한 번에 끝내는
블록체인 개발 A to Z

Chapter 3

Lottery 컨트랙트 v1 개발

Chapter 3

Lottery 컨트랙트 v1 개발

Lottery 컨트랙트 테스트하기 - truffle test

테스트코드의 중요성

- 블록체인의 특성상 컨트랙트는 한 번 배포하고 나면 수정이 불가능함
- 코드가 공개되어 있고 누구나 접근이 가능하여 컨트랙트의 안정성이 무엇보다 중요
- 자세하고 다양한 케이스에 대해 충분히 테스트한 후 배포하는게 좋음
- 테스트 케이스 이력을 남김으로써 사용법 문서처럼 사용 가능

Truffle test

- Truffle은 mocha 프레임워크를 기반으로 컨트랙트 테스트 환경에 맞게 변형함
- 매 테스트마다 초기화된 환경의 컨트랙트로 테스트 진행
- 디폴트로 web3 라이브러리 제공
- 디폴트로 chai 테스트 라이브러리의 assert, expect 함수 제공

Ganache CLI 실행

- 명령어: ganache
- Truffle 테스트코드 실행히 Ganache GUI가 아닌 Ganache CLI 사용하기
    - 왜? Ganache GUI는 무거워서 테스트코드의 실행 속도를 못따라감

```
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 lottery-truffle % ganache
ganache v7.0.0 (@ganache/cli: 0.1.1, @ganache/core: 0.1.1)
Starting RPC server

Available Accounts
==================
(0) 0xFA40b5916bc001E1b7c55F14e0965636C13FBf7D (1000 ETH)
(1) 0x3C3a9ab8A27be418B6EalAd66f74a497eE3fE25e (1000 ETH)
(2) 0xDcAdeB15CDeD60F17df50eCAF8d6c36c4942f169 (1000 ETH)
(3) 0x03B070e4037c7A77590dcFeE2F058c498005E952 (1000 ETH)
(4) 0xF278fEF57a9D619b53895D4bd19E479d53897c40 (1000 ETH)
(5) 0x7928AF09701712BDC3850CC01a50B435CB2Ee9E7 (1000 ETH)
(6) 0x78352d11E6a9F970F9F3f2D8C25e12BB45F5B43e (1000 ETH)
(7) 0x8347C454eAD90476C7687E85BBe6b35226CFf6c7 (1000 ETH)
(8) 0xbA9BA7E1D66B2dD7D4542D2FB7db6d858302e645 (1000 ETH)
(9) 0x79e6f080aB0f0DC0176e533c1cFba81B5e4B2875 (1000 ETH)

Private Keys
==================
(0) 0x273e1bc03f3b73d5cf46751ff0c960f6530153078608cc99a2727145b02483e4
(1) 0xba696143dda5d857a6b62bcd9e936cc323e69ac5a695ffe83d2bc9b1fbedefdc
(2) 0x05bab2b63a8739d054cbeb4aabc39dcbc9d6447fa87798bad26f99ba900583c0
(3) 0xd5545214153b944c4a1b2a1df75de61dfaadc968dbb9e0c9fe9fa10d780e1810
(4) 0x64985fc2f0e09986b0333bfd6ffd290f14be944c202e740ab92ada535db0f23c
(5) 0x16dae2c3473878307ebf71822752d17aae668aa80cdcf7c6ed9ed8228ac91477
(6) 0x249d60dd1c779e223eea7e2df7c290b9eb58005b024735421399e7558a218f82
(7) 0x6ab6ef113bd9441ec97fbf8ef3627c846b132f085d049e4e4d8428c0c3579d67
(8) 0xe4dbe0dc4a472a27308d68c4ccb67efcd853e974221b19f41b260ed9d9edfd8f
(9) 0xe23dcc4702c348765ab618d7ba49ff765e1fdd853d46891b87db9c88db195e88

HD Wallet
==================
Mnemonic:      feed pattern paper sing express vocal infant pig unfair deal dash wide
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
==================
2000000000

BlockGas Limit
==================
30000000

Call Gas Limit
==================
50000000

Chain Id
==================
1337

RPC Listening on 127.0.0.1:8545
```

