Practical session for blockchain

(6) Token

Giyeong Lee[1] Jungyoon Song[1]

[1]Financial Risk Engineering Lab.
Seoul National University

May 9, 2022

Token

- **Coin** is crypto having its own main net.
  - Bitcoin, Ethereum, etc.
- **Token** is crypto used to achieve individual purposes and functions on the main net of other coins.
  - Tether USD, BNB, etc.

Purpose of Token

1. To use as a token(voucher) to be used for dApp services.
   - Using tokens has lower fees than using platform coins directly.

2. To raise funds for dApp development or any other project via **ICO(Initial Coin Offering)**.
   - e.g. EOS project

Examples of Ethereum-based Tokens

- **Tether USD(USDT)** : An Ethereum-based stablecoin issued by Bitfinex.
  - 1 USDT is pegged to 1 dollar.

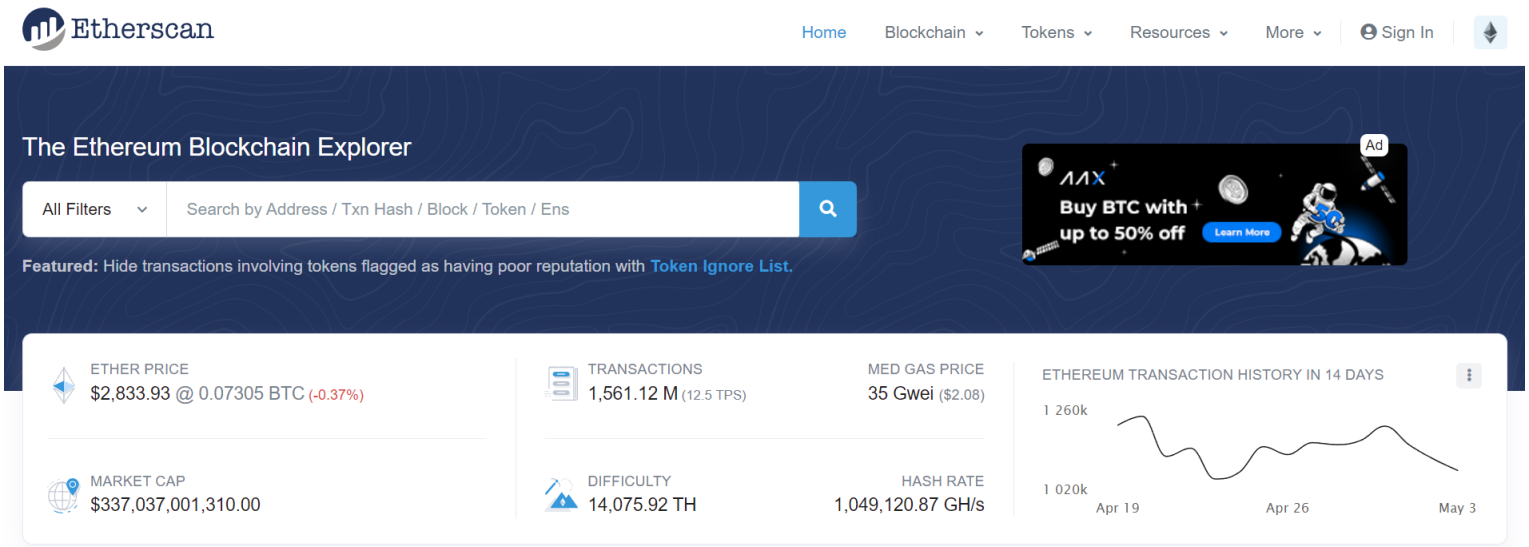
- **Binance Coin(BNB)** : a token issued by Binance that can be used within the Binance ecosystem.
  - With the launch of its own blockchain platform, Binance Chain, BNB moved to Binance Chain.

Stablecoin

Stablecoin is crypto whose value is pegged to a fiat currency to keep the value of the crypto stable.

