

예제로 배우는 스마트 컨트랙트 개발

오재훈

jaehoon@nethru.co.kr

Byzantium [Megara Geth1.7](#)

Lab 0. 실습 준비

Step 1. Mist 설치하기

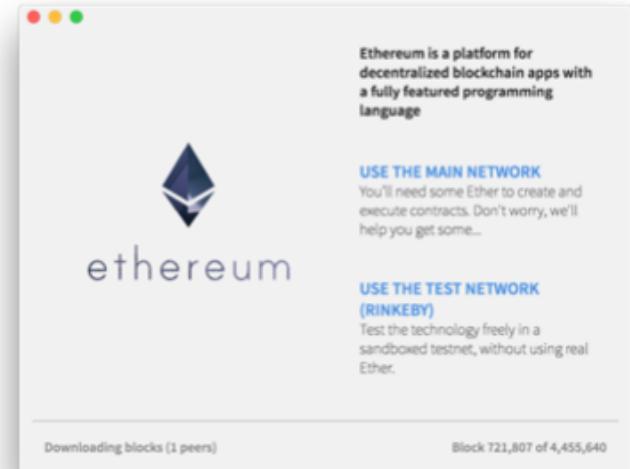
1. Mist 다운로드

- <https://github.com/ethereum/mist/releases>
- Mac 용 : [mist-macosx-0-9-2.dmg](#)
- Windows 용 : [mist-installer-0-9-2.exe](#)

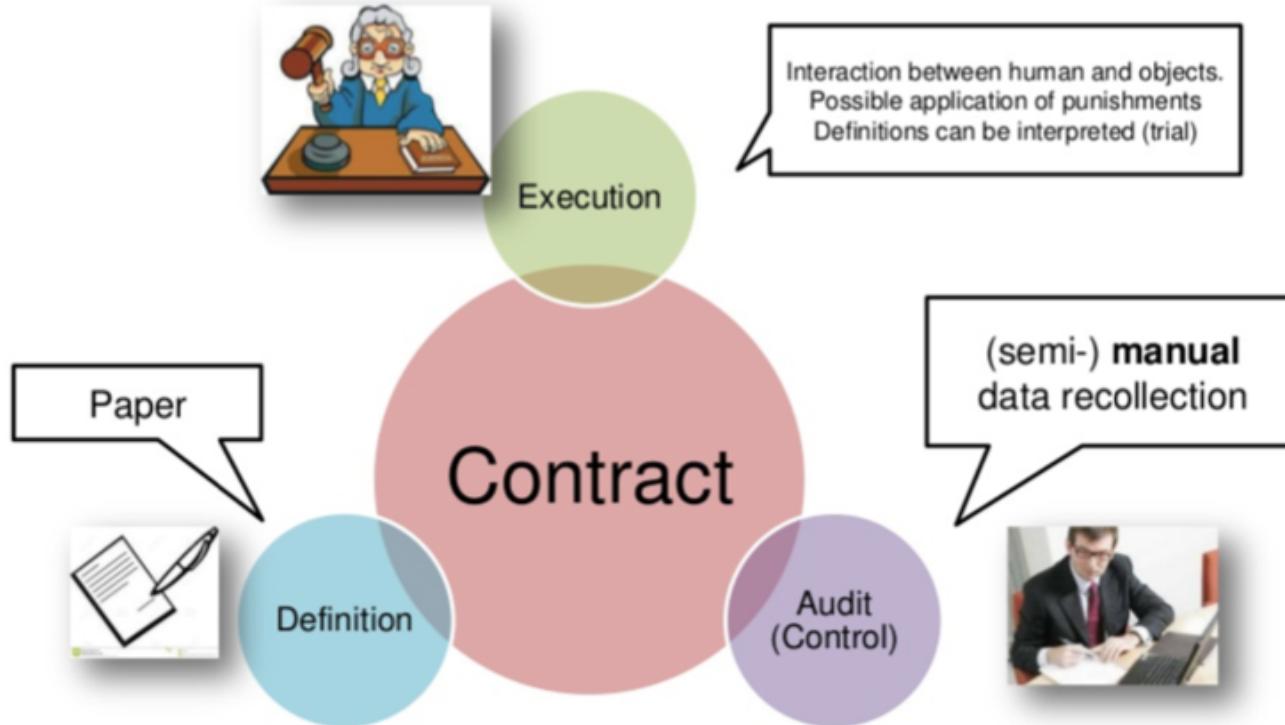
1. Mist 설치 하기

1. Mist 실행하기

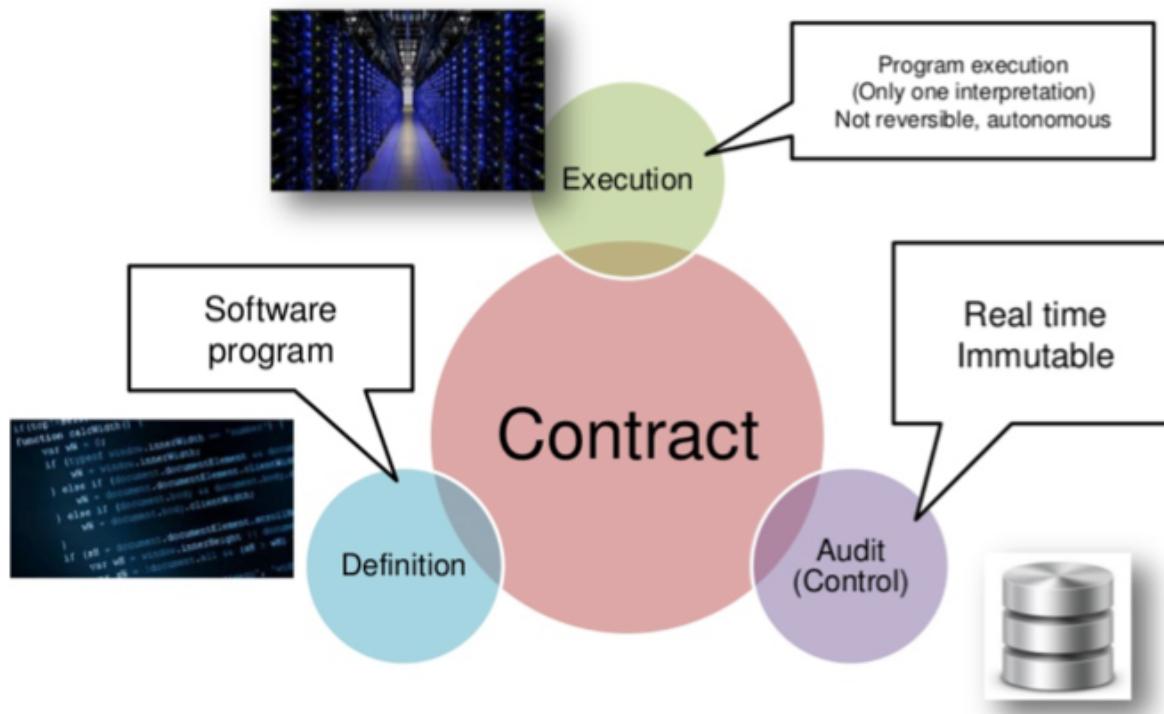
- Mist 를 실행하면, node 를 실행하기 위한 바이너리들을 설치하기 시작한다.
- 설치가 종료되면 오른쪽 화면이 나타난다.
- geth 클라이언트를 다운로드 했는지 확인한다.
 - Mac : ~/Library/Application Support/Mist/binaries/Geth/unpacked/geth
 - Windows : C:\Users\사용자계정\AppData\Roaming\Mist\binaries\Geth\unpacked\geth
- Mist 를 종료한다.



Traditional Contract



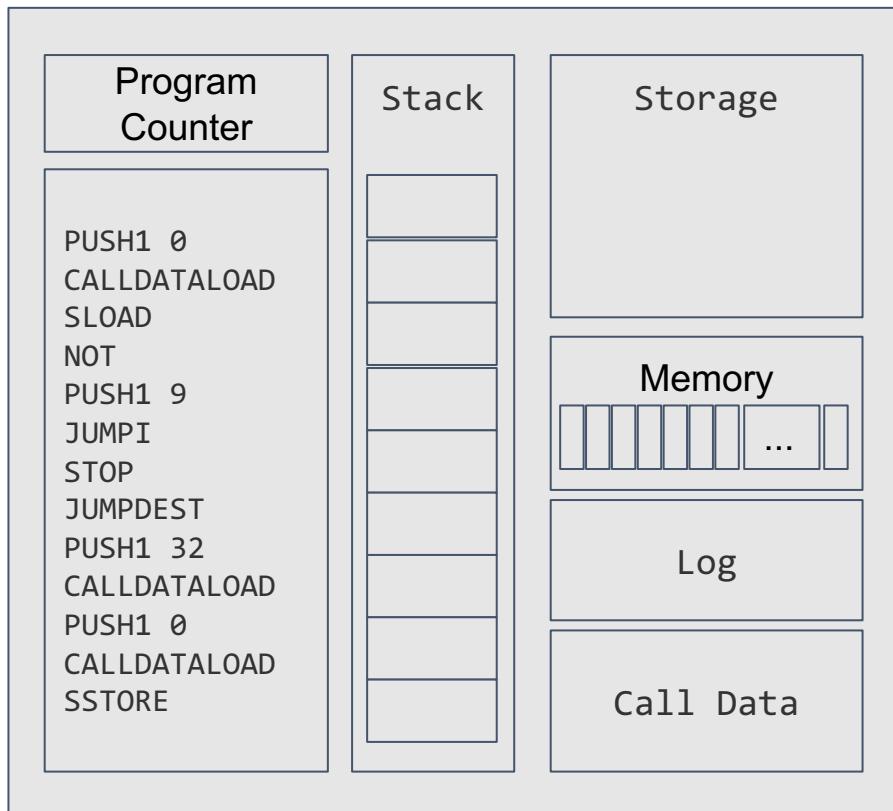
Smart Contract



Solidity

Ethereum 스마트 컨트랙을 개발할 때 가장 많이 사용되는 언어
문법은 자바 언어와 유사함
객체지향 언어
정적 타입언어
스코프 : Function Scope (참고 : Block Scope)

Ethereum Virtual Machine



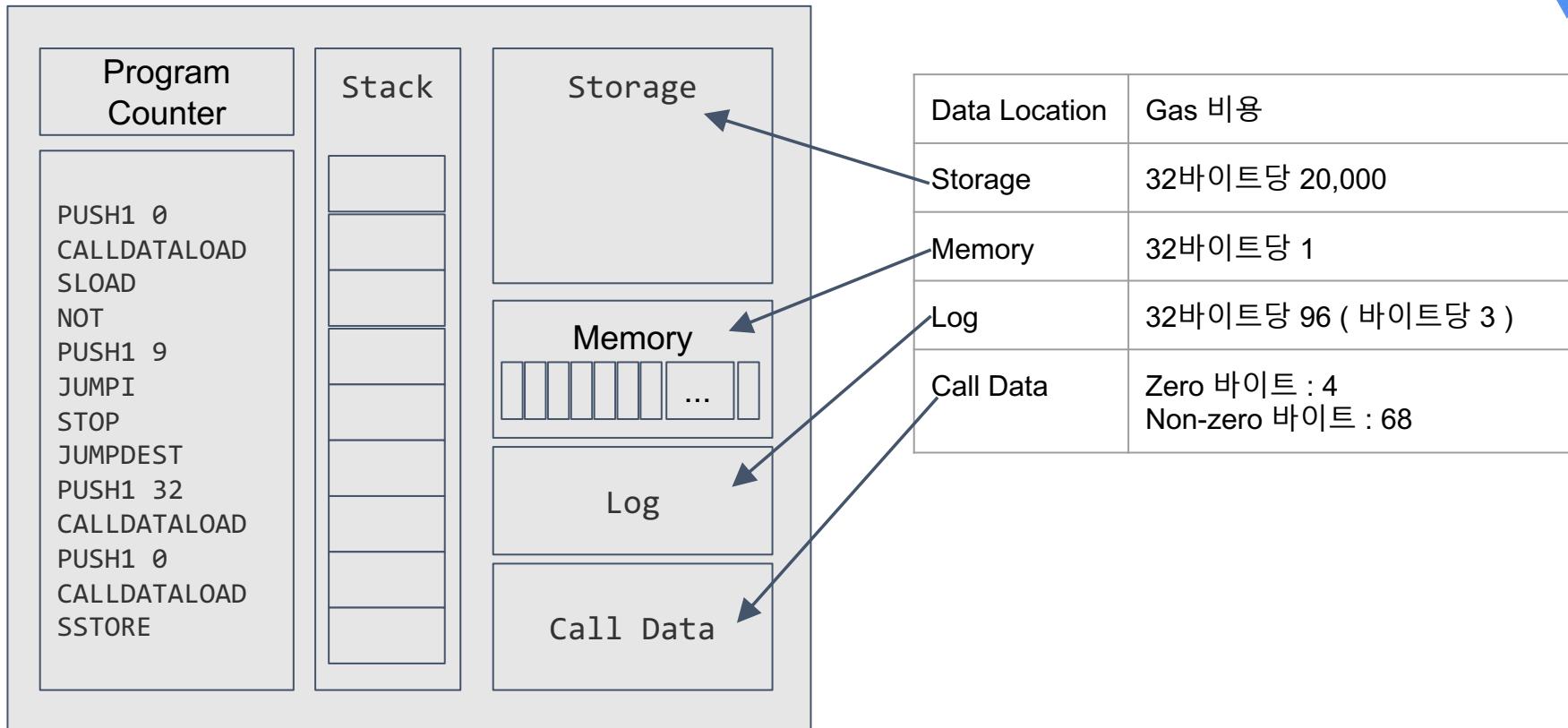
256 비트 가상 머신
스택기반 머신
Turing Complete

Halting Problem - Gas Cost Model

	Fuel (gas)	Fee (paid in Ether)
General	<p>Every operation in the EVM consumes a pre-defined amount of gas: not changeable by user.</p> <p>Every transaction has a user-specified startGas</p>	Every transaction has a user-specified gas price (current default is $0.02\mu\text{ETH}$ per gas)
At start of transaction	<p>Originator should provide enough fuel: startGas.</p> <p><u>remainingGas</u> = startGas [1a]</p>	Originator must pay for all the fuel. $\text{startGas} \times \text{gas price} = \text{Ether placed in escrow}$ [1b]
Each operation	<u>remainingGas</u> is decreased by operation's gas consumption [2]	Deferred until unsuccessful or successful transaction
Unsuccessful transaction	<p><u>remainingGas</u> is zero and there are operations remaining.</p> <p>This causes an Out of Gas exception and all operations are undone [3]</p>	All the escrowed Ether is paid to miner [4]
Successful transaction [5]	All <u>remainingGas</u> is refunded to originator	$(\text{startGas} - \text{remainingGas}) \times \text{gas price} = \text{fee paid to miner}$ $\text{remainingGas} \times \text{gas price} = \text{refund}$

<https://docs.google.com/spreadsheets/d/1m89CVujrQe5LAFJ8-YAUCCNK950dUzMQPMJBxRtGCqs/edit#gid=0>

Halting Problem - Gas Cost Model



Lab 1. 계정 생성 + 마이닝

Step 1. 파일 준비

GitHub에서 내려받거나

```
% git clone https://github.com/etherstudy/smartcontract/tree/master/mistrun
```

Mac

USB의 SmartContract 디렉토리를 사용자 계정 디렉토리로 복사한다. (사용자 계정을 Jaehoon이라고 하자.)
(복사할 디렉토리 이름 : /Users/Jaehoon/SmartContract)

Windows

USB의 SmartContract 디렉토리를 사용자 계정 디렉토리로 복사한다. (사용자 계정을 Jaehoon이라고 하자.)
(복사할 디렉토리 이름 : C:\Users\Jaehoon\SmartContract)

Step 2. 터미널 띄우기

터미널을 세개 띄운다.

2. geth

```
joehoon:mac$ (master) ls
gethclient.sh  gethconsole.sh  runmist.sh
joehoon:mac$ (master) ./gethclient.sh
WARN [01-28|22:27:19] No etherbase set and no accounts found as default
INFO [01-28|22:27:19] Starting peer-to-peer node           instance=Geth/v1.7.2-stable-1db4ecdc/darwin-amd64/
INFO [01-28|22:27:19] Allocated cache and file handles   database=/Users/jaejoon/troot/mistrun/mac/test-data/geth/chaindata
INFO [01-28|22:27:19] Writing custom genesis block
INFO [01-28|22:27:19] Initialized chain configuration
config={"ChainID": 1337, "Homestead": 0, "DAO": false, "DAOsupport": false}
EIP150: 0 EIPs
WARN [01-28|22:27:19] Starting P2P networking
INFO [01-28|22:27:19] started whisper v.5.0
INFO [01-28|22:27:19] RLPx listener up                   self="enode://7525d167d6735b6387c01705f2c5785661585c3fb33aa0623a09e8b6c54dc49f04178429e5f726a360a219[::]:5760"
INFO [01-28|22:27:19] IPC endpoint opened: test-net/geth.ipc
INFO [01-28|22:27:19] HTTP endpoint opened: http://127.0.0.1:8545
INFO [01-28|22:27:23] Mapped network port                proto=tcp  export=57606
```

Geth client

4. geth

```
> joehoon:mac$ (master) clear
joehoon:mac$ (master) ./gethconsole.sh
Welcome to Geth's JavaScript console!
instance: Geth/v1.7.2-stable-1db4ecdc/darwin-amd64/
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net
> eth.accounts
[]
```

Geth console

5. Mist Helper

```
joehoon:mac$ (master) ls
gethclient.sh  gethconsole.sh  runmist.sh  test-data  test-net
joehoon:mac$ (master) ./runmist.sh
[2018-01-28 22:29:38.127] [INFO] main - Running in production mode: true
[2018-01-28 22:29:38.160] [INFO] EthereumNode - undefined 'fast' 'fast'
[2018-01-28 22:29:38.161] [INFO] EthereumNode - Defaults loaded: geth dev fast
[2018-01-28 22:29:38.505] [INFO] main - Starting in Mist mode
[2018-01-28 22:29:38.666] [INFO] Db - Loading db: /Users/[...]
[2018-01-28 22:29:38.674] [INFO] Windows - Creating commo
[2018-01-28 22:29:38.675] [INFO] Windows - Create seconda
[2018-01-28 22:29:38.710] Mist[46132:7567088] *** WARNING: transparent titlebar. This will break when linking again
[2018-01-28 22:29:38.818] [INFO] updateChecker - Check fo
[2018-01-28 22:29:41.498] [INFO] Windows - Create primary window: main, owner: notset
[2018-01-28 22:29:41.506] [INFO] Windows - Create primary window: splash, owner: notset
[2018-01-28 22:29:41.566] [ERROR] main - Couldn't infer if computer automatically syncs time. Error: checkEnabled is not support
ed on this operating system
at Object.checkEnabled (/Applications/Mist.app/Contents/Resources/app.asar/node_modules/os-timesync/index.js:97:30)
```

mist

Step 3. Mist 실행 (Mac)

모든 터미널에서 /Users/JaeHoOn/SmartContract/mistrun/mac 디렉토리로 이동한다.

