한 번에 끝내는
블록체인 개발 A to Z

Chapter 3

Lottery 컨트랙트 v1 개발

Chapter 3

Lottery 컨트랙트 v1 개발

CommitRevealLottery 컨트랙트 테스트하기 - hardhat test

컨트랙트 연결하기

- 테스트 케이스 실행 전, before()에서 CommitRevealLottery 컨트랙트 배포 후 전역변수에 저장
- Hardhat에서 기본적으로 제공해주는 signers 리스트를 전역 변수에 저장
  - signers = await ethers.getSigners()
- 여러 번 배포할 경우를 대비해, 배포 함수를 선언하고 Hardhat에서 제공하는 loadFixture() 이용해 배포

```
const { loadFixture } = require("@nomicfoundation/hardhat-network-helpers");
require("@nomicfoundation/hardhat-chai-matchers");
const { expect } = require("chai");
const { ethers, network } = require("hardhat");

describe("CommitRevealLottery", () => {
    async function deployCommitRevealLottery() {
        const Signers = await ethers.getSigners();

        const CommitRevealLotteryContract = await ethers.getContractFactory("CommitRevealLottery");
        const CommitRevealLottery = await CommitRevealLotteryContract.deploy();

        return { CommitRevealLottery, Signers };
    }

    let commitRevealLottery;
    let signers;

    before(async () => {
        const { CommitRevealLottery, Signers } = await loadFixture(deployCommitRevealLottery);
        commitRevealLottery = CommitRevealLottery;
        signers = Signers;
    });
```

배포 후 생성자 값 체크

- commitCloses가 현재 블록 넘버보다 DURATION 만큼 더 후 인지 체크
- revealCloses가 commitCloses보다 DURATION 만큼 더 후 인지 체크
- Hardhat의 이더리움 테스트용 확장된 expect() 덕분에, BigNumber 타입이 아닌 숫자 타입과도 호환되어 편리함
    - duration: BigNumber, currentBlockNumber: Number
    - expect(commitCloses).to.equal(duration.add(currentBlockNumber));

```
describe("Constructor", () => {
    it("commitCloses & revealCloses should be set correctly", async () => {
        const commitCloses = await commitRevealLottery.commitCloses();
        const revealCloses = await commitRevealLottery.revealCloses();
        const duration = await commitRevealLottery.DURATION();
        console.log(`commitCloses: ${commitCloses}, revealCloses: ${revealCloses}, duration: ${duration}`);

        const currentBlockNumber = await ethers.provider.getBlockNumber();
        console.log(`current block number: ${currentBlockNumber}`);

        expect(commitCloses).to.equal(duration.add(currentBlockNumber));
        expect(revealCloses).to.equal(commitCloses.add(duration));
    });
});
```

enter() 테스트

- 먼저 require() 구문이 잘 동작하는지 체크
- 0.01 ETH 미만의 금액 전송시 revert가 잘 되는지 확인하기 위해 Hardhat에서 제공하는 이더리움 테스트용으로 확장된 expect 기능 이용
- 참고) 내장된 hardhat node 사용시, revert시 블록넘버 증가함. 외부 로컬 hardhat node 사용시, revert시 블록넘버 증가 x → Hardhat 버그라 추후 fix 예정

```
function enter(bytes32 commitment) public payable {
    require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
    require(block.number < commitCloses, "commit duration is over");

    commitments[msg.sender] = commitment;
}
```

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

        const secret = 12345;
        const commit = ethers.utils.solidityKeccak256(["address", "uint256"], [signers[1].address, secret]);
        console.log(`commit: ${commit}`);

        // let currentBlockNumber = await ethers.provider.getBlockNumber();
        // console.log(`current block number: ${currentBlockNumber}`);

        await expect(commitRevealLottery.connect(signers[1]).enter(commit, { value: enterAmt })).to.be.revertedWith("msg.value should be greater than or equal to 0.
        01 ether");

        // currentBlockNumber = await ethers.provider.getBlockNumber();
        // console.log(`current block number: ${currentBlockNumber}`);
    });
```

enter() 테스트

- 3명의 player가 enter 하는 상황 테스트
- 한 명씩 enter 할 때마다 컨트랙트의 ETH balance가 의도한대로 느는지, players 배열에 의도한대로 account가 저장되는지 체크

```
it("Enter 3 players and check values", async () => {
    const enterAmt = ethers.utils.parseEther("0.01");
    console.log(`enterAmt: ${enterAmt}`);

    // player1 enter
    const secret1 = 12345;
    const commit1 = ethers.utils.solidityKeccak256(["address", "uint256"], [signers[1].address, secret1]);
    console.log(`commit1: ${commit1}`);

    await commitRevealLottery.connect(signers[1]).enter(commit1, { value: enterAmt });