Truffle test

- 명령어: truffle test
  - test 디렉토리에 있는 모든 테스트파일 실행
  - 특정 테스트 파일만 실행하고 싶다면 명령어 뒤에 path 주기
    - 예) truffle test test/lottery.test.js

```
chunjunghyun@cheonjeonghyeon-ui-MacBook-Pro-16 lottery-truffle % truffle test test/lottery.test.js
Using network 'development'.


Compiling your contracts...
===========================
> Compiling ./contracts/CommitRevealLottery.sol
> Compiling ./contracts/Lottery.sol
> Artifacts written to /var/folders/1_/4z2c2cjn2p70mvdlv4j8606h0000gn/T/test--35027-tG9xEvTMDMRV
> Compiled successfully using:
   - solc: 0.8.15+commit.e14f2714.Emscripten.clang
[
  '0xFA40b5916bc001E1b7c55F14e0965636C13FBf7D',
  '0x3C3a9ab8A27be418B6Ea1Ad66f74a497eE3fE25e',
  '0xDcAdeB15CDeD60F17df50eCAF8d6c36c4942f169',
  '0x03B070e4037c7A77590dcFeE2F058c498005E952',
  '0xF278fEF57a9D619b53895D4bd19E479d53897c40',
  '0x7928AF09701712BDC3850CC01a50B435CB2Ee9E7',
  '0x78352d11E6a9F970F9F3f2D8C25e12BB45F5B43e',
  '0x8347C454eAD90476C7687E85BBe6b35226CFf6c7',
  '0xbA9BA7E1D66B2dD7D4542D2FB7db6d858302e645',
  '0x79e6f080aB0f0DC0176e533c1cFba81B5e4B2875'
]


  Contract: Lottery
lottery address: 0x8D51E2B4183fF18F885876144867891bE419d52B
    Constructor
      ✔ Owner should be set to accounts[0]
    Enter
enterAmt: 9000000000000000
      ✔ Should revert if a player enters less than 0.01 ether (220ms)
enterAmt: 10000000000000000
[]
      ✔ Enter 5 players and check values (292ms)
    PickWinner
      ✔ Should revert if pickWinner is called by not owner
```

컨트랙트 테스트 틀 만들기

- contract()
  - mocha 프레임워크의 describe() 기능 + contract() 내부의 테스트 함수들 실행 전, 컨트랙트들을 새롭게 배포하여 초기화된 환경에서 테스트가 진행되도록 함
  - 연결된 블록체인 노드에 로드되어있는 account 리스트 제공

```
const Lottery = artifacts.require("Lottery");

contract("Lottery", accounts => {
    console.log(accounts);
```

컨트랙트 연결하기

- before()
  - contract() 실행 후 & 내부의 테스트 함수들이 실행되기 전에 실행되는 부분
  - 여기서 deployed된 컨트랙트를 읽어와서 연결해주기 좋음

```
const Lottery = artifacts.require("Lottery");

contract("Lottery", accounts => {
    console.log(accounts);

    let lottery;

    before(async () => {
        lottery = await Lottery.deployed();
        console.log(`lottery address: ${lottery.address}`);
    });
```

assertion 타입

- describe()
  - 하나의 큰 테스트 목차로 묶고싶을 때 사용
- it()
  - 하나의 테스트 케이스
  - describe() 안에 여러 개의 it() 가능
- assertion 종류
  - assert: 보통의 함수 형식으로 사용
    - assert.함수(actual, expected)
  - expect: 줄글 형식으로 사용
    - expect(actual).줄글(expected)
  - should: 줄글 형식으로 사용
    - actual.should.줄글(expected)

```
describe("Constructor", () => {
    it("Owner should be set to accounts[0]", async () => {
        const owner = await lottery.owner();
        assert.equal(owner, accounts[0]);
        expect(owner).to.equal(accounts[0]);
        owner.should.equal(accounts[0]);
    });
});
```