1. Geth client 터미널에서 gethclient.sh 를 실행한다.
2. Geth Console 터미널에서 gethconsole.sh 를 실행한다.
3. Mist 터미널에서 runmist.sh 를 실행한다.



Step 3. Mist 실행 (Windows)

모든 터미널에서 C:\Users\Jaehoon\SmartContract\Windows 디렉토리로 이동한다.

1. Geth client 터미널에서 gethclient.bat 를 실행한다.
2. Geth Console 터미널에서 gethconsole.bat 를 실행한다.
3. Mist 터미널에서 runmist.bat 를 실행한다.



Step 4. 계정 생성

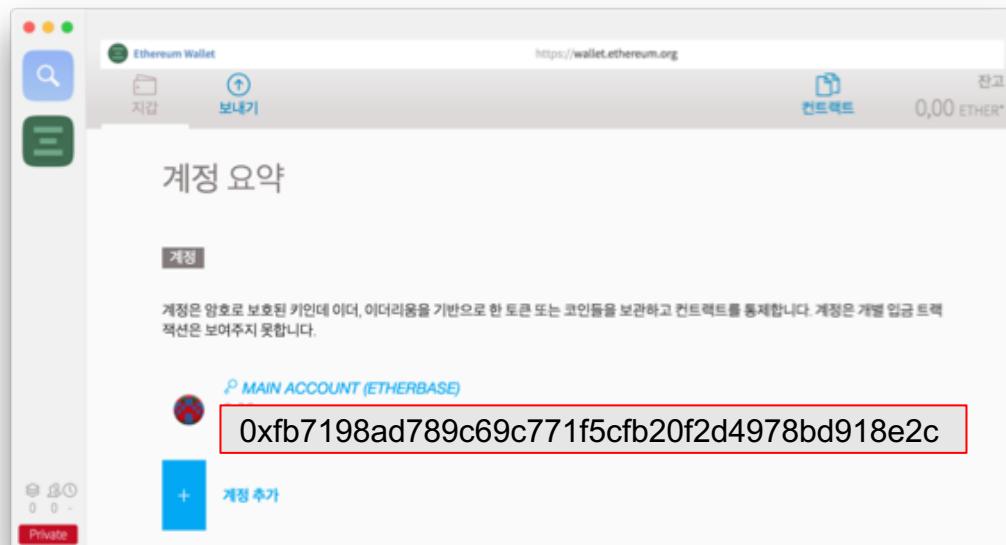
계정 생성 방법

1. Mist에서 생성하기
2. **Geth console**에서 생성하기

```
Geth console에서 계정 생성하기  
> eth.accounts  
[]  
  
> personal.newAccount()  
Passphrase: <Enter your password>  
Repeat passphrase: <Enter your password>  
"0xfb7198ad789c69c771f5cfb20f2d4978bd918e2c"  
  
> eth.accounts  
["0xfb7198ad789c69c771f5cfb20f2d4978bd918e2c"]  
>
```

Step 4. Mist 에서 계정 생성 확인

신규 계정이 생성되면 Mist 에 Main Account 가 표시된다.



Step 5. Mining 제어하기

Geth console에서 마이닝
제어하기

- 마이닝 시작 :
miner.start()
- 마이닝 중지 :
miner.stop()

The image shows two terminal windows. The top window, titled '3. geth', displays the command 'miner.start()' followed by 'null' and 'miner.stop()' followed by 'true'. The bottom window, titled '1. geth', shows a continuous stream of mining logs from the Geth node. These logs include messages like 'INFO [10-28|15:50:35] Mining too far in the future', 'INFO [10-28|15:50:37] Commit new mining work', 'INFO [10-28|15:50:37] Successfully sealed new block', and 'INFO [10-28|15:50:37] Øblock reached canonical chain'. It also shows the creation of potential blocks and the sealing of new blocks, along with their respective hashes and timestamps.

```
> miner.start()
null
> miner.stop()
true

INFO [10-28|15:50:35] Mining too far in the future
INFO [10-28|15:50:37] Commit new mining work
INFO [10-28|15:50:37] Successfully sealed new block
INFO [10-28|15:50:37] Øblock reached canonical chain
INFO [10-28|15:50:37] ↗mined potential block
INFO [10-28|15:50:37] Commit new mining work
INFO [10-28|15:50:38] Successfully sealed new block
INFO [10-28|15:50:38] Øblock reached canonical chain
INFO [10-28|15:50:38] ↗mined potential block
INFO [10-28|15:50:38] Commit new mining work
INFO [10-28|15:50:38] Successfully sealed new block
INFO [10-28|15:50:38] Øblock reached canonical chain
INFO [10-28|15:50:38] ↗mined potential block
INFO [10-28|15:50:38] Mining too far in the future
INFO [10-28|15:50:40] Commit new mining work

1. geth
wait=2s
number=73 txs=0 uncles=0 elapsed=2.004s
number=73 hash=564197..8821d2
number=68 hash=31a365..e27bbc
number=73 hash=564197..8821d2
number=74 txs=0 uncles=0 elapsed=104.746µs
number=74 hash=e6ca5c..213a43
number=69 hash=c11183..eda591
number=74 hash=e6ca5c..213a43
number=75 txs=0 uncles=0 elapsed=127.473µs
number=75 hash=2f620f..784d7e
number=76 hash=1e09f3..6db637
number=75 hash=2f620f..784d7e
wait=2s
number=76 txs=0 uncles=0 elapsed=2.000s
```

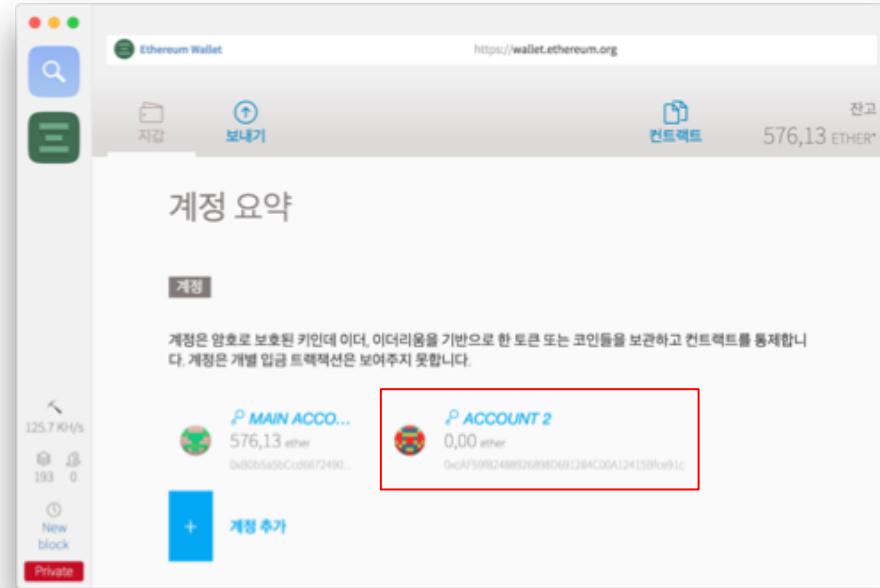
Lab 2. 이더 송금하기

Step 1. 계정 추가하기

다른 계정으로 이더(ether)를 송금해 보자.
이더를 송금하려면 계정이 두 개 이상 필요하다.

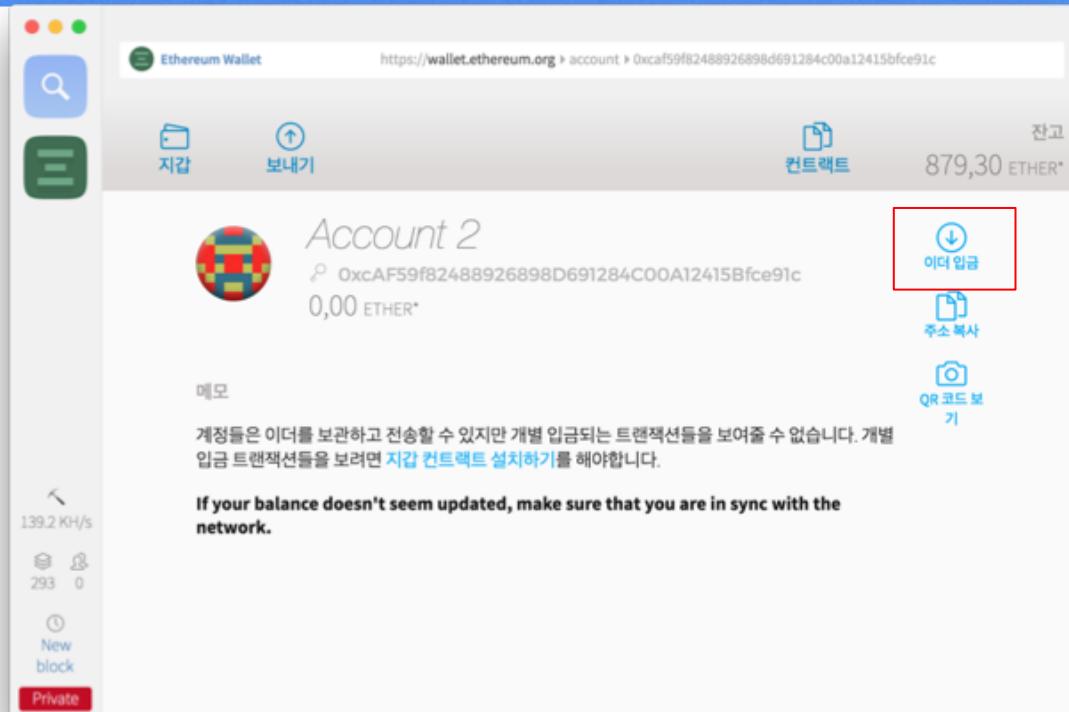
1. 계정을 만들 때, 테스트를 쉽게 하려면 Main 계정과 같은 암호를 사용하는 것이 좋다.
2. Geth console에서 추가로 계정을 하나 만든다. (계정 생성 참조)
3. 생성된 계정이 Wallet에 표시되는지 확인한다.

Wallet에서 “계정 추가”를 클릭해서 계정을 추가할 수도 있다.

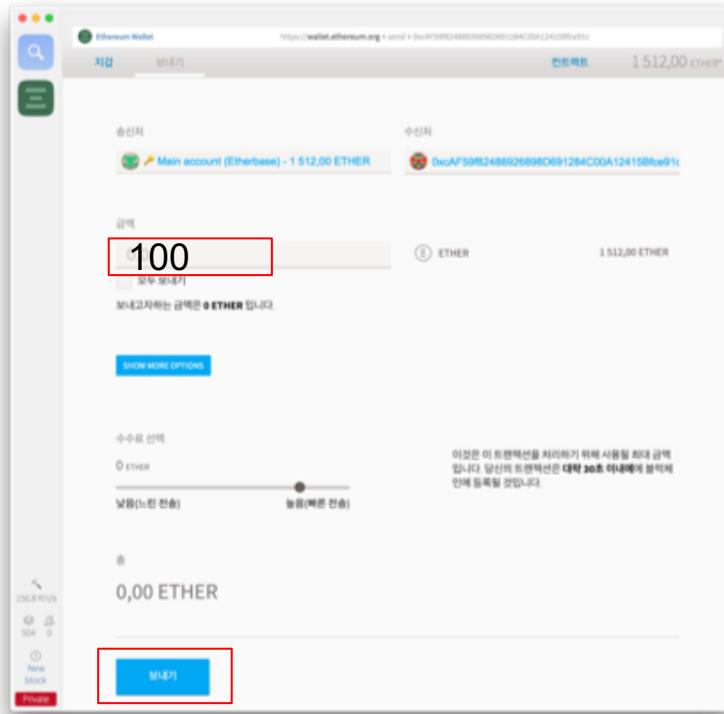


Step 2. 수신 계정 선택

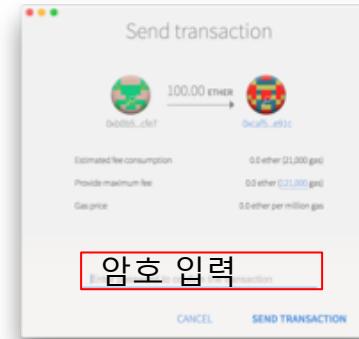
1. 새로 추가한 계정("Account 2")를 클릭한다.
2. 오른쪽 윗부분에 있는 "이더 입금"을 클릭한다.



Step 3. 송금하기

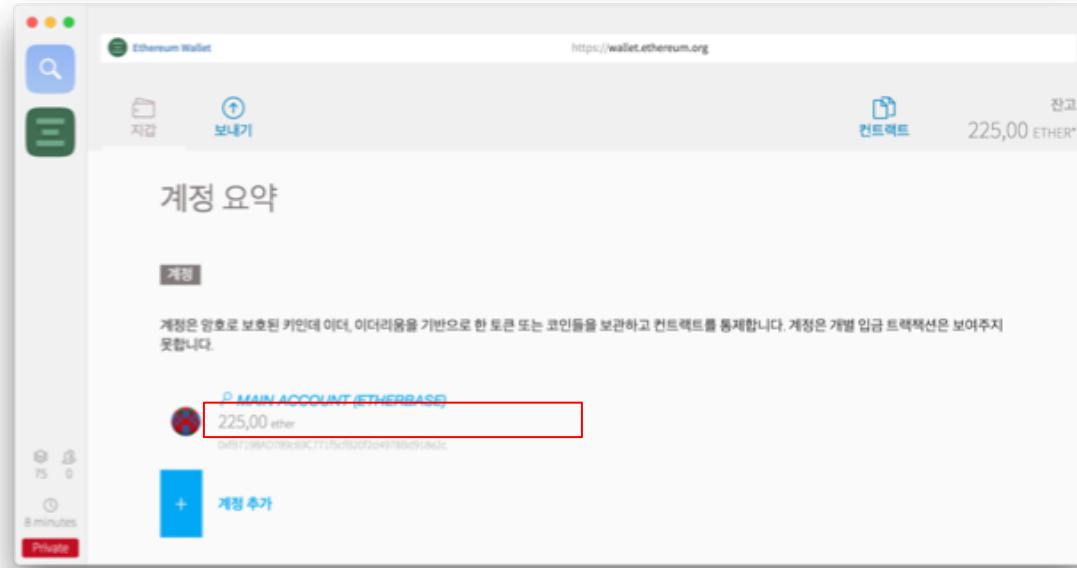


1. Geth console에서 마이닝을 중지한다. (`miner.stop()`)
2. 금액에 “100”(이더)를 입력한다.
3. “보내기”를 클릭한다.
4. “Send Transaction” 화면에서 계정의 비밀번호를 입력한다.
5. “Send Transaction” 을 클릭한다.

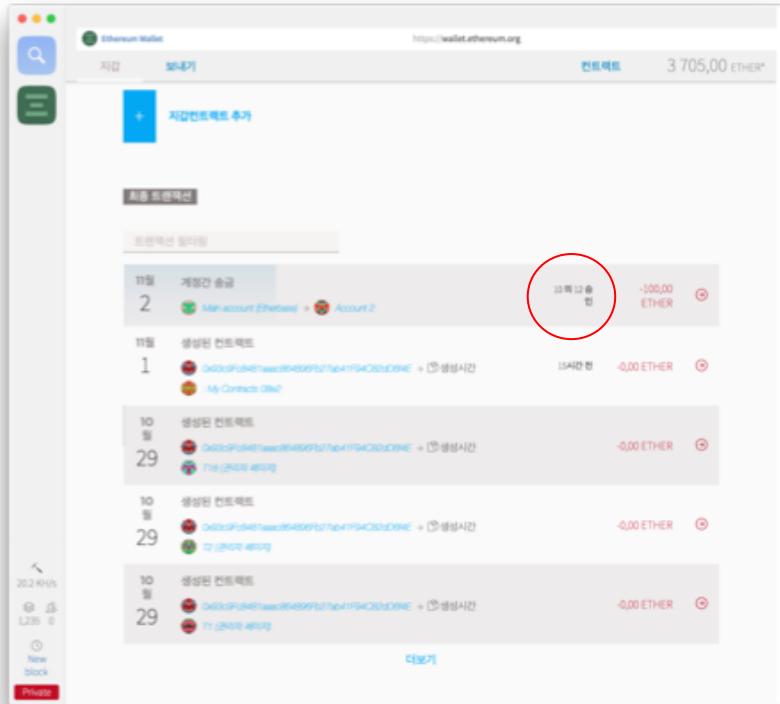


Step 5. Mining 결과확인

マイニング을 시작하면 Main 계정에 Ether 가 쌓이기 시작한다.



Step 4. 송금 진행



송금을 요청하면 자동으로 “지갑” 화면으로 이동한다.

1. 지갑 화면 아래쪽에 송금 요청한 트랜잭션이 표시되는지 확인한다.
2. geth console에서 마이닝을 시작한다. (`miner.start()`)
3. Mist 화면에서 트랜잭션에 대한 동의가 일어나는 과정을 확인한다.

Step 5. 송금 확인

The screenshot shows the Ethereum Wallet interface at <https://wallet.ethereum.org>. The top bar displays 'Ethereum Wallet' and the balance '4 032,00 ETHER'. The main area is titled '최근 트랜잭션' (Recent Transactions) and lists several transactions. Transaction 2 is highlighted with a red circle and a red arrow pointing to it, labeled '클릭해서 상세정보 확인' (Click to view detailed information). The transaction details are as follows:

날짜	계정간 송금	수신자	금액	상세보기
2	계정간 송금	Main account [Etherbase] + Account 2	-0,00 ETHER	(상세보기)
1	생성된 컨트랙트	0x00000000000000000000000000000000 + 0x00000000000000000000000000000000	-0,00 ETHER	(생성시간) 11월 2일 오전 9시 46분 (본인 세이프)
10	생성된 컨트랙트	0x00000000000000000000000000000000 + 0x00000000000000000000000000000000	-0,00 ETHER	(생성시간) 11월 2일 오전 9시 46분 (본인 세이프)
29	생성된 컨트랙트	0x00000000000000000000000000000000 + 0x00000000000000000000000000000000	-0,00 ETHER	(생성시간) 11월 2일 오전 9시 46분 (본인 세이프)
10	생성된 컨트랙트	0x00000000000000000000000000000000 + 0x00000000000000000000000000000000	-0,00 ETHER	(생성시간) 11월 2일 오전 9시 46분 (본인 세이프)
29	생성된 컨트랙트	0x00000000000000000000000000000000 + 0x00000000000000000000000000000000	-0,00 ETHER	(생성시간) 11월 2일 오전 9시 46분 (본인 세이프)

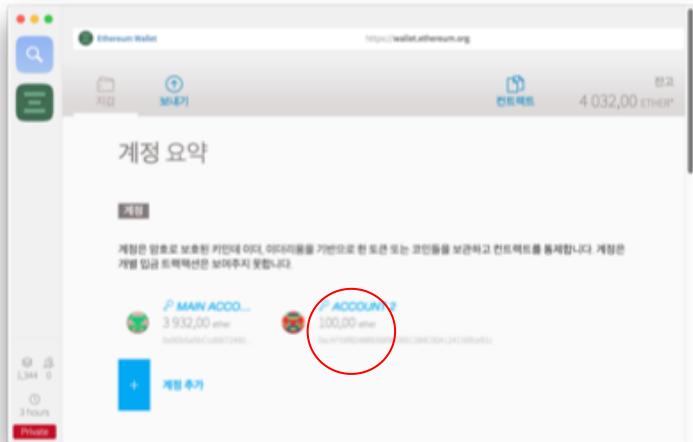
트랜잭션이 완료되면 “승인 정보”가 표시되던 자리에 트랜잭션이 요청된 시간이 표시된다.

