Ethereum Smart Contract

조성은

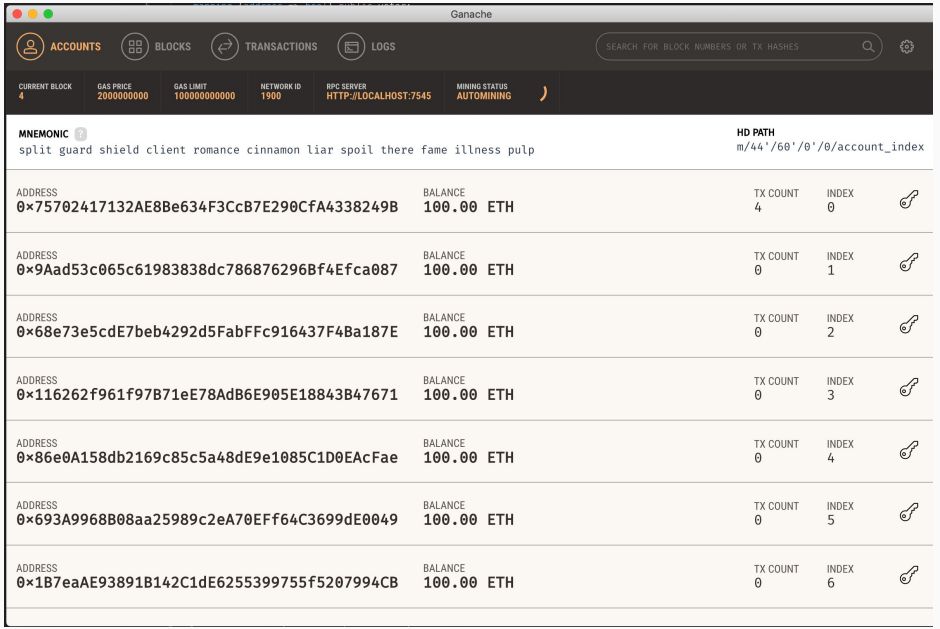
목차

소스코드 공유

https://github.com/Hexlant/EthereumContractStudy

Ganache

https://truffleframework.com/ganache



Ganache 소개

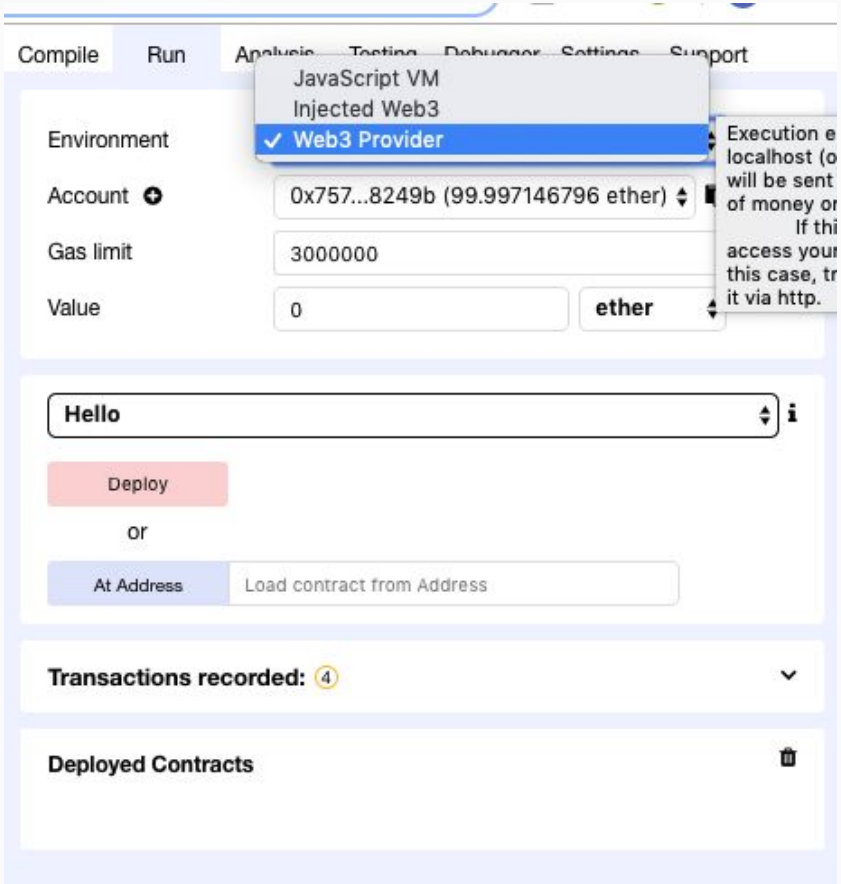
실제 메인넷에 배포하여 테스트 할 경우, 비용과 시간이 많이 들어갑니다. 따라서 대부분의 사람들은 테스트 넷이나 프라이빗 네트워크를 구축하여 테스트를 진행합니다.

하지만 테스트넷을 구축하고 사용하기에는 복잡한 절차가 있습니다.

가나슈는 로컬에 가상의 네트워크를 구축하여, 보다 빠르게 테스트 할 수 있는 테스트 도구입니다.