배포 후 생성자 값 체크

- 이 테스트의 목적은 배포 후 생성자 값이 잘 세팅되었는지 체크하는 것
- 즉, 이 테스트의 제목, 목차는 Constructor라고 지을 수 있음 → describe(“Constructor”)
- 이 하위 항목에서 테스트하고자 하는 바들을 it()으로 정의
- Lottery owner가 accounts[0]으로 잘 세팅되어있는지 체크
- assert.equal(actual, expected) = expect(actual).to.equal(expected) = actual.should.equal(expected)

```
describe("Constructor", () => {
    it("Owner should be set to accounts[0]", async () => {
        const owner = await lottery.owner();
        assert.equal(owner, accounts[0]);
        expect(owner).to.equal(accounts[0]);
        owner.should.equal(accounts[0]);
    });
});
```

enter() 테스트

- 이 테스트의 목적은 enter() 기능을 테스트하는 것
- 즉, 이 테스트의 제목, 목차는 Enter라고 지을 수 있음 → describe(“Enter”)
- 먼저 require() 구문이 잘 동작하는지 체크
- revert가 잘 되는지 확인하기 위해 truffle-assertion npm 모듈 사용

```
function enter() public payable {
    require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
    players.push(payable(msg.sender));
}
```

```
describe("Enter", () => {
    it("Should revert if a player enters less than 0.01 ether", async () => {
        const enterAmt = web3.utils.toWei("0.009", "ether");
        console.log(`enterAmt: ${enterAmt}`);

        await truffleAssert.reverts(lottery.enter({ from: accounts[1], value: enterAmt }));
    });
```

enter() 테스트

- 5명의 player가 enter 하는 상황 테스트
- 한 명씩 enter 할 때마다 컨트랙트의 ETH balance가 의도한대로 느는지, players 배열에 의도한대로 account가 저장되는지 체크

```
it("Enter 5 players and check values", async () => {
    const enterAmt = web3.utils.toWei("0.01", "ether");
    console.log(`enterAmt: ${enterAmt}`);

    // player1 enter
    await lottery.enter({ from: accounts[1], value: enterAmt });

    // check values
    // assert
    assert.equal(await lottery.getBalance(), enterAmt);
    assert.deepEqual(await lottery.getPlayers(), [accounts[1]]);
    // expect
    expect((await lottery.getBalance()).toString()).to.equal(enterAmt);
    expect(await lottery.getPlayers()).to.deep.equals([accounts[1]]);
    // should
    ((await lottery.getBalance()).toString()).should.equal(enterAmt);
    (await lottery.getPlayers()).should.deep.equals([accounts[1]]);

    // player2 enter
    let result = await lottery.enter({ from: accounts[2], value: enterAmt });
    console.log(result.logs);
    assert.equal(await lottery.getBalance(), web3.utils.toBN(enterAmt).mul(web3.utils.toBN(2)).toString()); // toString() 붙여야함
    assert.deepEqual(await lottery.getPlayers(), [accounts[1], accounts[2]]);
```

pickWinner() 테스트

- pickWinner()는 컨트랙트 owner만 호출 가능
- owner가 아닌 계정이 콜했을 때 정상적으로 revert 되는지 확인

```
describe("PickWinner", () => {
    it("Should revert if pickWinner is called by not owner", async () => {
        // owner: accounts[0]
        await truffleAssert.reverts(lottery.pickWinner({ from: accounts[1] }));
    });
```