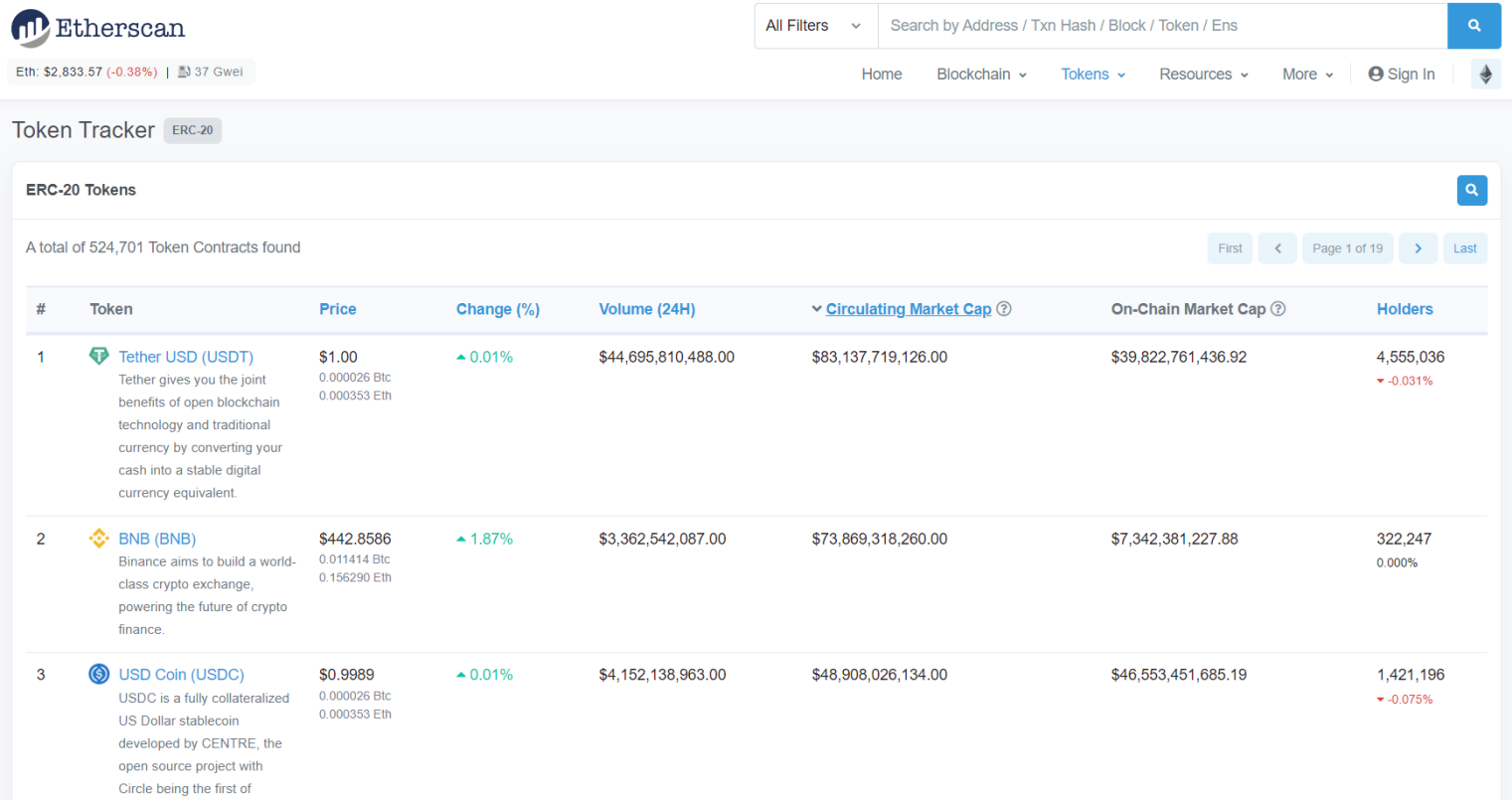
Etherscan

- **Etherscan** is an Ethereum blockchain explorer.