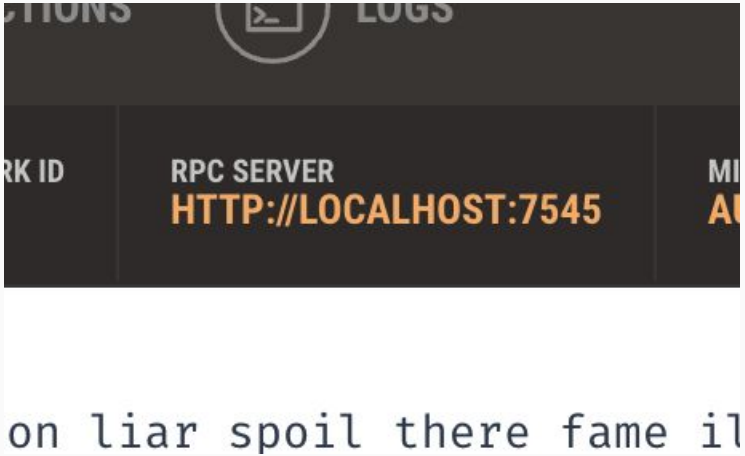
Ganache - Remix와 연동하기

Run>Environment>Web3 Provider 선택



Ganache - Remix와 연동하기

Ganache의 RPC Server 주소 입력



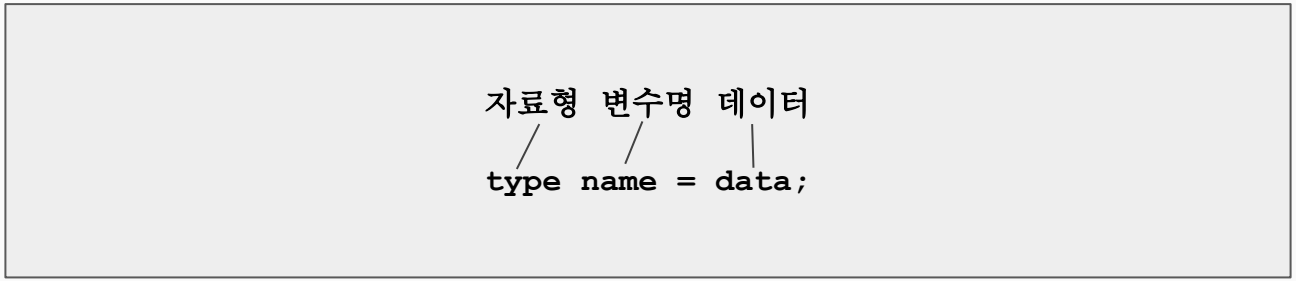
솔리디티 소스 파일

- 솔리디티 소스 파일의 확장자는 .sol이다. 파일 내에서 pragma solidity를 사용, 컴파일러 버전을 지정할 수 있다.

```
pragma solidity ^0.4.21
```

기본자료형

- 자료형 : 데이터의 종류
- 변수 선언 시 데이터 생략 가능
- constant 키워드 사용 시 상수
- Public 상태변인 경우 기본적으로 Getter 기능 형태로 읽을 수 있음



기본자료형

- 정수형 : int, uint (8, 16, 24, 32, 64, 128, 256)
- 참거짓형(bool) : true or false
- 나열형(enum) : 개발자가 정의하는 자료형

```
enum Tier {Bronze, Silver, Gold, Platinum, Diamond}
```

주소형

- address 키워드
- 20바이트
- 컨트랙트의 주소 저장
- 40자리의 16진수 정수
- 함수제공
  - balance : 해당 주소의 이더 잔고 조회
  - transfer, send : 해당 주소로 이더를 송금할 때 사용

```
address recipient = 0xABC;
uint256 balance = recipient.balance;
```

주소형

- 이더리움을 받을 주소는 payable 옵션이 붙어있어야 함.
- send : 송금 실패 시 false 값을 반환하며 코드 계속 실행
- transfer : 송금 실패시 즉시 오류 발생시켜 컨트랙트 실행이 취소됨
    - 반드시 send를 사용해야 하는 경우가 아니라면 transfer 사용을 권장

```
pragma solidity ^0.5.0;

contract AddressContract {
     function exampleAddress() {

          address sender = this;
          address payable recipient = 0xABC;

          recipient.transfer(50);
     }
}
```

튜플

- 크기가 컴파일 전에 미리 정해진 데이터들의 묶음
- 괄호로 묶어 선언

```
pragma solidity ^0.5.0;

contract TupleContract {

     function exampleTuple() public pure returns (uint8 a, uint8 b){

          (uint8 x, uint8 y, bool z) = (1, 2, true);
          (x, y) = (y , x);
          (a, b) = (x, y);
     }
}
```

함수

- visibility option : external / public / internal / private
- constant option: view / pure
- payable option : payable

```
function name(x1, x2, … , xn) option returns (y1, y2 …, yn){
      …
}
```

Visibility Option

- public : 함수의 Default Visibility, 컨트랙트 바깥과, 안쪽에서 호출이 가능합니다.
- private : 동일한 컨트랙트 안에서만 접근이 가능합니다.
- external : 컨트랙트 외부에서만 접근가능합니다. 내부에서 쓰기위해선, this.func() 형태로 씁니다.

