίδιος την εκτελεί. Επομένως, αφού τουλάχιστον δύο από τους τρεις διαχειριστές δέχονται τη συναλλαγή, αυτή ολοκληρώνεται.

### returnMoneyAfterCompleted

client: "0x78731D3Ca6b7E34aC0F8"

tokensToReturn: "100"

transact

admin1\_accep...

admin1\_accep...

0: bool: true

admin2\_accep...

admin2\_accep...

0: bool: false



```
[vm] from: 0x4B2...C02db to: Final_p18029.returnMoneyAfterCompleted(address,uint256) 0x7EF...8CB47 value: 0 wei  
data: 0xe03...00064 logs: 0 hash: 0xec3...3d680
```

Βλέπουμε πώς τα tokens μεταφέρθηκαν επιτυχώς αφού το balance του συγκεκριμένου λογαριασμού αυξήθηκε κατά 100 tokens.

### balanceOf

tokenOwner: "0x78731D3Ca6b7E34aC0F8"

call

0: uint256: 0

### balanceOf

tokenOwner: "0x78731D3Ca6b7E34aC0F82"

call

0: uint256: 100

3. Κάθε ένας από τους διαχειριστικούς λογαριασμούς θα πρέπει να είναι σε θέση να αντικαθιστά τον λογαριασμό του με κάποιο άλλον.
4. Θα επιτρέπεται με την έγκριση δύο εκ των τριών διαχειριστικών λογαριασμών να αλλάξει ο έτερος διαχειριστικός λογαριασμός.

Με παρόμοιο σκεπτικό όπως και το προηγούμενο ερώτημα, ορίζονται μεταβλητές που αναπαριστούν το αν ένας διαχειριστής δέχεται ένας έτερος διαχειριστής να αλλάξει τη διεύθυνση του ή όχι. Οποιοσδήποτε από τους διαχειριστές καλώντας την αντίστοιχη συνάρτηση `admin_acceptAddressChange`, μπορεί να μεταφέρει πώς αποδέχεται αλλαγή διεύθυνσης από έτερο διαχειριστή. Τελικά, έχει δημιουργηθεί η συνάρτηση `changeAddress` την οποία μπορεί να καλέσει κάποιος `admin` και να αλλάξει τη διεύθυνση του δίνοντας μια καινούργια ως όρισμα σε περίπτωση που και οι δύο έτεροι το αποδέχονται.

```
bool public admin1_acceptsAddressChange = false;
bool public admin2_acceptsAddressChange = false;
bool public admin3_acceptsAddressChange = false;
```

```
// address change approval function for all admins
function admin1_acceptAddressChange() external inState(State.Initial) {
    require(msg.sender == admin1, "Only admin1 can do that!");
    admin1_acceptsAddressChange = true;
}

function admin2_acceptAddressChange() external inState(State.Initial) {
    require(msg.sender == admin2, "Only admin2 can do that!");
    admin2_acceptsAddressChange = true;
}

function admin3_acceptAddressChange() external inState(State.Initial) {
    require(msg.sender == admin3, "Only admin3 can do that!");
    admin3_acceptsAddressChange = true;
}
```

```
// each admin can change his address
function changeAddress(address new_address) external inState(State.Initial) onlyAdmin {
    if (msg.sender == admin1 && (admin2_acceptsAddressChange == true && admin3_acceptsAddressChange == true)) {
        admin1 = payable(new_address);
    }
    else if (msg.sender == admin2 && (admin1_acceptsAddressChange == true && admin3_acceptsAddressChange == true)) {
        admin2 = payable(new_address);
    }
    else if (msg.sender == admin3 && (admin1_acceptsAddressChange == true && admin2_acceptsAddressChange == true)) {
        admin3 = payable(new_address);
    }
}
```

Παρακάτω δείχνουμε ένα παράδειγμα της διαδικασίας που αναλύσαμε, όπου ο admin3 προσπαθεί να αλλάξει τη διεύθυνση του.


Αρχικά:

```
getAdminAdd...  
  
0: address: admin_1 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4  
1: address: admin_2 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2  
2: address: admin_3 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db
```

Κλήση changeAddress από τον admin3 αφού πρώτα καλέσουμε την admin\_acceptAddressChange και από τους δύο άλλους λογαριασμούς, ώστε να γίνουν true οι αντίστοιχες μεταβλητές που δείχνουν πώς αποδέχονται την αλλαγή αυτή.