트랜잭션을 클릭해서 트랜잭션의 상세 정보를 확인한다.

A detailed view of a transaction is shown in a red-bordered box. The transaction ID is 0xd3ca7d9b5b3c62bbff6d75043db6dd34e9086fa63c19619514a33d5630c1ce83. It was created on November 2, 2017, at 9:46 AM, with 119 confirmations. The transaction details are as follows:

트랜잭션
0xd3ca7d9b5b3c62bbff6d75043db6dd34e9086fa63c19619514a33d5630c1ce83
2017년 11월 2일 목요일 오전 9시 46분 (본인 119 승인)
금액 100,00 ETHER
발신처 Main account [Etherbase]
수신처 Account 2
지급 수수료 0,00 ETHER
가스 사용 21 000 트랜잭션 비용
가스 가격 0,00 ETHER 맥안 가스당
블록 1226 0xe2de842b0cdfa150af45f96a00bdab400dc141...

Step 5. 계좌 잔액 확인



지갑 화면에서 “Account 2” 의 잔액 확인한다.
Geth console에서 mining 멈춘다.

트랜잭션

이더리움 트랜잭션

- EOA(External Owned Account) 가 다른 계정으로 보내는 메시지

트랜잭션에 포함되는 정보

1. 수신자(recipient)
2. 송신자(sender)를 식별할 수 있는 시그너처
3. 송신 금액(value)
4. 데이터 필드(선택사항) : 컨트랙트에 보내는 메시지
5. Startgas : 최대로 소비할 수 있는 가스량
6. Gasprice : 수신자가 지불할 가스당 이더

트랜잭션 종류

- 송금 트랜잭션
- 컨트랙트 생성 트랜잭션 (수신자 주소가 0)
- 컨트랙트 메소드 호출 트랜잭션

Lab 3. Contract 만들기

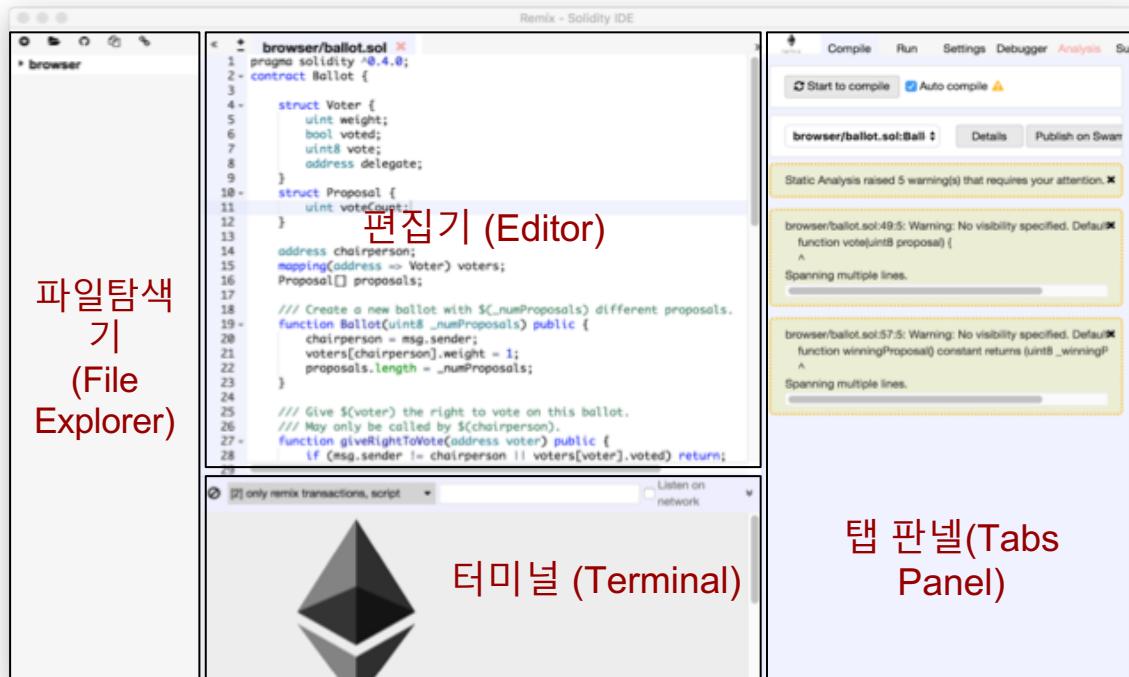
Step 0. Remix IDE 열기



Remix IDE 를 이용해서 스마트 컨트랙을 개발하고 설치할 수도 있다

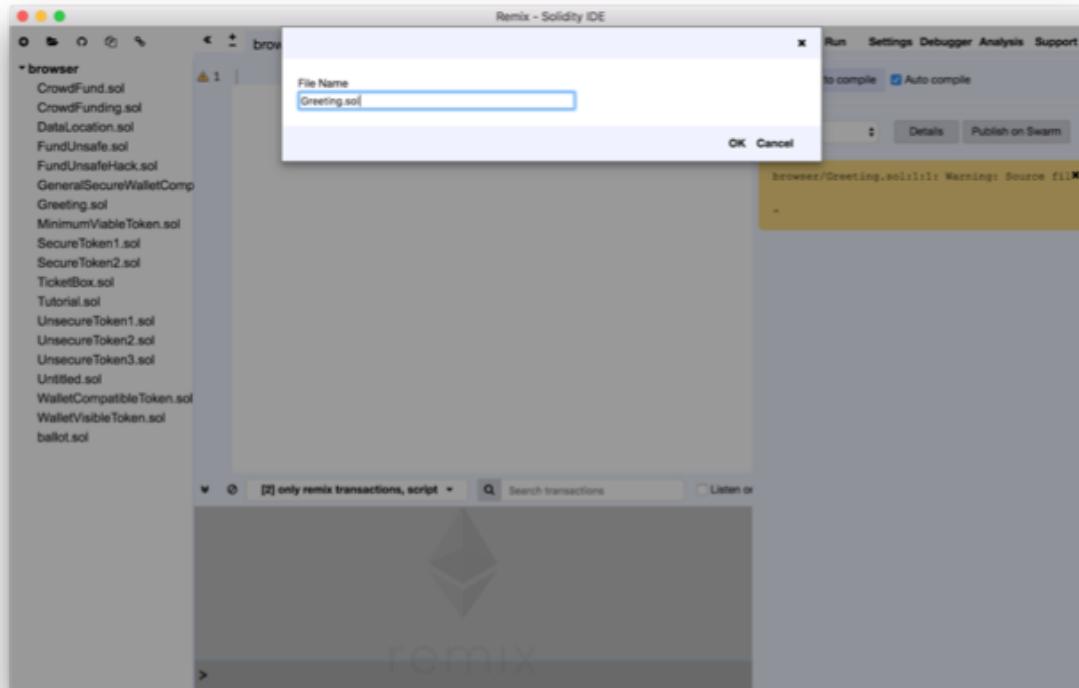
1. Mist 메뉴 창에서 “개발”을 클릭
2. “Remix IDE 열기” 메뉴 클릭

Step 0. Remix IDE 창 구조



Remix IDE는 파일탐색기, 편집기, 터미널, 탭 판넬로 구성된다.

Step 1. 첫번째 Contract 만들기



1. 왼쪽 메뉴에서 \oplus 버튼 클릭
2. File Name 에 “Greeting.sol” 입력하기

Greeting v1

“Hello” 라는 문자열을 반환하는 sayHello 를 가진 컨트랙을 만든다.

Contract 이름 : Greeting

함수 이름 : function sayHello() returns (string)

Greeting v1

```
pragma solidity ^0.4.18;

contract Greeting {
    function sayHello() external pure returns (string) {
        return "hello";
    }
}
```

Version Pragma

표현	설명
<code>^0.4.18</code>	0.x.x 와 호환
<code><1.2.7</code>	1.2.7 미만과 호환
<code><=1.2.7</code>	1.2.7 이하와 호환
<code>>1.2.7</code>	1.2.7 초과와 호환
<code>>=1.2.7</code>	1.2.7 이상과 호환
<code>=1.2.7</code>	1.2.7 만 호환
<code>1.2.7</code>	<code>=1.2.7</code> 과 동일
<code>1.2.7 1.2.8</code>	1.2.7 이나 1.2.8
<code>>=1.2.9 <2.0.0</code>	1.2.9 보다 크거나 같고, 2.0.0 미만

Functions

```
function name(<parameter types>)
  [internal|external|public|private]
  [pure|constant|view|payable]
  [returns (<return types>)] {
  ...
}
```

Functions - State Mutability

State Mutability : `pure` / `constant` / `view` / `payable`

컨트랙트 state 변수에 대한 읽기, 쓰기를 제한

State Mutability	State 읽기	State 쓰기	이더 송금(잔액 변경)
<code>pure</code>	N	N	N
<code>constant</code>	Y	N	N
<code>view</code>	Y	N	N
<code>payable</code>	Y	Y	Y
기본	Y	Y	Y

Functions - 상태 Write

아래의 경우 상태가 변경된다고 가정

1. state variable 에 기록하기
2. event 를 발생시키기
3. 다른 컨트랙 생성하기
4. selfdestruct 사용하기
5. call 을 통해서 Ether 보내기
6. view/pure 함수가 아닌 함수를 부르기
7. EVM 의 call 을 사용하기
8. 특정 opcode 를 포함하는 인라인 assembly 사용하기

Functions - 상태 Read

아래의 경우 상태가 변경된다고 가정

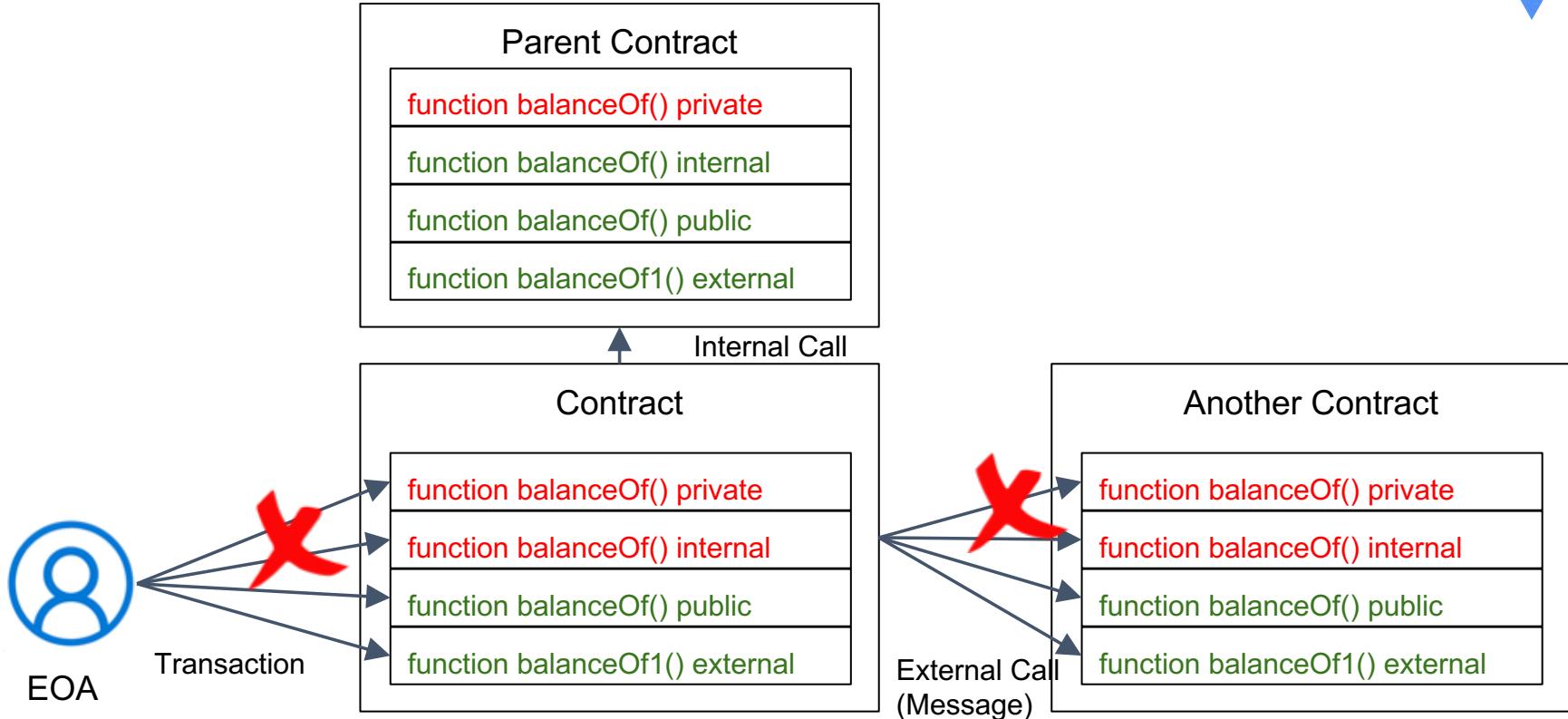
1. state variable 을 읽기
2. this.balance 를 접근하거나 <address>.balance 를 읽기
3. block, tx, msg 의 멤버 접근하기 (msg.data, msg.sig 제외)
4. pure 로 선언되지 않은 함수를 부르기
5. 특정 opcode 를 포함하는 인라인 어셈블리를 사용하기

Function Visibility

Function Visibility 타입 : external, public, internal, private (기본은 public)

Visibility	설명
external	EOA(External Owned Account) 에서만 부를 수 있음 데이터가 클 때 유리함 (CallData 를 메모리로 복사하지 않기 때문)
public	EOA, 다른 컨트랙트, 컨트랙트 내부에서 부를 수 있음 EOA 가 public 메소드를 부르면 많은 비용이 발생함 (CallData 를 메모리로 복사하기 때문에)
internal	컨트랙트(혹은 컨트랙트를 상속하는 컨트랙트) 내부에서만 부를 수 있음
private	컨트랙트 안에서만 부를 수 있음 (상속한 컨트랙트에서는 부를 수 없음)

Function Visibility



Greeting v2

sayHello에서 출력할 문자열을 변경하는 함수 changeHello를 추가한다.
sayHello는 changeHello의 패러미터 값을 출력해야 한다.

Contract 이름 : Greeting

추가할 함수 : function changeHello(string message)

State Variables

컨트랙트 내부에서 정의된 변수들

이더리움의 영구 스토리지에 저장됨 (키/값 구조)

Gas 가 상대적으로 많이 소모됨 (32 바이트당 20,000)

가시성	설명
public	모두가 접근할 수 있고, Getter function 이 자동으로 생성됨 ex) string public name;
internal	컨트랙트(혹은 컨트랙트를 상속하는 컨트랙트) 내부에서만 사용할 수 있음
private	컨트랙트 내부에서만 접근 가능함. 상속한 컨트랙트에서는 사용 불가

Types

Value Types	bool, int8, int16, int24, int32, ..., int256 (int) uint8, uint16, uint24, uint32,..., uint256 (uint) address (20바이트, 이더리움 계정 주소) 고정 크기 byte 배열 : bytes1 (byte), byte2, byte3, ..., bytes32 동적 크기 byte 배열 : bytes, string enum function
Reference Types	배열(array), 구조체(struct) Data location 을 지정할 수 있음 : storage 혹은 memory 데이터를 복사하는 비용이 많이 들 수 있기 때문에 어디에 저장할지 선택해야 함

Value Types

Value Types

- 함수 호출시 값으로 전달됨
- 항상 복사해서 사용됨

Literals

- 불리안 : true, false
- 정수(int) : 1, 2, -16
- 양의정수(uint) : 1, 2, 16
- 실수 : 1., .1, 5/2
- 문자열 : “abc”, “\n”, “\xNN”, “\uNNNN”
- 16진수 : hex"48656c6c6f"
- Address : 0xABF0f2

Reference Types

- 배열, struct
- 복사 비용이 매우 높음 (복사시 gas 를 많이 소모)
- Data Location 을 고려해야 함

Type Conversion - Implicit

Value 타입들 간에만 변환 가능

유실되는 정보가 없을 때만 가능

- `uint8` ⇒ `uint16` : 가능
- `int8` ⇒ `uint256` : 불가능
- `uint` ⇒ `bytes` : 가능
- `bytes` ⇒ `uint` : 불가능
- `uint160` ⇒ `address`

Type Conversion - Explicit

string \Rightarrow bytes

- string hello = "Hello";
- bytes b = bytes(hello);

bytes \Rightarrow string

- string memory hello = "Hello";
- bytes memory helloBytes = bytes(hello);
- string memory hello = string(helloBytes);

Type Deduction

- Implicit 타입 변환은 안 되지만 타입을 강제로 변환할 수 있음
- 단, 타입 변환시 정보가 손실 되기 때문에 손실되는 정보가 의미가 없는지 확인 필요함
- ex)
 - `uint16 a = 0x1234;`
 - `uint8 b = uint8(a);`

bytes

- 임의 길이의 byte 데이터
- byte[] 와 유사하지만 데이터를 패킹한다.
- byte[] 를 사용하는 것보다 bytes 가 실제 메모리를 더 적게 사용한다.
- 길이가 정해진 경우에는 bytes1, ... bytes32 를 사용하라. (gas 소모량이 적다)
 - bytes memory name = new bytes(10);
 - bytes.length

string

- string 은 bytes 와 동일하다.
- length 가 없다. 길이를 알려면 bytes 로 변환해야 한다.
 - bytes(s).length
- index 접근을 할 수 없다. Index 로 접근하려면 bytes 로 변환해야 한다.
 - bytes(s)[7] = 'a';

length

- length : 배열의 원소 수를 반환한다.
- 동적 storage 배열인 경우 length 를 변경하면 storage 의 배열 크기를 변경할 수 있음
- 메모리 배열의 크기는 생성될 때 고정되어 있음

push

- 배열의 마지막에 새로운 원소를 덧붙인다(append).
- 새로운 배열의 길이를 반환한다.
- Dynamic storage 배열과 bytes 에만 push 를 사용할 수 있음

```
contract C {  
    bytes private title;  
  
    function setTitle(string _title) external {  
        title = bytes(_title);  
        title.length = 10;  
        title.push(1);  
    }  
}
```

Memory Array

Allocating

- `uint[] memory a = new uint[](7);`
- `bytes memory b = new bytes(10);`

Method

- `length` 만 사용 가능
- `push` 를 사용할 수 없음

Array Literals

Array Literal

- 고정 크기 배열
- 배열의 타입은 원소들의 공통 타입

예제

- `uint8[3] memory a = [1,2,3];`
- `uint16[3] memory a = [1,2,256];`
- `uint[3] memory b = [uint(1),2,3];`

Greeting v2

```
pragma solidity ^0.4.18;

contract Greeting {
    string hello = "hello";

    function sayHello() public view returns (string) {
        return hello;
    }

    function changeHello(string message) external {
        hello = message;
    }
}
```

Greeting v3

sayHello 에서 국가를 패러미터로 입력 받아서 Korea 인 경우 “안녕” 을 출력한다.