  Public 함수보다 가스 소모가 적습니다.
- internal : 해당 컨트랙트 또는, 상속받은 컨트랙트에서 호출이 가능합니다.

```
function name(x1, x2, … , xn) option returns (y1, y2 …, yn){
      …

}
```

Constant Options

상수함수는 트랜잭션을 생성하지 않습니다.

상수함수에서는 상태변수의 값을 바꾸는 것이 불가능하며, 조회만 가능합니다.

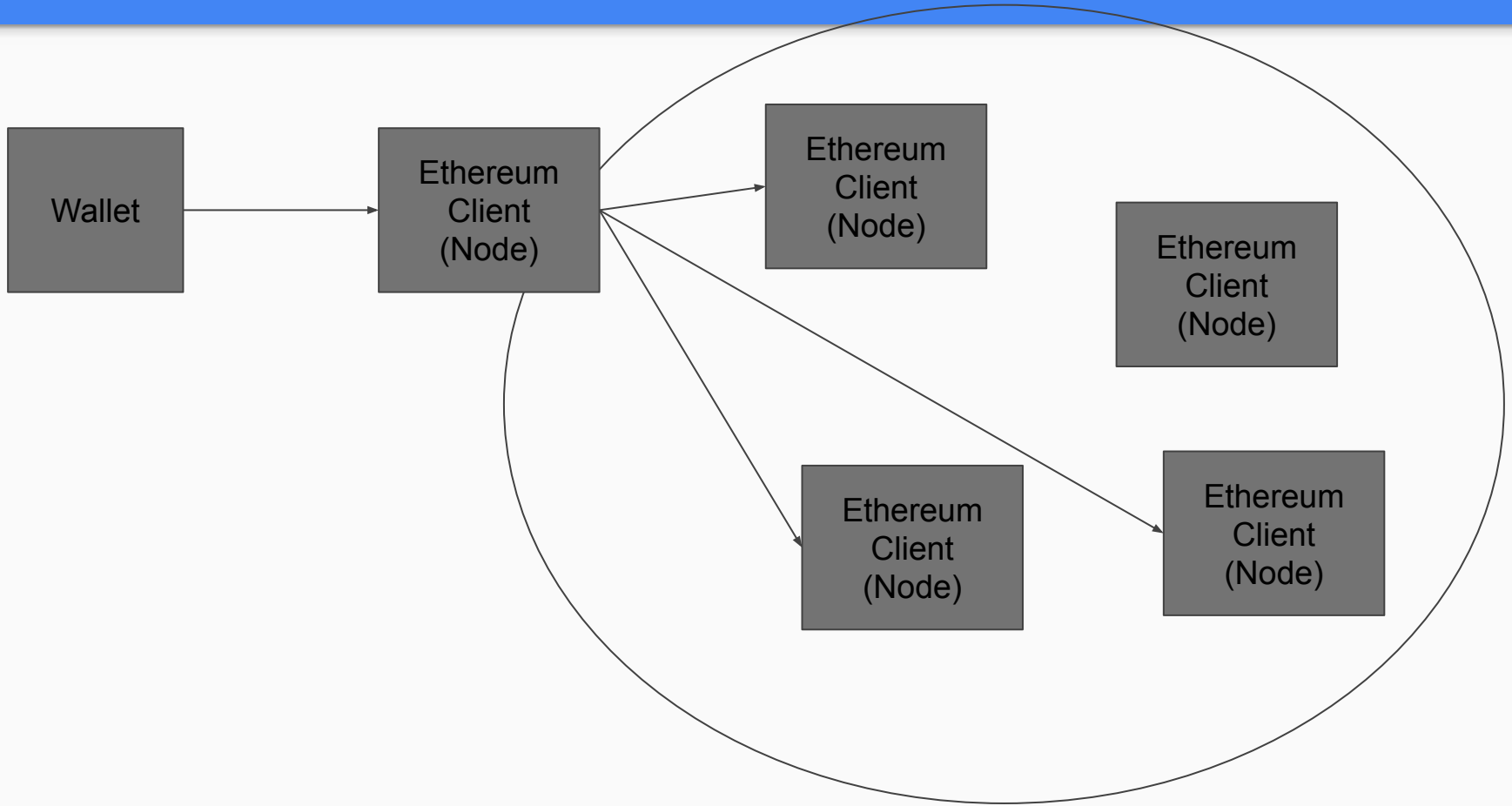
상수함수는 트랜잭션을 생성하지 않기 때문에 event를 발생시키지 않습니다.

- view : 상태변수의 값을 읽을 수 있습니다.
  상태변수 값을 가져와 다양한 연산을 수행하고 그 결과를 리턴할 수 있습니다.
- pure : 상태변수에 접근이 불가능 합니다.