pickWinner() 테스트

- pickWinner 전, 후 상황 확인 및 테스트
- pickWinner() 호출 전, 5명의 player의 ETH 잔액 체크
    - pickWinner 후, winner의 ETH balance가 0.05 ETH 늘었는지 체크하기 위함
- pickWinner() 호출

```
it("PickWinner", async () => {
    console.log(">>> before pickWinner");

    // check players' ETH balances before pickWinner
    const account1ETHBal_bef = await web3.eth.getBalance(accounts[1]);
    console.log(`account1's ETH balance: ${account1ETHBal_bef}`);
    const account2ETHBal_bef = await web3.eth.getBalance(accounts[2]);
    console.log(`account2's ETH balance: ${account2ETHBal_bef}`);
    const account3ETHBal_bef = await web3.eth.getBalance(accounts[3]);
    console.log(`account3's ETH balance: ${account3ETHBal_bef}`);
    const account4ETHBal_bef = await web3.eth.getBalance(accounts[4]);
    console.log(`account4's ETH balance: ${account4ETHBal_bef}`);
    const account5ETHBal_bef = await web3.eth.getBalance(accounts[5]);
    console.log(`account5's ETH balance: ${account5ETHBal_bef}`);

    // pickWinner
    console.log(">>> pickWinner");
    await lottery.pickWinner();
```

pickWinner() 테스트

- pickWinner() 호출 후
- lotteryId가 정상적으로 1 증가했는지 체크
- lotteryHistory에 0회차 winner가 잘 저장됐는지 체크
- 5명의 player의 ETH balance 체크
- pickWinner() 호출 전후로 0.05 ETH 늘어난 계정 체크

```
console.log(">>> after pickWinner");

const lotteryId = await lottery.lotteryId();
console.log(`lotteryId: ${lotteryId}`);
assert.equal(lotteryId, 1);

const winner = await lottery.lotteryHistory(lotteryId - 1);
console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

// check players' ETH balances after pickWinner
const account1ETHBal_aft = await web3.eth.getBalance(accounts[1]);
console.log(`account1's ETH balance: ${account1ETHBal_aft}`);
const account2ETHBal_aft = await web3.eth.getBalance(accounts[2]);
console.log(`account2's ETH balance: ${account2ETHBal_aft}`);
const account3ETHBal_aft = await web3.eth.getBalance(accounts[3]);
console.log(`account3's ETH balance: ${account3ETHBal_aft}`);
const account4ETHBal_aft = await web3.eth.getBalance(accounts[4]);
console.log(`account4's ETH balance: ${account4ETHBal_aft}`);
const account5ETHBal_aft = await web3.eth.getBalance(accounts[5]);
console.log(`account5's ETH balance: ${account5ETHBal_aft}`);

// check balance difference
console.log(`account1 balance difference: ${web3.utils.toBN(account1ETHBal_aft).sub(web3.utils.toBN(account1ETHBal_bef))}`);
console.log(`account2 balance difference: ${web3.utils.toBN(account2ETHBal_aft).sub(web3.utils.toBN(account2ETHBal_bef))}`);
console.log(`account3 balance difference: ${web3.utils.toBN(account3ETHBal_aft).sub(web3.utils.toBN(account3ETHBal_bef))}`);
console.log(`account4 balance difference: ${web3.utils.toBN(account4ETHBal_aft).sub(web3.utils.toBN(account4ETHBal_bef))}`);
console.log(`account5 balance difference: ${web3.utils.toBN(account5ETHBal_aft).sub(web3.utils.toBN(account5ETHBal_bef))}`);
```