Contract 이름 : **GlobalGreeting**

변경할 함수 : function sayHello(string country) returns (string)

Contract 추가항목

- 상태 변수(state variable) : mapping (string => string) countryToHello;
- 생성자

mapping

Mapping : 다른 언어의 Map 혹은 Dictionary 와 같은 자료구조

- Key 와 Key 에 해당하는 값을 매핑하기 위해 사용
- Map 에 해당 Key 없으면 Value 타입의 기본 값(zero value)를 반환함

```
mapping (address => uint) balanceOf;
```

- balanceOf[addr] = 10;
- uint balance = balanceOf[addr];

```
mapping (address => mapping ( address => uint ) ) transactions;
```

- transactions[from][to] = 10;
- uint y = transactions[from][t];

Greeting v3

```
pragma solidity ^0.4.18;

contract GlobalGreeting {
    string hello = "Hello";

    mapping (string => string) countryToHello;

    function GlobalGreeting() public {
        countryToHello["Korea"] = "안녕";
    }

    function sayHello(string country) external view returns (string) {
        return countryToHello[country];
    }

    function changeHello(string message) external {
        hello = message;
    }
}
```

Greeting v4

여러 나라의 인사 메시지를 추가/삭제할 수 있는 함수들을 추가한다.

추가할 함수

1. function addHello(string country, string message) external
2. Function deleteHello(string country) external

delete

`delete a;`

- 변수 a 타입의 초기 값(zero value)을 a에 할당한다.

타입	변수 선언	delete 후 값
int, uint	<code>int a = 10; uint a = 10;</code>	0
static array	<code>uint[3] a = [1,2,3];</code>	[0,0,0]
dynamic array	<code>uint[] a;</code>	크기가 0인 배열
string	<code>string a = "Hello"</code>	Empty string
struct	<code>GreetingMessage a;</code>	각 필드 타입의 초기 값
mapping		오류

Greeting v5

GlobalGreeting 에 헤어질 때 사용할 수 있는 인사를 추가한다.

함수 : sayGoodbye(string country) external returns (string)

Greeting v6

```
struct GreetingMessage {  
    string hello;  
    string goodbye;  
}
```

이 구조체를 이용하도록 sayHello, sayGoodbye, addHello, addGoodbye 함수를 수정한다.

Greeting v6

구조체 생성 방법

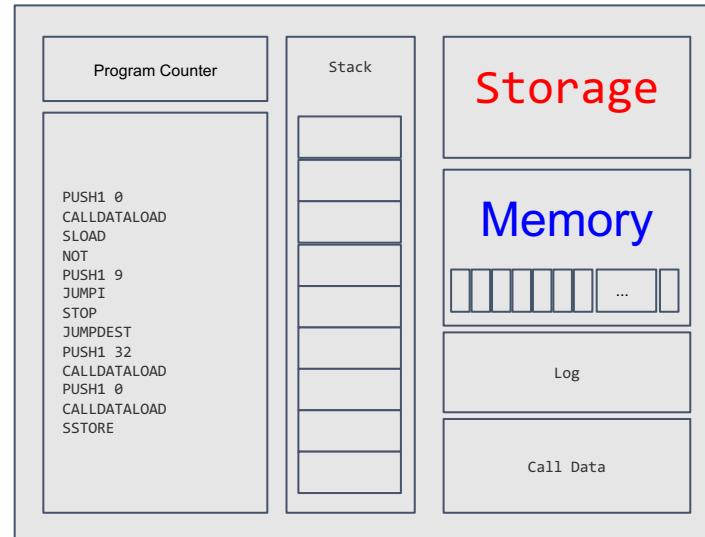
- `GreetingMessage("Hi!","Goodbye!");`
- `GreetingMessage({goodbye:"Goodbye!", hello: "Hi!"});`
- `GreetingMsg memory m;`
 - `m.goodbye = "Goodbye";`
 - `m.hello = "Hello";`

Data Location

모든 complex type(**array, struct**) 은 data location 을 가지고 있다.

- Data Location : memory, storage

변수 컨텍스트	기본	변경
State variable	storage	불가능
Local variable	storage	가능
함수 패러미터	memory	가능
함수 반환값	memory	가능



Data Location

Data location 에 따라 assignment 가 다르게 동작함

독립적인 복사본을 생성하는 assignment

- Storage 와 메모리 사이의 assignment
- State variable 을 변경하는 assignment

참조만을 복사하는 assignment

- Local storage 에 대한 assignment
- Memory 에 저장된 reference type 에서 memory 에 저장된 reference type 으로 assign 하면 복사본을 만들지 않는다.

Data Location

소스 Data Location	타겟 Data Location	복사/참조
memory	memory	GreetingMsg memory m3 =GreetingMsg("Hello","Goodbye!"); GreetingMsg memory m4 = m3; ⇒ 참조
memory	storage	GreetingMsg memory m3 = GreetingMsg("Hello","Goodbye!"); countryToGreeting[""] = m4; ⇒ 복사
storage	memory	GreetingMsg memory x = countryToGreeting["Korea"]; ⇒ 복사
storage	storage	GreetingMsg storage x = countryToGreeting["Korea"]; ⇒ 참조

Lab 4. Contract 제거하기

KillableGreeting

```
pragma solidity ^0.4.18;

contract KillableGreeting {
    address owner;

    function KillableGreeting() public {
        owner = msg.sender;
    }

    function kill() external {
        if (msg.sender == owner) {
            selfdestruct(owner);
        }
    }
}
```

selfdestruct 를 이용하면 더 이상 사용하지 않을 스마트 컨트랙을 제거할 수 있다.

- selfdestruct 는 컨트랙에 남아 있는 잔액을 패러미터로 지정된 계정으로 모두 보낸다.
- selfdestruct 함수는 suicide 의 이름을 변경한 것이다.
- selfdestruct 된 컨트랙은 더 이상 사용할 수 없다.

Special Variables - msg

이더리움의 메시지 데이터 정보가 들어 있는 변수

변수	타입	설명
msg.sender	account	메시지 송신자
msg.value	uint	메시지와 함께 전송된 ether 양
msg.gas	uint	남은 가스
msg.sig	bytes4	calldata 의 첫번째 4 바이트
msg.data	bytes	calldata 전체

Lab 5. Token Contract 만들기

Token

이더리움 에코 시스템에서 유동성있고 교환가능한 상품

- 예 : 코인, 마일리지 포인트, 금 소유 확인증

토큰 컨트랙트

- 토큰을 관리하기 위한 컨트랙트
- 발행 토큰 량, 토큰 장부(계정, 잔액) 관리, 송금, 인출 등을 지원함

2017년 10월 현재 Ethereum ERC20 토큰 수 :
10,565

토큰 컨트랙트

총 발행량 : 1,000

계정	잔액
0xAB...	100
0xCD...	200
0xEF...	300

Minimum Viable Token

```
pragma solidity ^0.4.16;

contract MinimumViableToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

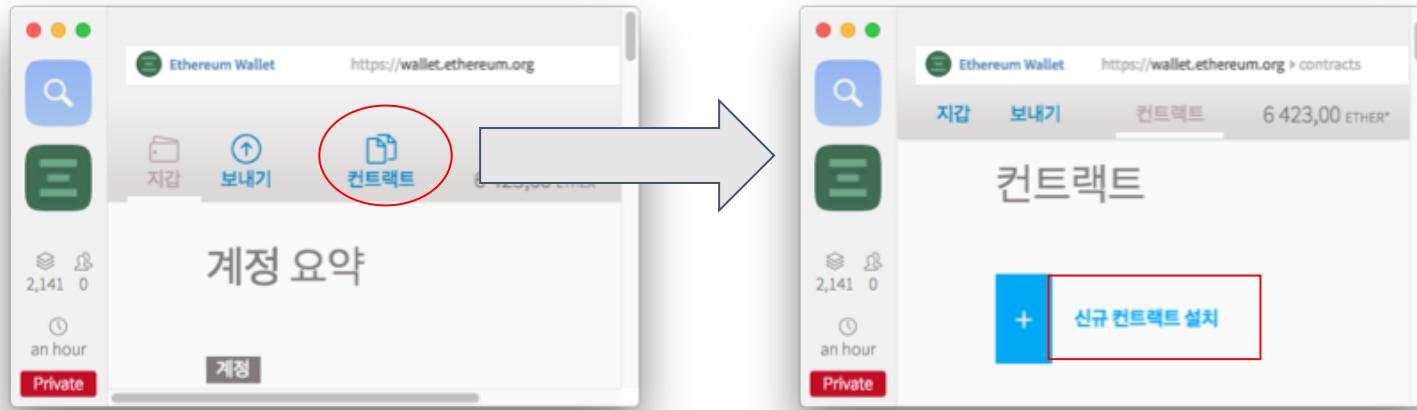
    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MinimumViableToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;           // Give the creator all initial tokens
        }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        balanceOf[msg.sender] -= _value;                 // Subtract from the sender
        balanceOf[_to] += _value;                        // Add the same to the recipient
        }

}
```

Step 1. 컨트랙트 생성 화면으로 이동

1. 컨트랙트 버튼 클릭
2. 신규 컨트랙트 설치 버튼 클릭



Step 2. 소스 코드 복사



MinimumViableToken 코드를 “솔리디티 컨트랙트 소스 코드”로 복사한다.

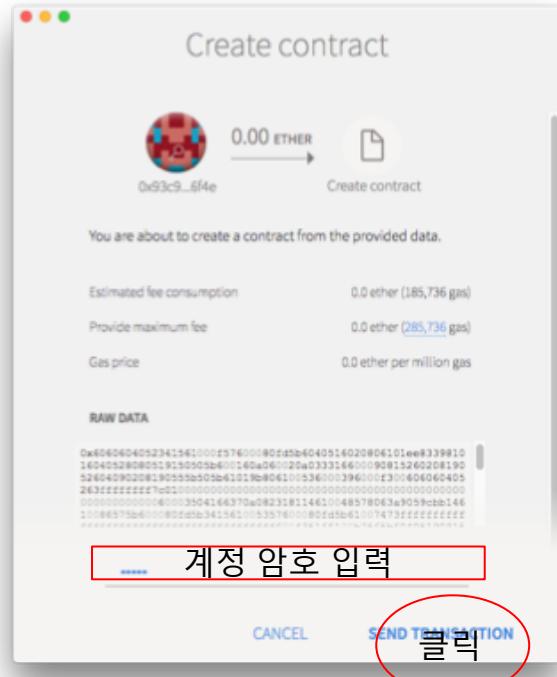
컨트랙트 코드 컴파일에 문제가 없으면, 오른쪽 화면에 “컨트랙트 선택”창이 나타난다.

Step 3. Token 컨트랙트 설치



1. “컨트랙트 선택” 을 클릭한다.
2. “Minimum Viable Token” 선택한다.
3. Initial Supply 를 1000 으로 입력한다.
4. 화면을 아래로 스크롤해서 “설치” 클릭 한다.

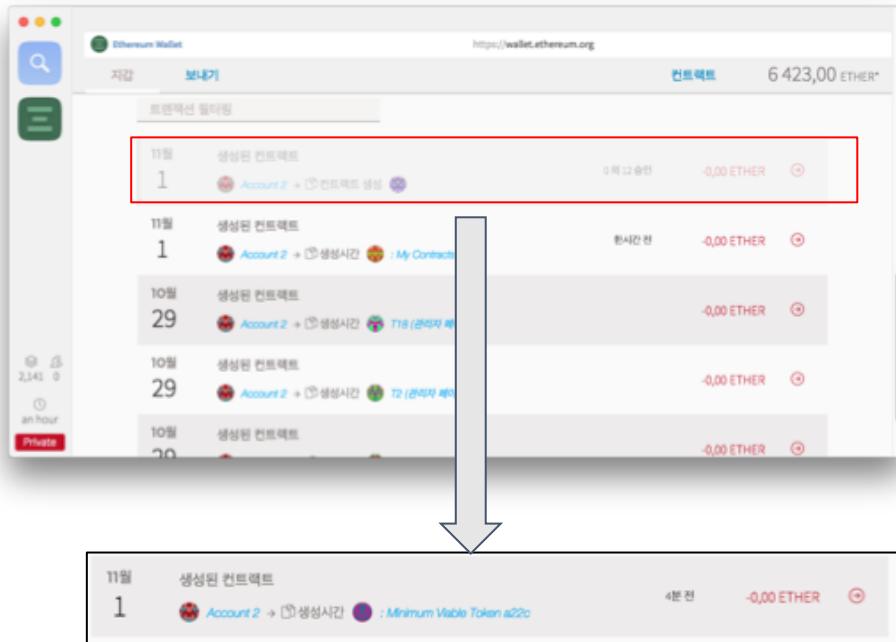
Step 4. Token 컨트랙트 생성 Sign 하기



컨트랙트 생성을 요청하면 “Create Contract” 창이 뜨고 계정의 암호를 묻는다.

1. 계정 암호를 입력한다.
2. SEND TRANSACTION 버튼 클릭한다

Step 5. 토큰 컨트랙트 트랜잭션 확인

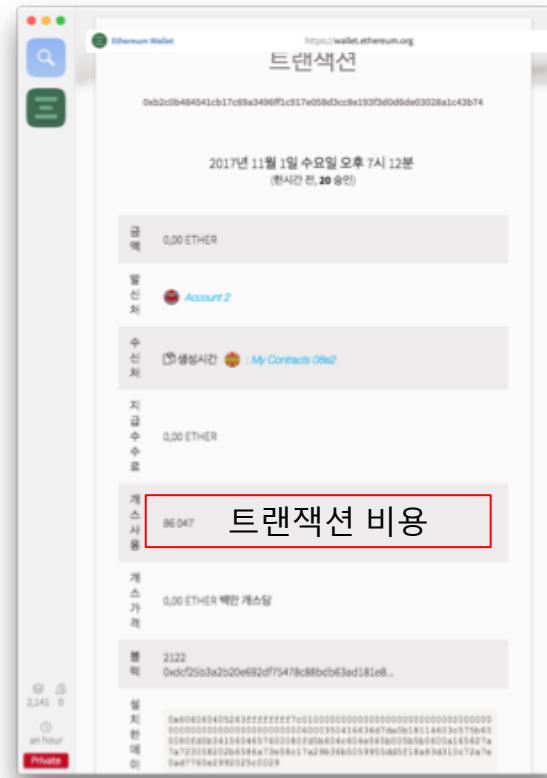


컨트랙트를 생성하면, 지갑에 컨트랙트 생성 트랜잭션이 추가된다.

트랜잭션이 마이닝되지 않으면 트랜잭션이 승인되지 않은 상태로 있다.

1. Geth console에서 “miner.start()” 명령어로 miner를 동작시킨다.
2. 트랜잭션이 동의과정을 거쳐서 승인이 되는 것을 확인한다.
3. 트랜잭션 처리가 완료된 것을 확인한다.

Step 5. 컨트랙트 생성 트랜잭션 확인



토근 컨트랙트 생성 트랜잭션이 완료되면 트랜잭션 상세 내용을 살펴본다.

트랜잭션 비용

- 송금 트랜잭션 : 21,000
- 컨트랙트 생성 트랜잭션 비용 : $21,000 + \alpha$

Gas Price

- <https://etherscan.io/chart/gasprice>



Step 6. Token 컨트랙트 확인



A screenshot of the Ethereum Wallet interface. The top bar shows the URL 'https://wallet.ethereum.org/account/0xA22CeCde214036DA1335c38bb62d18678edB9b26'. The main content area displays the token information for 'Minimum Viable Token a22'. It shows the address '0xA22CeCde214036DA1335c38bb62d18678edB9b26' and the balance '0,00 ETHER'. On the right side, there are several buttons: '이더 충급' (Deposit Ether), '주소 복사' (Copy Address), 'QR 코드 보기' (QR code), and '인터넷 브라우저 보기' (View in Browser). At the bottom left, it says '컨트랙트 정보 접속' (Contract info) and 'Private'.

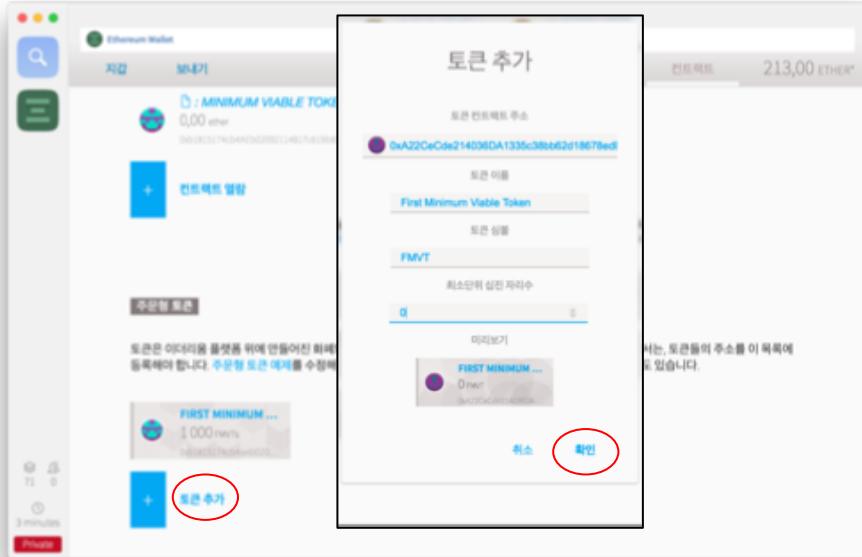
트랜잭션이 종료되면 생성된 컨트랙트를 클릭하여 컨트랙트 상세 화면으로 이동한다.
주소를 더블 클릭한 후 복사한다.