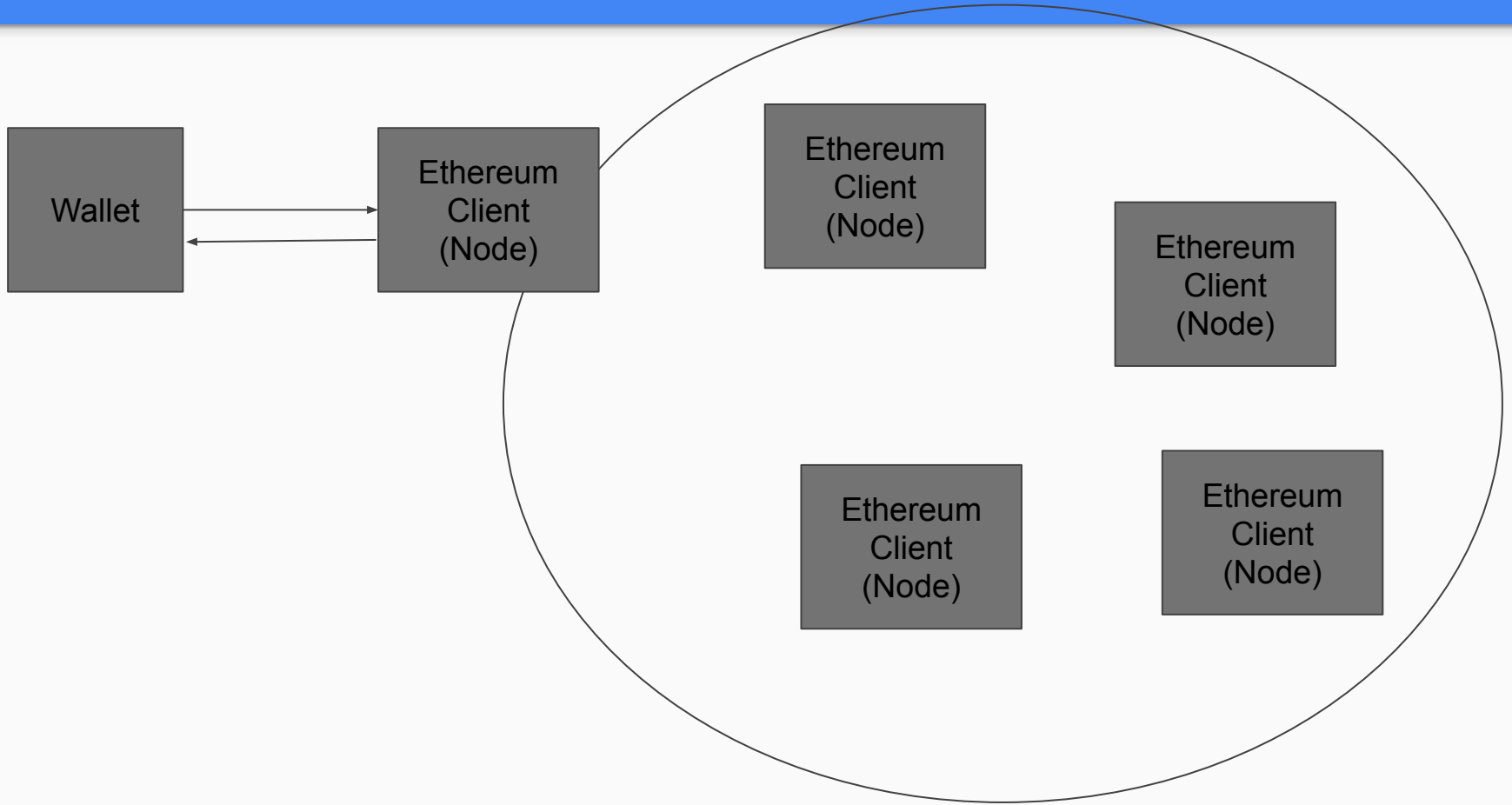
Constant Options

상수함수라고 하더라도 연산에 대한 gas측정이 가능합니다.
트랜잭션으로 호출하는 함수에서, 상수함수를 호출하면, Gas가 소모됩니다.