    // check values
    expect(await commitRevealLottery.getBalance()).to.equal(enterAmt);
    expect(await commitRevealLottery.commitments(signers[1].address)).to.equal(commit1);

    // player2 enter
    const secret2 = 12346;
    const commit2 = ethers.utils.solidityKeccak256(["address", "uint256"], [signers[2].address, secret2]);
    console.log(`commit2: ${commit2}`);

    await commitRevealLottery.connect(signers[2]).enter(commit2, { value: enterAmt });
    expect(await commitRevealLottery.getBalance()).to.equal(enterAmt.mul(2));
    expect(await commitRevealLottery.commitments(signers[2].address)).to.equal(commit2);
```

enter() 테스트

- 3명의 player가 enter 하는 상황 테스트
- 4번째 사용자가 enter 시도하는 순간, “commit duration is over” require 구문에서 정상적으로 revert 되는지 체크

```
function enter(bytes32 commitment) public payable {
    require(msg.value >= .01 ether, "msg.value should be greater than or equal to 0.01 ether");
    require(block.number < commitCloses, "commit duration is over");

    commitments[msg.sender] = commitment;
}
```

```
// player4 enter should revert
const secret4 = 12348;
const commit4 = ethers.utils.solidityKeccak256(["address", "uint256"], [signers[4].address, secret4]);
console.log(`commit4: ${commit4}`);

await expect(commitRevealLottery.connect(signers[4]).enter(commit4, { value: enterAmt })).to.be.revertedWith("commit duration is over");
```

reveal() 테스트

- 각 player 별로 컨트랙트의 createCommitment()를 통한 commit 값과 직접 만든 commit값 같은지 체크
- reveal() 전후로 isAlreadyRevealed()가 false → true로 전환되는지 체크
- 다시 같은 player가 같은 secret 값으로 reveal() 시도하면, “You already revealed” require 구문에서 정상적으로 revert 되는지 체크
- 각 player가 reveal 할 때마다 players 배열에 잘 저장되는지 체크

```
function reveal(uint256 secret) public {
    require(block.number >= commitCloses, "commit duration is not closed yet");
    require(block.number < revealCloses, "reveal duration is already closed");
    require(!isAlreadyRevealed(), "You already revealed");

    bytes32 commit = createCommitment(secret);
    require(commit == commitments[msg.sender], "commit not matches");

    seed = keccak256(abi.encodePacked(seed, secret));
    players.push(msg.sender);
}
```

```
describe("Reveal", () => {
    it("Reveal 3 players", async () => {
        // player1 reveal
        const secret1 = 12345;
        const commit1 = ethers.utils.solidityKeccak256(["address", "uint256"], [signers[1].address, secret1]);

        let commit = await commitRevealLottery.connect(signers[1]).createCommitment(secret1);
        expect(commit).to.equal(commit1);

        let isAlreadyRevealed = await commitRevealLottery.connect(signers[1]).isAlreadyRevealed();
        expect(isAlreadyRevealed).to.equal(false);

        await commitRevealLottery.connect(signers[1]).reveal(secret1);

        isAlreadyRevealed = await commitRevealLottery.connect(signers[1]).isAlreadyRevealed();
        expect(isAlreadyRevealed).to.equal(true);

        await expect(commitRevealLottery.connect(signers[1]).reveal(secret1)).to.be.revertedWith("You already revealed");

        const player1 = await commitRevealLottery.players(0);
        expect(player1).to.equal(signers[1].address);
```

reveal() 테스트

- Hardhat node의 블록을 하나 마이닝하여 블록 넘버를 1 증가시켜, 현재 블록 넘버가 revealCloses 블록 넘버에 도달하게 만듦
  - await network.provider.send("hardhat_mine", ["0x1"])
- 그 후 reveal 시도시, "reveal duration is already closed" require 구문에서 정상적으로 revert 되는지 체크

```
function reveal(uint256 secret) public {
    require(block.number >= commitCloses, "commit duration is not closed yet");
    require(block.number < revealCloses, "reveal duration is already closed");
    require(!isAlreadyRevealed(), "You already revealed");

    bytes32 commit = createCommitment(secret);
    require(commit == commitments[msg.sender], "commit not matches");

    seed = keccak256(abi.encodePacked(seed, secret));
    players.push(msg.sender);
}
```

```
let currentBlockNumber = await ethers.provider.getBlockNumber();
console.log(`current block number: ${currentBlockNumber}`);