Minimum Viable Token 의 문제

지금 생성한 Minimum Viable Token 은 여러가지 문제를 가지고 있다.

1. 토큰 생성자에게 1000 토큰을 부여했지만, Mist 화면에 표시되지 않는다.
 - a. 토큰 생성자가 자신이 토큰을 소유했다는 사실을 알기 어렵다.
1. 토큰을 전송하는 함수(transfer)에서 조건을 검사하지 않는다.
 - a. 송신자가 자신이 가진 토큰보다 더 많은 토큰을 전송할 수 있다.
 - b. 토큰 수신자가 보유한 토큰에 Overflow 가 발생하면, 토큰이 모두 사라질 수 있다.

Step 7. Token 추가하기

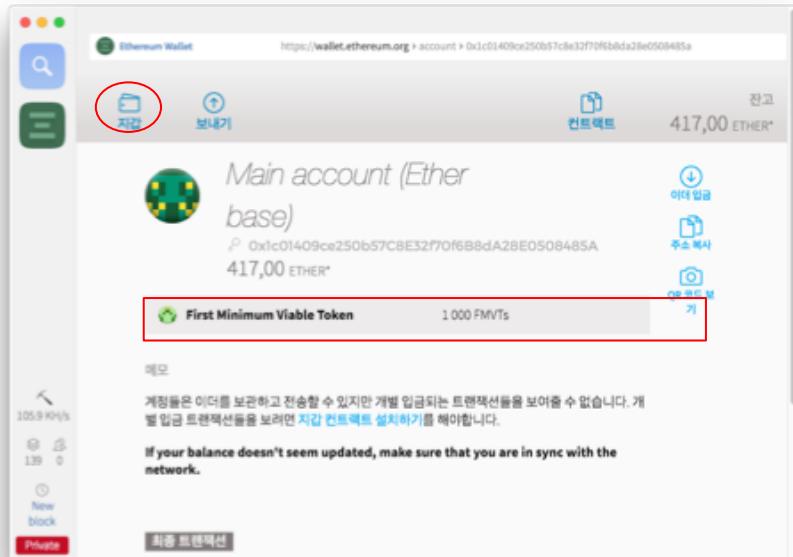


Minimum Viable Token은 지갑에 바로 등록되지 않기 때문에 토큰 잔액을 확인하기 어렵다.

Minimum Viable Token의 잔액을 지갑에서 확인하려면, 토큰을 수동으로 추가해야 한다.

1. “컨트랙트” 링크를 클릭한다.
2. 아래쪽으로 스크롤해서 “토큰 추가” 버튼을 클릭한다.
3. 토큰 컨트랙트 추가 화면에서 다음과 같이 입력한다.
 - a. “토큰 컨트랙트 주소”에 좀전에 복사한 주소를 입력
 - b. 토큰이름 : First Minimum Viable Token
 - c. 토큰심볼 : FMVTs
 - d. 최소단위 십진 자리수 : 0
4. 확인을 클릭한다.

Step 8. 보유 토큰 확인



토큰이 추가되면, Minimum Viable Token 을 생성한 계정에 토큰 잔액이 표시된다.

1. “지갑” 링크를 클릭한다.
2. 컨트랙트를 생성한 계정을 클릭한다.
3. 컨트랙트를 생성한 계정의 “First Minimum Viable Token” 잔액이 **1000 FMVT** 인지 확인한다.

Step 9. 토큰 송금하기

계정	잔액
Main Account	1,000



계정	잔액
Main Account	500
Account 2	500

Step 9. 토큰 송금하기



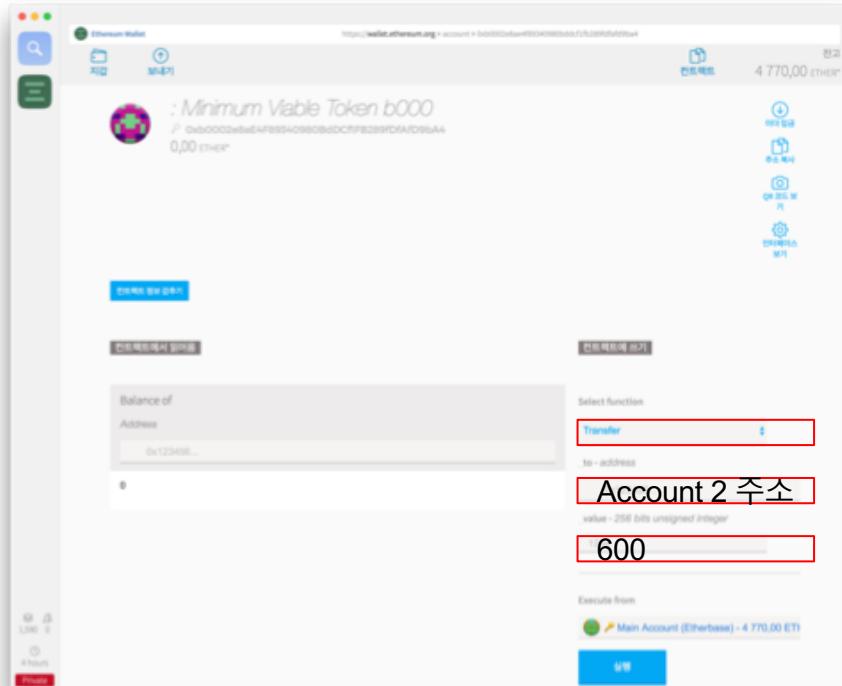
토큰도 이더와 마찬가지 방법으로 다른 계정에 송금할 수 있다. Main 계정에서 Account 2 로 500 FMVTs 를 송금해보자.

1. Account 2 계정을 클릭한다.
2. “이더 입금” 을 클릭한다.
3. “금액” 에 500을 입력한다.
4. “First Minimum Viable Token” 을 클릭한다.

송금이 완료되면 송금 결과확인한다.

1. Main 계정을 클릭해서 잔액을 확인한다.
2. Account 2 계정을 클릭해서 잔액을 확인한다.

Step 10. 컨트랙트 함수 부르기



토큰 컨트랙트 함수를 불러서 토큰을 전송할 수도 있다. Main 계정에서 “Account 2”로 토큰을 100개 전송해 보자.

1. “컨트랙트”를 클릭한다.
2. “Minimum Viable Token” 컨트랙트를 클릭한다.
3. “컨트랙트에 쓰기”에서 “함수 선택”을 클릭한다.
4. “to - address”에 “Account 2” 계정의 주소를 입력한다.
5. “value”에 600을 입력한다.
6. “Execute from”에서 “Main Account”를 선택한다.
7. 실행을 클릭한다.

Step 9. 토큰 송금하기

계정	잔액
Main Account	500
Account 2	500



계정	잔액
Main Account	?
Account 2	1,100

Lab 6. Wallet Visible Token

Wallet Visible Token

```
pragma solidity ^0.4.16;

contract WalletVisibleToken {
    mapping (address => uint256) public balanceOf;

    event Transfer(address _from, address _to, uint _value);

    function WalletVisibleToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;          // Give the creator all initial tokens
    }

    function transfer(address _to, uint256 _value) {
        balanceOf[msg.sender] -= _value;                // Subtract from the sender
        balanceOf[_to] += _value;                       // Add the same to the recipient
        Transfer(msg.sender,_to,_value);
    }
}
```

Events

Step 1. Wallet Visible Token 만들기

Mission : Wallet Visible Token 컨트랙트를 생성한다.

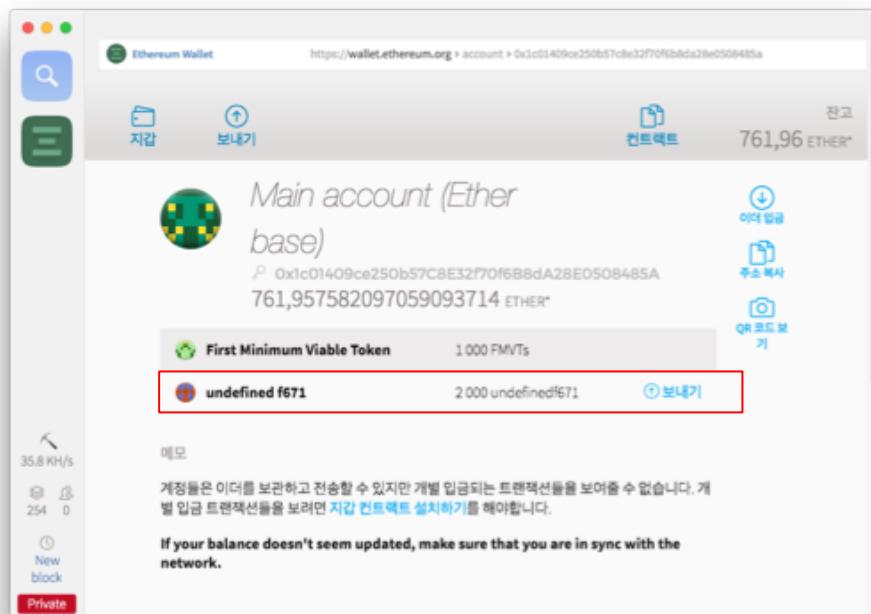
(과정은 Minimum Viable Token 생성과 동일하다.

)

- Initial Supply 는 “2000” 으로 지정

토큰 생성이 완료되면 Main 계정 토큰 잔액을 확인한다.

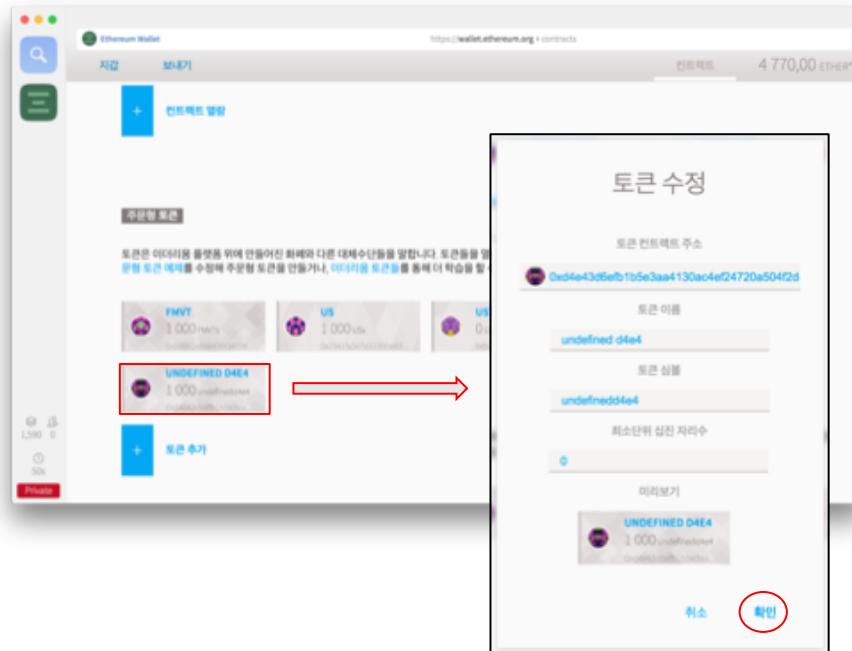
- 토큰 이름, 토큰 심벌 등 토큰에 대한 정보를 지정하지 않았기 때문에, undefined 로 표시된다.



Step 2. 토큰 정보 수정

새로 생성된 토큰 정보를 수정한다.

1. “컨트랙트”를 클릭한다.
2. “주문형 토큰”에서 “UNDEFINED ...” 인 토큰을 클릭한다.
3. 토큰 수정 화면에서 다음과 같이 입력한다.
 - a. 토큰 이름 : Wallet Visible Token
 - b. 토큰 심벌 : WVTs
 - c. 최소단위 십진수 : 0
4. 확인을 클릭해서 토큰 정보를 등록한다.



Step 3. 토큰 송금

Main Account에서 Account 2로 100 토큰을 송금하자

계정	잔액
Main Account	1,000

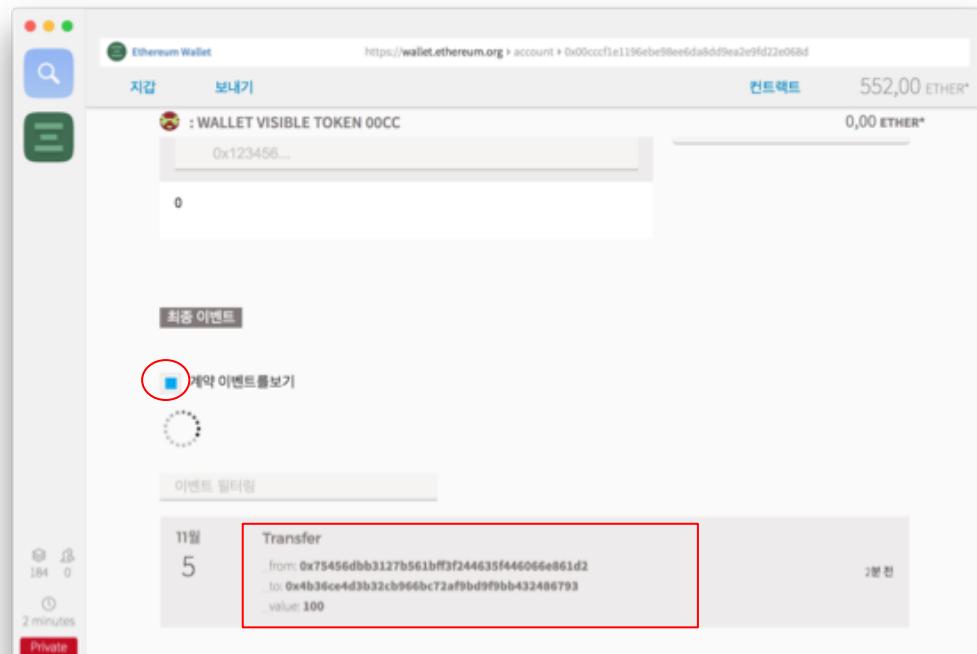


계정	잔액
Main Account	900
Account 2	100

Step 4. 트랜잭션 이벤트 확인

다음과 같이 트랜잭션 이벤트를 확인한다.

1. “Wallet Visible Token” 컨트랙트를 선택한다.
2. 화면을 아래쪽으로 스크롤해서 “계약 이벤트를 보기”를 클릭한다.
3. 방금전에 송금할 때 발생한 Event 가 기록되었는지 확인한다.

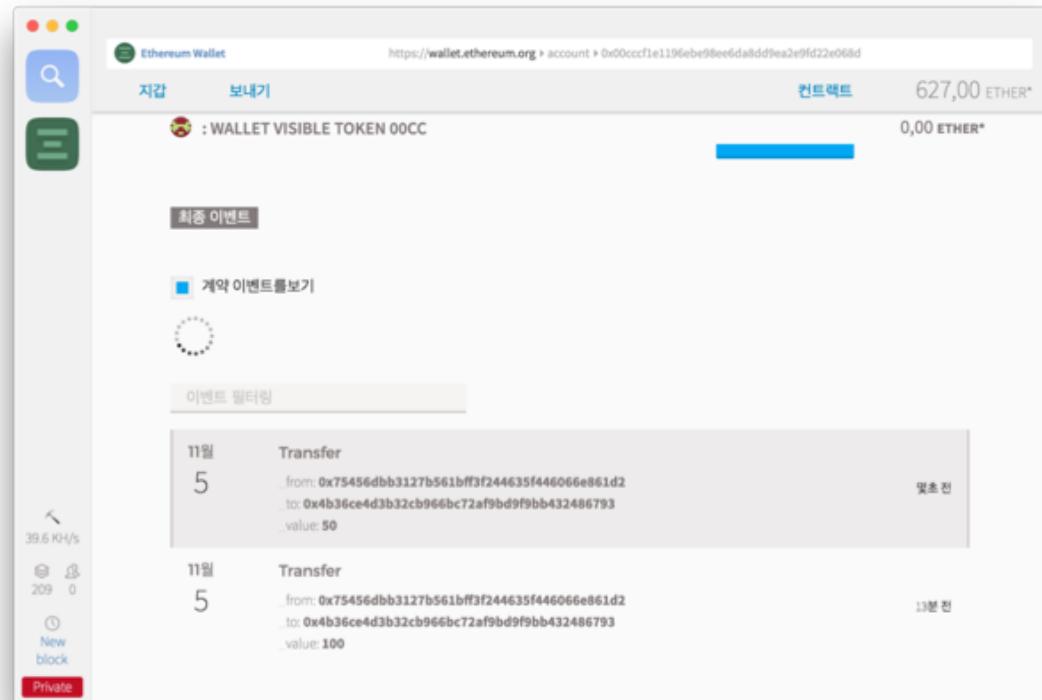


Step 5. Transfer 함수 실행

Wallet Visible Token 의 Transfer 함수를 이용해서 “Account 2”로 50 토큰을 송금해 보자.

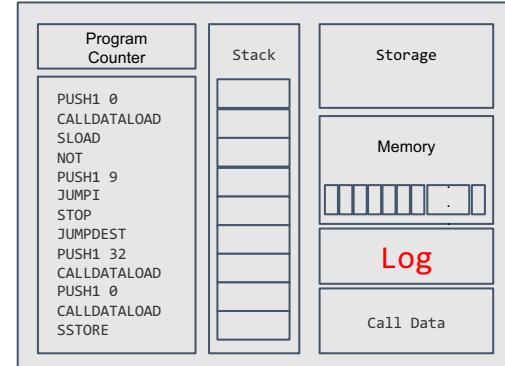
- “이더 입금”에서 전송한 이벤트와 Transfer 함수를 부른 이벤트가 동일한 것을 확인할 수 있다.

확인이 끝나면, “계약 이벤트를 보기” 옵션을 끈다.