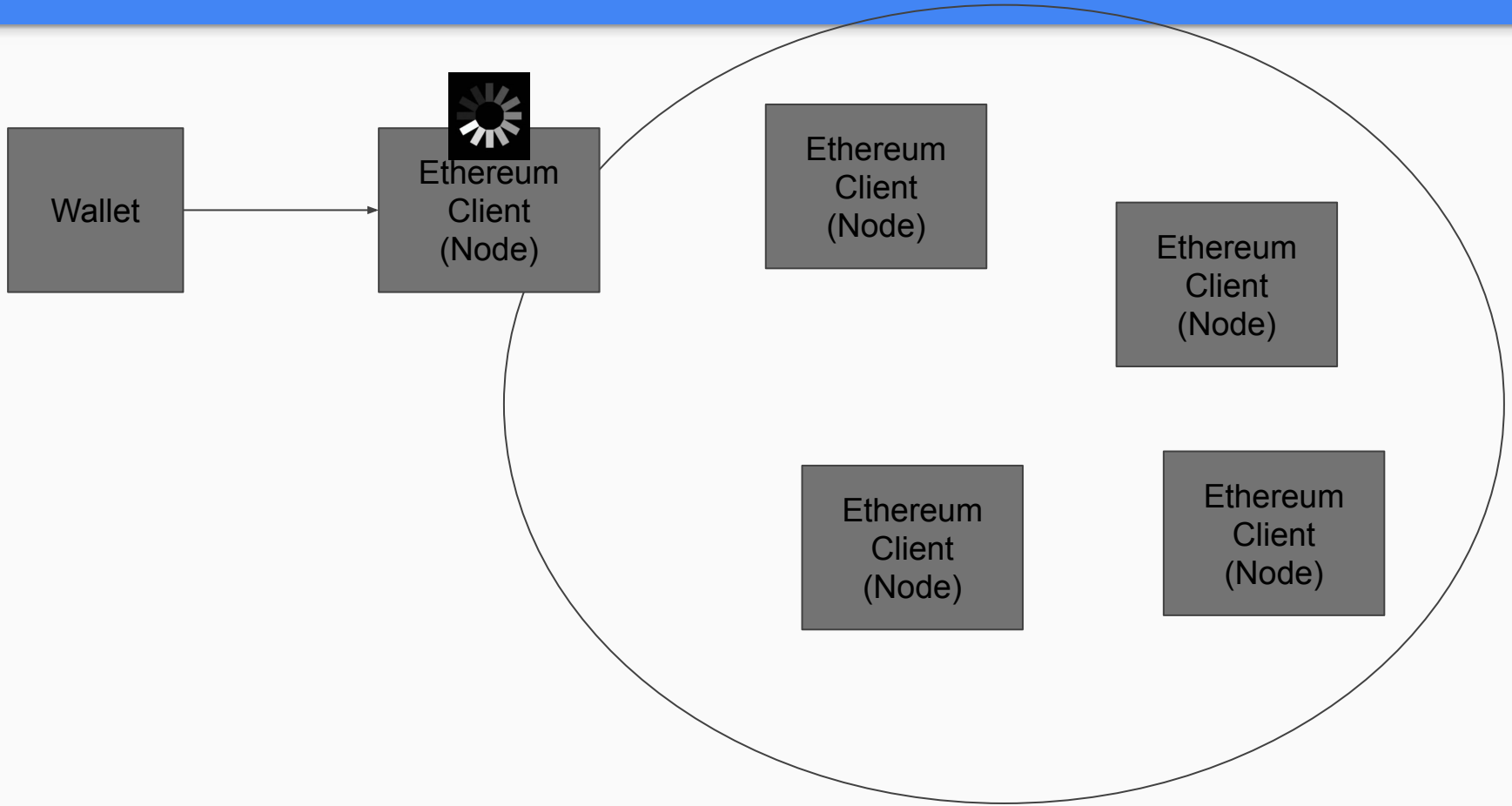
트랜잭션이 필요한 함수 호출



트랜잭션이 필요없는 상수 함수 호출(Read Only)



상수함수에 무한루프 적용 시



Payable Option

Eth를 받을 수 있게 하는 옵션입니다.
Payable Option이 없는 함수를 호출 할 때에는 Eth를 넣어서 트랜잭션을 보낼 수 없습니다.

Fallback 함수

프로그램의 의도하지 않은 동작을 할 때 오류를 다루기 위한 함수
함수명이

- 컨트랙트에 트랜잭션이 왔지만, 아무런 함수도 실행 시키지 않았을 때
- 컨트랙트에 없는 함수를 호출할 때.

```
function () {....};
```

생성자

- 컨트랙트가 생성될 때 딱 한번만 실행
- 보통 변수들의 초기값 세팅에 사용

```
contract Constructor{
      uint count;
      address from;
      address to;

      constructor(uint _count, address _from, address _to){
            count = _count;
            from = _from;
            to = _to;
      }

}
```

조건문

- 주어진 조건이 참 또는 거짓인지 여부에 따라 다른 작업 수행

```
if(조건A){
      조건A가 참인경우 수행
}
else if(조건B){
      조건A가 거짓이며 조건B가 참인 경우
}
else{
      조건A와 조건B모두 거짓인 경우
}
```

삼항 조건 연산자

- if else문을 사용하면 길어지는 코드의 길이를 대신하는 연산자
- expression 이 참이면 valueT 반환
- expression 이 거짓이면 valueF 반환

```
variable = expression ? valueT : valueF;
```

for 문

- 반복되는 코드 표현

```
for(초기값; 조건식; 증가 또는 감소식){
      반복할 코드
}
```

```
uint sum = 0;

for(uint = 0; i < 1000; i++){
      sum += i;
}
```

while 문

- for 문과 마찬가지로 반복되는 코드 표현
- for문과 달리 종료 조건식만 표현

```
while(조건식){
      반복할 코드
}
```

- 조건식 진입 전에 반복할 코드를 선 실행 후 반복진행

```
do{
      반복할 코드
}while(조건식);
```

배열

- 연속된 데이터를 묶는 성질
- 크기(size) : 배열의 크기
- 색인(index) : 배열의 위치

```
contract ArrayContract{
     uint[3] arrMy1;
     uint[3] arrMy2 = [0, 1, 2];

     function example(){
          arrMy1[0] = 0;
          arrMy1[1] = 1;
          arrMy1[2] = 2;

          arrMy2[1] = 11;
     }
}
```