// 블록 마이닝을 통해 블록 넘버 1 증가시킴
await network.provider.send("hardhat_mine", ["0x1"]);

currentBlockNumber = await ethers.provider.getBlockNumber();
console.log(`current block number: ${currentBlockNumber}`);

await expect(commitRevealLottery.connect(signers[3]).reveal(secret3)).to.be.revertedWith("reveal duration is already closed");
```

pickWinner() 테스트

- pickWinner() 호출 후, lotteryId가 1로 잘 증가했는지, 0회차 lotteryHistory에 winner가 잘 저장됐는지 체크

```
describe("PickWinner", () => {
    it("PickWinner", async () => {
        await commitRevealLottery.connect(signers[1]).pickWinner();

        const winner = await commitRevealLottery.winner();
        const lotteryId = await commitRevealLottery.lotteryId();

        expect(lotteryId).to.equal(1);
        expect(await commitRevealLottery.lotteryHistory(lotteryId - 1)).to.equal(winner);
    });
});
```

withdrawPrize() 테스트

- withdrawPrize() 호출 전, player 3명의 ETH balance 체크
- winner가 아닌 계정으로 withdrawPrize() 호출하면, “You’re not the winner” require 구문에서 정상적으로 revert 되는지 체크
- winner 계정으로 withdrawPrize() 호출
  - web3에선 간단하게 {from: winner} 옵션으로 가능했으나, ethers에선 winner address에 매핑되는 signer로 세팅해줘야함
  - signers 배열을 루프를 돌며 signer.address == winner인 경우에 해당하는 signer를 리턴하는 함수 필요

```
async function findMatchingSigner(signers, account) {
    for (let i = 0; i < signers.length; i++) {
        if (account == signers[i].address) return signers[i];
    }
    return null;
}
```

```
describe("WithdrawPrize", () => {
    it("WithdrawPrize", async () => {
        console.log(">>> before withdrawPrize");

        // check players' ETH balances before pickWinner
        const account1ETHBal_bef = await ethers.provider.getBalance(signers[1].address);
        console.log(`account1's ETH balance: ${account1ETHBal_bef}`);
        const account2ETHBal_bef = await ethers.provider.getBalance(signers[2].address);
        console.log(`account2's ETH balance: ${account2ETHBal_bef}`);
        const account3ETHBal_bef = await ethers.provider.getBalance(signers[3].address);
        console.log(`account3's ETH balance: ${account3ETHBal_bef}`);

        console.log(">>> withdrawPrize");

        await expect(commitRevealLottery.withdrawPrize()).to.be.revertedWith("You're not the winner");

        let winner = await commitRevealLottery.winner();
        const winnerSigner = await findMatchingSigner(signers, winner);
        await commitRevealLottery.connect(winnerSigner).withdrawPrize();
```

withdrawPrize() 테스트

- withdrawPrize() 호출 후, player 3명의 ETH balance 체크
- withdrawPrize() 호출 전후로 0.03 ETH 늘어난 계정 체크
- 변수들 모두 잘 리셋됐는지 체크

```
// check players' ETH balances after pickWinner
const account1ETHBal_aft = await ethers.provider.getBalance(signers[1].address);
console.log(`account1's ETH balance: ${account1ETHBal_aft}`);
const account2ETHBal_aft = await ethers.provider.getBalance(signers[2].address);
console.log(`account2's ETH balance: ${account2ETHBal_aft}`);
const account3ETHBal_aft = await ethers.provider.getBalance(signers[3].address);
console.log(`account3's ETH balance: ${account3ETHBal_aft}`);

// check balance difference
console.log(`account1 balance difference: ${account1ETHBal_aft.sub(account1ETHBal_bef)}`);
console.log(`account2 balance difference: ${account2ETHBal_aft.sub(account2ETHBal_bef)}`);
console.log(`account3 balance difference: ${account3ETHBal_aft.sub(account3ETHBal_bef)}`);

// check if values are reset well
winner = await commitRevealLottery.winner();
expect(winner).to.equal(ethers.constants.AddressZero);

for (let i = 0; i < 3; i++) {
    let commit = await commitRevealLottery.connect(signers[i]).commitments(signers[i].address);
    expect(commit).to.equal(ethers.constants.HashZero);
}

await expect(commitRevealLottery.players(0)).to.be.reverted;

const currentBlockNumer = await ethers.provider.getBlockNumber();
const commitCloses = await commitRevealLottery.commitCloses();
const revealCloses = await commitRevealLottery.revealCloses();
const duration = await commitRevealLottery.DURATION();

expect(commitCloses).to.equal(duration.add(currentBlockNumer));
expect(revealCloses).to.equal(commitCloses.add(duration));
```