**changeAddress**

new\_address: "0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c"

 **transact**

Τελικά βλέπουμε πώς η διεύθυνση του admin3 έχει αλλάξει:

```
getAdminAdd...  
  
0: address: admin_1 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4  
1: address: admin_2 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2  
2: address: admin_3 0xCA35b7d915458EF540aDe6068dFe2F44E8fa733c
```

Στο σημείο αυτό, πριν προχωρήσουμε στην επεξήγηση των προηγούμενων ερωτημάτων, είναι σημαντικό να αναφέρουμε πώς οι διευθύνσεις των admin2, admin3 ορίζονται από τον constructor κατά το deployment του συμβολαίου, ενώ ως admin1 ορίζεται ο deployer του συμβολαίου.

```
// constructor is called when the smart contract is deployed  
constructor() payable {  
    balances[address(this)] = totalSupply_;  
  
    // the admin addresses are defined when the contract is deployed,  
    // however, they can change them later if they wish  
    // admin1 is the deployer of the contract  
    admin1 = payable(msg.sender);  
    admin2 = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);  
    admin3 = payable(0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db);  
}
```

5. Κατά την πληρωμή μιας παραγγελίας θα πρέπει να ορίζεται το αναγνωριστικό της παραγγελίας. Σημειώνεται ότι το αναγνωριστικό της παραγγελίας γνωστοποιείται στον χρήστη από το ηλεκτρονικό κατάστημα μετά την υποβολή της παραγγελίας. Συνεπώς, η δημιουργία του δεν αφορά το έξυπνο συμβόλαιο.
6. Οι πληρωμές τίθενται αρχικά σε μια κατάσταση «αναμονής αποδοχής» από το ηλεκτρονικό κατάστημα και το χρηματικό ποσό που καταβάλλεται μέσω αυτών δεν προστίθενται ακόμα διαθέσιμο χρηματικό ποσό του συμβολαίου.

Για την αναπαράσταση των σταδίων μιας πληρωμής δημιουργήθηκε το enum State που έχει τα στάδια Initial και Pending. Αφού γίνει αίτημα πληρωμής από τον πελάτη, η πληρωμή τίθεται σε κατάσταση Pending χωρίς το χρηματικό ποσό να προστίθεται ακόμα στο διαθέσιμο ποσό του συμβολαίου.

```
// represents the orders state
enum State {Initial, Pending}
State public state;
```

Για την πραγματοποίηση πληρωμών από τους πελάτες του καταστήματος έχει υλοποιηθεί η συνάρτηση `pay(string memory order_id, uint tokensToSend)`. Ο πελάτης, δίνοντας το αναγνωριστικό της παραγγελίας που ήδη γνωρίζει μετά την πραγματοποίηση της παραγγελίας καθώς και τον αριθμό των tokens που στέλνει μπορεί να κάνει ένα αίτημα για πληρωμή. Το αίτημα αυτό στη συνέχεια μπορούν να το δουν οι διαχειριστές και είτε να αποδεχτούν, είτε να απορρίψουν την πληρωμή. Είναι προφανές, πώς για να εκτελεστεί η συγκεκριμένη συνάρτηση, γίνεται έλεγχος για το εάν ο πελάτης έχει αρκετά νομίσματα στην κατοχή του για την πληρωμή.

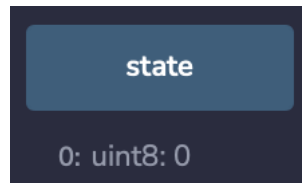
```
// allow the buyer to send money for his purchase, using his order identifier
function pay(string memory order_id, uint tokensToSend) external inState(State.Initial) {
    state = State.Pending; // change the state to pending after the buyer pays for his purchase
    buyer = payable(msg.sender); // buyer is the user who invoked this function
    require(balances[buyer] >= tokensToSend, "You don't have enough tokens to complete this purchase!");

    balances[buyer] -= tokensToSend;

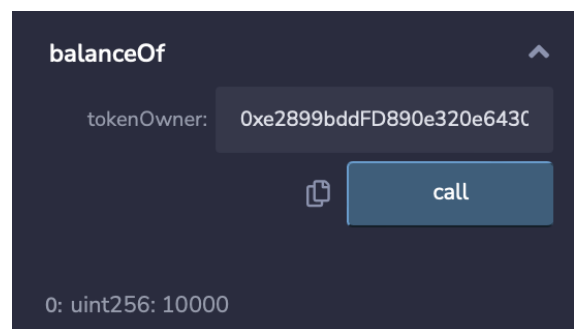
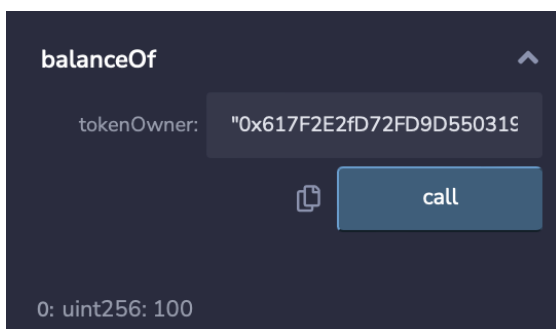
    orderId = order_id;
    amount_sent = tokensToSend;
}
```