배열

- 함수 지원
  - length : 배열의 크기 반환
  - push : 배열의 마지막 위치에 값을 추가하고 길이 반환. 저장소 동적 배열에서 사용
- 정적배열 : 크기가 정해져 있음
- 동적배열 : 실행 시 크기가 정해짐

```
contract ArrayContract{
      uint[3] arrStatic = [1, 2, 3];
      uint[3] arrDynamic = [1, 2, 3];

      function example(){
            uint[] arrLocalDynamic = new uint[](8);
      }
}
```

메모리 배열

- 메모리에 저장된 데이터는 컨트랙트의 실행이 종료되면 지워짐
- 지역변수의 경우 memory 키워드로 메모리에 저장 가능
- new 키워드 사용 시 임시로 메모리에 저장되는 배열

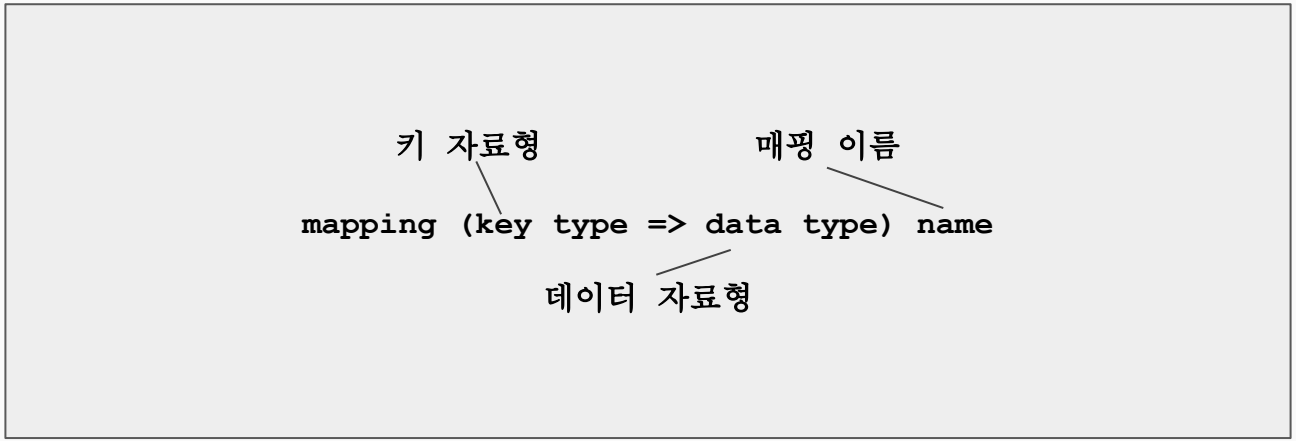
```
contract ArrayContract{
      ...

      function example(){
            uint[3] memory arr = [1, 2, 3];
            uint[] memory arr2 = new uint[](3);
      }
}
```

저장소 배열

- 블록체인에 저장되어 컨트랙트 실행이 종료되어도 영구히 저장
- 메모리에 저장하는것보다 많은 gas사용
- 상태 변수는 무조건 저장소에 저장
- 지역 변수 기본으로 저장소에 저장

맵핑 (Mapping)

- Key-Value의
- 상태변수로만 사용 가능



맵핑 (Mapping)

- 키, 밸류를 사용하여 선언, 접근, 대입이 가능
- 키에는 맵핑, 동적배열, 스마트 컨트랙트, 구조체 제외 모든 자료형 사용 가능
- 밸류에는 맵핑을 포함한 모든 자료형 사용 가능

```
contract MappingContract{
     mapping (address => uint) public balances;

     function example(uint input){
          balances[msg.sender] = input;
     }
}
```

투표 컨트랙트

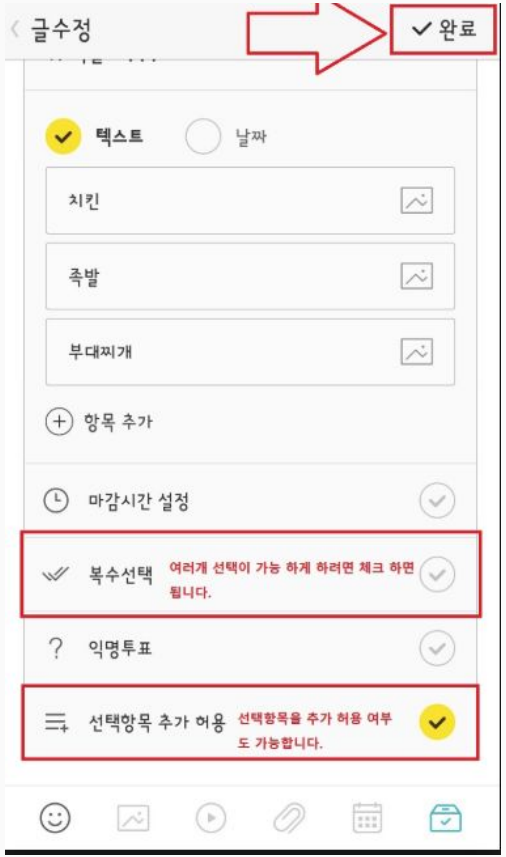
요구사항

1. 투표항목을 만들 수 있다.