Etherscan

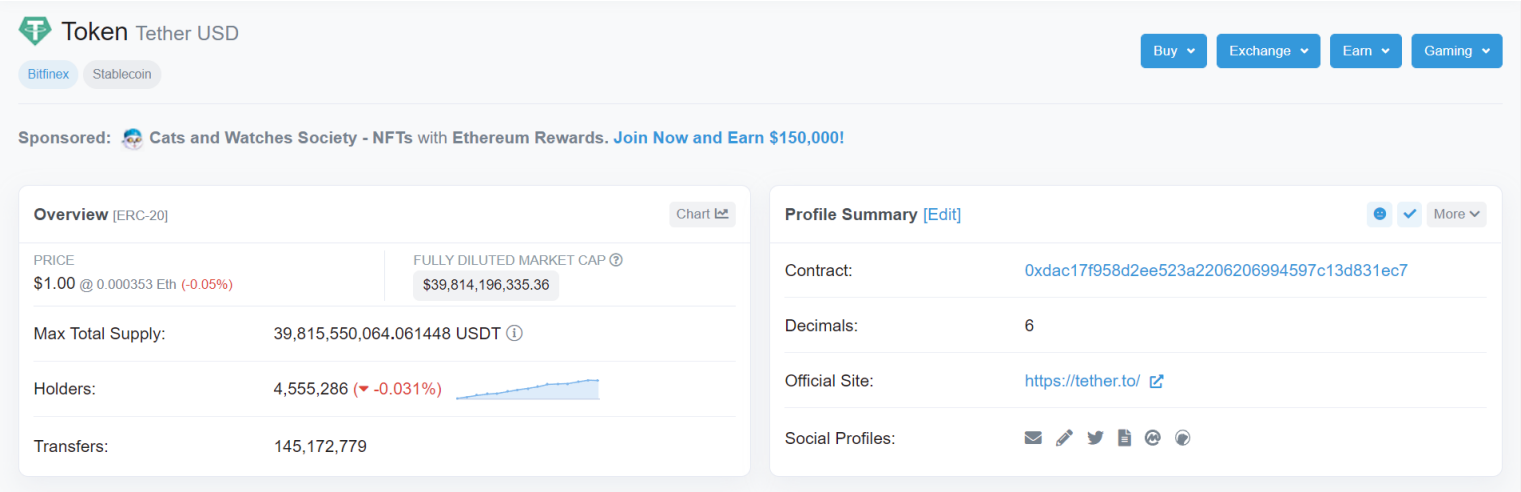
- Provides information about Ethereum-based tokens.



Etherscan

- Provides an overview of tokens and a list of TXs(token transfer, DEX trades, etc.)

- Also provides the source code of the token.
  - [Profile Summary - Contract - Tab: Contract]



ERC-20

- **Ethereum Request for Comment 20(ERC-20)** is a technical standard used for smart contracts on the Ethereum blockchain, developed in 2015.
- ERC-20 defines a common list of rules that an Ethereum token has to implement, giving developers the ability to program how new tokens will function within the Ethereum ecosystem.
- ERC-20 uses syntax from Solidity 0.4.17 (or above)

Token Contract Based on ERC-20

- For compatibility with the Ethereum network, ERC-20 specifies the following standards for Ethereum-based token contracts.

ERC-20 Specification: Methods (1)

1. *totalSupply* : Returns the total token supply.
2. *balanceOf* : Returns the account balance of another account with address *_owner*.

Token Contract Based on ERC-20

ERC-20 Specification: Methods (2)

3. *transfer* : Transfers *_value* amount of tokens to address *_to*.
4. *transferFrom* : Transfers *_value* amount of tokens from address *_from* to address *_to*.
5. *approve* : Allows *_spender* to withdraw from your account multiple times, up to the *_value* amount.
6. *allowance* : Returns the amount which *_spender* is still allowed to withdraw from *_owner*.

Token Contract Based on ERC-20

ERC-20 Specification: Optional Methods

1. *name* : Returns the name of the token.
2. *symbol* : Returns the symbol of the token.
3. *decimals* : Returns the number of decimals the token uses.

Token Contract Based on ERC-20

ERC-20 Specification: Events

1. *Transfer* : MUST trigger when tokens are transferred, including zero value transfers.
2. *Approval* : MUST trigger on any successful call to *approve*.