Παρακάτω φαίνεται ένα παράδειγμα εκτέλεσης της συγκεκριμένης λειτουργίας του smart contract.

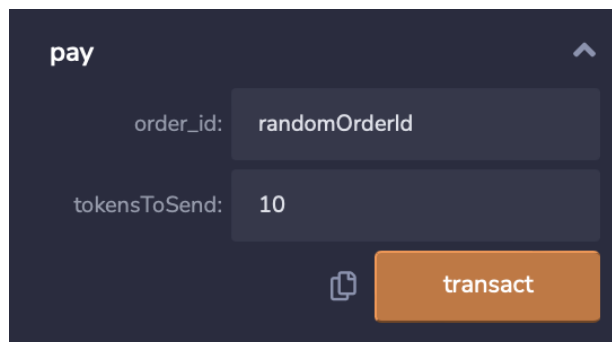
Αρχικά state = 0, δηλαδή Initial.



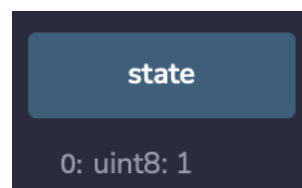
Αφού ο πελάτης αγοράσει κάποια νομίσματα με eth, έχει 100 tokens στη κατοχή του, ενώ το συμβόλαιο έχει 10000 tokens, όσο είναι και το total supply που έχουμε ορίσει.



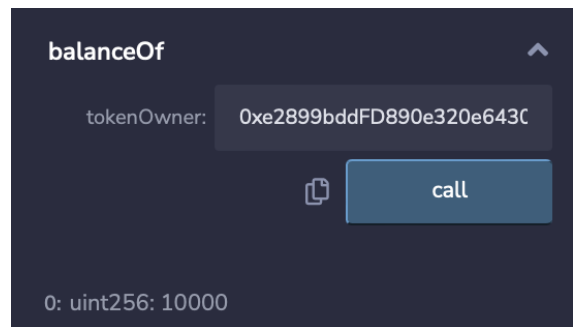
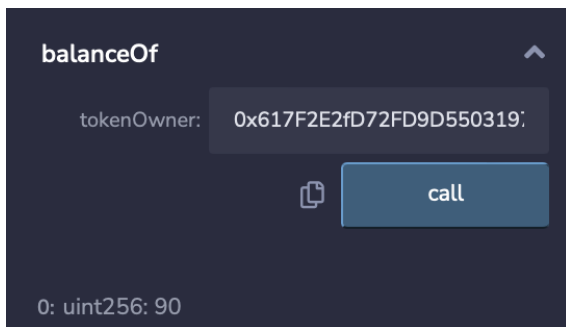
Στη συνέχεια, ο πελάτης καλεί τη συνάρτηση pay χρησιμοποιώντας το αναγνωριστικό της παραγγελίας του και τον αριθμό tokens που στέλνει για την πληρωμή.



Μετά την πληρωμή, δείχνουμε παρακάτω πώς το state μετατράπηκε σε Pending, αφαιρέθηκαν 10 tokens από το λογαριασμό του πελάτη, τα οποία δεν προστέθηκαν στον λογαριασμό του συμβολαίου αφού το balance φαίνεται πώς παραμένει ίδιο, αλλά μπορούν να προστεθούν μετά από επιβεβαίωση από τους διαχειριστές όπως θα δούμε αργότερα.