getRandomNumber() 테스트

- getRandomNumber() 사용하는 pickWinner() 호출 후
- getRandomNumber()의 랜덤값과 getRandomNumber()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크
- 계산해서 예측한 winner와 실제 winner가 같은지 체크

```
it("Calculate winner - getRandomNumber", async () => {
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const randomNum = await lottery.getRandomNumber();
    console.log(`randomNumber: ${randomNum}`);

    const blockNumber = await web3.eth.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await web3.eth.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = web3.utils.toBN(web3.utils.keccak256(web3.utils.encodePacked({value: await lottery.owner(), type: "address"}, {value:
    currentBlock.timestamp, type: "uint256"}))).toString();
    console.log(`calculated random number: ${calculatedRandomNum}`);
    assert.equal(randomNum, calculatedRandomNum);

    const calculatedWinnerIndex = web3.utils.toBN(calculatedRandomNum).mod(web3.utils.toBN("5")).toString();
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    assert.equal(winner, accounts[Number(calculatedWinnerIndex) + 1]);
});
```

getRandomNumberV2() 테스트

- getRandomNumberV2() 사용하는 pickWinner() 호출 후
- 계산해서 예측한 winner와 실제 winner가 같은지 체크
- 여기선 getRandomNumberV2()의 랜덤값과 getRandomNumberV2()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크 x
    - V2의 경우, pickWinner() 호출시 V2에서 사용하는 players 배열이 리셋됨 → pickWinner()가 호출된 후 호출하는 V2의 랜덤값은 pickWinner() 실행시 사용된 랜덤값과 다른 값

```
it("Calculate winner - getRandomNumberV2", async () => {
    // V2에서만 getRandomNumberV2() 호출후 해당 랜덤값을 web3를 이용해서 구한 랜덤값과 비교 x -> 이미 pickWinner가 불리고 나면 players 배열이 리셋되기 때문에 사후에 구할 수 없음
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const blockNumber = await web3.eth.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await web3.eth.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = web3.utils.toBN(web3.utils.keccak256(web3.utils.encodePacked({value: currentBlock.difficulty, type: "uint256"}, {value:
    currentBlock.timestamp, type: "uint256"}, {value: [accounts[1], accounts[2], accounts[3], accounts[4], accounts[5]], type: "address[]"}))).toString();
    console.log(`calculated random number: ${calculatedRandomNum}`);

    const calculatedWinnerIndex = web3.utils.toBN(calculatedRandomNum).mod(web3.utils.toBN("5")).toString();
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    assert.equal(winner, accounts[Number(calculatedWinnerIndex) + 1]);
});
```

getRandomNumberV3() 테스트

- getRandomNumberV3() 사용하는 pickWinner() 호출 후
- getRandomNumberV3()의 랜덤값과 getRandomNumberV3()의 PRNG를 구현하여 계산한 랜덤값이 같은지 체크
- 계산해서 예측한 winner와 실제 winner가 같은지 체크

```
it("Calculate winner - getRandomNumberV3", async () => {
    const lotteryId = await lottery.lotteryId();
    console.log(`lotteryId: ${lotteryId}`);

    const winner = await lottery.lotteryHistory(lotteryId - 1);
    console.log(`winner at lotteryId ${lotteryId - 1}: ${winner}`);

    const randomNum = await lottery.getRandomNumberV3();
    console.log(`randomNumber: ${randomNum}`);

    const blockNumber = await web3.eth.getBlockNumber();
    console.log(`block number: ${blockNumber}`);

    const currentBlock = await web3.eth.getBlock(blockNumber);
    console.log(`current block:`, currentBlock);

    const calculatedRandomNum = web3.utils.toBN(web3.utils.keccak256(web3.utils.encodePacked({value: currentBlock.parentHash, type: "bytes32"}, {value:
    currentBlock.timestamp, type: "uint256"}))).toString();
    console.log(`calculated random number: ${calculatedRandomNum}`);
    assert.equal(randomNum, calculatedRandomNum, "calculated random number is not matching");

    const calculatedWinnerIndex = web3.utils.toBN(calculatedRandomNum).mod(web3.utils.toBN("5")).toString();
    console.log(`calculated winner index: ${calculatedWinnerIndex}`);
    assert.equal(winner, accounts[Number(calculatedWinnerIndex) + 1]);
});
```