Events

- EVM logging 기능을 이용하도록 설계된 장치
- 블록체인의 transaction log 에 패러미터와 함께 기록된다.
- 컨트랙트 내부에서는 접근할 수 없음
- 정의: 컨트랙트 안에서 선언
 - **event** Transfer(address _from, address _to, uint _value);
 - **event** Transfer(address **indexed** _from, address **indexed** _to, uint _value);
- 사용: 컨트랙트 함수에서
 - Transfer(_from, _to, _value);



Events

1. 트랜잭션을 생성한 Application 에 결과값을 반환하는 용도
2. 트랜잭션을 생성한 Application 을 Trigger 하기 위해서
3. 히스토리 데이터를 저장하기 위한 용도
 - a. Log 영역은 Storage 영역에 비해 비용이 아주 저렴
 - b. Event 의 패러미터 세개까지 indexed 를 지정할 수 있음
 - c. indexed 패러미터를 이용해서 검색에 사용할 수 있음

Require & Revert

```
pragma solidity ^0.4.16;

contract UnsecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) return;
        balanceOf[_to] += _value;

        if (balanceOf[msg.sender] < _value) return;
        balanceOf[msg.sender] -= _value;
    }
}
```

```
pragma solidity ^0.4.16;

contract UnsecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) return;
        if (balanceOf[msg.sender] < _value) return;

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

Require & Revert

```
pragma solidity ^0.4.16;

contract UnsecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) return;
        if (balanceOf[msg.sender] < _value) return;

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

```
pragma solidity ^0.4.16;

contract SecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) revert();
        balanceOf[_to] += _value;

        if (balanceOf[msg.sender] < _value) revert();
        balanceOf[msg.sender] -= _value;
    }
}
```

Require & Revert

```
pragma solidity ^0.4.16;

contract SecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) revert();
        balanceOf[_to] += _value;

        if (balanceOf[msg.sender] < _value) revert();
        balanceOf[msg.sender] -= _value;
    }
}
```

```
pragma solidity ^0.4.16;

contract SecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) revert();
        if (balanceOf[msg.sender] < _value) revert();

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

Require & Revert

```
pragma solidity ^0.4.16;

contract SecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        if (balanceOf[_to] + _value < balanceOf[_to]) revert();
        if (balanceOf[msg.sender] < _value) revert();

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

```
pragma solidity ^0.4.16;

contract SecureToken {
    mapping (address => uint256) public balanceOf;

    function UnsecureToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;
    }

    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]);

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

Lab 7. Mist 호환 토큰

Wallet Compatible Token

```
pragma solidity ^0.4.16;

contract WalletCompatibleToken {
    string public constant name = "Wallet Compatible Token";
    string public constant symbol = "WCTs";
    uint8 public constant decimals = 18;

    mapping (address => uint256) public balanceOf;

    event Transfer(address _from, address _to, uint _value);

    function WalletCompatibleToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply;          // Give the creator all initial tokens
    }

    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value);      // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                // Subtract from the sender
        balanceOf[_to] += _value;                      // Add the same to the recipient
        Transfer(msg.sender, _to, _value);
    }
}
```

Step 1. Wallet Compatible Token 만들기

Mission : Wallet Compatible Token 컨트랙트를 생성한다.

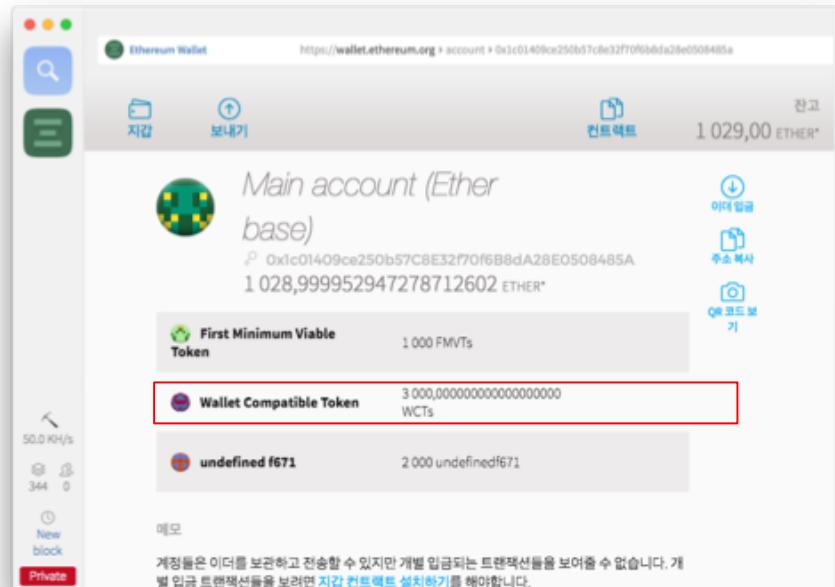
(과정은 Minimum Viable Token 생성과 동일하다.

)

- Initial Supply 는
“3,000,000,000,000,000,000” 으로 지정

토큰 생성이 완료되면 Main 계정 토큰 잔액을 확인한다.

토큰 잔액이 왜 “3 000” 밖에 되지 않을까?



Decimals

Solidity 는 부동 소수점 연산을 제공하지 않고 정수만을 사용해서 계산함

decimals 는 소수점 자리수를 정의함

사용자에게 토큰을 표시할 때 소수점 자리수를 이용해서 보여줌

Token	Decimals	Total Tokens
10,000	0	10,000
10,000	1	1,000.0
10,000	2	100.00
1,000,000,000,000,000,000	18	1.00000 00000 00000 000

Decimals

Ether는 decimals이 18이다.

- 이더는 모두 wei 단위로 저장되고 관리된다.
- Szabo, Finney, Ether로 지정해도 실제로는 wei 단위로만 기록된다.

단위	Wei
wei	1
K wei	1,000 wei
M wei	1,000,000 wei
G wei	1,000,000,000 wei
Szabo	1,000,000,000,000 wei
Finney	1,000,000,000,000,000 wei
Ether	1,000,000,000,000,000,000 wei

Lab 8. Mist 호환 토큰2

Wallet Compatible Token

```
pragma solidity ^0.4.16;

contract WalletCompatibleToken {
    string public name;
    string public symbol;
    uint8 public decimals;

    mapping (address => uint256) public balanceOf;

    event Transfer(address _from, address _to, uint _value);

    function WalletCompatibleToken(string tokenName,string tokenSymbol,uint8 decimalUnits,uint256 initialSupply) {
        name = tokenName;
        symbol = tokenSymbol;
        decimals = decimalUnits;
        balanceOf[msg.sender] = initialSupply;          // Give the creator all initial tokens
    }
}

...
```

Step 1. Wallet Compatible Token 만들기

Mission : Wallet Compatible Token 컨트랙트를 생성한다.

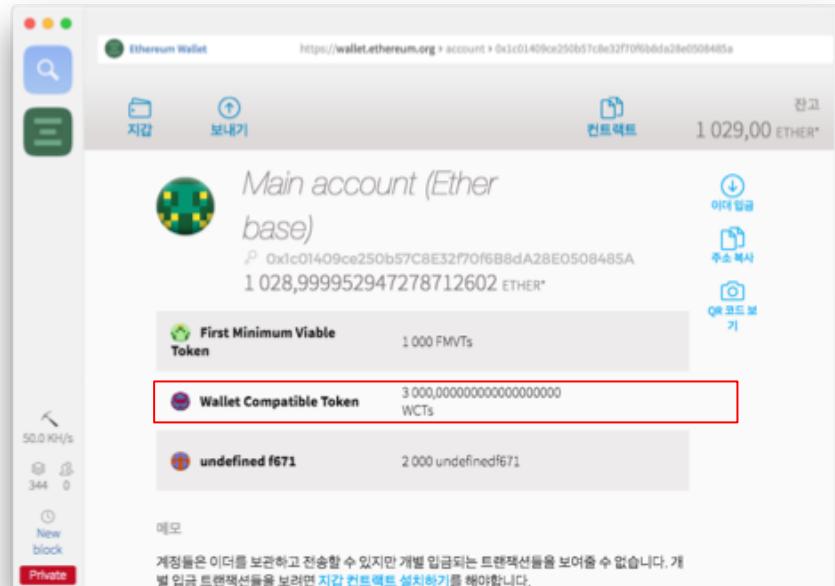
(과정은 Minimum Viable Token 생성과 동일하다.

)

- Initial Supply 는
“3,000,000,000,000,000,000” 으로 지정

토ken 생성이 완료되면 Main 계정 토큰 잔액을 확인한다.

토Ken 잔액이 왜 “3 000” 밖에 되지 않을까?



ERC20 Token Standard

```
contract ERC20 {  
    function totalSupply() constant returns (uint totalSupply);  
    function balanceOf(address _owner) constant returns (uint balance);  
    function transfer(address _to, uint _value) returns (bool success);  
    function transferFrom(address _from, address _to, uint _value) returns (bool success);  
    function approve(address _spender, uint _value) returns (bool success);  
    function allowance(address _owner, address _spender) constant returns (uint remaining);  
  
    event Transfer(address indexed _from, address indexed _to, uint _value);  
    event Approval(address indexed _owner, address indexed _spender, uint _value);  
}
```

Lab 9. CrowdFund

Crowd Fund



1980년 5월, 서울 택시운전사. “광주? 돈 워리, 돈 워리! 아이 베스트 드라이버”
택시운전사 만섭(송강호)은 외국손님을 태우고 광주에 갔다 통금 전에 돌아오면
밀린 월세를 갚을 수 있는 거금 10만원을 준다는 말에
독일기자 피터(토마스 크레취만)를 태우고 영문도 모른 채 길을 나선다.

광주 그리고 사람들. “모르겄어라, 우덜도 우덜한테 와 그라는지...”
어떻게든 택시비를 받아야 하는 만섭의 기지로 검문을 뚫고 겨우 들어선 광주.
위험하니 서울로 돌아가자는 만섭의 만류에도
피터는 대학생 재식(류준열)과 황기사(유해진)의 도움 속에 촬영을 시작한다.
그러나 상황은 점점 심각해지고 만섭은 집에 혼자 있을 딸 걱정에 점점 초조해지
는데...

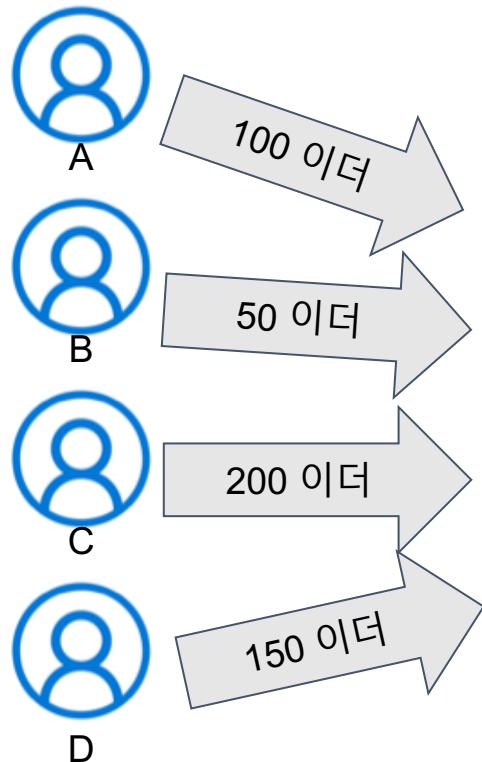
CrowdFund 컨트랙트

여러분은 영화 "택시 운전사" 제작을 위한 크라우드 펀드 스마트 컨트랙트를 만들어야 합니다. 택시 운전사 를 제작하기 위해서는 최소 500 이더가 필요합니다.

요구사항

1. 모금 기간은 분단위로 지정된다.
2. EOA 가 펀드에 청약하면, 청약한 금액에 해당되는 증표(토큰)을 발행한다.
3. 청약 금액과 증표 비율은 1:1로 한다.
4. 모금 기간안에 모금 목표 금액을 달성하면 영화제작사에게 모금금액을 모두 송금한다.
5. 모금 기간이 지나도 모금 목표 금액이 달성되지 않으면 청약자들에게 청약한 금액을 모두 반환한다.

Crowd Fund



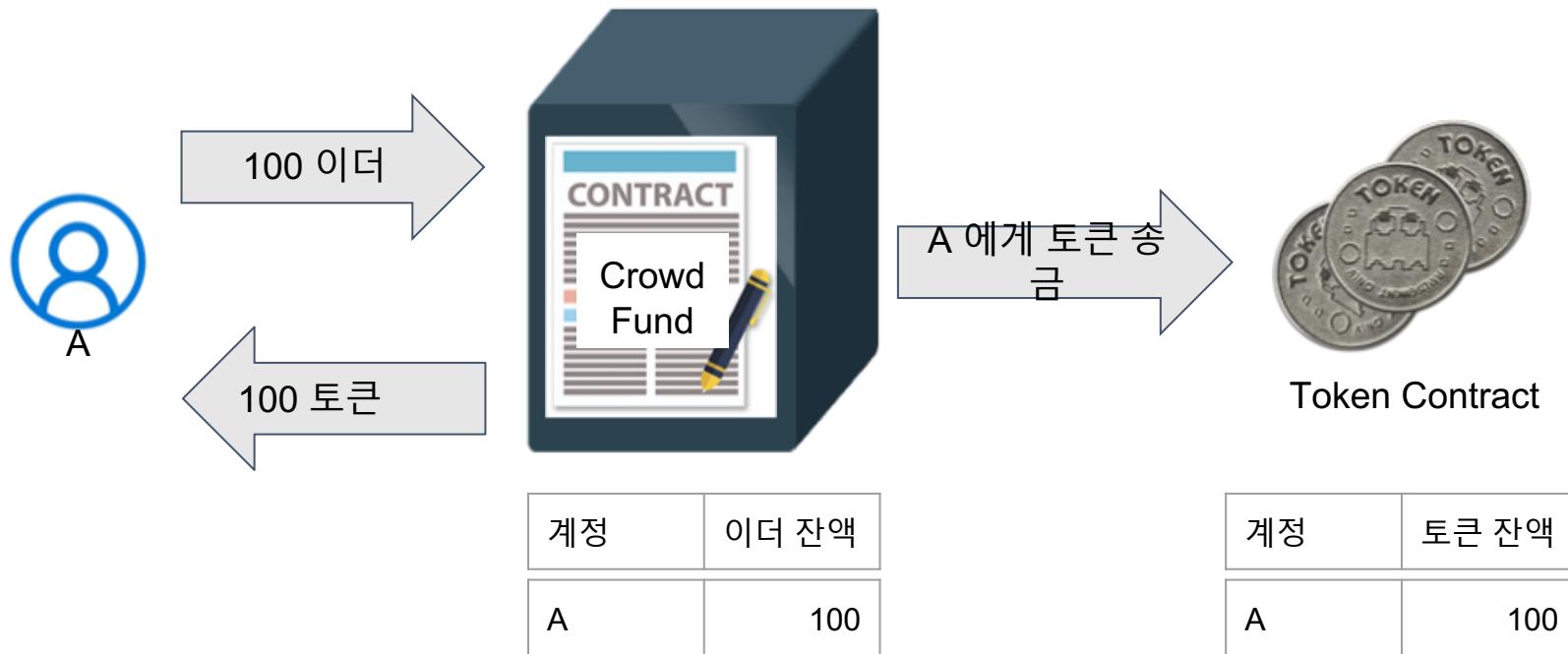
목표금액: 500 이더

계정	잔액
A	100
B	50
C	200
D	150



택시 운전사
제작사
(beneficiary)

Crowd Fund



Step 1. TaxiDriver Token 생성하기

1. Main 계정에서 WalletCompatibleToken.sol 을 이용해서 Taxi Driver Fund 를 위한 토큰을 만든다.
 - a. Name : TaxiDriver Token
 - b. Symbol : TDs
 - c. Decimal Units : 0
 - d. Initial Supply : 1000
2. Main 계정의 “TaxiDriver Token” 잔액이 1,000인지 확인한다.
3. TaxiDriver Token 의 주소를 복사해 둔다.

Step 2. CrowdFund 컨트랙트 생성하기

설치할 컨트랙트를 선택하세요

Crowd Fund

컨스트럭터 입력값들

If successful send to - address

0x123456...

Funding goal in ethers - 256 bits unsigned integer

1234

Duration in minutes - 256 bits unsigned integer

1234

Ether cost of each token - 256 bits unsigned integer

1234

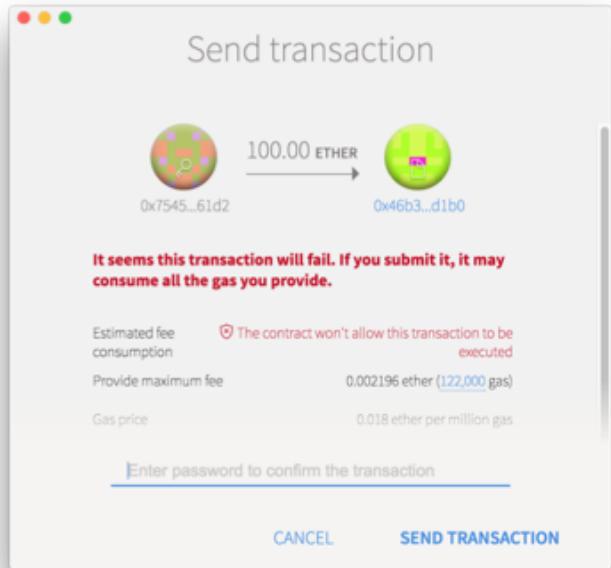
Address of token used as reward - address

0x123456...

CrowdFund.sol 을 이용해서 새로운 컨트랙트를 생성한다. 생성자 패러미터는 다음과 같이 입력한다.

1. If successful send to : Main account 가 아닌 계정
2. Funding goal in ethers : 200
3. Duration in minutes : 10
4. Ether cost of each token : 1
5. Address of token used ad reward : Step1 에서 만든 컨트랙트의 주소

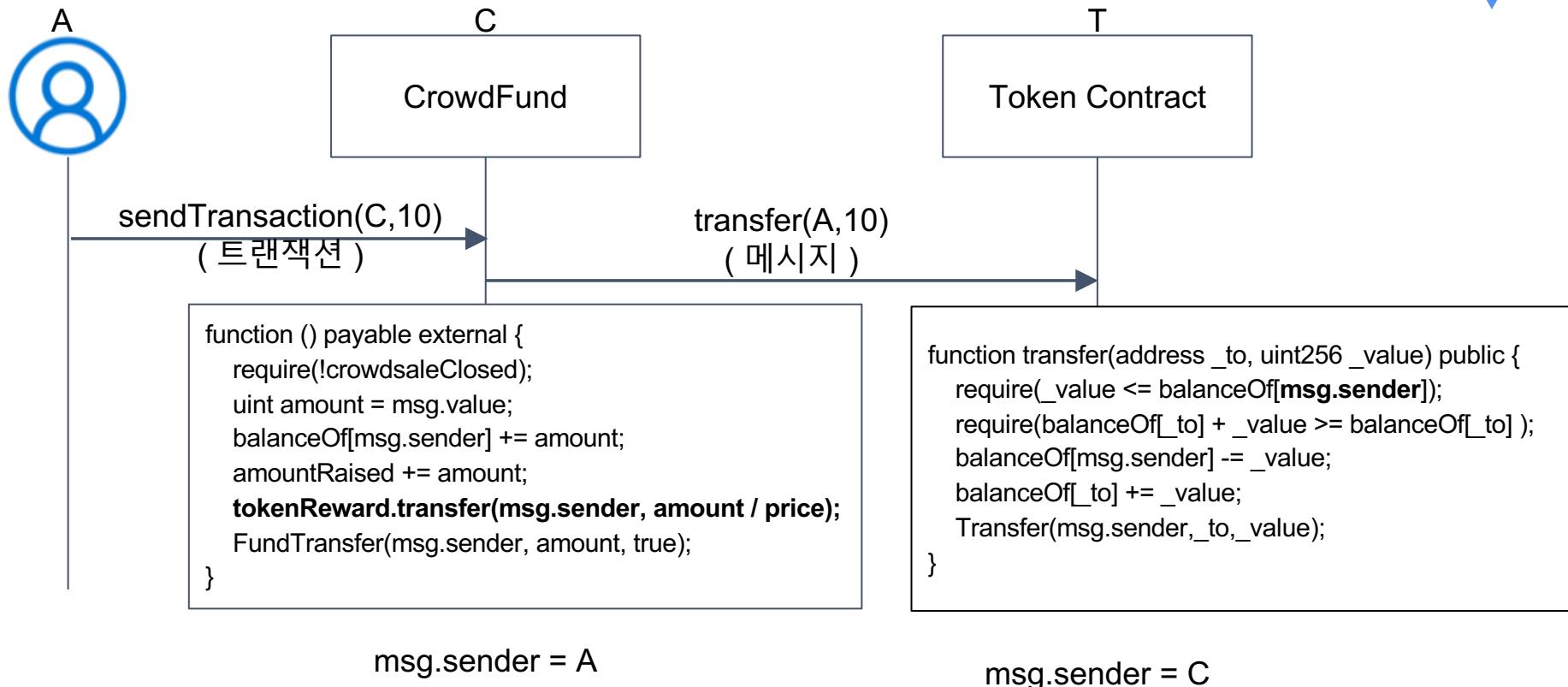
Step 3. CrowdFund 로 송금



“Account 2” CrowdFund 컨트랙트로 송금을 시도하자.

1. “컨트랙트”를 클릭해서 컨트랙트 목록을 표시한다.
2. CrowdFund 컨트랙트를 클릭한다.
3. “이더 입금”을 클릭한다.
4. 금액에 100을 입력한다.
5. “전송” 버튼을 클릭한다.
6. 왼쪽과 같은 Send Transaction 화면이 뜨는지 확인한다.

Step 3. CrowdFund 로 송금

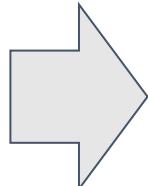


Function Modifier

- 반복되는 논리를 한 곳에서 관리하기 위한 방법
 - python 의 decorator
 - Java 의 AOP 와 유사함
- Ether 를 전송받는 함수에는 반드시 payable modifier 를 지정해야 함