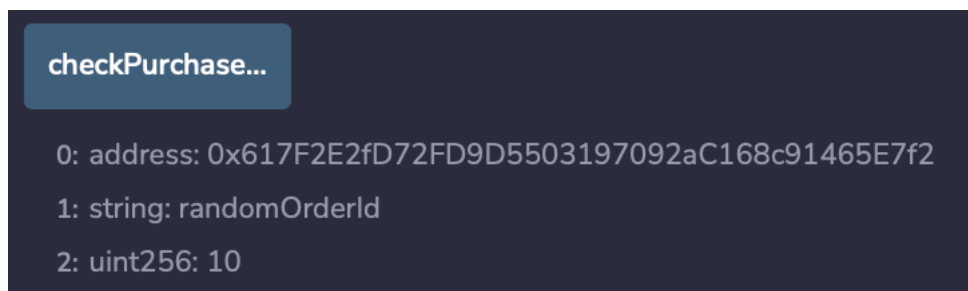




*7. Το ηλεκτρονικό κατάστημα θα ενημερώνει το έξυπνο συμβόλαιο για την αποδοχή ή την απόρριψη, της πληρωμής, μιας παραγγελίας.*

Αφού γίνει το αίτημα πληρωμής από τον πελάτη όπως αναλύθηκε παραπάνω, οι διαχειριστές μπορούν να καλέσουν τη συνάρτηση `checkPurchaseDetails` που έχει υλοποιηθεί ώστε να ελέγξουν τα στοιχεία της πληρωμής. Η συνάρτηση αυτή καθώς και το αποτέλεσμα της κλήσης της από κάποιον καθηγητή με βάση το παράδειγμα που δείξαε προηγουμένως φαίνονται παρακάτω.

```
// the admins can check the details of a purchase so that it can be either accepted or declined
function checkPurchaseDetails() external view inState(State.Pending) onlyAdmin returns (address, string memory, uint) {
    return (buyer, orderId, amount_sent);
}
```



Αφού ελεγχθούν τα στοιχεία της πληρωμής, οποιοσδήποτε από τους διαχειριστές μπορεί να καλέσει είτε τη συνάρτηση `acceptPurchase`, είτε την `declinePurchase` για να αποδεχτεί ή να απορρίψει μια συναλλαγή αντίστοιχα. Σε περίπτωση που η συναλλαγή γίνει αποδεκτή, το ποσό προστίθεται στο διαθέσιμο ποσό του συμβολαίου, ενώ διαφορετικά απορρίπτεται και τα χρήματα επιστρέφονται στον αγοραστή. Παρακάτω φαίνονται οι συναρτήσεις και το αποτέλεσμα της αποδοχής μιας πληρωμής από το κατάστημα που οδηγεί στο να αυξηθεί το `balance` κατά 10 tokens, όσο δηλαδή και το ποσό που πλήρωσε ο πελάτης.

```
// each admin can accept a purchase
function acceptPurchase() external inState(State.Pending) onlyAdmin {
    state = State.Initial;           // change the state to initial after purchase is accepted
    balances[address(this)] += amount_sent; // transfer the tokens to the contracts address

    reset();                         // reset state variables after purchase has been completed
}

// the admins can decline a purchase
function declinePurchase() external inState(State.Pending) onlyAdmin {
    state = State.Initial;           // change the state to initial after purchase is declined by an admin
    balances[buyer] += amount_sent; // buyer gets his money back if the purchase is declined by an admin

    // reset state variables after purchase has been declined
    reset();
}
```

admin1 αποδέχεται την πληρωμή.



[vm] from: 0x5B3...eddC4 to: Final\_p18029.acceptPurchase() 0xe28...4157A value: 0 wei data: 0xca7...e0c6e  
logs: 0 hash: 0x632...5f838

Τα tokens προστίθενται στο συμβόλαιο αφού το balance από 10000 γίνεται 10010.

balanceOf

tokenOwner:

0xe2899bddFD890e320e643044c6b95B9B0b841

call

0: uint256: 10010

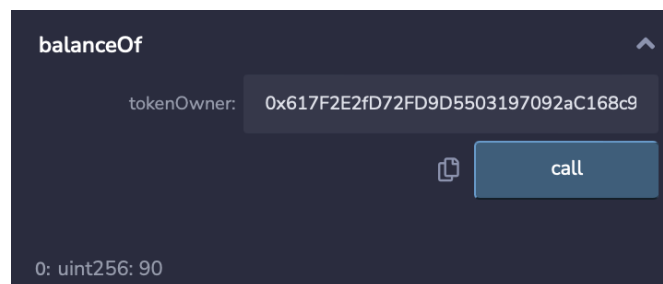
8. Για όσο διάστημα η πληρωμή μιας παραγγελίας βρίσκεται σε κατάσταση «αναμονής αποδοχής», ο πελάτης που πραγματοποίησε την πληρωμή μπορεί να την ακυρώσει και να πάρει τα χρήματα του πίσω. Επίσης, στην περίπτωση που το ηλεκτρονικό κατάστημα απορρίψει την πληρωμή τα χρήματα επιστρέφονται αυτόματα στον πελάτη.

