

한 번에 끝내는 블록체인 개발 A to Z

Chapter 3

Lottery 컨트랙트 v1 개발

Chapter 3

Lottery 컨트랙트 v1 개발

Lottery 컨트랙트 구현하기 (1)

Truffle 컨트랙트 개발환경 구축

- lottery-truffle이라는 디렉토리를 생성
- lottery-truffle 디렉토리로 이동
- Truffle 개발환경으로 초기화: truffle init

```
lottery-truffle -- zsh -- 117x50
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 ch3 % mkdir lottery-truffle
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 ch3 % cd lottery-truffle
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 lottery-truffle % truffle init

Starting init...
=====

> Copying project files to /Users/chunjunghyun/junghyun/fastcampus/gambling_dapp_개발_실습/ch3/lottery-truffle

Init successful, sweet!

Try our scaffold commands to get started:
  $ truffle create contract YourContractName # scaffold a contract
  $ truffle create test YourTestName        # scaffold a test

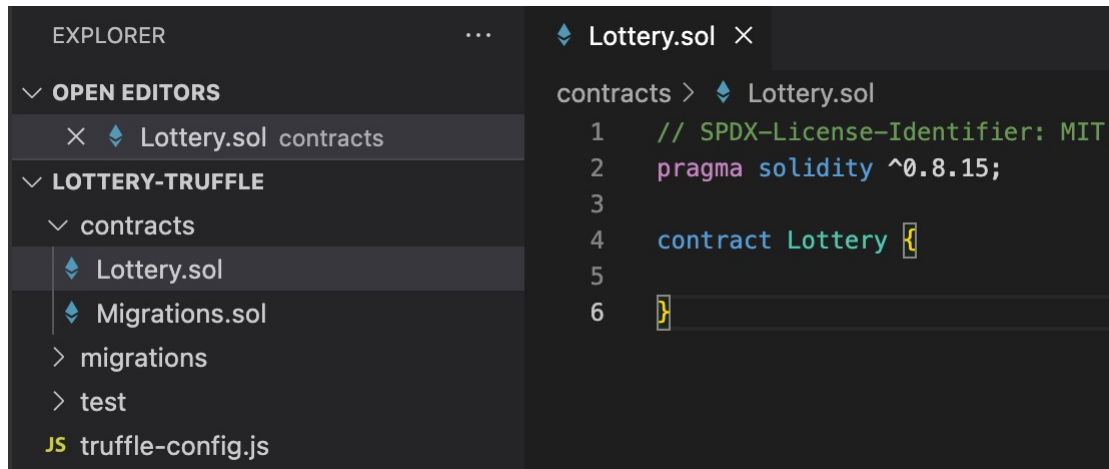
http://trufflesuite.com/docs

chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 lottery-truffle % ls
contracts      migrations      test            truffle-config.js
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 lottery-truffle %
```

Lottery

컨트랙트 구현하기 - 틀 생성

- SPDX-License-Identifier: MIT: 모든 컨트랙트 소스코드 상단에 SPDX License 정보를 주석으로 추가해줘야함
 - MIT License: 누구나 무상으로 소스코드 사용 가능
- pragma solidity ^0.8.15: 솔리디티 컴파일러에게 컨트랙트에서 사용할 솔리디티 버전을 알려줌
 - ^0.8.15 의미: 0.8.15 <= 허용 < 0.9.0