Implementation: Token Contract

[Ex. 1] Token Contract (1) State Variables

```
pragma solidity >0.7.0 <0.8.0;

contract MyToken{
    uint256 public totalSupply; // uint256 == uint
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping(address => uint256)) private approved;

    string public name;
    string public symbol;
    uint256 public decimals;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

Nested Mapping

A mapping can map to another mapping.

- To retrieve the value, all keys should be passed.
- e.g. mapping[key1][key2]

Implementation: Token Contract

[Ex. 1] Token Contract (2) Constructor

```
constructor (string memory _name, string memory _symbol,
             uint256 _supply, uint256 _decimals) {
    name = _name;
    symbol = _symbol;
    decimals = _decimals;

    balanceOf[msg.sender] = _supply * 10 ** _decimals;
    totalSupply = _supply * 10 ** _decimals;
}
```

Implementation: Token Contract

[Ex. 1] Token Contract (3) Transfer

```
function isValidTransfer(address _from, address _to, uint256 _value)
internal view returns (bool isValid) {
    if (balanceOf[_from] >= _value && balanceOf[_to] + _value >= balanceOf[_to]) {
        isValid = true;
    } else {
        isValid = false;
    }

    return isValid;
}

function transfer(address _to, uint256 _value) public returns (bool success) {
    if (isValidTransfer(msg.sender, _to, _value)) {
        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;

        emit Transfer(msg.sender, _to, _value);
        success = true;
    } else {
        success = false;
    }

    return success;
}
```

Implementation: Token Contract

[Ex. 1] Token Contract (4) Approve

```
function approve(address _spender, uint256 _value) public returns (bool success) {
    if (approved[msg.sender][_spender] + _value >= approved[msg.sender][_spender]) {
        approved[msg.sender][_spender] = approved[msg.sender][_spender] + _value;

        emit Approval(msg.sender, _spender, _value);
        success = true;
    } else {
        success = false;
    }
}

function allowance(address _owner, address _spender)
public view returns (uint256 remaining) {
    remaining = approved[_owner][_spender];
    return remaining;
}
```

Implementation: Token Contract

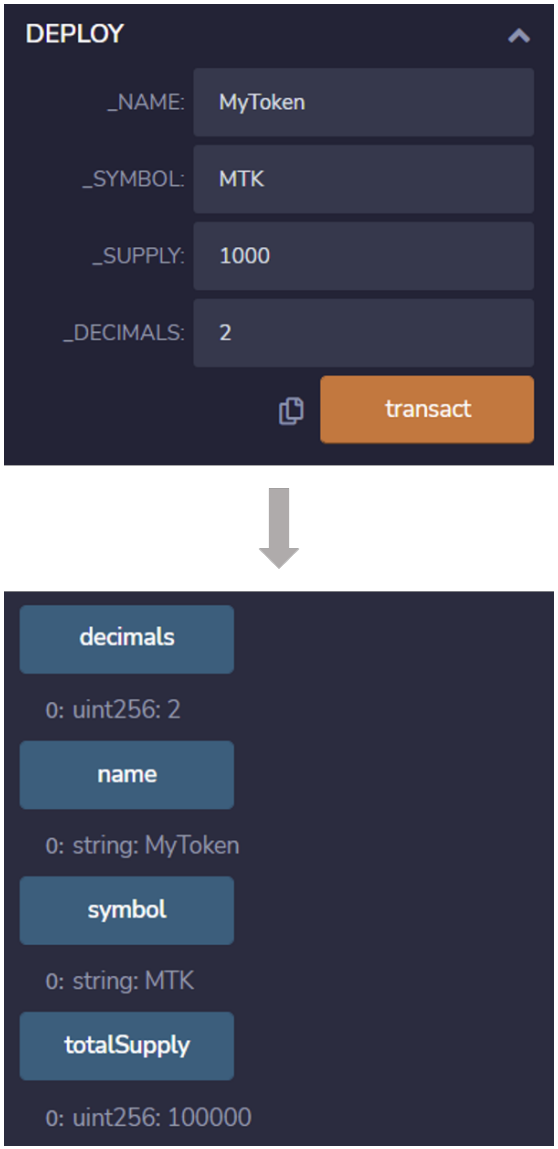
[Ex. 1] Token Contract (5) TransferFrom