Για όσο η κατάσταση της συναλλαγής είναι σε στάδιο Pending, που σημαίνει δηλαδή ότι δεν έχει ολοκληρωθεί ακόμα, ο πελάτης μπορεί να καλέσει τη συνάρτηση `cancelPurchase`, παίρνοντας τα χρήματά του πίσω. Οποιοσδήποτε διαχειριστής, καλώντας τη `declinePurchase` όπως είδαμε παραπάνω, αντίστοιχα με την `acceptPurchase`, μπορεί να απορρίψει μια πληρωμή και τα χρήματα να επιστραφούν στον πελάτη.

```
// the buyer can cancel his purchase if it's still pending
function cancelPurchase() external inState(State.Pending) onlyBuyer {
    state = State.Initial; // change the state to initial after the buyer cancels his purchase
    balances[buyer] += amount_sent; // buyer gets his tokens back after cancelling the purchase

    // reset state variables after purchase has been canceled
    reset();
}
```

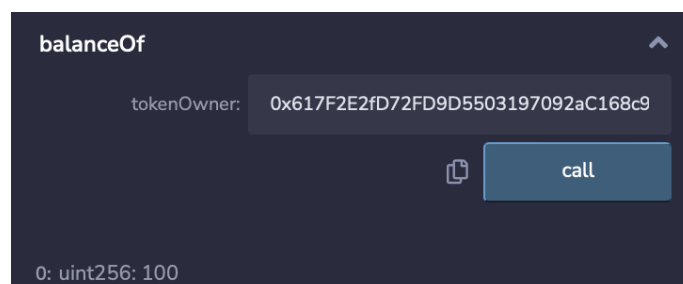
Μετά την κλήση της `pay` για μια πληρωμή των 10 tokens, ο πελάτης έχει στην κατοχή του 90 tokens.



Ο πελάτης ακυρώνει την πληρωμή αφού ακόμη είναι Pending καλώντας τη `cancelPurchase`.

```
✓ [vm] from: 0x617...5E7f2 to: Final_p18029.cancelPurchase() 0x93f...C96CC value: 0 wei data: 0xab4...60a26 logs: 0
hash: 0x8d8...35e93
```

Τα 10 tokens του επιστράφηκαν, αφού ακύρωσε την πληρωμή του και από 90 έχει πλέον 100 ξανά στην κατοχή του.



9. Μετά την αποδοχή της πληρωμής μιας παραγγελίας, το καταβαλλόμενο ποσό προστίθεται στο διαθέσιμο χρηματικό ποσό του συμβολαίου και η πληρωμή δεν μπορεί να ακυρωθεί ή να απορριφθεί μονομερώς από τον πελάτη ή το ηλεκτρονικό κατάστημα αντίστοιχα. Για την αποφυγή σύγχυσης, σημειώνεται ότι σε περίπτωση που ένας πελάτης ζητήσει την ακύρωση της παραγγελίας του μετά την «αποδοχή της πληρωμής», το ηλεκτρονικό κατάστημα μπορεί να επιστρέψει τα χρήματα κάνοντας απλώς μια μεταφορά προς τον πελάτη.

Όπως είδαμε παραπάνω από την ανάλυση του ζητούμενου (7), μετά την αποδοχή της πληρωμής από κάποιον διαχειριστή, το καταβαλλόμενο ποσό προστίθεται στο διαθέσιμο ποσό του συμβολαίου.

Η πληρωμή έπειτα δεν μπορεί να απορριφθεί είτε από τον πελάτη είτε από το κατάστημα, χρησιμοποιώντας το modifier `inState` που έχει υλοποιηθεί. Συγκεκριμένα, αφού ολοκληρωθεί μια πληρωμή μέσω της `acceptPurchase`, το `state` της συναλλαγής επαναφέρεται σε `Initial`. Αυτό έχει ως αποτέλεσμα κανείς να μην μπορεί πλέον να απορρίψει τη συναλλαγή αφού στις συναρτήσεις `declinePurchase`, `cancelPurchase` απαιτείται μέσω του modifier `inState(State.Pending)` στον ορισμό της συνάρτησης, η συναλλαγή να βρίσκεται σε φάση `Pending`, κάτι που μετά την ολοκλήρωση της πληρωμής εξηγήσαμε πώς δεν συμβαίνει. Άρα, με αυτό τον τρόπο επιτυγχάνεται το ζητούμενο της εκφώνησης και κανείς δεν μπορεί να απορρίψει μια συναλλαγή, αφού αυτή ολοκληρωθεί. Παρακάτω φαίνεται ο modifier `inState` που χρησιμοποιούμε για το σκοπό αυτό.

```
modifier inState(State state_) {  
    require(state == state_, "You can't do that in this current state of the order!");  
    _;  
}
```