2. 1인 1표 1개 항목만 투표할 수 있다.

3. 투표에 대해 event를 남겨 확인이 가능해야한다.

4. 몇개의 항목이 있는지, 어떤 항목이 있는지 알 수 있다



ERC20

EIP (Ethereum Improvement Proposal)
https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

20번째에 제안된 표준 토큰에 대한 정의

표준 토큰의 기본 인터페이스에 대한 정의로,
적어도 이더리움에서 움직이는 토큰이면 이 기능들을 지원해야 한다는 제안.

ERC20

표준을 지키지 않더라도 컨트랙트 배포에는 문제 없음.

다만 대부분의 거래소, 지갑 서비스들이 ERC20 표준 토큰을 지원하기에,
표준을 지키지 못하면 사용성이 떨어짐.

표준기능 이외에 다양한 기능을 추가로 넣을 수 있음.

Spec

METHODS

name
symbol
decimals
totalSupply
balanceOf
transfer
transferFrom
approve
allowance

Events

Transfer
Approval

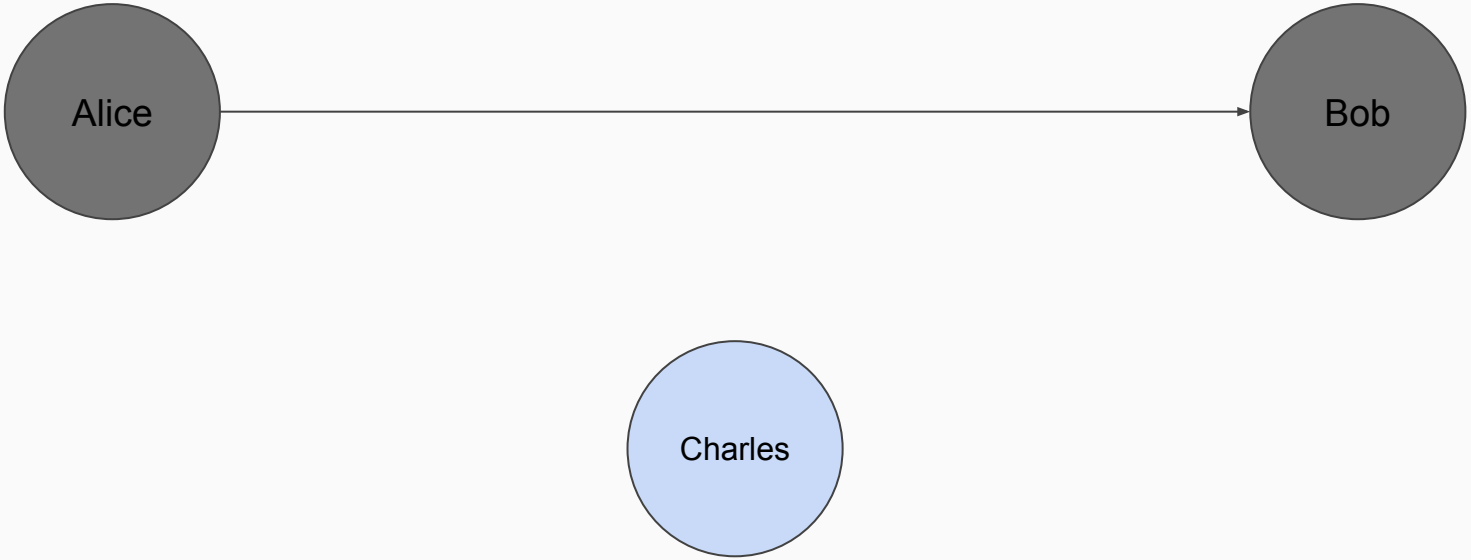
ERC-20 Interface

1. totalSupply() public view returns (uint256 totalSupply)
   전체 토큰의 발행량을 가져옵니다.
2. balanceOf(address _owner) public view returns (uint256 balance)
   _owner가 가진 토큰의 잔액을 가져옵니다.
3. transfer(address _to, uint256 _value) public returns (bool success)
   _value만큼의 토큰을 _to 주소로 전송합니다
4. transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
   _from의 주소에서 _to주소로 _value만큼의 토큰을 보냅니다.
5. approve(address _spender, uint256 _value) public returns (bool success)
   _spender에게 _value만큼의 토큰사용을 허용합니다.
6. allowance(address _owner, address _spender) public view returns (uint256 remaining)
   _owner가 _spender에게 사용을 허락한 토큰 양을 가져옵니다.

Transfer

Alice : transfer(bob, 100)

TransferFrom



Charles : transferFrom(alice ,bob, 100)

TransferFrom을 쓰는 이유는?

여러 개의 지갑의 자산을 관리해야 할 때.

이더리움은 Transaction을 처리 할 때 Gas를 지불할 ETH를 필요로 함.
따라서 ERC-20토큰만 있는 경우, 토큰 전송을 할 수 없음.

여러개의 지갑을 관리한다고 했을때 각 지갑에 ETH를 일정량 보관해야 하며, 매번 토큰 전송시마다 ETH 수량을 체크해야함.