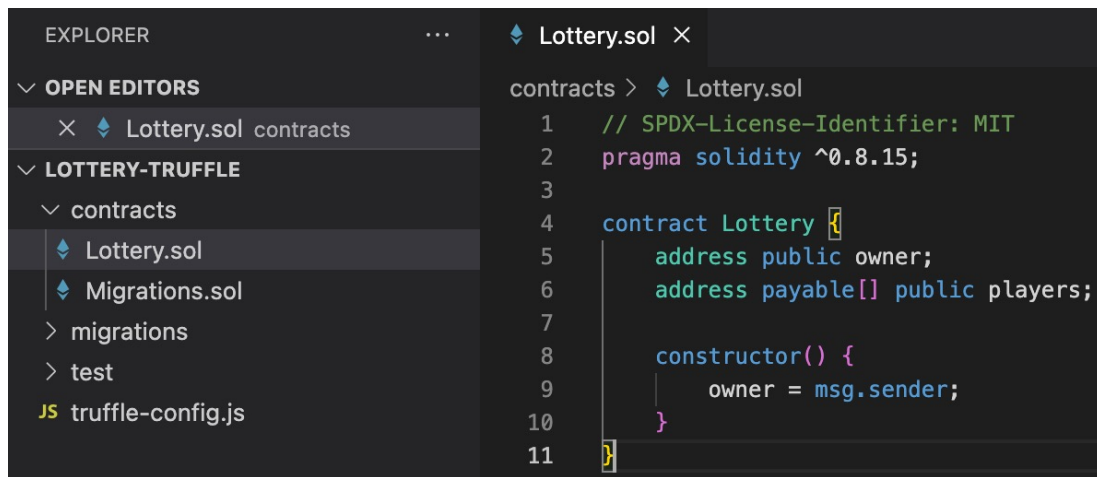
```
EXPLORER
├── OPEN EDITORS
│   └── Lottery.sol contracts
├── LOTTERY-TRUFFLE
│   └── contracts
│       ├── Lottery.sol
│       ├── Migrations.sol
│       ├── migrations
│       ├── test
│       └── truffle-config.js
└── ...

contracts > Lottery.sol
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.15;
3
4  contract Lottery {
5
6  }
```

Lottery

컨트랙트 구현하기 - 필요 변수들

- owner: address 타입의 owner 변수에 컨트랙트 배포시 사용한 address(= msg.sender)를 저장
- players: Lottery에 참여하는 address 리스트
 - address payable: ETH를 수신할 수 있는 address 타입
 - 나중에 winner 선정시 players 리스트 중에 선정 → winner는 ETH를 받으므로 address payable 타입이어야함



The screenshot shows the VS Code interface. On the left, the Explorer panel displays the project structure: 'OPEN EDITORS' with 'Lottery.sol' open, and 'LOTTERY-TRUFFLE' containing 'contracts' (with 'Lottery.sol' and 'Migrations.sol'), 'migrations', 'test', and 'truffle-config.js'. On the right, the Editor panel shows the code for 'Lottery.sol'.

```
contracts > Lottery.sol
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.15;
3
4  contract Lottery {
5      address public owner;
6      address payable[] public players;
7
8      constructor() {
9          owner = msg.sender;
10     }
11 }
```

Lottery 컨트랙트 구현하기 - enter()

- public payable function
 - enter() 함수는 사용자로부터 ETH를 전송받을 목적의 함수이므로 payable 타입이어야함
- require(msg.value >= .01 ether, “”)
 - msg.value: enter() 함수를 호출한 address가 전송한 ETH value
 - 의미: 사용자가 전송한 ETH value는 0.01 ETH 이상이어야함. 미만일 경우 revert

```
contracts > Lottery.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.15;
3
4 contract Lottery {
5     address public owner;
6     address payable[] public players;
7
8     constructor() {
9         owner = msg.sender;
10    }
11
12    function enter() public payable {
13        require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
14        players.push(payable(msg.sender));
15    }
16 }
```

- players.push(payable(msg.sender))
 - msg.sender는 그냥 address 타입.
address payable 타입의 배열인
players에 저장하려면 payable
타입으로 converting 필요

Lottery

컨트랙트 구현하기 - getBalance()

- `address(this).balance`
 - `address(this)`: 이 컨트랙트의 주소
 - 즉, 이 컨트랙트가 가지고 있는 총 ETH balance

```
Lottery.sol ×
contracts > Lottery.sol
1  // SPDX-License-Identifier: MIT
2  pragma solidity ^0.8.15;
3
4  contract Lottery {
5      address public owner;
6      address payable[] public players;
7
8      constructor() {
9          owner = msg.sender;
10     }
11
12     function getBalance() public view returns (uint256) {
13         return address(this).balance;
14     }
15
16     function enter() public payable {
17         require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
18         players.push(payable(msg.sender));
19     }
20 }
```