Για το δεύτερο κομμάτι του ζητούμενου (9), δηλαδή την επιστροφή χρημάτων στον αγοραστή μετά την ολοκλήρωση της πληρωμής, έχουμε ήδη αναλύσει το κομμάτι αυτό στο ζητούμενο (2) που μιλήσαμε για τη συνάρτηση `returnMoneyAfterCompleted`, όπου το κατάστημα μέσω των διαχειριστών μπορεί πολύ απλά να κάνει μια μεταφορά `tokens` στον αγοραστή, υπό την προϋπόθεση ότι τουλάχιστον δύο από τους τρεις διαχειριστές αποδέχονται τη συναλλαγή.

10. Οι τιμές των προϊόντων του καταστήματος θα ορίζονται σε ένα ηλεκτρονικό νόμισμα ιδιοκτησίας του ηλεκτρονικού καταστήματος και οι πληρωμές των πελατών δεν θα γίνονται σε ΕΤΗ αλλά στο εν λόγω νόμισμα. Το νόμισμα αυτό θα πρέπει να κατασκευαστεί βάσει του προτύπου EIP-20 [4] (γνωστό και ως ERC20).

Για την ικανοποίηση του παραπάνω ζητουμένου, στο smart contract που υλοποιήσαμε, δημιουργήσαμε ένα token βάσει του προτύπου ERC20, που περιέχει όλες τις απαραίτητες συναρτήσεις και μεταβλητές, όπως μπορούμε να δούμε στο συγκεκριμένο σύνδεσμο <https://eips.ethereum.org/EIPS/eip-20>. Το token το ονομάζουμε p18029Coin και ορίζουμε ως αρχικό total supply του 10000 tokens, το οποίο όπως θα δούμε στη συνέχεια μπορεί να αυξηθεί καθώς οι χρήστες θα έχουν τη δυνατότητα να κάνουν mint tokens μέσω της buyTokens. Για κάθε τι που έχει υλοποιηθεί για το συγκεκριμένο token υπάρχουν αναλυτικά επεξηγηματικά σχόλια στον παρακάτω κώδικα.

```
event Transfer(address indexed from, address indexed to, uint tokens);
event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
event BuyTokens(address buyer, uint256 ethAmount, uint256 tokenAmount);

string public constant name = "p18029Coin";
string public constant symbol = "PCN";
// initialize the tokens total supply and exchange rate, can be changed later by the admins
uint256 totalSupply_ = 10000;
uint256 tokensPerEth = 100;

mapping(address => uint256) balances; // map an address to its balance
mapping(address => mapping (address => uint256)) allowed; // store the number of tokens a delegate address can withdraw from
another one

// constructor is called when the smart contract is deployed
constructor() payable {
    balances[address(this)] = totalSupply_;

    // the admin addresses are defined when the contract is deployed,
    // however, they can change them later if they wish
    // admin1 is the deployer of the contract
    admin1 = payable(msg.sender);
    admin2 = payable(0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2);
    admin3 = payable(0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db);
}

// return the total supply of tokens of our contract
function totalSupply() public view returns (uint256) {
    return totalSupply_;
}

// return the token balance of a specific account
function balanceOf(address tokenOwner) public view returns (uint) {
    return balances[tokenOwner];
}
```

```

/*
 * receiver is the address of the account that will receive tokens
 * numTokens is the number of tokens that will be sent to the receiver account
 */

```

```

function transfer(address receiver, uint numTokens) public returns (bool) {
    require(balances[msg.sender] >= numTokens);
    balances[msg.sender] -= numTokens;
    balances[receiver] += numTokens;
    emit Transfer(msg.sender, receiver, numTokens);
    return true;
}

```

```

// allows delegate to withdraw from the msg.senders account multiple times, up to the numTokens

```

```

function approve(address delegate, uint numTokens) public returns (bool) {
    allowed[msg.sender][delegate] = numTokens;
    emit Approval(msg.sender, delegate, numTokens);
    return true;
}

```

```

// returns the amount of tokens the delegate is allowed to withdraw from owner

```

```

function allowance(address owner, address delegate) public view returns (uint) {
    return allowed[owner][delegate];
}