```
    function transferFrom(address _from, address _to, uint256 _value)
    public returns (bool success) {
        if (allowance(_from, msg.sender) > 0 && isValidTransfer(_from, _to, _value)) {
            balanceOf[_from] -= _value;
            balanceOf[_to] += _value;
            approved[_from][msg.sender] -= _value;

            emit Transfer(_from, _to, _value);
            success = true;
        } else {
            success = false;
        }

        return success;
    }
}
```

Execution: Token Contract

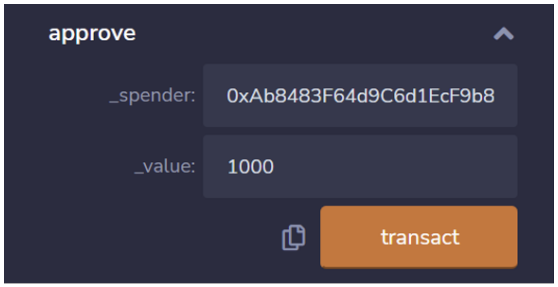


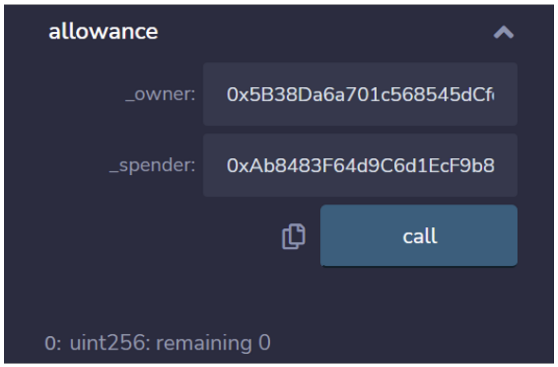
1. Set the name, symbol, supply, and decimals for the token contract.
2. Deploy the contract and check every variable is set correctly.

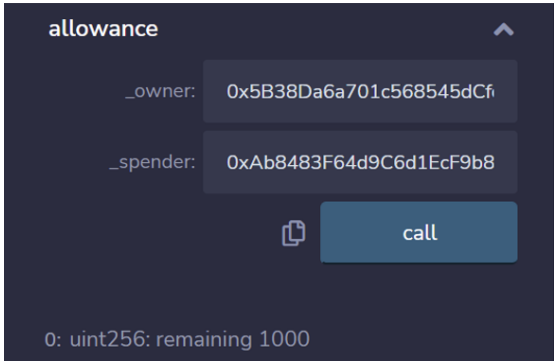
Execution: Token Contract

1. Using *transfer*, transfer tokens from the account that deployed the contract to another account.
2. See what happens if you transfer more than the tokens in the account.

Execution: Token Contract







1. Using *approve*, set the amount of tokens that another account can transfer.
2. Using *allowance*, check if the value(*remaining*) is set correctly.

Execution: Token Contract

1. Change to the approved account and use *transferFrom* to transfer money to another account.
2. See what happens if you transfer more than the approved tokens.

Additional Features

- We will implement two additional features:
  1. Blacklist Management
  2. Cashback for members

Implementation: Additional Features

[Ex. 2] Additional Features (1) State Variables

```
pragma solidity >0.7.0 <0.8.0;

contract MyTokenWithFeatures{
    uint256 public totalSupply; // uint256 == uint
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping(address => uint256)) private approved;

    string public name;
    string public symbol;
    uint256 public decimals;

    event Transfer(address indexed _from, address indexed _to, uint256 _value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
```

Implementation: Additional Features

[Ex. 2] Additional Features (1) State Variables (Cont.)

```
// For the blacklist and membership
address public manager;
modifier onlyManager() {
    require(msg.sender == manager, "Only the manager can call this function.");
    _;
}

// Blacklist
mapping (address => uint8) public blacklist;

event Blacklisted(address indexed _address);
event DeletedFromBlacklist(address indexed _address);
event RejectedPaymentToBlacklistedAddr(address indexed _from, address indexed _to,
uint256 _value);
event RejectedPaymentFromBlacklistedAddr(address indexed _from, address indexed _to,
uint256 _value);

// Membership
mapping (address => uint8) public memberships;
uint256 public cashbackRate;    // 0-100, 100 means 100%

event Cashback(address indexed _from, address indexed _to, uint256 _cashback);
```

Implementation: Additional Features

[Ex. 2] Additional Features (2) Constructor