Lottery 컨트랙트 구현하기 - getPlayers()

- Players는 address payable 타입의 배열이므로, return시 address payable[] memory여야함
- memory: players 값은 storage에 저장되어 있는 값이므로 이 내용을 읽어서 return하고자 할 경우엔 memory 타입이어야함

```
contracts > Lottery.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.15;
3
4 contract Lottery {
5     address public owner;
6     address payable[] public players;
7
8     constructor() {
9         owner = msg.sender;
10    }
11
12    function getBalance() public view returns (uint256) {
13        return address(this).balance;
14    }
15
16    function getPlayers() public view returns (address payable[] memory) {
17        return players;
18    }
19
20    function enter() public payable {
21        require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
22        players.push(payable(msg.sender));
23    }
24 }
```


Lottery 컨트랙트 구현하기 - getRandomNumber()

- `abi.encodePacked(owner, block.timestamp)`: owner와 block.timestamp 각각을 bytes로 converting한 값을 concat한 값
- 이를 keccak256 해시 알고리즘으로 해시한 값
- 이를 uint256으로 converting한 값
→ 랜덤값

```
Lottery.sol x
contracts > Lottery.sol
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.15;
3
4 contract Lottery {
5     address public owner;
6     address payable[] public players;
7
8     constructor() {
9         owner = msg.sender;
10    }
11
12    function getBalance() public view returns (uint256) {
13        return address(this).balance;
14    }
15
16    function getPlayers() public view returns (address payable[] memory) {
17        return players;
18    }
19
20    function enter() public payable {
21        require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
22        players.push(payable(msg.sender));
23    }
24
25    function getRandomNumber() public view returns (uint256) {
26        return uint256(keccak256(abi.encodePacked(owner, block.timestamp)));
27    }
28 }
```

Lottery 컨트랙트 구현하기

- pickWinner()

- onlyOwner modifier: pickWinner() 함수는 누구나 호출할 수 있으면 안됨. 컨트랙트 오너만 호출 가능하도록 제한하는 역할
- index = getRandomNumber() % players.length: 랜덤값을 참여한 players 수로 나눈 나머지 → 즉, 참여한 players 중에 랜덤하게 뽑겠다는 의미
- players[index].call{value: address(this).balance}(""): 랜덤하게 뽑힌 player에게 컨트랙트의 모든 ETH를 전송
- players = new address payable[](0): 다음 회차를 위해 players 배열 초기화

```
7      uint256 public lotteryId;  
8      mapping(uint256 => address) public lotteryHistory;
```

```
31     function pickWinner() public onlyOwner {  
32         uint256 index = getRandomNumber() % players.length;  
33  
34         lotteryHistory[lotteryId] = players[index];  
35         lotteryId++;  
36  
37         (bool success, ) = players[index].call{value: address(this).balance}("");  
38         require(success, "Failed to send Ether");  
39  
40         players = new address payable[](0);  
41     }  
42  
43     modifier onlyOwner {  
44         require(msg.sender == owner);  
45         _;  
46     }
```

Lottery 컨트랙트 구현하기 - getRandomNumberV2, V3()

- block.timestamp 외에 또다른 블록 상태값 이용하는 것도 가능
- 이렇게 컨트랙트 변수 및 블록 상태값을 이용한 PRNG는 같은 트랜잭션 내에서 값이 동일함 → 블록체인은 deterministic 하기 때문
- 즉, 블록체인 특성상 같은 트랜잭션 내에서 랜덤값이 다를 수 없음

```
31     function getRandomNumberV2() public view returns (uint256) {  
32         return uint256(keccak256(abi.encodePacked(block.difficulty, block.timestamp, players)));  
33     }  
34  
35     function pickWinner() public onlyOwner {  
36         uint256 index = getRandomNumberV2() % players.length;
```

```
35     function getRandomNumberV3() public view returns (uint256) {  
36         return uint256(keccak256(abi.encodePacked(blockhash(block.number - 1), block.timestamp)));  
37     }  
38  
39     function pickWinner() public onlyOwner {  
40         uint256 index = getRandomNumberV3() % players.length;
```