```

```

/*
 * transfer tokens from an account to another one
 * owner is the address of the balances from which we will transfer the numTokens
 * buyer is the address in the balances that we will credit the numTokens
 * numTokens is the number of tokens to be transferred from owner to buyer
 */

```

```

function transferFrom(address owner_, address buyer_, uint numTokens) public returns (bool) {
    require(numTokens <= balances[owner_]);
    require(numTokens <= allowed[owner_][msg.sender]); // allowed[owner][msg.sender] = the number of tokens the msg.sender is allowed
to withdraw from the owner

```

```

    balances[owner_] -= numTokens;
    balances[buyer_] += numTokens;
    allowed[owner_][msg.sender] -= numTokens;
    emit Transfer(owner_, buyer_, numTokens);
    return true;
}

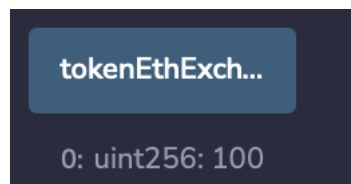
```

11. Το ηλεκτρονικό κατάστημα θα πρέπει να είναι σε θέση να ορίζει, όποτε το επιθυμεί, τη συναλλαγματική αξία του νομίσματος του σε ETH.

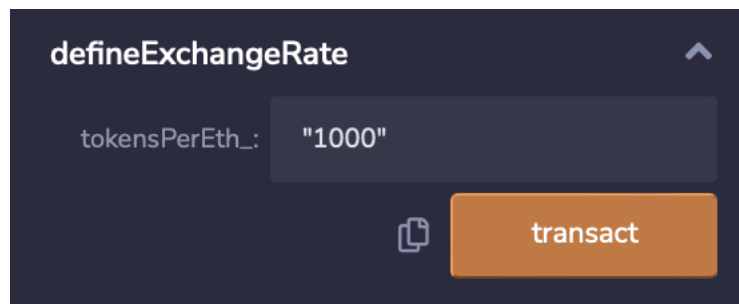
Για τον ορισμό της συναλλαγματικής αξίας του νομίσματος σε σχέση με το eth έχει υλοποιηθεί η συνάρτηση `defineExchangeRate` που μπορεί να κληθεί μόνο από τους διαχειριστές (`modifier onlyBuyer`). Αρχικά, όπως φαίνεται στον παραπάνω κώδικα του token, όταν γίνεται `deploy` το συμβόλαιο ορίζουμε πώς 1 eth = 100 tokens, ωστόσο αυτό μπορεί να αλλάξει μέσω της παρακάτω συνάρτησης από τους διαχειριστές.

```
// the exchange rate between our token and eth can be changed using this function
function defineExchangeRate(uint tokensPerEth_) public onlyAdmin {
    tokensPerEth = tokensPerEth_;
}
```

Αρχικά, 1 eth = 100 tokens.

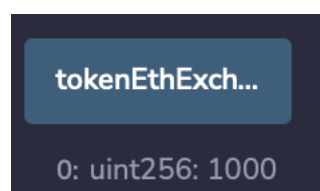


Κλήση της `defineExchangeRate(1000)` από κάποιον διαχειριστή ώστε 1 eth = 1000 tokens.



[vm] from: 0x5B3...eddC4 to: Final\_p18029.defineExchangeRate(uint256) 0x406...2Cfbc value: 0 wei data: 0x2fc...003e8 logs: 0  
hash: 0x376...7e5cc

Τελικά, 1 eth = 1000 tokens.



*12. Οι πελάτες του ηλεκτρονικού καταστήματος θα μπορούν να ανταλλάσσουν ETH για νομίσματα του καταστήματος, όποτε το επιθυμούν, ακόμα και όταν δεν πραγματοποιούν κάποια αγορά.*

Για την εκπόνηση του παραπάνω ζητουμένου, έχει υλοποιηθεί η συνάρτηση `buyTokens` μέσω της οποίας οι πελάτες μπορούν στην ουσία να ανταλλάξουν eth για tokens ανάλογα με το exchange rate εκείνη τη στιγμή. Η συγκεκριμένη συνάρτηση έχει οριστεί ως payable που σημαίνει πώς για να κληθεί, ο χρήστης θα πρέπει να “εισάγει” ένα ποσό σε eth που θέλει να ανταλλάξει για το token του καταστήματος.