```
constructor (string memory _name, string memory _symbol,
             uint256 _supply, uint256 _decimals) {
    manager = msg.sender;

    name = _name;
    symbol = _symbol;
    decimals = _decimals;

    balanceOf[msg.sender] = _supply * 10 ** _decimals;
    totalSupply = _supply * 10 ** _decimals;
}
```

Implementation: Additional Features

[Ex. 2] Additional Features (3) Blacklist

```
function isBlacklisted(address _address) public view returns (bool inBlacklist) {
    inBlacklist = blacklist[_address] == 1;
    return inBlacklist;
}

function pushBlacklist(address _address) public onlyManager {
    blacklist[_address] = 1;
    emit Blacklisted(_address);
}

function deleteFromBlacklist(address _address) public onlyManager {
    blacklist[_address] = 0;
    emit DeletedFromBlacklist(_address);
}
```

Implementation: Additional Features

[Ex. 2] Additional Features (4) Membership and Cashback

```
function setMembership(address _address, uint8 _isMember) public onlyManager {
    memberships[_address] = _isMember;
}

function setCashbackRate(uint256 _cashbackRate) public onlyManager {
    require(_cashbackRate >= 0 && _cashbackRate <= 100, "Invalid cashback rate.");
    cashbackRate = _cashbackRate;
}
```

Implementation: Additional Features

[Ex. 2] Additional Features (5) Transfer

```
function isValidTransfer(address _from, address _to, uint256 _value)
internal view returns (bool isValid) {
    if (balanceOf[_from] >= _value && balanceOf[_to] + _value >= balanceOf[_to]) {
        isValid = true;
    } else {
        isValid = false;
    }

    return isValid;
}
```

Implementation: Additional Features

[Ex. 2] Additional Features (5) Transfer (Cont.)

```
function transfer(address _to, uint256 _value) public returns (bool success) {
    if (isBlacklisted(msg.sender)) {
        emit RejectedPaymentFromBlacklistedAddr(msg.sender, _to, _value);
        success = false;
    } else if (isBlacklisted(_to)) {
        emit RejectedPaymentToBlacklistedAddr(msg.sender, _to, _value);
        success = false;
    } else if (isValidTransfer(msg.sender, _to, _value)) {
        if (_to == manager) {
            uint256 cashback = _value * cashbackRate / 100;
            _value -= cashback;
            emit Cashback(msg.sender, _to, cashback);
        }

        balanceOf[msg.sender] -= _value;
        balanceOf[_to] += _value;

        emit Transfer(msg.sender, _to, _value);
        success = true;
    } else {
        success = false;
    }

    return success;
}
```

Implementation: Additional Features

[Ex. 2] Additional Features (6) Approve

```
function approve(address _spender, uint256 _value) public returns (bool success) {
    if (approved[msg.sender][_spender] + _value >= approved[msg.sender][_spender]) {
        approved[msg.sender][_spender] = approved[msg.sender][_spender] + _value;

        emit Approval(msg.sender, _spender, _value);
        success = true;
    } else {
        success = false;
    }
}

function allowance(address _owner, address _spender)
public view returns (uint256 remaining) {
    remaining = approved[_owner][_spender];
    return remaining;
}
```

[Ex. 2] Additional Features (7) TransferFrom

```
    function transferFrom(address _from, address _to, uint256 _value)
    public returns (bool success) {
        if (isBlacklisted(_from)) {
            emit RejectedPaymentFromBlacklistedAddr(_from, _to, _value);
            success = false;
        } else if (isBlacklisted(_to)) {
            emit RejectedPaymentToBlacklistedAddr(_from, _to, _value);
            success = false;
        } else if (allowance(_from, msg.sender) > 0 && isValidTransfer(_from, _to, _value)) {
            if (_to == manager) {
                uint256 cashback = _value * cashbackRate / 100;
                _value -= cashback;
                emit Cashback(msg.sender, _to, cashback);
            }

            balanceOf[_from] -= _value;
            balanceOf[_to] += _value;
            approved[_from][msg.sender] -= _value;

            emit Transfer(_from, _to, _value);
            success = true;
        } else {
            success = false;
        }

        return success;
    }
}
```

References

- Etherscan, `https://etherscan.io`
- EIP-20: Token Standard, `https://eips.ethereum.org/EIPS/eip-20`