만일 여러개의 지갑들이 관리자 지갑에 Approve를 최대치로 주게되면, 이후부터는 각지갑이 토큰을 전송하지 않고, 관리자 지갑을 통해 자산 이동이 가능해짐.

Decimals

이더리움에서 숫자의 표현은 정수형으로만 가능
소수점 이하 표현이 불가능 하다.

따라서 서로 약속에 의해 10의 18승을 1로 표현함
1 ETH = 10 ** 18 Wei

```
function name() public view returns (string)
function symbol() public view returns (string)
function decimals() public view returns (uint8)
function totalSupply() public view returns (uint256)
function balanceOf(address _owner) public view returns (uint256 balance)
function transfer(address _to, uint256 _value) public returns (bool success)
function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)
function approve(address _spender, uint256 _value) public returns (bool success)
function allowance(address _owner, address _spender) public view returns (uint256 remaining)

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

실습

ERC-20 기능 구현하기.

소소코드 :ERC20Skeleton.sol

Name, Symbol, Decimals

토큰의 이름과 심볼, 표현가능한 소수점 자리수는 변경할 일이 없음.

어차피 Set기능을 지원 하지 않을 것이라면,
Function으로 만들 필요없이 상태변수로 만들수 있다.

```
string public name;
string public symbol;
uint8 public decimals;
```

Underflow/Overflow

uint8 일때, 0에서 -1을 하면, 255가 됩니다.
uint8 일때, 255에서 +1을 하면 0이 됩니다.

잔액이 0원인 사람이 부자가 될 수도 있고,
부자는 거지가 될 수 있습니다.

따라서 산술 연산시에는 Underflow / Overflow에 대한 예외처리가 되어야 합니다.

BEC, SMT 토큰 무한생성?

Safemath는 왜 써야하는가

실습

Safemath

Safemath는 산술연산에서 발생 할 수 있는 여러가지 예외상황을 처리합니다.

소스코트 : Safemath.sol

ERC20 확장하기

1. 추가발행 기능 구현
   토큰 최초 발행 이후, 추가적으로 발행 할 수 있도록 합니다.

2. 관리자 기능 구현
   추가 발행 기능을 관리자만 호출 할 수 있도록 합니다.
   Admin.sol

3. Ethereum으로 토큰 구매하기
   Ethereum을 보내면 토큰을 구매할 수 있도록 합니다.

슬롯머신 컨트랙트

슬롯머신 게임 에제를 통해 다음과 같은 내용을 학습합니다.

1. 블록체인 위에서 랜덤 값을 생성하는 방법.
2. 컨트랙트 코드에서 트랜잭션, 블록의 정보를 가져오는 방법
3. 컨트랙트를 삭제하는 방법

4. 도박의 위험성

게임 구현을 위한 상태변수

```
struct game {
        address player;     //게임의 참가자
        bool win;               // 승/패 여부
        uint256 bettingAmount; // 배팅금액
        uint256 gameResult;     // 게임에서 나온 랜덤 값
        uint256 reward;             // 승리배당
}
address public owner;  //슬롯머신 컨트랙트의 주인입니다.
game[] public games;   //게임의 기록입니다.
```

게임 결과 이벤트

```
event GameResult(
        address player,
        bool win,
        uint256 amount,
        uint8 number1,
        uint8 number2,
        uint8 number3,
        uint256 reward
        );
```

컨트랙트 주인의 역할

```
    constructor() public {
        owner = msg.sender;                 // 컨트랙트를 배포한 사람이 컨트랙트의 주인이 된다
    }

    function destroy() public{
        require(owner == msg.sender);
        selfdestruct(owner);                // 게임을 삭제하고, 컨트랙트가 가지고 있는 이더리움을
돌려받는다
    }

    function chargeMoney() public payable{
        require(owner == msg.sender);       //게임 배당금을 채워 넣는다.
    }
```

배팅 기능 구현

```
function bet() public payable{

    require(msg.value > 0);                          //배팅금액이 0이면 참여불가
    require(address(this).balance > msg.value*8);    //최대 배당금보다 컨트랙트가 이더가 적으면 안된다.

    bool win = false;

    uint32 gameResult = uint32(blockhash(block.number-1))% 1000;      // 블록의 해시값으로 부터 숫자를 얻는다
    uint8 n1 = uint8( gameResult / 100 );                             // 각 자리수 별로 숫자를 획득
    uint8 n2 = uint8( (gameResult%100) / 10 );
    uint8 n3 = uint8(  gameResult % 10 );
```

배팅 기능 구현

```
uint256 reward = msg.value;
if(n1 == n2) { reward = reward*2; win = true; }  // 같은 숫자가 나올때마다 리워드 금액 2배 증가
if(n2 == n3) { reward = reward*2; win = true; }
if(n1 == n3) { reward = reward*2; win = true; }
if(win){
    msg.sender.transfer(reward);                    // 승리 시, 배당금 전송
} else {
    reward = 0;
}
    emit GameResult( msg.sender, win, msg.value, n1, n2, n3, reward);
    games.push( game( msg.sender, win, msg.value, gameResult, reward)); // 게임이력 저장
}
```