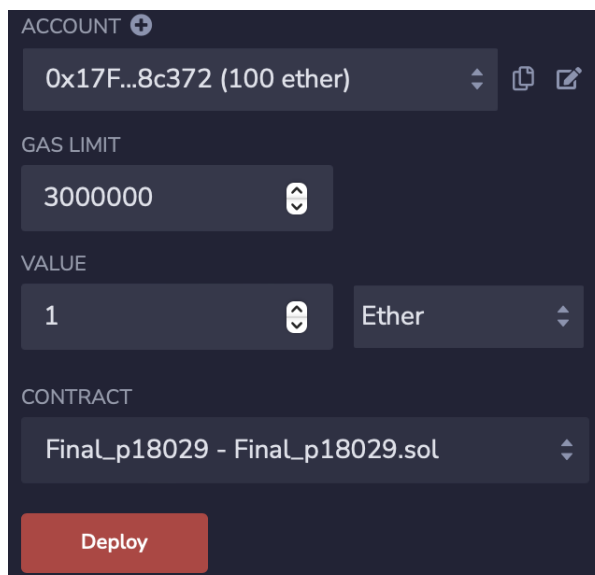
```
// minting shop tokens using ether
function buyTokens() external payable {
    uint256 tokensToBuy = (msg.value / 10 ** 18) * tokensPerEth;
    require((msg.value / 10 ** 18) > 0, "You should send some eth in order to buy tokens");


    balances[msg.sender] += tokensToBuy;



    emit Transfer(address(this), msg.sender, tokensToBuy);
    emit BuyTokens(msg.sender, msg.value, tokensToBuy);
}
```

Παρακάτω αναλύουμε ένα παράδειγμα κλήσης της συγκεκριμένης συνάρτησης. Καλούμε τη συνάρτηση εισάγοντας 1 eth από έναν πελάτη ο οποίος αρχικά έχει 0 tokens. Περιμένουμε πώς μετά την κλήση θα έχει 100 tokens αφού το exchange rate μετά το deployment του smart contract ορίζει πώς 1 eth = 100 tokens.


Εισάγουμε 1 eth από έναν λογαριασμό πελάτη.





ACCOUNT 

0x17F...8c372 (100 ether)  


GAS LIMIT

3000000 

VALUE

1  Ether 

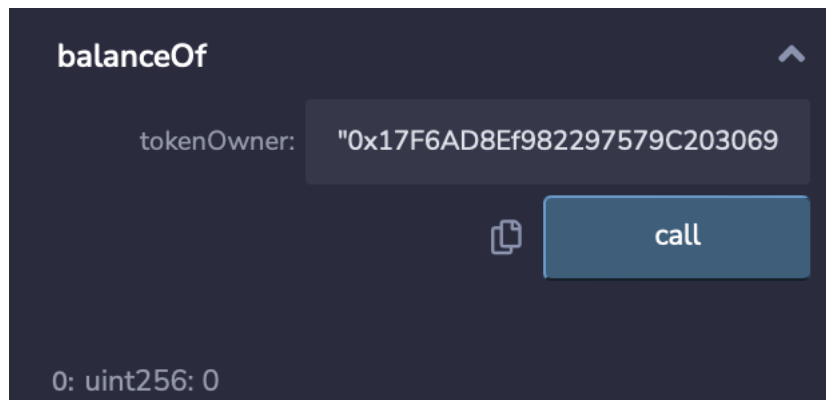
CONTRACT

Final\_p18029 - Final\_p18029.sol 

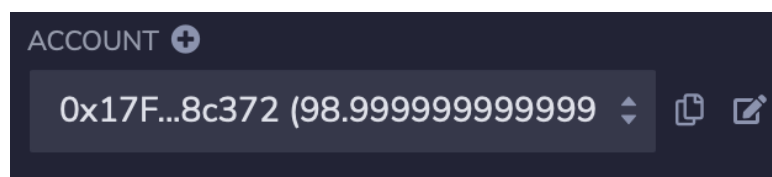
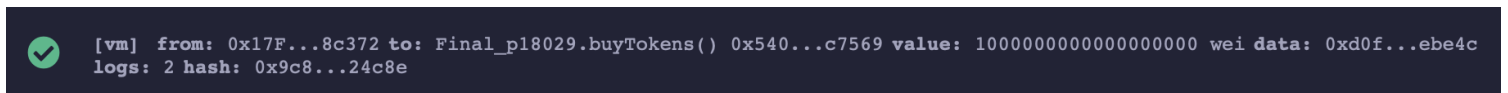
Deploy



Αρχικά έχει 0 tokens.



Καλείται η buyTokens εισάγοντας 1 eth και το eth balance του λογαριασμού μειώνεται ανάλογα.



Τελικά ο πελάτης λαμβάνει 100 tokens, όπως φαίνεται παρακάτω.