```
function open() {
    require(msg.sender == owner);
    open = true;;
}
```

```
function close() {
    require(msg.sender == owner);
    open = false;
}
```



```
modifier onlyOwner {
    require(msg.sender == owner);
}

function open() onlyOwner {
    open = true;
}

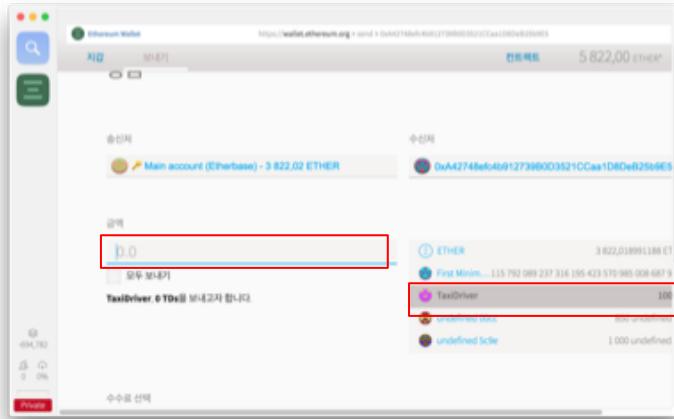
function close() onlyOwner {
    open = false;
}
```

Function Modifier - payable

- payable modifier 를 지정해야 이더를 수신할 수 있다
- payable 을 지정하지 않은 함수에 이더를 전송하면 트랜잭션이 실패한다.

```
modifier payable() {
    this.balance += msg.value;
    _;
}
```

Step 4. CrowdFund 에 토큰 송금

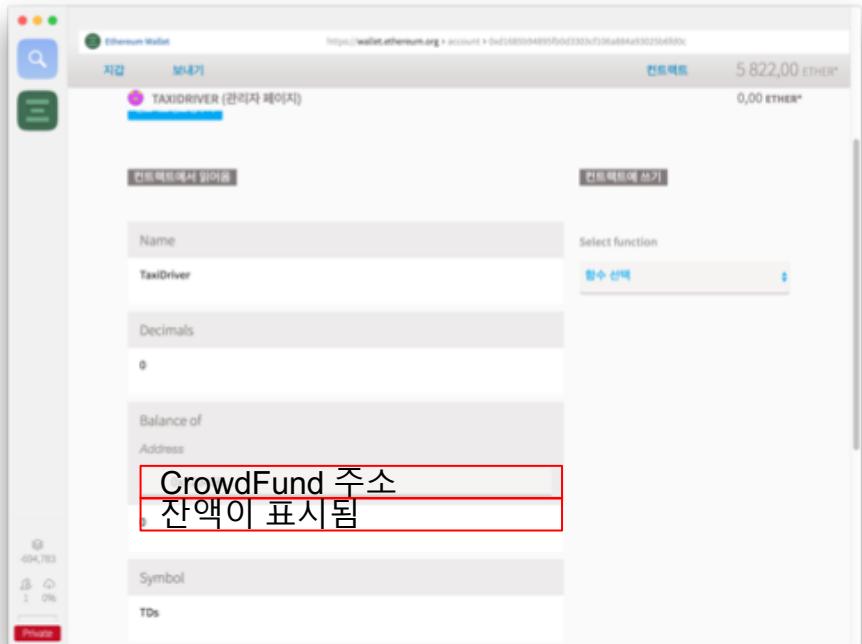


EOA에서 CrowdFund에 송금했을 때, TaxiDriver Token을 보상으로 제공하려면 CrowdFund가 TaxiDriver Token을 보유해야 한다.

보유한 토큰의 50%인 500 토큰을 CrowdFund 컨트랙트로 송금한다.

1. 컨트랙트 버튼을 클릭한다.
2. CrowdFund 컨트랙트를 생성한다.
3. 이더 입금을 클릭한다.
4. “송신처” 계정을 선택한다. (TaxiDriver Token을 생성한 계정)
5. 금액에 500을 입력한다.
6. 전송을 클릭한다.

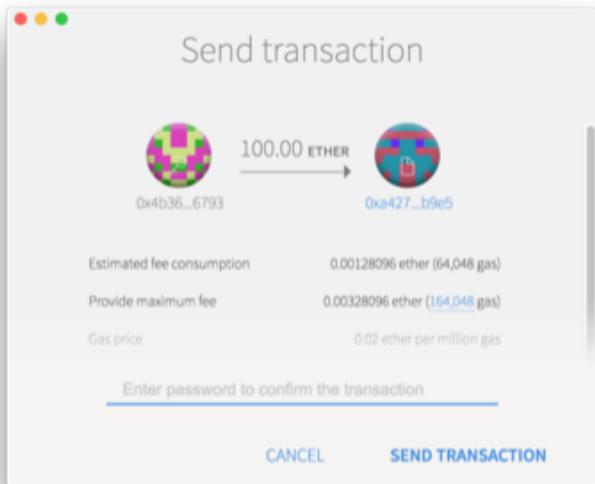
Step 5. CrowdFund 보유 토큰 확인



아래와 같이 CrowdFund 보유 토큰을 확인한다.

1. CrowdFund 컨트랙트의 주소를 복사한다.
2. “컨트랙트”를 클릭해서 컨트랙트 목록을 표시 한다.
3. “TaxiDriver” 컨트랙트를 클릭한다.
4. “Balance of”에 CrowdFund 컨트랙트 주소를 복사한다.
5. 보유 잔액이 500인지 확인한다.

Step 6. CrowdFund 청약



“Account 2” CrowdFund 컨트랙트로 송금을 시도하자.

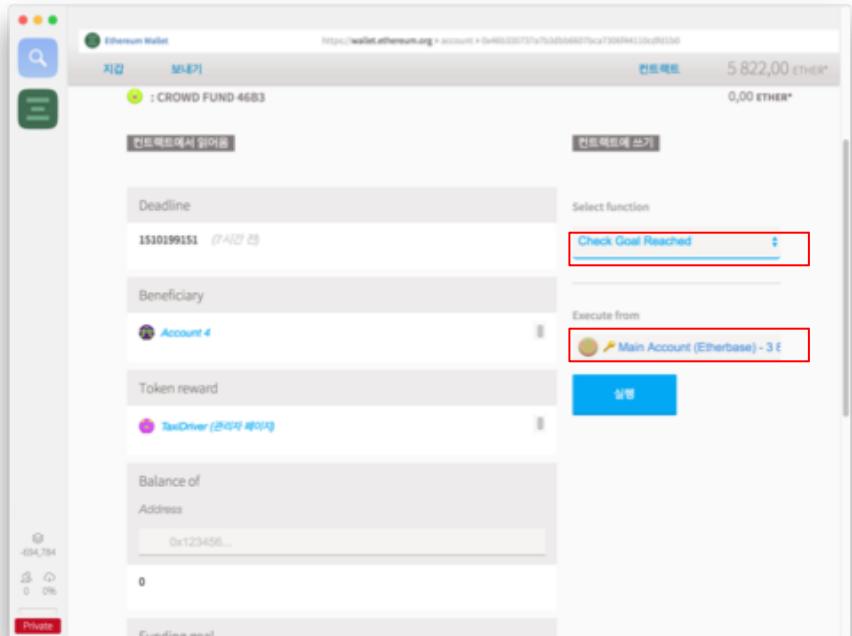
1. “컨트랙트”를 클릭해서 컨트랙트 목록을 표시한다.
2. CrowdFund 컨트랙트를 클릭한다.
3. “이더 입금”을 클릭한다.
4. 금액에 100을 입력한다.
5. “전송” 버튼을 클릭한다.
6. Send Transaction 화면이 오류 없이 뜨는지 확인한다.
7. 암호를 입력하고 “Send Transaction” 을 클릭한다.

Step 7. CrowdFund 청약 결과 확인

“지갑” 을 클릭한 후 “Account 2” 계정에서 다음을 확인한다.

1. “100 TDs” 가 입금되었다.
2. “100” 이더가 감소하였다.

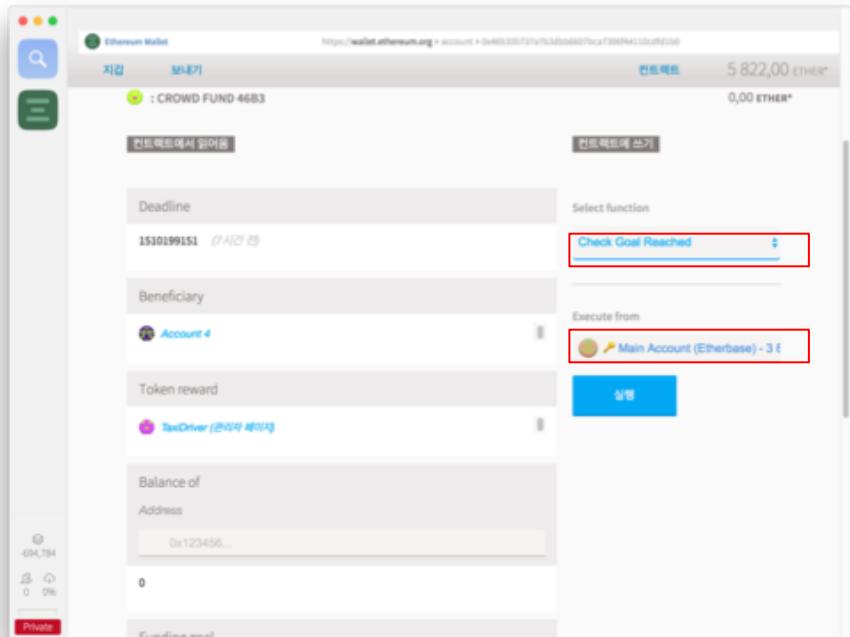
Step 8. CrowdFund 인출하기



기간이 만료되면 CrowdFund 컨트랙트의 CheckGoalReached 를 부른다.

1. “컨트랙트” 버튼을 클릭한다.
2. “함수 선택” 을 클릭한다.
3. “CheckGoalReached” 를 선택한다.
4. “실행” 을 클릭한다.

Step 8. CrowdFund 인출하기



기간이 만료되면 CrowdFund 컨트랙트의 CheckGoalReached 를 부른다.

1. “컨트랙트” 버튼을 클릭한다.
2. “함수 선택” 을 클릭한다.
3. “SafeWithdrawal” 를 선택한다.
4. “실행” 을 클릭한다.

Special Keywords

	Special Keywords
Ether Units	<code>wei, finney, szabo, ether</code> <ul style="list-style-type: none">• <code>1 ether == 1000 finney</code>• <code>1 finney == 1000 szabo</code>• <code>1 szabo == 1000,000,000,000 wei</code>
Time Units	<code>seconds, minutes, hours, days, weeks, years</code> <ul style="list-style-type: none">• <code>1 == 1 seconds</code>• <code>1 minutes == 60 seconds</code>• <code>1 hours == 60 minutes</code>• <code>1 days == 24 hours</code>• <code>1 weeks == 7 days</code>• <code>1 years == 365 days</code>

Special Variables - block & transaction

변수	타입	설명
block.coinbase	address	현재 블록의 마이너 주소
block.difficulty	uint	현재 블록의 difficulty
block.gaslimit	uint	현재 블록의 gaslimit
block.number	uint	현재 블록 넘버
block.timestamp	uint	현재 블록의 timestamp (초)
block.blockhash(blockNumber)	bytes32	주어진 블록의 해시값. 현재 블록을 포함해서 최근 256개 블록에서만 동작
변수	타입	설명
tx.gasprice	uint	트랜잭션의 gas price
tx.origin	address	트랜잭션 송신자

Lab 10. Remix 를 이용한 Debugging

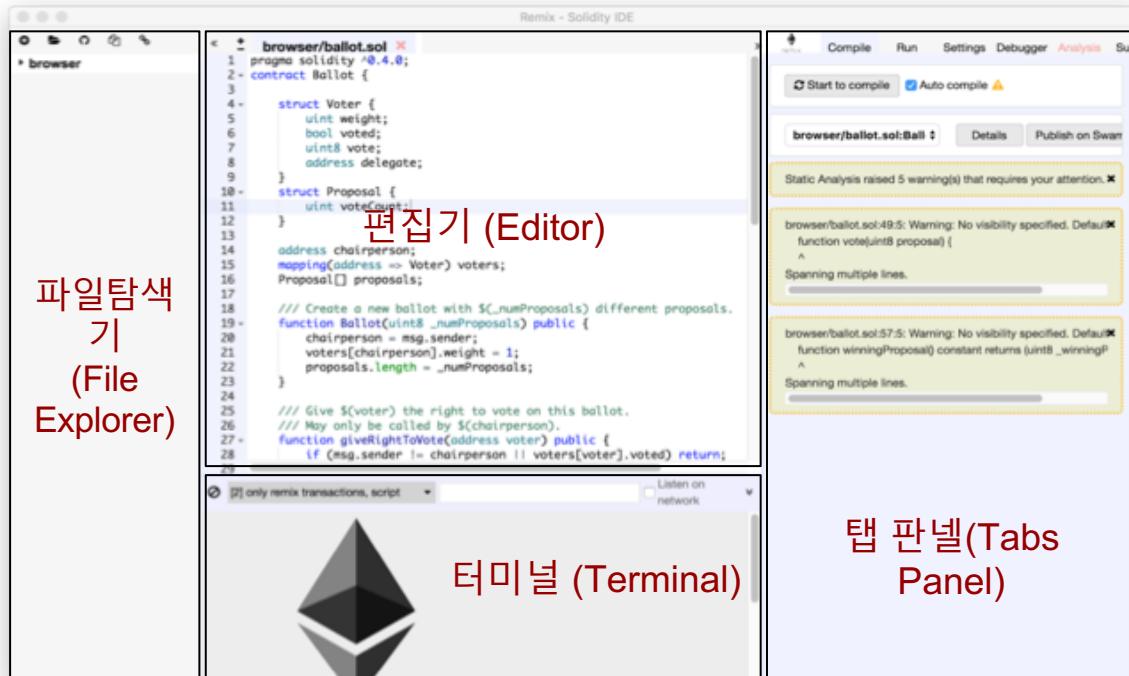
Step 0. Remix IDE 열기



Remix IDE 를 이용해서 스마트 컨트랙을 개발하고 설치할 수도 있다

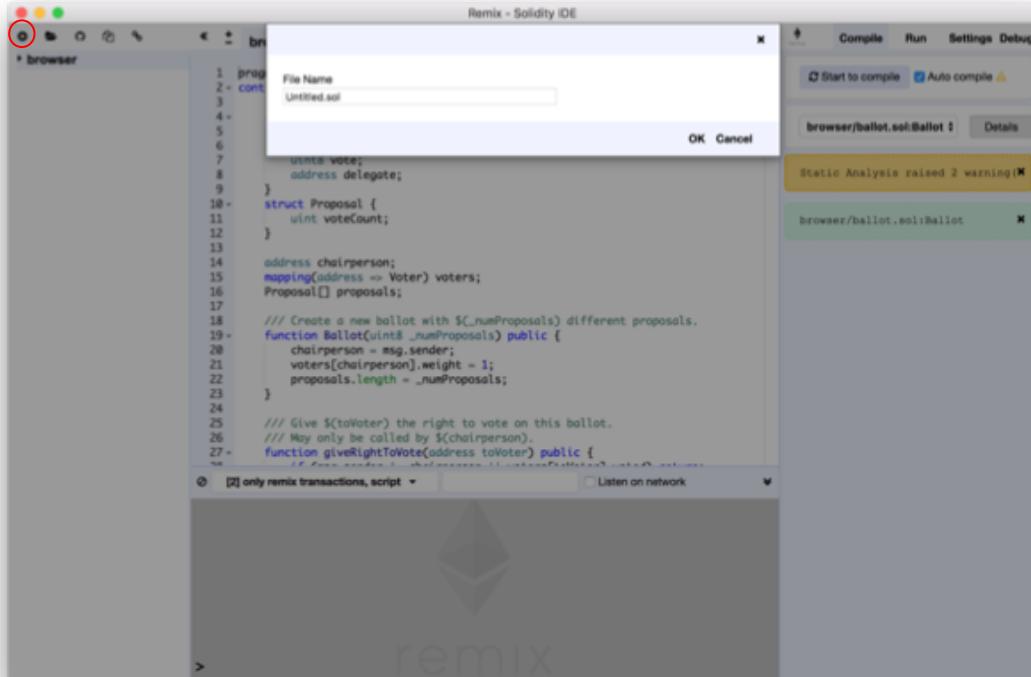
1. Mist 메뉴 창에서 “개발”을 클릭
2. “Remix IDE 열기” 메뉴 클릭

Step 0. Remix IDE 창 구조



Remix IDE는 파일탐색기, 편집기, 터미널, 탭 판넬로 구성된다.

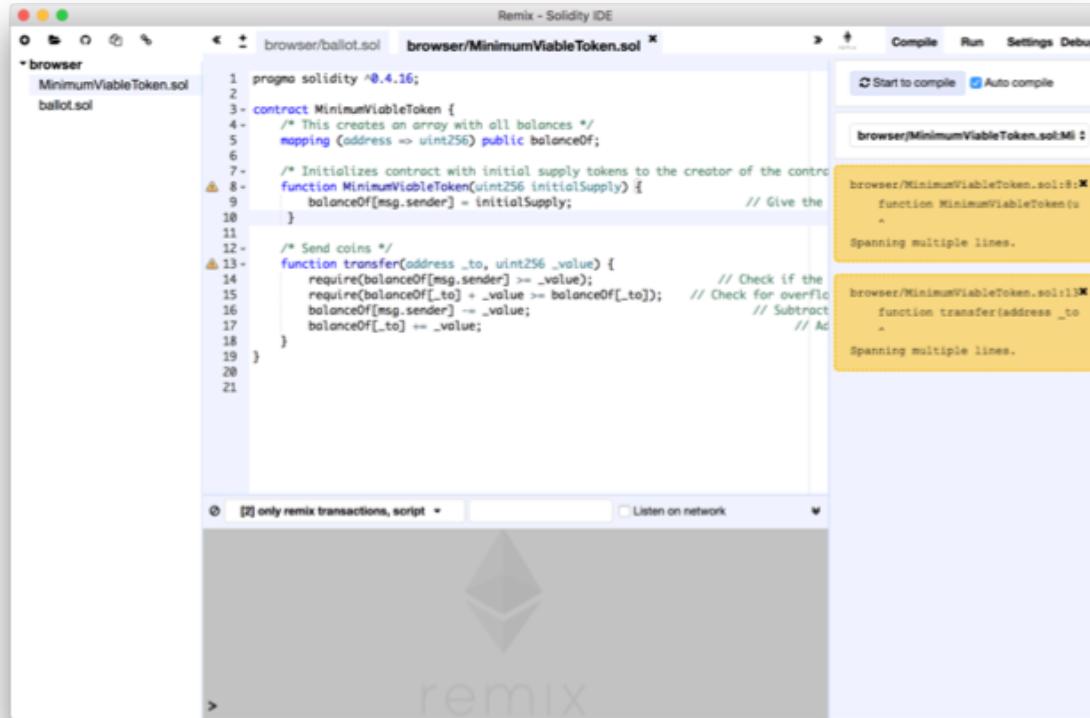
Step 1. 컨트랙트 파일 생성



다음 과정을 이용해서 컨트랙트 파일을 생성한다.

1. 파일탐색기 왼쪽 위에 있는 “파일 생성” 버튼을 클릭한다.
2. File Name 에 “MinimumViableToken.sol” 을 입력한다.

Step 2. 컨트랙트 입력



The screenshot shows the Remix Solidity IDE interface. On the left, there's a file tree with a 'browser' folder containing 'MinimumViableToken.sol' and 'ballot.sol'. The main editor tab is 'browser/MinimumViableToken.sol'. The code is as follows:

```
pragma solidity ^0.4.16;
contract MinimumViableToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;
    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MinimumViableToken(uint256 initialSupply) {
        balanceOf[msg.sender] = initialSupply; // Give the sender some tokens
    }
    /* Send coins */
    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value); // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflow
        balanceOf[msg.sender] -= _value; // Subtract sender's balance
        balanceOf[_to] += _value; // Add recipient's balance
    }
}
```

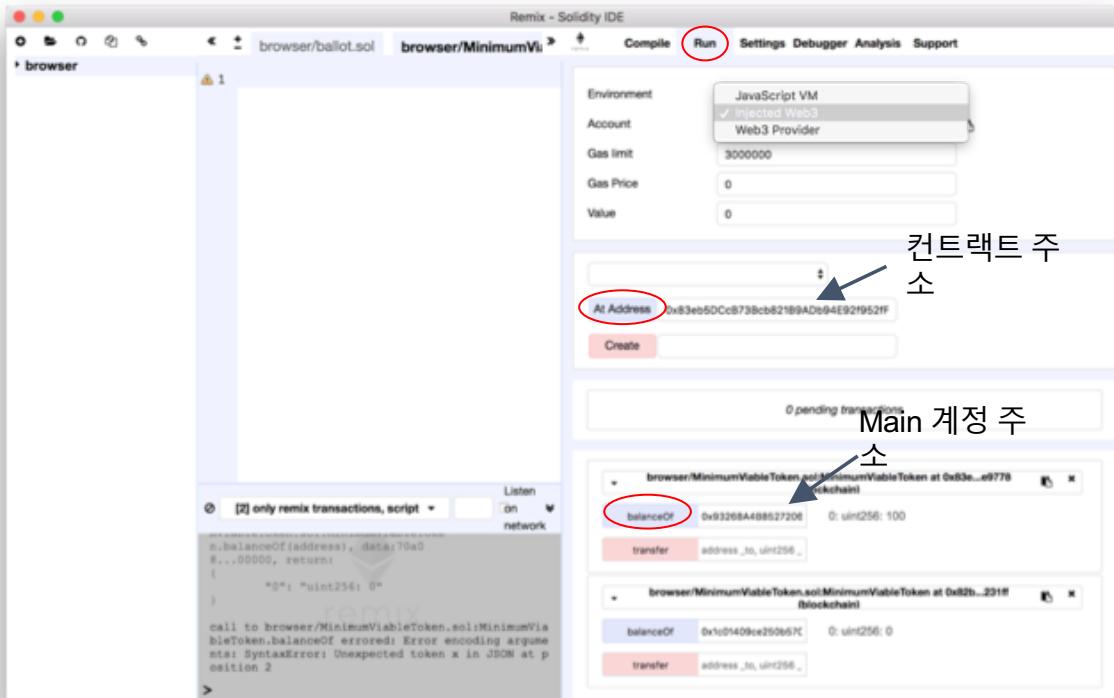
The 'Compile' tab is active, showing two error messages in yellow boxes:

- browser/MinimumViableToken.sol:8: function MinimumViableToken() Spanning multiple lines.
- browser/MinimumViableToken.sol:13: function transfer(address _to) Spanning multiple lines.

다음 과정을 이용해서 컨트랙트 파일을 생성한다.

1. 편집창에 Minimum Viable Token 소스를 복사한다.
2. Compile 탭에서 컴파일 과정에서 경고가 나오는지 확인한다.
3. 함수의 Visibility 를 명시하지 않았기 때문에 issue 가 발생한다.
4. 함수 이름 뒤에 visibility 지정자로 public 을 입력한다.

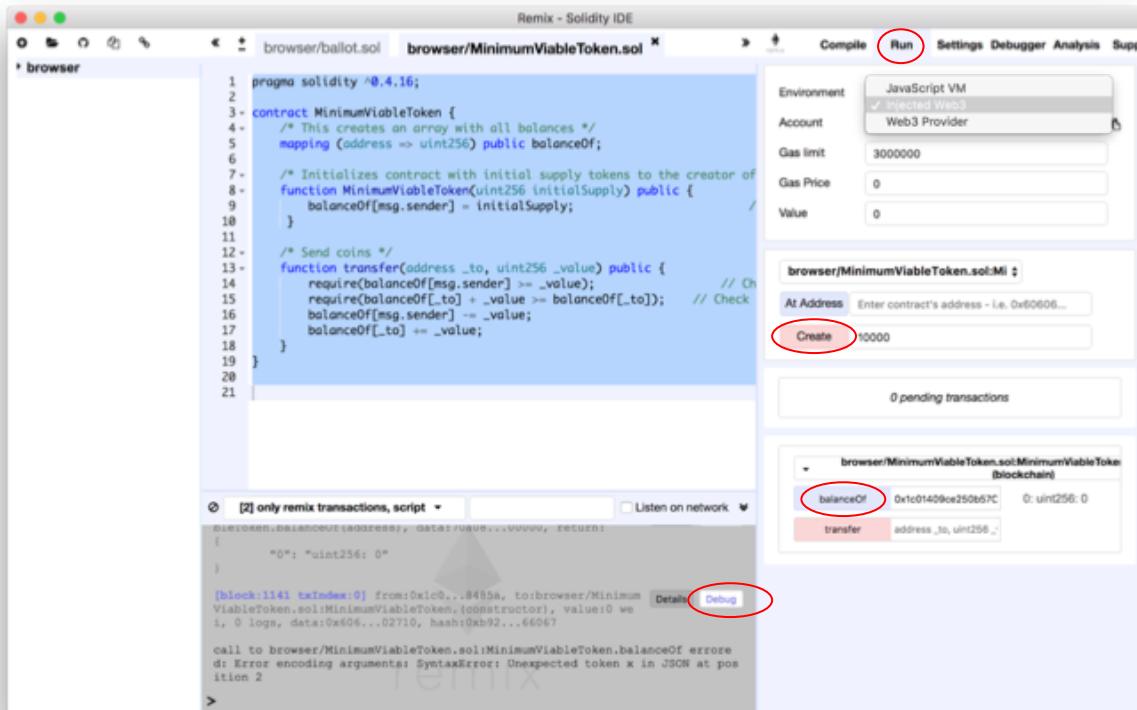
Step 3. 기존 컨트랙트 실행하기



Remix IDE에서 기존 컨트랙트를 사용 할 수 있다.

1. 템 패널에서 Run 템을 클릭한다.
2. Environment에서 Injected Web3를 선택한다.
3. Mist 메인화면에서 “First Minimum Viable Token” 컨트랙트의 주소를 복사한다.
4. At Address에 “First Minimum Viable Token” 컨트랙트의 주소를 입력한다.
5. “At Address”를 클릭한다.
6. balanceOf에 Main 계정 주소를 입력한다.
7. balanceOf를 클릭해서 계정의 잔액을 확인한다.

Step 4. 디버그하기



Run 탭에서 다음과 같이 컨트랙트를 신규로 생성한다.

1. Run 탭을 클릭한다.
2. Create 창 옆에 initialSupply 를 10000 으로 입력한다.
3. Create 버튼을 클릭한다.
4. balanceOf 창에 Main 계정을 입력한다.
5. balanceOf 를 클릭한다.
6. 트랜잭션 터미널에서 “Debug” 를 클릭한다.

Lab 11. DAO Hack

Address

변수 및 함수	타입	설명
address.balance	uint256	주소의 잔액 (Wei 단위로)
address.transfer(uint256 amount)		주어진 주소로 amount 만큼의 Wei 를 보낸다. 실패하면 throw. (2300 gas)
address.send(uint256 amount)	bool	주어진 주소로 amount 만큼의 Wei 를 보낸다. 실패하면 false 반환. 실패시 처리로직을 구현해야 함 (2300 gas 만 보냄)
address.call(...)	bool	임의의 컨트랙에 있는 함수를 불러야 할 때 사용한다. EVM 의 CALL 을 부른다. 실패하면 false 를 반환한다. <ul style="list-style-type: none">- ether 값 지정 가능 : address.call.value(1 ether)()- gas 값 지정 가능 : address.call.gas(100000)()
address.callcode(...)	bool	EVM 의 CALLCODE 를 부른다. 실패하면 false 를 반환 <ul style="list-style-type: none">- gas(), value() 옵션 사용 가능
address.delegatecall(...)	bool	EVM 의 DELEGATECALL 을 부른다. 실패하면 false 를 반환 call 과 동일하지만, 불린 함수에서 현재 컨트랙의 storage, balance 등을 사용 다른 컨트랙에 있는 코드를 라이브러리로 사용할 때 필요함 <ul style="list-style-type: none">- gas() 옵션 사용 가능, value() 옵션은 사용할 수 없음

이 코드의 문제는?

```
pragma solidity ^0.4.0;

contract Fund {
    mapping(address => uint) shares;

    function withdraw() public {
        if (msg.sender.call.value(shares[msg.sender])())
            shares[msg.sender] = 0;
    }
}
```

이 코드는?

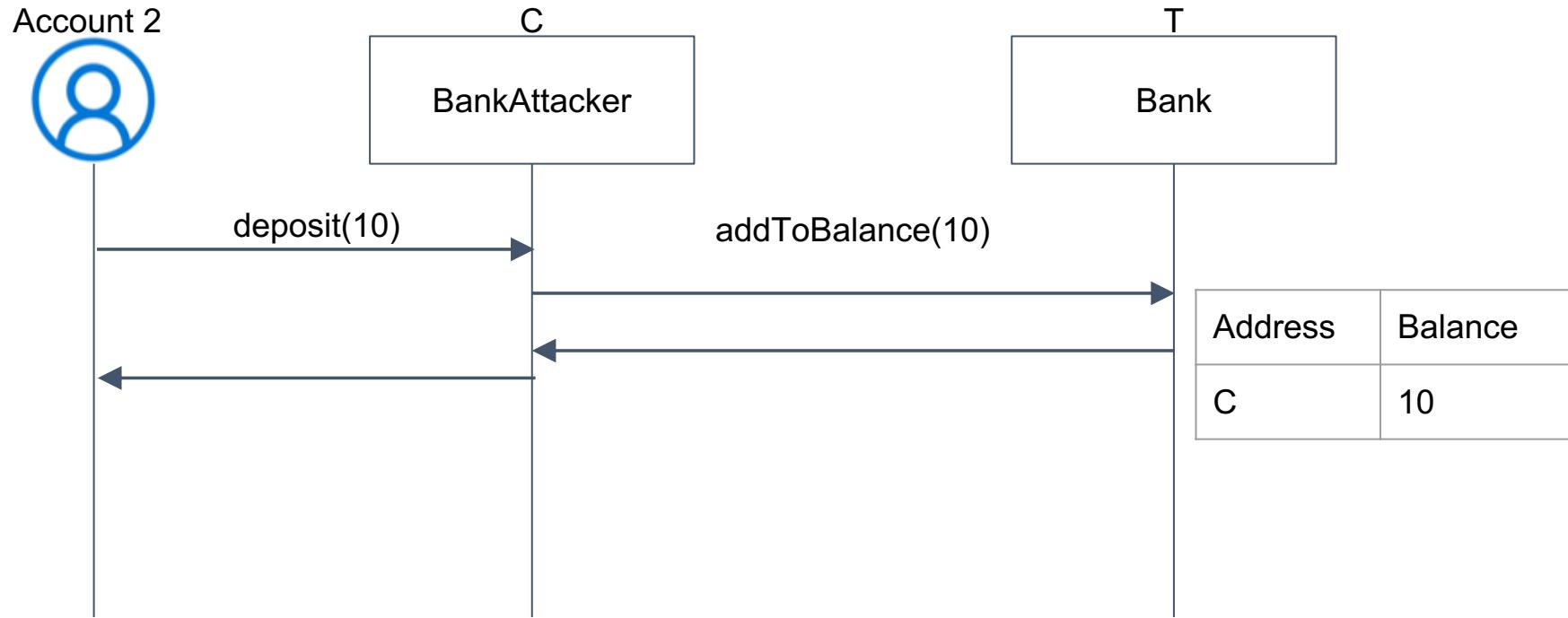
```
pragma solidity ^0.4.0;

contract Fund {

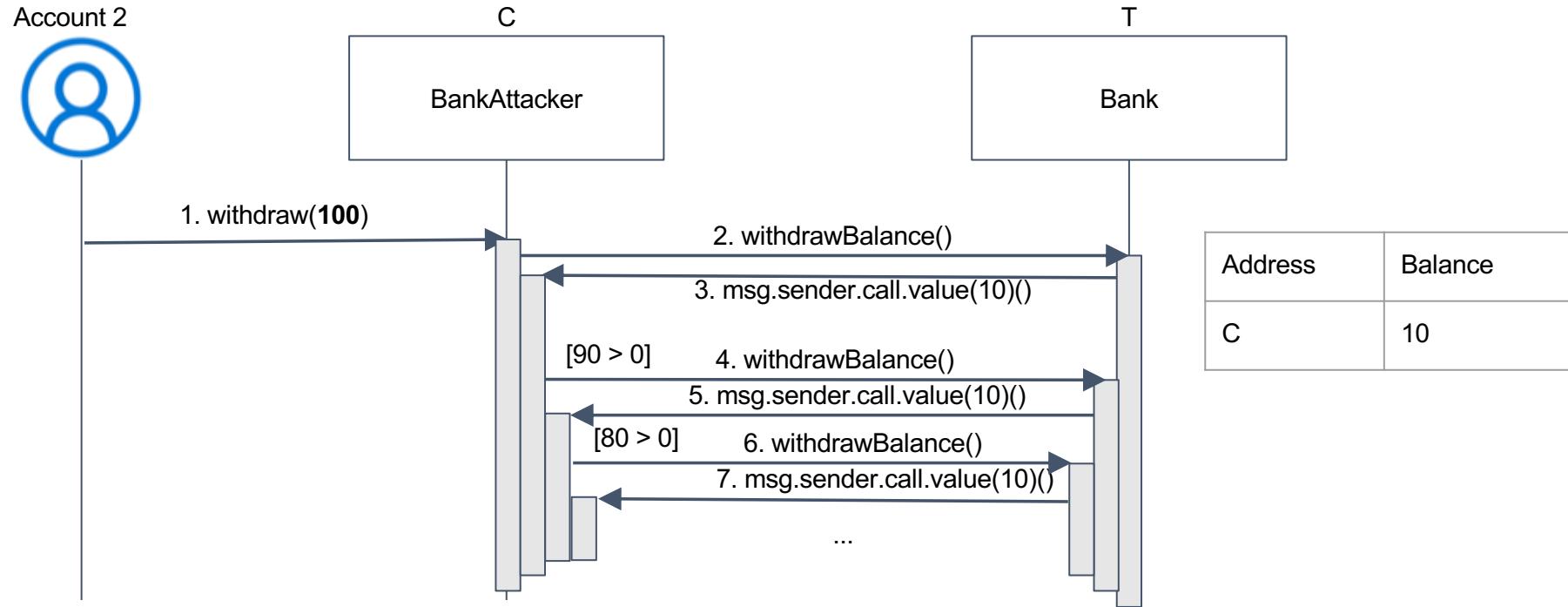
    mapping(address => uint) shares;

    function withdraw() public {
        if (msg.sender.send(shares[msg.sender]))
            shares[msg.sender] = 0;
    }
}
```

DAO Hack



DAO Hack



Check-Effects-Interaction Pattern

```
pragma solidity ^0.4.11;

contract Fund {
    mapping(address => uint) shares;

    function withdraw() public {
        var share = shares[msg.sender];
        shares[msg.sender] = 0;
        msg.sender.transfer(share);
    }
}
```